SAS® Viya® Architecture for Practitioners

Slides and Notes

SAS® Viya® Architecture for Practitioners was developed by Edoardo Riva and SAS Education Content Development. Instructional design was provided by SAS Education Digital Development.

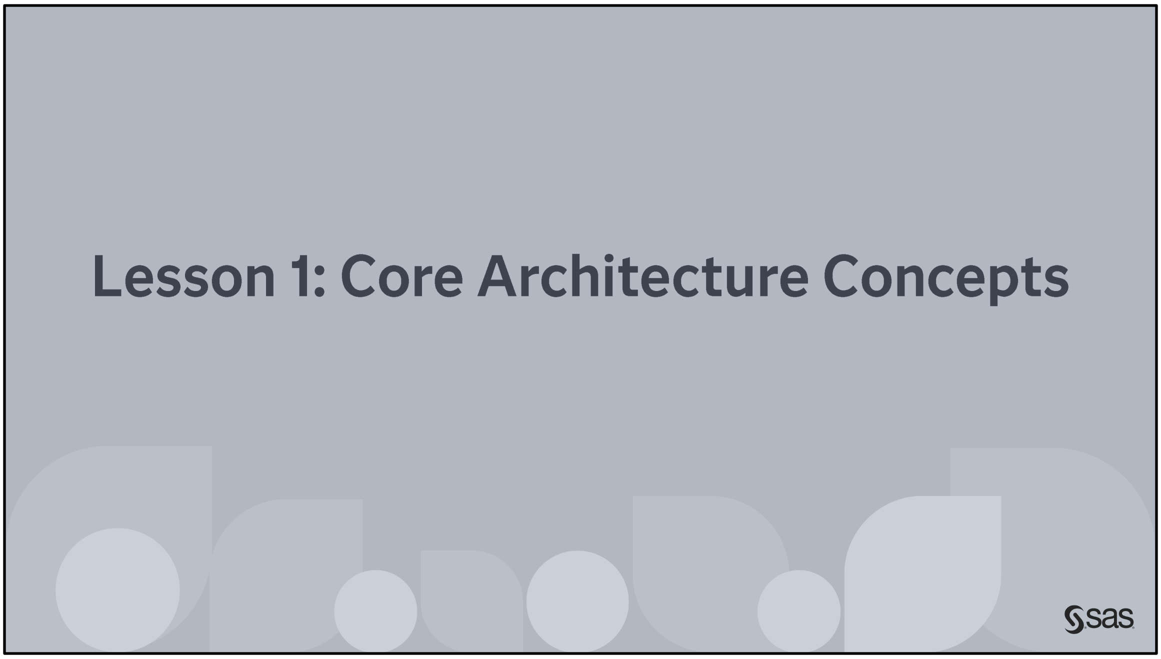
SAS® Viya® Architecture for Practitioners Slides and Notes
Content based on SAS® Viya® Release LTS 2025.03

Course code EASPCV2, prepared date June 27, 2025.
http://support.sas.com/training/

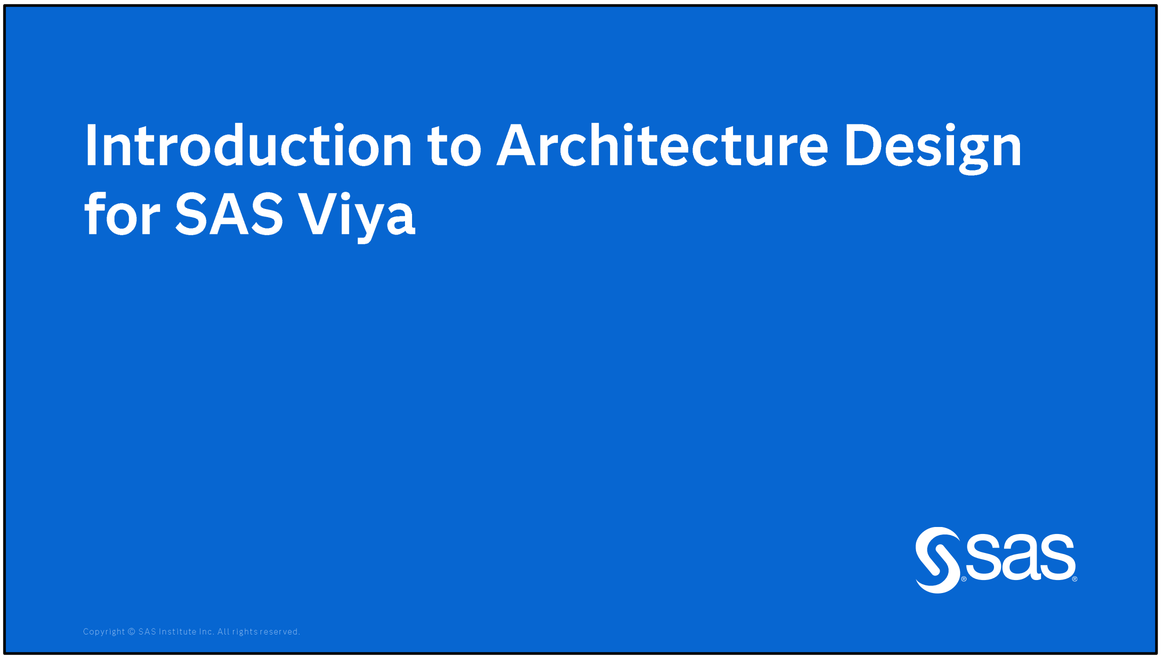
For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at http://support.sas.com/training/ as well as in the Training Course Catalog.

For a list of SAS books (including e-books) that relate to the topics covered in this course notes, visit https://www.sas.com/sas/books.html or call 1-800-727-0025. US customers receive free shipping to US addresses.

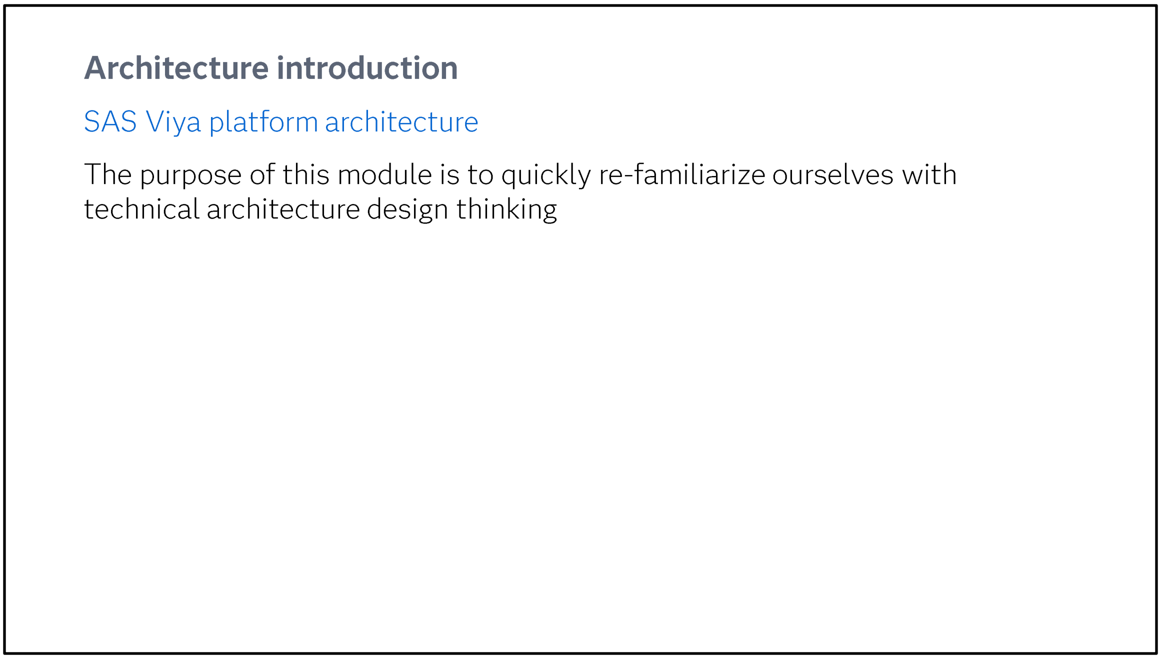
Course Overview



Lesson 1: Core Architecture Concepts



Introduction to Architecture Design for SAS Viya



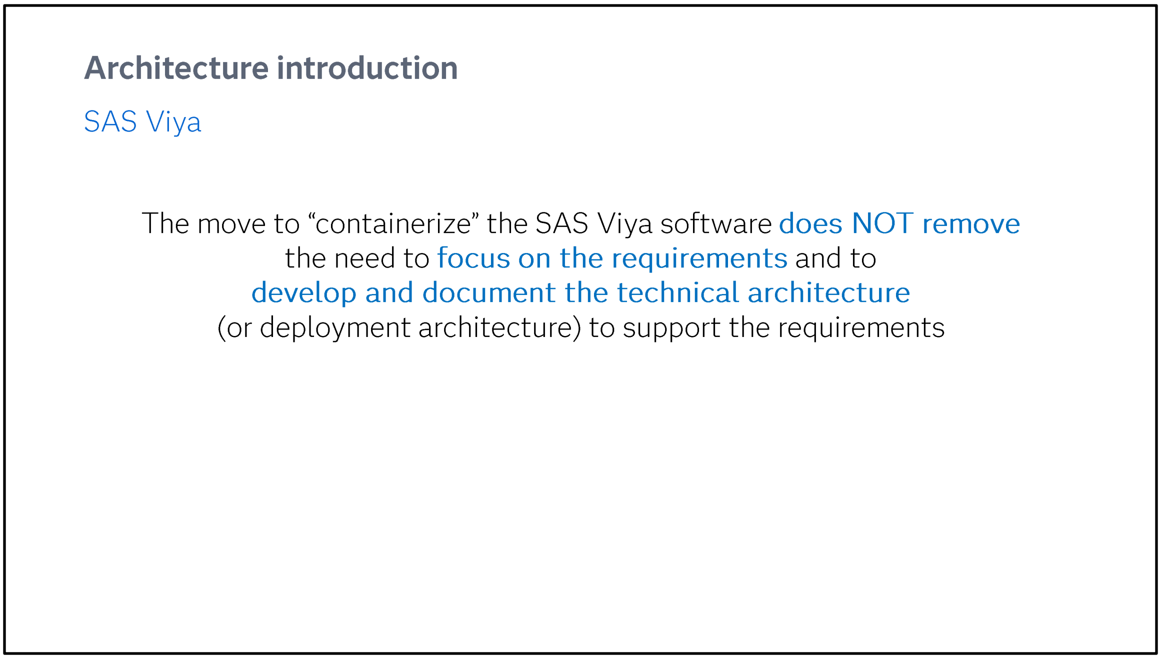
Architecture introduction

SAS Viya platform architecture

The purpose of this module is to quickly re-familiarize ourselves with technical architecture design thinking

Let us quickly remind ourselves of technical architecture thinking and why it is important.

These concepts can be generally applied to multi-user software, whether it be from SAS or other software vendors.



Architecture introduction

SAS Viya

The move to “containerize” the SAS Viya software **does NOT remove** the need to **focus on the requirements** and to **develop and document the technical architecture** (or deployment architecture) to support the requirements

Let’s be clear from the outset. The use of containers and Kubernetes doesn’t remove the need to do some technical architecture design.

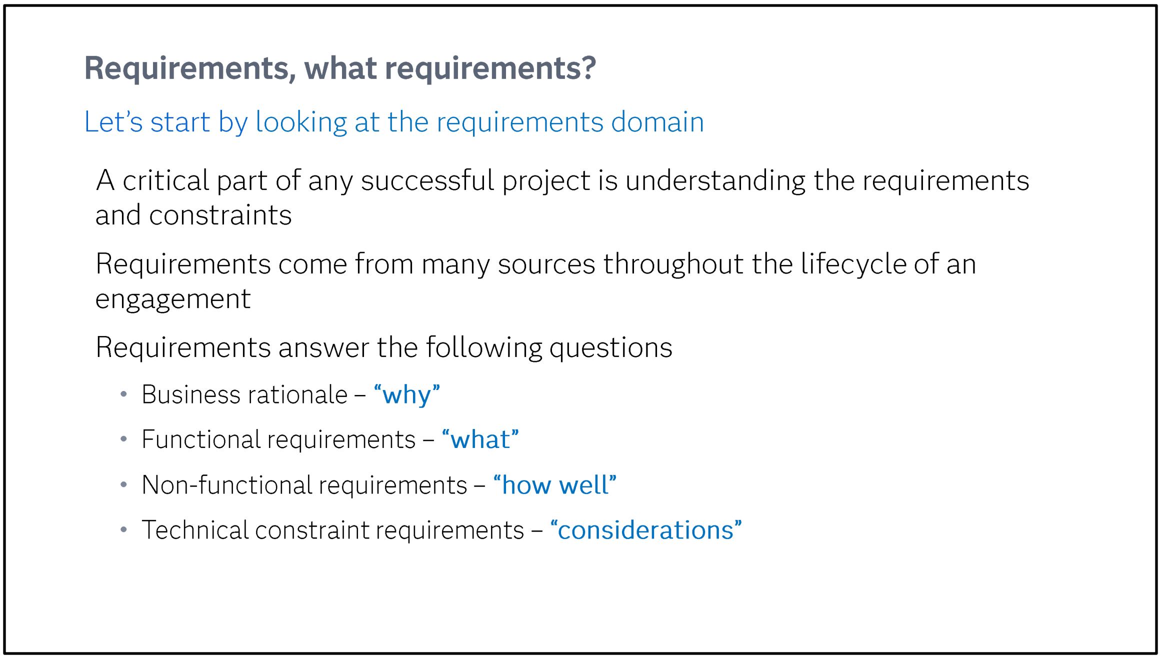
There is an old saying “Failing to plan is planning to fail”.

An organization must allocate sufficient focus and time to the capture and documentation of non-functional requirements when considering the deployment of new software. Failure to do so increases the risk that the organization will not achieve the desired results and business outcomes.

What are the primary drivers that ultimately lead us to a set of requirements?

Is it security of the system? Or perhaps the speed at which the processing analytical workloads can be completed? Or is high availability (HA) a primary driver?

For some organizations they may wish address all those drivers with the same or different weighting of importance.



Requirements, what requirements?

Let’s start by looking at the requirements domain

A critical part of any successful project is understanding the requirements and constraints

Requirements come from many sources throughout the lifecycle of an engagement

Requirements answer the following questions

- Business rationale – **“why”**
- Functional requirements – **“what”**
- Non-functional requirements – **“how well”**
- Technical constraint requirements – **“considerations”**

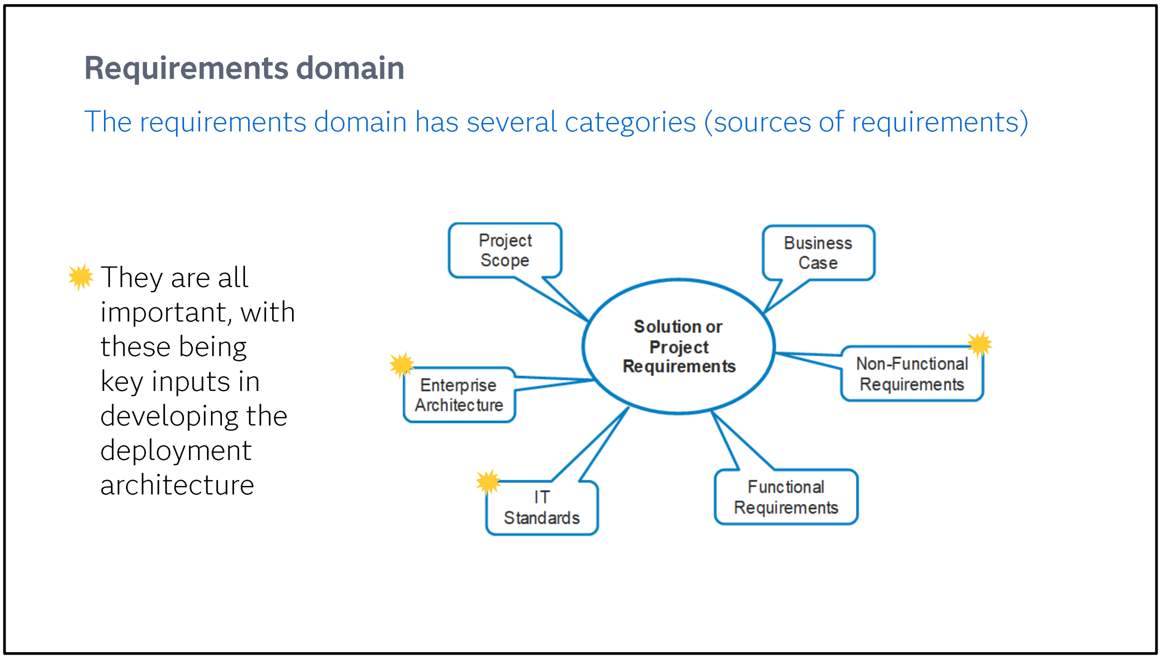
Back to some good old fashion basics, in that the organization needs to have sight of, if the technology provided is going to be a success enabler. In addition when working with additional parties such as SAS professional services and or a systems integrator, the clarity that documented requirements brings is immeasurable.

Both the organization and its partners need understand what the organizations team are ultimately trying to achieve and what technologies they need to achieve their end business goals. This is the why of what is being done.

The organization will likely have multiple stakeholders, each of whom will have identified functional requirements. For functional requirements the examples could be that the system performs credit risk analysis, that there are programming user interfaces, there are point and click features for creating reports, and so on.

Along the way, the organizations stakeholders will discuss how performant the SAS environment will be required to perform. This could range from how fast a particular analytical routine will complete against a known table of data, or how easy a non-programmer can complete a set of tasks, or migrate content or update the environment with patches or a new release.

Wrapping it up we have considerations .These could be technical, financial and time based constraints. For example there could be a constraint requiring Viya to use a specific authentication or encryption method, or for Viya to write to specific type of relational database or storage technology.



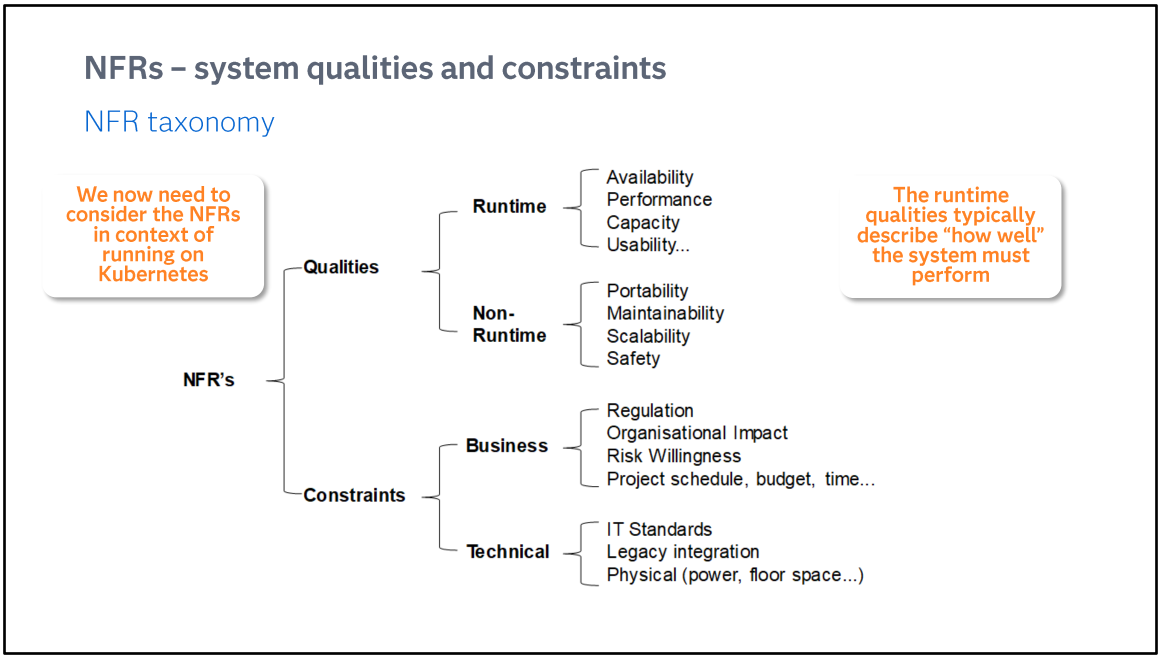
Requirements domain

The requirements domain has several categories (sources of requirements)

- They are all important, with these being key inputs in developing the deployment architecture

From a technical architecture perspective these are all important. The IT standards and non-functional requirements are key inputs. These are often driven by or derived from the enterprise architecture principles.

Business case and functional requirements are key inputs during the technology selection phase for an organizations business project. While the enterprise architecture, IT standards and NFRs are key inputs to the design process. The project scope and business case also provide an input, especially on cost and the desired business outcomes.



NFRs – system qualities and constraints

NFR taxonomy

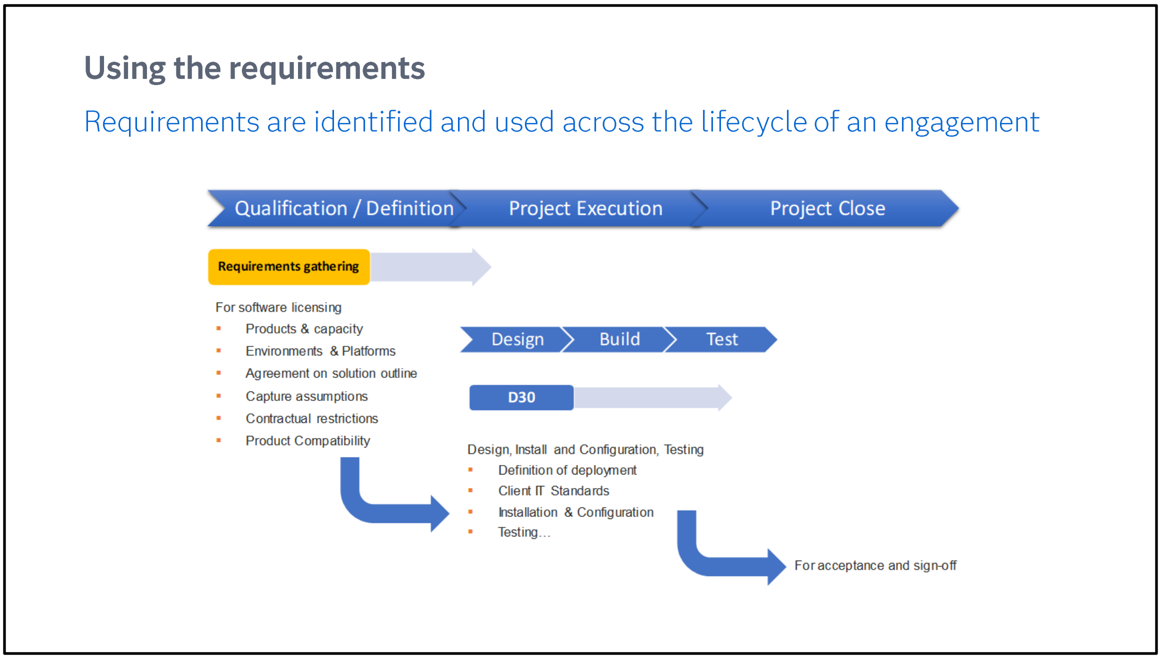
We now need to consider the NFRs in context of running on Kubernetes

- NFR's
  - Qualities
    - Runtime
      - Availability
      - Performance
      - Capacity
      - Usability...
    - Non-Runtime
      - Portability
      - Maintainability
      - Scalability
      - Safety
  - Constraints
    - Business
      - Regulation
      - Organisational Impact
      - Risk Willingness
      - Project schedule, budget, time...
    - Technical
      - IT Standards
      - Legacy integration
      - Physical (power, floor space...)

The runtime qualities typically describe "how well" the system must perform

We can further break down the non-functional requirements (NFRs) into a few areas. These broadly fall into two categories: Qualities and Constraints.

The qualities of a system can be grouped into two areas, runtime and non-runtime. Most of the technical architecture content provided within learning materials provided by SAS focus on the runtime, non-runtime and technical constraints areas. Whilst business constraints are important, they are typically addressed by other stakeholders within a project rather the technical architects.



Using the requirements

Requirements are identified and used across the lifecycle of an engagement

For software licensing

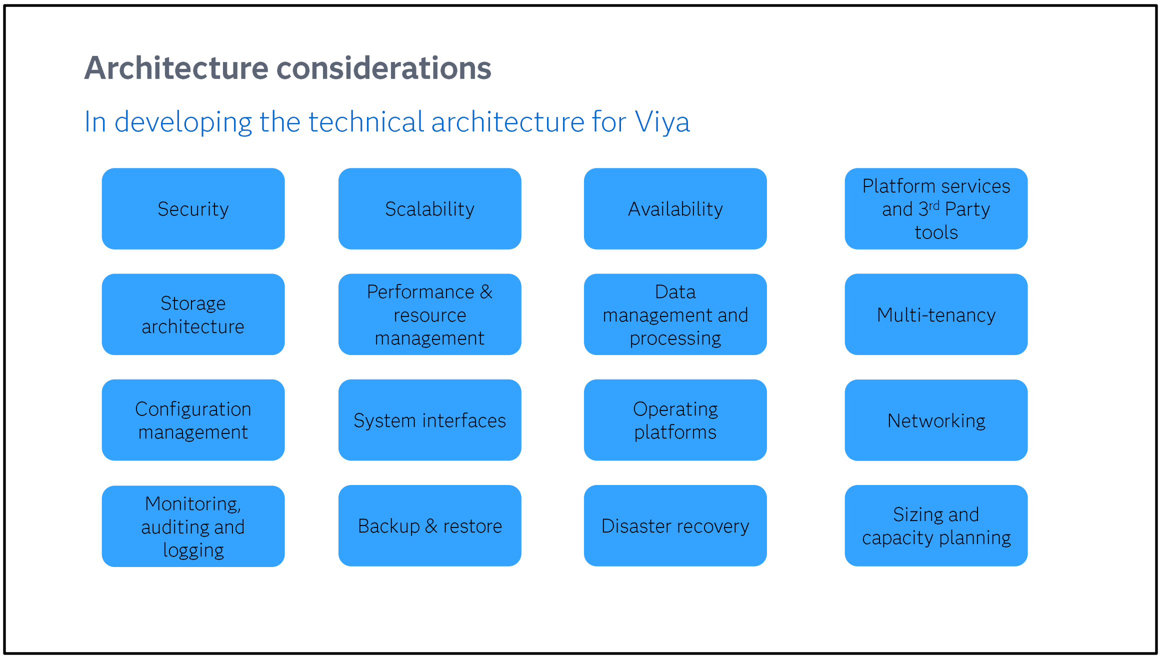
- Products & capacity
- Environments & Platforms
- Agreement on solution outline
- Capture assumptions
- Contractual restrictions
- Product Compatibility

Design, Install and Configuration, Testing

- Definition of deployment
- Client IT Standards
- Installation & Configuration
- Testing…

For acceptance and sign-off

Here we illustrate that requirements are used throughout the project lifecycle. They start during the phase to qualify or define the opportunity, through to the design, build and test phases of the project. If you are curious, the term “D30” refers to a document that has been used to document the architecture design for a customer organization when they have engaged SAS Professional Services teams. It typically includes the architecture requirements, constraints and proposed designs so that both the customer organization and SAS team have a common reference point.
Sometimes you will also hear the term D-A-D or “Dad” referring to the design document, where DAD is the acronym for Detailed Architecture Design document.



Architecture considerations

In developing the technical architecture for Viya

Security	Scalability	Availability	Platform services and 3rd Party tools
Storage architecture	Performance & resource management	Data management and processing	Multi-tenancy
Configuration management	System interfaces	Operating platforms	Networking
Monitoring, auditing and logging	Backup & restore	Disaster recovery	Sizing and capacity planning

When we think about the non-functional requirements and constraints, we can use the considerations listed here (which are based on qualities and constraints) as a good starting point.

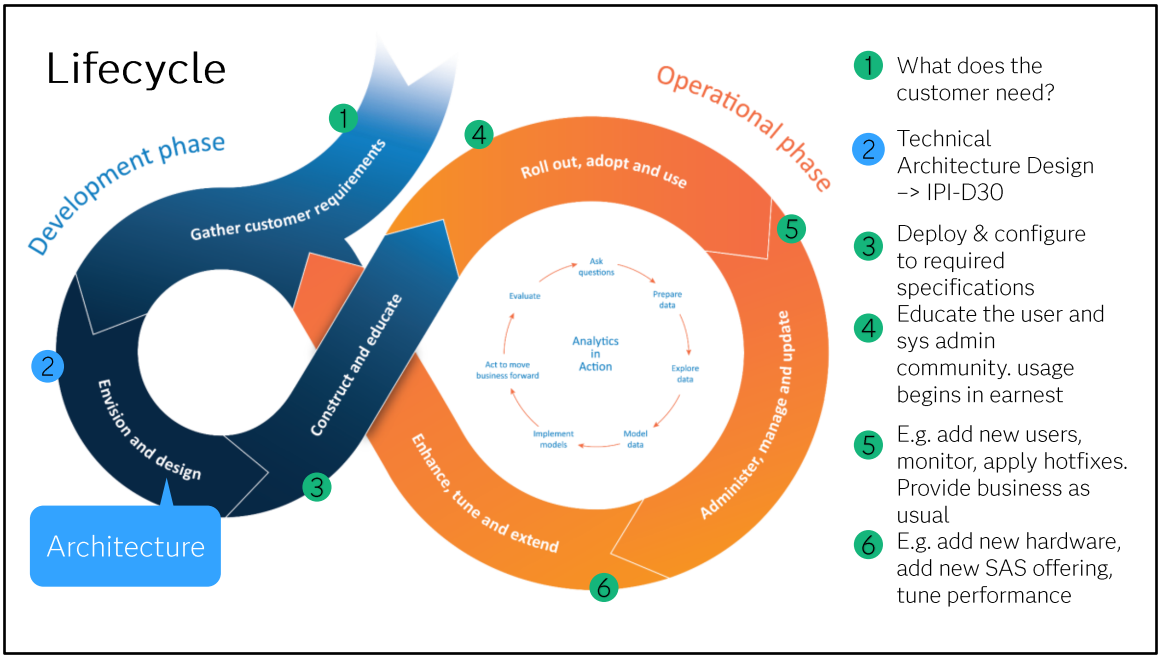
Even when using Kubernetes, these requirements and constraints need to be considered so that the organization is comfortable their SAS environment will be fit for purpose.

So, taking a few of the considerations as an example:

Operating platforms for SAS Viya – Cloud Managed or “self-installed” Kubernetes cluster? Is the Kubernetes platform in the Cloud or on-premises?

Monitoring and logging changed significantly between Viya 3.5 and Viya 4. What are the new considerations and constraints at Viya 4?

Platform services and 3rd Party tools – there are a number of third party required software components that we need for the Viya software to install and run.



Lifecycle

1 What does the customer need?

2 Technical Architecture Design –> IPI-D30

3 Deploy & configure to required specifications

4 Educate the user and sys admin community. usage begins in earnest

5 E.g. add new users, monitor, apply hotfixes. Provide business as usual

6 E.g. add new hardware, add new SAS offering, tune performance

This slide puts into context the inputs of technical architecture design when we think about the SAS software lifecycle from the viewpoint of an organization.

The customer organization and SAS begin the process of determining what technology is needed to meet a set of business goals.

Then comes the architecture design based on the stated requirements and constraints. This is followed by

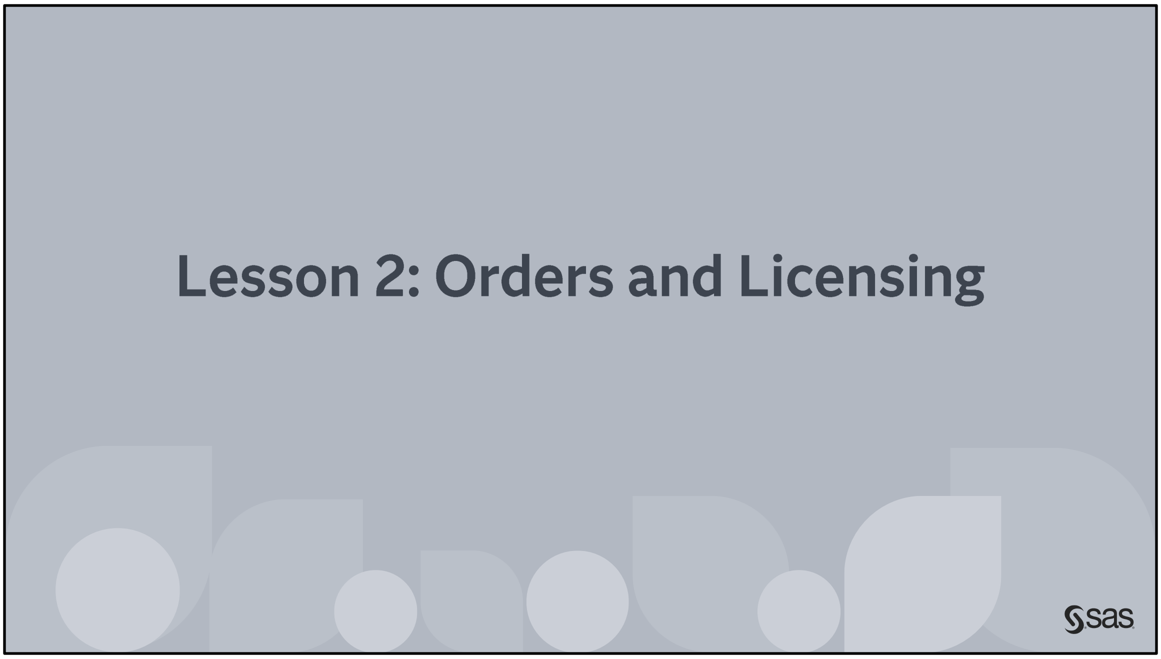
the building or deploying of the SAS software.

Once the software is available to the user community, the software gets used to deliver business value. This phase normally includes educating the user community and as well initial administration of the platform.

During the lifetime of the SAS platform, there will be a need to add new users, change authorization to content and generally feed and care for the platform. This is sometimes referred to as day-to-day operations.

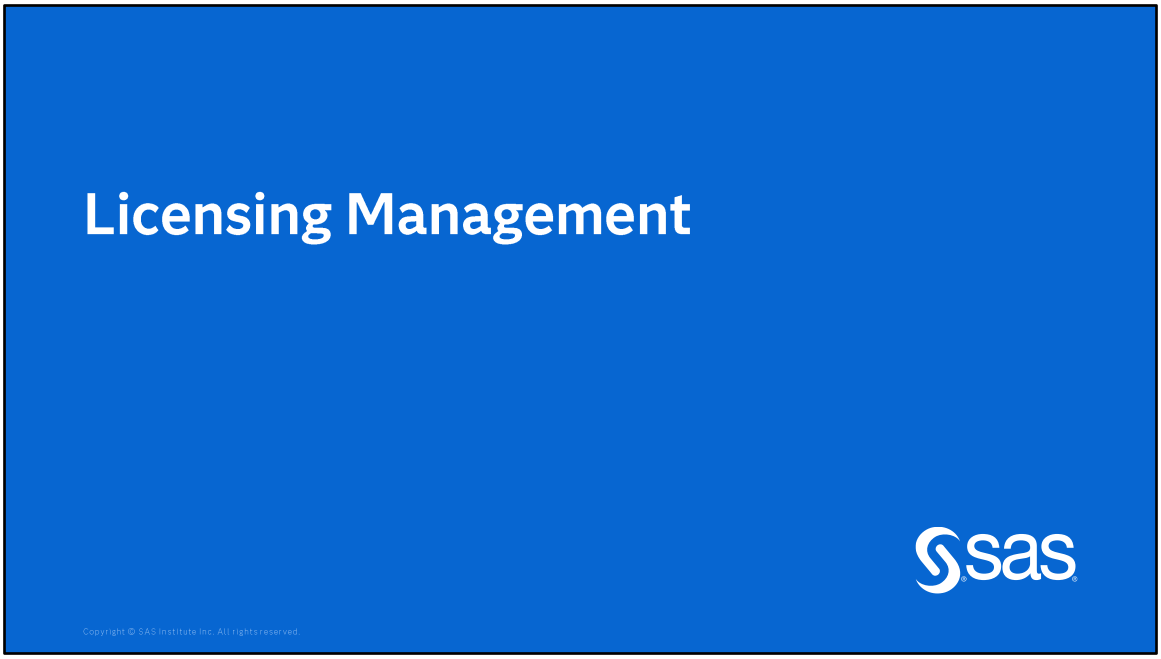
Tweaking the SAS platform to improve security, adding new hardware to improve performance or adding new SAS software can be options to enhance, tune and extend the platform.

As you can see, the Development phase is smaller than the Operational phase. That’s because the environment could be operational for multiple years, where as the architecture work may take only a matter of weeks. However, for an operational system to achieve a desired longevity, the work produced at the architecture stage will be a significant contributor to the longevity of the system.

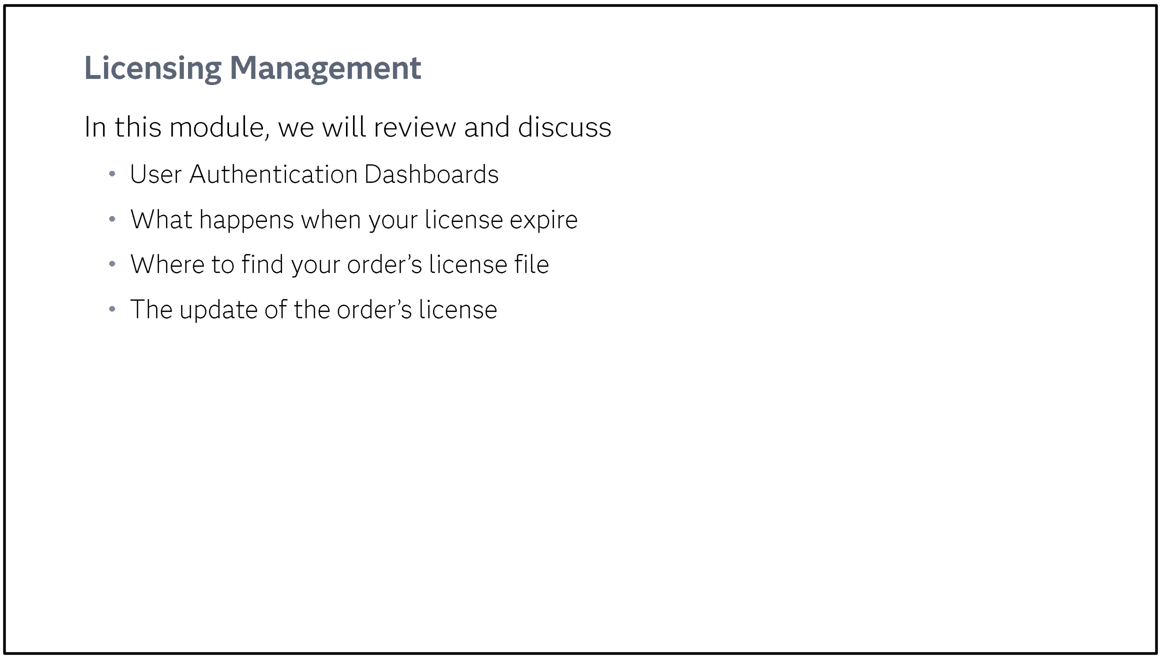


Lesson 2: Orders and Licensing

2.1 Licensing



Licensing Management

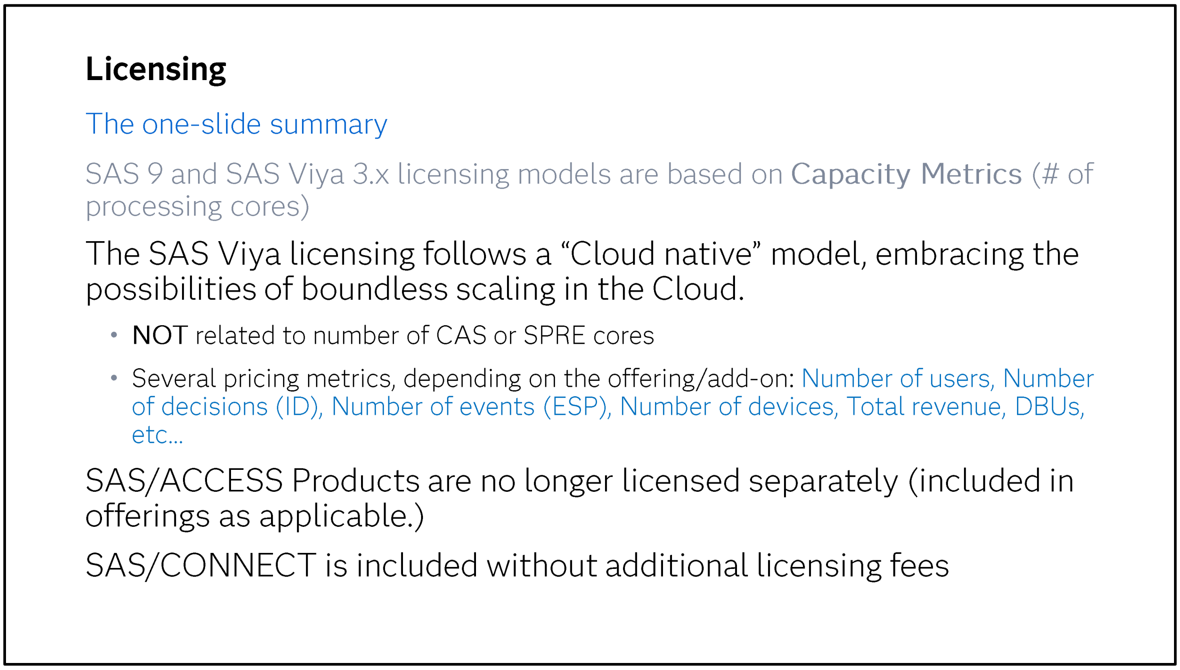


Licensing Management

In this module, we will review and discuss

- User Authentication Dashboards
- What happens when your license expire
- Where to find your order's license file
- The update of the order's license

In this module, we want to:
review the user authentication dashboards,
see what happens when your license expires,
see where to find your order's license files,
and discuss the update of the order's license.



Licensing

The one-slide summary

SAS 9 and SAS Viya 3.x licensing models are based on **Capacity Metrics** (# of processing cores)

The SAS Viya licensing follows a “Cloud native” model, embracing the possibilities of boundless scaling in the Cloud.

- NOT related to number of CAS or SPRE cores
- Several pricing metrics, depending on the offering/add-on: Number of users, Number of decisions (ID), Number of events (ESP), Number of devices, Total revenue, DBUs, etc…

SAS/ACCESS Products are no longer licensed separately (included in offerings as applicable.)

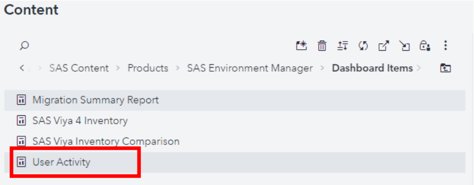
SAS/CONNECT is included without additional licensing fees

The licensing model in SAS 9 and Viya 3.5 is based on the capacity metrics, which is the number of CPU or physical cores.
With Viya 4, we are going with a model based on the business size or number of users (Authorized Users) rather than by core counts.
Cloud environments offers greater resource elasticity, and this new licensing model allows the CPU resources allocated to the SAS software to be scaled up and down as needed in a Cloud environment.

One of the other key differences is that with SAS Viya, all SAS/ACCESS and SAS/CONNECT products are now shipped in most of the offerings without additional fees.

User Authentication Dashboard

- SAS Environment Manager includes a “User Activity” dashboard, based on the AUDIT table to keep track of user’s activity.



Content

SAS Content > Products > SAS Environment Manager > Dashboard Items >

- Migration Summary Report
- SAS Viya 4 Inventory
- SAS Viya Inventory Comparison
- User Activity

- A report to keep track of the Authenticated Users is available in the “User Authentication” tab.
- It can help to identify any gap between the number/types of licensed users and the real number of users who are using the environment.

SAS Environment manager, which is really the SAS Administrator’s UI, includes a dashboard with a set of reports to follow the Viya platform activity.
One of the reports is called “User Authentication”.

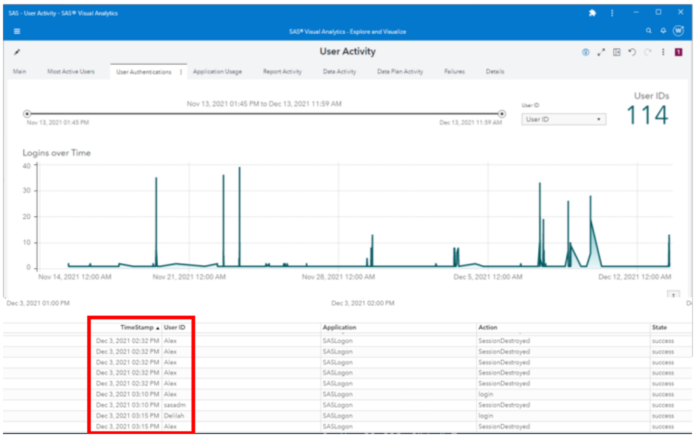
It can be used to track the Authenticated Users and help to identify any gap between the number/types of licensed users and the real number of users who are using the environment.

User Authentication Dashboard

Refreshed every 2 hours by default

Follow logins over time.

List authenticated users



TimeStamp ▲	User ID	Application	Action	State
Dec 3, 2021 02:32 PM	Alex	SASLogon	SessionDestroyed	success
Dec 3, 2021 02:32 PM	Alex	SASLogon	SessionDestroyed	success
Dec 3, 2021 02:32 PM	Alex	SASLogon	SessionDestroyed	success
Dec 3, 2021 02:32 PM	Alex	SASLogon	SessionDestroyed	success
Dec 3, 2021 02:32 PM	Alex	SASLogon	SessionDestroyed	success
Dec 3, 2021 03:10 PM	Alex	SASLogon	login	success
Dec 3, 2021 03:10 PM	sasadm	SASLogon	SessionDestroyed	success
Dec 3, 2021 03:15 PM	Delilah	SASLogon	login	success
Dec 3, 2021 03:15 PM	Alex	SASLogon	SessionDestroyed	success

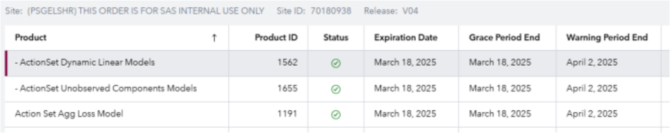
The dashboard data are refreshed every two hours by default, but you can change that.
So, before the first two hours of a freshly deployed environment, you will not see anything.
But after that, there will be a task in the background that will load the records from the audit table in CAS, and it will also populate this kind of report.

As you can see in the screenshot, you can follow the logins over time.
You can also list the authenticated users, which can be very useful when, as a customer, you want to ensure you remain compliant with the number of users defined in your licensing agreement.

The dashboard also lists the authenticated users, and you have additional tabs like “report activity”, “data activity”, and others, giving you more insights on what the end users are doing in the environment.

License expiration

- Since April 1st, 2024, the SAS Viya orders have :
  - No GRACE period
  - A default WARNING period: expiration date + 15 days

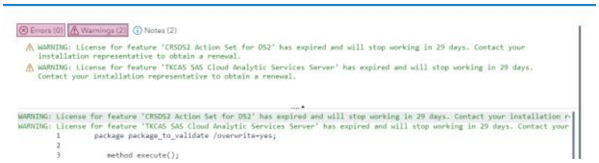


Site: (PSGELSHR) THIS ORDER IS FOR SAS INTERNAL USE ONLY   Site ID: 70180938   Release: V04

Product ↑	Product ID	Status	Expiration Date	Grace Period End	Warning Period End
- ActionSet Dynamic Linear Models	1562	✓	March 18, 2025	March 18, 2025	April 2, 2025
- ActionSet Unobserved Components Models	1655	✓	March 18, 2025	March 18, 2025	April 2, 2025
Action Set Agg Loss Model	1191	✓	March 18, 2025	March 18, 2025	April 2, 2025

SAS now send renewal invoices 30 days before the license expiration date.
Customers will receive a warning for 15 days after the expiration date.
…Then the software will stop functioning…

Example of what you'll see if your system is about to expire ("Warning period").



```
⚠ WARNING: License for feature 'CRSDS2 Action Set for DS2' has expired and will stop working in 29 days. Contact your
  installation representative to obtain a renewal.
⚠ WARNING: License for feature 'TKCAS SAS Cloud Analytic Services Server' has expired and will stop working in 29 days.
  Contact your installation representative to obtain a renewal.

WARNING: License for feature 'CRSDS2 Action Set for DS2' has expired and will stop working in 29 days. Contact your installation r
WARNING: License for feature 'TKCAS SAS Cloud Analytic Services Server' has expired and will stop working in 29 days. Contact your
      1         package package_to_validate /overwrite=yes;
      2
      3            method execute();
```

Since April 1st, 2024, the GRACE period has been set to 0 for all the Viya orders and there is a default WARNING period of 15 days after the expiration date.

Before a license is about to expire, the SAS representative at the customer site, will receive the renewal invoices along with the corresponding full-term authorization code.
The renewal invoices will be sent exactly 30 days before the license expiration date
This allows customers to apply the license authorization code before the previous one expires, ensuring uninterrupted service.

If the new authorization codes are not entered, the customer will enter the WARNING period, and the users will start to see Warning messages as on the screen.
At the end of the WARNING period the software will stop functioning.

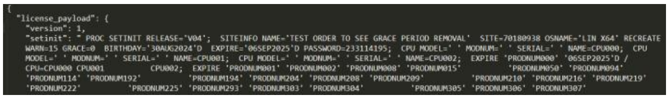
Licensing

Where to find the license file ?

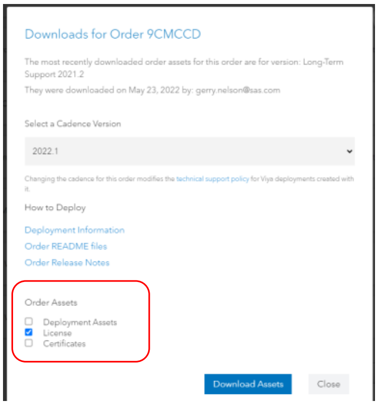
The license file can be downloaded from the my.sas.com portal

The license file is only available with the `.jwt` format (JSON Web Token)

- This file is encoded, not human readable ☹
- But not too complicated to decode ☺



```
{
  "license_payload": {
    "version": 1,
    "setinit": " PROC SETINIT RELEASE='V04';  SITEINFO NAME='TEST ORDER TO SEE GRACE PERIOD REMOVAL'  SITE=70180938 OSNAME='LIN X64' RECREATE
    WARN=15 GRACE=0  BIRTHDAY='30AUG2024'D  EXPIRE='06SEP2025'D PASSWORD=233114195;  CPU MODEL=' ' MODNUM=' ' SERIAL=' ' NAME=CPU000;  CPU
    MODEL=' ' MODNUM=' ' SERIAL=' ' NAME=CPU001;  CPU MODEL=' ' MODNUM=' ' SERIAL=' ' NAME=CPU002;  EXPIRE 'PRODNUM000' '06SEP2025'D /
    CPU=CPU000 CPU001          CPU002;  EXPIRE 'PRODNUM001' 'PRODNUM002' 'PRODNUM008' 'PRODNUM015'          'PRODNUM050' 'PRODNUM094'
    'PRODNUM114' 'PRODNUM192'          'PRODNUM194' 'PRODNUM204' 'PRODNUM208' 'PRODNUM209'          'PRODNUM210' 'PRODNUM216' 'PRODNUM219'
    'PRODNUM222'          'PRODNUM225' 'PRODNUM293' 'PRODNUM303' 'PRODNUM304'          'PRODNUM305' 'PRODNUM306' 'PRODNUM307'
```



Downloads for Order 9CMCCD

The most recently downloaded order assets for this order are for version: Long-Term Support 2021.2

They were downloaded on May 23, 2022 by: gerry.nelson@sas.com

Select a Cadence Version

2022.1

Changing the cadence for this order modifies the technical support policy for Viya deployments created with it.

How to Deploy

Deployment Information

Order README files

Order Release Notes

Order Assets

☐ Deployment Assets
☑ License
☐ Certificates

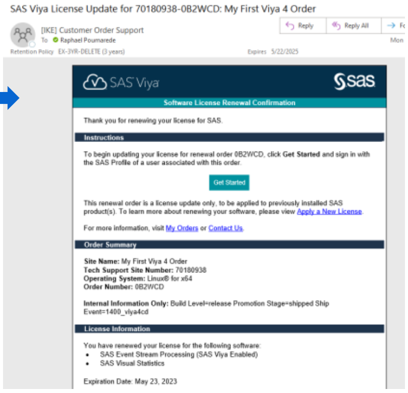
Now if you want to update your license, you will need the new license file, so where can you find it?
It can be found and downloaded from the my.sas.com portal.
It comes as a JWT (Json Web Token) file, which is not a human readable format.
A base64 encoding is used to hide the content but is very easy to decode.

Note that you can also automate the license file download with the orders CLI (the orders CLI is a command-line tool to interact programmatically with my.sas.com portal).

Licensing

Update your software license

- When the license is renewed, an email is sent to the customer with a link to the my.sas.com portal.
- The ORDER number remains the same (for a standard customer order).
- The **expiration date** is updated in the my.sas.com portal.



SAS Viya License Update for 70180938-0B2WCD: My First Viya 4 Order

[IKE] Customer Order Support
To Raphael Poumarede
Retention Policy EX-3YR-DELETE (3 years) Expires 5/22/2025

Software License Renewal Confirmation

Thank you for renewing your license for SAS.

Instructions

To begin updating your license for renewal order 0B2WCD, click **Get Started** and sign in with the SAS Profile of a user associated with this order.

This renewal order is a license update only, to be applied to previously installed SAS product(s). To learn more about renewing your software, please view Apply a New License.

For more information, visit My Orders or Contact Us.

Order Summary

Site Name: My First Viya 4 Order
Tech Support Site Number: 70180938
Operating System: Linux® for x64
Order Number: 0B2WCD

Internal Information Only: Build Level=release Promotion Stage=shipped Ship Event=1400_viya4cd

License Information

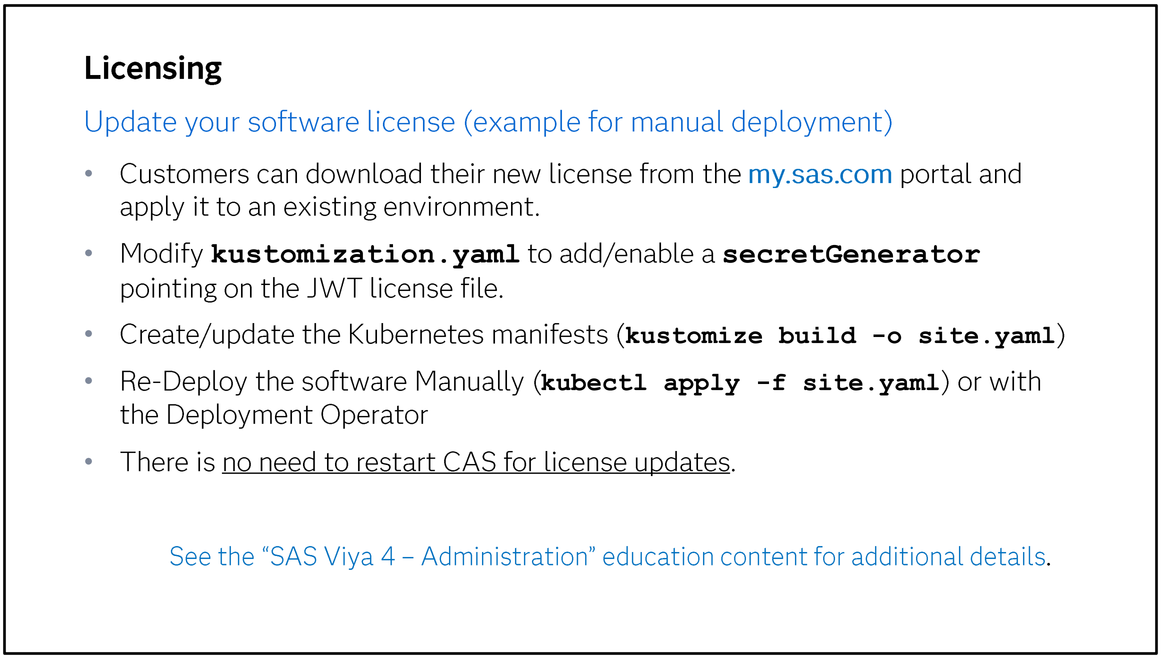
You have renewed your license for the following software:

- SAS Event Stream Processing (SAS Viya Enabled)
- SAS Visual Statistics

Expiration Date: May 23, 2023

So, what is the process when you need to update the license?

An email is sent to the customer representative after the software license has been renewed.
The ORDER number remains the same, but the expiration date is updated in the my.sas.com portal.



Licensing

Update your software license (example for manual deployment)

- Customers can download their new license from the my.sas.com portal and apply it to an existing environment.
- Modify **`kustomization.yaml`** to add/enable a **`secretGenerator`** pointing on the JWT license file.
- Create/update the Kubernetes manifests (**`kustomize build -o site.yaml`**)
- Re-Deploy the software Manually (**`kubectl apply -f site.yaml`**) or with the Deployment Operator
- There is no need to restart CAS for license updates.

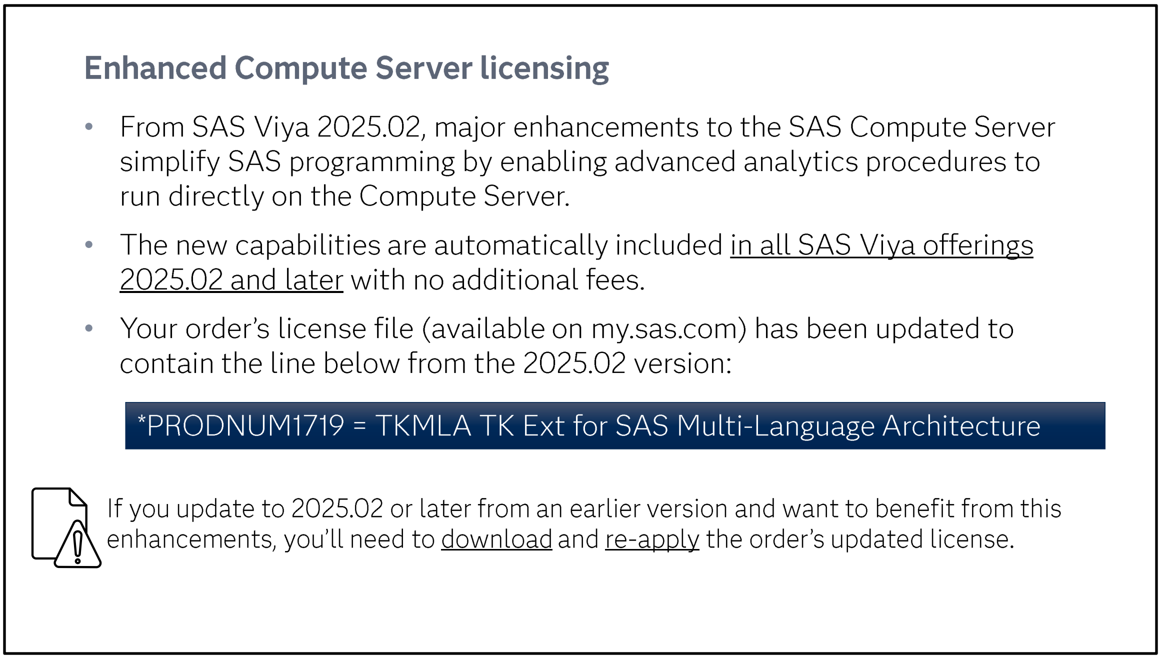
See the "SAS Viya 4 – Administration" education content for additional details.

Now, how do you update the software license in your running environment?

Here are the summarized steps to update the license in the case of a manual deployment. Basically, you'll have to download the new license file, modify the kustomization.yaml file to reference it and then update your manifest and deployment (but that will depend on your deployment method).

The disruption should be minimal, there is no need to restart CAS to pick up the new license.

See the official documentation or the SAS Viya Administration education modules for more details and hands-on on how to renew the license.



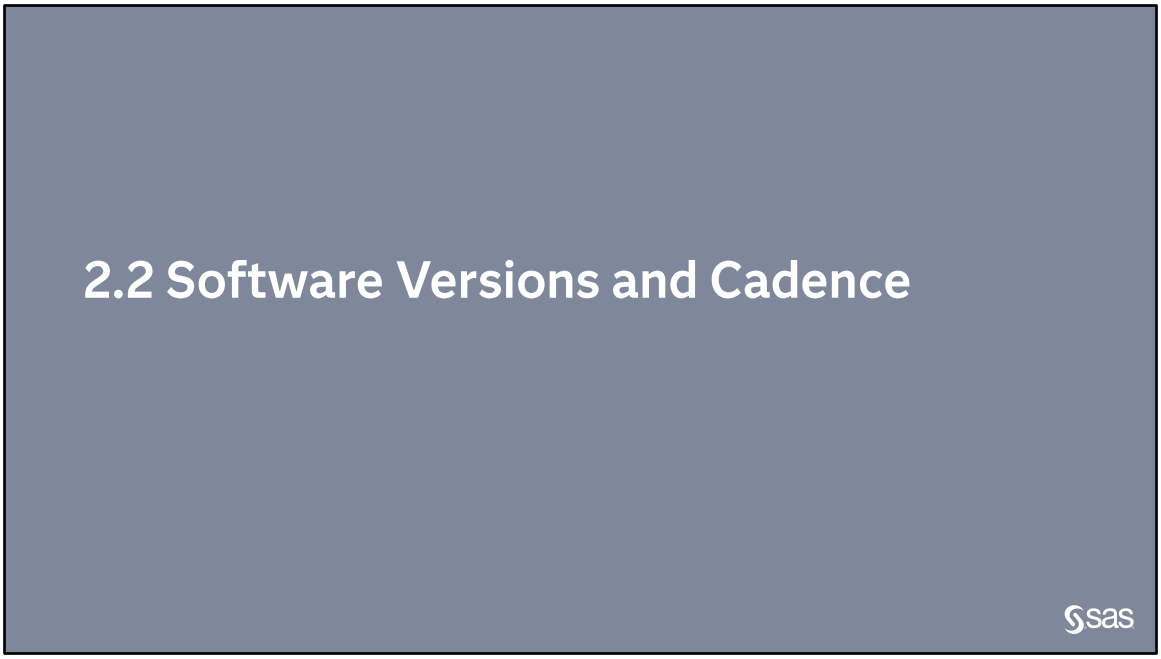
Enhanced Compute Server licensing

- From SAS Viya 2025.02, major enhancements to the SAS Compute Server simplify SAS programming by enabling advanced analytics procedures to run directly on the Compute Server.
- The new capabilities are automatically included in all SAS Viya offerings 2025.02 and later with no additional fees.
- Your order's license file (available on my.sas.com) has been updated to contain the line below from the 2025.02 version:

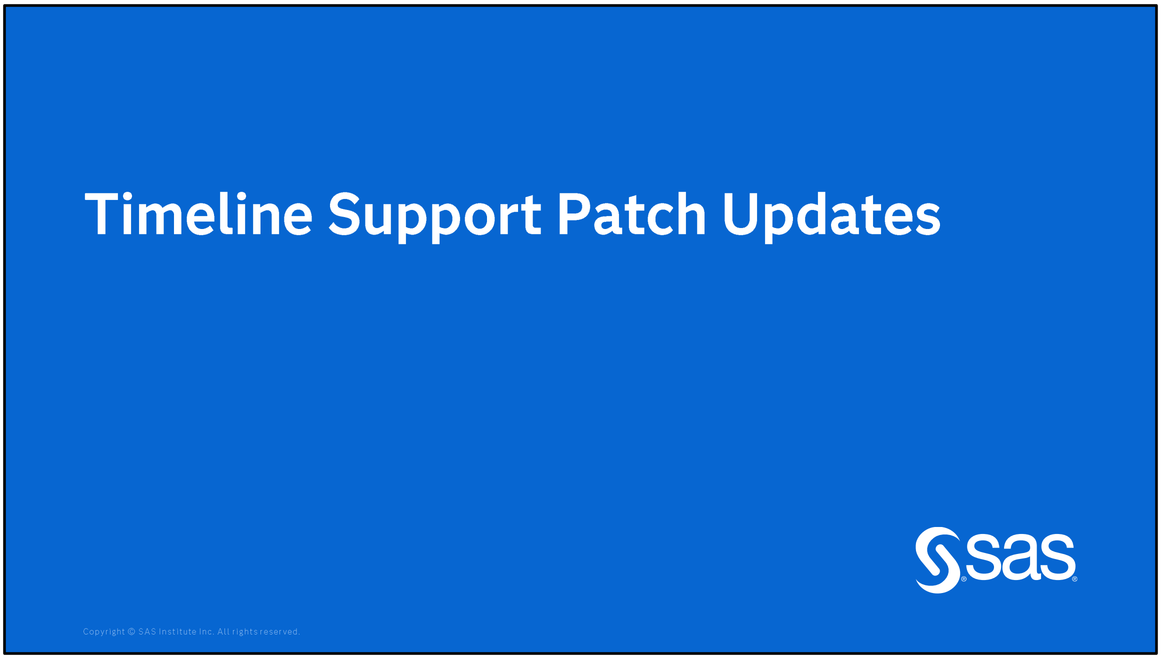
```
*PRODNUM1719 = TKMLA TK Ext for SAS Multi-Language Architecture
```

If you update to 2025.02 or later from an earlier version and want to benefit from this enhancements, you'll need to download and re-apply the order's updated license.

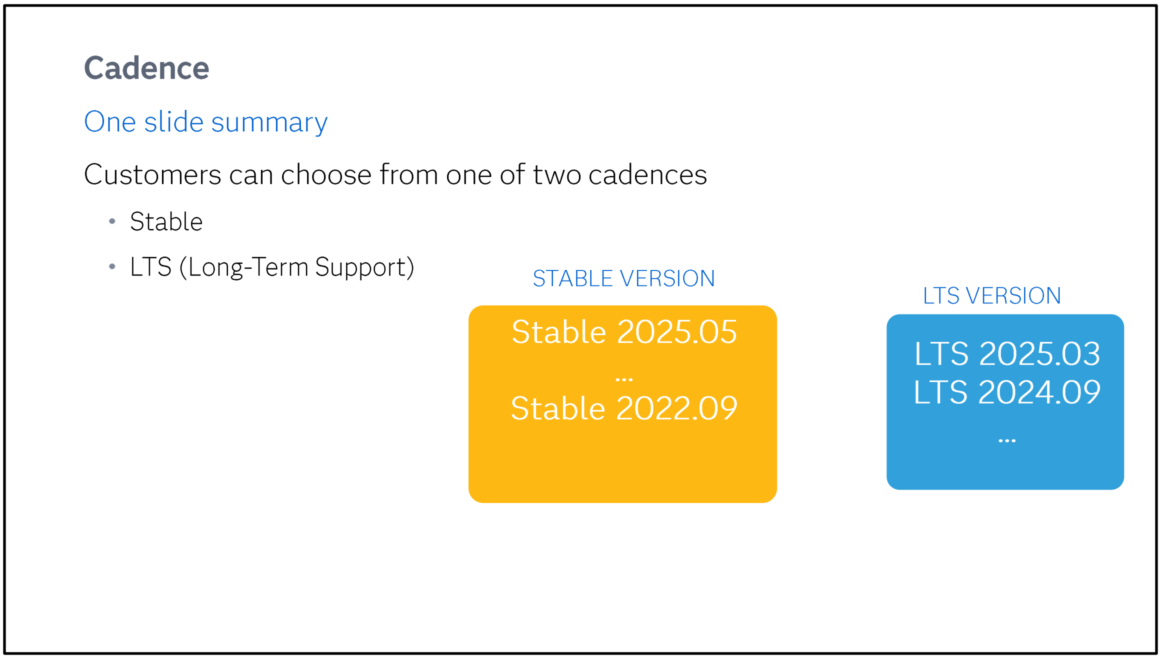
Starting in SAS Viya 2025.02, major enhancements to the SAS Compute Server simplify SAS programming by enabling advanced analytics procedures to run directly on the Compute Server - eliminating the need for a CAS server in many cases.
These updates introduce in-process execution, multithreading support, and expanded native Python integration, providing greater flexibility and efficiency.
These new capabilities are automatically included in all SAS Viya offerings with no additional fees, but they perform a license check to verify that they are executing in a licensed environment.
On the my.sas.com portal, the license files and deployment assets have been updated to include the required license key from the version 2025.02.
But if you are running a Viya version older than 2025.02 and want to update to a more recent version to benefit from this enhancements, you'll need to download and re-apply the order's updated license.



2.2 Software Versions and Cadence



Timeline Support Patch Updates



Cadence

One slide summary

Customers can choose from one of two cadences

- Stable
- LTS (Long-Term Support)

STABLE VERSION

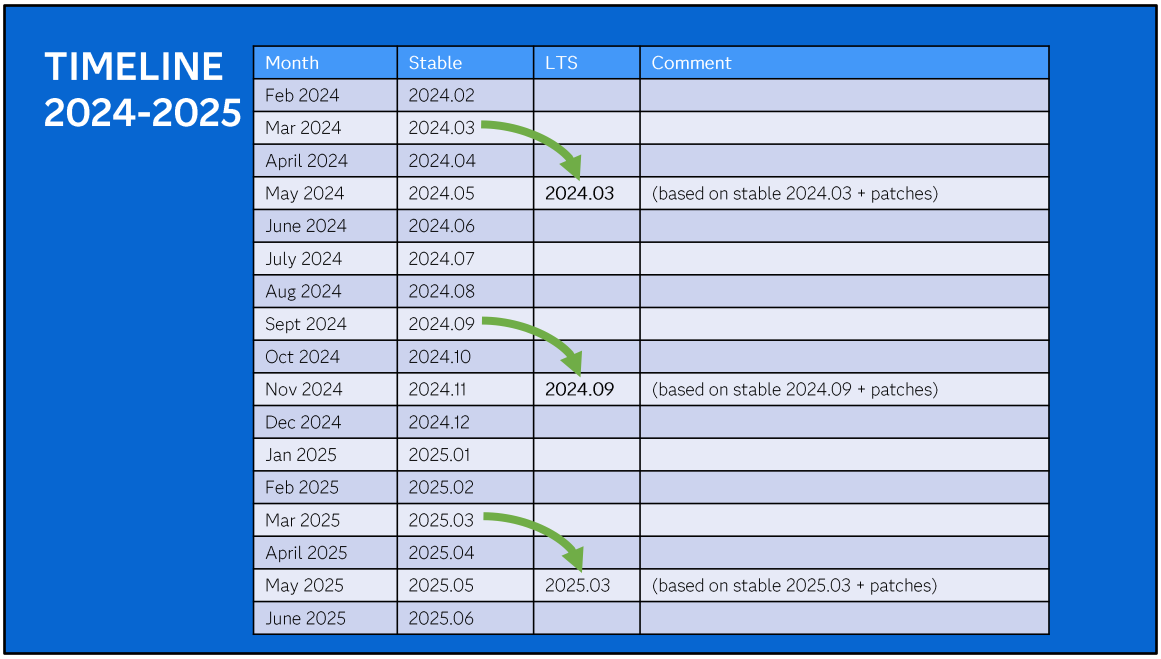
Stable 2025.05
…
Stable 2022.09

LTS VERSION

LTS 2025.03
LTS 2024.09
…

For their Viya 4 platform, customers must choose between two types of cadence :
Either “stable” or LTS (for Long Term Support).

Stable and LTS versions are expressed on 2 digits (like 2025.03) and the prefix of the version either Stable or LTS is mandatory to know which cadence type is used (otherwise, for example if you just say 2024.09, we don’t know if you are talking about the Stable or LTS cadence).



TIMELINE 2024-2025

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		

In this table, you can see the stable version cadence and the LTS version cadence.

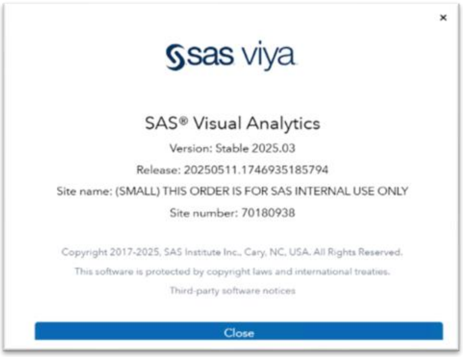
Usually, the LTS versions are released in May and November and are based on the stable version from two months prior.
But sometimes there is an exception to this rule,

for example, LTS 2023.10 was based on the October stable version from one month prior.
Sometimes it can happen because the SAS R and D wants to release some very important features in the LTS version, and those features were not ready for the version two months prior (the two months are a little bit like a buffer).

Continuous Delivery Cadence

All version aligned

All products versions are aligned.
For example, instead of Visual Analytics 8.x and Event Stream Processing 7.x, it’s now Visual Analytics Stable 2025.04 and Event Stream Processing LTS 2025.03



SAS® Visual Analytics

Version: Stable 2025.03

Release: 20250511.1746935185794

Site name: (SMALL) THIS ORDER IS FOR SAS INTERNAL USE ONLY

Site number: 70180938

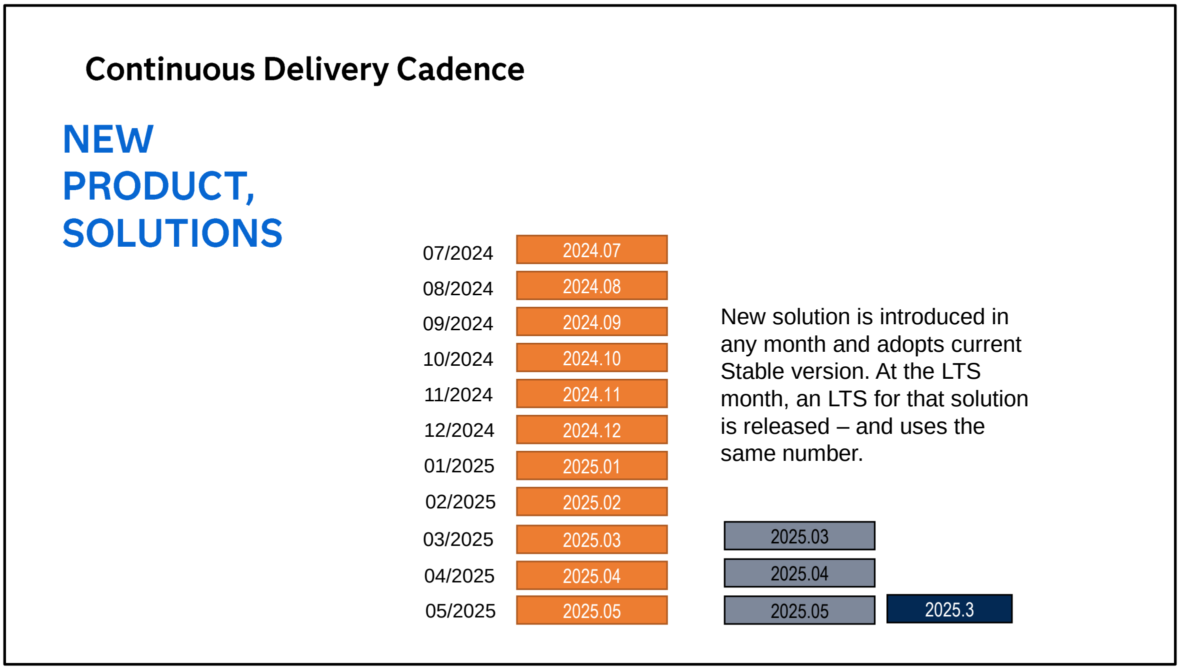
Copyright 2017-2025, SAS Institute Inc., Cary, NC, USA. All Rights Reserved.

This software is protected by copyright laws and international treaties.

Third-party software notices

Another important change is that all products, whether it is Visual Analytics, Model Manager, or Event Stream Processing, have a consistent version scheme.
So for example, instead of referring to “Visual Analytics 8.4” and “Event Stream Processing 7.1”, you refer to Visual Analytics 2023.09 and Event Stream Processing 2023.03.

All products have stable and LTS releases where new features are progressively added on a monthly basis.



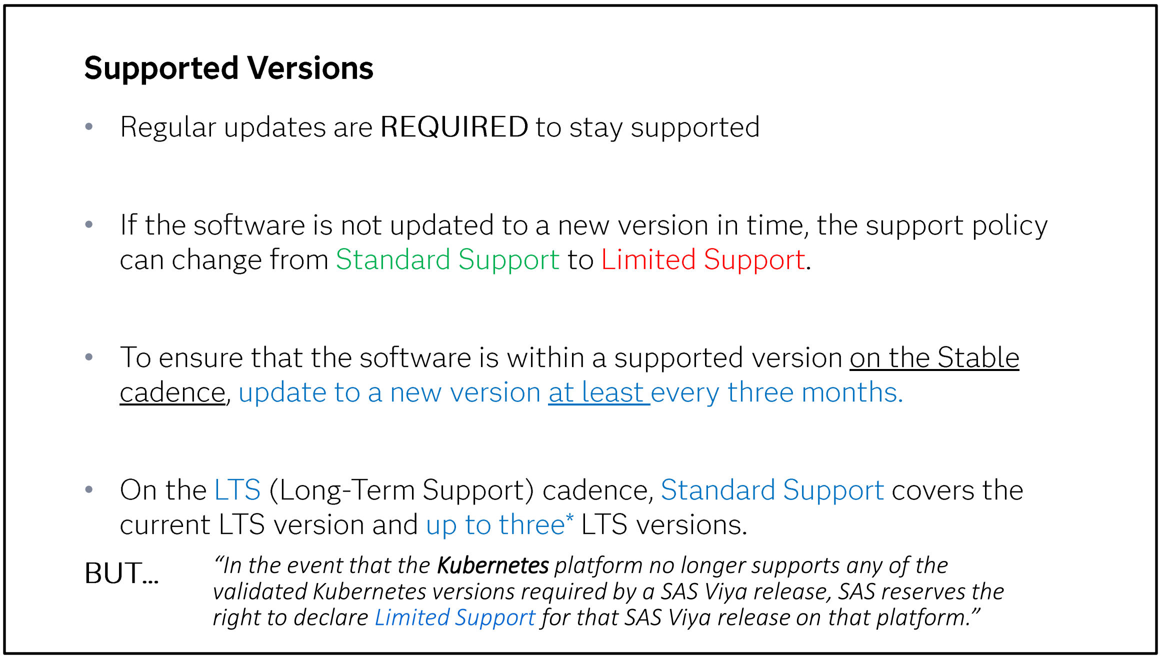
Continuous Delivery Cadence

NEW PRODUCT, SOLUTIONS

07/2024	2024.07		
08/2024	2024.08		
09/2024	2024.09		
10/2024	2024.10		
11/2024	2024.11		
12/2024	2024.12		
01/2025	2025.01		
02/2025	2025.02		
03/2025	2025.03	2025.03	
04/2025	2025.04	2025.04	
05/2025	2025.05	2025.05	2025.3

New solution is introduced in any month and adopts current Stable version. At the LTS month, an LTS for that solution is released – and uses the same number.

You can see in this timeline that new solutions or products that are introduced in any month will adopt the current Stable version.
At the LTS month, when an LTS for that solution is released, it will use the same LTS number as the other offerings.



Supported Versions

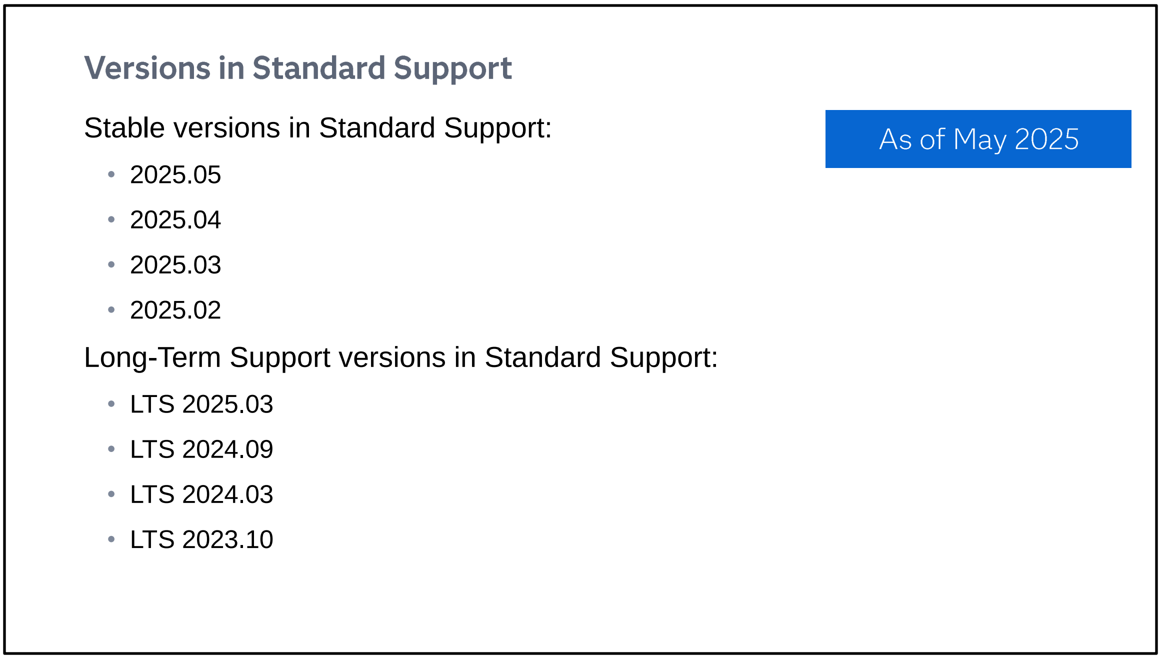
- Regular updates are REQUIRED to stay supported
- If the software is not updated to a new version in time, the support policy can change from Standard Support to Limited Support.
- To ensure that the software is within a supported version on the Stable cadence, update to a new version at least every three months.
- On the LTS (Long-Term Support) cadence, Standard Support covers the current LTS version and up to three* LTS versions.

BUT… *"In the event that the **Kubernetes** platform no longer supports any of the validated Kubernetes versions required by a SAS Viya release, SAS reserves the right to declare Limited Support for that SAS Viya release on that platform."*

To keep the software current, you should update to the latest version for your cadence as soon as it becomes available.
If the software is not updated to a new version in time, the support policy can change from standard support to limited support.
If you are on a stable cadence where you get a new version each month, you need to update to a new version at least every three months. Otherwise, you will fall under the "limited support policy".

On the LTS cadence, the standard support covers the current LTS version and up to three LTS versions. But because you only have a new version every six months, it means that you could potentially be supported for a period of two years (instead of four months with the stable cadence).

However, there is a caveat to this rule, which is that in two years your Kubernetes version will very likely be out of date.
The initial plan was to provide two-year support windows for the customer choosing the LTS version – the LTS cadence. But it's hard to keep this promise because the Kubernetes system itself is evolving much faster.
This specific case will be discussed in further modules.



Versions in Standard Support

As of May 2025

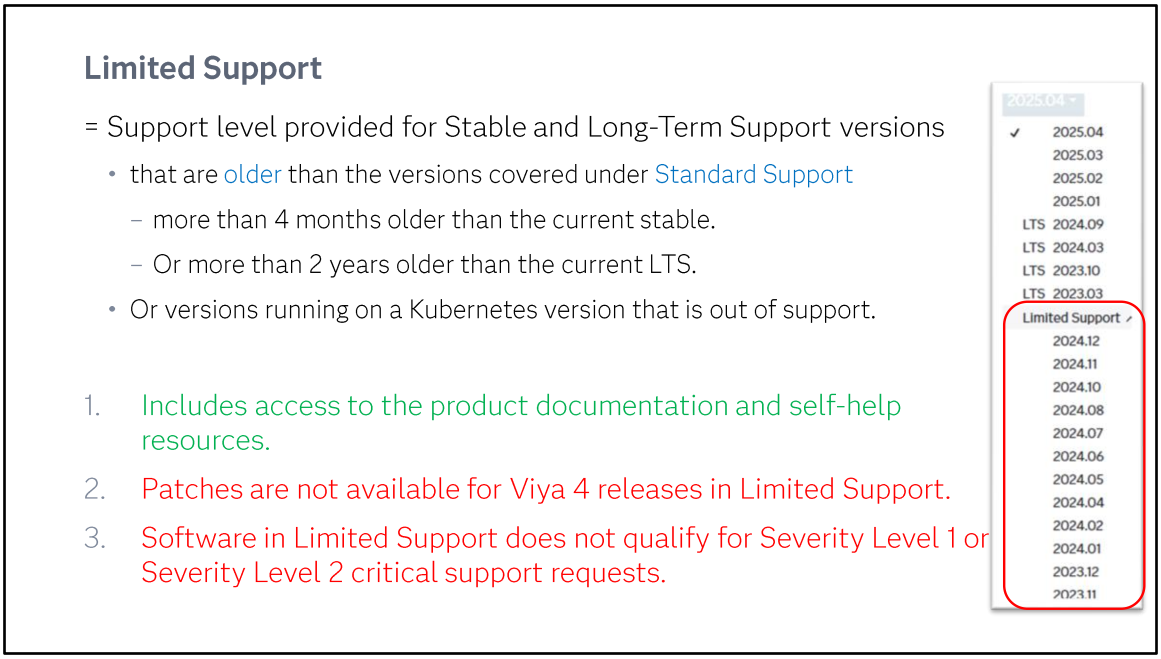
Stable versions in Standard Support:

- 2025.05
- 2025.04
- 2025.03
- 2025.02

Long-Term Support versions in Standard Support:

- LTS 2025.03
- LTS 2024.09
- LTS 2024.03
- LTS 2023.10

Here are the fully supported versions AS OF end of May 2025.
Anything that was older than this falls under ”Limited Support”.
For example, as of May 2025, the July stable version, 2025.01 and the first 2023 LTS version (2023.03) were already under the “Limited support” policy.



Limited Support

= Support level provided for Stable and Long-Term Support versions

- that are older than the versions covered under Standard Support
  - more than 4 months older than the current stable.
  - Or more than 2 years older than the current LTS.
- Or versions running on a Kubernetes version that is out of support.

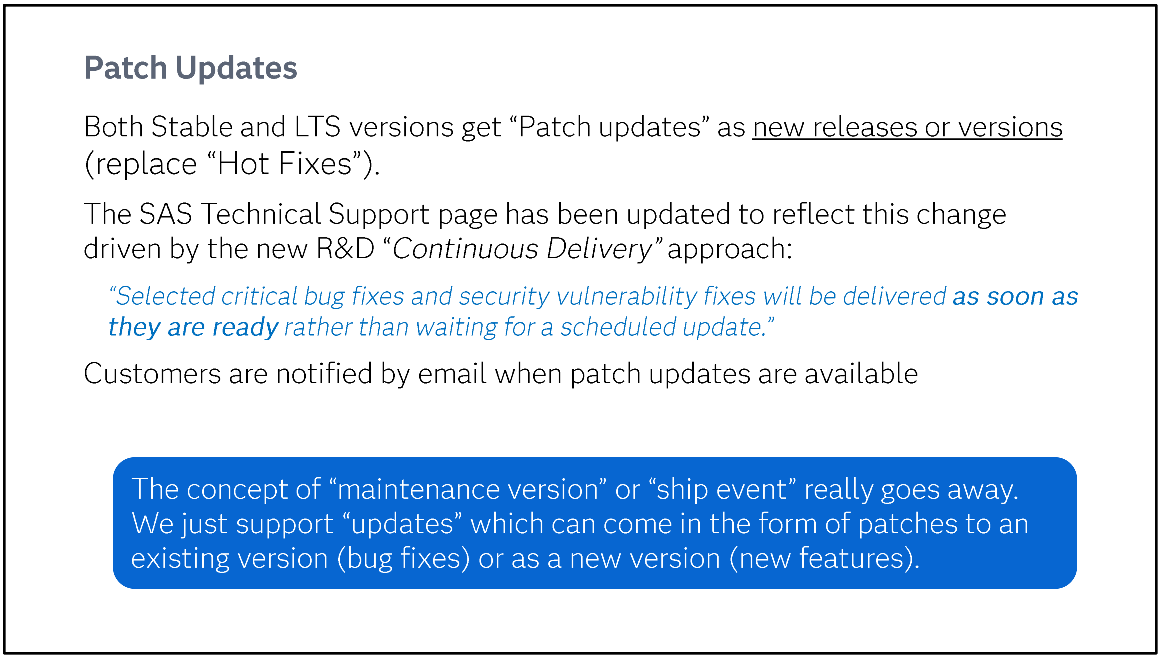
1. Includes access to the product documentation and self-help resources.
2. Patches are not available for Viya 4 releases in Limited Support.
3. Software in Limited Support does not qualify for Severity Level 1 or Severity Level 2 critical support requests.

What does it mean when you go to limited support, when your version is too old and now it's under limited support?

With limited support you still have access to the product documentation and self-help resources. If you look today in the documentation, you can see that even for the version under Limited support, the documentation is online and be accessed using the version drop-down menu.
But the problem with limited support is that patches are not available for the Viya 4 release.
Also, you cannot open severity level one or severity level two critical support requests.

Finally, when entering to “limited support” the update path to a support standard version is not guaranteed (as all combinations of these type of update haven’t been tested by SAS)

This is not ideal for many customers, and they prefer to remain in the “standard support” windows.



Patch Updates

Both Stable and LTS versions get "Patch updates" as new releases or versions (replace "Hot Fixes").

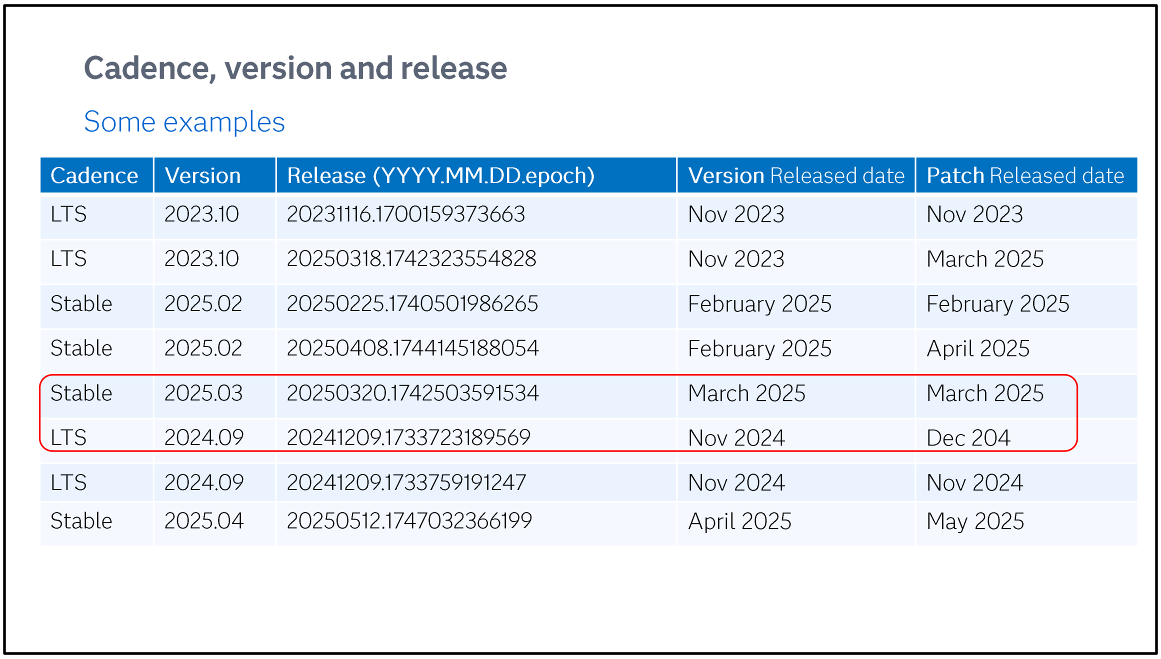
The SAS Technical Support page has been updated to reflect this change driven by the new R&D "*Continuous Delivery*" approach:

> *"Selected critical bug fixes and security vulnerability fixes will be delivered **as soon as they are ready** rather than waiting for a scheduled update."*

Customers are notified by email when patch updates are available

The concept of "maintenance version" or "ship event" really goes away. We just support "updates" which can come in the form of patches to an existing version (bug fixes) or as a new version (new features).

Now let's look at patch updates.
Both stable and LTS cadences can get patch updates within the same version.
So, in the same version, for example 2025.03, you can have many different releases or patch updates.
What this means is that the concept of a hotfix is no longer relevant, because now you have a continuous delivery approach, where the bug fixes can be delivered at anytime within the same monthly version through new patch updates.



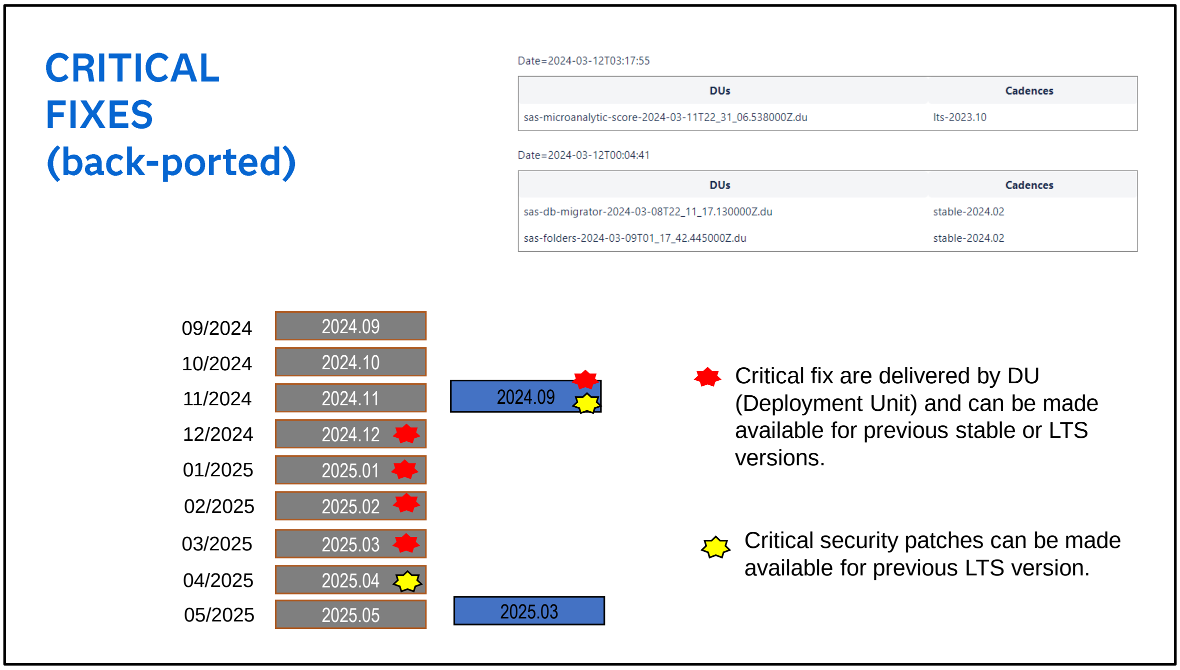
Cadence, version and release

Some examples

Cadence	Version	Release (YYYY.MM.DD.epoch)	Version Released date	Patch Released date
LTS	2023.10	20231116.1700159373663	Nov 2023	Nov 2023
LTS	2023.10	20250318.1742323554828	Nov 2023	March 2025
Stable	2025.02	20250225.1740501986265	February 2025	February 2025
Stable	2025.02	20250408.1744145188054	February 2025	April 2025
Stable	2025.03	20250320.1742503591534	March 2025	March 2025
LTS	2024.09	20241209.1733723189569	Nov 2024	Dec 204
LTS	2024.09	20241209.1733759191247	Nov 2024	Nov 2024
Stable	2025.04	20250512.1747032366199	April 2025	May 2025

It's important to use the right terms and make the distinction between cadence, version, and release.
In this table, you can see examples that illustrate the distinction between cadence, version, and release.
You can see that we can have multiple releases for the same LTS or stable version.
The shipping date: Year, Month, and Day, is included in the release name.
There is a string or number that corresponds to a specific time in the day but is expressed with the EPOCH format. You can convert this format using a Linux command or with an online convertor.
Note that, even the same day, multiple releases can be made available for the same version.

The calendar date is the same, but the EPOCH date will be different.



CRITICAL FIXES (back-ported)

Date=2024-03-12T03:17:55

DUs	Cadences
sas-microanalytic-score-2024-03-11T22_31_06.538000Z.du	lts-2023.10

Date=2024-03-12T00:04:41

DUs	Cadences
sas-db-migrator-2024-03-08T22_11_17.130000Z.du	stable-2024.02
sas-folders-2024-03-09T01_17_42.445000Z.du	stable-2024.02

09/2024	2024.09	
10/2024	2024.10	
11/2024	2024.11	2024.09 ✸ ✩
12/2024	2024.12 ✸	
01/2025	2025.01 ✸	
02/2025	2025.02 ✸	
03/2025	2025.03 ✸	
04/2025	2025.04 ✩	
05/2025	2025.05	2025.03

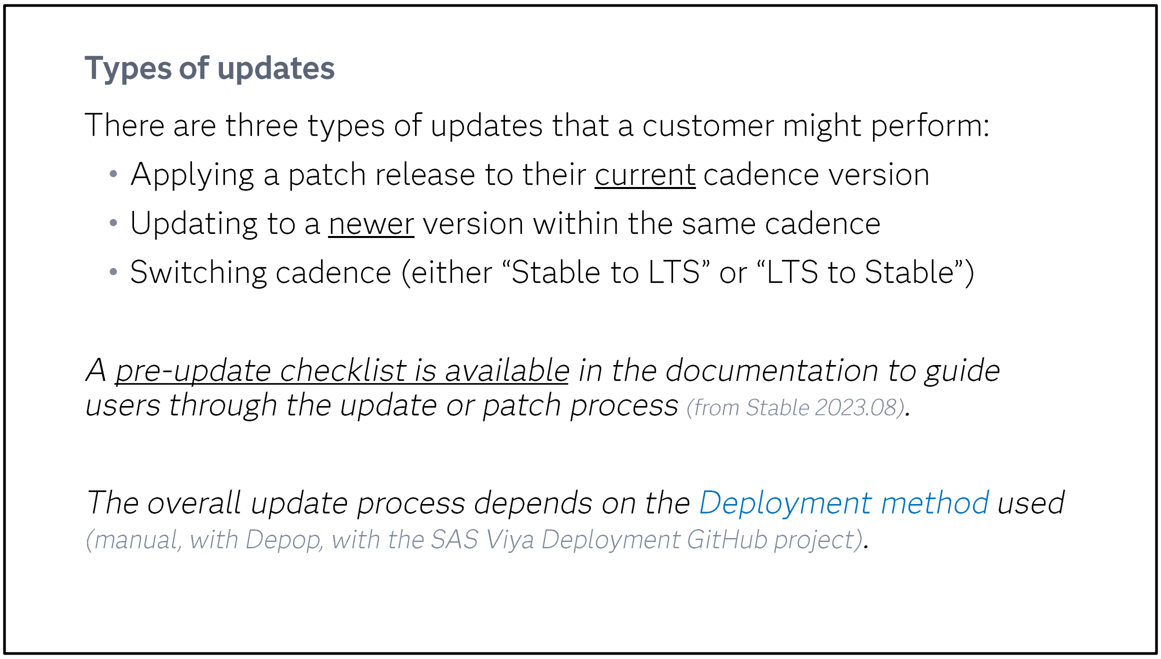
✸ Critical fix are delivered by DU (Deployment Unit) and can be made available for previous stable or LTS versions.

✩ Critical security patches can be made available for previous LTS version.

As discussed before, both Stable and LTS will get patch updates for bug fixes (what some people might call hot fixes).
The patches will impact specific DU (Deployment Unit).
Selected critical bug fixes and security vulnerability fixes will be delivered as soon as they are ready rather than waiting for a scheduled update and they can also be backported to previous stable or LTS versions.

Note that the current SAS Product Security Policy mandates patching critical security vulnerabilities only in the latest LTS and upcoming Stable releases.

Update Considerations



Types of updates

There are three types of updates that a customer might perform:

- Applying a patch release to their current cadence version
- Updating to a newer version within the same cadence
- Switching cadence (either “Stable to LTS” or “LTS to Stable”)

A pre-update checklist is available in the documentation to guide users through the update or patch process (from Stable 2023.08).

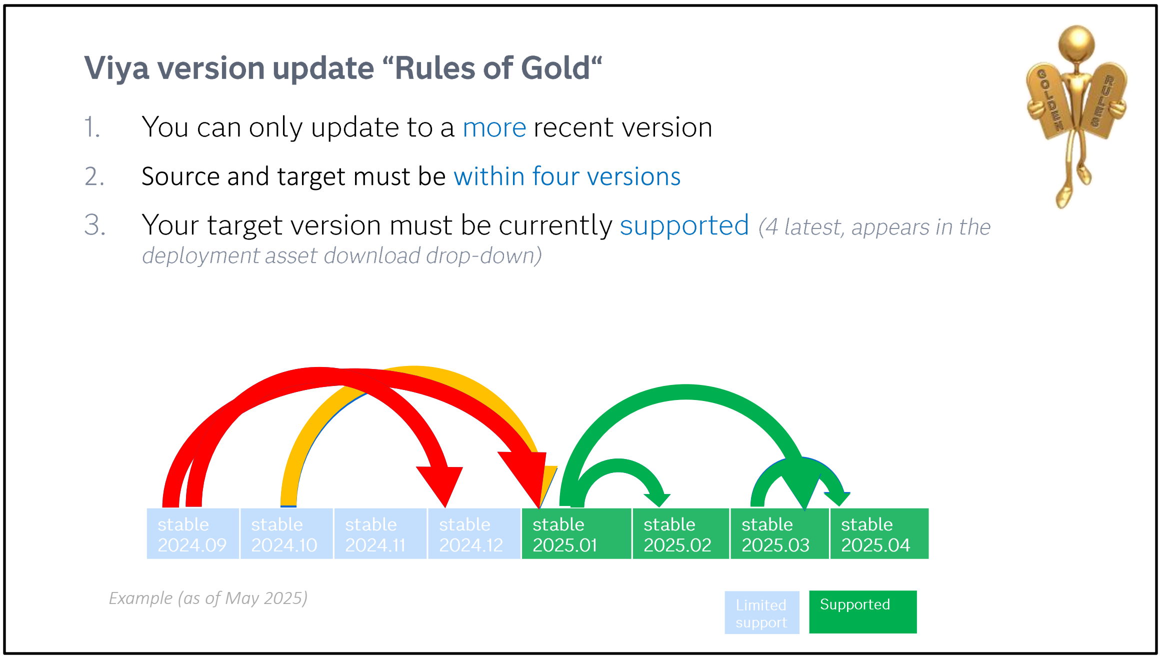
The overall update process depends on the Deployment method used *(manual, with Depop, with the SAS Viya Deployment GitHub project)*.

There are three types of updates that are possible.
The first update type is to apply a patch release in the current cadence version.
The second type of update is to update to a newer version within the same cadence, like from stable 2025.03 to stable 2025.04.
The last type of update is to switch cadence. In this case, you move from the stable cadence to the LTS cadence or from the LTS cadence to the stable cadence.
The official documentation provides a pre-update checklist.
This checklist lists all the things you need to check as part of the update process.
Several questions are asked in the checklist:
Is the current Viya version in Standard Support? Is the version of Kubernetes in the current environment supported for the target version? Multiple CAS servers ? SingleStore ? Etc...)
Based on the answers you will see what are the required actions or check to perform as they depend on the circumstances of the update
it also remind some best practices for update (create backup, run inventory scan, inform SAS Viya users of a possible outage/disruption and so on)

The checklist helps identify the rules, restrictions, and what to do for each type of update.

The overall update process depends on the deployment method.
Depending on whether you have chosen the manual deployment, the deployment operator, or deployment with the SAS deployment GitHub project, you might have different steps to follow for your update process.

However, there are some very important rules that remain the same in terms of version updates.



Viya version update “Rules of Gold“

1. You can only update to a more recent version
2. Source and target must be within four versions
3. Your target version must be currently supported *(4 latest, appears in the deployment asset download drop-down)*

Example (as of May 2025)

Limited support
Supported

Here are the three rules.
Rule number one: you can only update to a more recent version. For example, you can only move from 2025.02 to 2025.03 and then to 2025.04, and so on. You cannot move from .04 to .03, for example.
It seems obvious, but you will see that in some cases it's not so obvious.
Rule number two: The source and target version must be within the four-versions window. If it's more than four versions, then it's not supported.
And Rule number 3: Your target version must be currently supported. The target version must be in the four latest versions. Otherwise, it's also not possible or not supported. Well, it's possible, but it's not supported.

Let's look at an example of that,

First case we move to the latest version from the previous one.

So, something like that

As you can imagine, it is perfectly okay and fully supported.
Let’s take another case

We update but not to the latest version.

This move is okay too.
Now something a bit different

We jump from version N to version N+2

And this one is fine as well, because we are still in the supported four-versions windows

Now what about going from an older unsupported version to a supported version like that

Is it a supported update path ?
Well…It's kind of in the middle. Basically, you can try this kind of update.
And this scenario has actually been tested by the customer lifecycle testing team at SAS.

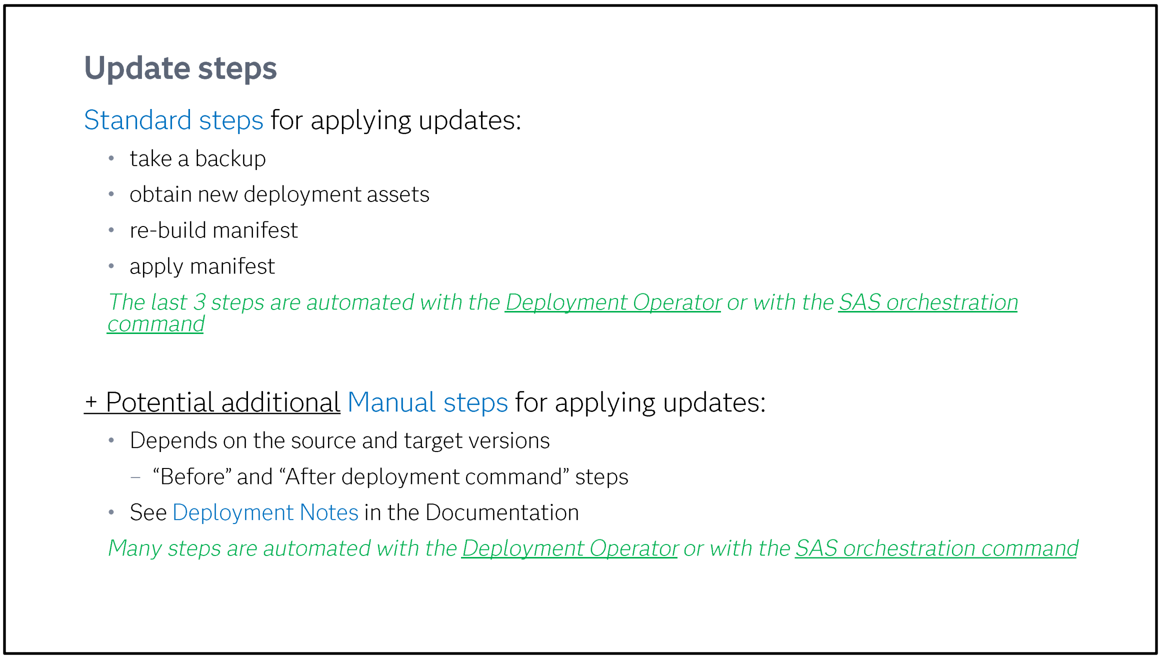
But…. it was tested when the target release was made available. So it was tested in January 2025 in this case, and it was working at that time.
But since then, there have been a lot of intermediate releases (or patch updates) that have been published.
So, it might not work for some reason. It will work in about 90% of the cases, but the remaining 10% of cases could be problematic.
The doc currently explains that while it is possible, in case of problem with this update path, it will not be covered by the SAS technical support, instead the customer need to require assistance from SAS Consulting.

If we look at this other scenario,

moving from an even older version to a supported version…this one is not a supported update path because you are breaking rule number two here : source and target have more than 4 versions of difference.

And this one, where we move from an unsupported version to another unsupported version in the four-versions range…

is also not supported because you are breaking rule number three.



Update steps

Standard steps for applying updates:

- take a backup
- obtain new deployment assets
- re-build manifest
- apply manifest

The last 3 steps are automated with the Deployment Operator or with the SAS orchestration command

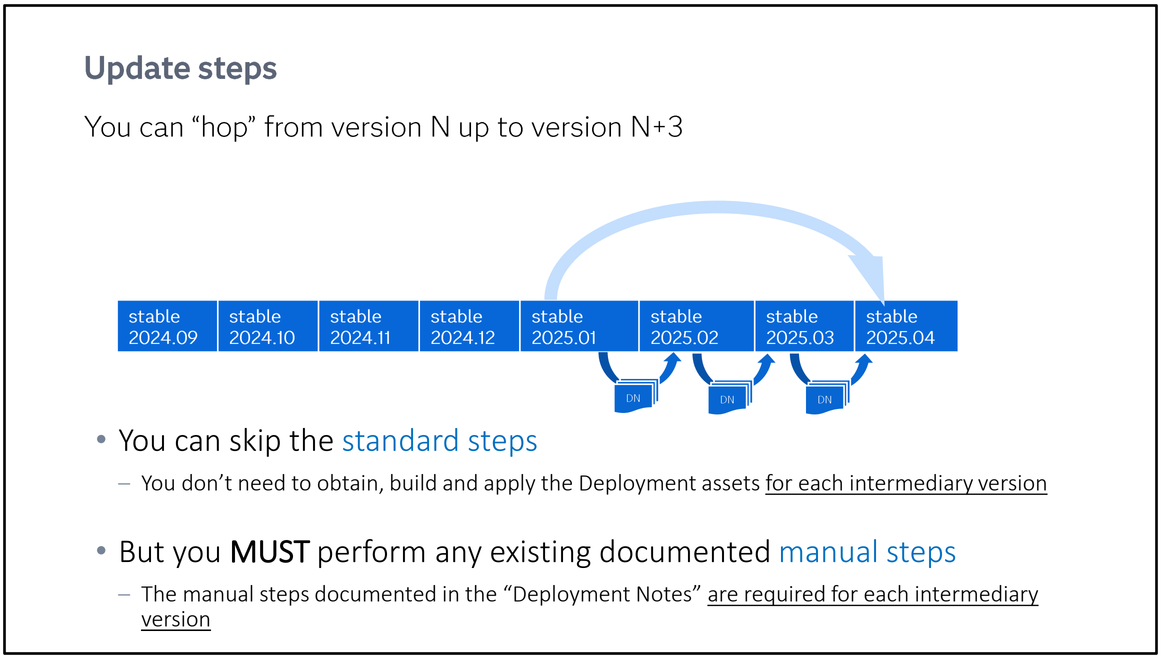
+ Potential additional Manual steps for applying updates:

- Depends on the source and target versions
  - "Before" and "After deployment command" steps
- See Deployment Notes in the Documentation

Many steps are automated with the Deployment Operator or with the SAS orchestration command

Now let's see what it takes to update the software.
You have two types of steps:
First you have standard, recurring steps for applying an update.
take a backup,
obtain new deployment assets,
re-build manifest,
and apply manifest.
They can easily be automated because they are always the same. The last three steps are also automated with the deployment operator or with the SAS orchestration command.

The second type of steps is manual steps.
These depend on the source and target versions.
If you go into the documentation, you see the deployment notes for each version. The notes list the specific commands or steps to follow before or after your deployment.
Sometimes there are only a few of them, so there is almost nothing to do. But sometimes there can be a lot of things to do.
It really depends on what has changed in the new version.
Also, when there are a lot of things to do, you can avoid several steps if you use the deployment operator or the SAS orchestration command.



Update steps

You can “hop” from version N up to version N+3

stable 2024.09 | stable 2024.10 | stable 2024.11 | stable 2024.12 | stable 2025.01 | stable 2025.02 | stable 2025.03 | stable 2025.04

- You can skip the standard steps
  - You don’t need to obtain, build and apply the Deployment assets for each intermediary version
- But you **MUST** perform any existing documented manual steps
  - The manual steps documented in the “Deployment Notes” are required for each intermediary version

Something important to note is that you can “hop” from version N to up to version “N plus three.”

You can do this kind of thing for example:

When you do this, you don't have to re-download the deployment asset, rebuild, and reapply for each intermediate version.

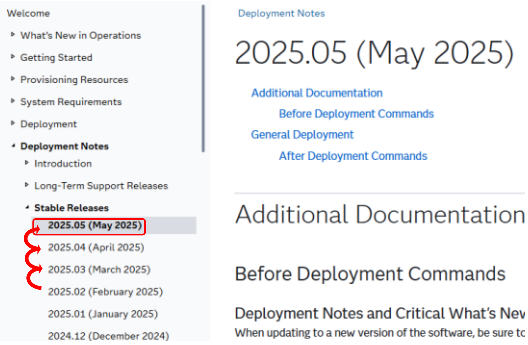
So, for example, you can do that and skip these “standard steps.” You don't need to obtain, build, and apply the deployment asset for each intermediary version.

However, what you need to do for each intermediary version is to proceed with the manual steps that are documented in the deployment notes.
So that's the difference.
You must read the deployment notes and complete them between each version.

Deployment Notes = Manual steps between versions

From time to time, some manual steps may be required to move from one version to another

- A Deployment Note describes a change to the deployment process that was used by an earlier version of the SAS Viya
- All manual steps between versions will need to be applied (for an upgrade in place)
  - For example, moving from Stable 2025.02 to 2025.05
  - ➢ All steps described in the “Deployment Notes” for 2025.03, 2025.04 and 2025.05 need to be applied
- “Before deployment” and “After deployment” commands
- Applies to both Stable and LTS versions
- SAS attempts to minimize the number of manual steps required



Let’s zoom in a little bit on the deployment notes. You can see the deployment notes in the official documentation. You must read them and complete them between each version.
Again, the manual steps are not always required. It depends on the target version and source version.

A deployment note describes a change to the deployment process that was used by an earlier version of SAS Viya.

All the manual steps between versions need to be applied. For example, if you go from 2025.02 to 2025.05, you have to read and apply all the deployment notes for 2025.03, 2025.04, and 2025.05.

You have to look at the "before deployment" and the "after deployment" comments.

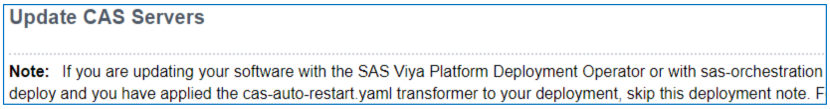
These rules apply to both stable and LTS versions, and SAS attempts to minimize the number of manual steps that are required.

Update automation with orchestrated deployment

For environments deployed with the SAS Viya Deployment Operator or the sas-orchestration command methods, some operations described in the "Deployment Notes" are automatically performed during the deployment

So, you can skip them during the update process.

- Such as :
  - Update CAS Servers (if cas-auto-restart.yaml transformer was applied)
  - Delete openshift routes to SASDrive (2024.12)
  - Scale Down Consul Before Update

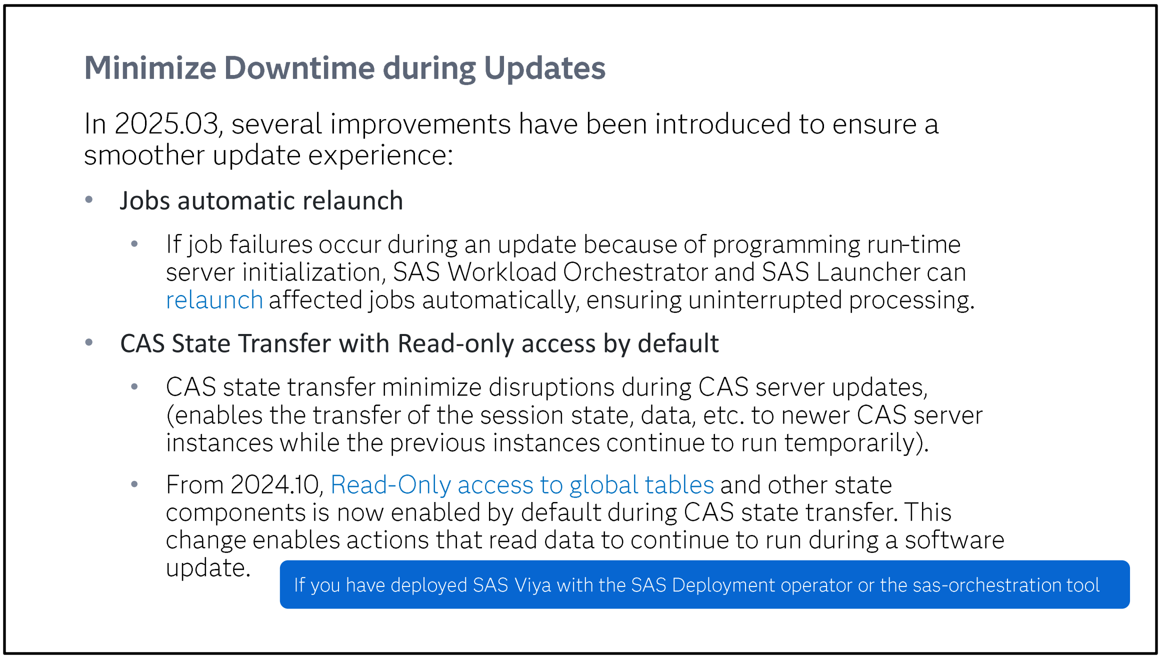


Update CAS Servers

Note: If you are updating your software with the SAS Viya Platform Deployment Operator or with sas-orchestration deploy and you have applied the cas-auto-restart.yaml transformer to your deployment, skip this deployment note. F

If you have deployed your environment with the SAS Deployment Operator or the SAS orchestration commands, some of the manual steps documented in the deployment notes are not required.
They will be performed automatically, behind the scenes, when you redeploy using the deployment operator or run the SAS orchestration commands.
Both the SAS Viya Deployment Operator and the "sas-orchestration command" methods are using the sas-orchestration container which can take care of the operations required during an update.

Here are some examples of tasks that are "not" required to be done when you choose those deployment methods.



Minimize Downtime during Updates

In 2025.03, several improvements have been introduced to ensure a smoother update experience:

- Jobs automatic relaunch
  - If job failures occur during an update because of programming run-time server initialization, SAS Workload Orchestrator and SAS Launcher can relaunch affected jobs automatically, ensuring uninterrupted processing.
- CAS State Transfer with Read-only access by default
  - CAS state transfer minimize disruptions during CAS server updates, (enables the transfer of the session state, data, etc. to newer CAS server instances while the previous instances continue to run temporarily).
  - From 2024.10, Read-Only access to global tables and other state components is now enabled by default during CAS state transfer. This change enables actions that read data to continue to run during a software update.

If you have deployed SAS Viya with the SAS Deployment operator or the sas-orchestration tool

SAS continues to enhance rolling updates for the SAS Viya platform to minimize downtime and improve system resilience.

SAS Workload Orchestrator and SAS Launcher can relaunch the affected jobs automatically, ensuring uninterrupted processing and improving the Stability for the Compute Workloads.

On the CAS side, if you have deployed SAS Viya with the SAS Deployment operator or the sas-orchestration tool, then you can enable the CAS State transfer feature.
During the CAS server update it enables the transfer of the session state, data, etc. to newer CAS server instances while the previous instances continue to run temporarily.
From version 2024.10, Read-Only access to global tables and other state components is enabled by default during CAS state transfer. It allows various CAS clients (such as Visual analytics) to continue to run actions to read data during a software update operation.
It provides a better user experience by minimizing disruptions.

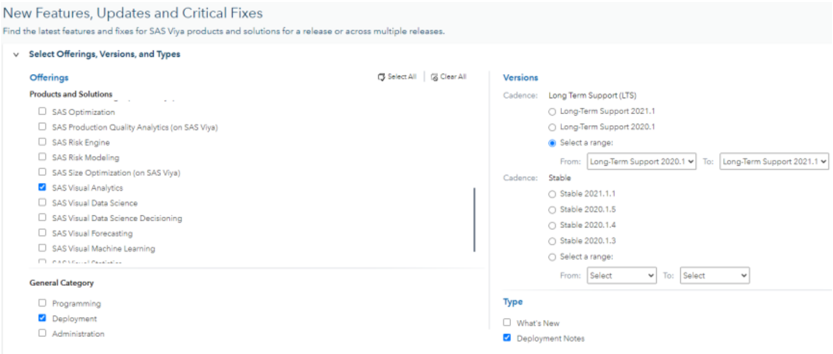
You can learn more about CAS State Transfer in the SAS Administration courses on learn.sas.com.

For more information about minimizing downtime during software updates, see the new section of the SAS Viya platform deployment documentation titled Deployment Considerations for Future Software Updates in the ”Getting Started with SAS Viya Platform Operations” section.

Software updates

Manual steps between versions

Online form to list the required deployment notes for a given source and target versions: `http://release-notes.sas.com/`



New Features, Updates and Critical Fixes

Find the latest features and fixes for SAS Viya products and solutions for a release or across multiple releases.

Select Offerings, Versions, and Types

Offerings

Products and Solutions

- ☐ SAS Optimization
- ☐ SAS Production Quality Analytics (on SAS Viya)
- ☐ SAS Risk Engine
- ☐ SAS Risk Modeling
- ☐ SAS Size Optimization (on SAS Viya)
- ☑ SAS Visual Analytics
- ☐ SAS Visual Data Science
- ☐ SAS Visual Data Science Decisioning
- ☐ SAS Visual Forecasting
- ☐ SAS Visual Machine Learning

General Category

- ☐ Programming
- ☑ Deployment
- ☐ Administration

Versions

Cadence: Long Term Support (LTS)

- ○ Long-Term Support 2021.1
- ○ Long-Term Support 2020.1
- ● Select a range:

From: Long-Term Support 2020.1 To: Long-Term Support 2021.1

Cadence: Stable

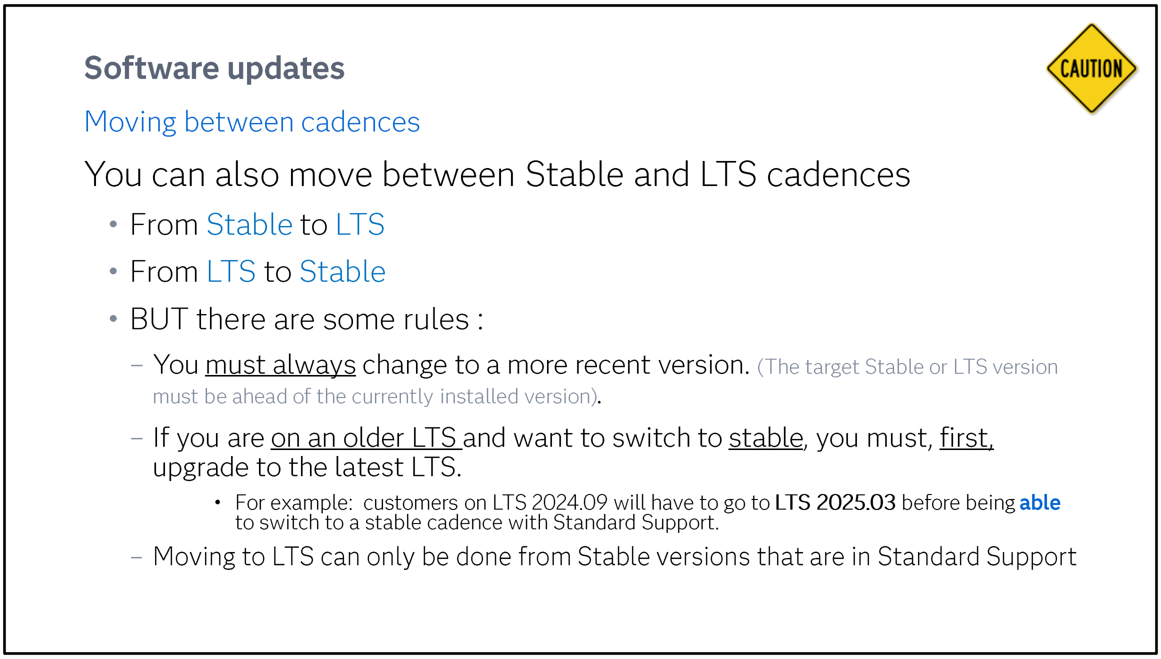
- ○ Stable 2021.1.1
- ○ Stable 2020.1.5
- ○ Stable 2020.1.4
- ○ Stable 2020.1.3
- ○ Select a range:

From: Select To: Select

Type

- ☐ What's New
- ☑ Deployment Notes

A useful page to bookmark is: release-notes.sas.com.
It is a form, that you complete to determine the required deployment notes.
First, you choose which product and solutions you have deployed.
Then you choose the versions – the source version and the target version.
After that, the form automatically generates the deployment notes that you must follow.
It will also give you the “what's new” information, such as "what are the new features", and so on.
It's a way to quickly identify the changes and the manual deployment steps required between a source and target version.



Software updates

Moving between cadences

You can also move between Stable and LTS cadences

- From Stable to LTS
- From LTS to Stable
- BUT there are some rules :
  - You must always change to a more recent version. (The target Stable or LTS version must be ahead of the currently installed version).
  - If you are on an older LTS and want to switch to stable, you must, first, upgrade to the latest LTS.
    - For example: customers on LTS 2024.09 will have to go to **LTS 2025.03** before being **able** to switch to a stable cadence with Standard Support.
  - Moving to LTS can only be done from Stable versions that are in Standard Support

Now, let's look at moving between cadences.

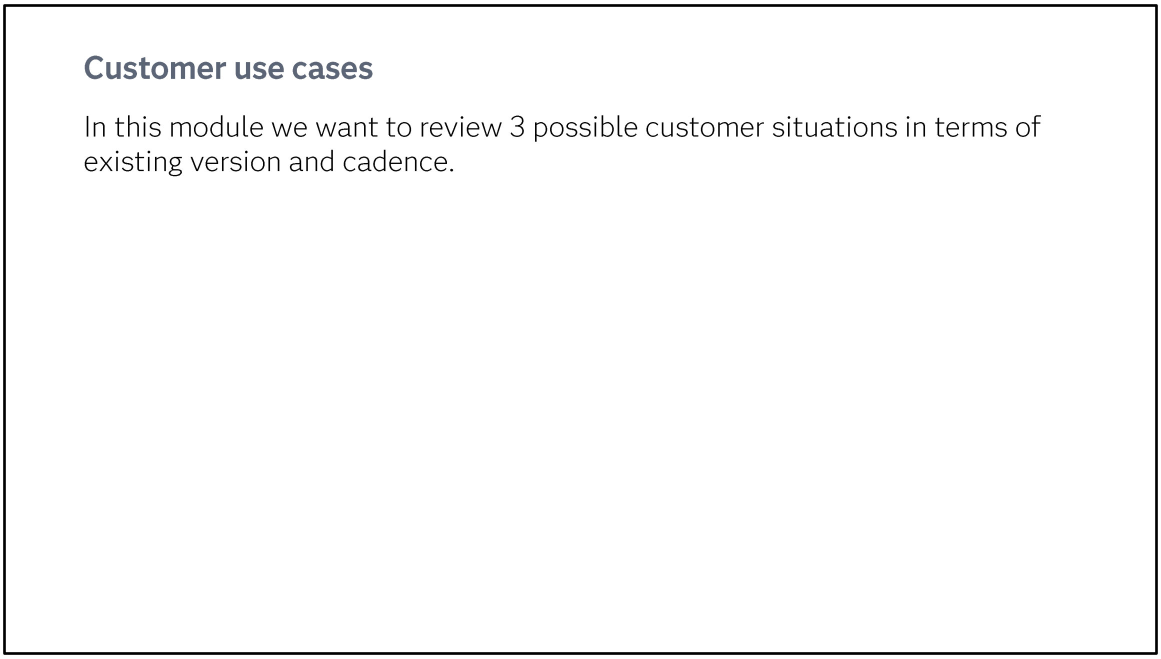
Remember, it's possible to move from stable to LTS cadence or from LTS to stable cadence, but again, there are some rules.

You must always change to a more recent version.
For example, the target stable or LTS version must be ahead of the currently installed version.

Also, if you are on an older LTS version and you want to switch to the stable cadence, you must first upgrade to the latest LTS version.
For example, a customer on LTS 2024.09 will first have to go to LTS 2025.03 before being able to switch to the stable cadence.

The last rule is that moving to LTS can only be done from a stable version that is currently in the standard support.

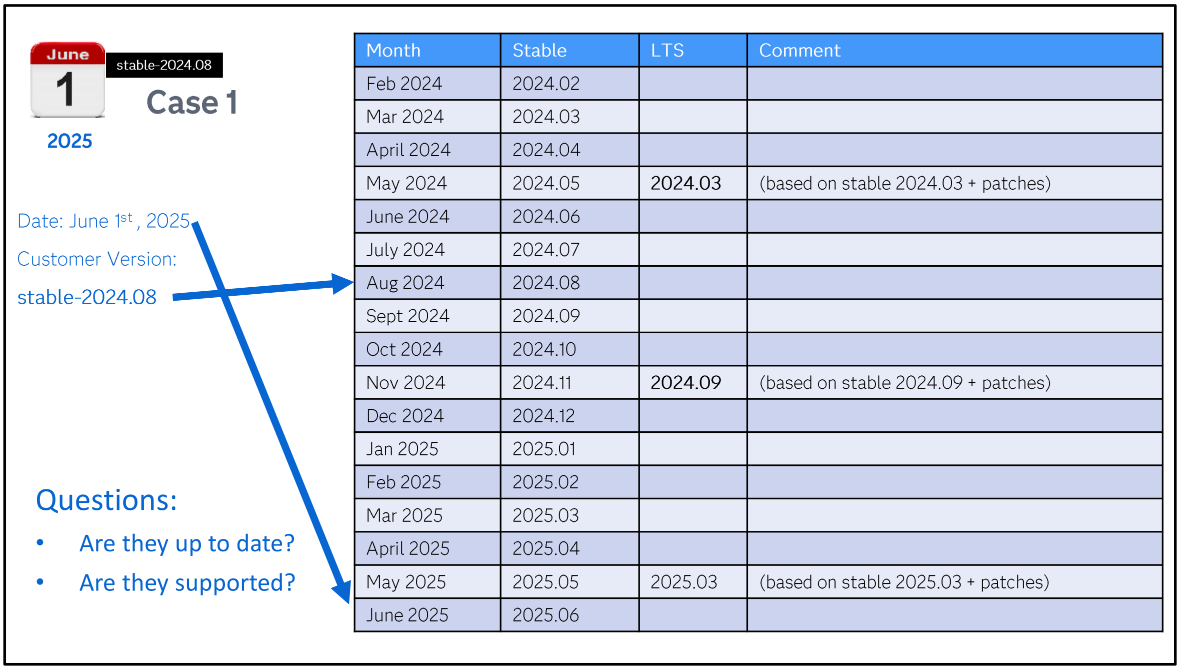
Customers Use Cases



Customer use cases

In this module we want to review 3 possible customer situations in terms of existing version and cadence.

In the other modules you've learned about the basics, timeline, and support considerations.
Then you looked at the update and cadence switch considerations.
In this module, you will put what you've learned into practice with three customer use cases.



June
1
2025
stable-2024.08

Case 1

Date: June 1st , 2025

Customer Version:

stable-2024.08

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		

Questions:

- Are they up to date?
- Are they supported?

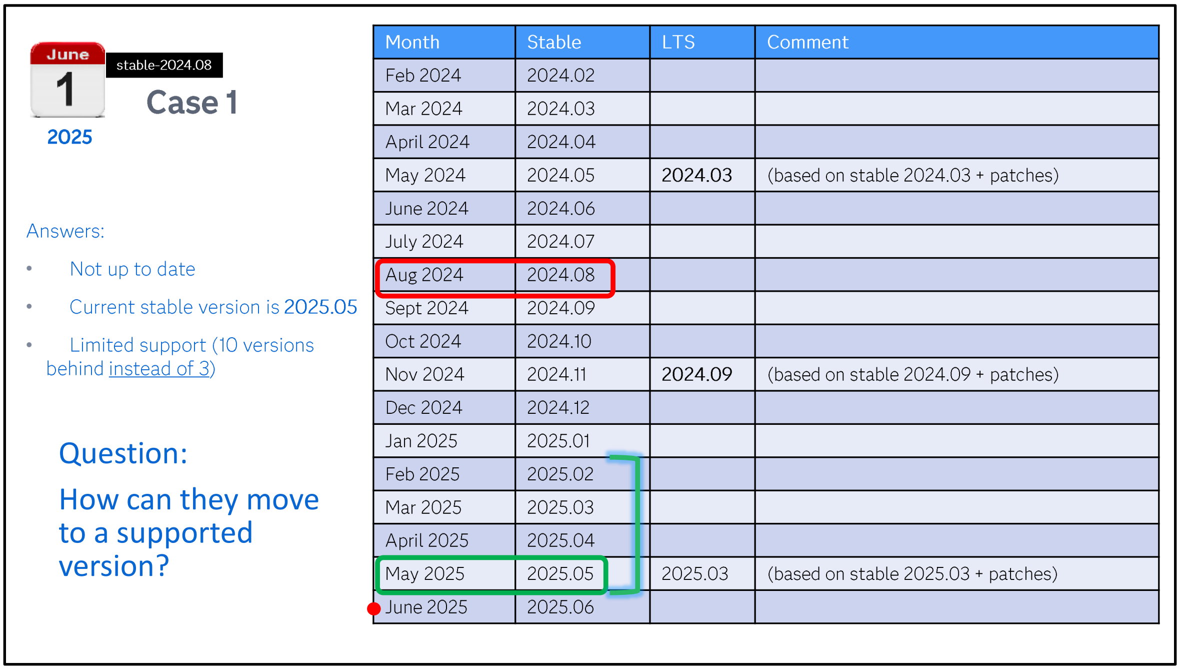
Let's look at three possible customer situations in terms of existing version and cadence.
This is the first case.
Let's imagine we that we are in June 2025,

and the customer is using the stable 2024.08 version.

Here are a few questions:

Are they up to date?
Are they supported?
Let's wait for a few second to let you think about the answers

Let's see what are the answers…



June 1 stable-2024.08

Case 1

2025

Answers:

- Not up to date
- Current stable version is 2025.05
- Limited support (10 versions behind instead of 3)

Question:

How can they move to a supported version?

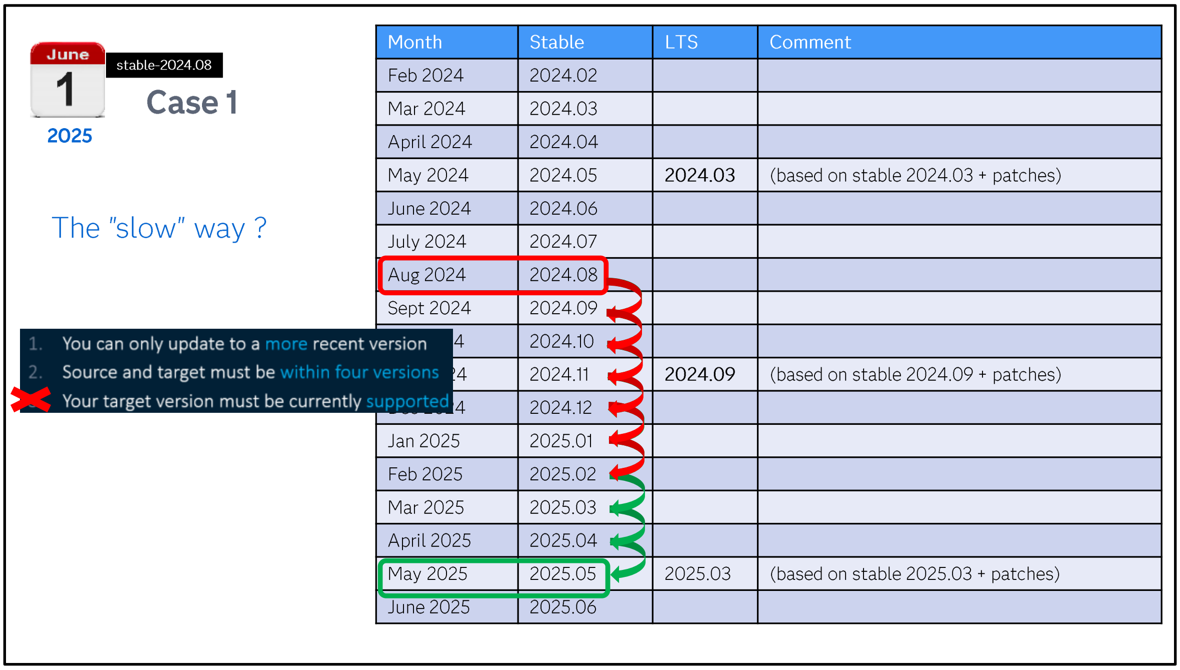
Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
● June 2025	2025.06		

Are they up to date? NO
Are they supported? NO (they are not in the four latest version range)

Here's another question:

How can they move to a supported version?
They can only move forward, and they must at least move to the February version

to be in the range of the four supported versions.
In the next slides we'll explore a few scenario to see how to move to a supported version in this case…



June 1 2025

stable-2024.08

Case 1

The "slow" way ?

1. You can only update to a more recent version
2. Source and target must be within four versions
3. Your target version must be currently supported

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
4	2024.10		
24	2024.11	2024.09	(based on stable 2024.09 + patches)
4	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		

Here is a first scenario.

Technically you could use what is referred to as the “slow way” – moving from one version to the next one.

First, you get the assets for the September version, and you apply them . Then you get the assets for the October version, and you apply them . Then you get the assets for the November version, do the same , and so on.

… … … … … …

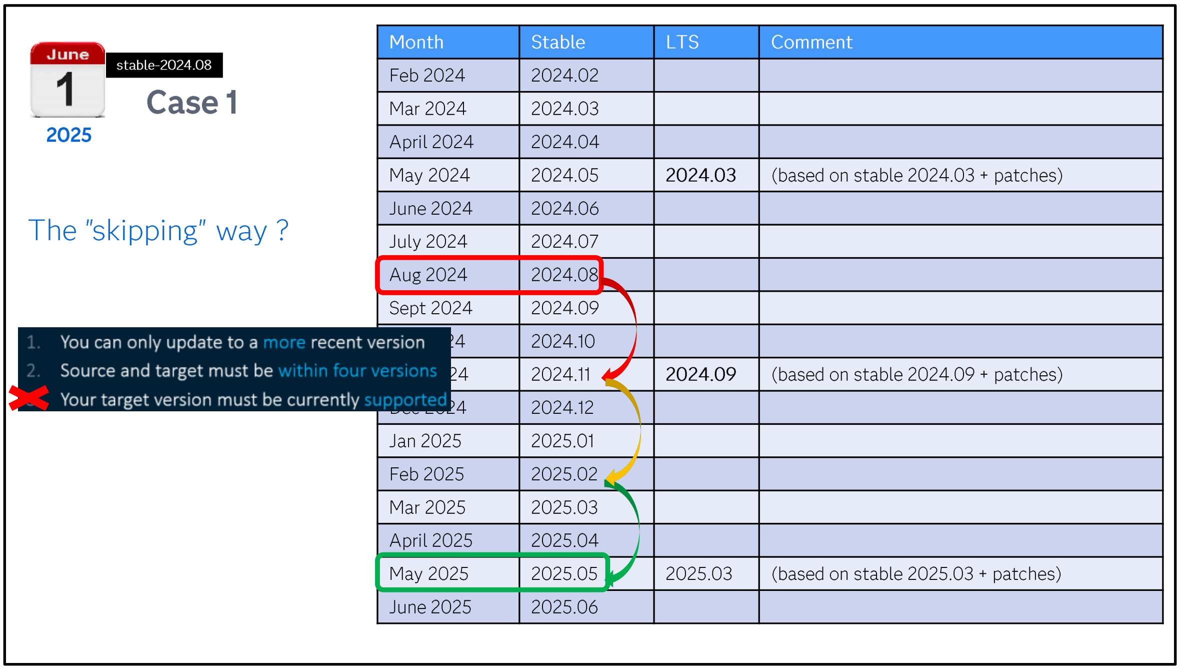
And of course, if there are manual steps between the versions, you need to do the associated tasks for each version update.

If you used this process, it would be very long and time-consuming (you would have to do all the update work you should have done in the last 10 months).

It's also something that is not supported or tested.

And it breaks the rule number 3 which is “Your target must be currently supported”

Finally, to apply this plan, you would need to find a way to get the old deployment assets…Remember that in the portal you can only download the last four versions.



June
1
2025

stable-2024.08

Case 1

The "skipping" way ?

1. You can only update to a more recent version
2. Source and target must be within four versions

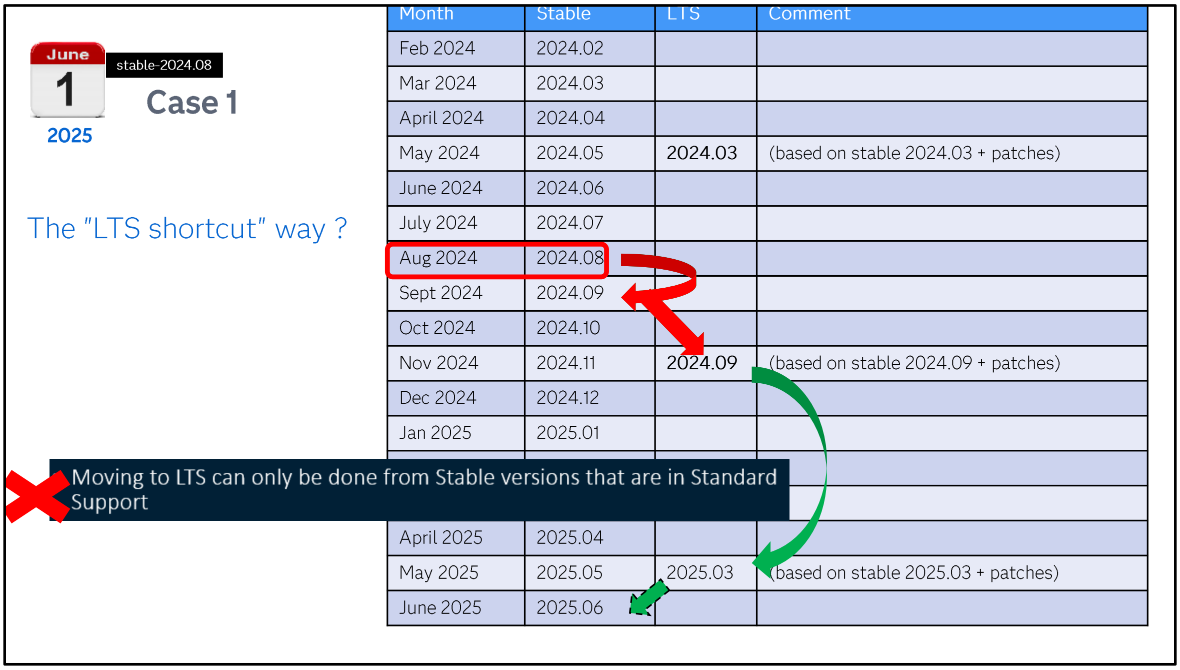
✗ Your target version must be currently supported

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
24	2024.10		
24	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		

The second scenario is the “skipping” way.
Remember, you are allowed to skip versions, but you cannot skip ten versions ahead in one shot, you can skip but within four versions.
For example, here you can directly get and apply the assets for the November version
, BUT if there are any manual steps in the deployment notes between the August and November versions, you need to do them.
Then you get and apply the assets for the February version and perform intermediate manual steps and finally, you do the same for the May version.

…
This would be more efficient than the slow way, but …

it is still against the rules.



June
1
2025

stable-2024.08

Case 1

The "LTS shortcut" way ?

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		

Moving to LTS can only be done from Stable versions that are in Standard Support

This last scenario is also interesting.

The LTS “shortcut” would be a cadence switch.
You know that LTS 2024.09 is the same (in terms of content) as Stable 2024.09, so going to Stable 2024.09 is technically the same as going to LTS 2024.09.

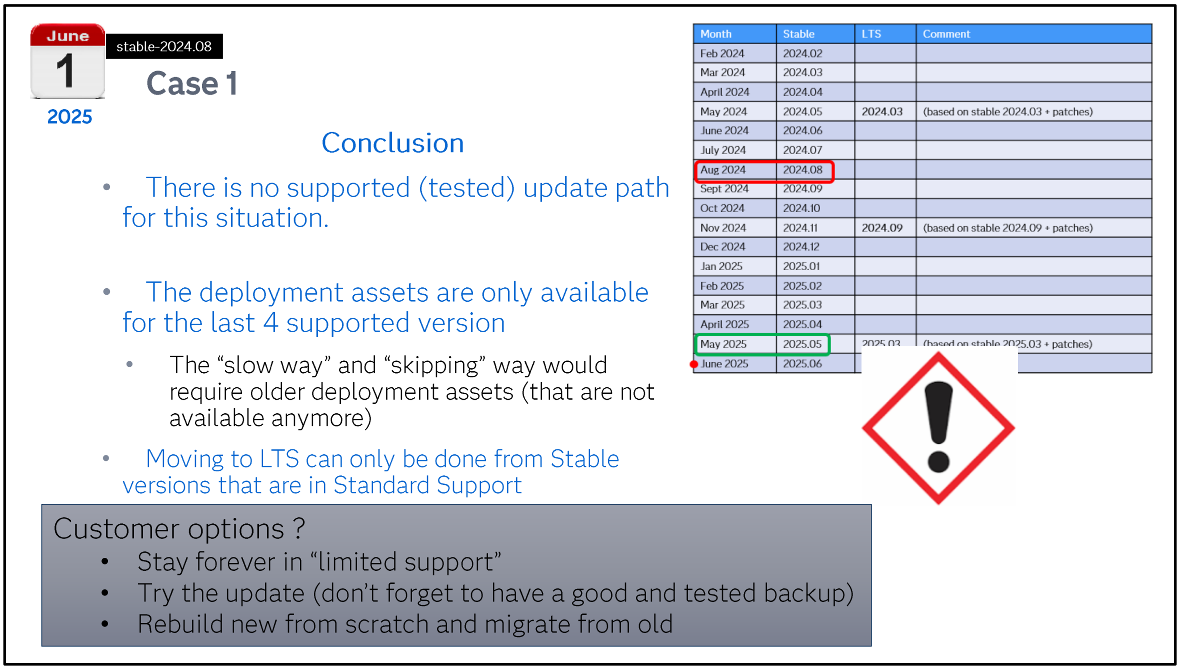
…

…
Now you are on the LTS cadence, so you can go to the next update,

which brings you to the six-months later LTS version: 2025.03.
And if you really wanted to, you could come back onto the Stable cadence when it is released in mid June.

But if this imaginary customer happens to fall ten months behind the LTS cadence, then it probably means that the LTS cadence is more appropriate for them.

However, once again, this scenario is not supported as it breaks one of the rules : “Switching to the LTS cadence can only be done from Stable versions that are in standard support” and it is not the case in this scenario.



June
1
2025

stable-2024.08

Case 1

Conclusion

- There is no supported (tested) update path for this situation.
- The deployment assets are only available for the last 4 supported version
  - The “slow way” and “skipping” way would require older deployment assets (that are not available anymore)
- Moving to LTS can only be done from Stable versions that are in Standard Support

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		

Customer options ?

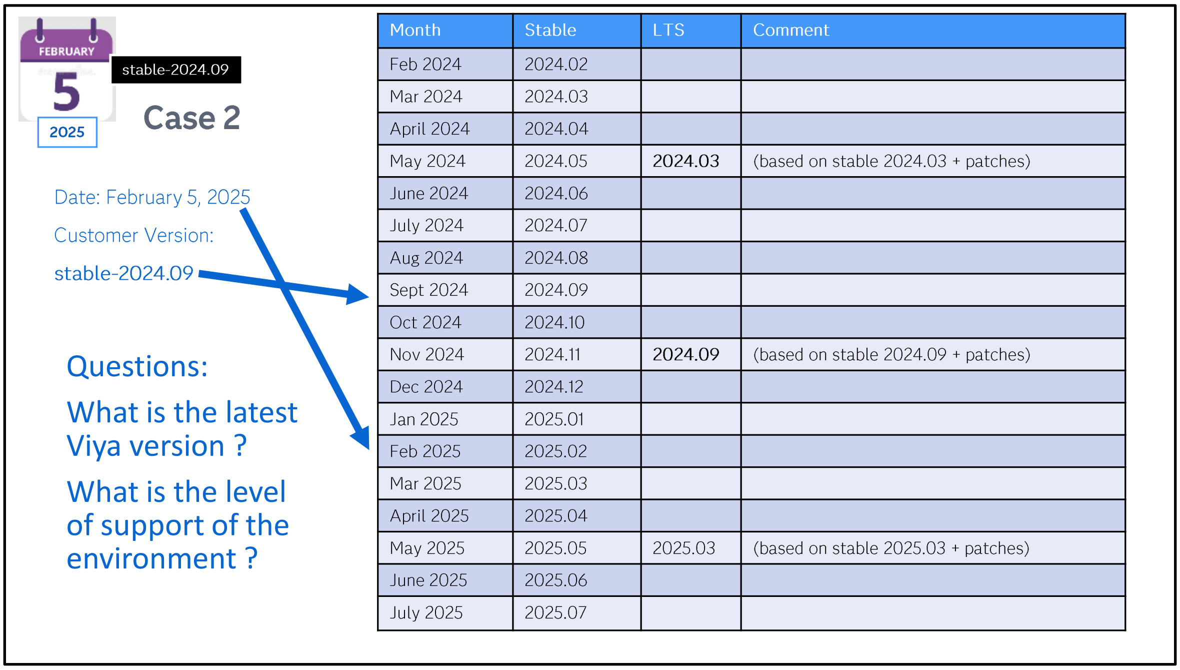
- Stay forever in “limited support”
- Try the update (don’t forget to have a good and tested backup)
- Rebuild new from scratch and migrate from old

What is the conclusion of this use case ?
There is no supported (tested) update path for this situation.
The only update path that is officially tested and fully supported is moving from a supported version to a supported version (so within the four supported versions window).

Second, the deployment assets are only available for the last four supported versions (you won’t see them in the “my.sas.com” portal). If you are running a very old version, it's not possible to download the older deployment assets to move from a limited support version to another limited support version.

And finally, switching from stable to LTS cadence can only be done from Stable versions that are in Standard Support.

The conclusion of this case review is that you should make sure that if the customer chooses the “stable” cadence, they are prepared to update on a monthly basis.



stable-2024.09

Case 2

Date: February 5, 2025

Customer Version:

stable-2024.09

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		
July 2025	2025.07		

Questions:

What is the latest Viya version ?

What is the level of support of the environment ?

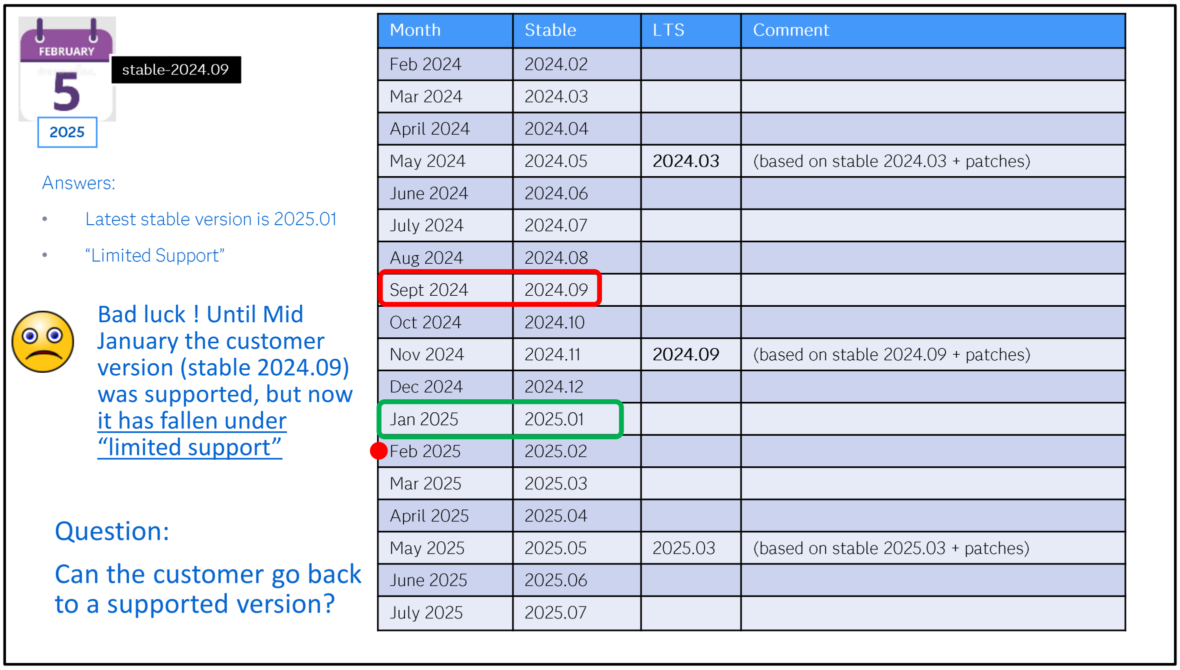
Let’s look at another case.
Imagine we are the 5th of February 2025.

The customer is running stable 2024.09.

Here are some questions:

What is the latest Viya version?
What is the level of support of the environment?
Let’s wait for a few second to let you think about these questions…

Now let’s review the answers.



FEBRUARY
5
2025

stable-2024.09

Answers:

- Latest stable version is 2025.01
- "Limited Support"

Bad luck ! Until Mid January the customer version (stable 2024.09) was supported, but now it has fallen under "limited support"

Question:

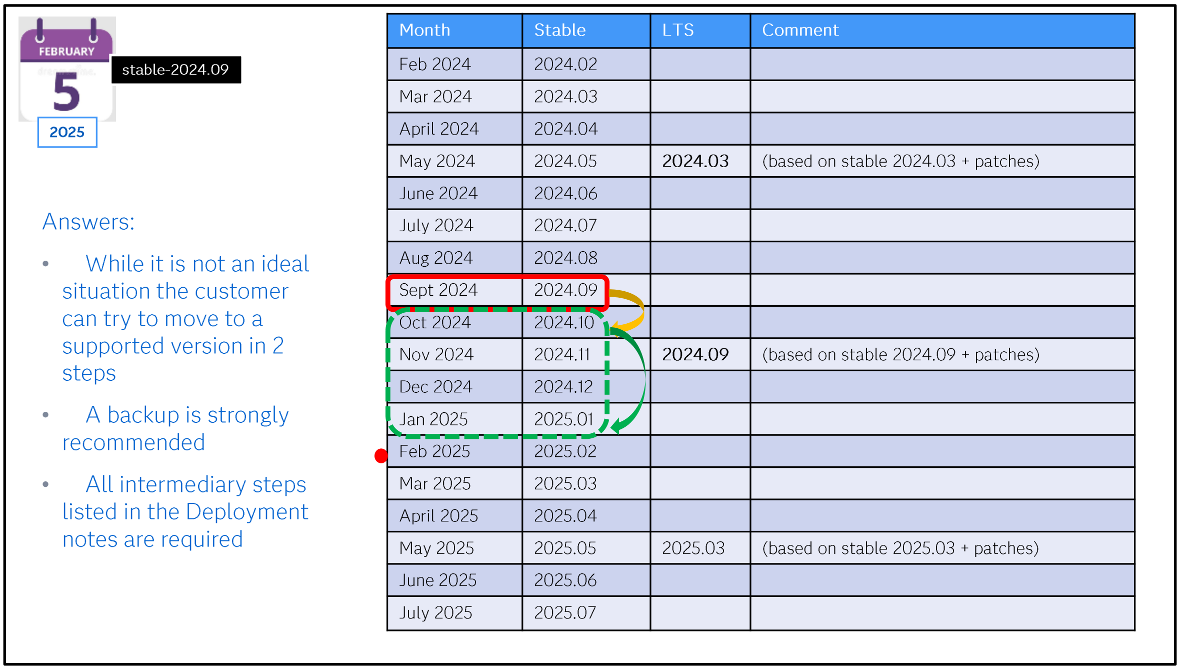
Can the customer go back to a supported version?

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
● Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		
July 2025	2025.07		

The latest stable version of Viya is 2025.01 (it is still the January version although we are in February, because the version of a given month is released during the second half of the month)
The current level of support in this use case is "Limited Support."

It's kind of "bad luck" for the customer because until mid-May, the customer version was supported !
But now the version has fallen just under the limited support model.
So, here's another question:
How can the customer go back to a supported version?

We'll try to answer the question on the next slides.



FEBRUARY
5
2025
stable-2024.09

Answers:

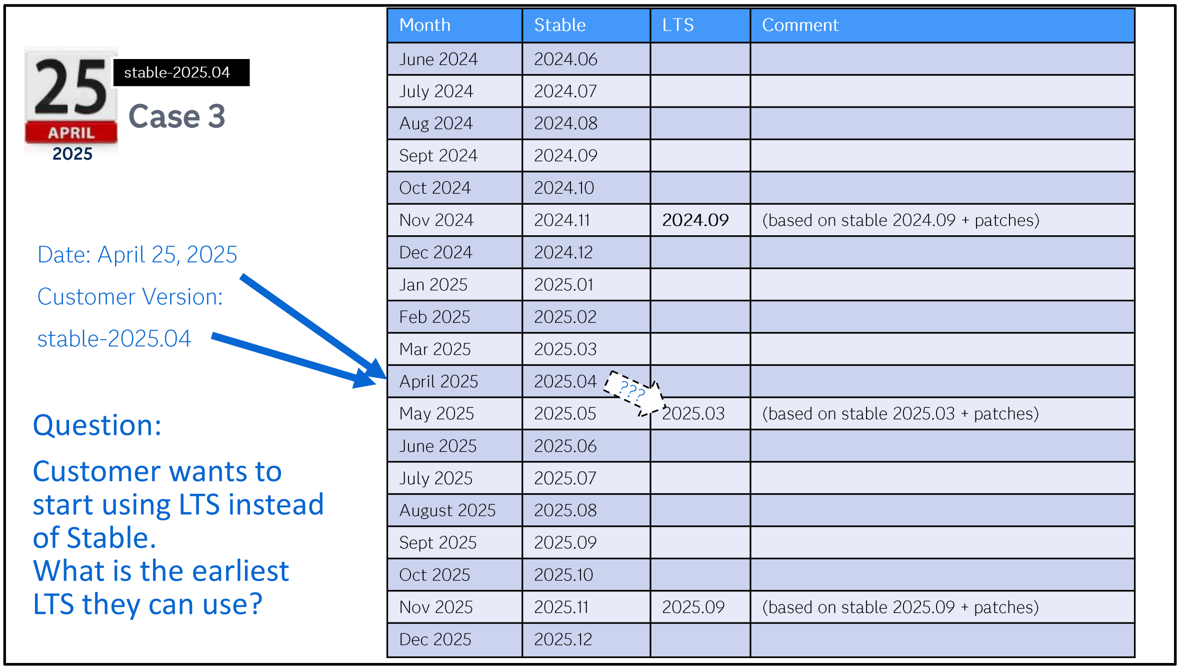
- While it is not an ideal situation the customer can try to move to a supported version in 2 steps
- A backup is strongly recommended
- All intermediary steps listed in the Deployment notes are required

Month	Stable	LTS	Comment
Feb 2024	2024.02		
Mar 2024	2024.03		
April 2024	2024.04		
May 2024	2024.05	2024.03	(based on stable 2024.03 + patches)
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		
July 2025	2025.07		

It's not an ideal situation, but the customer can try to move to a supported version in two steps.
For example, the customer can go from 2024.09 to 2024.10.

This first part of the update path is not fully supported.
But if it is successful, then the customer will be in a supported version

…at least until mid-February
Then from there, in a second step, it is possible to move from 2024.10 to 2025.01 and this second part of the update is supported.
So the update process will happen in two times, along with all the intermediary steps that are in the deployment notes.
There is a little risk on the first part, but it is possible to do this kind of things.



25 APRIL 2025

stable-2025.04

Case 3

Date: April 25, 2025

Customer Version:

stable-2025.04

Question:

Customer wants to start using LTS instead of Stable.
What is the earliest LTS they can use?

Month	Stable	LTS	Comment
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		
July 2025	2025.07		
August 2025	2025.08		
Sept 2025	2025.09		
Oct 2025	2025.10		
Nov 2025	2025.11	2025.09	(based on stable 2025.09 + patches)
Dec 2025	2025.12		

Let’s look at the third, and last, use case.

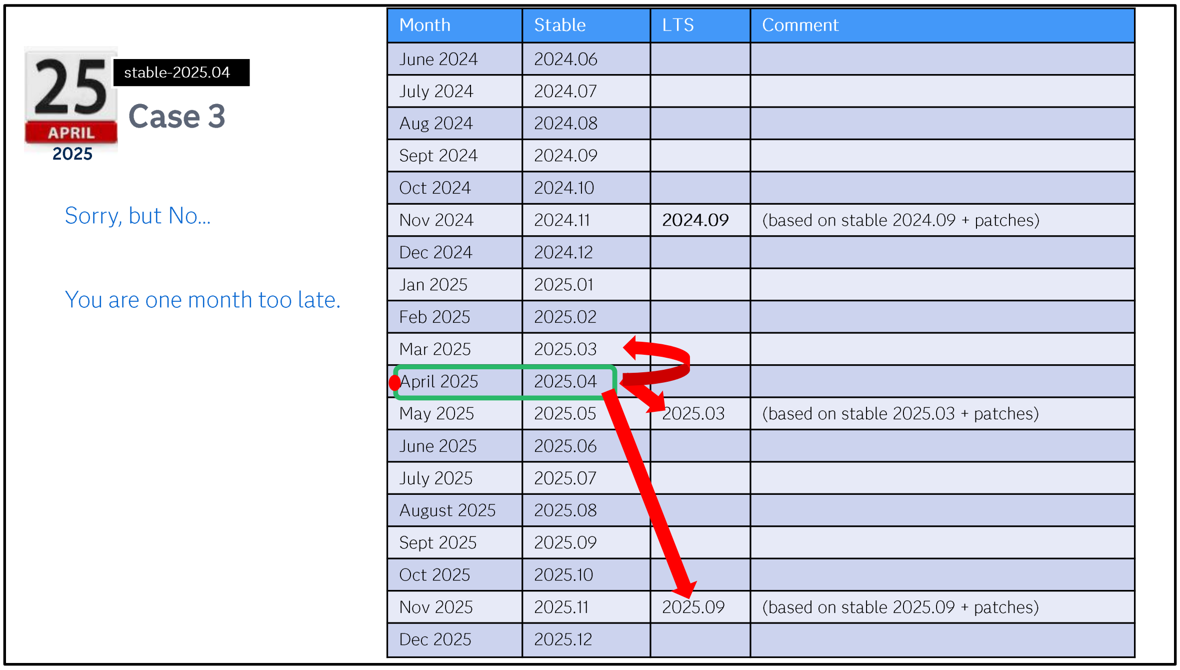
Let’s imagine we are at the end of April 2025,

the customer is up to date and is running the latest available version that was released mid-April.
But now, the customer has decided that, in their case, staying on the stable cadence was not worth performing the update every month.
So now they want to switch to the LTS version.

So here is another question for you : if the customer is at stable 2025.04 and they want to switch to the LTS cadence , can they simply go to LTS 2025.03?

Is that okay? It looks okay in the table, right?

Well…



Case 3

Sorry, but No...

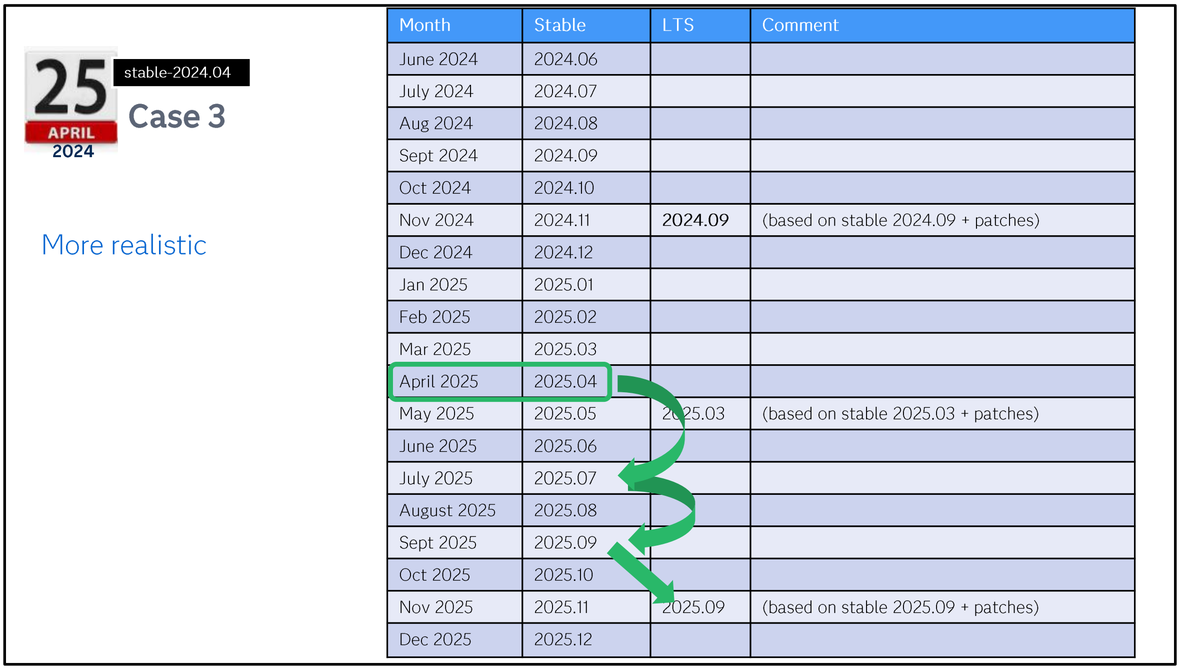
You are one month too late.

Month	Stable	LTS	Comment
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		
July 2025	2025.07		
August 2025	2025.08		
Sept 2025	2025.09		
Oct 2025	2025.10		
Nov 2025	2025.11	2025.09	(based on stable 2025.09 + patches)
Dec 2025	2025.12		

Actually, it is not okay. The customer is one month too late, because they are running stable 2025.04 and the next LTS (LTS 2025.03) is based on Stable 2025.03.

Since “LTS 2025.03” has the same code and content as “Stable 2025.03” moving to LTS 2025.03 from stable 2025.04 would be like going back to an older version.
…
The customer missed their window of opportunity to move to the LTS version.
The customer might say, okay fine. I’ll stay on this version and when the new LTS is available, I’ll move directly to it.

…
Notice that the arrow is red again. This is also not allowed because it is a jump 8 months into the future.



25 APRIL 2024

stable-2024.04

Case 3

More realistic

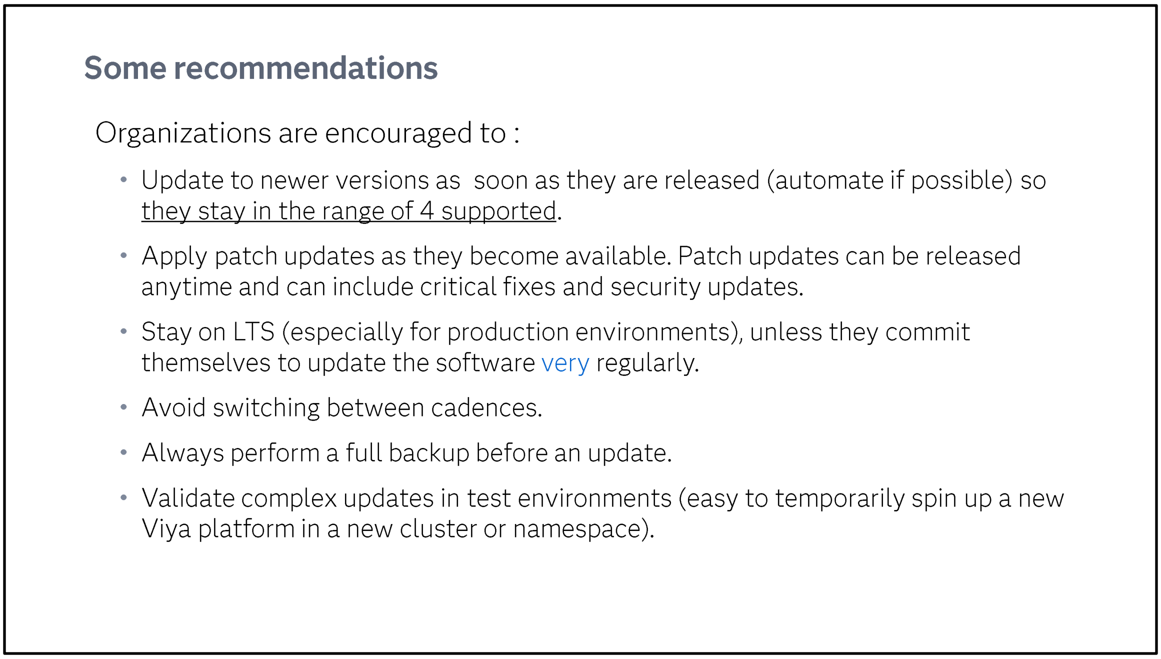
Month	Stable	LTS	Comment
June 2024	2024.06		
July 2024	2024.07		
Aug 2024	2024.08		
Sept 2024	2024.09		
Oct 2024	2024.10		
Nov 2024	2024.11	2024.09	(based on stable 2024.09 + patches)
Dec 2024	2024.12		
Jan 2025	2025.01		
Feb 2025	2025.02		
Mar 2025	2025.03		
April 2025	2025.04		
May 2025	2025.05	2025.03	(based on stable 2025.03 + patches)
June 2025	2025.06		
July 2025	2025.07		
August 2025	2025.08		
Sept 2025	2025.09		
Oct 2025	2025.10		
Nov 2025	2025.11	2025.09	(based on stable 2025.09 + patches)
Dec 2025	2025.12		

So, what can the customer do instead?
They stay on Stable.
They complete one update to the July version when it is out (2025.07).

And then they wait for Stable 2025.09 to be shipped and they follow the Stable 2025.09 steps to go to this version.

At this point it is very important that they don’t do another update!

Once LTS is officially released in November, they update to LTS 2025.09 by getting and applying the new deployment assets. There is no need to run the Deployment Notes because it is the same version.



Some recommendations

Organizations are encouraged to :

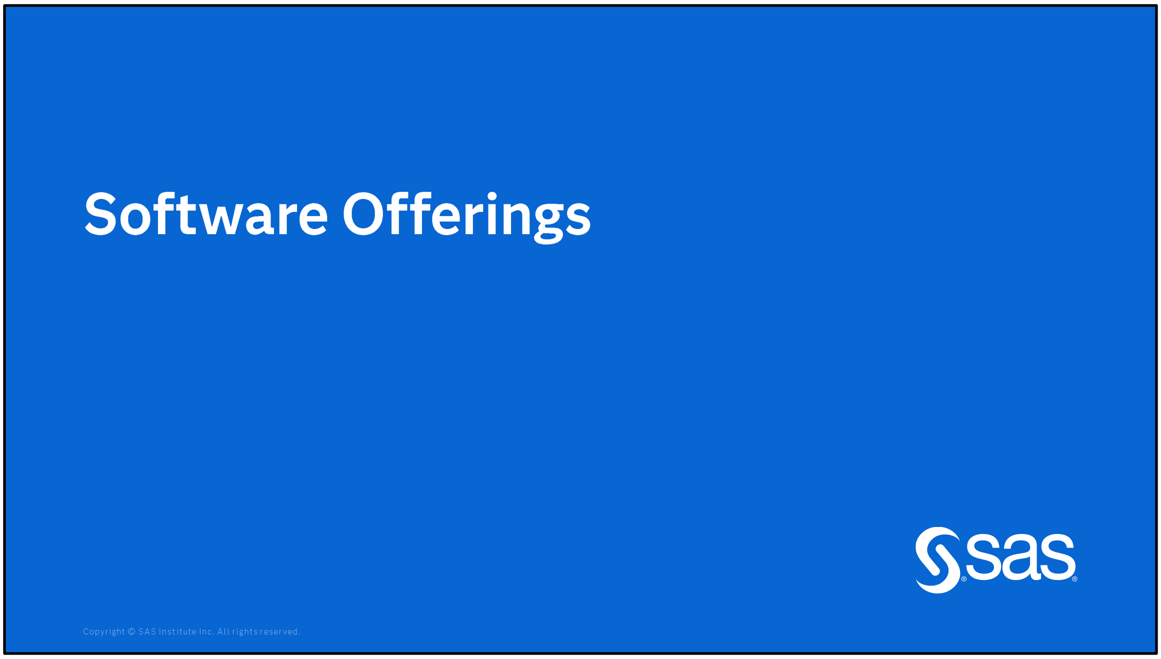
- Update to newer versions as soon as they are released (automate if possible) so they stay in the range of 4 supported.
- Apply patch updates as they become available. Patch updates can be released anytime and can include critical fixes and security updates.
- Stay on LTS (especially for production environments), unless they commit themselves to update the software very regularly.
- Avoid switching between cadences.
- Always perform a full backup before an update.
- Validate complex updates in test environments (easy to temporarily spin up a new Viya platform in a new cluster or namespace).

So that’s the end of the use cases review.

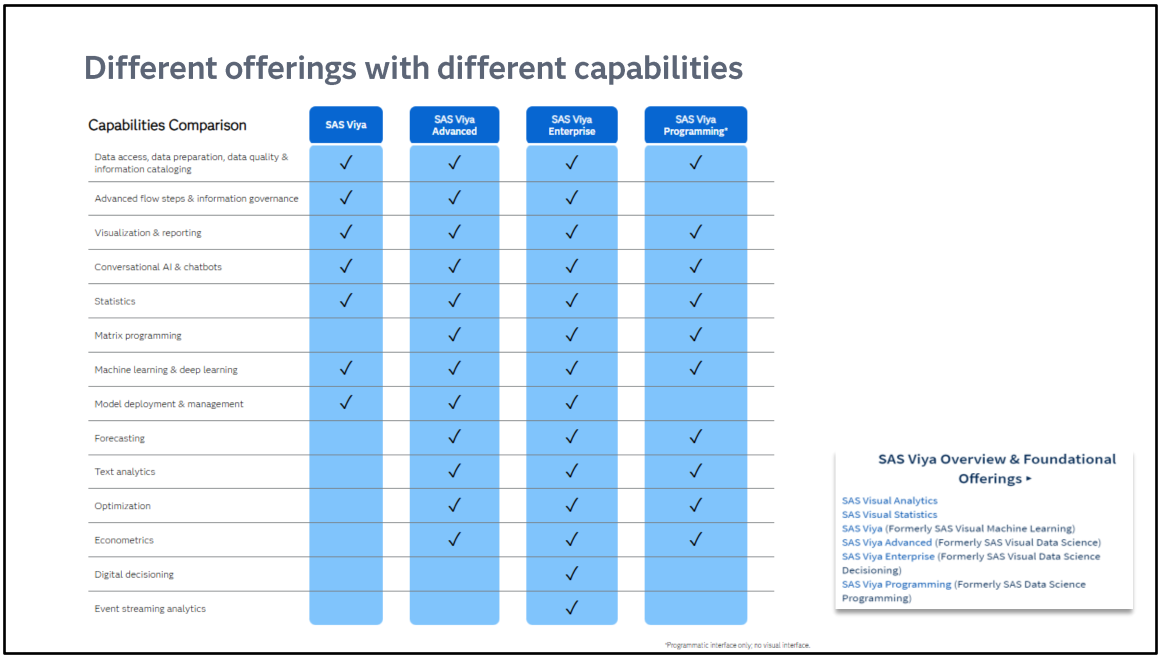
Here are some final recommendations:
- Update as regularly as possible. Remember the range of four. If for some reason you miss an update, you will soon be in limited support. With the stable version, it only takes four months, whereas with LTS there could be a two-year window for support.
Customers are encouraged to implement "update automation pipelines" to meet that goal of staying up to date with their Viya version.
It's also a good idea to automate content export and import operations to address the risks of failure for “update in place” operations and be able to restart from a fresh environment.
- It is also recommended to apply patch updates frequently.
- For production or sensitive environments, consider staying on the LTS cadence because it means less update work every month.
- If you can, avoid switching because it's not always possible. Or it's only possible at certain dates – which makes things even more complicated.
- Each time you update, make sure you have a good backup of all the user’s content.

Your customer should select a cadence that they can maintain, and they should be very clear about the risks and consequence of not doing that.

2.3 Software Offerings



Software Offerings



Different offerings with different capabilities

Capabilities Comparison	SAS Viya	SAS Viya Advanced	SAS Viya Enterprise	SAS Viya Programming*
Data access, data preparation, data quality & information cataloging	✓	✓	✓	✓
Advanced flow steps & information governance	✓	✓	✓	
Visualization & reporting	✓	✓	✓	✓
Conversational AI & chatbots	✓	✓	✓	✓
Statistics	✓	✓	✓	✓
Matrix programming		✓	✓	✓
Machine learning & deep learning	✓	✓	✓	✓
Model deployment & management	✓	✓	✓	
Forecasting		✓	✓	✓
Text analytics		✓	✓	✓
Optimization		✓	✓	✓
Econometrics		✓	✓	✓
Digital decisioning			✓	
Event streaming analytics			✓	

*Programmatic interface only; no visual interface.

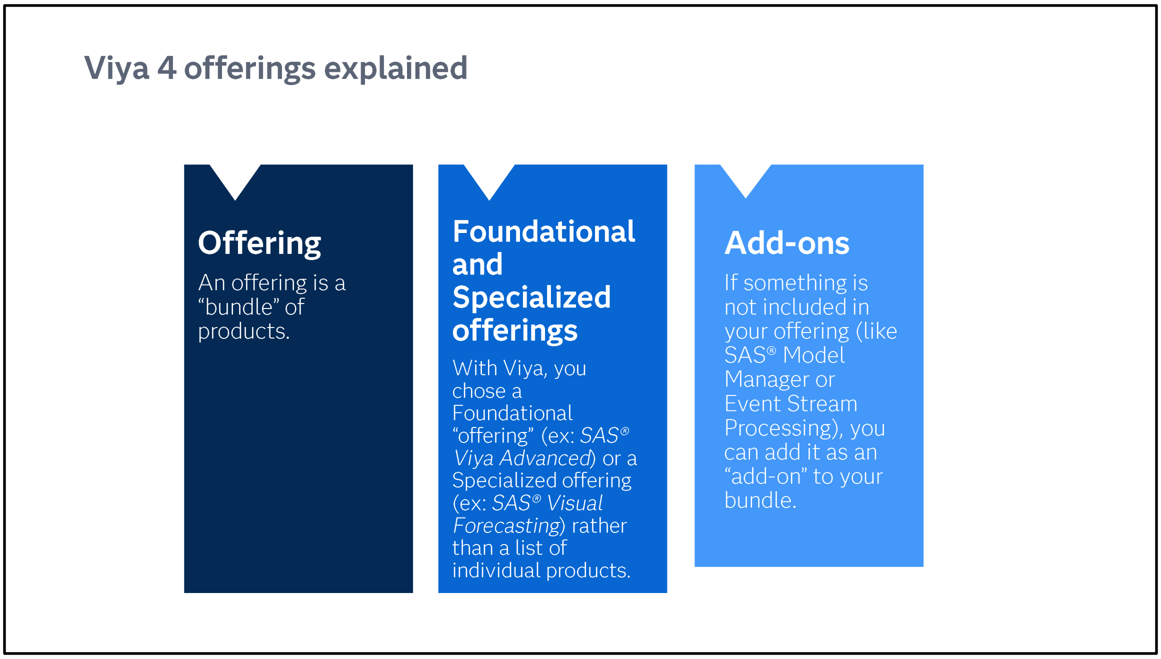
SAS Viya Overview & Foundational Offerings ▸

SAS Visual Analytics
SAS Visual Statistics
SAS Viya (Formerly SAS Visual Machine Learning)
SAS Viya Advanced (Formerly SAS Visual Data Science)
SAS Viya Enterprise (Formerly SAS Visual Data Science Decisioning)
SAS Viya Programming (Formerly SAS Data Science Programming)

In this module, you'll learn about the Viya 4 offerings.
You may ask yourself, "Why am I learning about offerings and products in a Deployment or Architecture module?"
The offerings have an impact on the components that are deployed in the environment, because depending on the offering, you can have things like MAS (for Micro Analytics Services), ESP, OpenSearch, Airflow, or other components.
The existence of those components in your environment will be driven by the offering.
Also, it has an impact on the topology in terms of footprint and sizing, so you don't need the same resources if you are only deploying the "Visual Analytics" offering versus if you are deploying the whole "SAS Viya Enterprise" offering.

In the table here on the screen, you see the capabilities of four of the Foundational offerings: SAS Viya, SAS Viya Advanced, SAS Viya Enterprise,
and also SAS Viya Programming (which is the "programming interface only" kind of environment that you learned about previously).

There are other Foundational offerings, like SAS Visual Analytics and SAS Visual Statistics.
The products portfolio has been optimized to include the six main "Foundational" Offerings that appear on the right side of the screen.



Viya 4 offerings explained

Offering

An offering is a “bundle” of products.

Foundational and Specialized offerings

With Viya, you chose a Foundational “offering” (ex: *SAS® Viya Advanced*) or a Specialized offering (ex: *SAS® Visual Forecasting*) rather than a list of individual products.

Add-ons

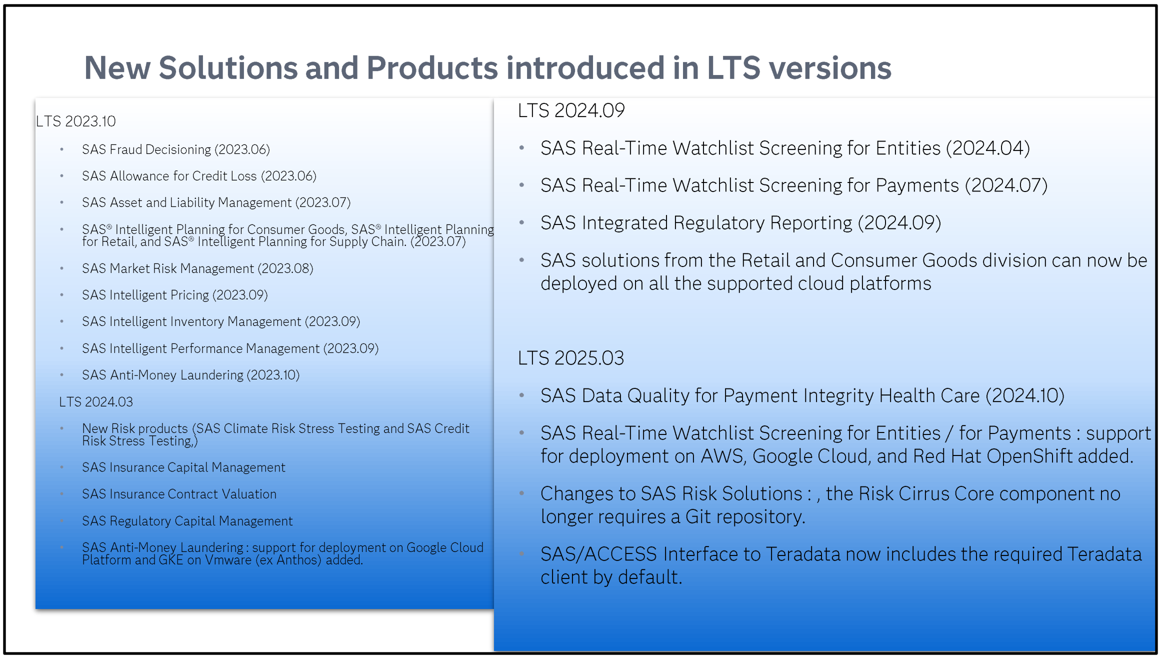
If something is not included in your offering (like SAS® Model Manager or Event Stream Processing), you can add it as an “add-on” to your bundle.

In the last half of 2020, there was a decision to simplify customer orders to only being able to order “offerings” rather than individual products.

* A “bundle” contains several products or components like Visual Analytics, Data preparation, Model Manager, ESP, and so on.

With Viya 4, you choose a product “bundle” or “offering”, instead of a long list of individual products (for example something called “SAS Viya advanced”).

If you want something that is not in your bundle, you can add it as an “add-on.“



New Solutions and Products introduced in LTS versions

LTS 2023.10

- SAS Fraud Decisioning (2023.06)
- SAS Allowance for Credit Loss (2023.06)
- SAS Asset and Liability Management (2023.07)
- SAS® Intelligent Planning for Consumer Goods, SAS® Intelligent Planning for Retail, and SAS® Intelligent Planning for Supply Chain. (2023.07)
- SAS Market Risk Management (2023.08)
- SAS Intelligent Pricing (2023.09)
- SAS Intelligent Inventory Management (2023.09)
- SAS Intelligent Performance Management (2023.09)
- SAS Anti-Money Laundering (2023.10)

LTS 2024.03

- New Risk products (SAS Climate Risk Stress Testing and SAS Credit Risk Stress Testing,)
- SAS Insurance Capital Management
- SAS Insurance Contract Valuation
- SAS Regulatory Capital Management
- SAS Anti-Money Laundering : support for deployment on Google Cloud Platform and GKE on Vmware (ex Anthos) added.

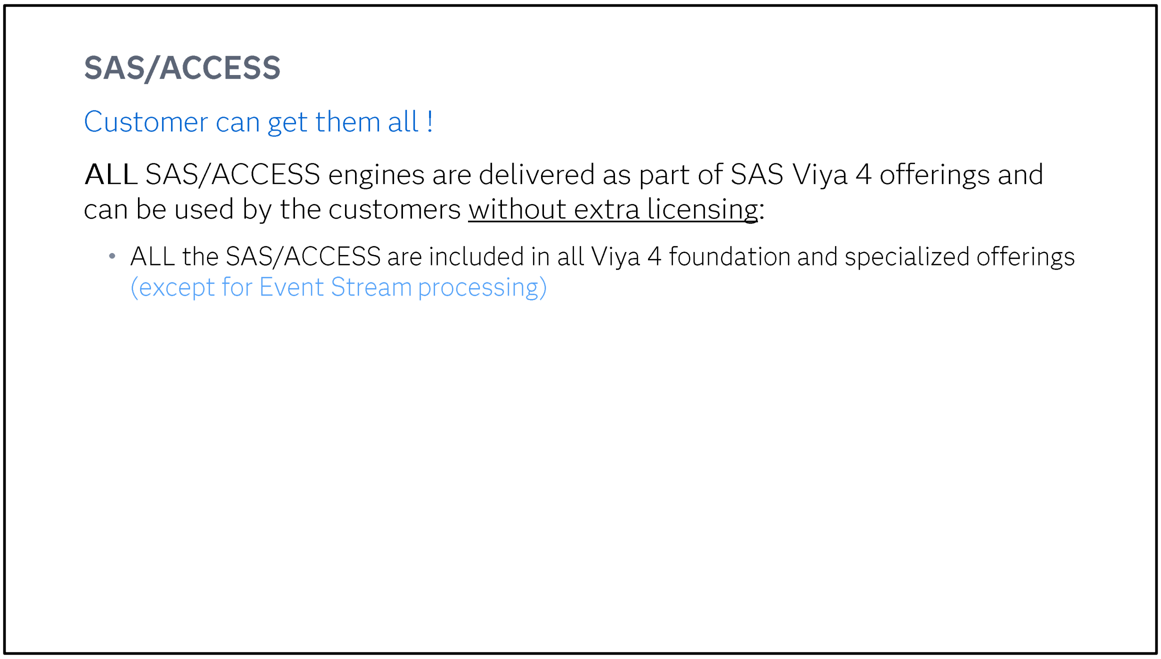
LTS 2024.09

- SAS Real-Time Watchlist Screening for Entities (2024.04)
- SAS Real-Time Watchlist Screening for Payments (2024.07)
- SAS Integrated Regulatory Reporting (2024.09)
- SAS solutions from the Retail and Consumer Goods division can now be deployed on all the supported cloud platforms

LTS 2025.03

- SAS Data Quality for Payment Integrity Health Care (2024.10)
- SAS Real-Time Watchlist Screening for Entities / for Payments : support for deployment on AWS, Google Cloud, and Red Hat OpenShift added.
- Changes to SAS Risk Solutions : , the Risk Cirrus Core component no longer requires a Git repository.
- SAS/ACCESS Interface to Teradata now includes the required Teradata client by default.

Since the first GA release of Viya 4 (which was LTS 2020.1 released in November 2020), a lot of work has been done to move more and more solutions to Viya 4, new solutions, new capabilities, new SAS/ACCESS engines, and new offerings (like SAS Data Engineering).
Each month, you can check the “What’s new in Operations” section to learn what new solutions are available on which Kubernetes platforms in the latest version of SAS Viya.



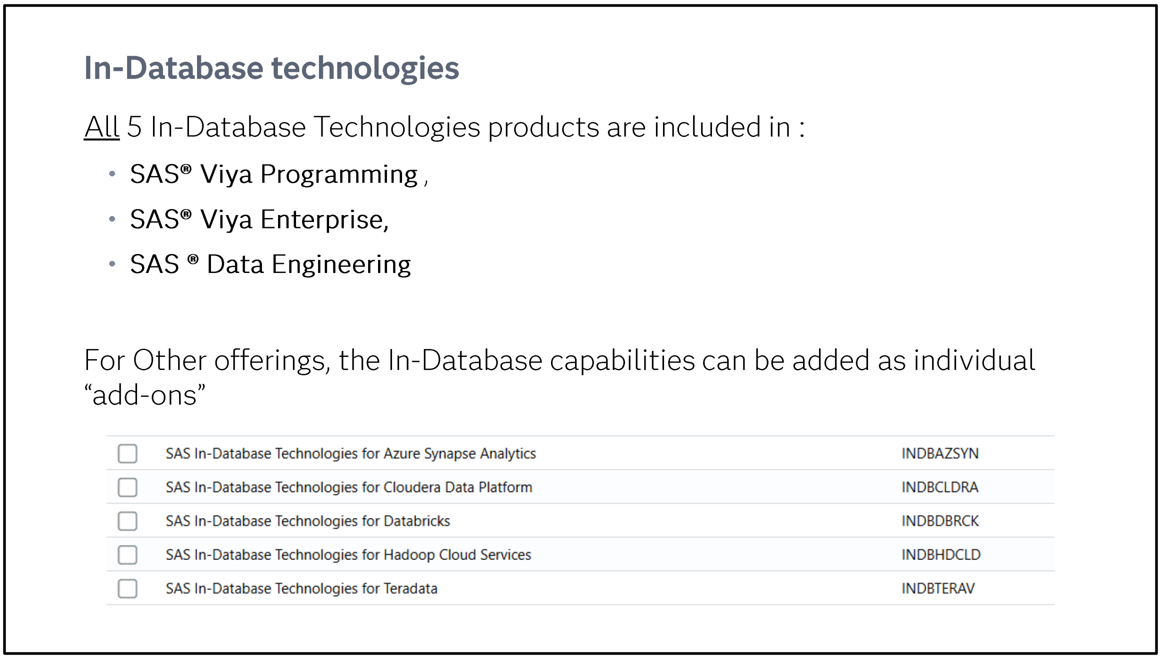
SAS/ACCESS

Customer can get them all !

ALL SAS/ACCESS engines are delivered as part of SAS Viya 4 offerings and can be used by the customers without extra licensing:

- ALL the SAS/ACCESS are included in all Viya 4 foundation and specialized offerings (except for Event Stream processing)

Something important to note (which is new with SAS Viya 4) is that all the SAS Access engines are included in the Foundational and Specialized Offerings (except for event stream processing and Visual investigator).



In-Database technologies

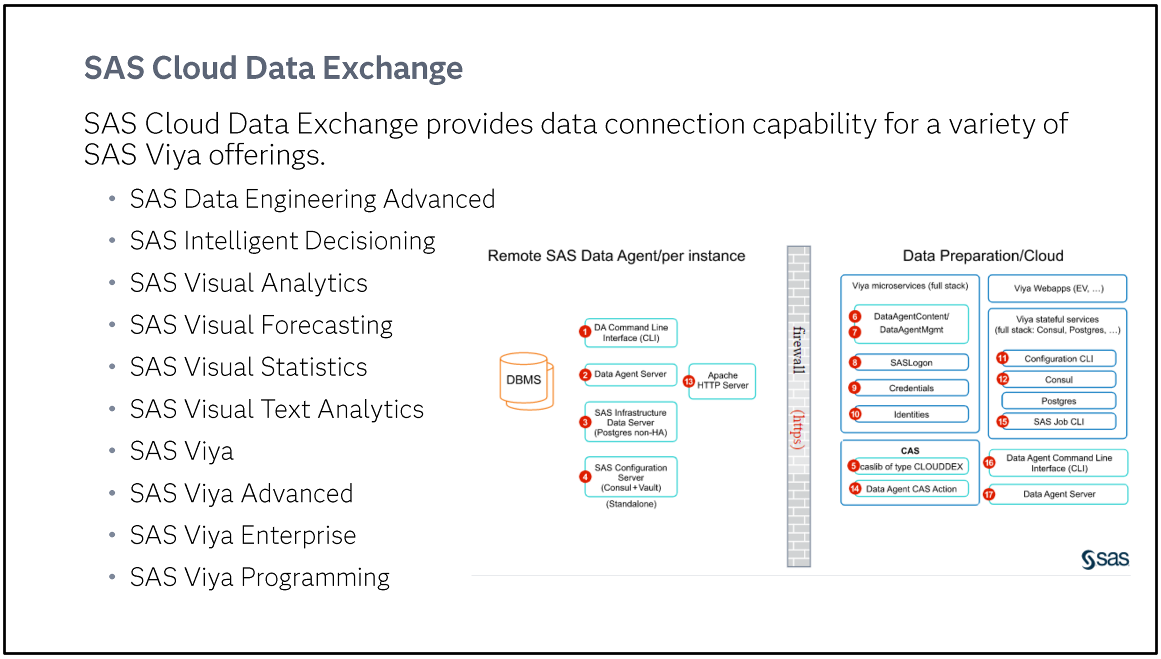
All 5 In-Database Technologies products are included in :

- SAS® Viya Programming ,
- SAS® Viya Enterprise,
- SAS ® Data Engineering

For Other offerings, the In-Database capabilities can be added as individual "add-ons"

☐	SAS In-Database Technologies for Azure Synapse Analytics	INDBAZSYN
☐	SAS In-Database Technologies for Cloudera Data Platform	INDBCLDRA
☐	SAS In-Database Technologies for Databricks	INDBDBRCK
☐	SAS In-Database Technologies for Hadoop Cloud Services	INDBHDCLD
☐	SAS In-Database Technologies for Teradata	INDBTERAV

Also, All the in-database technologies products will be included in these three offerings: SAS Viya Programming, SAS Viya Enterprise, and SAS Data Engineering.
If you have another type of offering, you can choose to add and license some in-database capabilities individually.



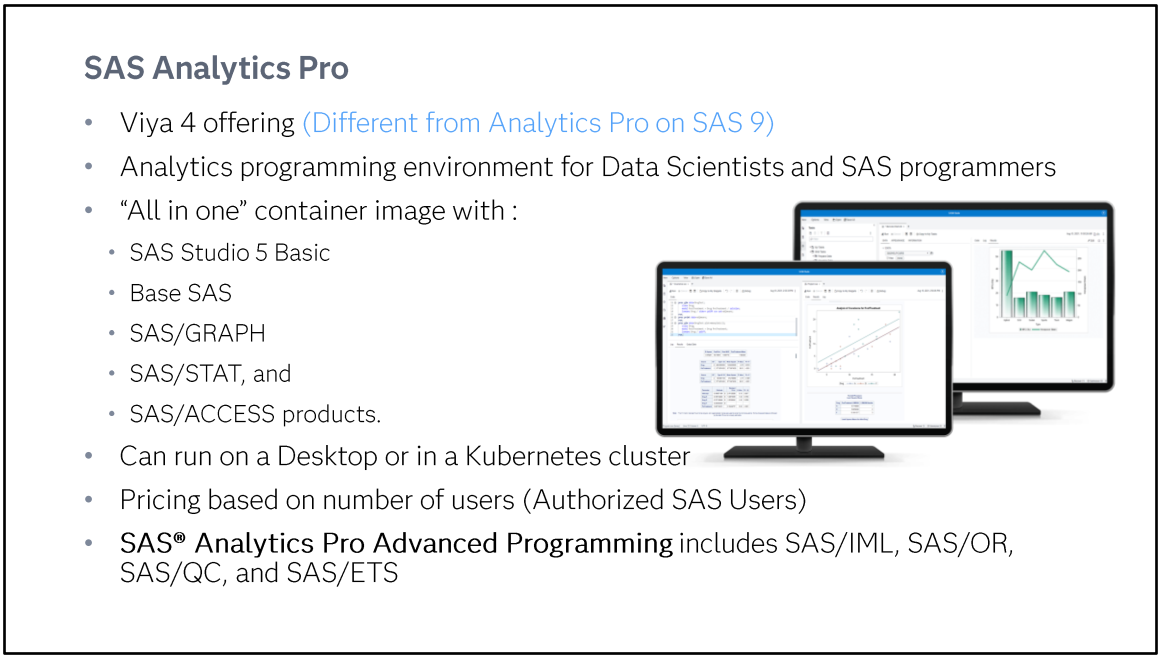
SAS Cloud Data Exchange

SAS Cloud Data Exchange provides data connection capability for a variety of SAS Viya offerings.

- SAS Data Engineering Advanced
- SAS Intelligent Decisioning
- SAS Visual Analytics
- SAS Visual Forecasting
- SAS Visual Statistics
- SAS Visual Text Analytics
- SAS Viya
- SAS Viya Advanced
- SAS Viya Enterprise
- SAS Viya Programming

Remote SAS Data Agent/per instance
DBMS
1 DA Command Line Interface (CLI)
2 Data Agent Server
13 Apache HTTP Server
3 SAS Infrastructure Data Server (Postgres non-HA)
4 SAS Configuration Server (Consul + Vault)
(Standalone)
firewall
(https)
Data Preparation/Cloud
Viya microservices (full stack)
6 7 DataAgentContent/ DataAgentMgmt
8 SASLogon
9 Credentials
10 Identities
CAS
5 caslib of type CLOUDDEX
14 Data Agent CAS Action
Viya Webapps (EV, ...)
Viya stateful services (full stack: Consul, Postgres, ...)
11 Configuration CLI
12 Consul
Postgres
15 SAS Job CLI
16 Data Agent Command Line Interface (CLI)
17 Data Agent Server
sas

What about Cloud Data Exchange?
This product is intended to provide a connection to a customer’s on-premise databases for the variety of the Viya offerings running in the Cloud.
As previously noted, it is included in most offerings (which are listed on the screen).
In the diagram here, you can see the architecture, that shows how this Cloud Data Exchange product is working (at a high level).
You have a co-located data agent running by default in Viya and a remote data Agent that is deployed on the customer site.
Basically, CDE lets you read the data from the on-premise environment of the customer and load that into your SAS Viya platform wherever it is.
Finally, if you want to learn more about SAS Cloud Data exchange, be aware that a dedicated course is available on learn.sas.com.



SAS Analytics Pro

- Viya 4 offering (Different from Analytics Pro on SAS 9)
- Analytics programming environment for Data Scientists and SAS programmers
- "All in one" container image with :
  - SAS Studio 5 Basic
  - Base SAS
  - SAS/GRAPH
  - SAS/STAT, and
  - SAS/ACCESS products.
- Can run on a Desktop or in a Kubernetes cluster
- Pricing based on number of users (Authorized SAS Users)
- **SAS® Analytics Pro Advanced Programming** includes SAS/IML, SAS/OR, SAS/QC, and SAS/ETS

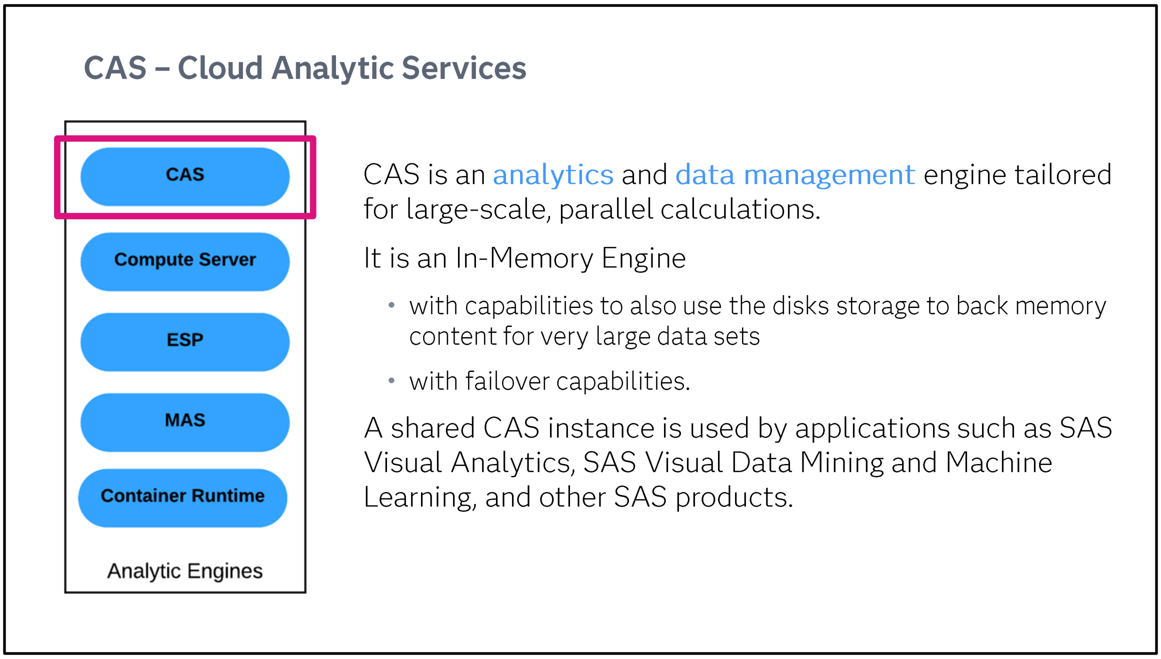
SAS Analytics Pro is a very specific offering.
It's a standalone offering for data scientists and SAS programmers who want to run SAS on their own machines.
It's a single big container with SAS Studio 5, base SAS, SAS Graph, SAS STAT, and SAS Access products and is very easy to deploy.
SAS Analytics Pro can run on a desktop or in a Kubernetes cluster and you have all the standard base SAS features.
The pricing is based on the number of users.
"SAS Analytics Pro advanced programming" is a more complete version of the offering which also includes SAS IML, SAS OR, SAS QC, and SAS ETS,
if the customer has more needs in terms of programming capabilities…

Lesson 3: Architecture Components

3.1 CAS

CAS

Within the list of SAS Viya components and services, let’s review the Cloud Analytic Services engine, or CAS in short, and understand its components.



CAS – Cloud Analytic Services

CAS

Compute Server

ESP

MAS

Container Runtime

Analytic Engines

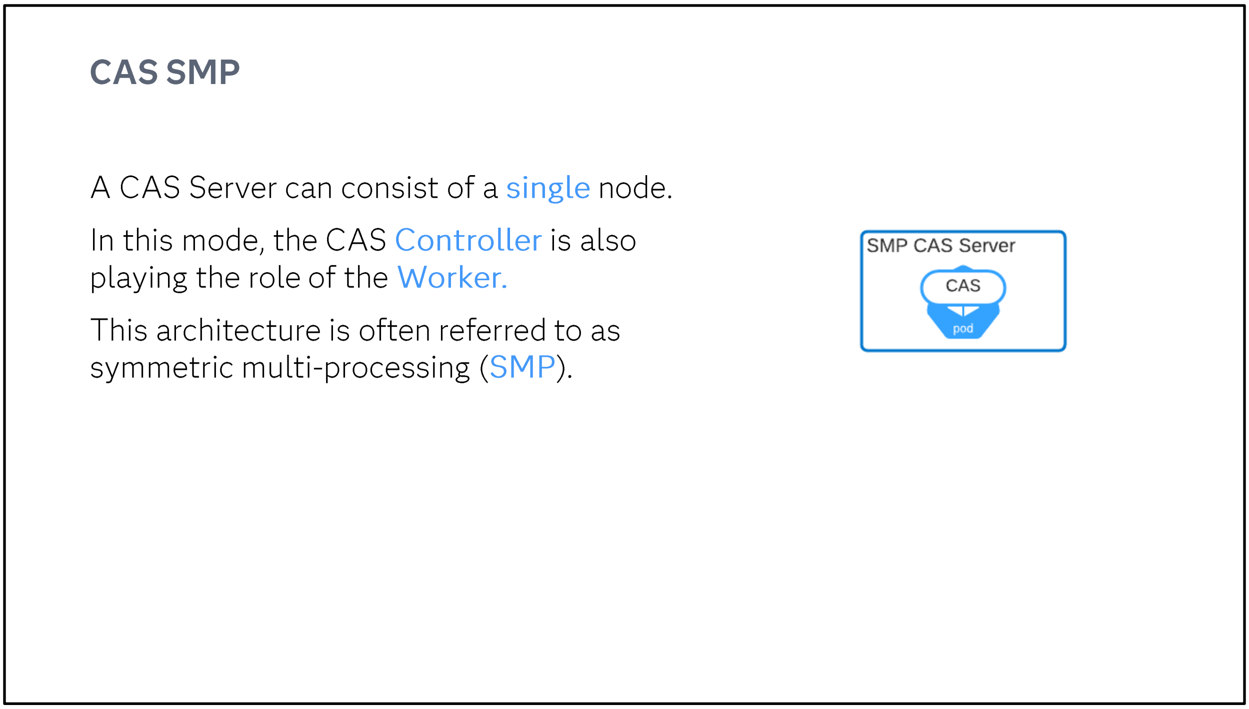
CAS is an analytics and data management engine tailored for large-scale, parallel calculations.

It is an In-Memory Engine

- with capabilities to also use the disks storage to back memory content for very large data sets
- with failover capabilities.

A shared CAS instance is used by applications such as SAS Visual Analytics, SAS Visual Data Mining and Machine Learning, and other SAS products.

The CAS engine provides different capabilities ranging from analytics to data management.
It’s an in-memory engine, tailored for large-scale, parallel calculations, that evolved from the LASR engine available with SAS 9. On top of that, it adds the capabilities to use the storage to back memory content if the memory is not enough, and to support failover in case of node failures or planned updates.
A shared CAS instance is used by applications such as SAS Visual Analytics, SAS Visual Data Mining and Machine Learning, and other SAS products.



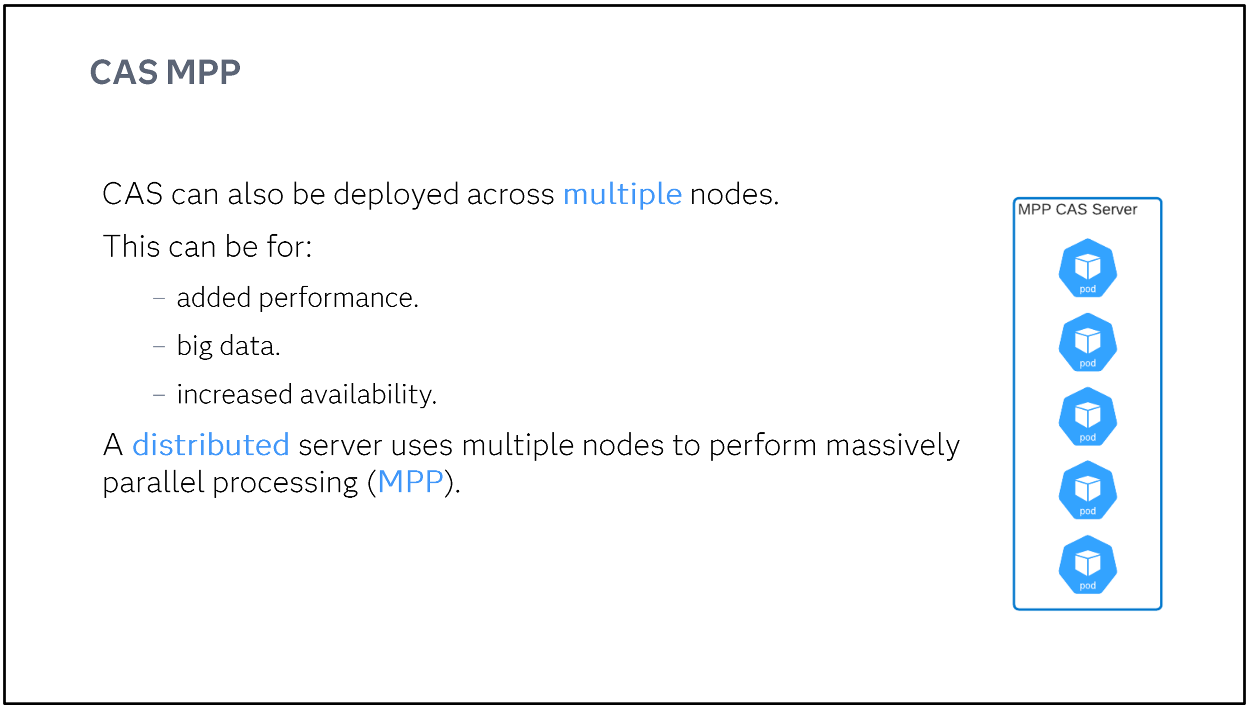
CAS SMP

A CAS Server can consist of a single node.

In this mode, the CAS Controller is also playing the role of the Worker.

This architecture is often referred to as symmetric multi-processing (SMP).

A CAS Server can consist of a single node. In this case, the single instance performs both the tasks of a controller (such as accepting client connections, enforcing security, summarizing and returning results), and the tasks of a worker (such as performing data management and data analysis on the rows of data that are in-memory).
CAS uses multiple CPUs and parallelizes tasks across multiple threads to speed up data analysis. This architecture is often referred to as symmetric multi-processing, or, in short, SMP.



CAS MPP

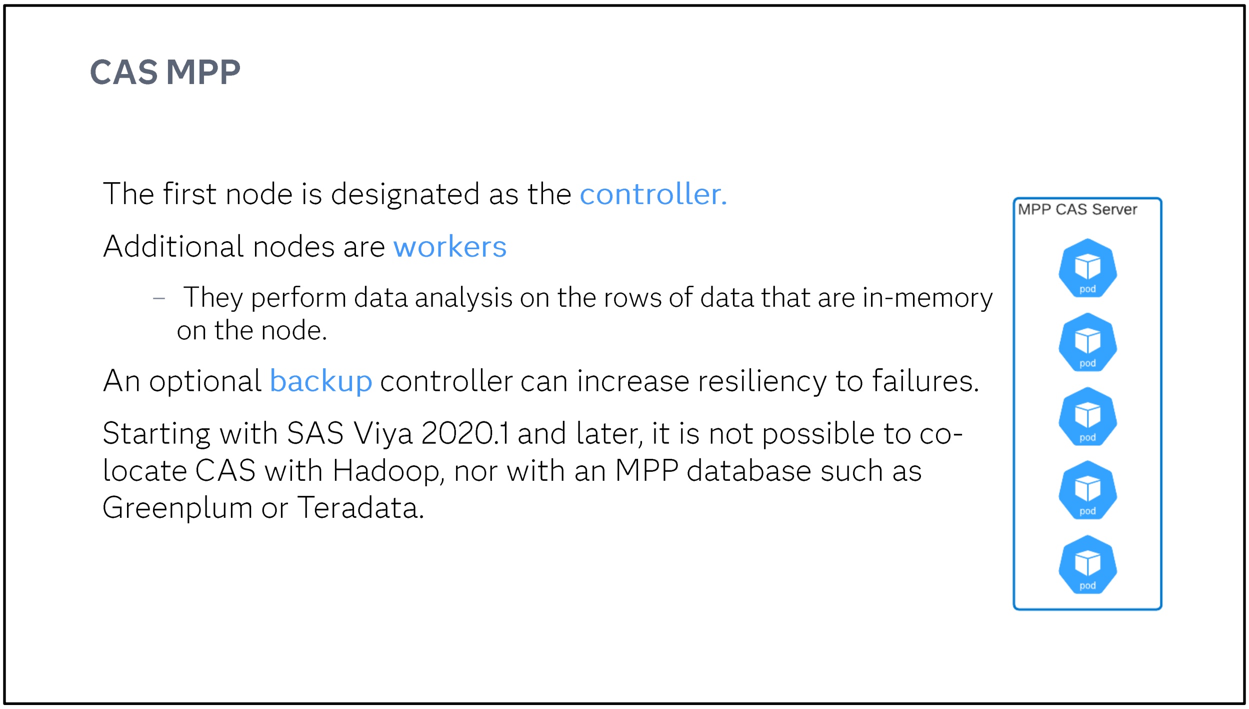
CAS can also be deployed across multiple nodes.

This can be for:

- added performance.
- big data.
- increased availability.

A distributed server uses multiple nodes to perform massively parallel processing (MPP).

To support very large data sets and provide additional performance, CAS can be deployed across multiple nodes. This comes with the added benefit of increased resilience to failures.
A distributed server uses multiple nodes to perform massively parallel processing, and thus it is called MPP.



CAS MPP

The first node is designated as the controller.

Additional nodes are workers

- They perform data analysis on the rows of data that are in-memory on the node.

An optional backup controller can increase resiliency to failures.

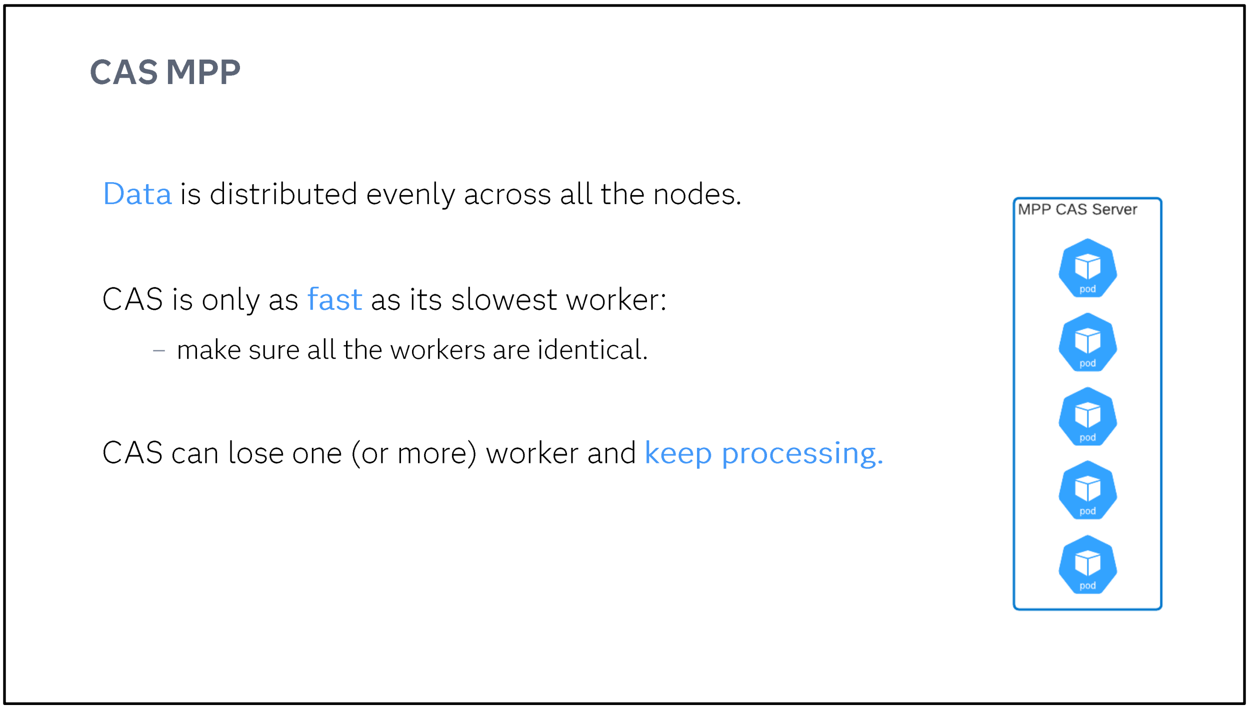
Starting with SAS Viya 2020.1 and later, it is not possible to co-locate CAS with Hadoop, nor with an MPP database such as Greenplum or Teradata.

In the case of multi-node deployment, one node is designated as the controller. Client applications communicate with the controller. The controller splits and distributes sub-tasks to the nodes and aggregates the results. The controller maintains the status of the cluster, including user sessions, and enforces security.

The other nodes are workers. They execute the analytics and data management jobs in parallel, each one acting on the table rows loaded in the node’s memory.

Optionally, it is possible to configure one additional node as a dedicated backup controller, ready to step in if the primary controller fails.

For those familiar with SAS Viya 3.5, it is worth noting that with SAS Viya 2020.1 and later it is not possible to co-locate CAS with Hadoop, nor with an MPP database such as Greenplum or Teradata.
CAS can still access data that is in Hadoop or in these databases, but CAS itself, the engine, cannot be co-located on the same nodes as these.



CAS MPP

Data is distributed evenly across all the nodes.

CAS is only as fast as its slowest worker:

- make sure all the workers are identical.

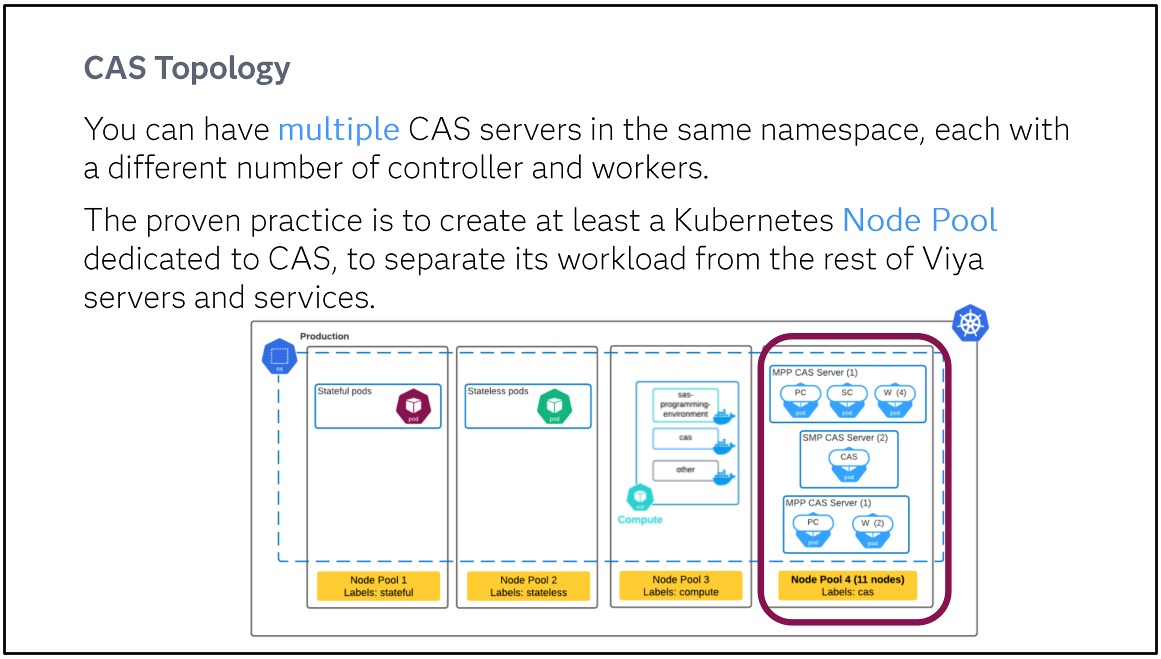
CAS can lose one (or more) worker and keep processing.

MPP CAS Server
pod
pod
pod
pod
pod

During the initial table loading, data is automatically and evenly distributed across all the nodes.
By default, data is not re-distributed when the number of nodes changes, for example when scaling up or down. If desired, CAS can be configured to automate data re-allocation in those cases.

A CAS distributed instance is only as fast as its slowest node. That's because the controller checks how data is distributed on the nodes, and then it splits the compute load across the nodes proportionally to data chunks distribution. By default, data is split evenly, and that's one reason why we recommend to have all the worker nodes allocated with the same physical resources.
In case of uneven workload distribution, the controller will have to wait on the slowest node before it can send back the results to the client.

A CAS distributed instance is resilient to failures. Depending on its configuration, it can lose one or more nodes and keep going.
That’s because data is split in chunks, and each chunk has multiple copies spread out on different nodes by default.



CAS Topology

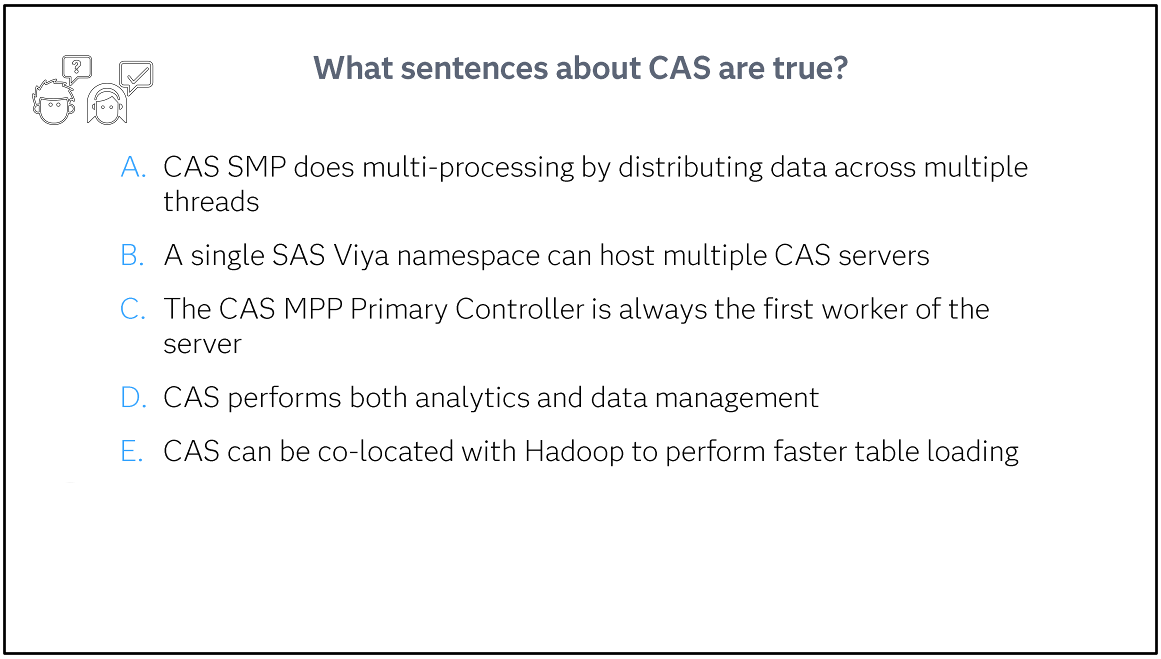
You can have multiple CAS servers in the same namespace, each with a different number of controller and workers.

The proven practice is to create at least a Kubernetes Node Pool dedicated to CAS, to separate its workload from the rest of Viya servers and services.

Multiple CAS servers can be deployed in the same environment, in the same namespace. You are free to configure each one as needed, mixing SMP and MPP instances. Each server can have a different configuration.

SAS recommends to configure the Kubernetes cluster with a dedicated node pool for CAS, and to configure SAS Viya to allocate each CAS pod to a dedicated Kubernetes node. This guarantees reliable performance, but it implies that, if you deploy additional CAS servers, you need more Kubernetes nodes.

In the example shown in this diagram, there are two MPP CAS servers and two SMP. The MPP server at the top has a primary and a secondary controller plus 4 workers; the MPP server at the bottom only has one primary controller and 2 workers. In total there are 11 CAS pods, so this environment would need 11 nodes in the dedicated node pool.



What sentences about CAS are true?

A. CAS SMP does multi-processing by distributing data across multiple threads

B. A single SAS Viya namespace can host multiple CAS servers

C. The CAS MPP Primary Controller is always the first worker of the server

D. CAS performs both analytics and data management

E. CAS can be co-located with Hadoop to perform faster table loading

Before we close the lesson, we have a quiz. Which of the following sentences are true?

A:

CAS SMP does multi-processing by distributing data across multiple threads.

B:

A single Viya namespace can host multiple CAS servers.

C:

The CAS MPP Primary Controller is always the first worker of the server.

D:

CAS performs both analytics and data management.

E:

CAS can be co-located with Hadoop to perform faster table loading.

Take a moment to think about this, and answer. If you want, you can pause here before listening.

Here come the answers.

A is true. CAS SMP does multi-processing across multiple threads. So even it deployed on a single node, you can still have multi-processing across all the available CPUs, spreading multiple threads on the cores.

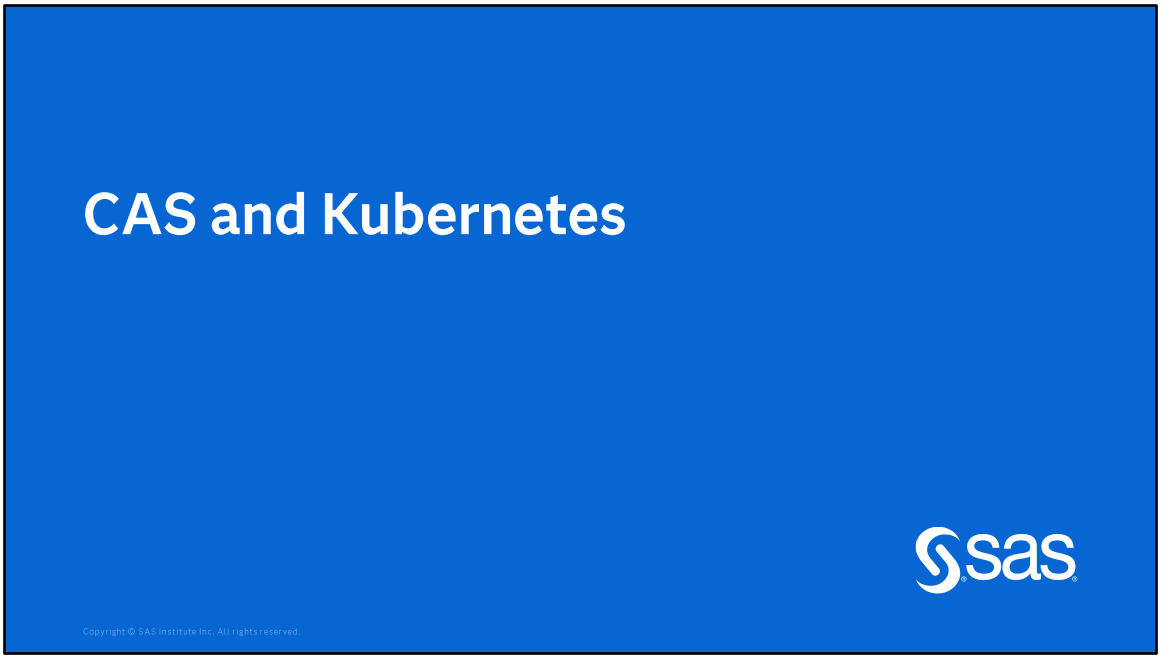
B is true as well; we have seen in the previous slide how it's possible to deploy multiple CAS servers in a single SAS Viya platform.

C is false. Why? Because when you have a distributed environment, a controller is not a worker. With MPP CAS servers, each CAS node has a specific role. It's only with single-host SMP CAS servers that a single node acts both as a controller and as a worker.

D is true. CAS is not only an analytics engine but it's also a data management engine.

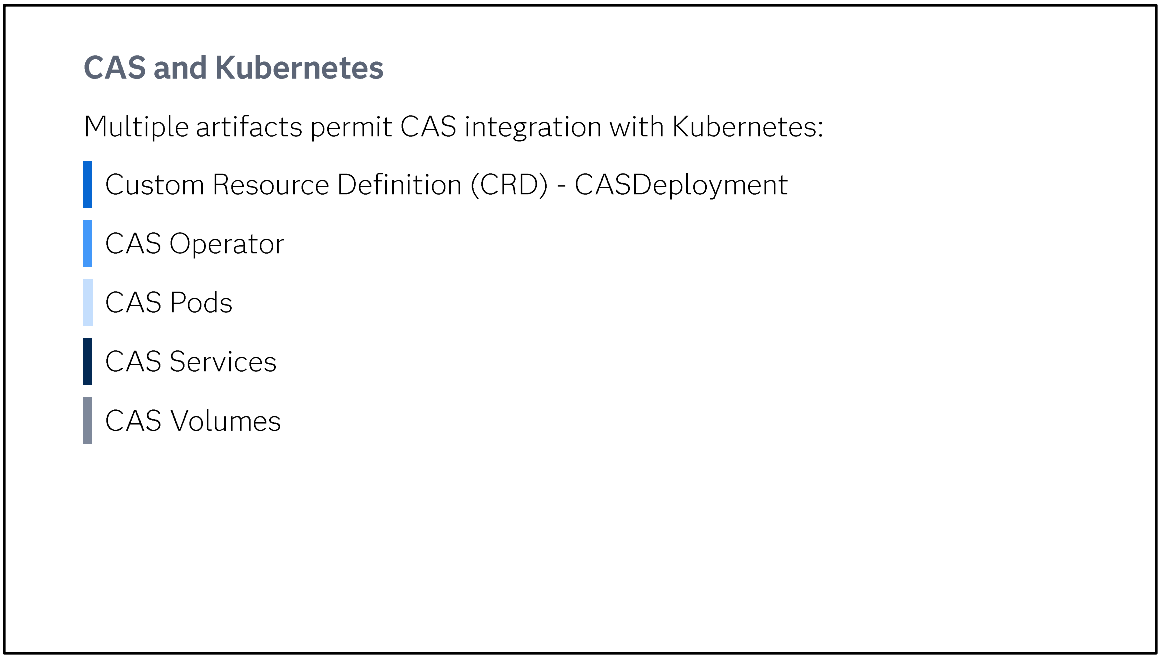
Finally, E is false, CAS cannot be co-located with Hadoop. That was a possibility with previous SAS versions that has been dropped with the current release in favors of Kubernetes integration.

In summary, the correct answers are "A", "B" and "D".



CAS and Kubernetes

There are multiple integration points between CAS architecture and Kubernetes. Let’s review the most significative.

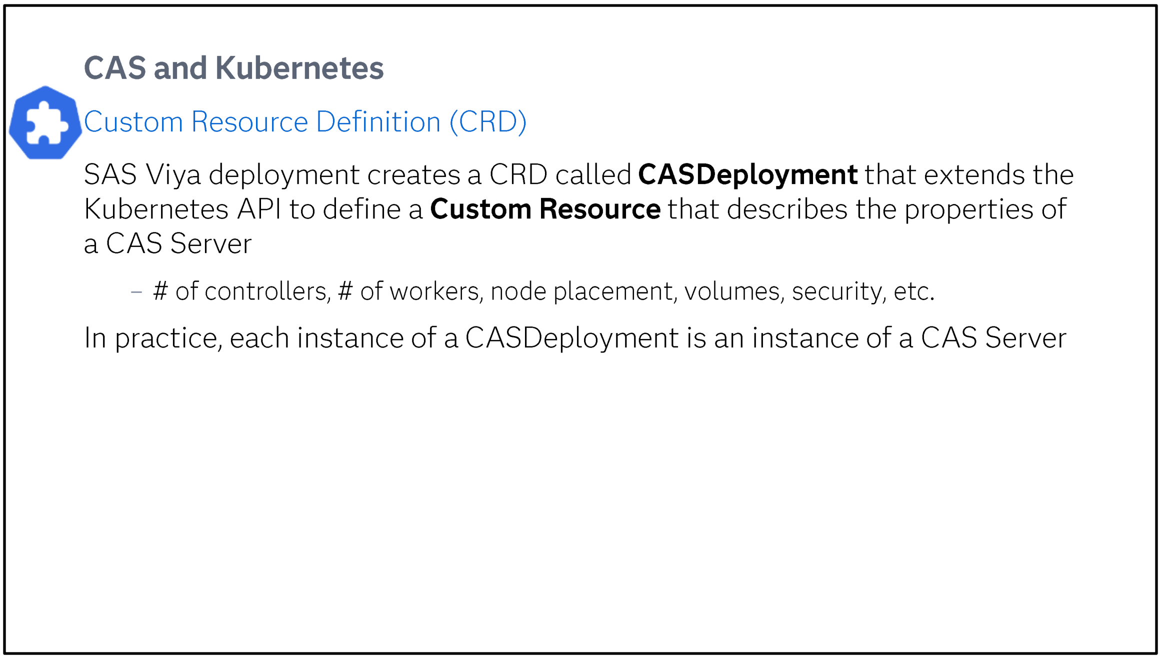


CAS and Kubernetes

Multiple artifacts permit CAS integration with Kubernetes:

- Custom Resource Definition (CRD) - CASDeployment
- CAS Operator
- CAS Pods
- CAS Services
- CAS Volumes

CAS integrates with Kubernetes in multiple ways. The SAS Viya platform leverages a variety of Kubernetes objects to manage CAS lifecycle and capabilities. We can list: a Custom Resource Definition, or, in short, CRD, named CASDeployment; a CAS operator; CAS pods; CAS services; and, finally, CAS Volumes.



CAS and Kubernetes

Custom Resource Definition (CRD)

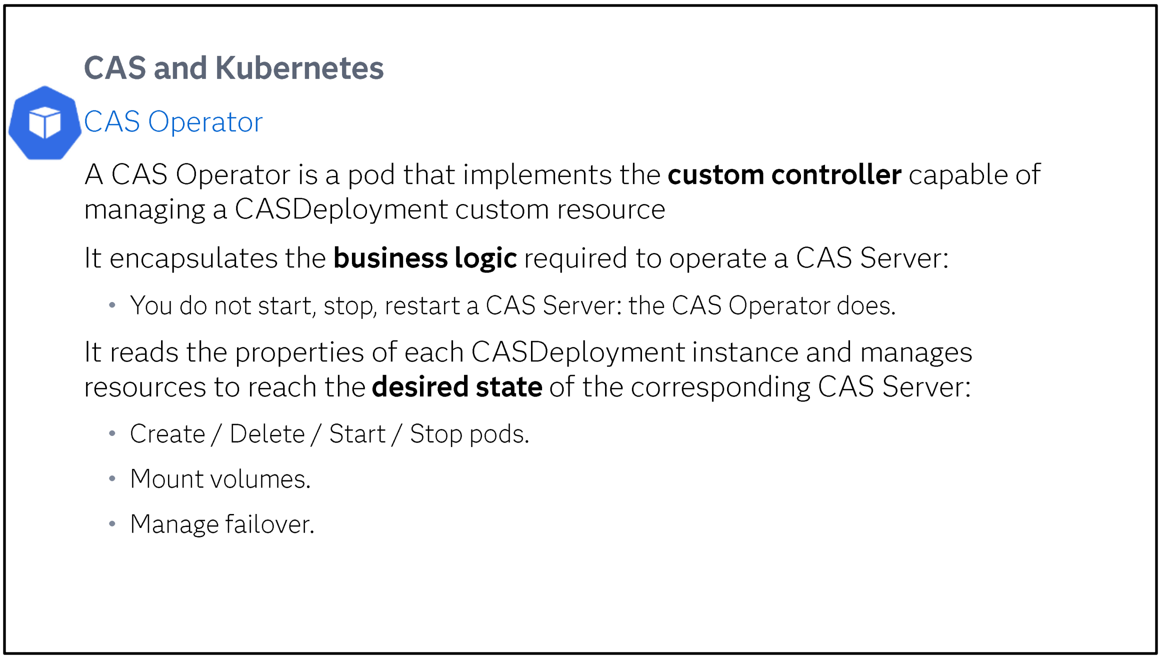
SAS Viya deployment creates a CRD called **CASDeployment** that extends the Kubernetes API to define a **Custom Resource** that describes the properties of a CAS Server

- # of controllers, # of workers, node placement, volumes, security, etc.

In practice, each instance of a CASDeployment is an instance of a CAS Server

Custom Resource Definitions are a Kubernetes artifact that permits to extend the Kubernetes API by defining custom objects.
The SAS Viya platform defines a Custom Resource called CASDeployment that describes the properties of a CAS Server, such as the number and roles of controllers, the number of workers, node placement, volumes, security, and so on.

In practice, there is an instance of a CASDeployment for each CAS Server deployed in the environment.



CAS and Kubernetes

CAS Operator

A CAS Operator is a pod that implements the **custom controller** capable of managing a CASDeployment custom resource

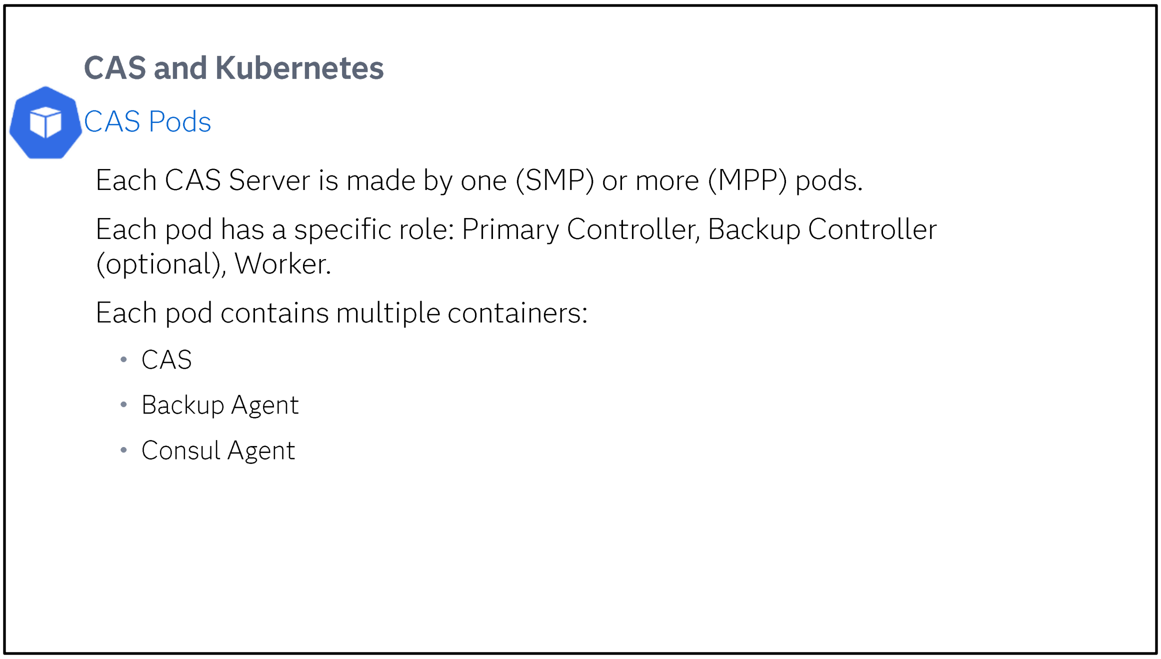
It encapsulates the **business logic** required to operate a CAS Server:

- You do not start, stop, restart a CAS Server: the CAS Operator does.

It reads the properties of each CASDeployment instance and manages resources to reach the **desired state** of the corresponding CAS Server:

- Create / Delete / Start / Stop pods.
- Mount volumes.
- Manage failover.

Since Kubernetes does not know how to manage custom resources, the SAS Viya platform provides a CAS Operator to do it.
The CAS Operator is a pod that contains the business logic to operate a CAS Server according to its description included in the CASDeployment custom resource.
According to Kubernetes practices, the CASDeployment custom resource contains the description of the desired state of a CAS server; the CAS Operator continuously monitors it and acts on the live Kubernetes cluster so that what is running corresponds to what is desired.
As an example, a SAS administrator does not start or stop CAS pods directly. Rather, an administrator changes a property of the CASDeployment resource to be “shutdown equal true” or “shutdown equal false”, and the CAS operator acts accordingly. In the same way, the CAS Operator mounts volumes in CAS pods, manages pod failover in case of failures, and much more.



CAS and Kubernetes

CAS Pods

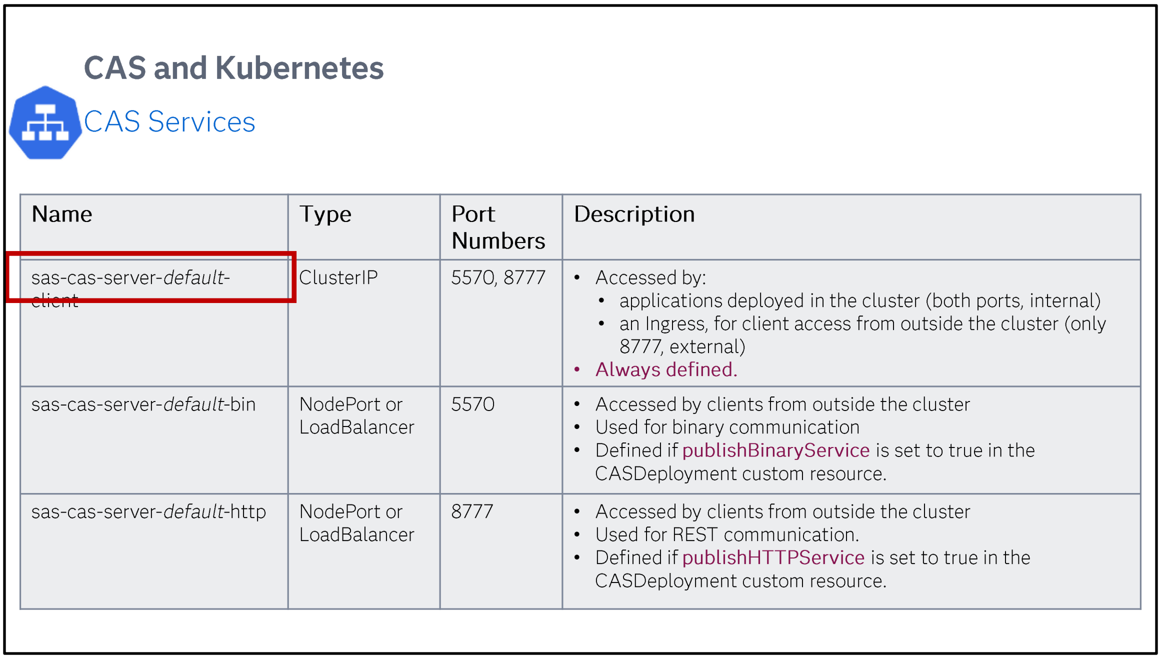
Each CAS Server is made by one (SMP) or more (MPP) pods.

Each pod has a specific role: Primary Controller, Backup Controller (optional), Worker.

Each pod contains multiple containers:

- CAS
- Backup Agent
- Consul Agent

Each instance of a CAS SMP server runs in one Kubernetes pod. CAS MPP servers are distributed in multiple pods. Each pod corresponds to a CAS node with a specific role – one pod will host the primary controller, one pod the backup controller, and then one pod for each CAS worker.
Each pod contains multiple containers, the exact composition depends on your site-specific configuration. Usually there is a container hosting the main CAS processes, one for a backup agent, one for a Consul agent, and possibly additional ones.



CAS and Kubernetes

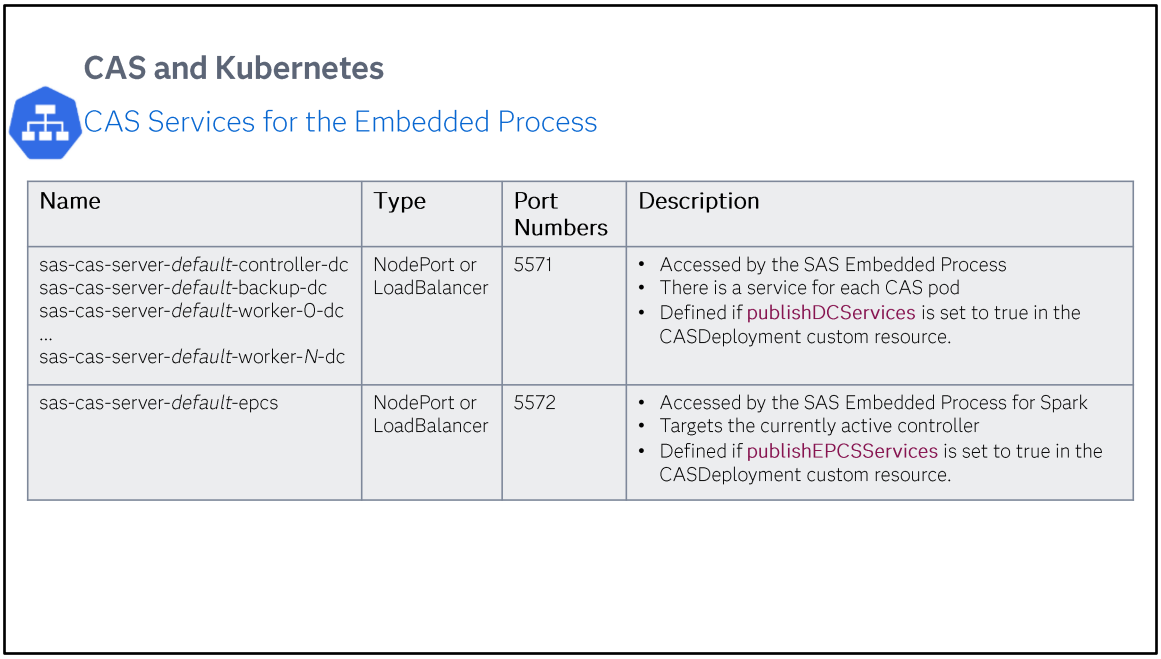
CAS Services

Name	Type	Port Numbers	Description
sas-cas-server-*default*-client	ClusterIP	5570, 8777	• Accessed by: • applications deployed in the cluster (both ports, internal) • an Ingress, for client access from outside the cluster (only 8777, external) • Always defined.
sas-cas-server-*default*-bin	NodePort or LoadBalancer	5570	• Accessed by clients from outside the cluster • Used for binary communication • Defined if publishBinaryService is set to true in the CASDeployment custom resource.
sas-cas-server-*default*-http	NodePort or LoadBalancer	8777	• Accessed by clients from outside the cluster • Used for REST communication. • Defined if publishHTTPService is set to true in the CASDeployment custom resource.

The SAS Viya platform can define multiple Kubernetes services to access CAS endpoints.

The default service is called “SAS CAS server default client”, and additional ones can be defined as required. More details are provided in the networking lessons.

If you deploy a secondary (or backup) CAS controller, Kubernetes takes care to always point these services to the controller that is currently active. Normally, this is the primary controller, and it will be the backup controller in case the primary fails.

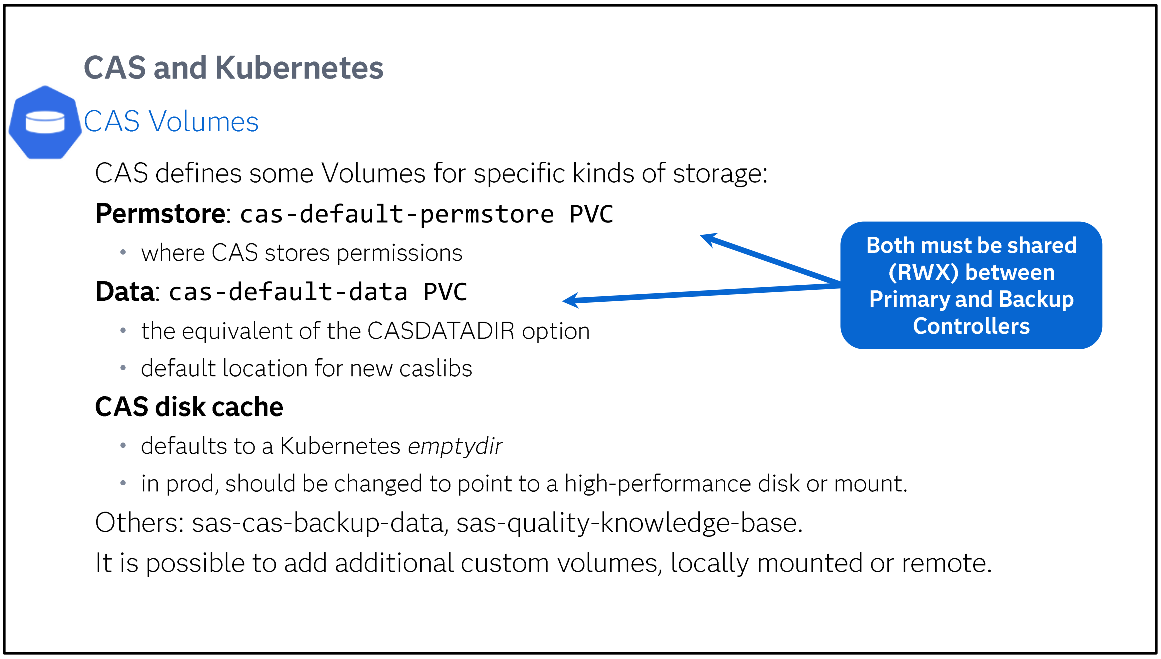


CAS and Kubernetes

CAS Services for the Embedded Process

Name	Type	Port Numbers	Description
sas-cas-server-*default*-controller-dc sas-cas-server-*default*-backup-dc sas-cas-server-*default*-worker-0-dc … sas-cas-server-*default*-worker-*N*-dc	NodePort or LoadBalancer	5571	• Accessed by the SAS Embedded Process • There is a service for each CAS pod • Defined if publishDCServices is set to true in the CASDeployment custom resource.
sas-cas-server-*default*-epcs	NodePort or LoadBalancer	5572	• Accessed by the SAS Embedded Process for Spark • Targets the currently active controller • Defined if publishEPCSServices is set to true in the CASDeployment custom resource.

If your site deploys the SAS embedded process in an external databases or in Hadoop, there will be additional Kubernetes services so that the SAS embedded process can reach the required CAS endpoints.



CAS and Kubernetes

CAS Volumes

CAS defines some Volumes for specific kinds of storage:

Permstore: `cas-default-permstore PVC`

- where CAS stores permissions

Data: `cas-default-data PVC`

- the equivalent of the CASDATADIR option
- default location for new caslibs

CAS disk cache

- defaults to a Kubernetes *emptydir*
- in prod, should be changed to point to a high-performance disk or mount.

Others: sas-cas-backup-data, sas-quality-knowledge-base.

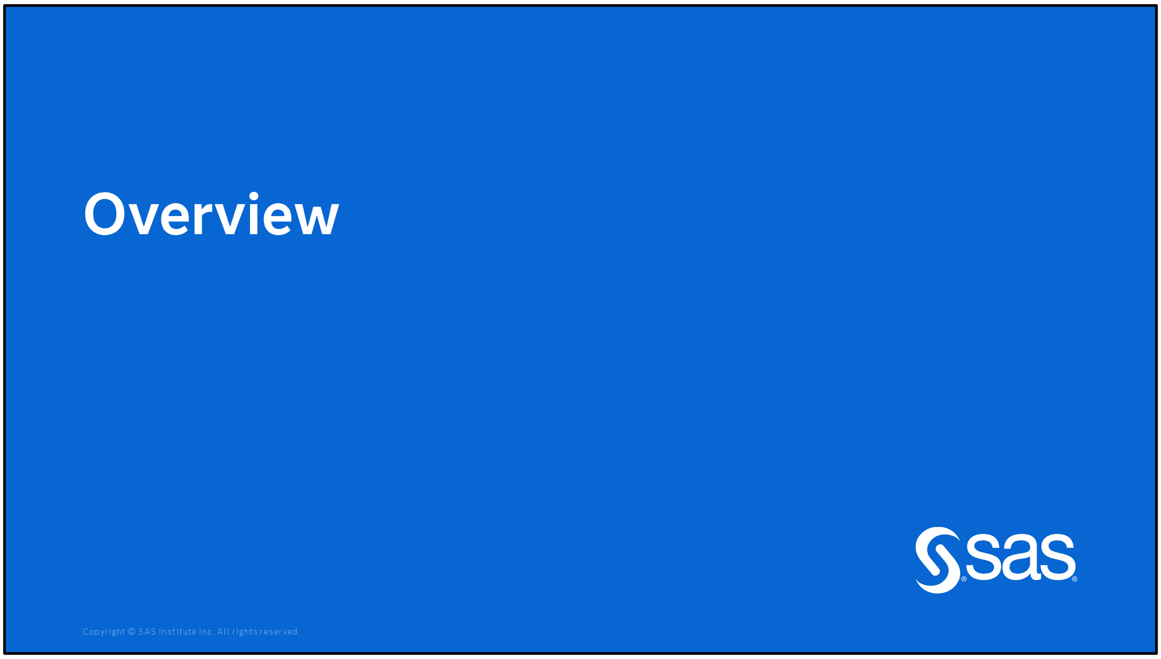
It is possible to add additional custom volumes, locally mounted or remote.

Both must be shared (RWX) between Primary and Backup Controllers

By default, data storage in containers is ephemeral. When a container is deleted, data stored locally is lost. For durable storage, data should be maintained in persistent volumes.
The SAS Viya platform includes multiple volumes dedicated to specific CAS areas, including the “Permstore” to store CAS permission definitions, the cas-default-data volume for the default data location for new CAS libraries, or the “CAS disk cache” as the backend area for in-memory tables.
It is possible to add additional custom volumes, using either locally mounted or remote storage

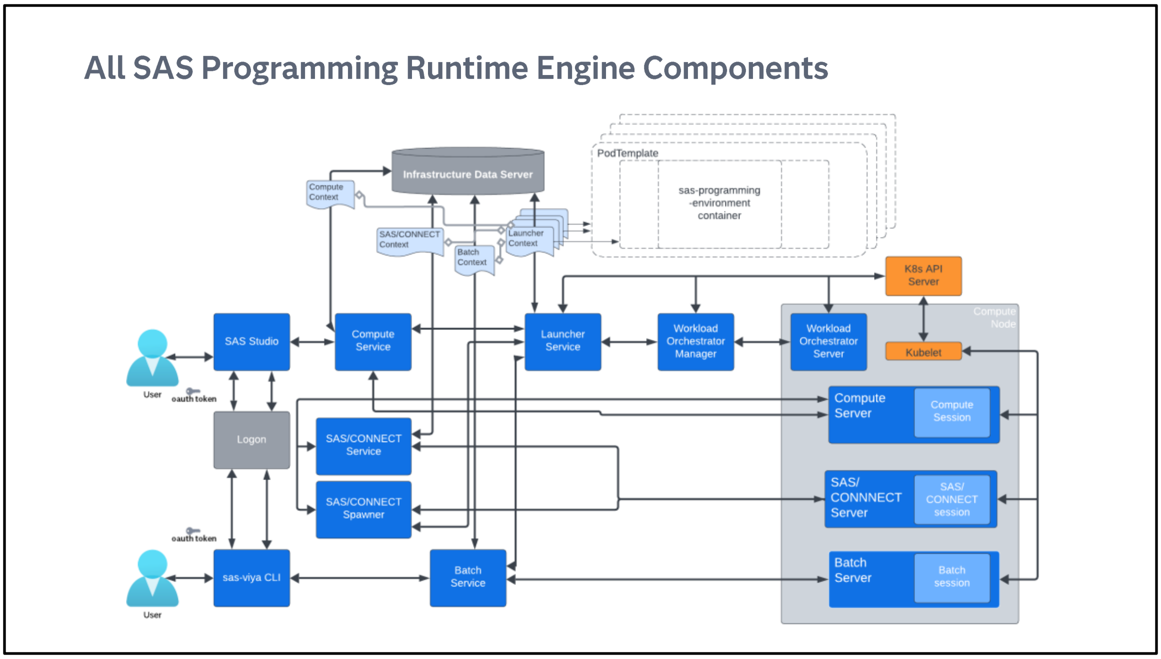
If you deploy a secondary (or backup) CAS controller, some of these volumes must be shared between the primary and backup controllers with RWX access.
Additional details can be found in the storage lessons.

3.2 SAS Programming Runtime Components



Overview

Let’s review some high-level diagrams that present the different components of the SAS programming runtime engine and help us identify and describe those components.

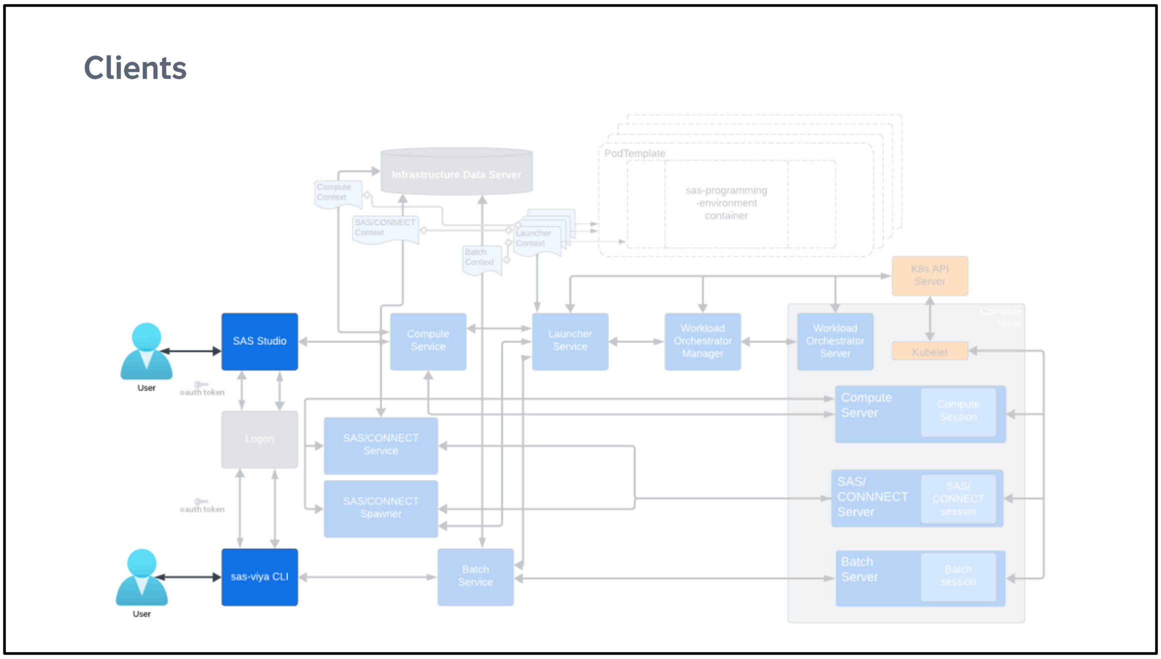


All SAS Programming Runtime Engine Components

Here we start with a rather complex diagram, where everything related to the SAS programming runtime engine is presented in a single page.
In the next slides we will highlight and discuss the individual components.
Here they are all presented together to let you familiarize with them.

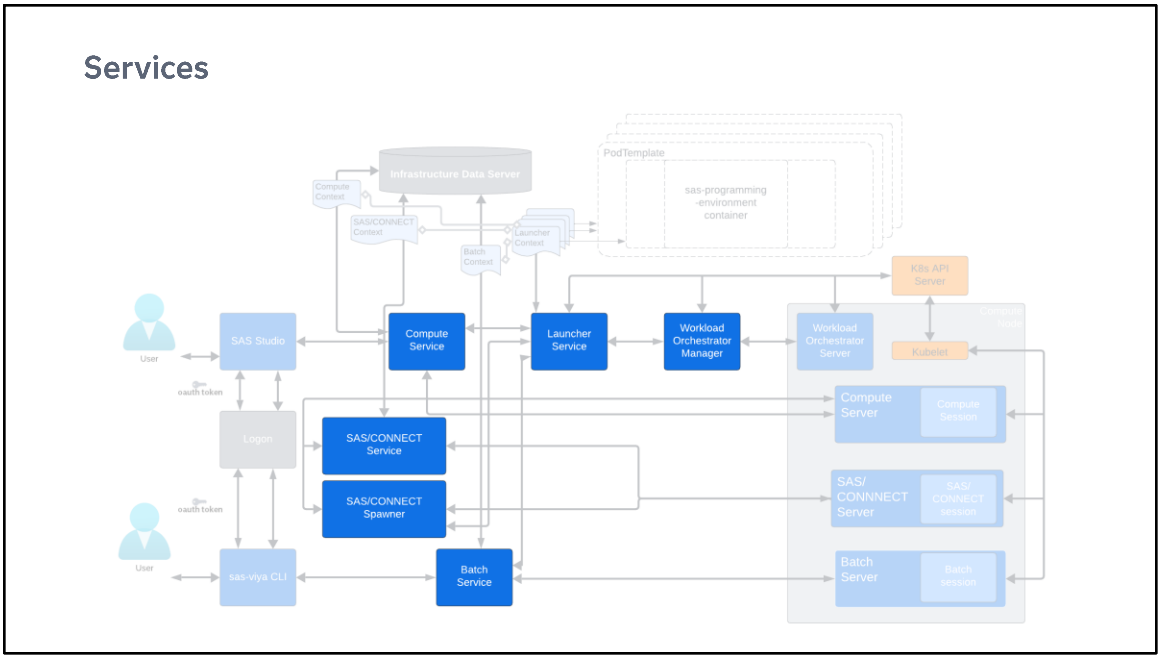
Just looking at the names in the boxes, you can recognize that there is a compute programming runtime, a SAS Connect programming runtime and a batch server programming runtime. We will highlight their peculiarities in the next slides.

Now, let’s read the diagram from left to right, highlighting different logical categories: clients, middle-tier services, backend servers.



Clients

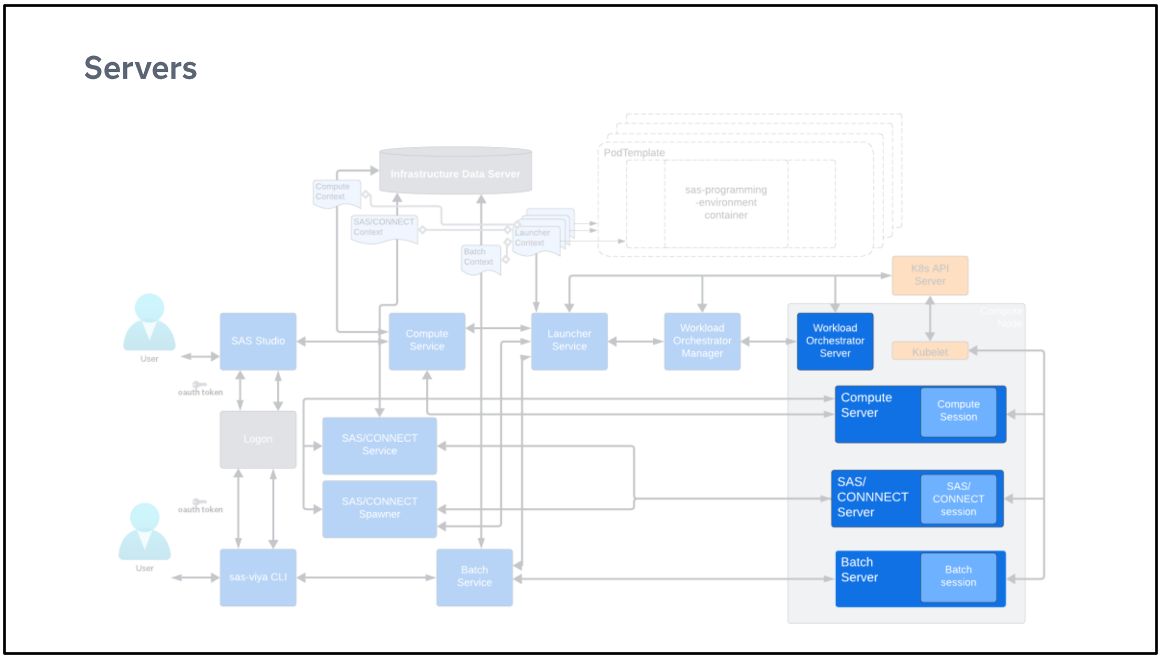
On the left side we can see some sample clients.
This is not an exhaustive representation, but we can see here two kinds of clients. There are visual applications, such as SAS Studio. Then there are interactive tools such as the sas-viya command-line interface or CLI.
In this example, SAS Studio will use the compute service to start a compute session.
The sas-viya CLI will use the batch service to start a batch job.



Services

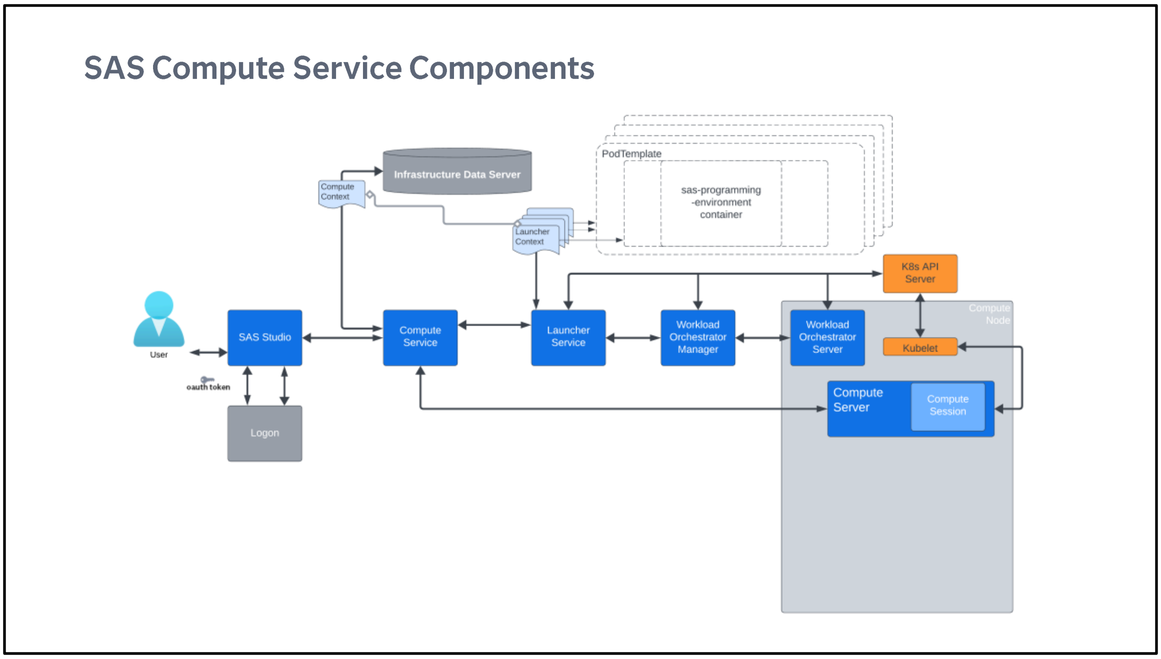
Here in the middle of the diagram we have the mid-tier services. These are services that provide the business functions to connect clients with backend servers. As an example, these services manage the backend configuration and settings by describing them in “contexts”, stored in the SAS Infrastructure Data server. These contexts are then used to allow the launcher service to launch a backend session or a job.

There are the compute service, the connect service and the connect spawner, and the batch service. They all talk to the launcher service.
The launcher service works together with the workload management components to talk to the Kubernetes API and launch the backend components.



Servers

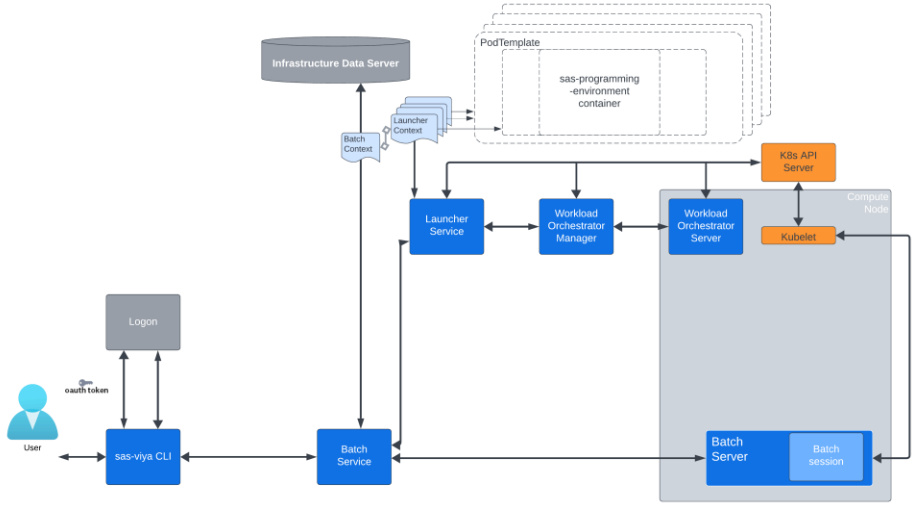
And, here on the right, we can see the backend components and servers.
There are the compute server, the connect server, the batch server.
They are all launched by the launcher service, via SAS workload orchestrator and the Kubernetes API.
They host a SAS session, which runs your SAS code. They are three different incarnations of the same engine, called the SAS Programming Runtime Engine.



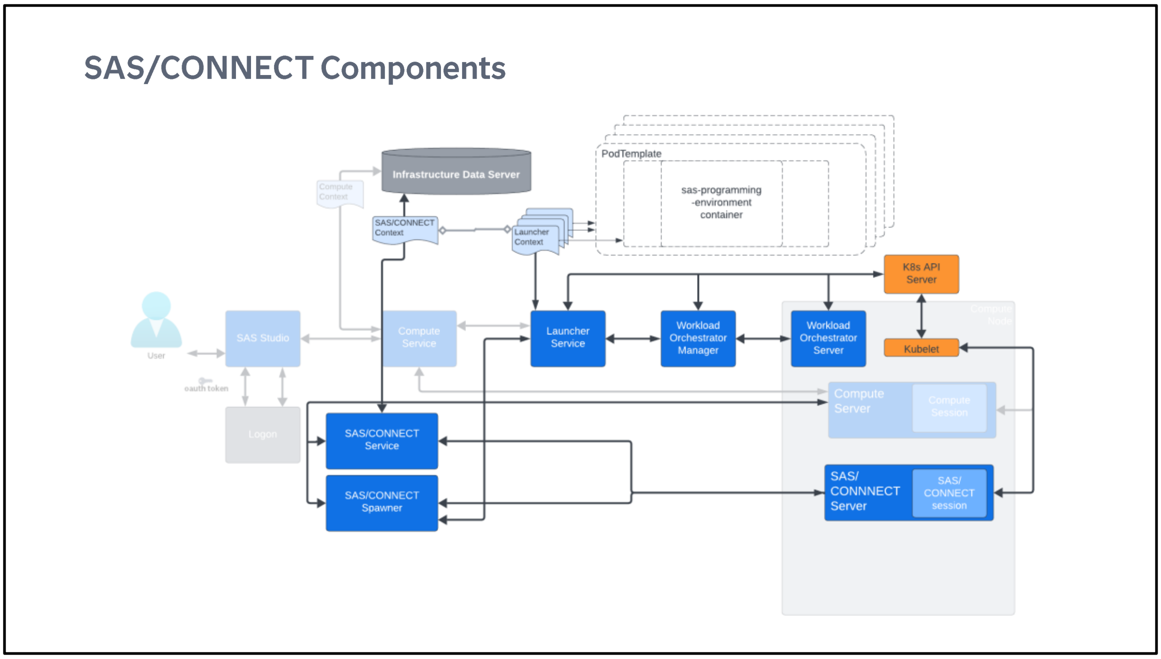
SAS Compute Service Components

If we shift our view from vertical columns to horizontal rows, we can group clients, mid-tier services, and backend servers in terms of functionality.
Here we highlight the SAS compute service components.
Moving from the left to the right, we can follow a user logging in SAS Studio and getting authenticated.
SAS Studio requests a compute session by talking to the compute service.
The compute service retrieves the compute context and the launcher context settings from the SAS Infrastructure Data Server. It can then request a new session from the launcher service.
The launcher service works together with the workload management components to talk to the Kubernetes API and start a backend compute server.
The compute server instantiates a user session which can interact with the frontend client, SAS Studio, through the compute service.
We can say that the compute service becomes a broker between the client and the backend session.

SAS Batch Components

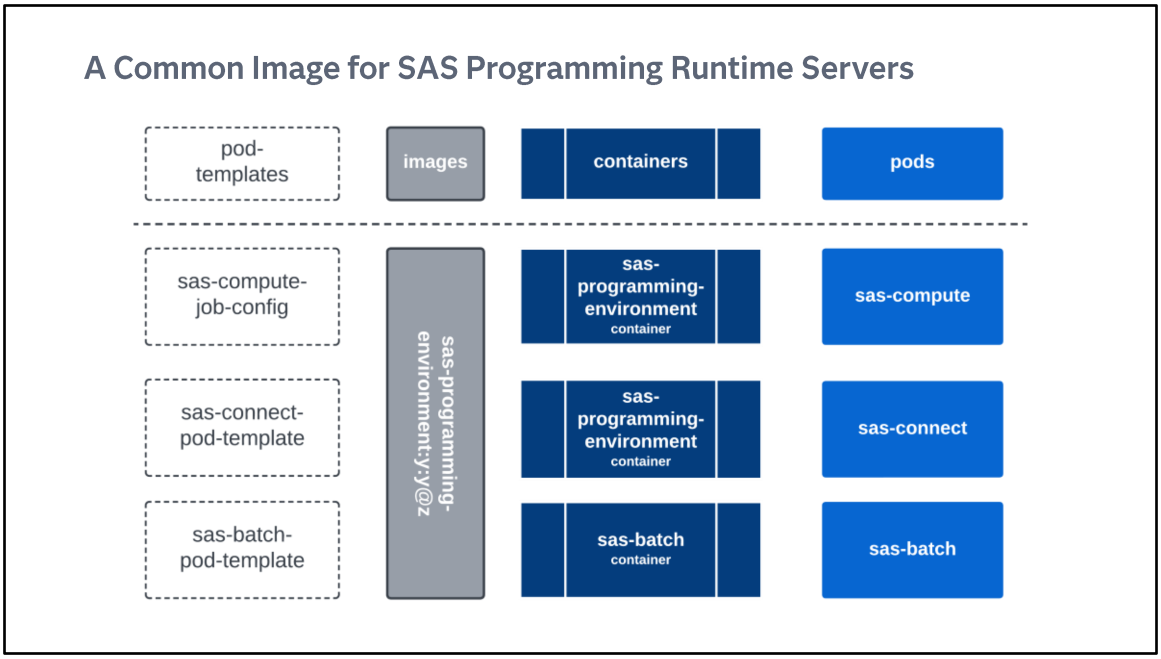


Similarly, if we look at the bottom row of the diagram, we can recognize the SAS batch components and describe a similar flow.
A user logs in, this time through the batch CLI, and is authenticated, as usual, through the SAS logon service.
The client talks to a mid-tier service, the batch service.
Again, the batch service is in charge of looking up the batch context and the launcher context, then passing all the information to the launcher service, with the request to start a new backend session.
This starts a batch server, via the workload management components and the Kubernetes API.
Again, the batch server communicates with the batch service, and the frontend CLI can send input and retrieve output from the batch service, including the job results.



SAS/CONNECT Components

Finally, the third instantiation of the compute service is SAS Connect.
You can see the SAS Connect components here in the middle row.
This time the client can be inside or outside the SAS Viya environment. It is not a frontend application but, usually, another SAS session.
In this example we left grayed out the compute server on the right to show that one was already running, probably started by SAS Studio, and now the compute server is our SAS Connect client.
As usual, the client talks to the middle tier services, and there are two different services that it can use.
It can choose to use the proprietary SAS Connect protocol and talk with the SAS Connect spawner, or use standard REST API interfaces with the SAS Connect service.
In both cases the middle tier services then talk to the launcher service to start the SAS Connect server in the backend, as always mediated by the workload management components and the Kubernetes API.
The backend server session and the SAS Connect client can either communicate via a direct connection, or proxied through the middle tier.

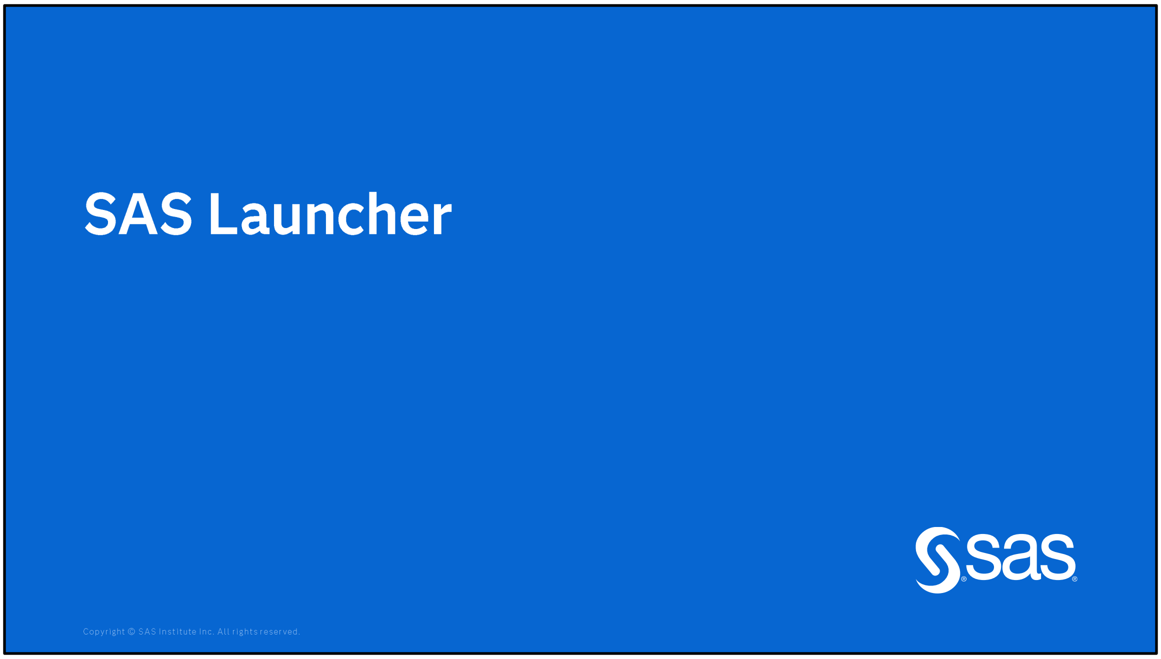


A Common Image for SAS Programming Runtime Servers

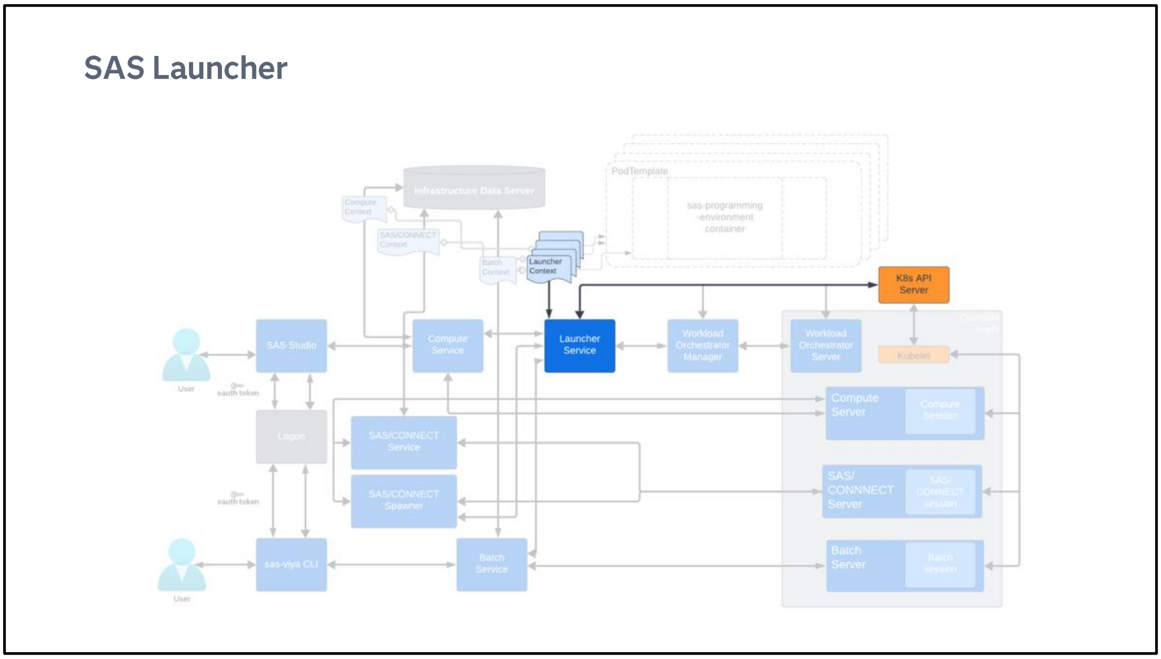
pod-templates	images	containers	pods
sas-compute-job-config	sas-programming-environment:y.y@z	sas-programming-environment container	sas-compute
sas-connect-pod-template		sas-programming-environment container	sas-connect
sas-batch-pod-template		sas-batch container	sas-batch

An interesting point of the previous components is that, although there are three different server incarnations, they all rely on a common container image that is instantiated in Kubernetes pods with different settings. SAS Viya uses Kubernetes pod templates to describe how to configure that common image into containers that run inside compute pods on Kubernetes. The common image, called SAS Programming Environment, provides SAS binaries to run SAS Compute servers, SAS Connect servers, and SAS Batch servers.

As an example, the sas-compute-job-config pod template references the sas-programming-environment image to build a container called SAS Programming Environment that runs inside a SAS Compute pod. SAS Connect and SAS Batch are similar, although the name of the containers can be slightly different. Pod templates contain information about how to instantiate the containers. As examples, how to set limits or resource requests on CPU and memory, how to mount storage volumes, for example for the SAS Work temporary area, and so on. It's all in the pod templates.

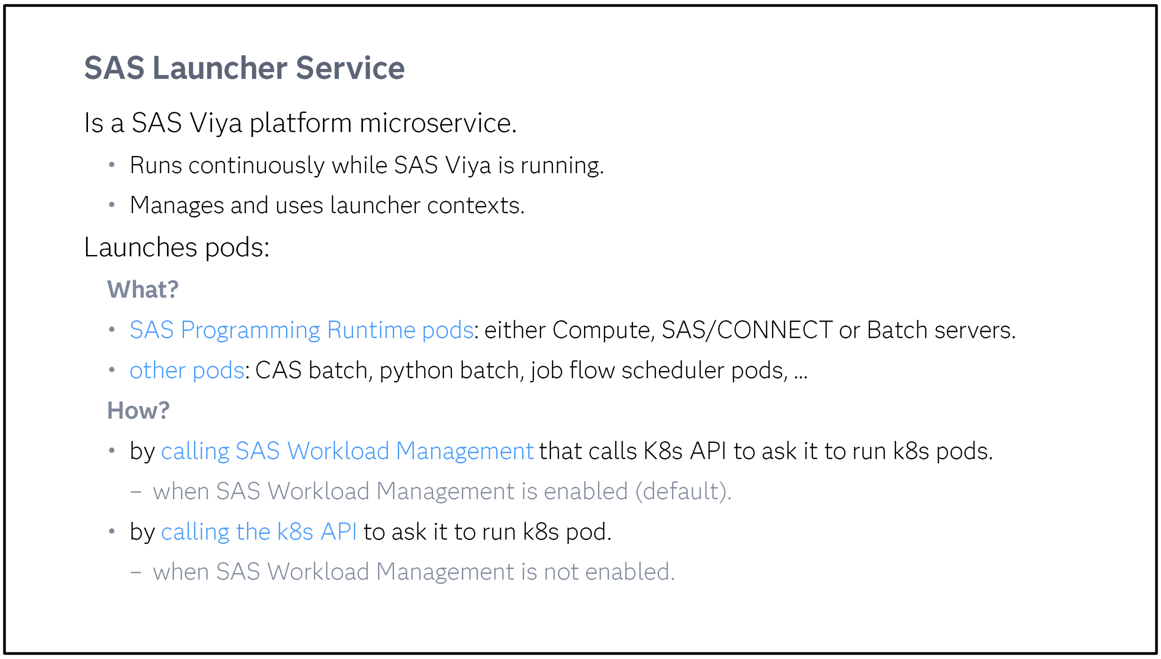


SAS Launcher



SAS Launcher

This lesson is part of a series that describes with additional detail each individual component of the programming runtime environment.
Here we focus on the SAS launcher.
Within the programming runtime environment, all services – whether it is a connect, a compute, or a batch service – they all talk to a gateway, and that is the launcher service.
In this diagram the launcher service sits in the middle of the picture. Notice that there two objects: a launcher service and a launcher context.



SAS Launcher Service

Is a SAS Viya platform microservice.

- Runs continuously while SAS Viya is running.
- Manages and uses launcher contexts.

Launches pods:

What?

- SAS Programming Runtime pods: either Compute, SAS/CONNECT or Batch servers.
- other pods: CAS batch, python batch, job flow scheduler pods, …

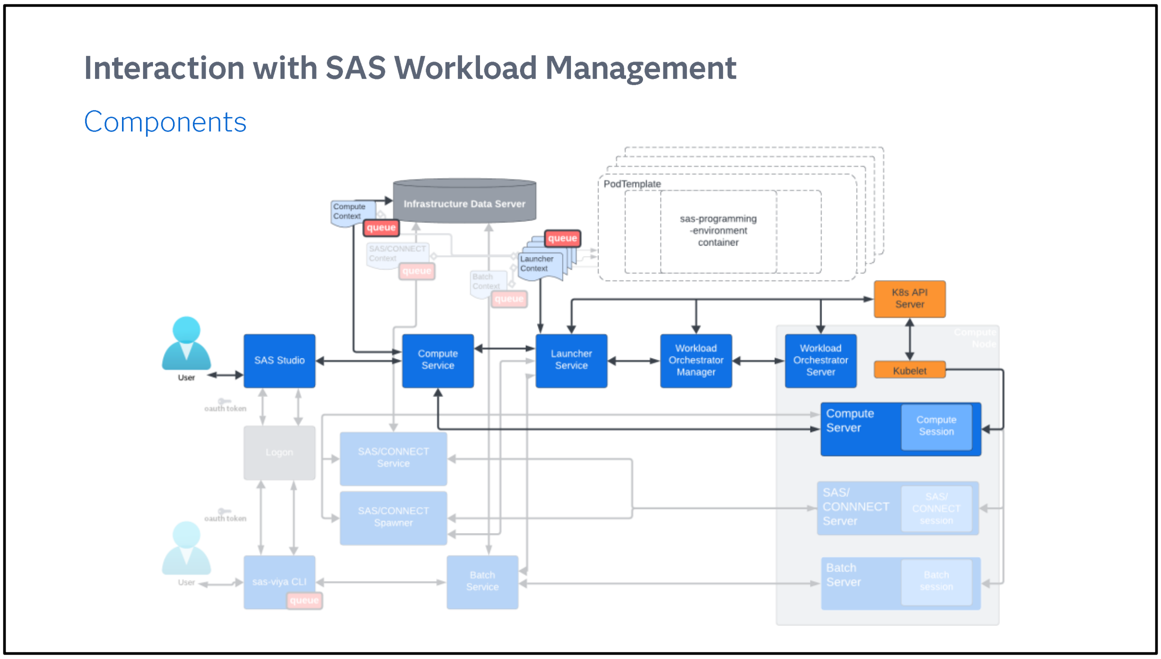
How?

- by calling SAS Workload Management that calls K8s API to ask it to run k8s pods.
  - when SAS Workload Management is enabled (default).
- by calling the k8s API to ask it to run k8s pod.
  - when SAS Workload Management is not enabled.

The launcher service is a SAS Viya microservice and, as such, it runs continuously while SAS Viya is running.
Its role is to manage launcher contexts and to launches pods.

What kind of pods?
It can launch the programming runtime pods, including compute pods, connect pods, or batch pods. As we have seen in other lessons, these are the ones that run the SAS code.
But it can also launch other pods.
It can launch a CAS batch server, it can launch Python batch pods, or job flow scheduler pods.
These are different kinds of pods that have a different pod template, and don't have a SAS engine inside of them, but they are still managed by the launcher service.

How does it work?
It depends on how the environment is configured.
With default deployments, SAS workload management is enabled. The launcher calls SAS workload management, which in turn calls the Kubernetes API to start the pods.
If instead you are using an older SAS Viya version, where SAS workload management was not included, or you have disabled workload management capabilities in your environment, then the launcher service will talk directly to Kubernetes.



Interaction with SAS Workload Management

Components

Here we have a view that shows how the launcher interacts with workload management in an example case, in which the SAS Studio client requires a backend compute server. The same would apply to a request for a connect or a batch server.

A user in SAS Studio selects a specific compute context, for example, context A. The compute context A is bound to launcher context A. The client sends this information to the compute service, which forwards it to the launcher service. The launcher service reads both the launcher context and the compute context, and uses them to build a specific environment, a specific configuration. With these it creates a full pod request and forwards it to SAS workload orchestrator, so that it can call the Kubernetes API to launch the compute server pod.

Many backend components talk to the Kubernetes API to know what's going on, but the command to start the backend pod comes from workload orchestrator.

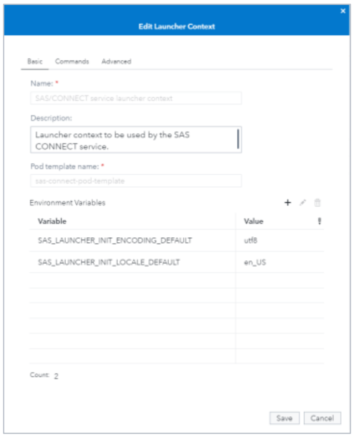
Queues are SAS Workload Management components that determine where the programming sessions/jobs can be executed. Queues also define the execution priorities.

Launcher Context

Specification that enables SAS administrators to apply environmental and access constraints on processes run by SAS Launcher Service.

Examples of things that can be defined in a launcher context:

- Environment variables
  - e.g., locale settings
- SAS Workload Orchestration queue



Name: * SAS/CONNECT service launcher context

Description: Launcher context to be used by the SAS CONNECT service.

Pod template name: * sas-connect-pod-template

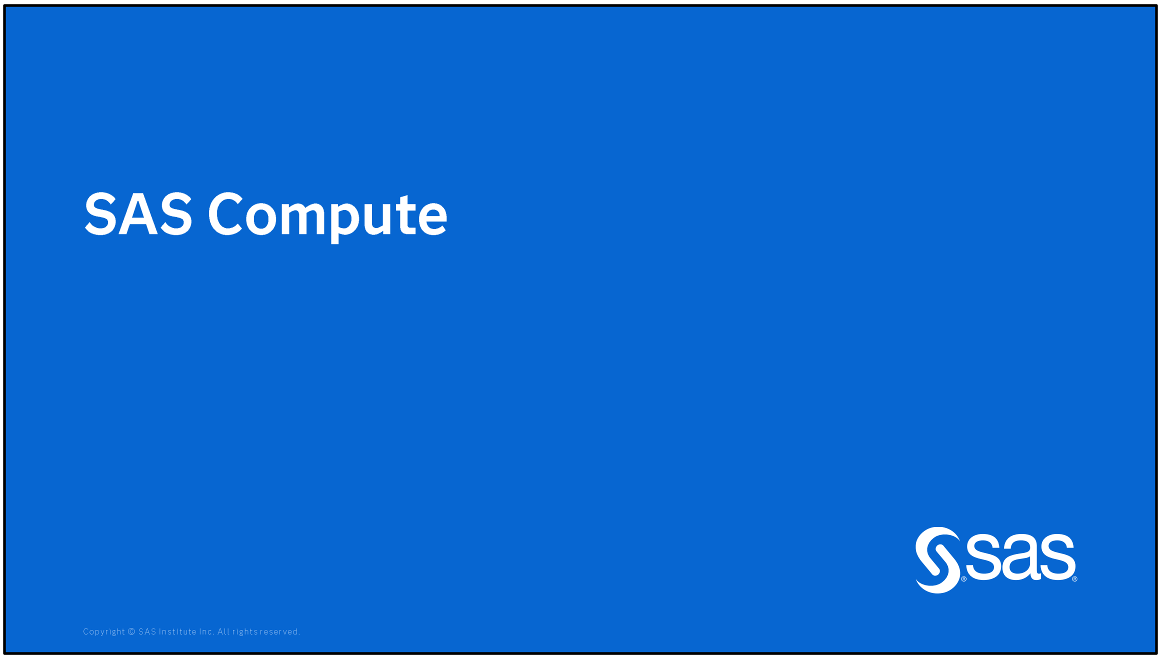
Environment Variables

Variable	Value
SAS_LAUNCHER_INIT_ENCODING_DEFAULT	utf8
SAS_LAUNCHER_INIT_LOCALE_DEFAULT	en_US

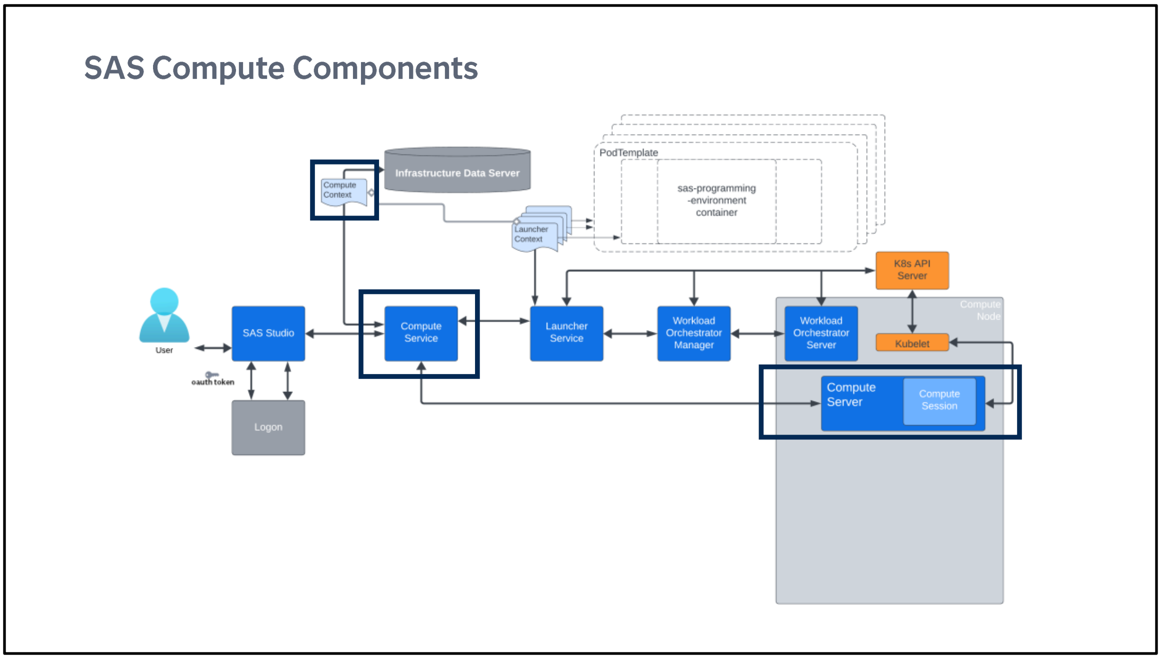
Count: 2

Since we mentioned contexts multiple time, let’s describe launcher contexts.
A context is a logical construct, it is a specification that lets SAS administrators group a specific set of configuration settings.
In detail, a launcher context controls the environment where the SAS backend will run.

As an example, an administrator can use a launcher context to define environment variables, or operating system library paths to load additional OS libraries.
A launcher context can optionally include a queue for SAS workload management.



SAS Compute

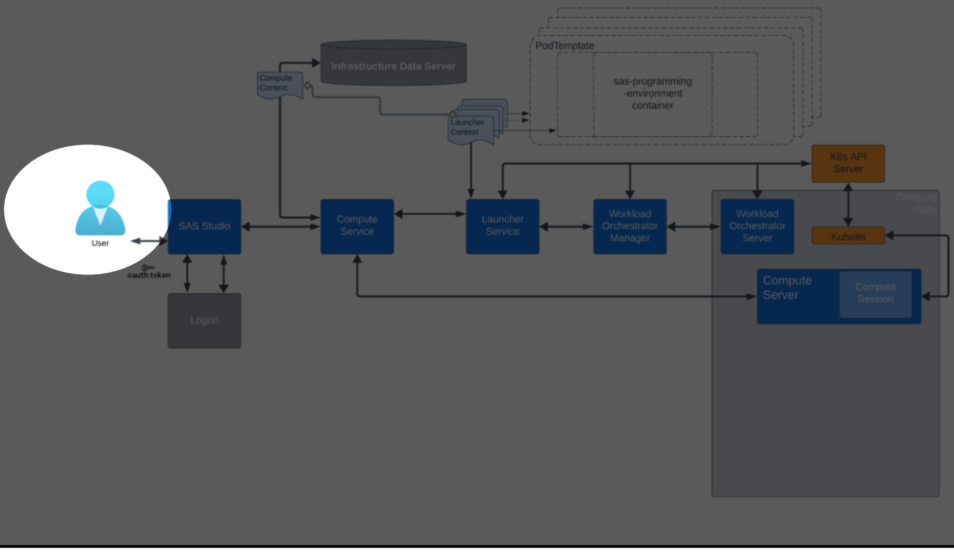


SAS Compute Components

We are continuing with the lessons that describe, with additional details, each individual component of the programming runtime environment.

Here we focus on the SAS Compute components.

SAS Compute Components



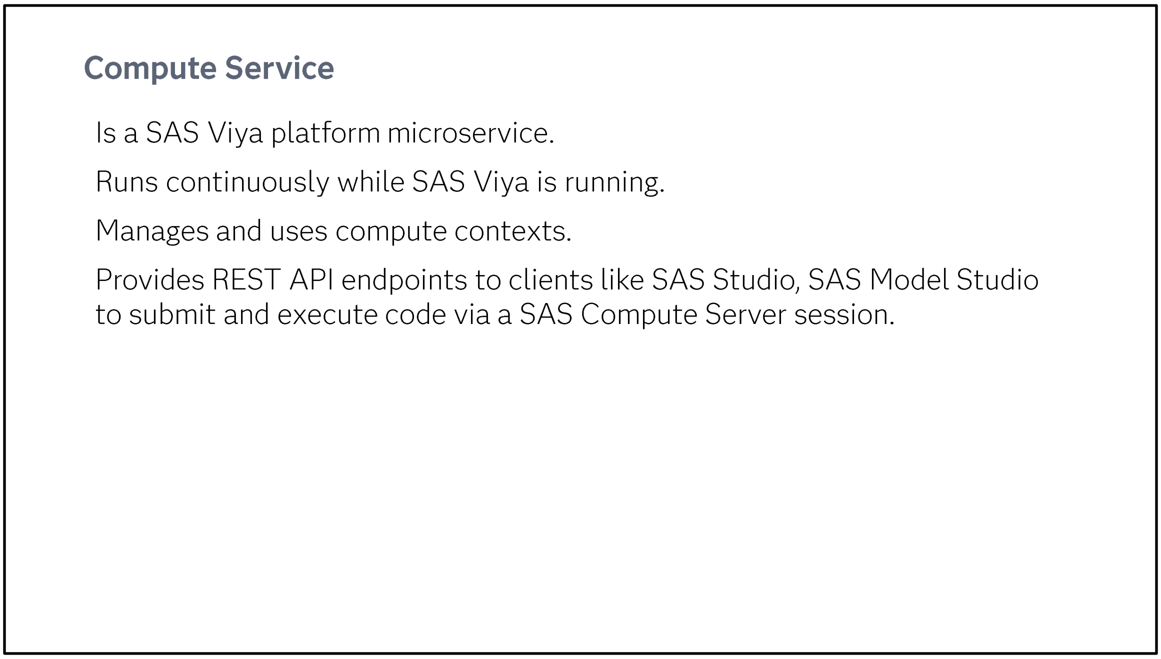
This diagram represents a sample use case.

A user logs into SAS studio, is authenticated.

The client requests a compute session by talking to the compute service.

The compute service looks at the compute context and the launcher context, then uses them to ask the launcher service to start a compute server via SAS Workload Management components.

The compute server starts a compute session and interacts, through the compute service, with the user.



Compute Service

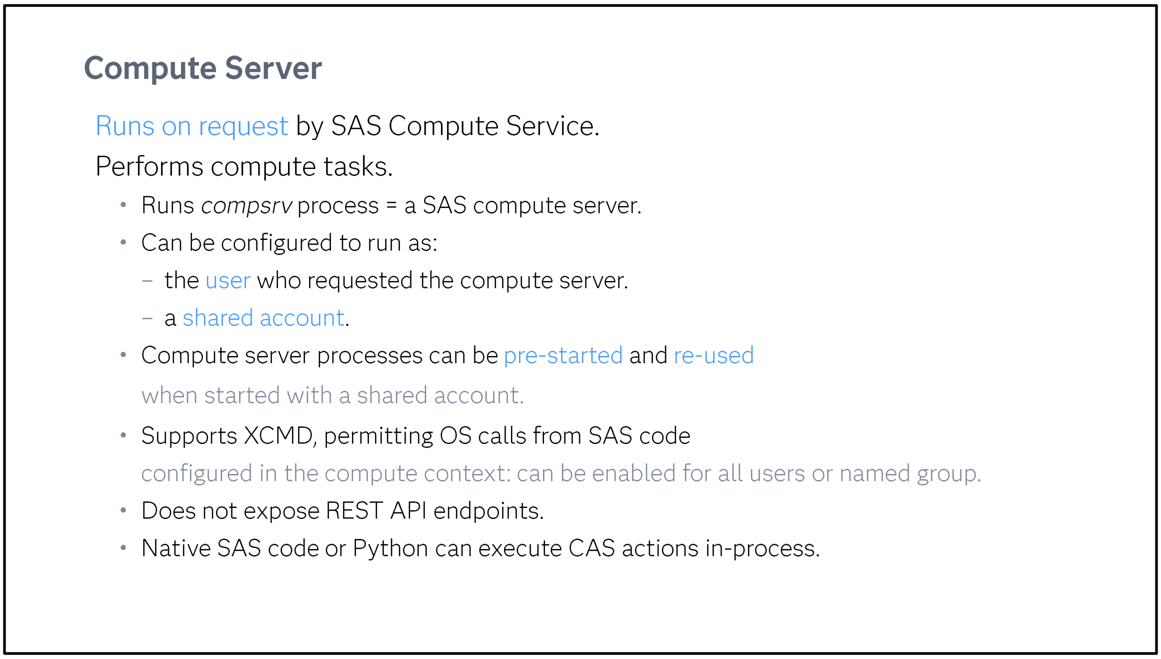
Is a SAS Viya platform microservice.

Runs continuously while SAS Viya is running.

Manages and uses compute contexts.

Provides REST API endpoints to clients like SAS Studio, SAS Model Studio to submit and execute code via a SAS Compute Server session.

The compute service is a SAS Viya microservice and, as such, it runs continuously while SAS Viya is running.
Its roles are to manage compute contexts, and to provide HTTPs API endpoints to frontend clients such as SAS Studio, SAS Model Studio and others.
The compute service acts as a broker to provide many capabilities, including:
Request the compute session start.
Pass SAS code to be executed.
Receive results and log messages.
Monitor the status of the compute sessions.
Pass all information to the client.
In summary, there is no direct communication between a client and a backend. It is always mediated by the compute service.



Compute Server

Runs on request by SAS Compute Service.

Performs compute tasks.

- Runs *compsrv* process = a SAS compute server.
- Can be configured to run as:
  - the user who requested the compute server.
  - a shared account.
- Compute server processes can be pre-started and re-used

  when started with a shared account.
- Supports XCMD, permitting OS calls from SAS code

  configured in the compute context: can be enabled for all users or named group.
- Does not expose REST API endpoints.
- Native SAS code or Python can execute CAS actions in-process.

The compute server is a backend pod that starts on request by the compute service and hosts a SAS compute server process.

It can either run under the identity of the user that requested it, or under a shared account defined by a SAS administrator.

Compute server pods can be started on-demand, to be stopped as soon as the client disconnects, or can be pre-started and re-used sequentially by multiple clients. In this case they must run under a shared account identity.

The usage of operating system calls and commands from SAS code is disabled by default, but can be enabled case by case in each compute context.

Compute servers do not expose any REST API endpoint, because, as we have seen in the previous slide, there is no direct communication between a client and the backend. It is always mediated by the compute service.

Starting in SAS Viya 2025.02, major enhancements to the SAS programming runtime engine enable advanced analytics procedures to run directly on the Compute Server. Native SAS code or Python can execute CAS actions in-process, eliminating the need for a CAS server in many cases.

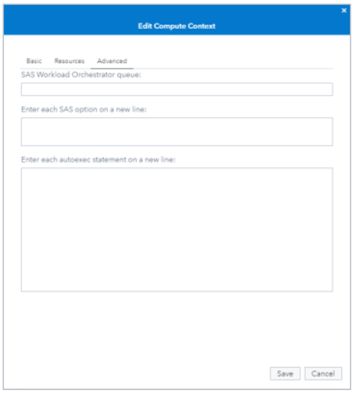
Compute Context

Compute servers are started with settings defined in **compute contexts.**

- like AppServer Contexts in SAS®9

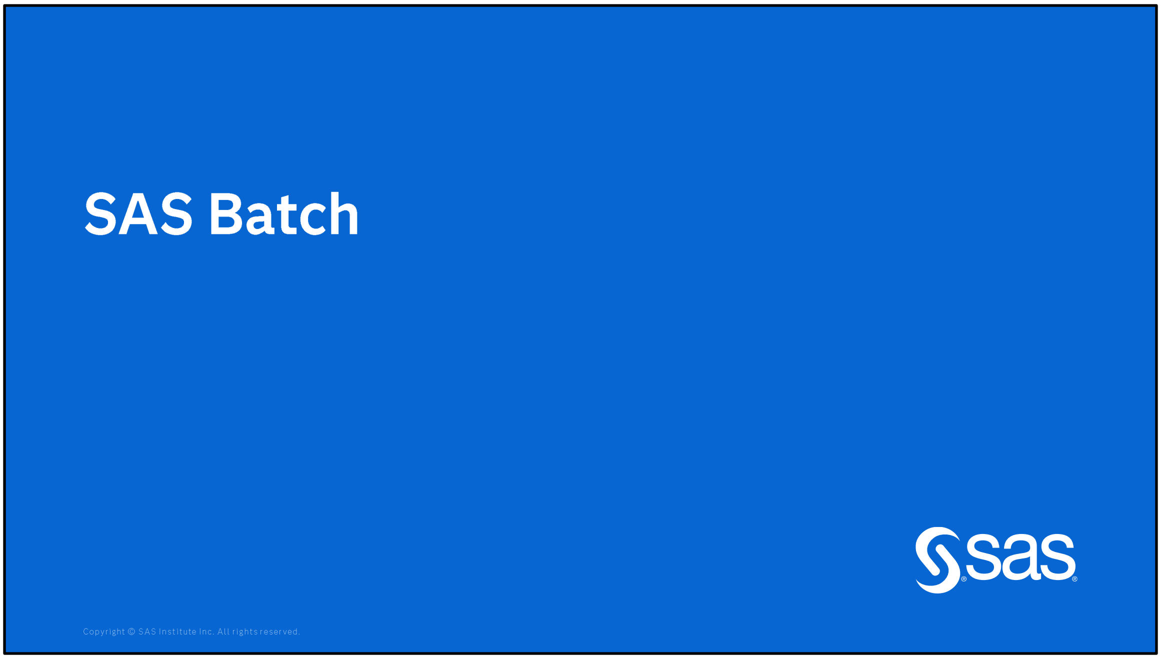
A compute context defines:

- Related launcher context.
- Identities (users or groups) who can use it (default is all Authenticated Users).
- Pre-allocated libraries.
- SAS Options.
- Autoexec code.
- Queue that will hold pods started by this context.
- Optional attributes that control how sessions are started and managed.

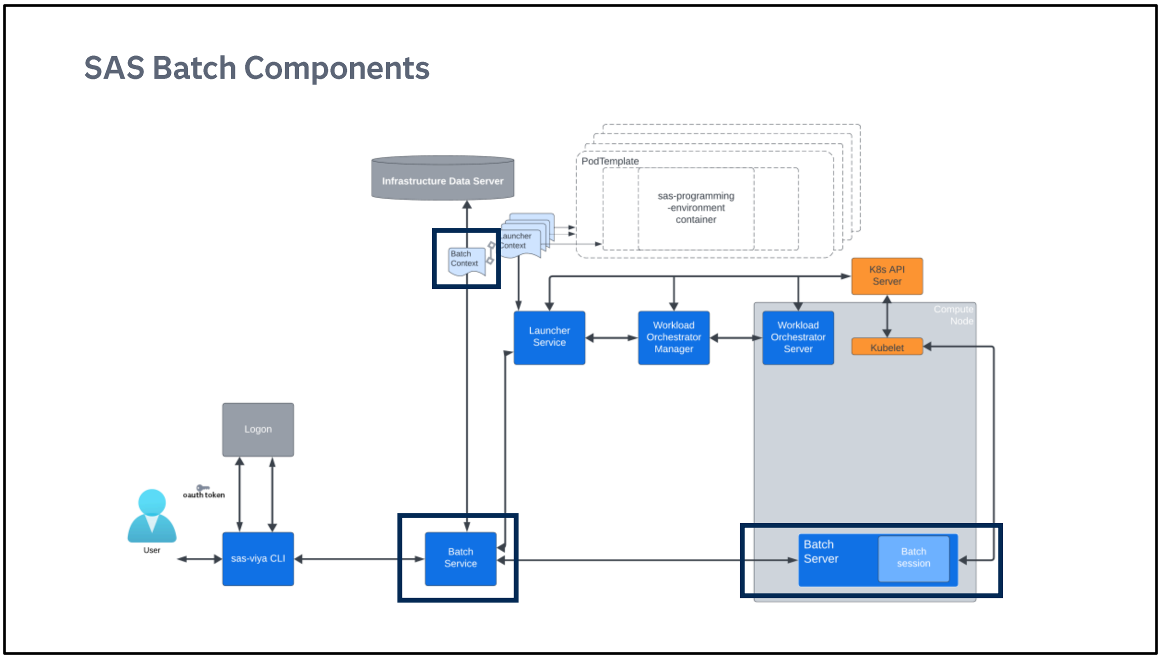

Edit Compute Context
×
Basic
Resources
Advanced
SAS Workload Orchestrator queue:
Enter each SAS option on a new line:
Enter each autoexec statement on a new line:
Save
Cancel

We have seen the compute service, then the compute server. Let’s now describe compute contexts. A compute context is a logical construct, to describe the settings used to start a compute server. If you are familiar with SAS 9 platforms, it’s an evolution of the concept of SAS application server contexts.

In detail, a compute context includes a related launcher contexts and defines which identities (users or groups) are allowed to use it (by default, that’s all authenticated users).
It also defines SAS options, autoexec code and pre-allocated libraries for the SAS server session that it will start.
A compute context can optionally include a queue used by SAS workload management and additional attributes.



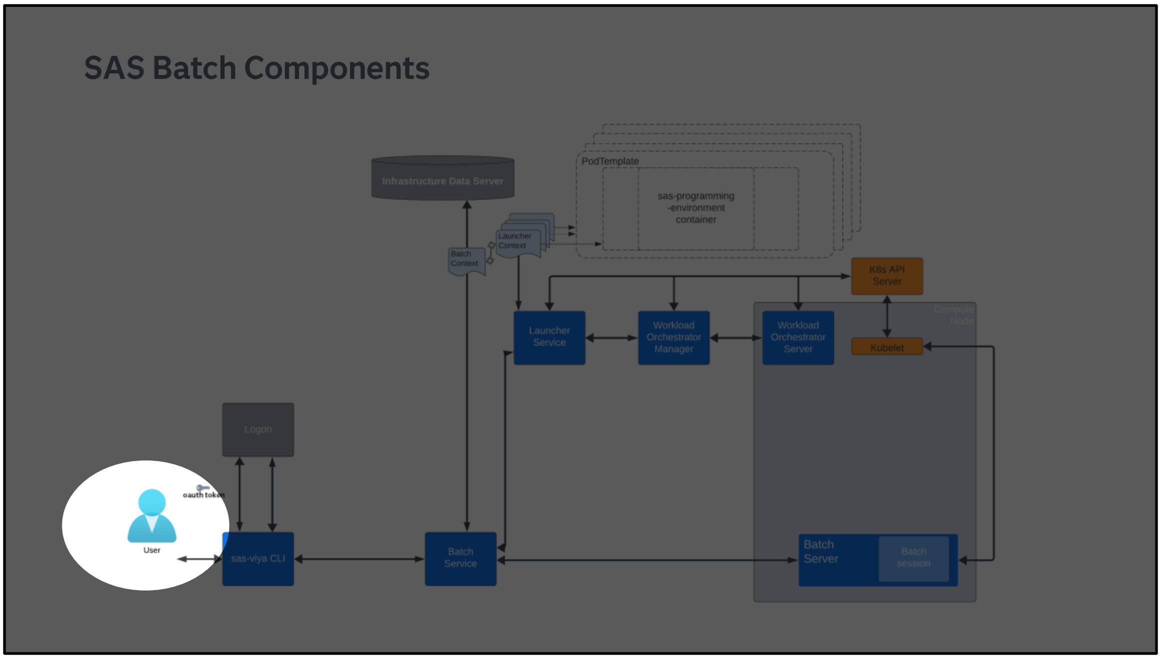
SAS Batch



SAS Batch Components

Let’s see another component of the programming runtime environment.

Here we focus on the SAS batch components.



SAS Batch Components

This diagram represents a default use case to execute long-running batch jobs.

A user, using the sas-viya CLI, after being authenticated, requests a batch job by talking to the SAS batch service.

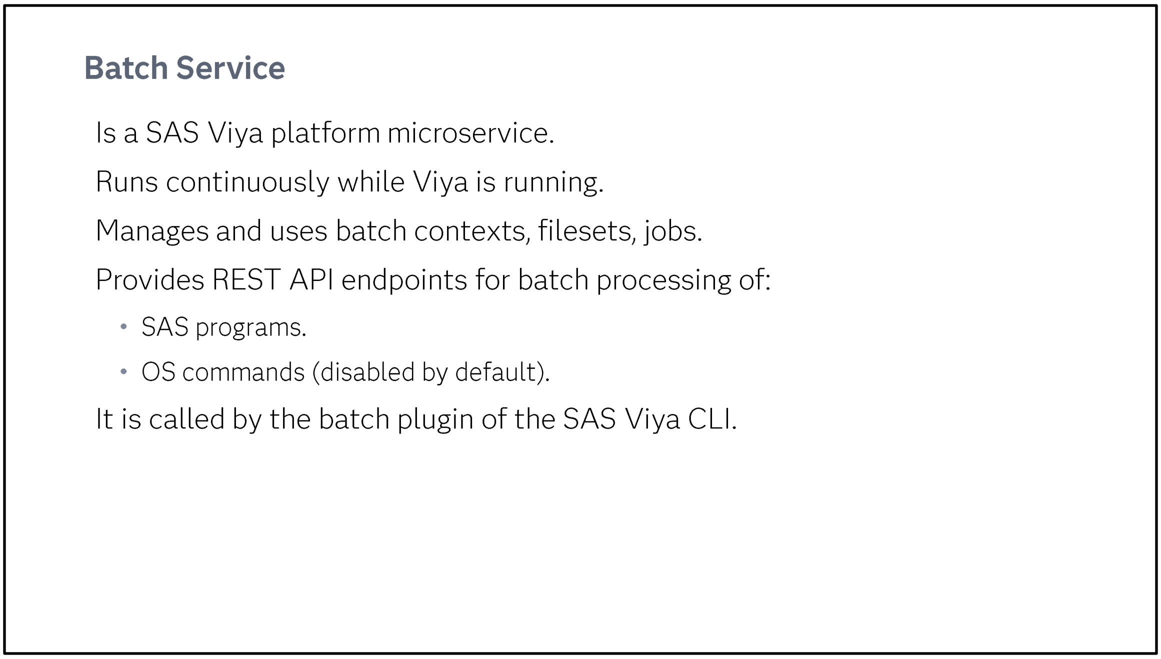
The SAS batch service loads the SAS code and associated files, and stores these as a file set into the SAS Infrastructure Data Server.
The SAS batch service looks
at the batch context and the launcher context, then uses them to ask the launcher service to start a SAS batch server via SAS workload management.

The SAS batch server starts and tells the SAS batch service that it is running, then reads the file set from SAS Infrastructure Data Server through the file service

The batch server executes the SAS code as a batch job, and interacts with the SAS batch service to provide it with the status of the job.

When the batch job is finished, the SAS batch server returns the results and log messages as a file set to the SAS batch service.

The user can operate synchronously or asynchronously with the SAS batch server. In the former case, logs and results are streamed back in real time. In the latter, the clients can connect to the batch service to request their download after the backend server is done.



Batch Service

Is a SAS Viya platform microservice.

Runs continuously while Viya is running.

Manages and uses batch contexts, filesets, jobs.

Provides REST API endpoints for batch processing of:

- SAS programs.
- OS commands (disabled by default).

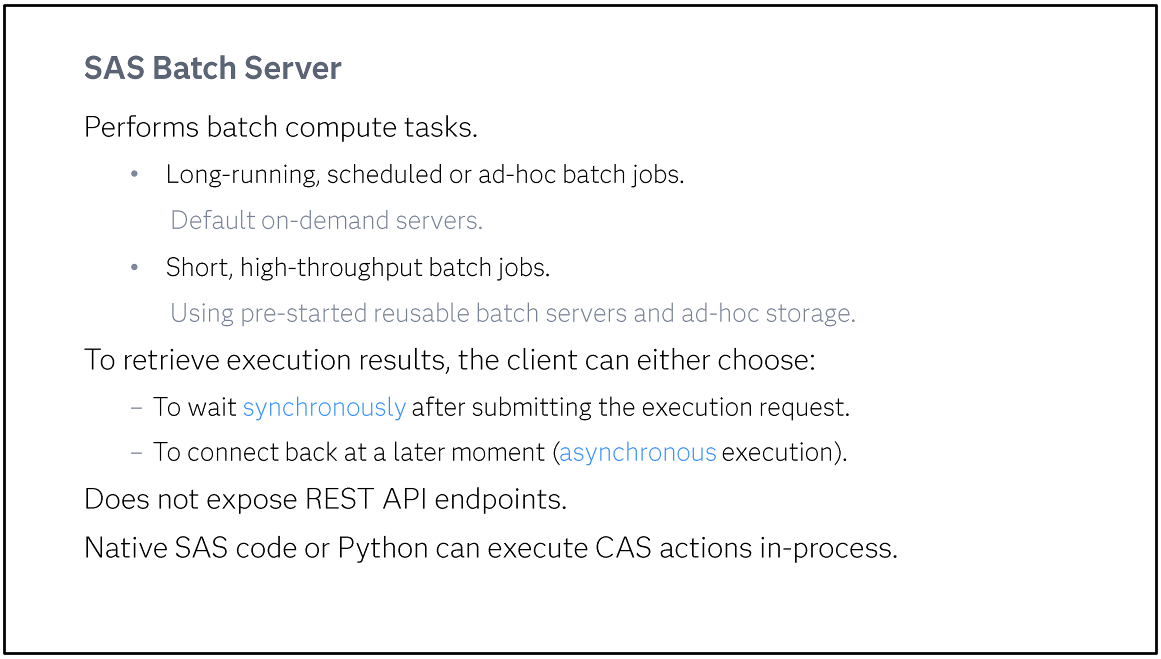
It is called by the batch plugin of the SAS Viya CLI.

The batch service is a SAS Viya microservice and, as such, it runs continuously while SAS Viya is running.

Its role is to manage batch contexts, filesets and jobs.

It provides HTTPS API endpoints to process SAS programs or operating systems commands, including Python scripts. For security reasons, the latter case is disabled by default; it can be enabled by an administrator.

The batch service has only one client, the SAS Batch CLI, which is a plug-in of the SAS Viya CLI.
If you are familiar with the sasgsub command-line utility used with SAS Grid Manager on SAS 9 platforms, you will recognize similar capabilities with SAS Batch.



SAS Batch Server

Performs batch compute tasks.

- Long-running, scheduled or ad-hoc batch jobs.

  Default on-demand servers.

- Short, high-throughput batch jobs.

  Using pre-started reusable batch servers and ad-hoc storage.

To retrieve execution results, the client can either choose:

- To wait synchronously after submitting the execution request.
- To connect back at a later moment (asynchronous execution).

Does not expose REST API endpoints.

Native SAS code or Python can execute CAS actions in-process.

The batch server is a backend pod that hosts a SAS batch server process.
Batch servers can execute batch jobs in two different ways, to accommodate different use cases.

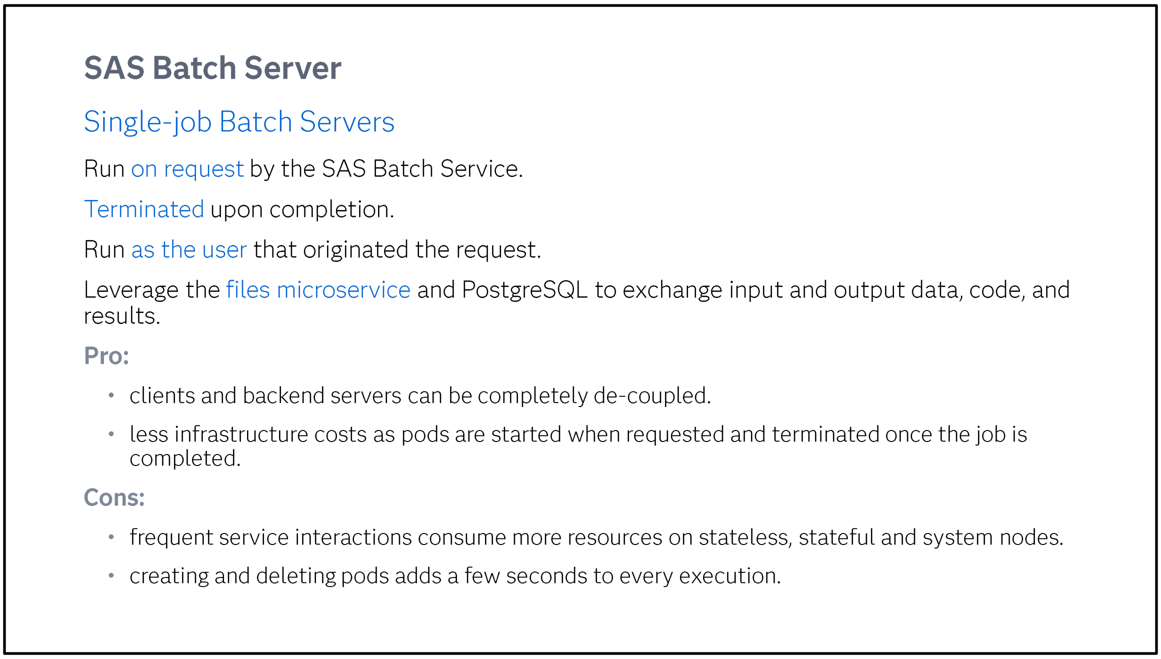
To execute long-running jobs, either scheduled or ad-hoc, batch server pods are started on demand when a request comes in from the batch service and shut down when the SAS program they are running completes. This is the default configuration.

To support the use case for short, high-throughput batch jobs, batch servers can be pre-started and utilize optimized input and output file storage. This new, ad-hoc configuration is available starting with the SAS Viya 2024.04 stable release or the SAS Viya 2024.09 long term support release.

In both cases, users can operate synchronously or asynchronously from the CLI. In the former case, logs and results are streamed back in real time. In the latter case, the client can disconnect, and later re-connect to the batch service to request their download, after the backend server is done.

Just as compute servers, batch servers, too, do not expose any REST API endpoint. Again, the reason is that there is no direct communication between a client and the backend. It is always mediated by the batch service.

Starting in SAS Viya 2025.02, major enhancements to the SAS programming runtime engine enable advanced analytics procedures to run directly on the Batch Server. Native SAS code or Python can execute CAS actions in-process, eliminating the need for a CAS server in many cases.



SAS Batch Server

Single-job Batch Servers

Run on request by the SAS Batch Service.

Terminated upon completion.

Run as the user that originated the request.

Leverage the files microservice and PostgreSQL to exchange input and output data, code, and results.

Pro:

- clients and backend servers can be completely de-coupled.
- less infrastructure costs as pods are started when requested and terminated once the job is completed.

Cons:

- frequent service interactions consume more resources on stateless, stateful and system nodes.
- creating and deleting pods adds a few seconds to every execution.

Let’s see some details about the configuration suitable for long-running jobs. This type of batch server is the default and is also called a single-job batch server.

In this case, batch server pods are created on demand, and start a new server process, each time a request comes in from the batch service.

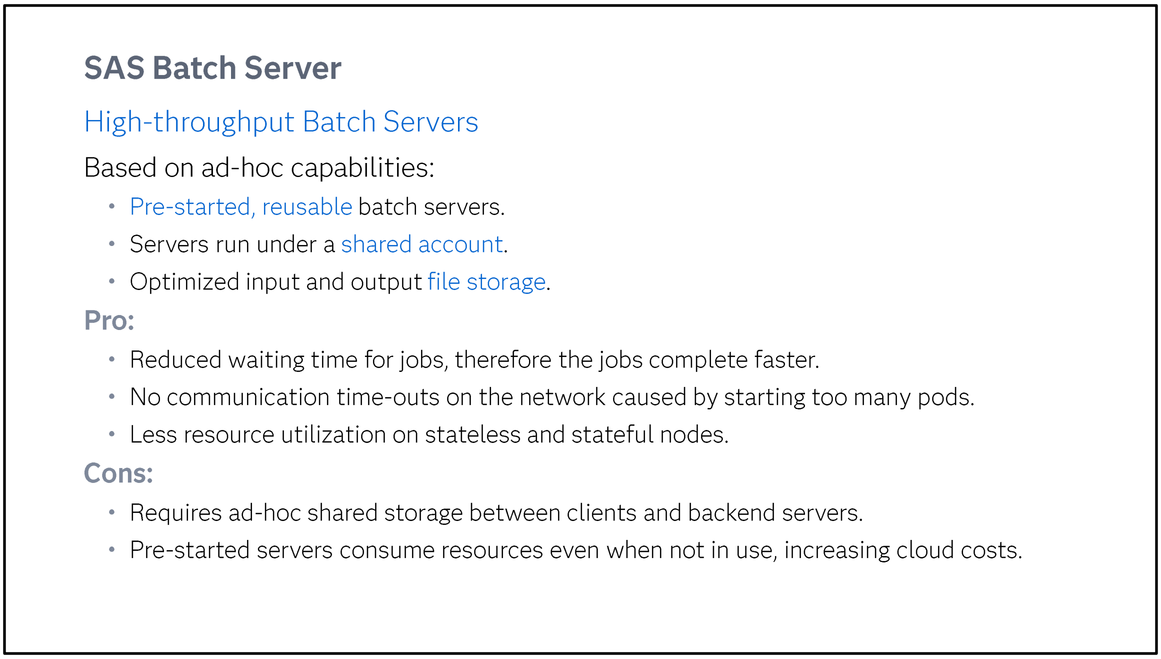
Each server terminates when the SAS program they are running completes, and the corresponding pod is removed.

A batch server retrieves the user identity from the incoming request, and runs as the user that originated it.

A batch server reads the program code and any other input files from the files microservice. It runs whatever code it was asked to run, and then it sends the results back to the files microservice. The files microservice leverages the SAS Infrastructure Dataserver, that is, PostgreSQL, to store and retrieve both file contents and metadata.

This functional architecture permits to completely de-couple clients and backend servers, which is often required for batch processing.
Single-job batch servers reduce infrastructure costs, as pods are started only when requested and terminated as soon as the job is completed.

To implement this separation, there are more and longer calls with and between mid-tier components, such as the files service. Plus, pods get created and deleted for every incoming job. All of this adds an overhead of a few seconds to each execution and increases resource consumption, on SAS nodes, but also on Kubernetes system nodes. In fact, creating and destroying many pods in a short timeframe consumes Kubernetes daemons resources, and can cause communication timeouts.



SAS Batch Server

High-throughput Batch Servers

Based on ad-hoc capabilities:

- Pre-started, reusable batch servers.
- Servers run under a shared account.
- Optimized input and output file storage.

Pro:

- Reduced waiting time for jobs, therefore the jobs complete faster.
- No communication time-outs on the network caused by starting too many pods.
- Less resource utilization on stateless and stateful nodes.

Cons:

- Requires ad-hoc shared storage between clients and backend servers.
- Pre-started servers consume resources even when not in use, increasing cloud costs.

SAS Batch servers can be configured as reusable batch servers to process multiple jobs. This type of batch server is suitable for short-running jobs that require high throughput.

High-throughput batch servers require proper architecture design and environment configuration. They leverage different capabilities.

A reusable batch server is a batch server pod that can process multiple jobs sequentially, without stopping and restarting backend pods. These servers decouple jobs from pod creation and initialization. A configurable number of reusable batch servers is usually pre-started at SAS Viya startup.

Pre-started batch servers run under a shared account that has been configured by a SAS administrator.

Optimized Input and output file storage requires shared storage accessible by batch clients and batch servers alike. Users can store input data and programs on that storage. Batch servers can read them in, and then store in the same location any output files and logs, immediately available to be retrieved by users. No interaction with any middle tier service is required.

This functional architecture permits reduced waiting time for jobs, that do not have to wait for pods to start, nor for files to be uploaded and downloaded. Therefore, the jobs complete faster.
Pre-starting pods avoids continuous calls to the Kubernetes API, which frees system resources and avoids communication time-outs.
Less frequent calls between SAS Viya services reduce resource utilization on stateless nodes, plus, by not using the files service, saves resources on PostgreSQL.

Those benefit come with some additional considerations.

This architecture is more complex and requires ad-hoc shared storage between clients and backend servers.
Pre-started servers consume resources even when not in use, causing extra costs, especially on cloud environments.

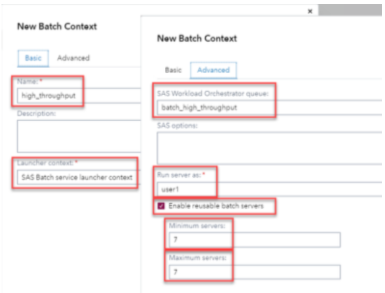
Batch Context

A **batch context** is a specification that contains the information to run a batch server

- Related launcher context
- SAS options
- SAS Workload Orchestration queue
- Runs server as, reusable server settings

Two contexts are defined by default:

- **default**
  - used for running SAS programs
- **default-cmd**
  - used for running arbitrary commands such as Python programs (disabled by default)



New Batch Context

Name: * high_throughput

Description:

Launcher context: * SAS Batch service launcher context

New Batch Context

SAS Workload Orchestrator queue: batch_high_throughput

SAS options:

Run server as: * user1

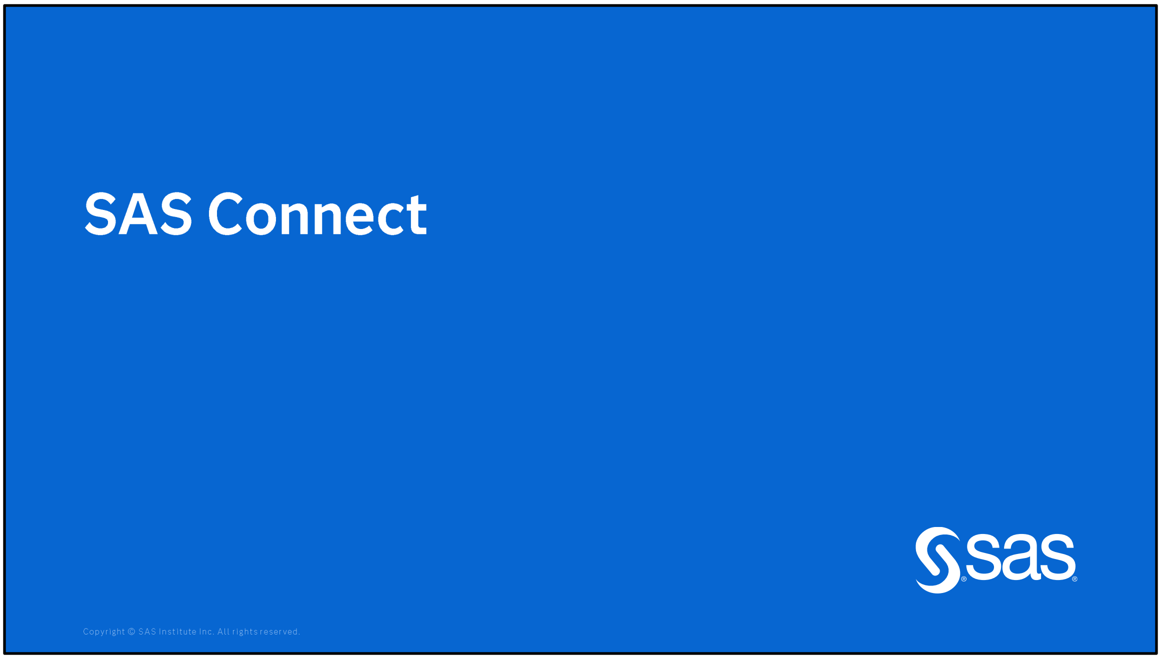
☑ Enable reusable batch servers

Minimum servers: 7

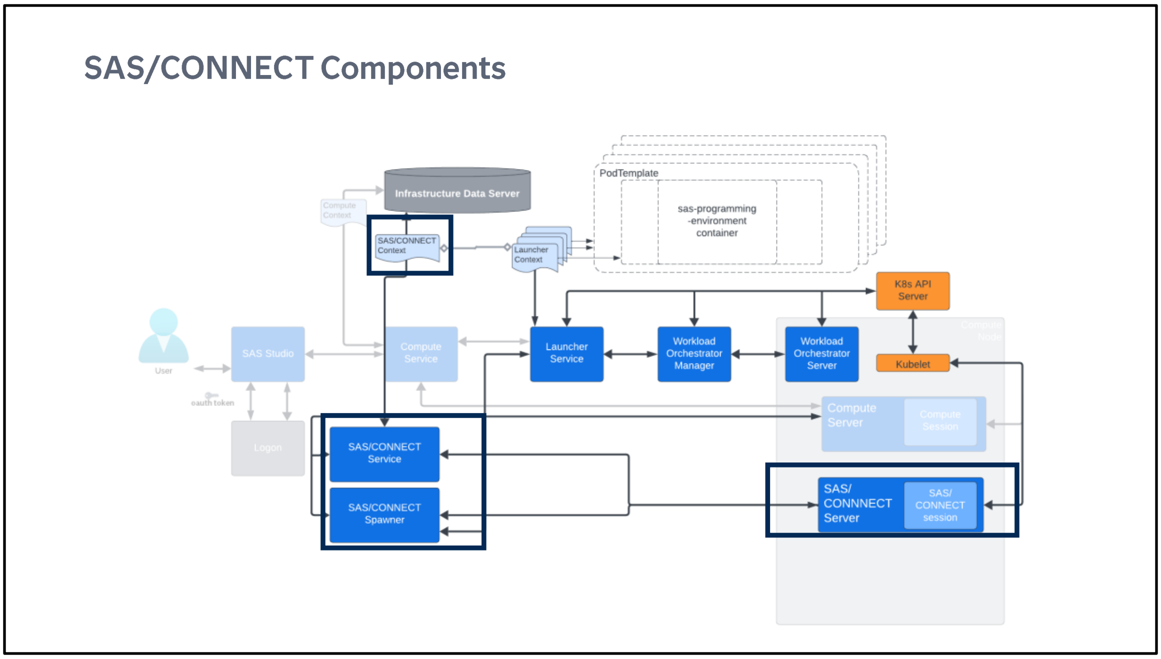
Maximum servers: 7

Let’s move on and review batch contexts.
A batch context is, again, a logical configuration that describes how to run a backend batch server. It includes a related launcher contexts, and, optionally, SAS Options, a SAS workload management queue, runs server as, and reusable server settings.
Multiple batch contexts can share an individual launcher context, but each batch can only link one launcher context.

By default, out of the box, there are two batch contexts defined: one that is used to run SAS programs and one that is used to run Python or arbitrary commands.
The latter is pre-configured to run Python, but, for security considerations, it is disabled by default. A SAS administrator can enable it when required.



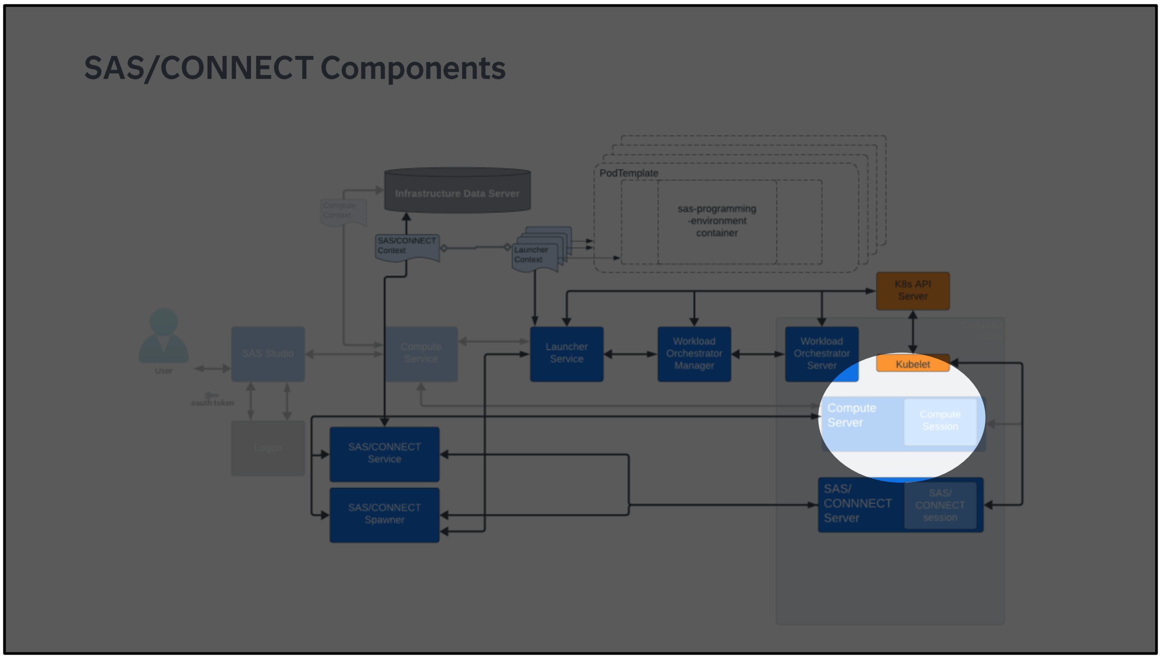
SAS Connect



SAS/CONNECT Components

Let’s see another component of the programming runtime environment.

Here we focus on the SAS connect components.



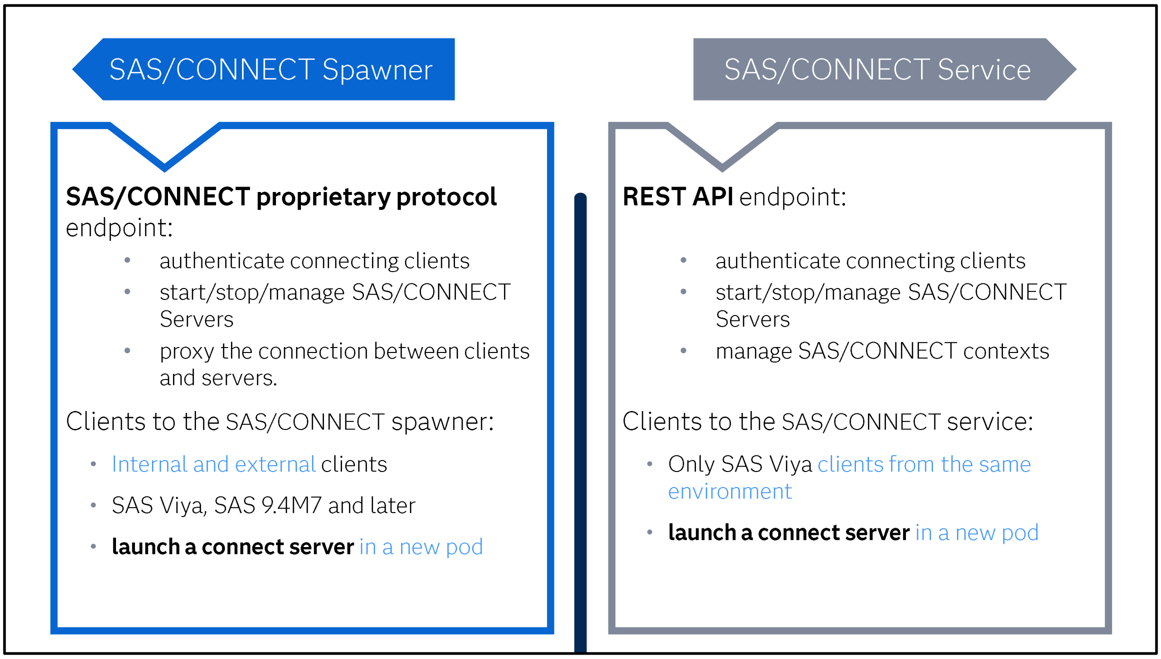
SAS/CONNECT Components

Just as we have seen in other lessons, this diagram represents a sample use case.
With SAS connect, the client is not a frontend application, but rather, it is another SAS session.
SAS connect clients can be inside or outside SAS Viya deployment cluster and namespace.

In this example, a compute server on the right is greyed out, to show that it was previously started by a frontend client, for example SAS Studio, but now the compute server itself acts as client to SAS connect components, using the SAS connect protocol.

You can see on the lower left the SAS connect middle-tier services, both the SAS Connect service, and the SAS Connect spawner.
They handle client authentication, and read the SAS connect context and the launcher context, then use the information from these contexts to ask the launcher service to start a backend SAS connect server.
As usual the launcher service works with SAS Workload management.

Finally, on the lower right there is the SAS connect server. The server runs a SAS session that interacts either directly or through the SAS connect spawner with the client that initiated the connection.



SAS/CONNECT Spawner

SAS/CONNECT proprietary protocol endpoint:

- authenticate connecting clients
- start/stop/manage SAS/CONNECT Servers
- proxy the connection between clients and servers.

Clients to the SAS/CONNECT spawner:

- Internal and external clients
- SAS Viya, SAS 9.4M7 and later
- **launch a connect server** in a new pod

SAS/CONNECT Service

REST API endpoint:

- authenticate connecting clients
- start/stop/manage SAS/CONNECT Servers
- manage SAS/CONNECT contexts

Clients to the SAS/CONNECT service:

- Only SAS Viya clients from the same environment
- **launch a connect server** in a new pod

As you may have noticed, a peculiarity of SAS connect is that with SAS Viya there are two different middle tier services.
The first one, the SAS Connect Spawner, is the evolution of the same service that has been available with SAS for decades, across multiple SAS versions, including SAS 8, SAS 9 and SAS Viya 3.
In the current version of SAS Viya it runs in a Kubernetes pod, just as any other component.

The SAS connect spawner provides an endpoint to the proprietary SAS Connect protocol. It does not have any REST API endpoints.

For that, there is the SAS connect service, which is a brand-new component first used with SAS Viya.
So, the SAS connect spawner uses a proprietary SAS protocol. The SAS connect service uses a standard HTTPS REST protocol.

Both services authenticate clients, start, stop, manage the backend servers.
The SAS connect spawner also acts as a proxy between each client and server, maintaining the communication line. Instead, when using the SAS connect service, clients and servers directly talk, once the connection is set up.

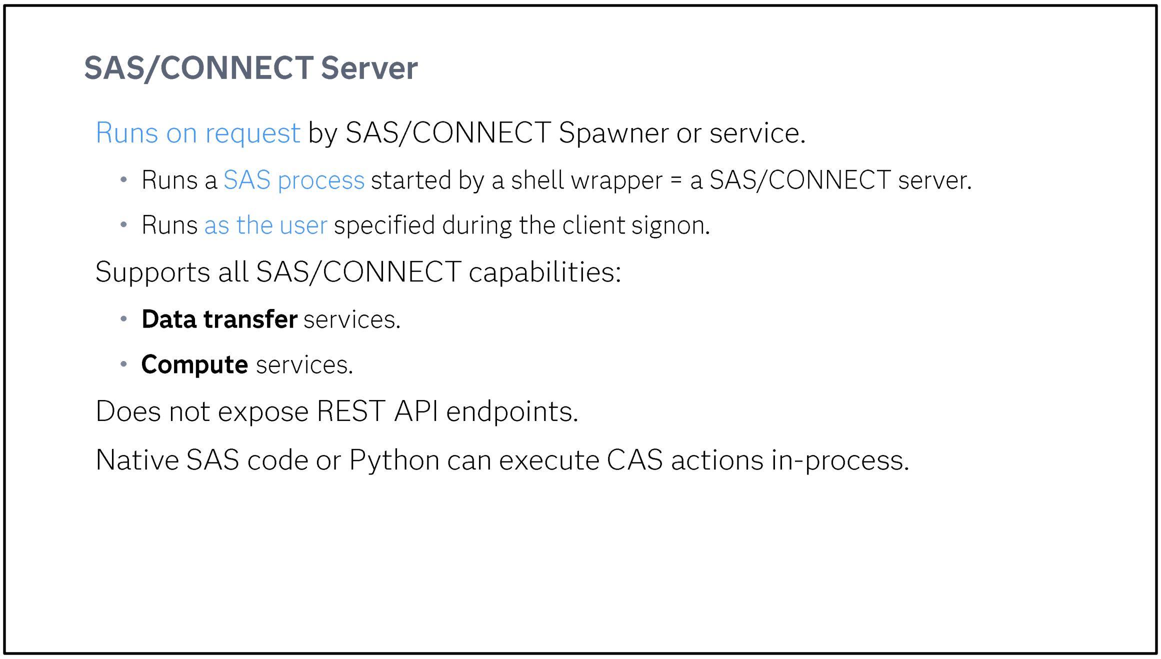
That’s because the SAS connect spawner can accept incoming requests from clients that sit outside the SAS Viya Kubernetes cluster, for example an on-prem SAS 9 environment. These clients have no way to talk to a SAS connect server except than proxied by the spawner.

The SAS connect service, on the contrary, can only service clients co-located in its environment.

Both services by default launch a backend connect server by starting a new pod.

If you are familiar with previous versions of the SAS/CONNECT software, you might be aware that the SAS connect spawner used to support a legacy way in which a connect server session could be started inside the spawner pod. This legacy capability was deprecated in the previous LTS release, and has been removed, starting with SAS Viya 2024.09.

In summary the sas connect spawner provides the best compatibility for external environments and migrated code, while the sas connect service is a modern microservice for SAS Viya clients.
Just like the compute service and the batch service, the connect service is the one that manages contexts.



SAS/CONNECT Server

Runs on request by SAS/CONNECT Spawner or service.

- Runs a SAS process started by a shell wrapper = a SAS/CONNECT server.
- Runs as the user specified during the client signon.

Supports all SAS/CONNECT capabilities:

- **Data transfer** services.
- **Compute** services.

Does not expose REST API endpoints.

Native SAS code or Python can execute CAS actions in-process.

The connect server is a backend pod that hosts a SAS connect server process.
Connect server pods are started on demand when a request comes in from the connect spawner or service. They run as the user identity specified in the client connection request, and run until the client issues a SIGNOFF command.

SAS Viya connect servers support all of SAS/CONNECT capabilities that have been available for years with previous SAS versions, including data transfer services and compute services.

Data transfer services enable you to move a copy of your data from one computer to another computer. The data is translated between computer architectures and SAS version formats, as necessary.
Compute services include remote code execution using RSUBMIT code blocks that enable directing the execution of SAS programs to one or more remote sessions. They also provide Remote SQL Pass-Through and Remote Library Services.

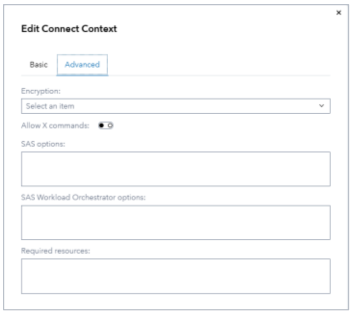
Just as the compute and the batch backend servers, they do not expose any REST APIs.

Starting in SAS Viya 2025.02, major enhancements to the SAS programming runtime engine enable advanced analytics procedures to run directly on the Connect Server. Native SAS code or Python can execute CAS actions in-process, eliminating the need for a CAS server in many cases.

SAS/CONNECT Context

A **connect context** provides information for a SAS/CONNECT client:

- how to use a **launcher**
  - related launcher context
  - SAS options
  - SAS Workload Orchestration queue and options
  - required resources to be used when starting the server
  - encryption type
- how to connect to a **spawner** (legacy)
  - host and port
  - encryption type



Edit Connect Context

Encryption: Select an item

Allow X commands: ●○

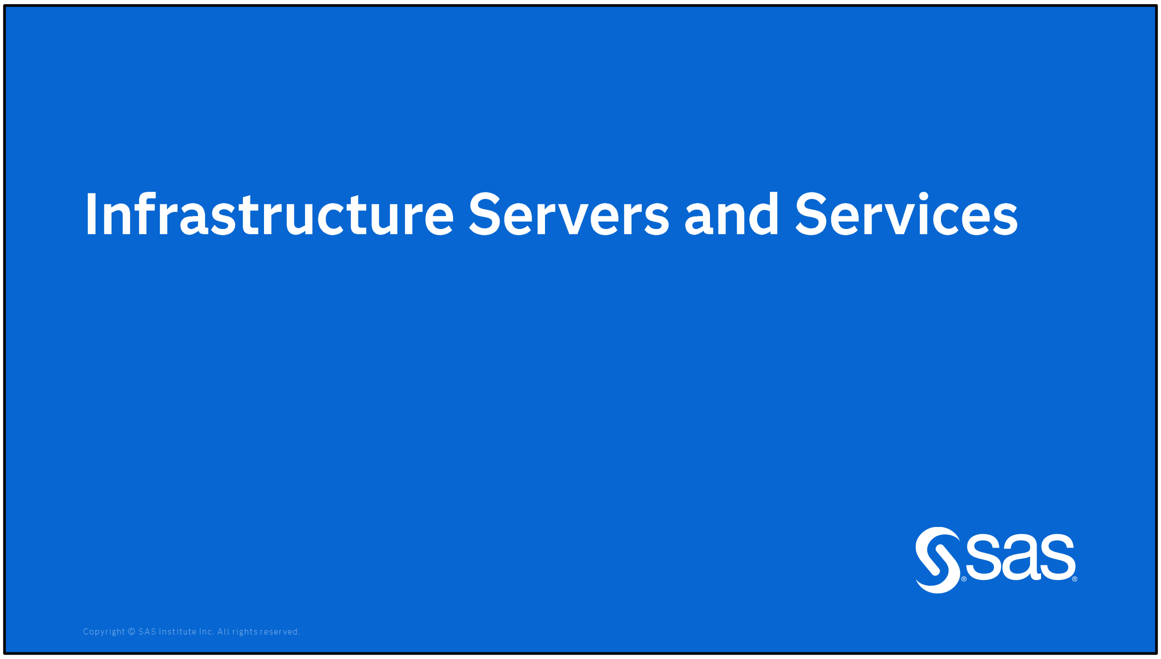
SAS options:

SAS Workload Orchestrator options:

Required resources:

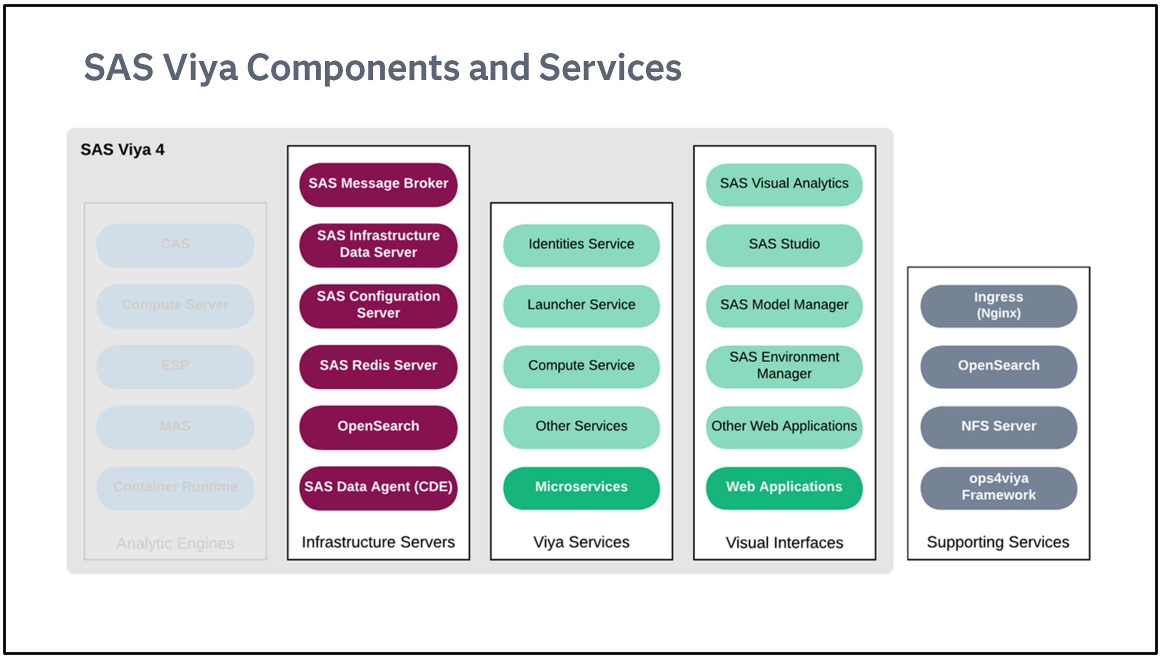
A SAS connect context is similar to compute and batch contexts, in that it provides information on how to run the backend server.
A SAS connect context can provide information for a SAS/CONNECT client to connect to a spawner or to provide information for the client to use a launcher to start a CONNECT server in its own pod.
That includes a related launcher context, and, optionally, SAS Options, a SAS workload management queue and SAS workload management options, and resources to be used.
Multiple connect contexts can share a single launcher context, but each connect service can only link one launcher context.

3.3 Other Components



Infrastructure Servers and Services

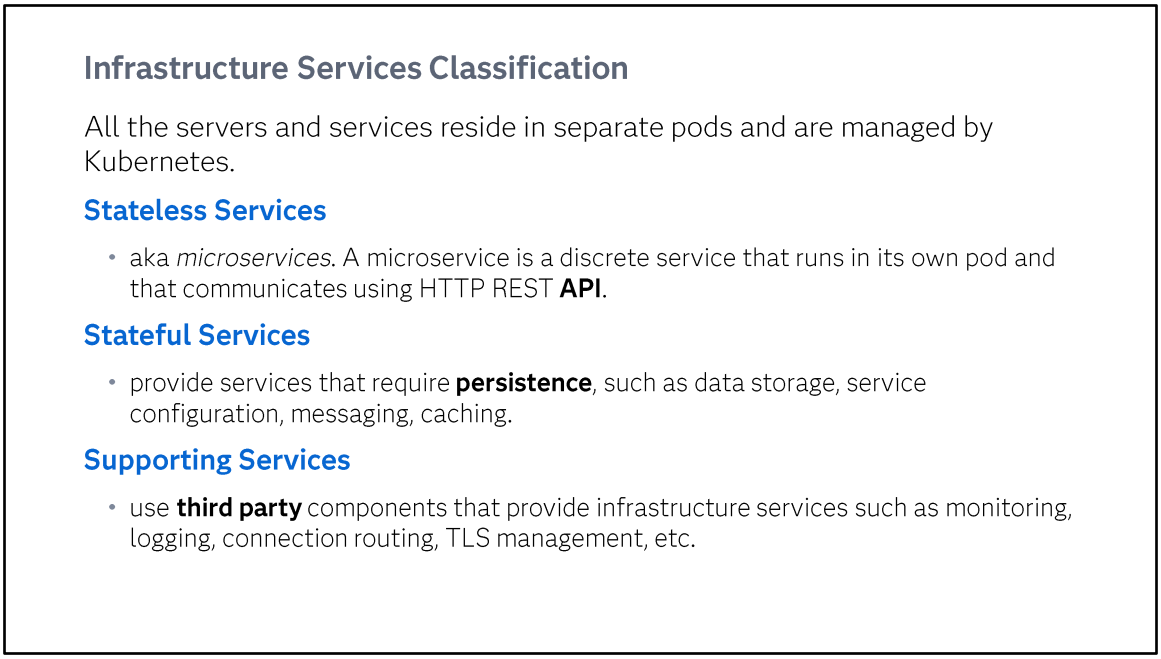
This lesson presents a classification of infrastructure servers and services and describes a high-level view of their architectural characteristics. Additional details are covered in other lessons.



SAS Viya Components and Services

In this architectural view, we can see a general classification of the main components and services, including analytic engines, infrastructure servers, SAS Viya services and web applications, and supporting services for the platform.

We covered the engines in other lessons, so let’s now focus on the other components.



Infrastructure Services Classification

All the servers and services reside in separate pods and are managed by Kubernetes.

Stateless Services

- aka *microservices*. A microservice is a discrete service that runs in its own pod and that communicates using HTTP REST **API**.

Stateful Services

- provide services that require **persistence**, such as data storage, service configuration, messaging, caching.

Supporting Services

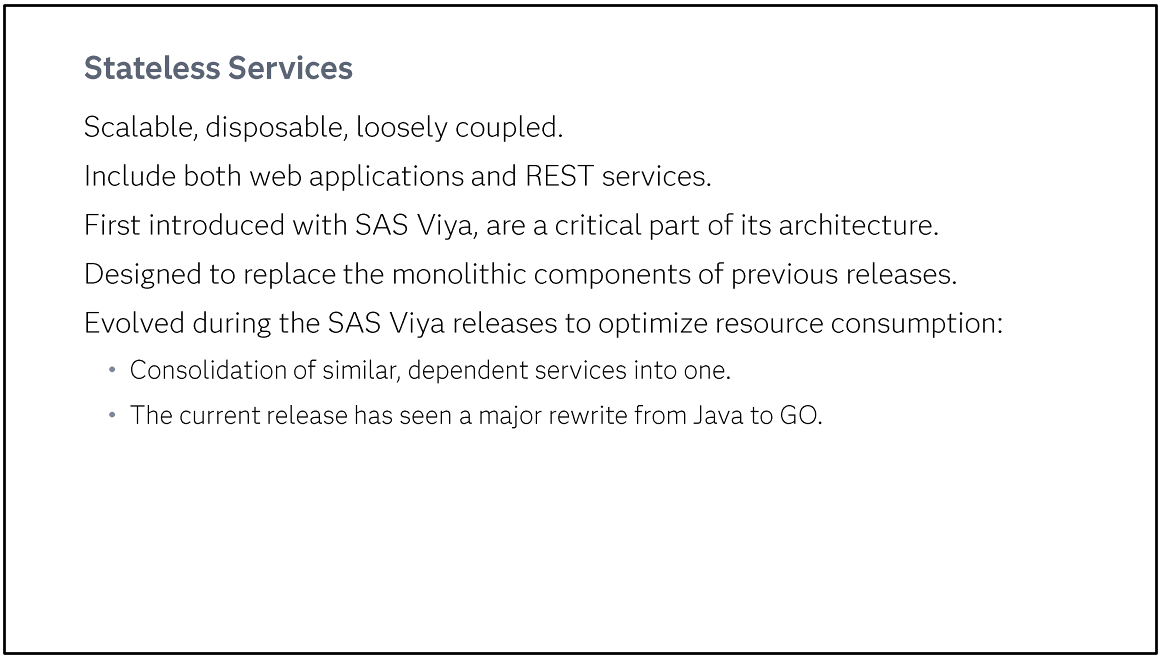
- use **third party** components that provide infrastructure services such as monitoring, logging, connection routing, TLS management, etc.

All the servers and services reside in one or more pods, managed by Kubernetes.
We usually can classify them in broad categories.

There are stateless services, also known as microservices, which are discrete services that run in their own pod and communicate using HTTP REST APIs.

Then there are stateful services, that require persistence. SAS Viya includes stateful servers to store data or service configuration, provide asynchronous inter service communication via messaging, and implement caching.

Finally, there are supporting services. Usually, these are third party components that provide infrastructure services, mostly from their own namespace, outside the SAS Viya platform. Examples include monitoring, logging, connection routing, certificates management for TLS, and so on.



Stateless Services

Scalable, disposable, loosely coupled.

Include both web applications and REST services.

First introduced with SAS Viya, are a critical part of its architecture.

Designed to replace the monolithic components of previous releases.

Evolved during the SAS Viya releases to optimize resource consumption:

- Consolidation of similar, dependent services into one.
- The current release has seen a major rewrite from Java to GO.

Stateless services, or microservices, are self-contained, lightweight software units that are loosely coupled, that is, depend on one another to the least extent possible. They can be deployed independently, started, stopped, managed and scaled individually.

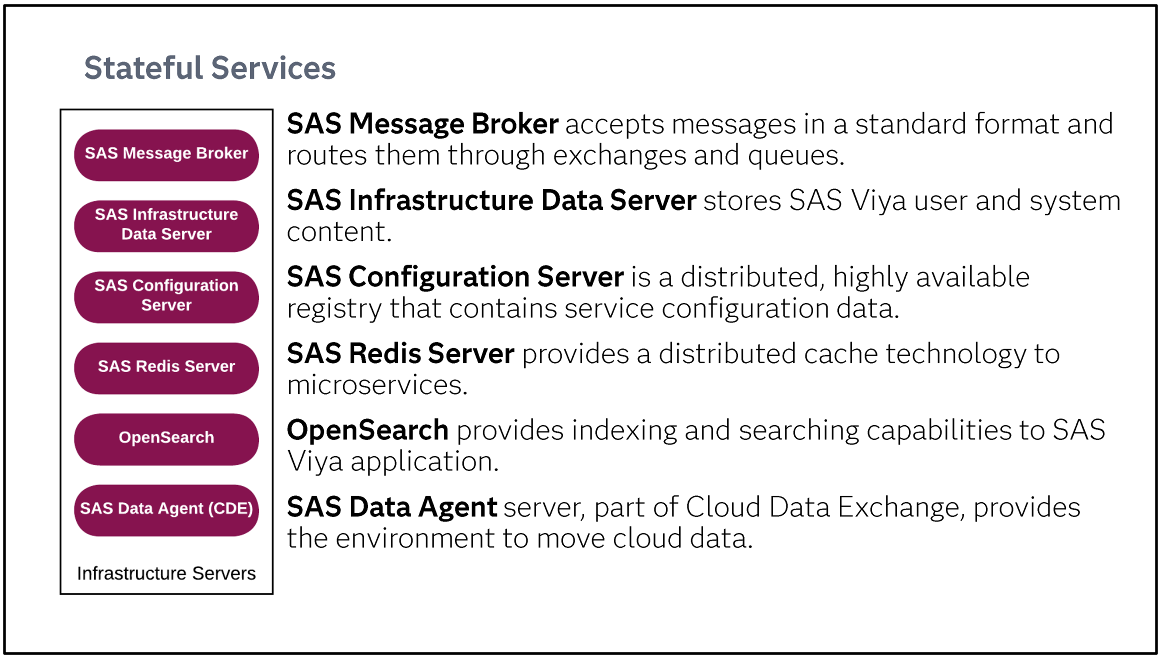
They provide a language-agnostic API; some also include a web interface for user interaction, so we classify them as web applications.

One or more instances of these processes can be running at any given time, and the number of sessions can change dynamically as demand changes.

The prefix “micro” doesn’t mean small, it refers to a single function or something that’s narrow in scope.

Stateless services are a critical piece of the SAS Viya platform; they are a ground-up rewrite of software that replaces the SAS 9 metadata-centric Web Infrastructure Platform.

With the cadence of subsequent SAS Viya releases, SAS continuously reassesses and sometimes rewrites stateless service to optimize their resource consumption; for example, similar, dependent services can be consolidated into a single pod; most services have been rewritten from Java to GO.



Stateful Services

SAS Message Broker
SAS Infrastructure Data Server
SAS Configuration Server
SAS Redis Server
OpenSearch
SAS Data Agent (CDE)
Infrastructure Servers

SAS Message Broker accepts messages in a standard format and routes them through exchanges and queues.

SAS Infrastructure Data Server stores SAS Viya user and system content.

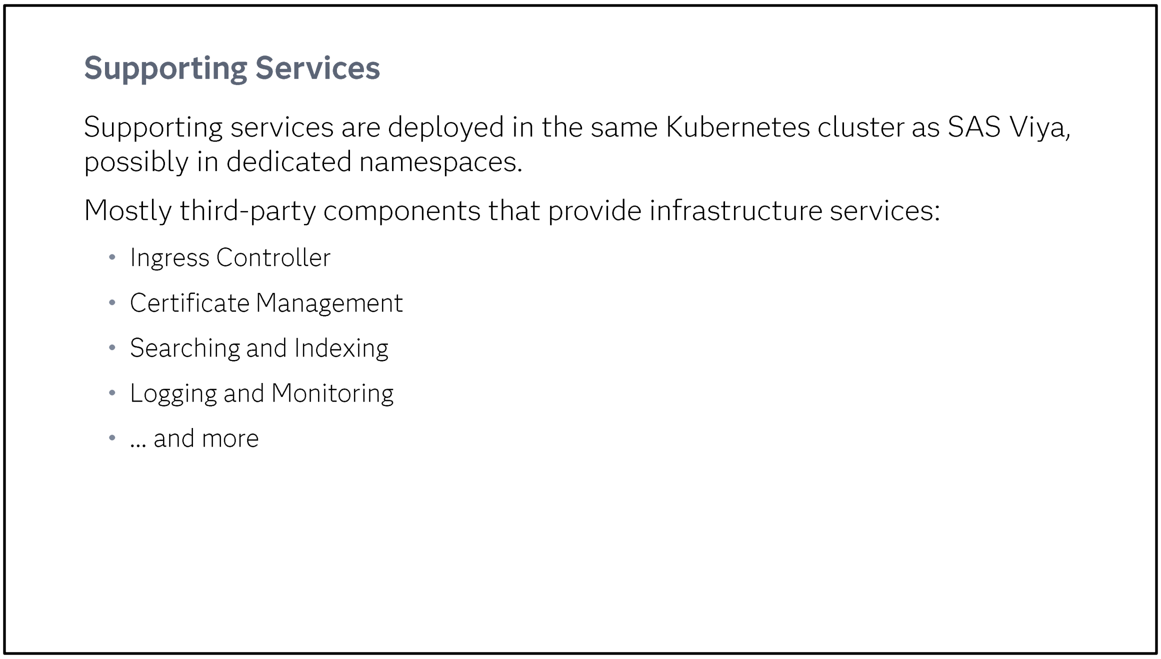
SAS Configuration Server is a distributed, highly available registry that contains service configuration data.

SAS Redis Server provides a distributed cache technology to microservices.

OpenSearch provides indexing and searching capabilities to SAS Viya application.

SAS Data Agent server, part of Cloud Data Exchange, provides the environment to move cloud data.

Let’s move to stateful services, that is, services that require persistence.
The SAS message broker accepts, routes, and manages asynchronous messages in a standard format.
The infrastructure data server is a database, currently using PostgreSQL, used to store both user and system content.
The configuration server is a distributed and highly available registry that contains service configuration.
The SAS Redis server provides a distributed cache technology to microservices.
OpenSearch provides indexing and searching capabilities to the SAS Viya applications.
Finally, the SAS Data Agent server, part of Cloud Data Exchange, provides an environment to move data between on-prem environments and the cloud.



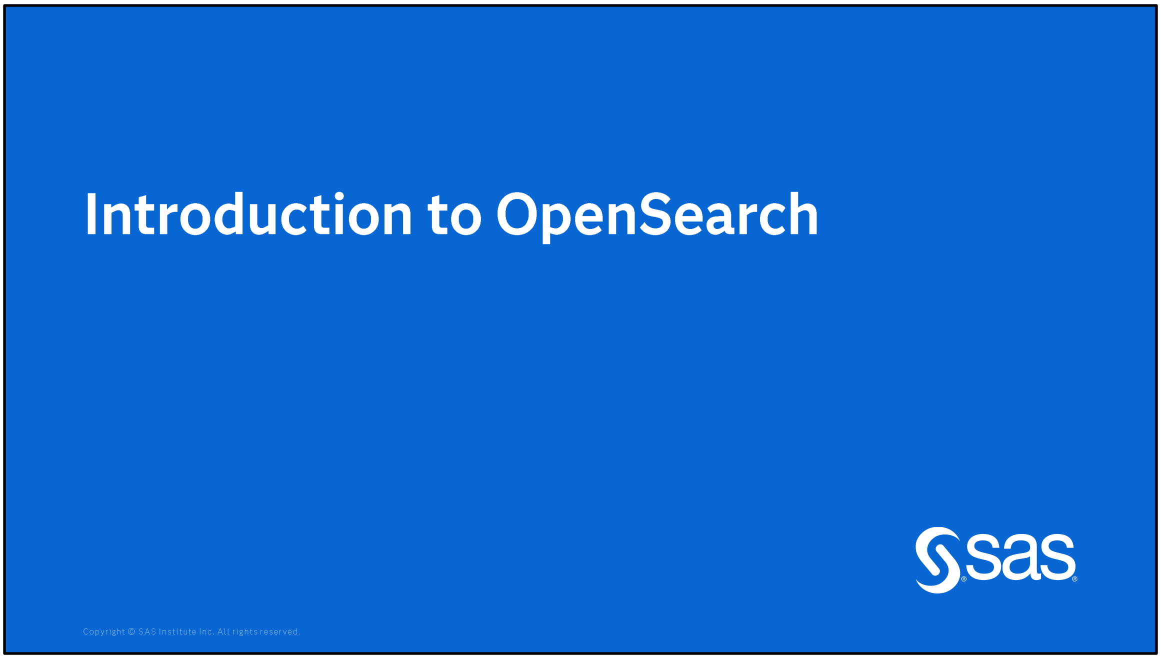
Supporting Services

Supporting services are deployed in the same Kubernetes cluster as SAS Viya, possibly in dedicated namespaces.

Mostly third-party components that provide infrastructure services:

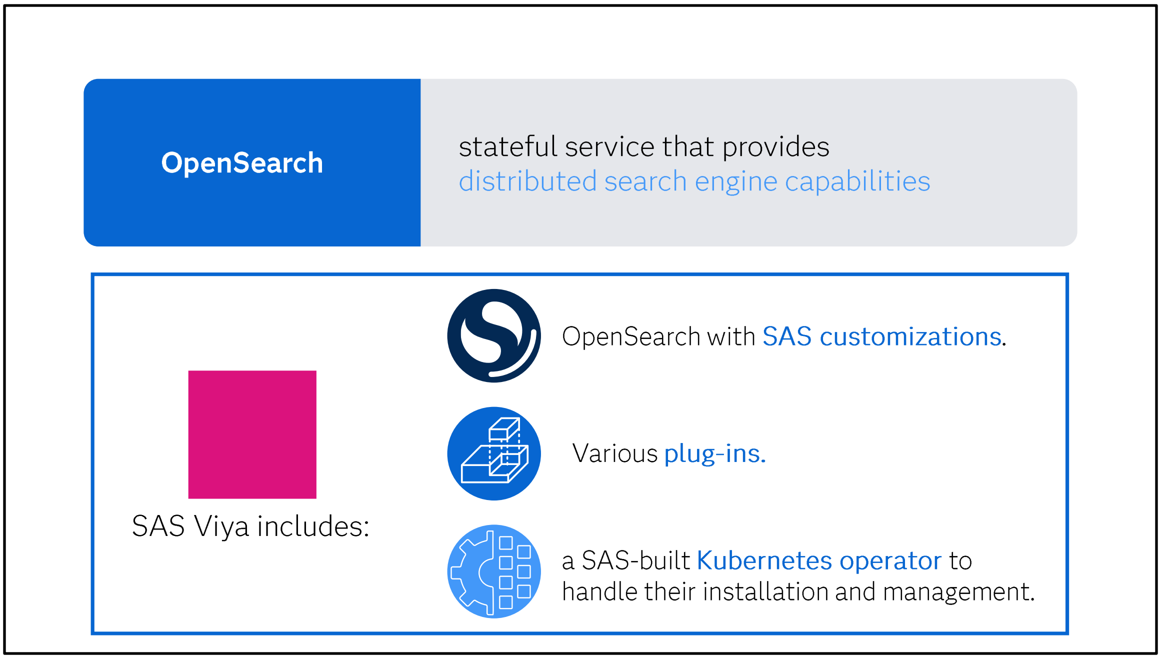
- Ingress Controller
- Certificate Management
- Searching and Indexing
- Logging and Monitoring
- ... and more

Supporting services are usually deployed in the same Kubernetes cluster as SAS Viya, possibly in dedicated namespaces.
These are mostly third-party components that provide infrastructure services, such as an ingress controller, a certificates manager, a searching and indexing server for logging and monitoring, and additional ones.



Introduction to OpenSearch

What are the main architectural considerations for OpenSearch?



OpenSearch

stateful service that provides
distributed search engine capabilities

SAS Viya includes:

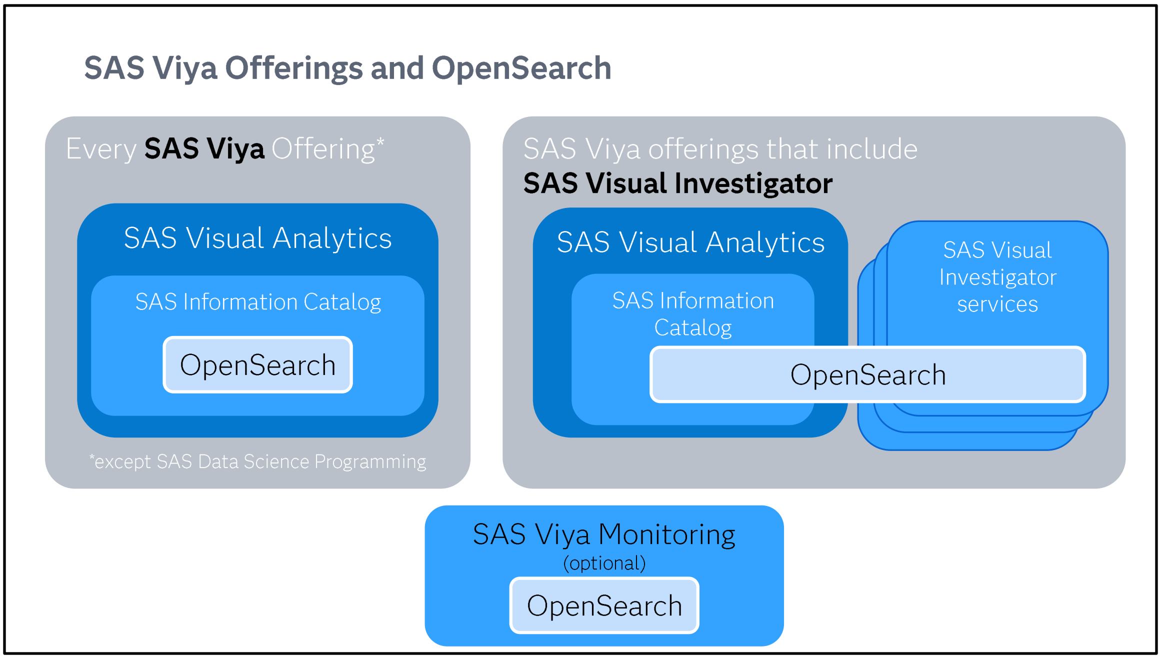
OpenSearch with SAS customizations.

Various plug-ins.

a SAS-built Kubernetes operator to
handle their installation and management.

OpenSearch is a stateful service that provides distributed search engine capabilities.

SAS Viya includes OpenSearch with SAS customizations, various plug-ins, and a SAS-built Kubernetes operator to handle their installation and management.
Previously, OpenSearch used to be called “Open Distro for Elasticsearch” or, in short, Opendistro. Some artifacts, such as SAS-provided Kubernetes objects, still include the name “opendistro”.



SAS Viya Offerings and OpenSearch

Every **SAS Viya** Offering*

SAS Visual Analytics

SAS Information Catalog

OpenSearch

*except SAS Data Science Programming

SAS Viya offerings that include
SAS Visual Investigator

SAS Visual Analytics

SAS Information Catalog

SAS Visual Investigator services

OpenSearch

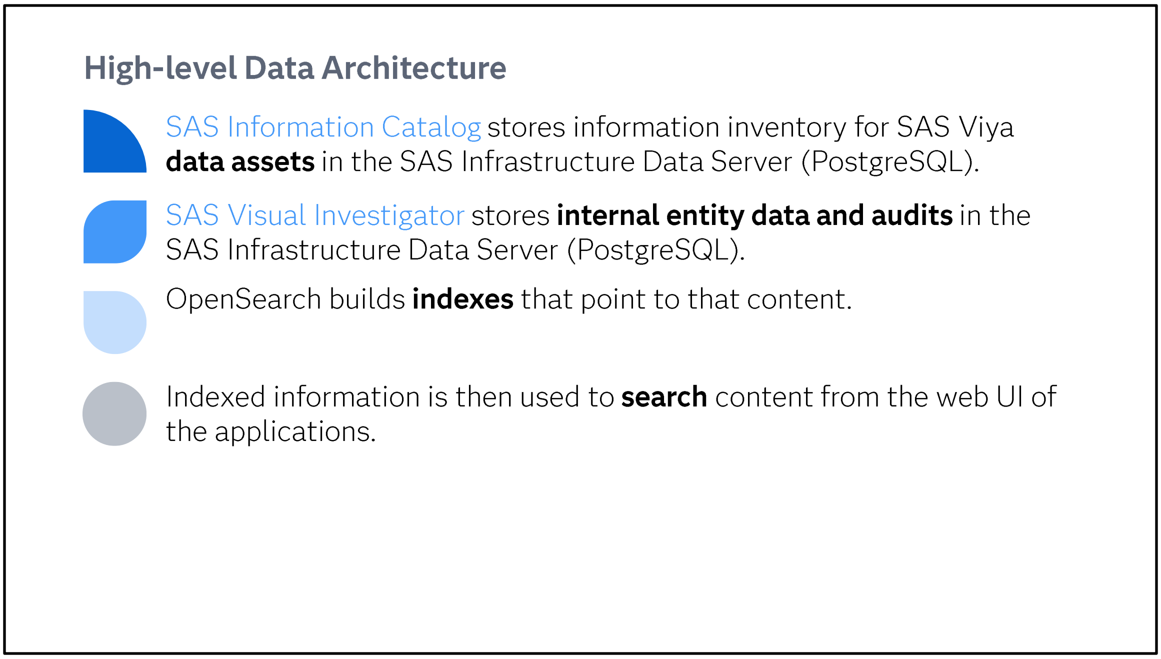
SAS Viya Monitoring
(optional)

OpenSearch

OpenSearch is included in SAS Information Catalog, which is included in all SAS Viya offerings except SAS Data Science Programming.

SAS Viya offerings that include SAS Visual Investigator leverage the same OpenSearch provided with SAS Information Catalog.
In later releases, additional solutions will leverage the included OpenSearch, such as Risk or Health and life sciences solutions.

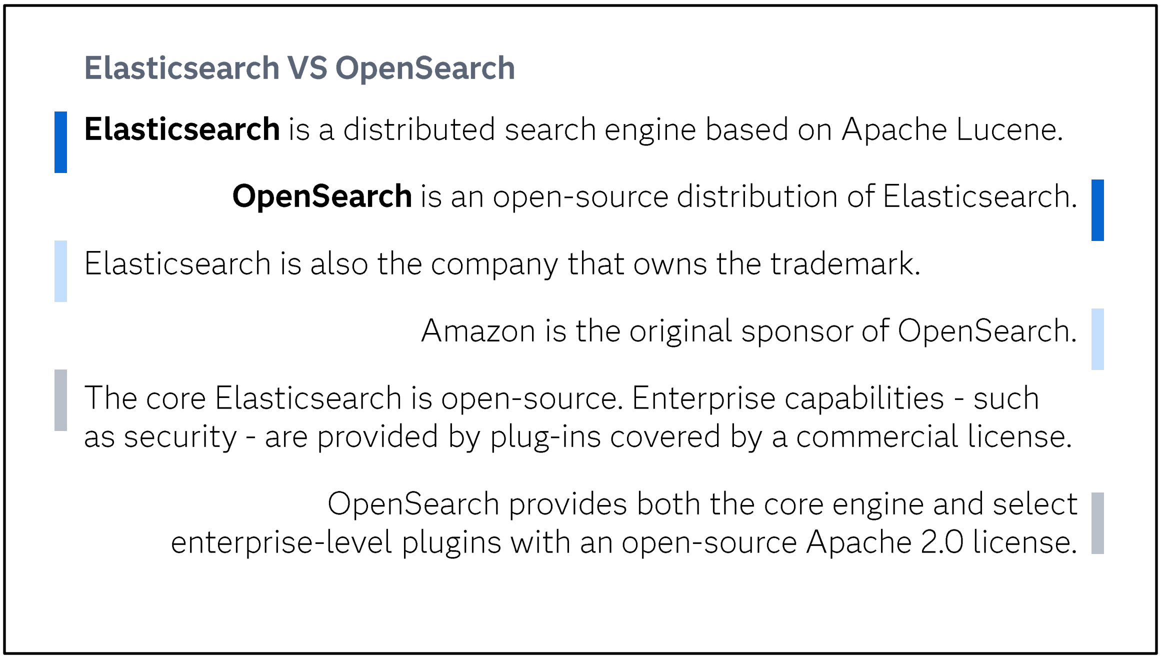
SAS Viya Monitoring for Kubernetes includes a different, dedicated instance of OpenSearch.



High-level Data Architecture

- SAS Information Catalog stores information inventory for SAS Viya **data assets** in the SAS Infrastructure Data Server (PostgreSQL).
- SAS Visual Investigator stores **internal entity data and audits** in the SAS Infrastructure Data Server (PostgreSQL).
- OpenSearch builds **indexes** that point to that content.
- Indexed information is then used to **search** content from the web UI of the applications.

In terms of high-level architecture, both SAS information Catalog and SAS Visual Investigator store their data in PostgreSQL. That includes SAS Information Catalog inventory and SAS Visual Investigator internal entity data and audits.
Then, they both use OpenSearch to build indexes on that content. These indexes enable faster search and retrieval of the content from the web interface of the applications.



Elasticsearch VS OpenSearch

Elasticsearch is a distributed search engine based on Apache Lucene.

OpenSearch is an open-source distribution of Elasticsearch.

Elasticsearch is also the company that owns the trademark.

Amazon is the original sponsor of OpenSearch.

The core Elasticsearch is open-source. Enterprise capabilities - such as security - are provided by plug-ins covered by a commercial license.

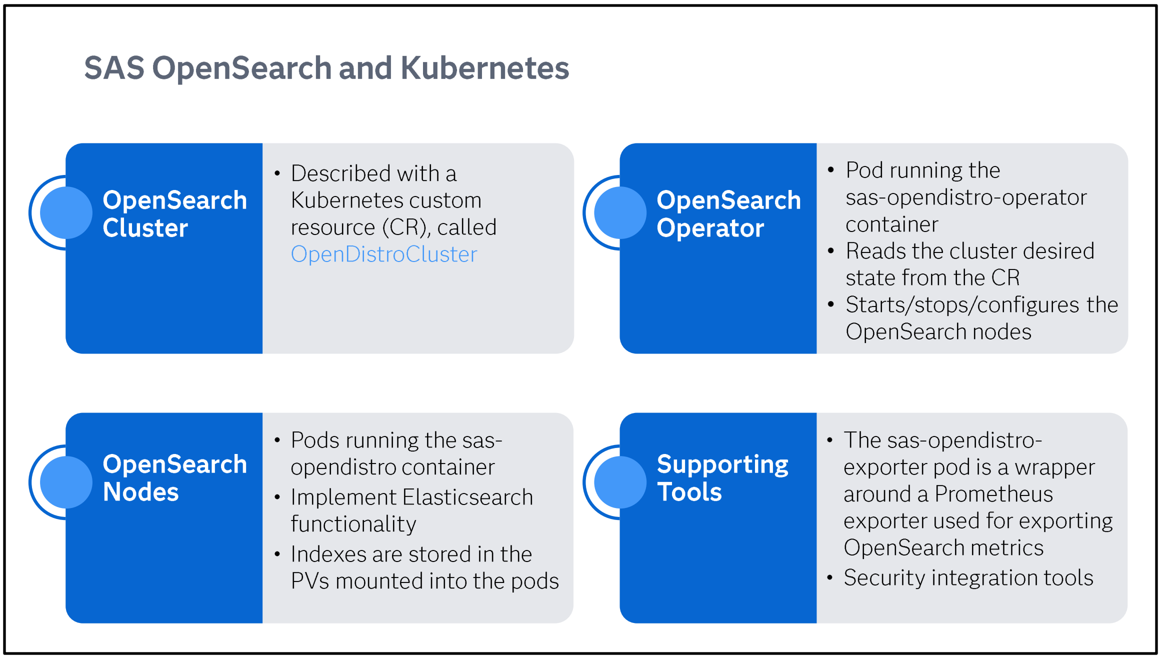
OpenSearch provides both the core engine and select enterprise-level plugins with an open-source Apache 2.0 license.

It's useful to have clarity about what's behind the names we use. We talk about OpenSearch, and then sometimes we mention Elasticsearch. What is the relationship?

Elasticsearch is a distributed search engine, based on the Apache Lucene project.
OpenSearch is an open-source distribution of Elasticsearch.

Elasticsearch is also the name of the company that owns the Elasticsearch product, including the trademark on the name.
Amazon is one of the first cloud-provider to offer a managed cloud version of the product. They forked the Elasticsearch project to create a new one, called Open Distro for Elasticsearch. That's the reason why some SAS artifacts include "opendistro" in their name.
Open Distro for Elasticsearch was eventually renamed with the current name of OpenSearch and is now managed by an independent foundation.

While the core Elasticsearch capabilities are available with an open-source license, enterprise capabilities such as the security plugins, are covered by a commercial license. Elasticsearch (the company) owns the rights on these additional components.
OpenSearch provides both the core and select enterprise-level plug-ins under a standard open-source license.



SAS OpenSearch and Kubernetes

OpenSearch Cluster

- Described with a Kubernetes custom resource (CR), called OpenDistroCluster

OpenSearch Operator

- Pod running the sas-opendistro-operator container
- Reads the cluster desired state from the CR
- Starts/stops/configures the OpenSearch nodes

OpenSearch Nodes

- Pods running the sas-opendistro container
- Implement Elasticsearch functionality
- Indexes are stored in the PVs mounted into the pods

Supporting Tools

- The sas-opendistro-exporter pod is a wrapper around a Prometheus exporter used for exporting OpenSearch metrics
- Security integration tools

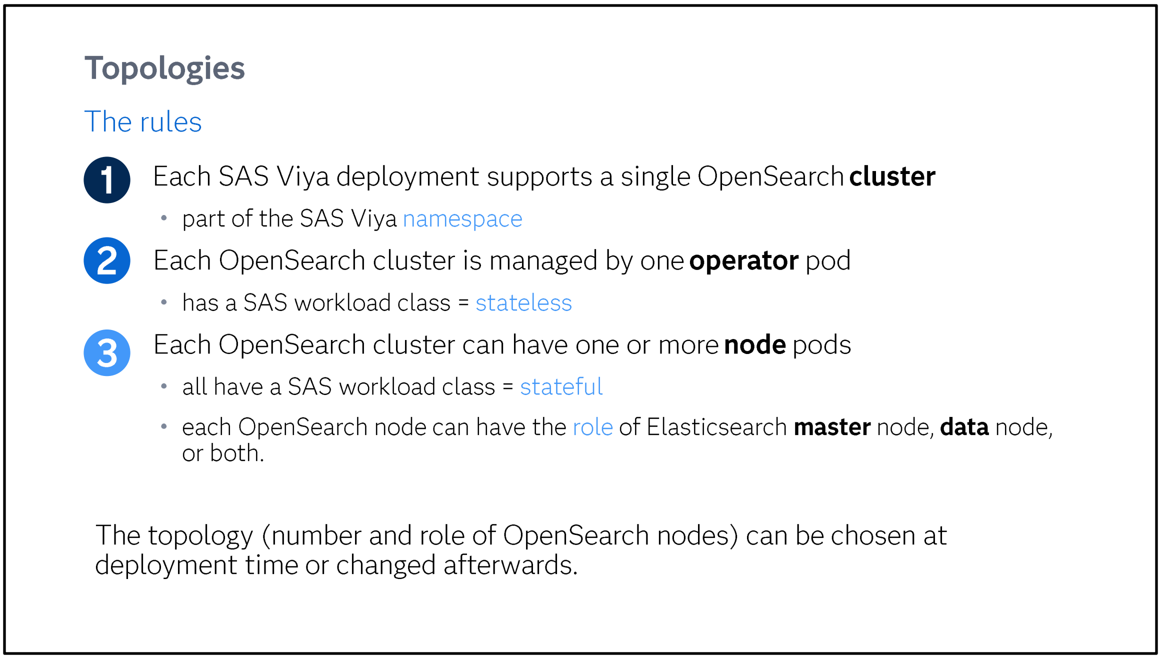
What Kubernetes artifacts are created or included with the OpenSearch instance provided with SAS Viya?

OpenSearch, even when running a single pod, is configured as a cluster. The cluster is described with a Kubernetes custom resource called “OpenDistroCluster”.
Just as we have seen with other SAS Viya components, such as CAS, custom resources describe a component configuration, including its topology, required resources, settings, and more.

The OpenDistroCluster custom resource is managed by a SAS-provided operator called sas-opendistro-operator.
The operator itself is a single pod. The operator reads the desired state of the cluster from the custom resource and acts on the Kubernetes pods accordingly, to start, stop and configure OpenSearch pods as specified.

By default, all the OpenSearch nodes are instantiated as pods running the sas-opendistro container.
Here, when we say nodes, we are referring to OpenSearch terminology. Nodes can be data nodes or master nodes; do not confuse them with physical nodes hosting the Kubernetes cluster. An OpenSearch node is a process that runs inside a pod, which runs on a Kubernetes node.
Each OpenSearch node contains the Elasticsearch indexes stored on physical volumes mounted in the pod.

Finally, the SAS-provided OpenSearch implementation includes additional supporting tools. For example, there are tools to export the metrics collected by the monitoring infrastructure, others to integrate OpenSearch authentication with SAS Viya, and more.



Topologies

The rules

1. Each SAS Viya deployment supports a single OpenSearch **cluster**
   - part of the SAS Viya namespace
2. Each OpenSearch cluster is managed by one **operator** pod
   - has a SAS workload class = stateless
3. Each OpenSearch cluster can have one or more **node** pods
   - all have a SAS workload class = stateful
   - each OpenSearch node can have the role of Elasticsearch **master** node, **data** node, or both.

The topology (number and role of OpenSearch nodes) can be chosen at deployment time or changed afterwards.

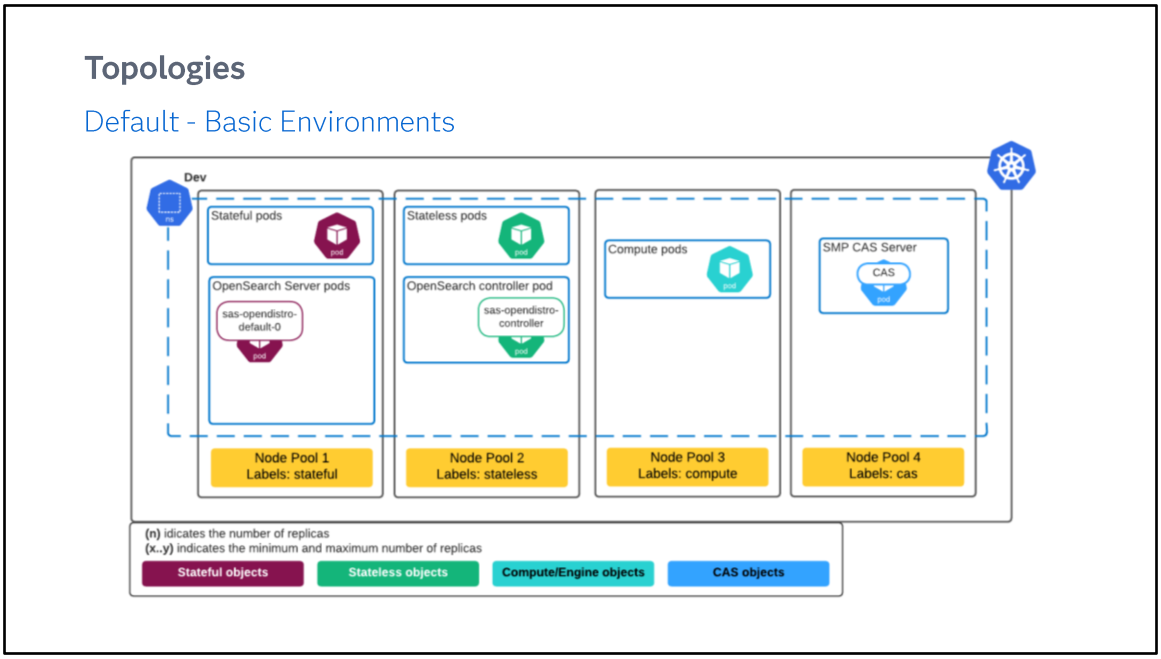
You are free to implement the OpenSearch topology that best suits your business requirements, as long as you follow a few rules.

Each SAS Viya deployment can have one, and only one, OpenSearch cluster per namespace. If you have multiple SAS Viya namespaces, then you'll have multiple corresponding instances of OpenSearch.

Each cluster is managed by a corresponding operator, running inside a pod labelled with the stateless workload class.

Each OpenSearch node runs in a pod with the stateful workload label. These can have two roles. They can be a master node, or a data node, or both at the same time.
The master nodes are the ones that talk to the clients and manage the state of the cluster.
The data nodes are the ones that build, store, and retrieve the indexes.

The topology can be configured during the initial deployment, or changed afterwards.
For example, you can accept the default one-node deployment to get started, then configure additional nodes as required.



Topologies

Default - Basic Environments

Dev

ns

Stateful pods

pod

OpenSearch Server pods

sas-opendistro-default-0

pod

Node Pool 1
Labels: stateful

Stateless pods

pod

OpenSearch controller pod

sas-opendistro-controller

pod

Node Pool 2
Labels: stateless

Compute pods

pod

Node Pool 3
Labels: compute

SMP CAS Server

CAS

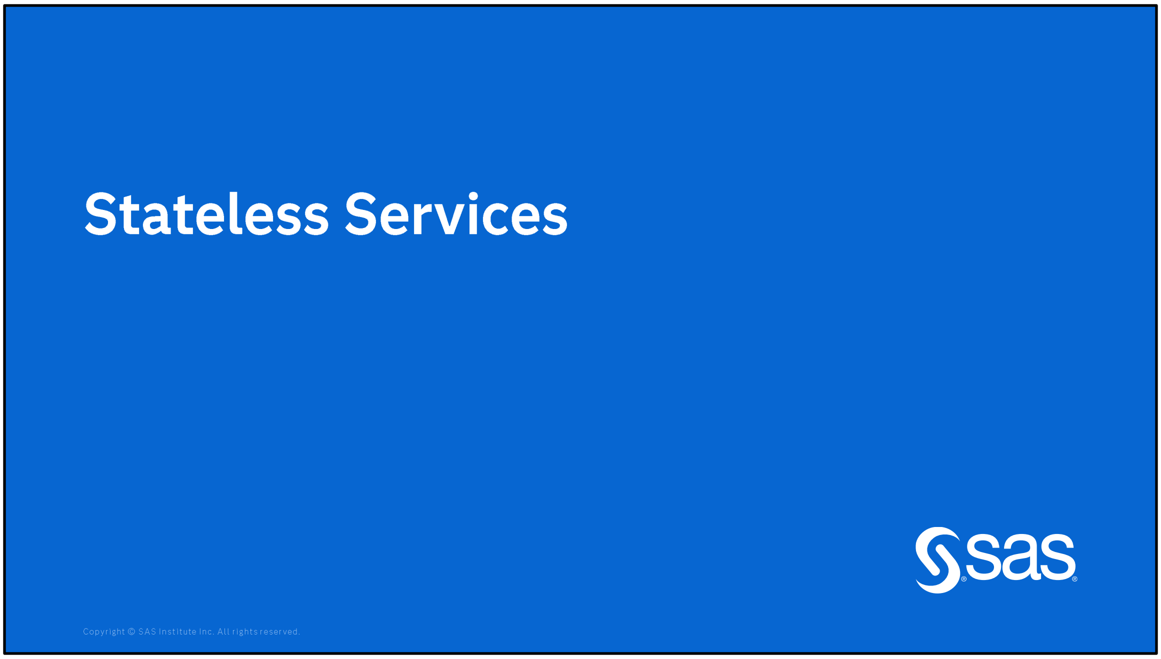
pod

Node Pool 4
Labels: cas

(n) idicates the number of replicas
(x..y) indicates the minimum and maximum number of replicas

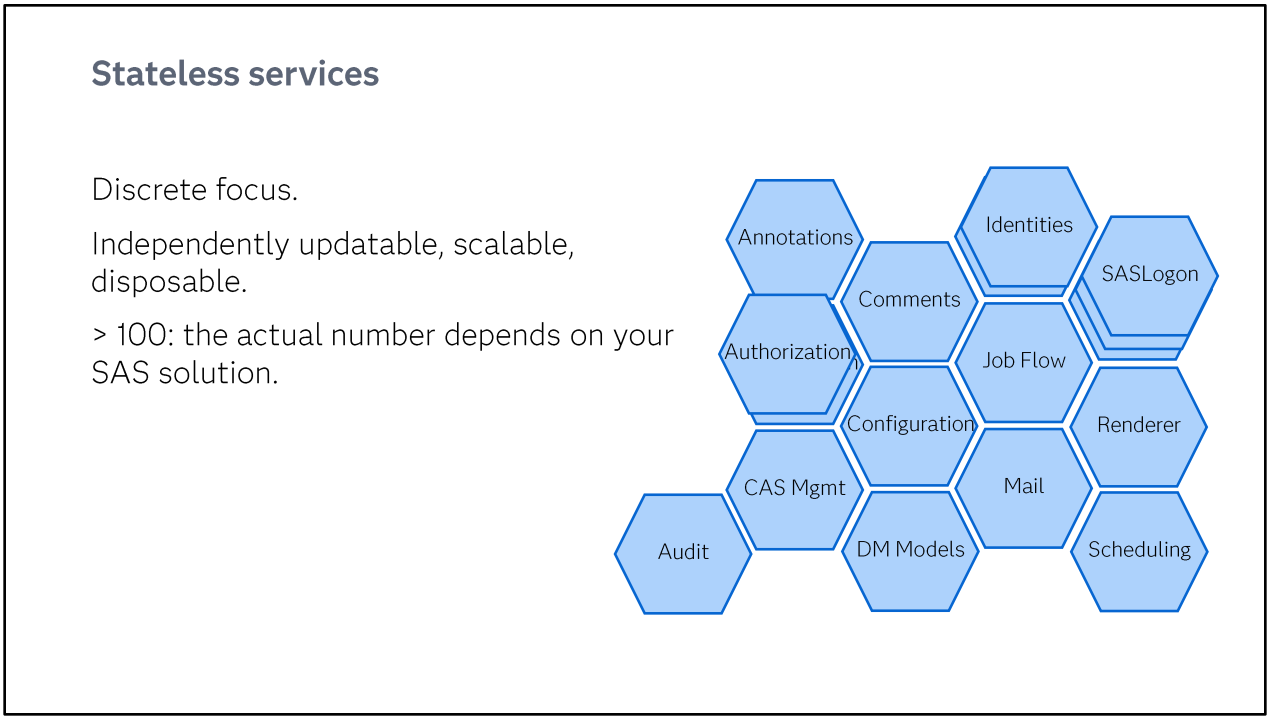
Stateful objects | Stateless objects | Compute/Engine objects | CAS objects

Here is an architectural diagram presenting a simple environment.
This is a default deployment with a single OpenSearch node serving all Elasticsearch roles (both master and data). This is running in a pod on the stateful nodepool. Then, there is the controller pod, running on the stateless nodepool.



Stateless Services

Let’s present a brief overview of the main architectural considerations about the stateless services, also known as microservices.



Stateless services

Discrete focus.

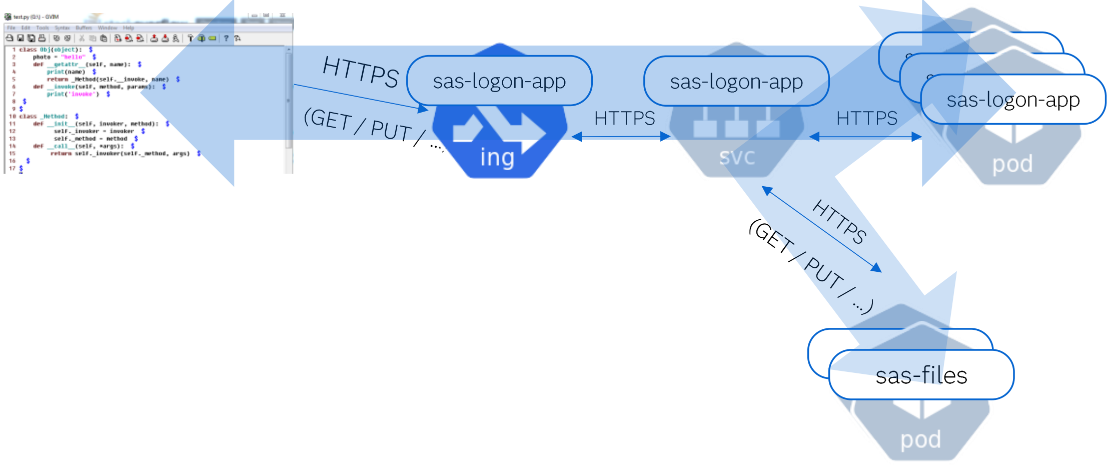
Independently updatable, scalable, disposable.

> 100: the actual number depends on your SAS solution.

Stateless services, or microservices, are self-contained, lightweight software units that are loosely coupled, that is, depend on one another to the least extent possible.
They can be deployed independently, started, stopped, managed and scaled individually.
Each SAS Viya platform deployment can have a different number of stateless services deployed and running, depending on the licensed software solution. Anyhow, expect to see more than a hundred pods running at any time.

RESTful APIs

Language of choice

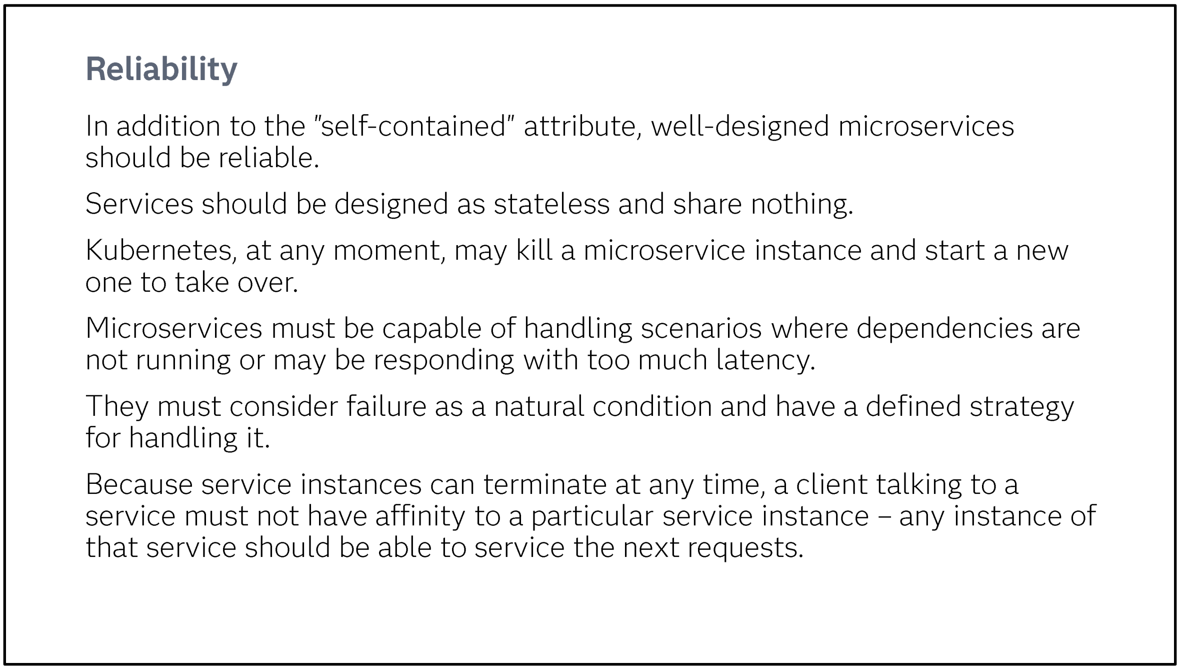


```
class Obj(object):  $
    photo = "hello"  $
    def __getattr__(self, name):  $
        print(name)  $
        return _Method(self.__invoke, name)  $
    def __invoke(self, method, params):  $
        print('invoke')  $
  $
$
class _Method:  $
    def __init__(self, invoker, method):  $
         self._invoker = invoker  $
         self._method = method  $
    def __call__(self, *args):  $
         return self._invoker(self._method, args)  $
  $
$
```

Internal and external components and languages communicate with microservices calling their API endpoints via HTTPS.
Most programming languages support RESTful API calls using the HTTP protocol.

External traffic (called “south-north”) goes through Kubernetes ingresses.

Internal traffic, such as between microservices of the same namespace (called “east-west”), goes trough Kubernetes services.
Microservices supporting web applications like SAS Visual Analytics have their own Web Application Server embedded within the microservice.



Reliability

In addition to the "self-contained" attribute, well-designed microservices should be reliable.

Services should be designed as stateless and share nothing.

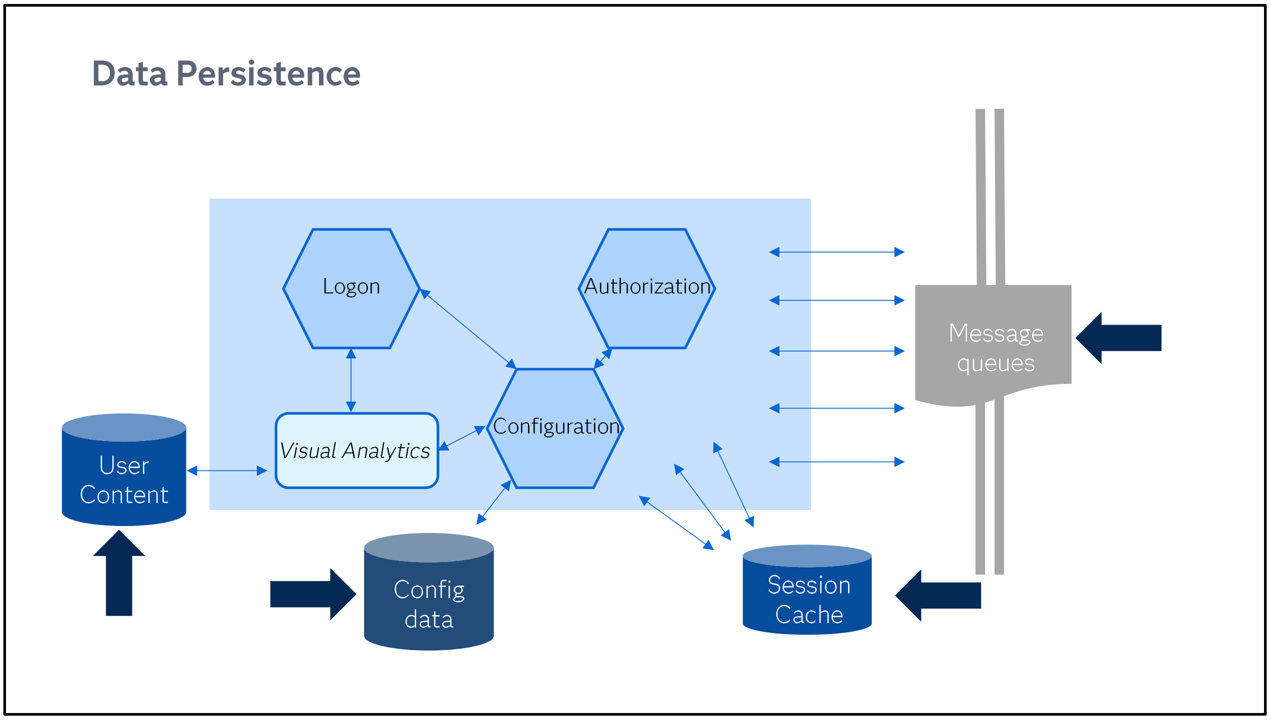
Kubernetes, at any moment, may kill a microservice instance and start a new one to take over.

Microservices must be capable of handling scenarios where dependencies are not running or may be responding with too much latency.

They must consider failure as a natural condition and have a defined strategy for handling it.

Because service instances can terminate at any time, a client talking to a service must not have affinity to a particular service instance – any instance of that service should be able to service the next requests.

In addition to the "self-contained" attribute, well-designed microservices should be reliable.
Services should be designed as stateless and share nothing.
Kubernetes, at any moment, may kill a microservice instance and start a new one to take over.
Microservices must be capable of handling scenarios where dependencies are not running or may be responding with too much latency.
They must consider failure as a natural condition and have a defined strategy for handling it.
Because service instances can terminate at any time, a client talking to a service must not have affinity to a particular service instance: any instance of that service should be able to service the next requests.



Data Persistence

Since microservices are stateless, they need some place to store persistent data.

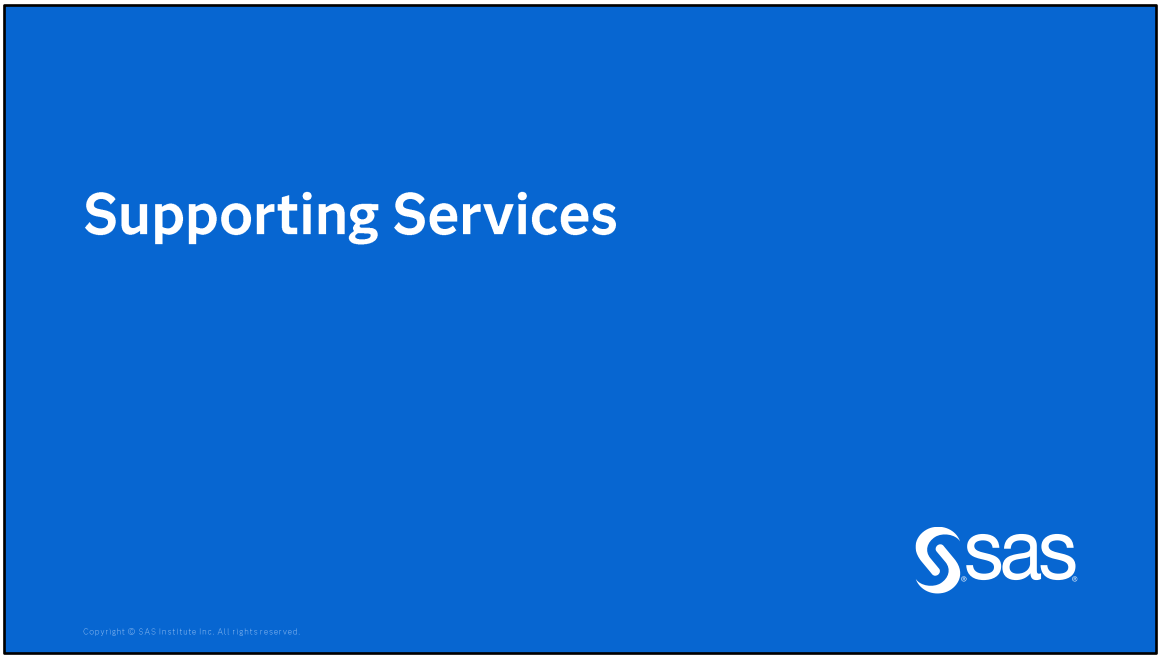
Configuration data is stored in SAS Configuration Server. This includes:
Registry of all configured microservice instances
JVM options like memory size
Logging levels, such as ERROR, WARN, INFO, DEBUG
And additional content.

User Content is stored in SAS Infrastructure Data Server. This includes:
Folders
Reports
Jobs
And additional content.

Message Queues allow services to report on activity and allow other services to monitor the activity.
As an example, configuration updates are asynchronous messages sent over to the message queue.

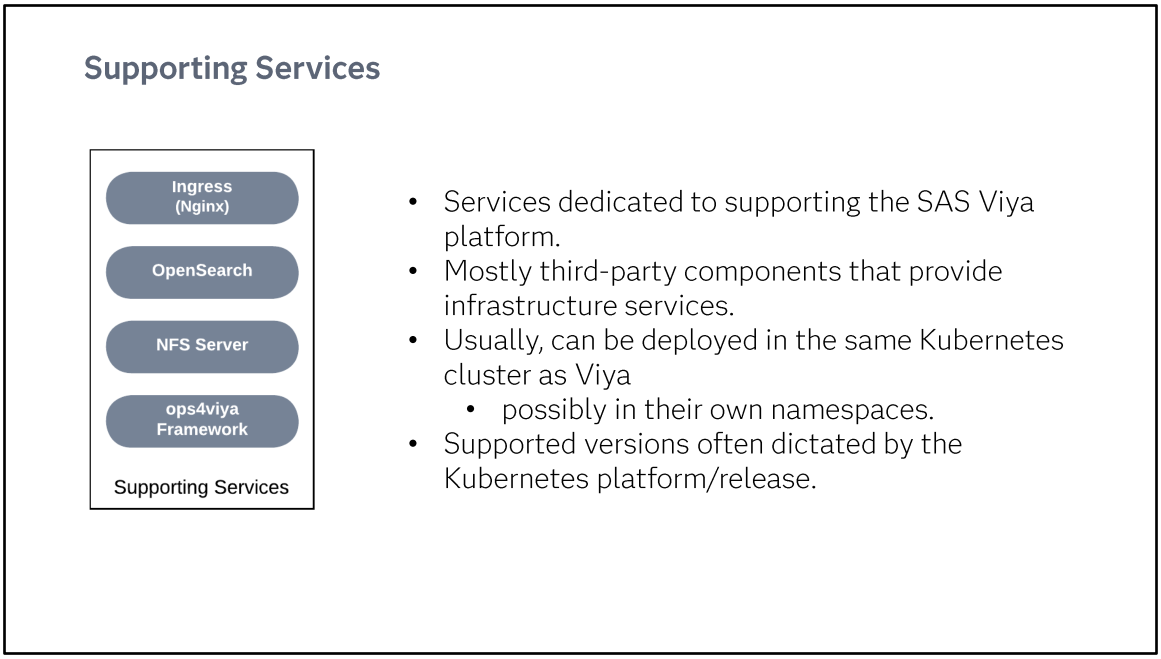
SAS Redis Server provides a distributed cache technology to microservices.

“Persistent” here is a loose term: SAS Redis Server and SAS Message Broker are cleared and reset at every restart, so they persist “for the duration of a session” or “as needed by the active service that used them”. SAS Infrastructure Data Server and SAS Configuration Server, on the contrary, provide long-term stores that survives reboots.



Supporting Services

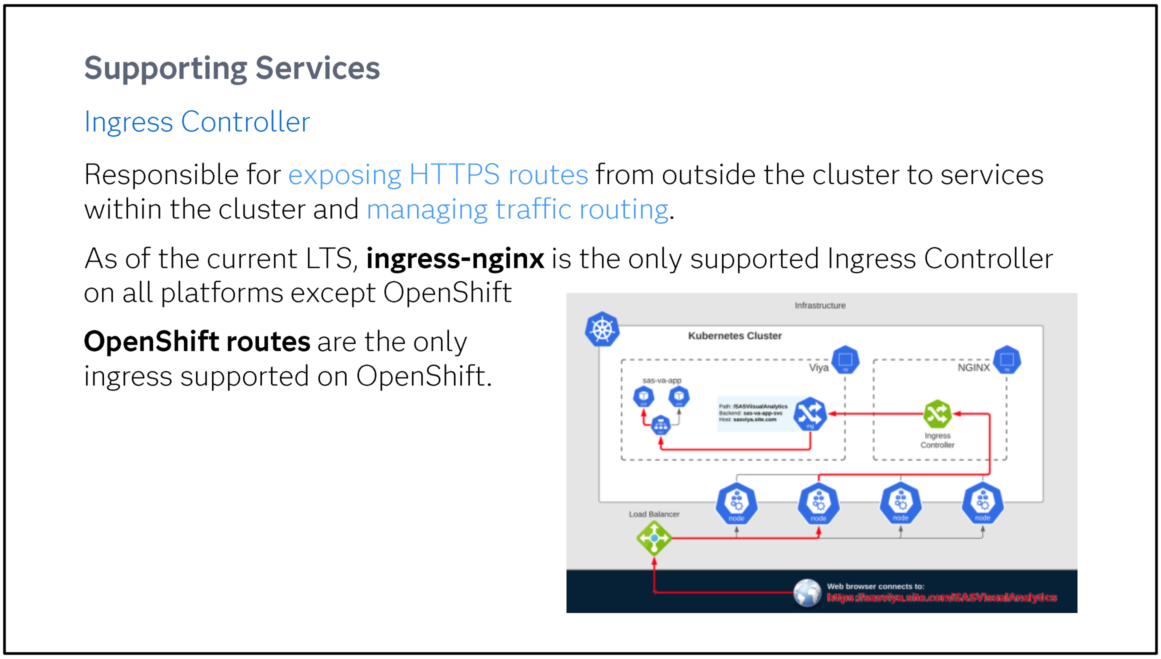
We will now present a brief description of the SAS Viya supporting services.



Supporting Services

- Services dedicated to supporting the SAS Viya platform.
- Mostly third-party components that provide infrastructure services.
- Usually, can be deployed in the same Kubernetes cluster as Viya
  - possibly in their own namespaces.
- Supported versions often dictated by the Kubernetes platform/release.

Here we present a high-level view of those supporting services.
Mostly, these are third-party components and third-party servers that provide infrastructure services to the SAS Viya platform.
Usually, we try to keep them as close as possible to the platform itself. So, they can be deployed in the same Kubernetes cluster, possibly in their own namespace.
When deploying third party components, pay attention to their system requirements. Most of the time, those are dictated by the Kubernetes version, more than by the SAS Viya version.



Supporting Services

Ingress Controller

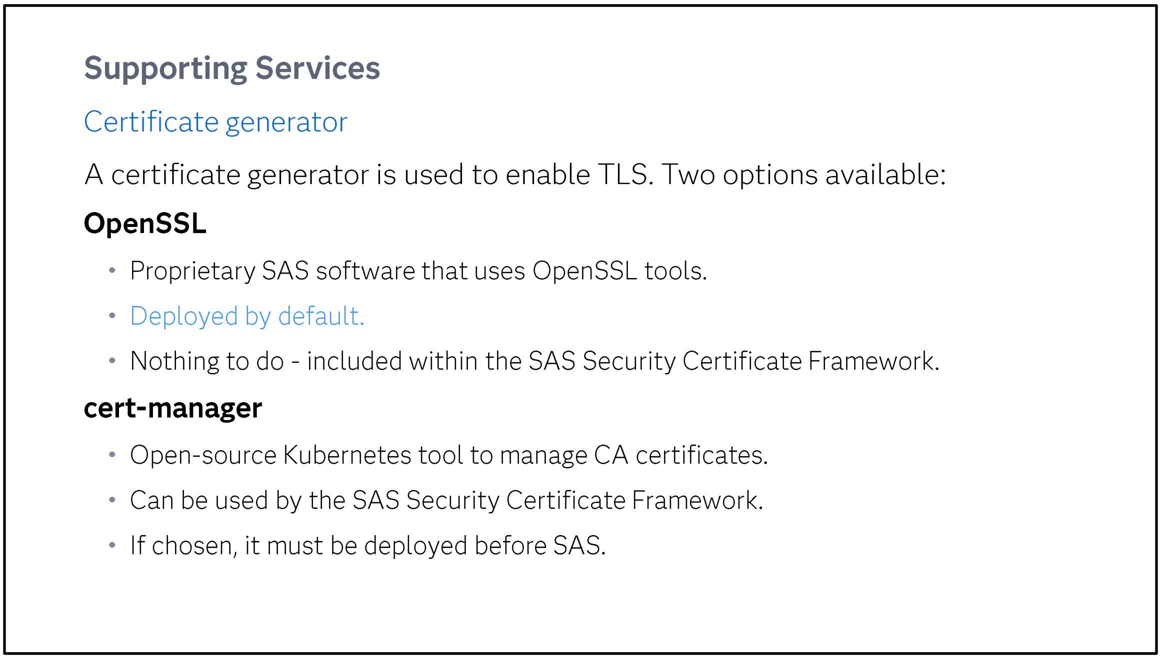
Responsible for exposing HTTPS routes from outside the cluster to services within the cluster and managing traffic routing.

As of the current LTS, **ingress-nginx** is the only supported Ingress Controller on all platforms except OpenShift

OpenShift routes are the only ingress supported on OpenShift.

The first supporting service that we cover is probably one of the most important one. It's actually a requirement. The SAS Viya platform needs an ingress controller to provide an HTTP route from outside the cluster to services running inside.
In Kubernetes there are multiple implementations of controllers, but, as of the current version, only ingress-nginx is supported for SAS Viya on most cloud environments. The exception is the OpenShift platform, where SAS Viya uses OpenShift routes.

Ingress controllers only expose HTTPS traffic. To enable user access to the CAS controller binary port or to the SAS/CONNECT port from hosts that are outside the cluster, a LoadBalancer is required on AKS, EKS, GKE.



Supporting Services

Certificate generator

A certificate generator is used to enable TLS. Two options available:

OpenSSL

- Proprietary SAS software that uses OpenSSL tools.
- Deployed by default.
- Nothing to do - included within the SAS Security Certificate Framework.

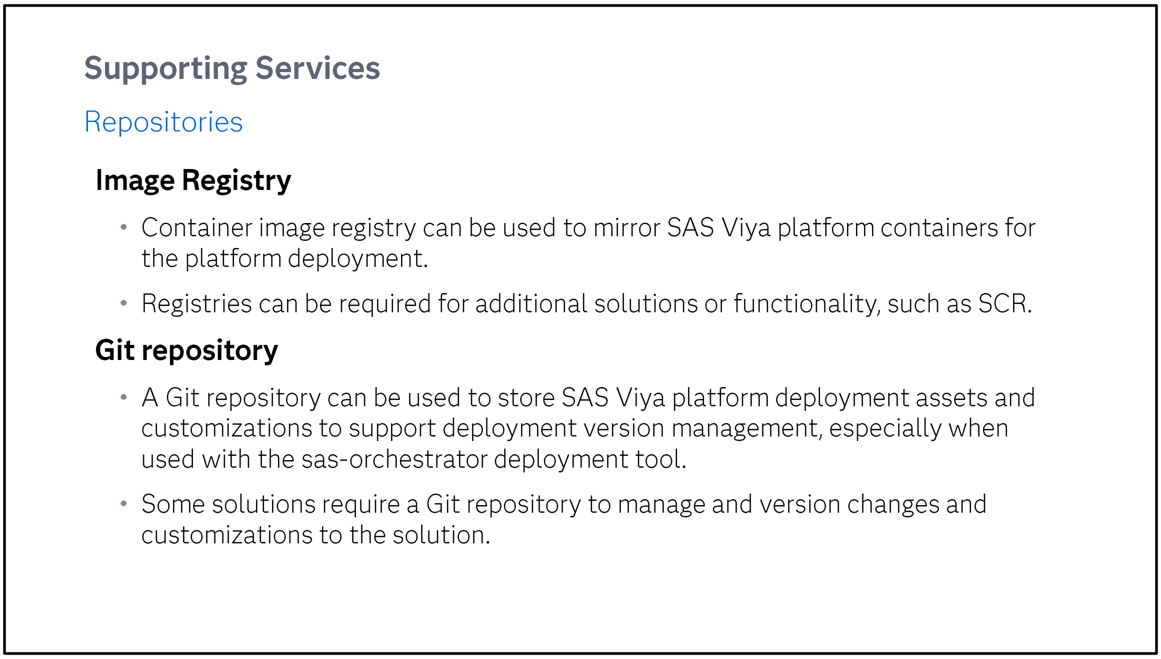
cert-manager

- Open-source Kubernetes tool to manage CA certificates.
- Can be used by the SAS Security Certificate Framework.
- If chosen, it must be deployed before SAS.

Another required component is a certificate generator. A certificate generator is used to enable TLS on network routes.
SAS Viya supports two options.

You can use the default option provided by SAS, a proprietary framework based on the OpenSSL open-source project. If you accept this default, the SAS Viya platform will use a set of OpenSSL tools to provide certificate management.

The second option is cert-manager.
This is an open-source set of tools to manage certificates. It used to be the previous default, and it can be still used by the SAS Security Certificate Framework.
If you choose to use cert-manager, it has to be deployed before deploying SAS Viya.
The SAS provided option is the preferred, in terms of platform integration, and easier scalability.



Supporting Services

Repositories

Image Registry

- Container image registry can be used to mirror SAS Viya platform containers for the platform deployment.
- Registries can be required for additional solutions or functionality, such as SCR.

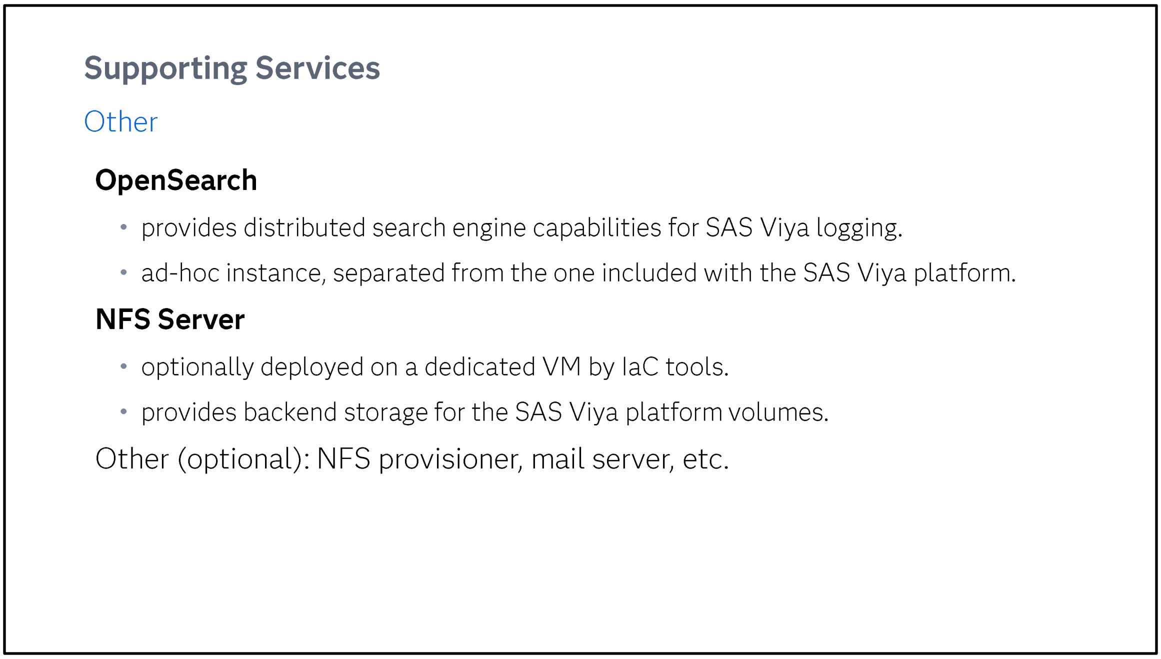
Git repository

- A Git repository can be used to store SAS Viya platform deployment assets and customizations to support deployment version management, especially when used with the sas-orchestrator deployment tool.
- Some solutions require a Git repository to manage and version changes and customizations to the solution.

Let’s move to optional third party components.

You can use a container image registry. This can be used to mirror SAS Viya platform containers for the platform deployment. Registries can also be required for additional solutions or functionality, such as SCR.
You can use, for example, a Docker image registry to store the SAS Viya images to create a mirror used to deploy at dark sites, that is, not connected to the internet.

GIT repositories can be used to store SAS Viya platform assets, for example, for the deployment operator, or it can be used by SAS Studio to version SAS code, or for some solutions.
Actually, some solutions require a GIT repository to manage their artifacts.



Supporting Services

Other

OpenSearch

- provides distributed search engine capabilities for SAS Viya logging.
- ad-hoc instance, separated from the one included with the SAS Viya platform.

NFS Server

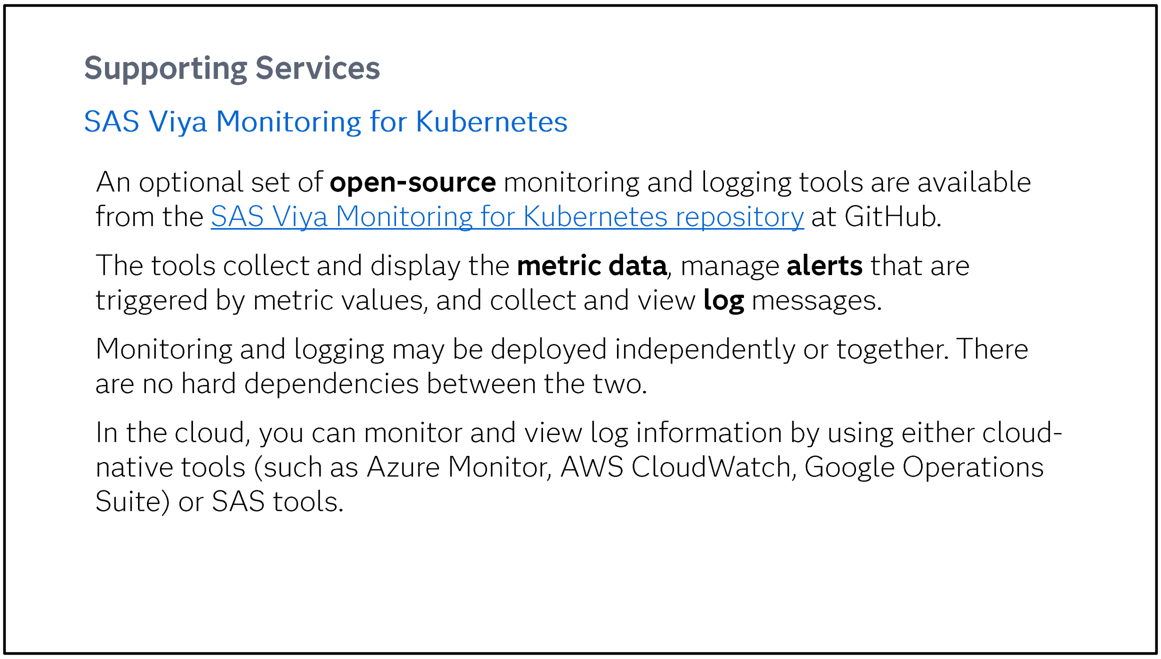
- optionally deployed on a dedicated VM by IaC tools.
- provides backend storage for the SAS Viya platform volumes.

Other (optional): NFS provisioner, mail server, etc.

An optional deployment of OpenSearch can provide indexing and searching capabilities for the monitoring and logging tools. This is sometimes called “external OpenSearch” to clarify that it’s a second instance, different from the one embedded with the SAS Viya platform, which provides similar capabilities to SAS Viya applications.

IAC tools, used to deploy SAS Viya, may optionally configure an NFS server running in a dedicated virtual machine, next to the Kubernetes cluster, to provide storage services to the SAS Viya platform.

Additional optional services can include an NFS provisioner, acting as an NFS client in the SAS Viya cluster. There could be a mail server in to provide mail capabilities to the SAS Viya services, for example, to send notifications to the administrators or for end-users to send reports as email attachments, and so on.



Supporting Services

SAS Viya Monitoring for Kubernetes

An optional set of **open-source** monitoring and logging tools are available from the SAS Viya Monitoring for Kubernetes repository at GitHub.

The tools collect and display the **metric data**, manage **alerts** that are triggered by metric values, and collect and view **log** messages.

Monitoring and logging may be deployed independently or together. There are no hard dependencies between the two.

In the cloud, you can monitor and view log information by using either cloud-native tools (such as Azure Monitor, AWS CloudWatch, Google Operations Suite) or SAS tools.

Let’s spend a few words on the SAS Viya monitoring and logging tools.
SAS provides an optional set of open-source tools that are not deployed together with the SAS Viya platform, but rather provided from GitHub for free.
Since they are not deployed with SAS Viya, they require their own namespace.
As the name implies, the monitoring tools are used to collect metric data and to manage alerts, providing a graphical interface to that. Then, logging tools can be used to collect and display log messages.
The two tools, monitoring and logging, can be deployed together, or individually. There are no hard dependencies between the two.
In the cloud, you can choose to use cloud-native tools provided by the cloud vendor: SAS Viya can integrate with Azure monitoring, AWS CloudWatch, Google Operations suite, and more.

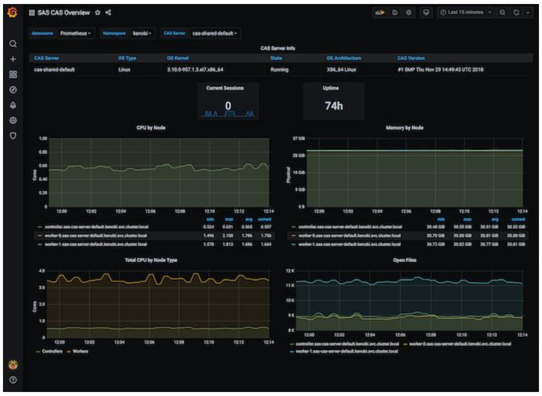
Components

SAS Viya Monitoring for Kubernetes

Prometheus

Alertmanager

Grafana



CAS Server Info

CAS Server	OS Type	OS Kernel	State	OS Architecture	CAS Version
cas-shared-default	Linux	3.10.0-957.1.3.el7.x86_64	Running	X86_64 Linux	#1 SMP Thu Nov 29 14:49:43 UTC 2018

CPU by Node

	min	max	avg	current
controller.sas-cas-server-default.kenobi.svc.cluster.local	0.524	0.631	0.565	0.557
worker-0.sas-cas-server-default.kenobi.svc.cluster.local	1.496	2.158	1.786	1.756
worker-1.sas-cas-server-default.kenobi.svc.cluster.local	1.578	1.813	1.686	1.664

Memory by Node

	min	max	avg	current
controller.sas-cas-server-default.kenobi.svc.cluster.local	30.48 GiB	30.55 GiB	30.51 GiB	30.52 GiB
worker-0.sas-cas-server-default.kenobi.svc.cluster.local	30.70 GiB	30.88 GiB	30.81 GiB	30.88 GiB
worker-1.sas-cas-server-default.kenobi.svc.cluster.local	30.72 GiB	30.82 GiB	30.77 GiB	30.81 GiB

The monitoring tools are based on standard, open-source components. They include Prometheus, Alert Manager, and Grafana.

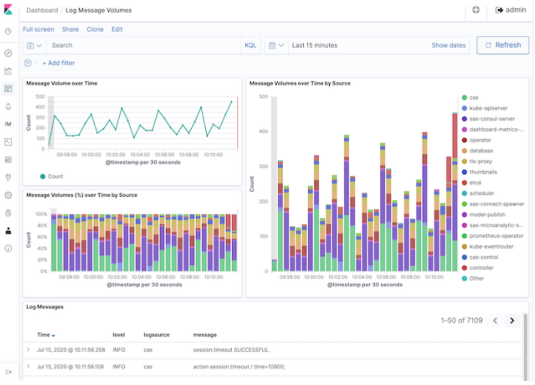
Components

SAS Viya Logging for Kubernetes

Fluent Bit

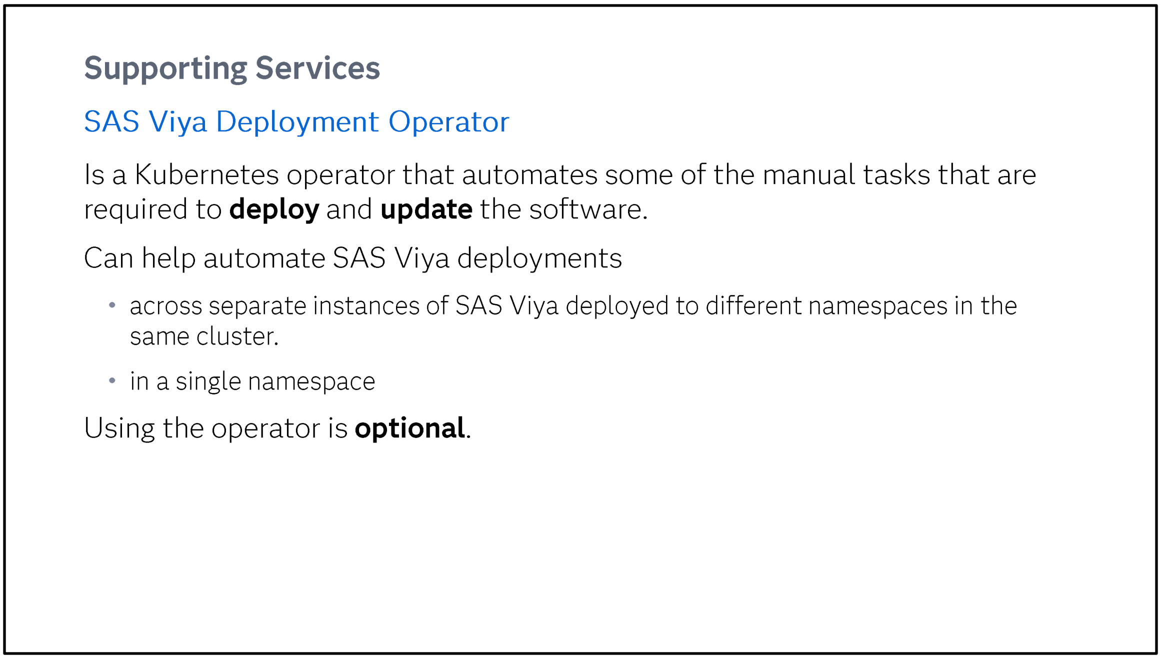
OpenSearch

OpenSearch Dashboards



Time	level	logsource	message
Jul 15, 2020 @ 10:11:56.208	INFO	cas	session.timeout SUCCESSFUL.
Jul 15, 2020 @ 10:11:56.108	INFO	cas	action session.timeout / time=10800;

Similarly, the logging tools provide Fluent Bit, to collect logs, OpenSearch, to store and index them, and OpenSearch Dashboards, to navigate through the logs.
Both for monitoring and logging, the SAS GitHub repositories include the required configuration to integrate them with the SAS Viya platform. They also include pre-configured dashboards, displays and queries.



Supporting Services

SAS Viya Deployment Operator

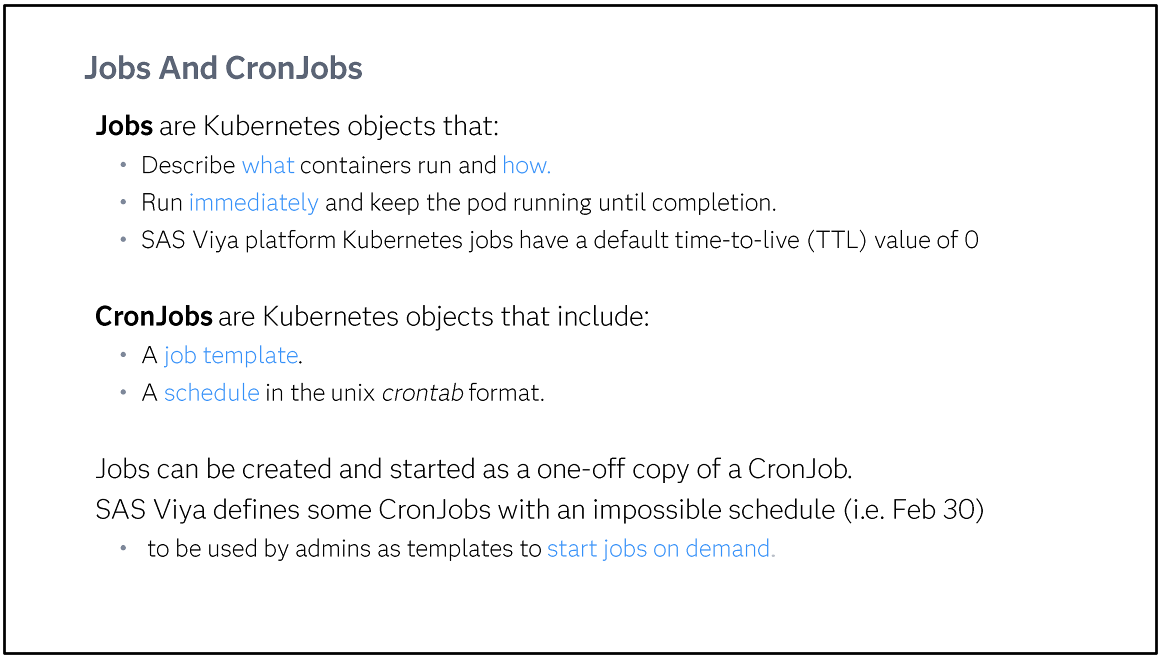
Is a Kubernetes operator that automates some of the manual tasks that are required to **deploy** and **update** the software.

Can help automate SAS Viya deployments

- across separate instances of SAS Viya deployed to different namespaces in the same cluster.
- in a single namespace

Using the operator is **optional**.

Moving on to additional tools that can support the operation of the SAS Viya platform, we can mention the SAS Viya Deployment Operator.
It is a Kubernetes operator and it can optionally be used to automate the manual tasks required to deploy and update the software.
It can be useful to automate SAS Viya deployments.
You can have a single deployment operator, used across multiple SAS Viya instances to manage multiple deployments, or you can co-locate it within an individual SAS Viya environment to manage only that one.
Using the operator is optional.



Jobs And CronJobs

Jobs are Kubernetes objects that:

- Describe what containers run and how.
- Run immediately and keep the pod running until completion.
- SAS Viya platform Kubernetes jobs have a default time-to-live (TTL) value of 0

CronJobs are Kubernetes objects that include:

- A job template.
- A schedule in the unix *crontab* format.

Jobs can be created and started as a one-off copy of a CronJob.

SAS Viya defines some CronJobs with an impossible schedule (i.e. Feb 30)

- to be used by admins as templates to start jobs on demand.

So far, we discussed about services running in Kubernetes as deployments or stateful sets. SAS Viya provides additional Kubernetes objects, such as Jobs and CronJobs.

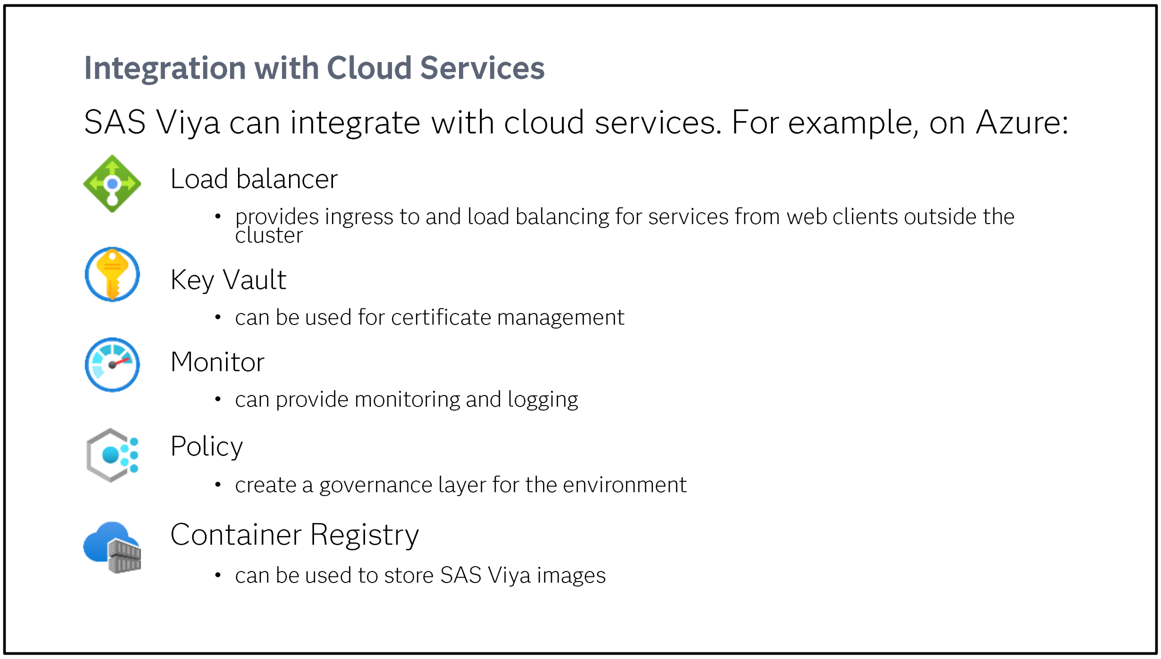
.Jobs are Kubernetes objects that describe what containers to run and how it should run. As soon as you define a Kubernetes job, it starts immediately, and it keeps running until it completes successfully.
SAS Viya uses some jobs at deployment time to configure or create some of the artifacts required by the platform.
Starting with 2024.12, all SAS Viya platform Kubernetes jobs have a default time-to-live (TTL) value of 0. This change was made in order to improve the update experience. The requirement to manually delete certain jobs no longer applies to most deployments.
As a result of this change, Kubernetes jobs that are generated on the SAS Viya platform are automatically deleted once they have run to completion. When jobs are deleted, the associated log data is also deleted. SAS strongly recommends installing a log aggregator to retain and manage log messages from Kubernetes resources that are automatically deleted over time.

CronJobs are similar, but instead of running immediately, they will eventually run when scheduled. A CronJob definition includes a job template, and a schedule in the UNIX crontab format.

Jobs can run as a scheduled CronJob, or you can create a one-off copy of a CronJob, that is started immediately.

SAS Viya includes some CronJobs defined with an impossible schedule, such as February 30, to be used by SAS administrators as templates to perform a specific task when required. As an example, there are CronJobs that can be copied as Jobs to start and stop the SAS Viya environment.
CronJobs, such as sas-start-all, sas-backup-job, and sas-restore-job, do not have a TTL value of 0, and

their logs are retained at completion.



Integration with Cloud Services

SAS Viya can integrate with cloud services. For example, on Azure:

Load balancer

- provides ingress to and load balancing for services from web clients outside the cluster

Key Vault

- can be used for certificate management

Monitor

- can provide monitoring and logging

Policy

- create a governance layer for the environment

Container Registry

- can be used to store SAS Viya images

Finally, since we are discussing many third-party components, it's important to remember that SAS Viya can optionally integrate with services provided by the cloud provider where it is running.
Here you can find some examples based on Microsoft Azure, but the same could be said on Amazon or on Google.

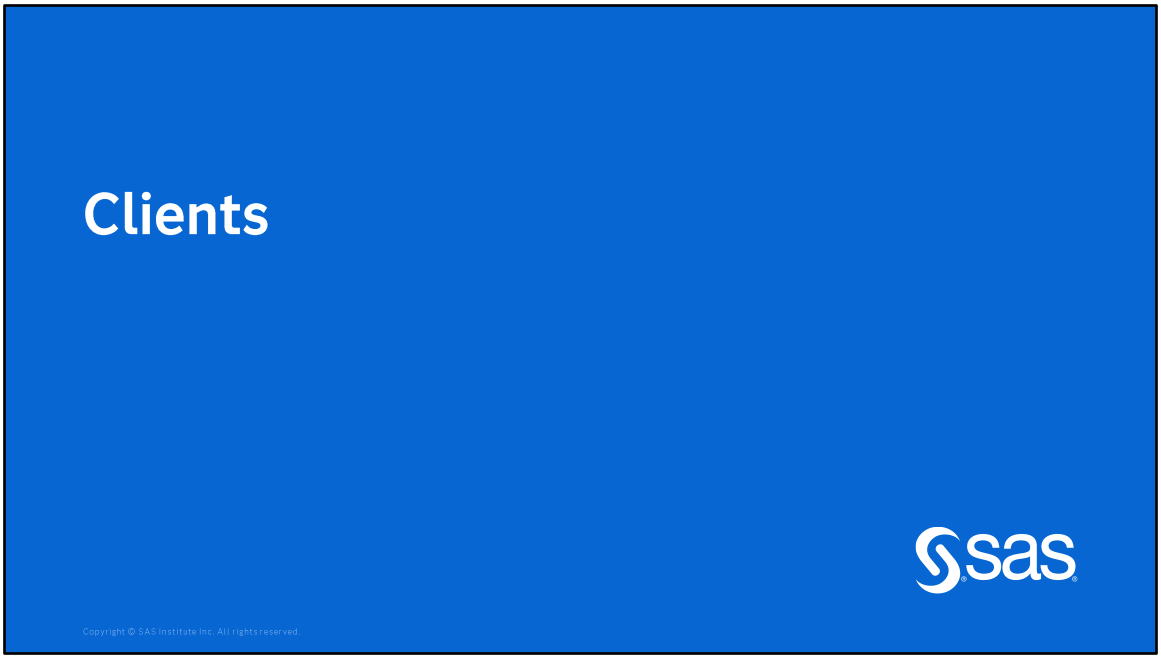
As soon as SAS Viya defines an external Kubernetes service, Azure creates a load balancer to provide ingress and load balancing to it.

You can optionally decide to use Azure Key Vault for certificate management. Customer-provided certificates can be stored using Azure Key Vault to simplify their lifecycle.

Azure Monitor – which provides Azure-native monitoring and logging tools - can be used instead of the SAS Viya Monitoring for Kubernetes framework.

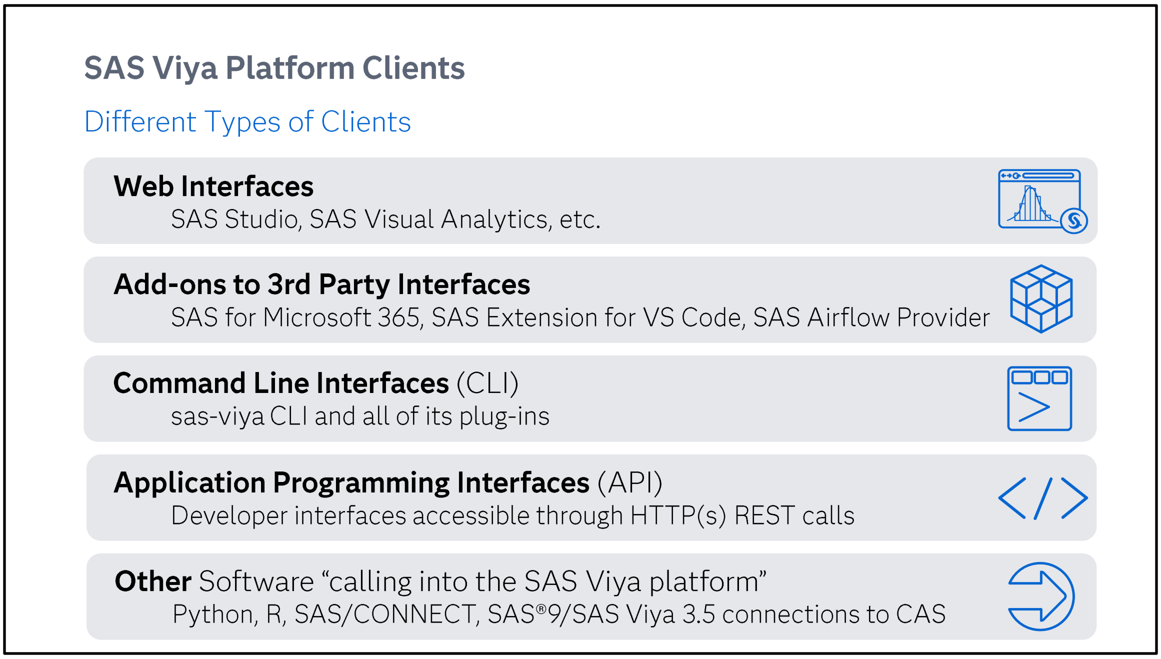
You can implement Azure policies to create a governance layer on top of your SAS Viya deployment.

You can use the Azure Container Registry to mirror SAS Viya container images.
The list could continue with many more cloud services.



Clients

Let’s describe some of the clients of the SAS Viya platform, highlighting the main non-functional differences from previous versions.



SAS Viya Platform Clients

Different Types of Clients

Web Interfaces
SAS Studio, SAS Visual Analytics, etc.

Add-ons to 3rd Party Interfaces
SAS for Microsoft 365, SAS Extension for VS Code, SAS Airflow Provider

Command Line Interfaces (CLI)
sas-viya CLI and all of its plug-ins

Application Programming Interfaces (API)
Developer interfaces accessible through HTTP(s) REST calls

Other Software “calling into the SAS Viya platform”
Python, R, SAS/CONNECT, SAS®9/SAS Viya 3.5 connections to CAS

Applications in the SAS Viya platform can be accessed by a variety of clients.

The most notable ones are web interfaces, such as SAS Studio, SAS Visual Analytics, SAS Model Studio, and many other web applications.

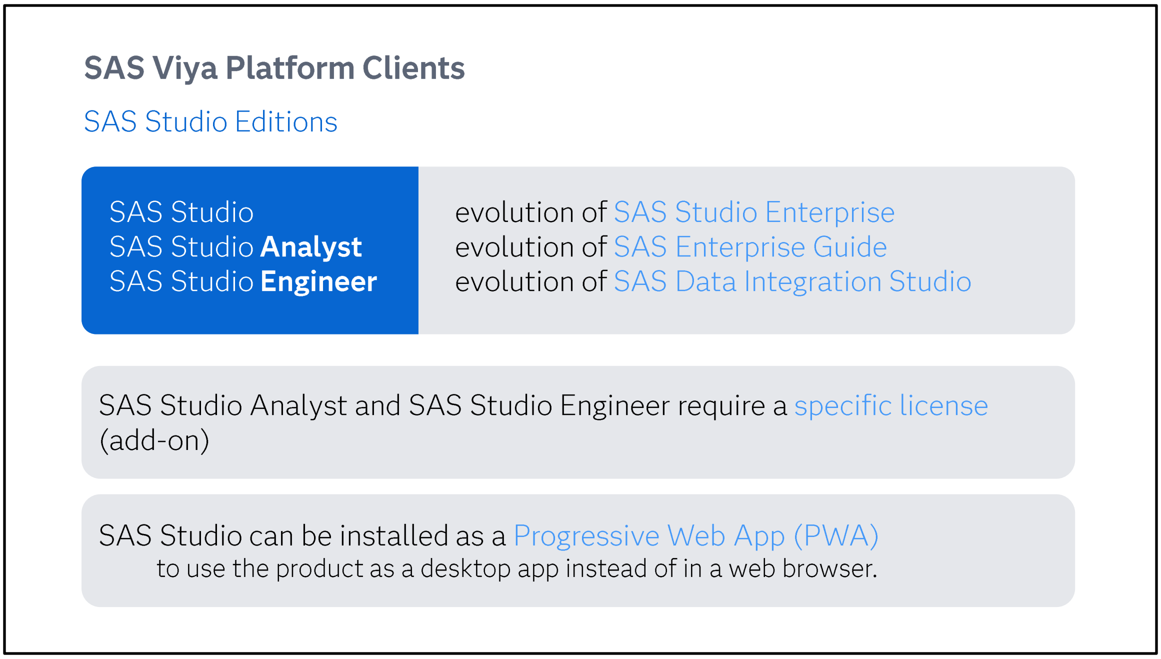
You can also leverage SAS Viya capabilities through add-ons to third-party interfaces. Examples include SAS for Microsoft 365, the SAS extension for Visual Studio Code, and the SAS Airflow Provider.

Then, there are command line interfaces such as the SAS Viya CLI, with all of its plug-ins. This is mostly for administrators, although some plug-ins, such as the SAS Batch CLI, are geared toward end-users.

SAS Viya applications provide application programming interfaces, or APIs, so that developers and third-party applications can interact with the SAS Viya Platform through HTTPS REST calls.

Finally, let’s not forget any other software that can call into the SAS Viya Platform, such as code written Python or R, and SAS Connect clients. Also, SAS 9 and other SAS Viya environments connecting into CAS can be classified as clients.

In the rest of the lesson, we will cover additional details about the first two client categories: web interfaces, and add-ons to third-party interfaces.



SAS Viya Platform Clients

SAS Studio Editions

SAS Studio	evolution of SAS Studio Enterprise
SAS Studio **Analyst**	evolution of SAS Enterprise Guide
SAS Studio **Engineer**	evolution of SAS Data Integration Studio

SAS Studio Analyst and SAS Studio Engineer require a specific license (add-on)

SAS Studio can be installed as a Progressive Web App (PWA)
to use the product as a desktop app instead of in a web browser.

The main programming client is SAS Studio.
SAS Studio, in the current SAS Viya platform, is available in three different editions.
SAS Studio, in its base edition, is the evolution of SAS Studio Enterprise available with previous SAS Viya versions.
SAS Studio Analyst can be considered the evolution of SAS Enterprise Guide, while SAS Studio Engineer is the evolution of SAS Data Integration Studio.

While the SAS Studio application that is deployed is the same between the three editions, capabilities included with SAS Studio Analyst or SAS Studio Engineer are enabled by a specific license. The base SAS Studio capabilities are included by default in almost every SAS Viya offering.

SAS Studio can also be installed as a progressive web application, so that the product can be run as a desktop application instead of running in your browser.

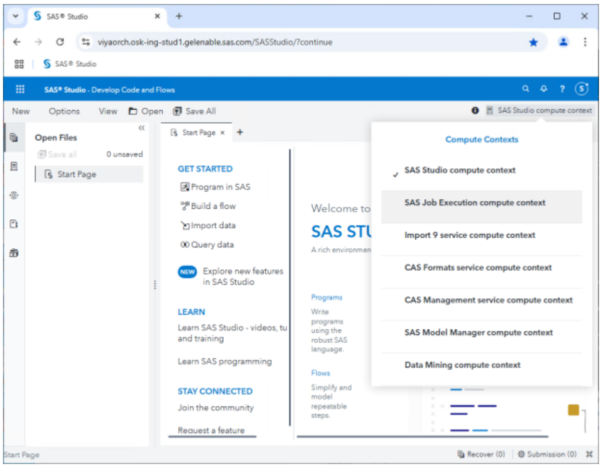
SAS Viya Platform Clients

SAS Studio non-functional Changes from Previous Editions

All SAS Studio editions are available from a single web application – different capabilities are enabled by the license.

Optimization: the interface is ready immediately, while the backend Compute Server starts asynchronously.

You can choose the Compute Server context to use.


SAS® Studio
viyaorch.osk-ing-stud1.gelenable.sas.com/SASStudio/?continue
SAS® Studio
SAS® Studio - Develop Code and Flows
New
Options
View
Open
Save All
SAS Studio compute context
Open Files
Save all
0 unsaved
Start Page
Start Page
GET STARTED
Program in SAS
Build a flow
Import data
Query data
NEW
Explore new features in SAS Studio
LEARN
Learn SAS Studio - videos, tu and training
Learn SAS programming
STAY CONNECTED
Join the community
Request a feature
Welcome to
SAS STU
A rich environmen
Programs
Write programs using the robust SAS language.
Flows
Simplify and model repeatable steps.
Compute Contexts
✓ SAS Studio compute context
SAS Job Execution compute context
Import 9 service compute context
CAS Formats service compute context
CAS Management service compute context
SAS Model Manager compute context
Data Mining compute context
Start Page
Recover (0)
Submission (0)

SAS Studio provides many improvements compared to previous versions. Here we want to highlight a few of interest to architects and deployment engineers.

First of all, although we have seen that there are three different editions of SAS Studio, they are all available as a single web application, in which the different capabilities are enabled by the specific license. Instead, in previous versions, different versions of SAS Studio required the deployment of different applications.
SAS 9 platforms include SAS Studio 3 Personal, Basic and Enterprise.
SAS Viya 3.5 includes SAS Studio Basic and SAS Studio Enterprise. Earlier SAS Viya releases include both SAS Studio 4 and SAS Studio 5.
SAS Viya 4 includes only one SAS Studio application, the evolution of SAS Studio Enterprise.

SAS Studio is optimized so that the interface is immediately available as soon as users log in. With previous versions, the application was not ready until the back-end server was up and running. Now, you do not need the back-end server until you are ready to submit your code.

Notice how, when you start SAS Studio in SAS Viya 4, you can see a spinning circle next to the SAS Studio Compute Server label, to indicate that the backend is initializing.

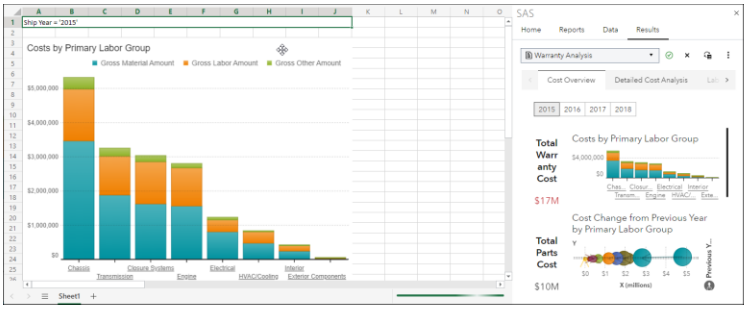
Also, in terms of back-end, you can now choose which compute contexts to use, between all of the ones defined in the environment. With previous SAS Viya versions, you only had one fixed compute context for SAS Studio.
After the compute backend is started, the highlighted label should indicate which context is used. If your environment has multiple contexts, you can switch between contexts by clicking there.

SAS Viya Platform Clients

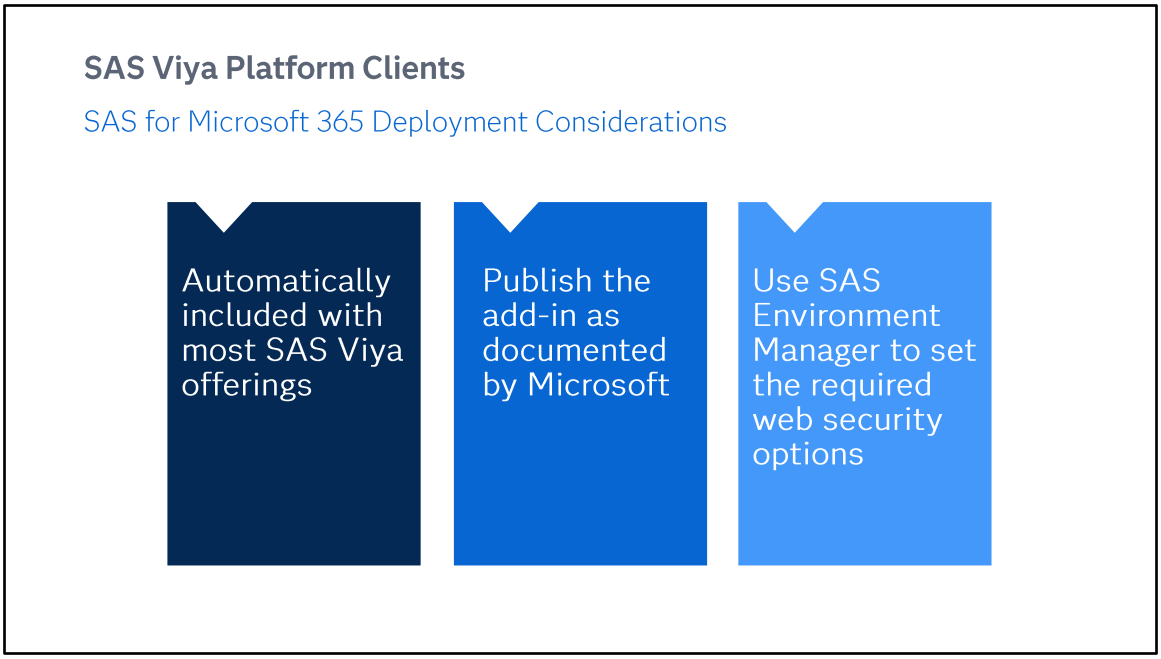
SAS for Microsoft 365

Can run in Microsoft Excel, Outlook, PowerPoint, Word

- on the **desktop**
- on the **web**



Let’s move on to the next client, SAS for Microsoft 365.
This is an integration that can run in Microsoft Excel, Outlook, PowerPoint, and Word.
It can work with the desktop versions, and with the web ones.
Following a continuous delivery methodology, initially it was only available in Excel, then, with subsequent SAS Viya releases it gained new capabilities and new integrations with Microsoft software.



SAS Viya Platform Clients

SAS for Microsoft 365 Deployment Considerations

Automatically included with most SAS Viya offerings

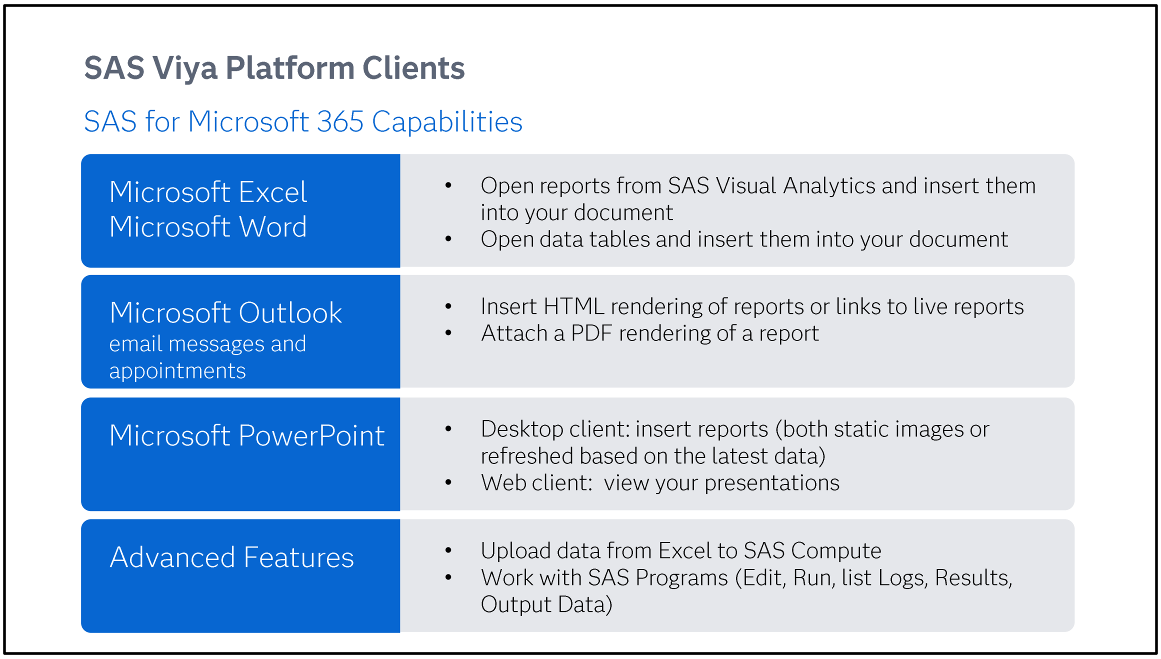
Publish the add-in as documented by Microsoft

Use SAS Environment Manager to set the required web security options

SAS for Microsoft 365 is automatically included with SAS Visual Analytics, and all offerings that include it – except SAS Viya Programming.

Given its integration with Microsoft software, the way it is deployed and configured depends on Microsoft requirements.
The web application that is deployed with SAS Viya is not integrated until you follow the documentation, published by Microsoft, about how to integrate a third-party plugin, such as this one, into the Microsoft ecosystem.

Once that is done, then you can use SAS Environment Manager to set the required web security options. This is all detailed in the deployment instructions.

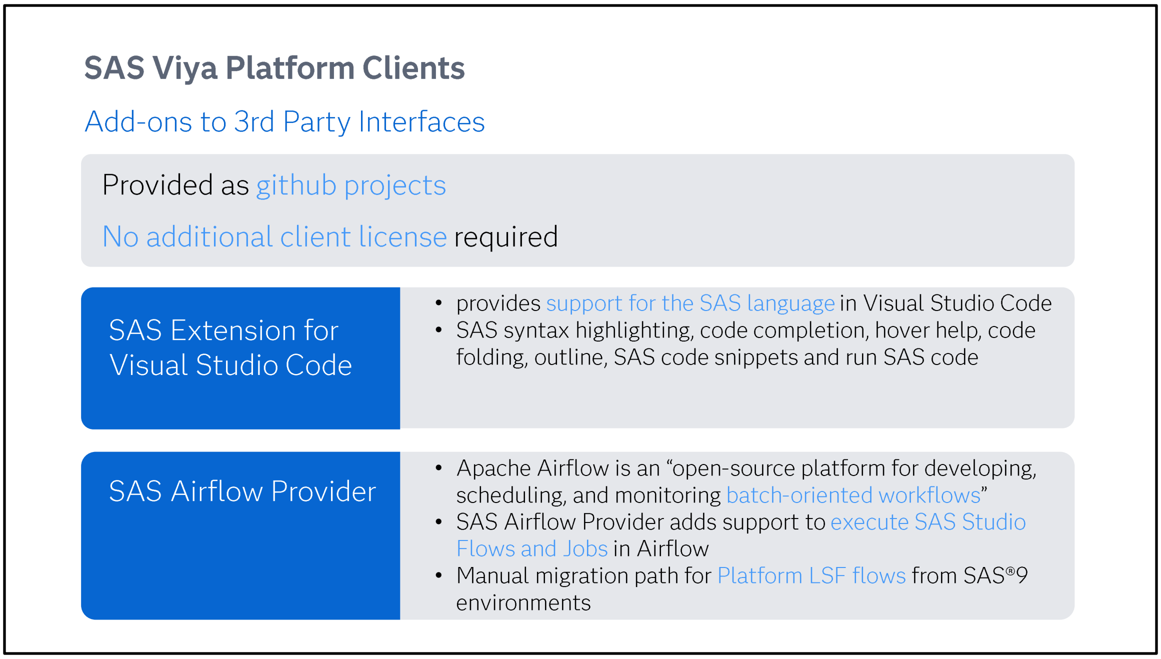


SAS Viya Platform Clients

SAS for Microsoft 365 Capabilities

Microsoft Excel Microsoft Word	• Open reports from SAS Visual Analytics and insert them into your document • Open data tables and insert them into your document
Microsoft Outlook email messages and appointments	• Insert HTML rendering of reports or links to live reports • Attach a PDF rendering of a report
Microsoft PowerPoint	• Desktop client: insert reports (both static images or refreshed based on the latest data) • Web client: view your presentations
Advanced Features	• Upload data from Excel to SAS Compute • Work with SAS Programs (Edit, Run, list Logs, Results, Output Data)

SAS for Microsoft 365 offers different capabilities based on whether you are using Excel, Word, Outlook, or PowerPoint.
All these Microsoft applications can open reports from Visual Analytics, rendering them either as HTML or as native views. Some applications can interact with data. Then, there are some advanced features, for example, to upload data from Excel to SAS Viya, or to work with SAS programs directly in Office.



SAS Viya Platform Clients

Add-ons to 3rd Party Interfaces

Provided as github projects

No additional client license required

SAS Extension for Visual Studio Code

- provides support for the SAS language in Visual Studio Code
- SAS syntax highlighting, code completion, hover help, code folding, outline, SAS code snippets and run SAS code

SAS Airflow Provider

- Apache Airflow is an "open-source platform for developing, scheduling, and monitoring batch-oriented workflows"
- SAS Airflow Provider adds support to execute SAS Studio Flows and Jobs in Airflow
- Manual migration path for Platform LSF flows from SAS®9 environments

SAS Viya provides additional add-ons to third party interfaces. These are provided as GitHub projects, which means you can download and install them without any additional client license. Obviously, you need a licensed SAS Viya environment to connect to from these clients.

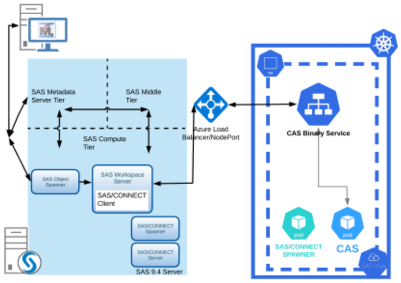
The SAS extension for Visual Studio code, provides support for the SAS language in Visual Studio code. If you are used to this coding interface, you can use it to write SAS code as well, with all the default coding intelligence available with other languages.

You can also download the SAS Airflow Provider, that is a plugin provider for Apache Airflow. Apache Airflow is an open-source platform for deploying, scheduling, and monitoring batch workflows. The SAS Airflow Provider adds support to execute SAS Studio flows and jobs in an Airflow environment. Integrating SAS Viya with Apache Airflow is a migration path for Platform LSF flows that you may have in SAS 9 environments.

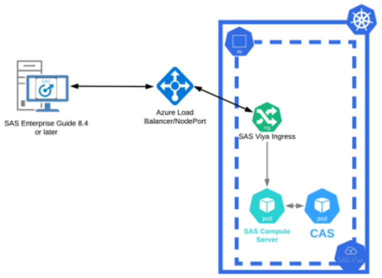
SAS Viya Platform Clients

The SAS Viya platform does not have any native desktop client.

Desktop clients from previous releases require a **workspace** or **connect server** running in the legacy environment to connect to SAS Viya.



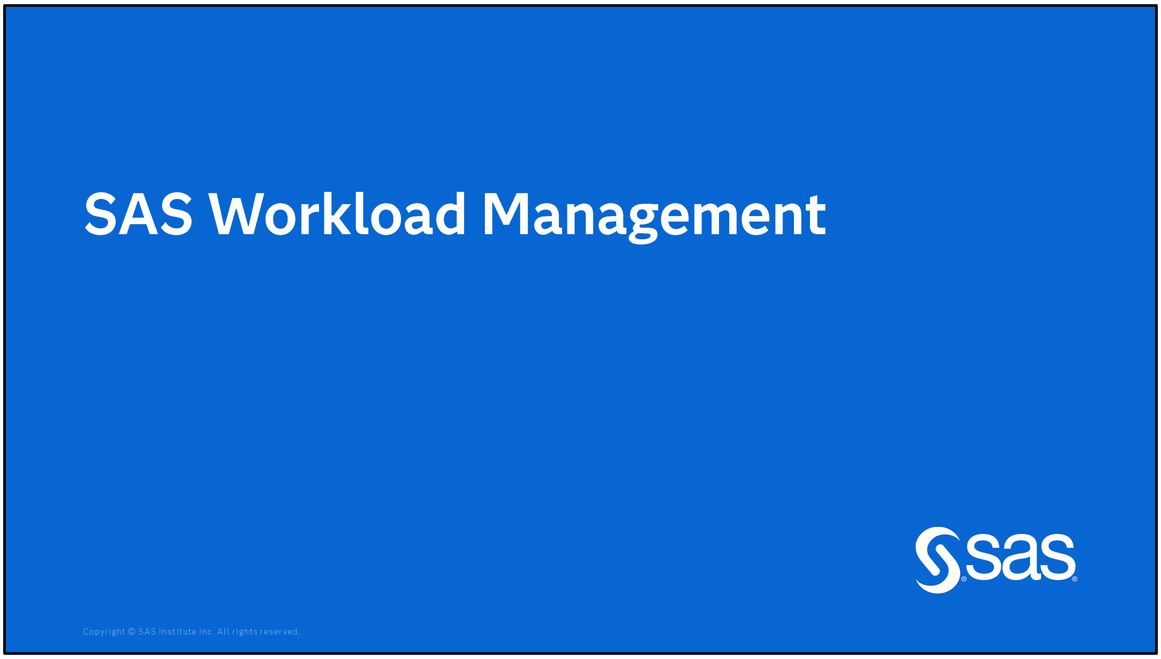
SAS Enterprise Guide 8.4 or later supports direct connections to SAS Viya compute servers via REST API.


SAS Enterprise Guide 8.4 or later
Azure Load Balancer/NodePort
SAS Viya Ingress
SAS Compute Server
CAS

The SAS Viya platform does not have any native desktop client.
This is a significative difference when you compare SAS 9 environments to SAS Viya.

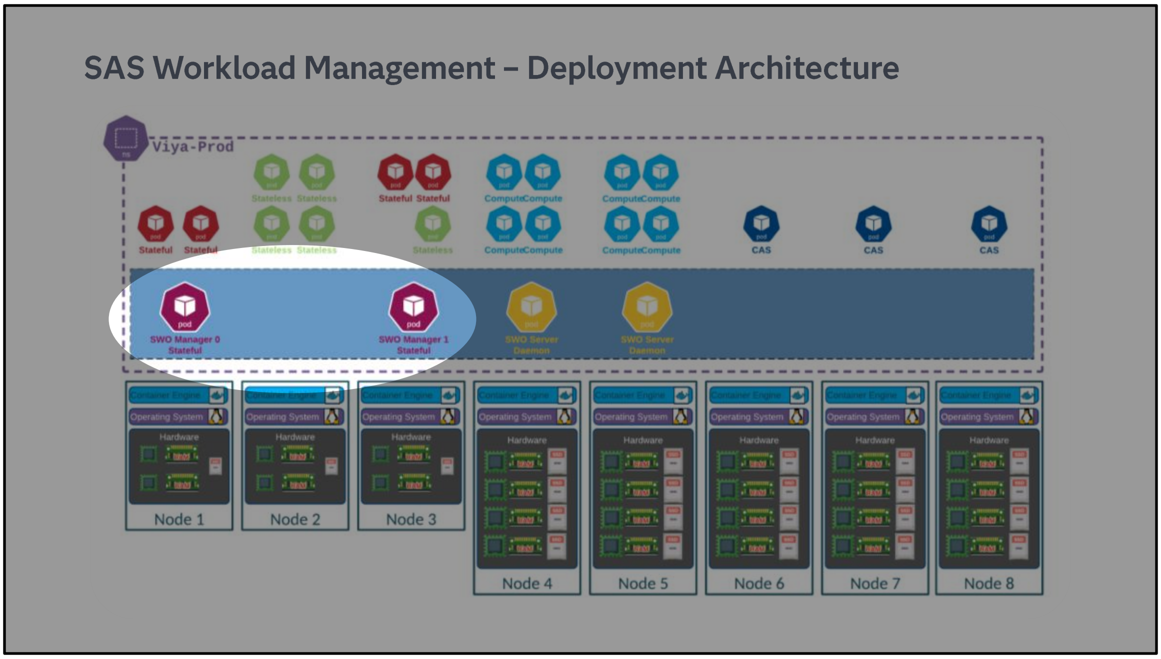
Most desktop clients that were available with SAS 9, such as SAS Data Integration Studio, cannot connect directly to SAS Viya. The reason is that SAS 9 clients use the proprietary IOM protocol to communicate, but that protocol is not available with SAS Viya.
You can still connect from these clients to SAS Viya, by deploying a full SAS 9 environment, then connecting through a SAS 9 server, which acts as a proxy between the client and the SAS Viya environment. In this case, you can use SAS 9 workspace servers or SAS 9 connect servers.

The only exception is SAS Enterprise Guide. Starting from version 8.4 and later, it supports the SAS Viya REST protocol, so you can use it without any proxy SAS 9 server. SAS Enterprise Guide can directly connect to SAS Viya compute servers REST API interfaces.



SAS Workload Management

In this lesson you can learn about the SAS Workload Management architecture, with a high-level overview of how SAS Workload Management manages compute jobs.
This is not a full class on SAS Workload Management.



SAS Workload Management – Deployment Architecture

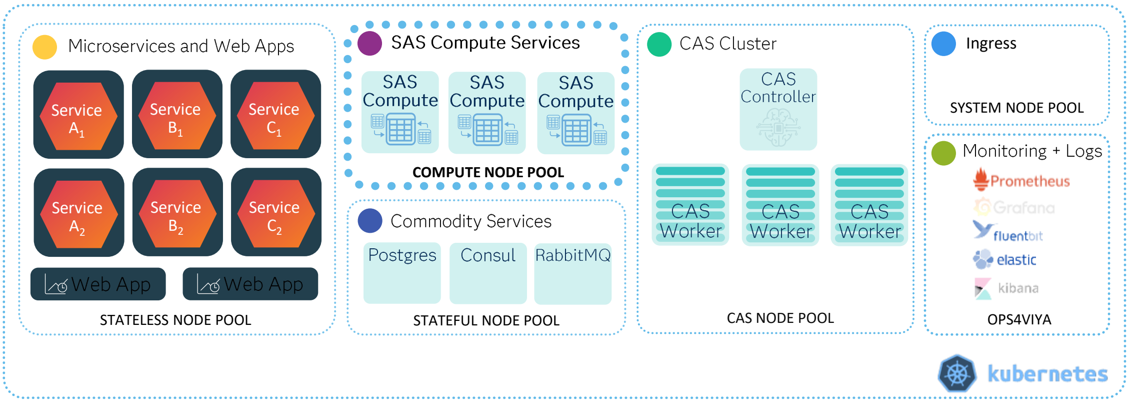
SAS Workload Management extends the workload-management capabilities of Kubernetes by adding priority-based queues.
These queues provide the ability to control where and when jobs are processed based on prioritization and resource allocation.

A key difference between the current SAS Viya release and previous versions, including SAS 9, is that SAS Workload Management does not require a different architecture design or the deployment of additional software. Almost all SAS Viya offerings include by default SAS Workload Management. You can then choose to enable it (which is the default) or disable it.
When SAS Workload Management is disabled, its pods are shut down and you will not find them running in the Kubernetes cluster.

SAS Workload Management includes a central server, called SAS Workload Orchestrator Manager, running on stateful nodes. By default, there are two pods for high availability; one is active, the other is in standby. The server is the brain of the workload management activity.

The other components are server daemons running as Kubernetes daemon sets on the compute nodes. Each pod manages and monitors the jobs running on its node, and is in constant communication with the SAS Workload Orchestrator Manager.

SAS Workload Management – focused on SAS Compute



Microservices and Web Apps

Service A_1 Service B_1 Service C_1

Service A_2 Service B_2 Service C_2

Web App Web App

STATELESS NODE POOL

SAS Compute Services

SAS Compute SAS Compute SAS Compute

COMPUTE NODE POOL

Commodity Services

Postgres Consul RabbitMQ

STATEFUL NODE POOL

CAS Cluster

CAS Controller

CAS Worker CAS Worker CAS Worker

CAS NODE POOL

Ingress

SYSTEM NODE POOL

Monitoring + Logs

Prometheus

Grafana

fluentbit

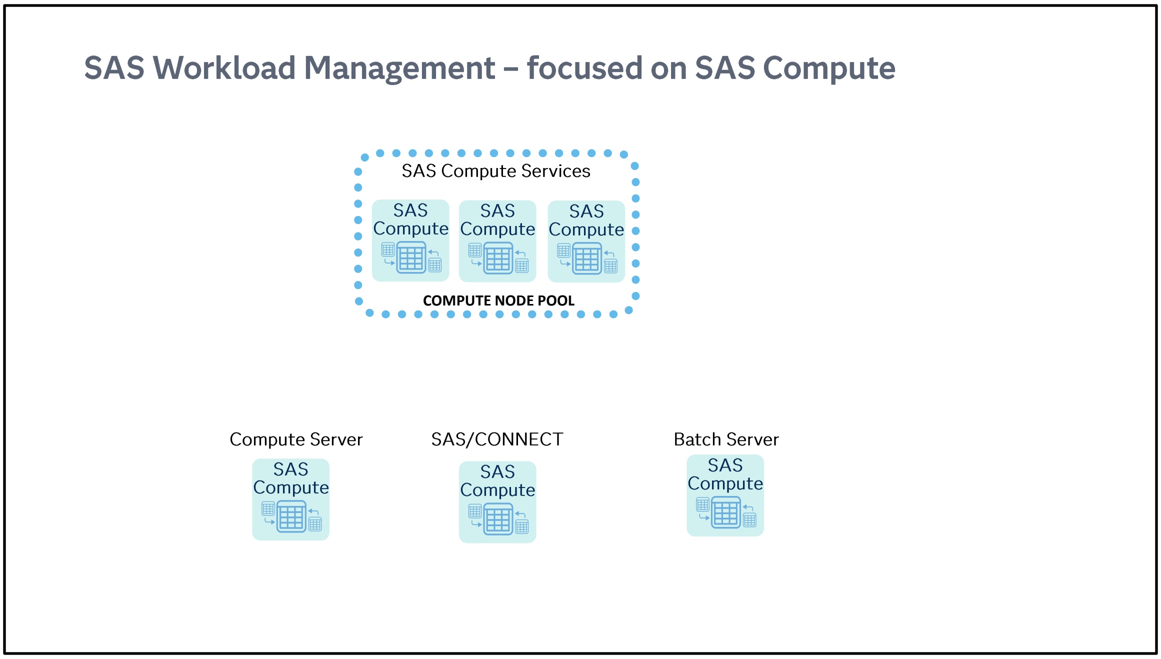
elastic

kibana

OPS4VIYA

kubernetes

Another key point is that SAS Workload Orchestrator can only manage SAS compute jobs.
We have seen that SAS Viya is composed by multiple components. SAS Workload Orchestrator will not manage the workload created by the middle-tier stateless components, nor the stateful services. It will not even manage CAS workloads.



SAS Workload Management – focused on SAS Compute

SAS Compute Services

SAS Compute

SAS Compute

SAS Compute

COMPUTE NODE POOL

Compute Server

SAS Compute

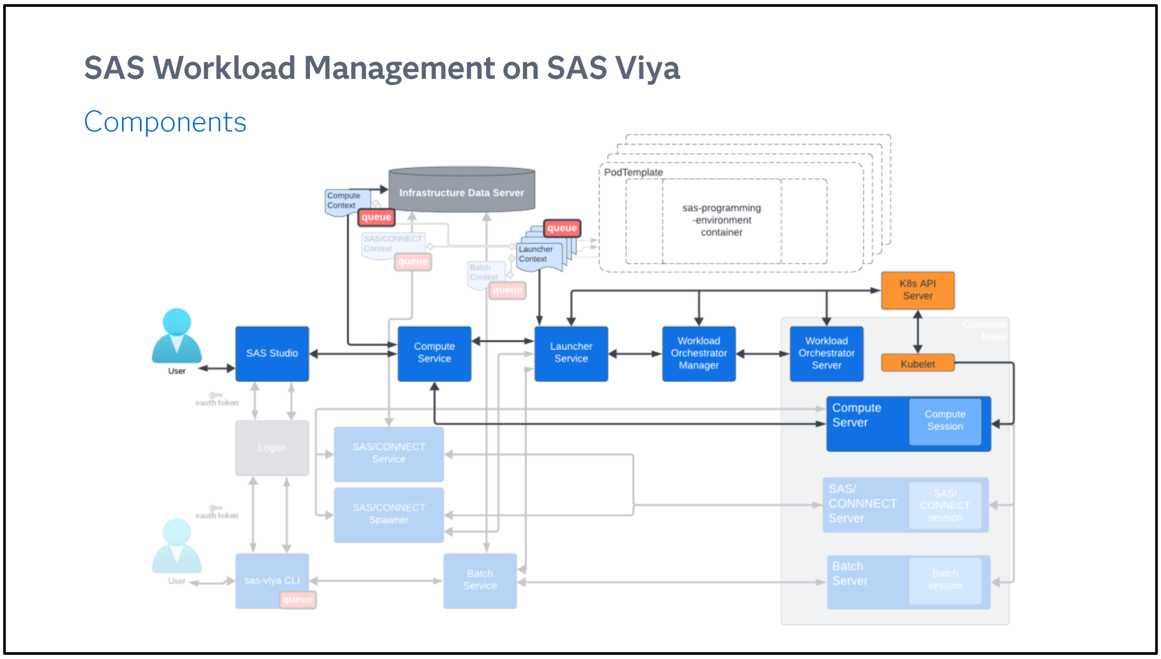
SAS/CONNECT

SAS Compute

Batch Server

SAS Compute

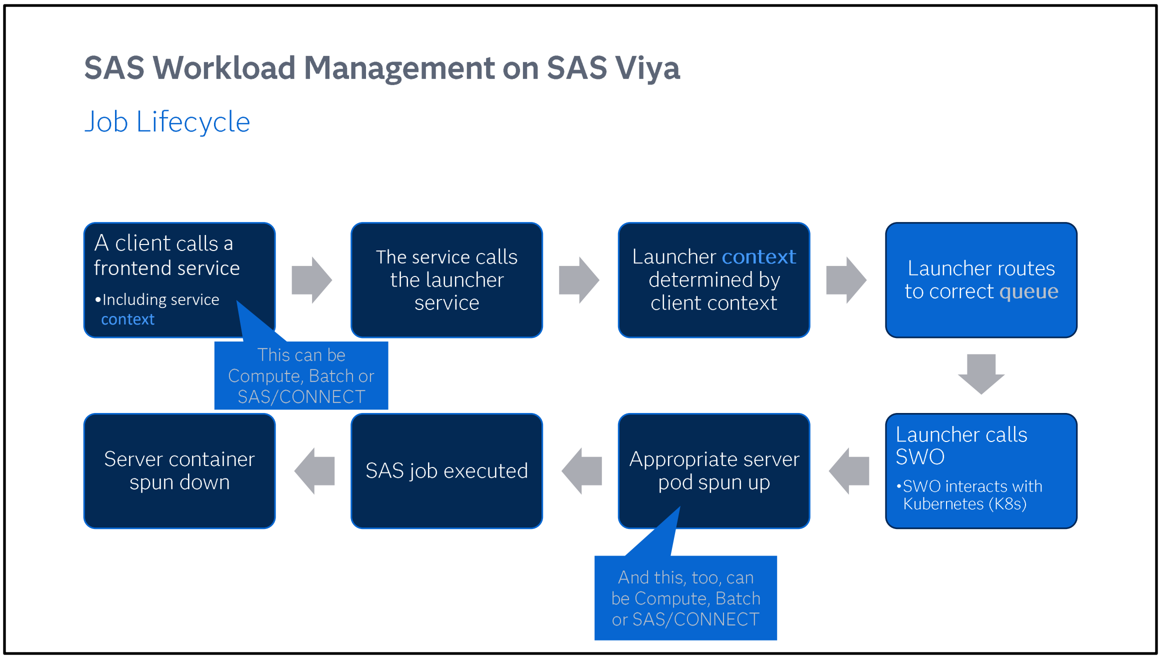
The only components that SAS Workload Orchestrator can manage are compute servers.
Whether it is a compute server, connect server,
or batch server, they can all use SAS Workload Orchestrator queues. These are the only components that are managed by SAS Workload Orchestrator.



SAS Workload Management on SAS Viya

Components

In terms of components, you might remember this diagram that we have introduced in another lesson. It highlights the components involved in a sample use case, a client such as SAS Studio that starts a backend compute server.
The client talks to a middle-tier compute service, which talks to a launcher, and then the ball rolls to the workload orchestrator manager, and finally, to the workload orchestrator server on a compute node.
SAS Workload Orchestrator queues can be linked, by an administrator, to a server context, or can be requested, by a user, on the CLI submitting a batch job.



SAS Workload Management on SAS Viya

Job Lifecycle

A client calls a frontend service

- Including service context

This can be Compute, Batch or SAS/CONNECT

The service calls the launcher service

Launcher context determined by client context

Launcher routes to correct queue

Launcher calls SWO

- SWO interacts with Kubernetes (K8s)

Appropriate server pod spun up

And this, too, can be Compute, Batch or SAS/CONNECT

SAS job executed

Server container spun down

Let’s highlight a typical job lifecycle.

A client, for example, SAS Studio calls a frontend service sending in a job to execute.
This could be a compute, batch, or connect service. The client call includes the request to use a specific context, and the context might include the request for a specific queue.

The frontend service calls the launcher microservice, including the requested context.

The launcher context, at this point, is determined by this client request. This context might include the specification of a queues, as well.

The launcher service analyzes the incoming request and all the contexts definitions and assigns the job to a queue. If there is no explicit request for a queue, then a default queue is assigned implicitly.

The launcher calls SAS Workload Orchestrator, which looks at the requested queue and applies all its configured policies in terms of priorities, resources requested, nodes requested, and, based on that, it assigns the job to an appropriate node that can satisfy the request.

The SAS Workload orchestrator sends to the Kubernetes API the command spin up a new pod on the selected node.

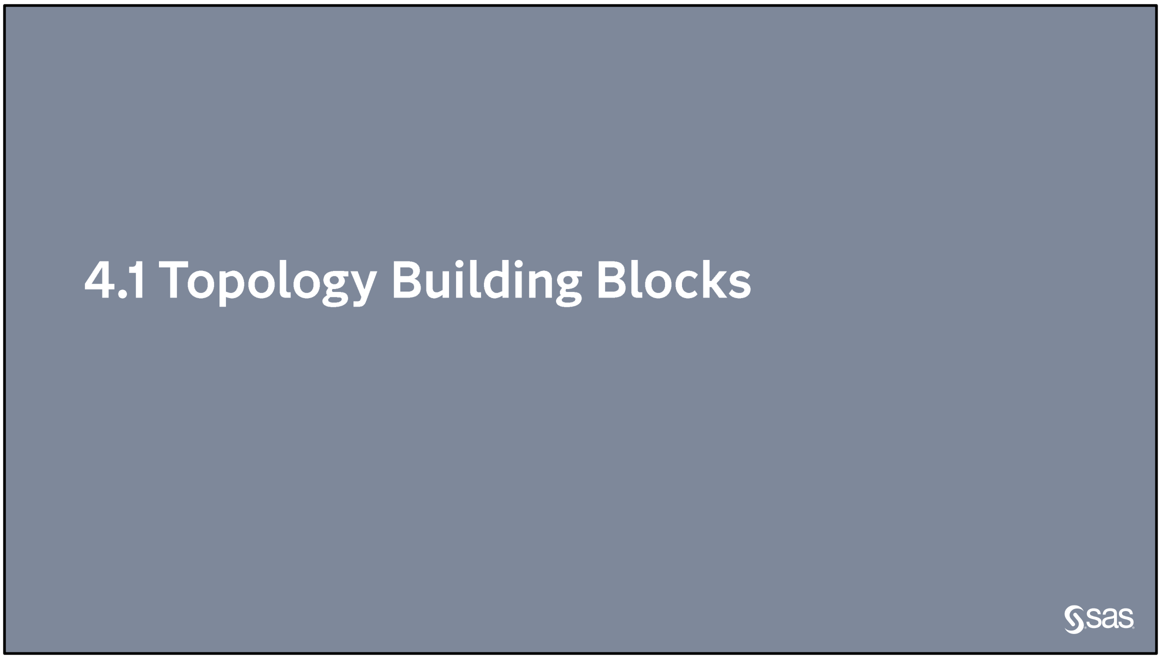
And this, too, can be a Compute, Batch or SAS/CONNECT server pod.

At this point, the SAS job is executed in the new server pod.

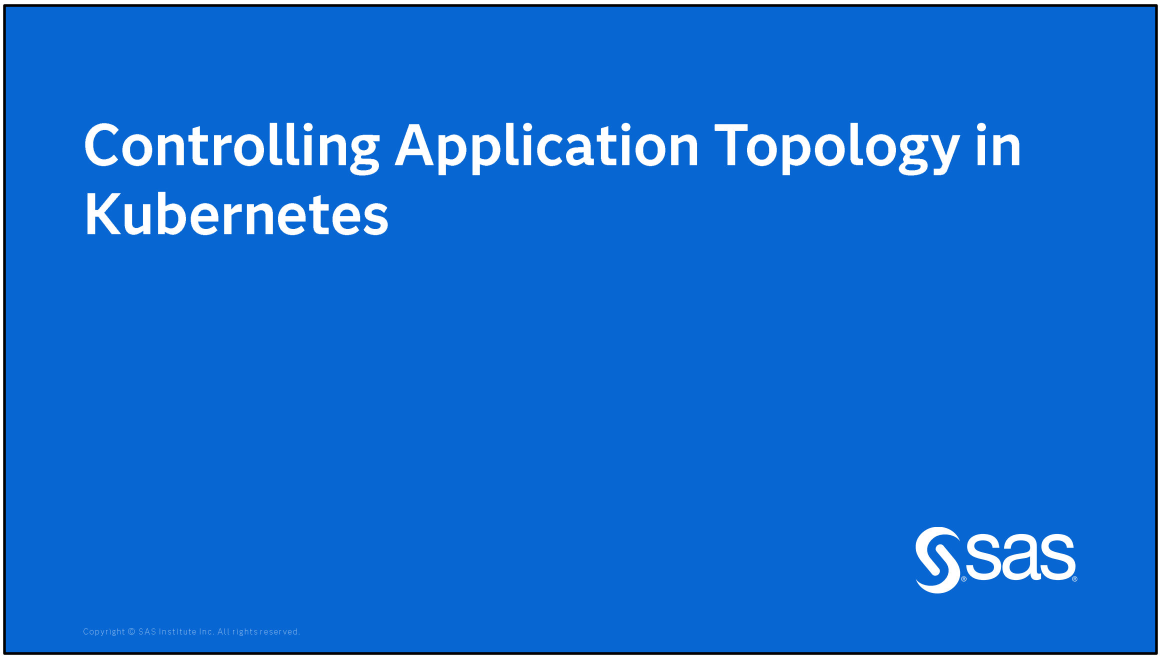
In the end, the pod is spun down.

The two highlighted steps on the right are specific to SAS Workload Management.

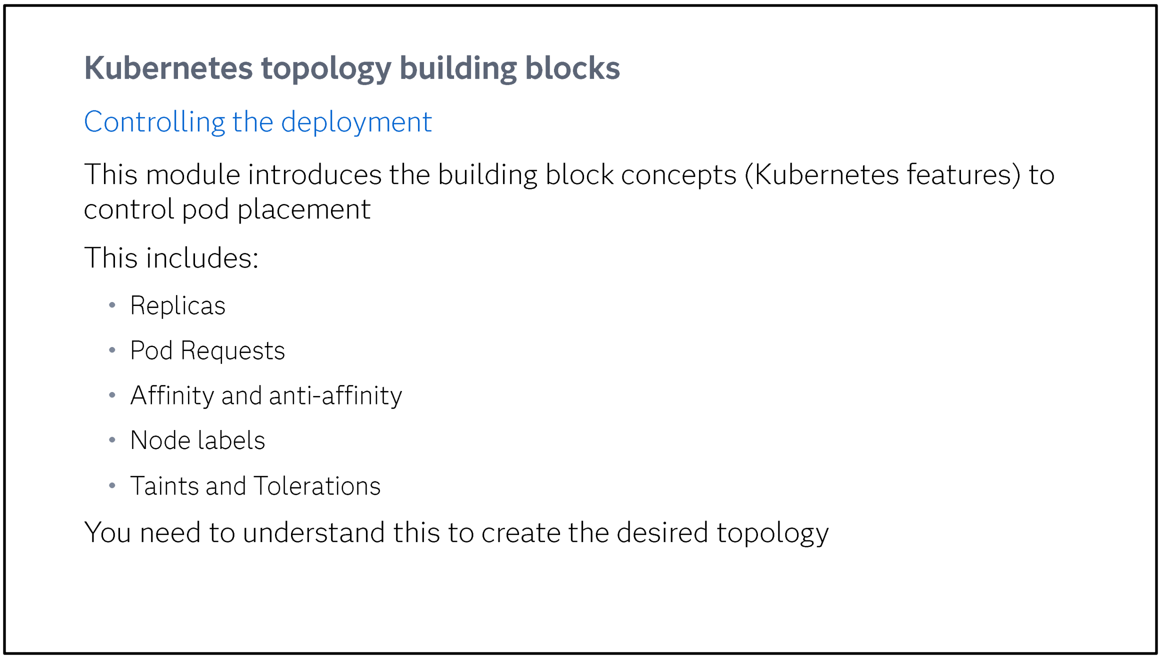
Lesson 4: Environment Topologies



4.1 Topology Building Blocks



Controlling Application Topology in Kubernetes



Kubernetes topology building blocks

Controlling the deployment

This module introduces the building block concepts (Kubernetes features) to control pod placement

This includes:

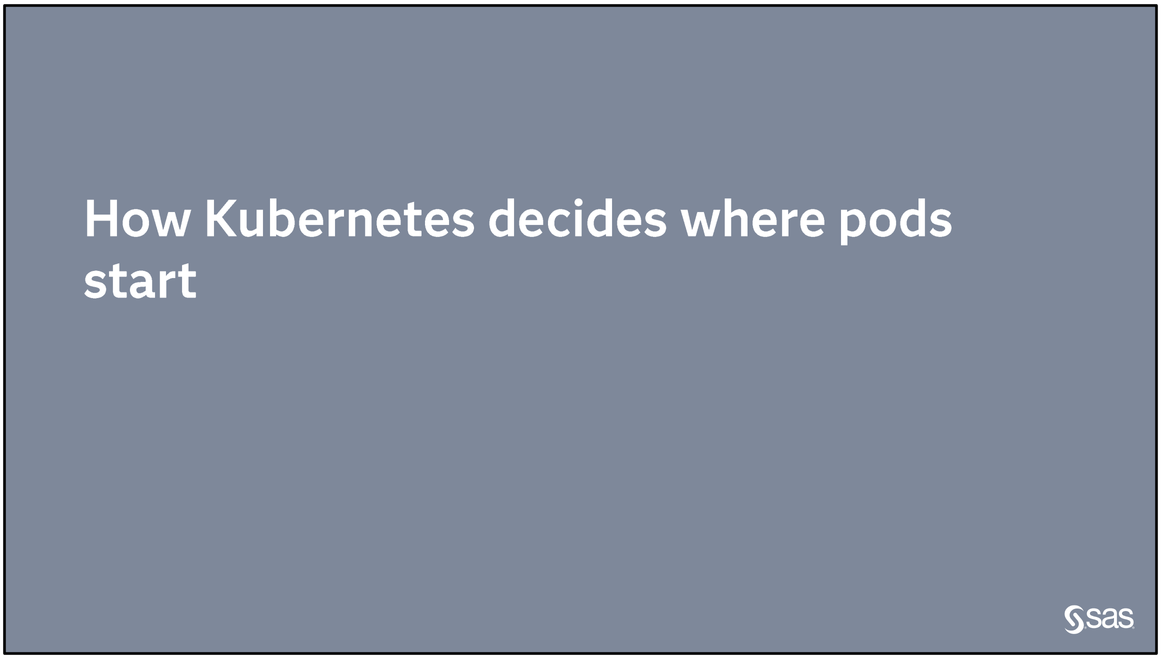
- Replicas
- Pod Requests
- Affinity and anti-affinity
- Node labels
- Taints and Tolerations

You need to understand this to create the desired topology

This module introduces the core Kubernetes concepts, the Kubernetes features, that you need to understand to control the deployment of the SAS Viya platform.

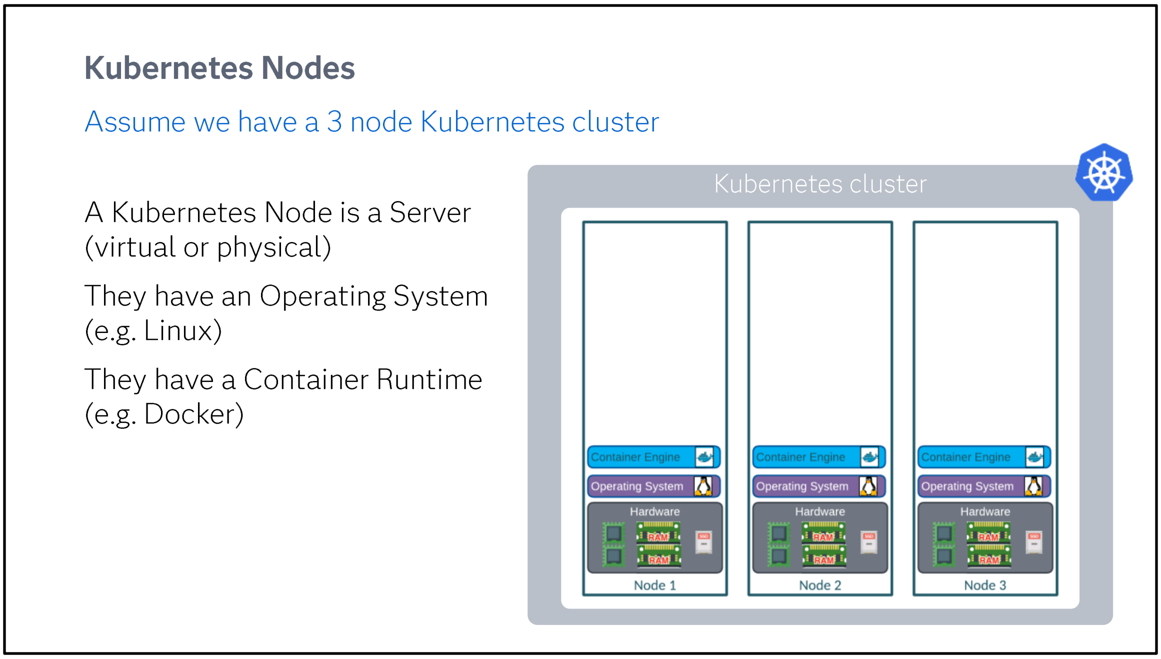
Kubernetes is a complex platform, this module will focus on the following:
Pod replicas.
Pod requests.
Affinity and anti-affinity. This can be considered at two levels; at the node level, and between pods.
Also, the role that node labels, taints and tolerations play.

You need to understand these concepts to create the desired topology for the deployment.



How Kubernetes decides where pods start

We will start by discussing how Kubernetes schedules a pod on a node.



Kubernetes Nodes

Assume we have a 3 node Kubernetes cluster

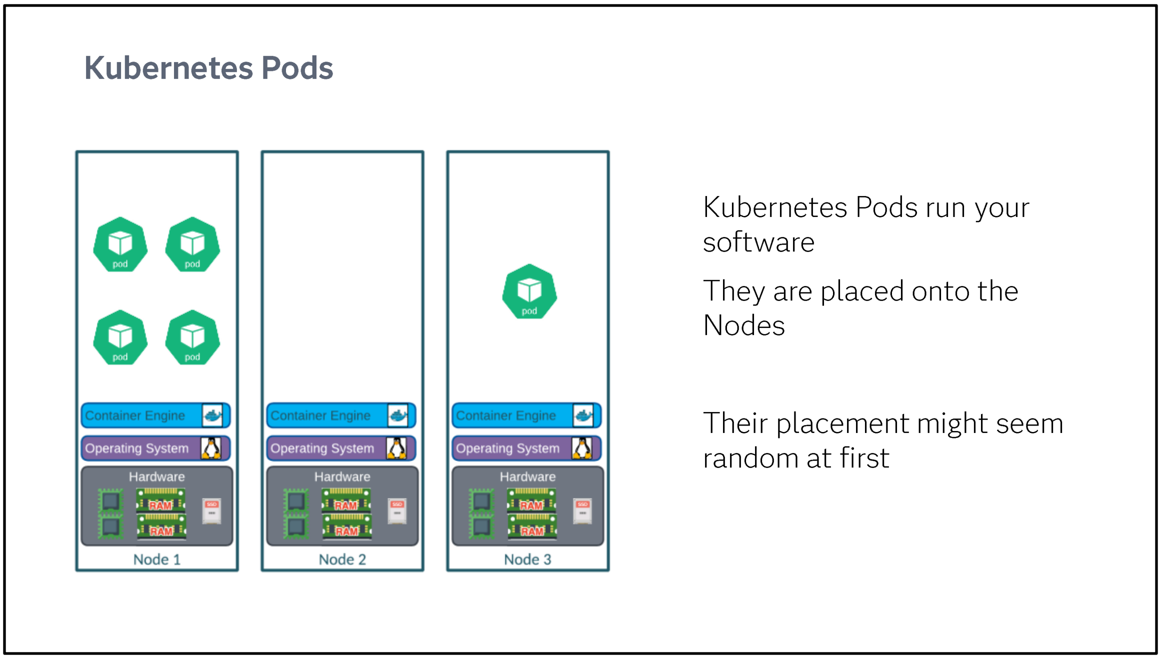
A Kubernetes Node is a Server (virtual or physical)

They have an Operating System (e.g. Linux)

They have a Container Runtime (e.g. Docker)

Here is a Kubernetes cluster, it has three nodes. A Kubernetes node is really just a server. It could be physical or virtual.

The servers, the nodes, have an operating system and a container runtime. For example, Docker.



Kubernetes Pods

Kubernetes Pods run your software

They are placed onto the Nodes

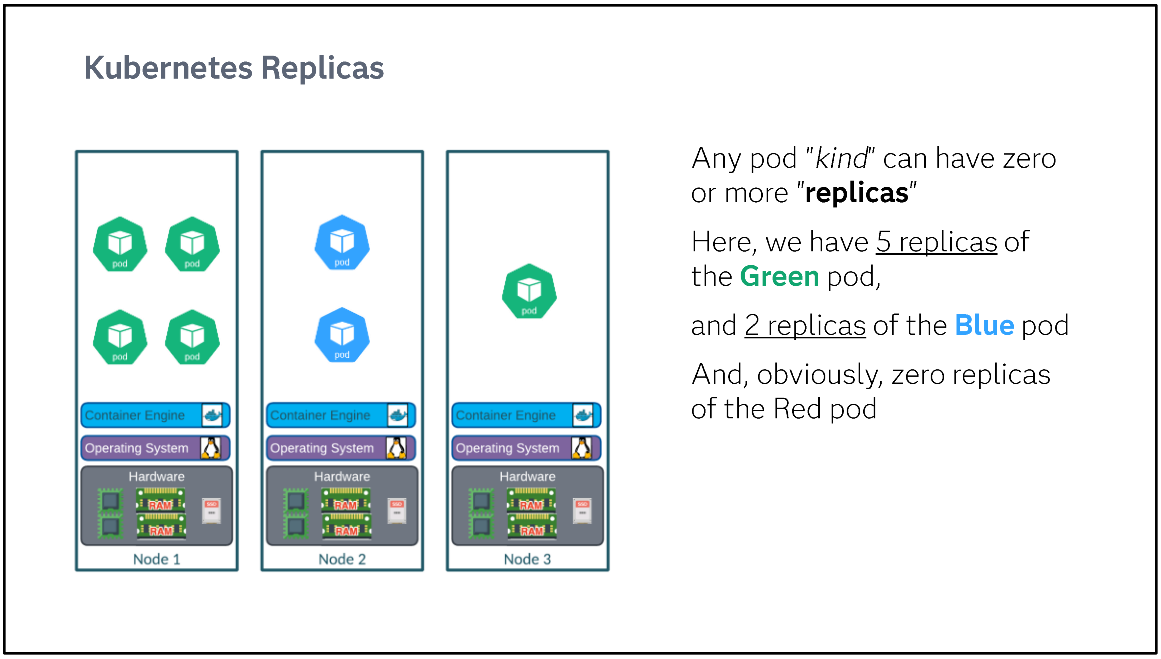
Their placement might seem random at first

Kubernetes Pods.

The Kubernetes pod is used to run the software.

A pod can contain one or more container images. Within the context of this workshop, the pods contain the SAS Viya application containers. There are also other pods running the supporting application components. Such as the ingress controller or a storage provisioner.

The pods are placed, or scheduled, onto the nodes. However, the pod placement might seem random at first. Here you can see four pods were started on Node One. And a fifth pod was started on Node Three.



Kubernetes Replicas

Any pod "*kind*" can have zero or more "**replicas**"

Here, we have 5 replicas of the **Green** pod,

and 2 replicas of the **Blue** pod

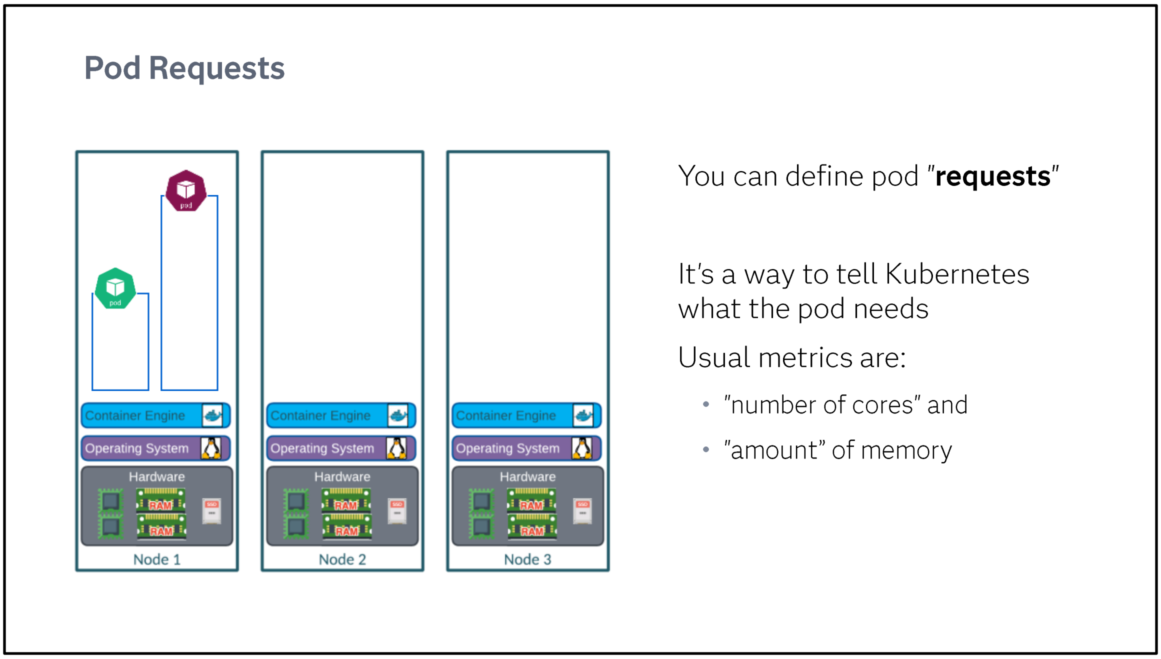
And, obviously, zero replicas of the Red pod

Kubernetes Replicas.

A pod can have zero or more replicas. Here the “Green” pod has five replicas.

And the “Blue” pod has two replicas.

These are the only pods running on the cluster. For example, there are no Red pods.



Pod Requests

You can define pod "**requests**"

It's a way to tell Kubernetes what the pod needs

Usual metrics are:

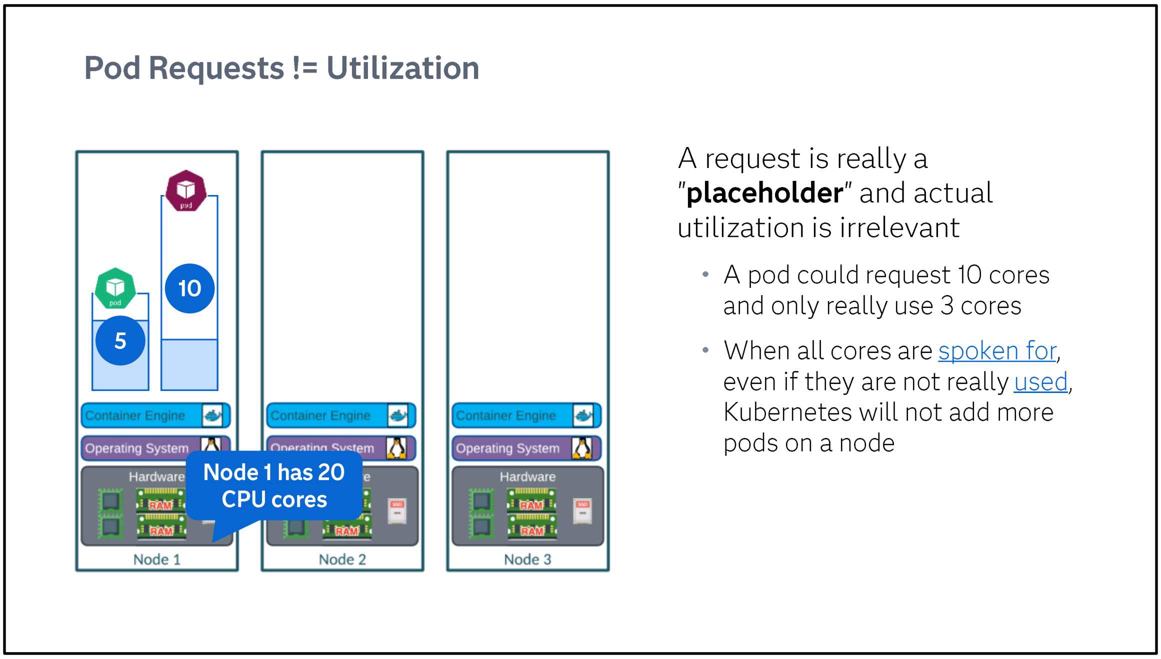
- "number of cores" and
- "amount" of memory

Pod Requests.

A request is a way to tell Kubernetes what a pod needs.

The request is expressed as a metric, usually, the number of C-P-U cores, or an amount of memory. But it could also be for other resources. For example, disk storage, ephemeral storage on a node.

In this example, perhaps the Green pod has a request for five CPU cores, and the Magenta pod is requesting ten cores.



Pod Requests != Utilization

A request is really a "**placeholder**" and actual utilization is irrelevant

- A pod could request 10 cores and only really use 3 cores
- When all cores are spoken for, even if they are not really used, Kubernetes will not add more pods on a node

Pod Requests does Not equate to node utilisation.

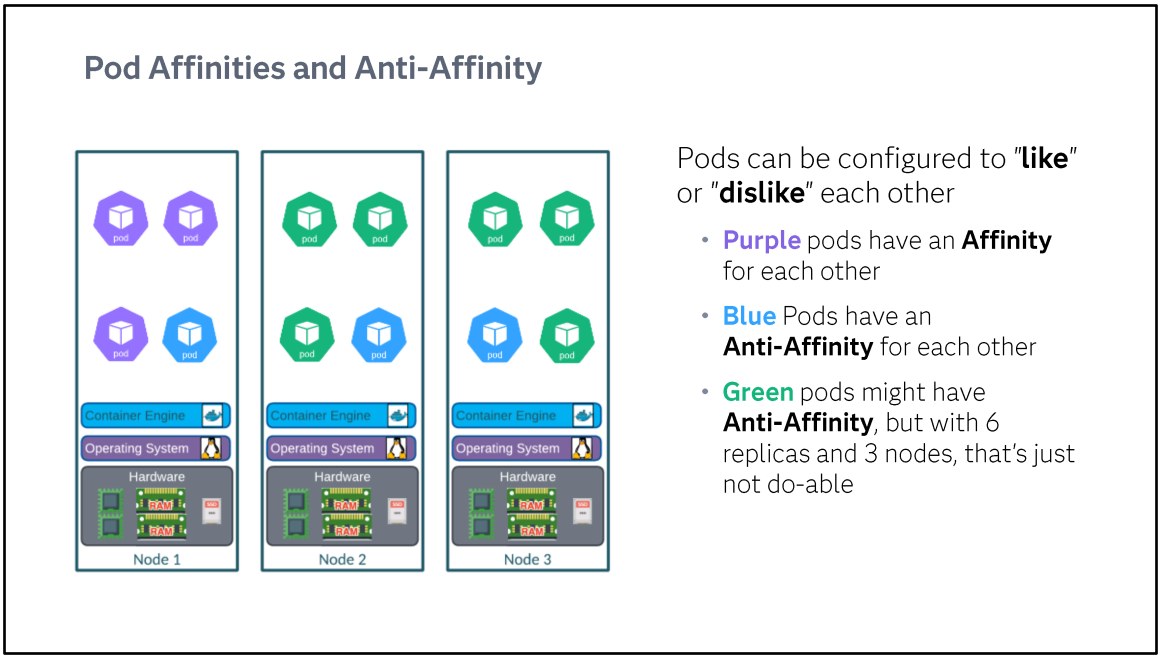
The request is a placeholder, a request for resources, the actual utilisation, or resource consumption, is irrelevant.

For example, the Magenta pod might request ten cores, but it is only using three cores. Where the Green pod has a request of five cores, but is using four cores.

When all the cores on a node are reserved, or spoken for, Kubernetes will not start any more pods on that node.

For example, if node one has twenty cores, and the Green pod has a request for five cores, and the Magenta pod has a request for ten cores, a second Magenta pod could not be started on that node.

This is due to the total number of CPU requests being greater than the node capacity. Five plus ten plus ten equals twenty-five. The total requests would be for twenty-five cores, but the node capacity is twenty cores.



Pod Affinities and Anti-Affinity

Pods can be configured to "**like**" or "**dislike**" each other

- **Purple** pods have an **Affinity** for each other
- **Blue** Pods have an **Anti-Affinity** for each other
- **Green** pods might have **Anti-Affinity**, but with 6 replicas and 3 nodes, that's just not do-able

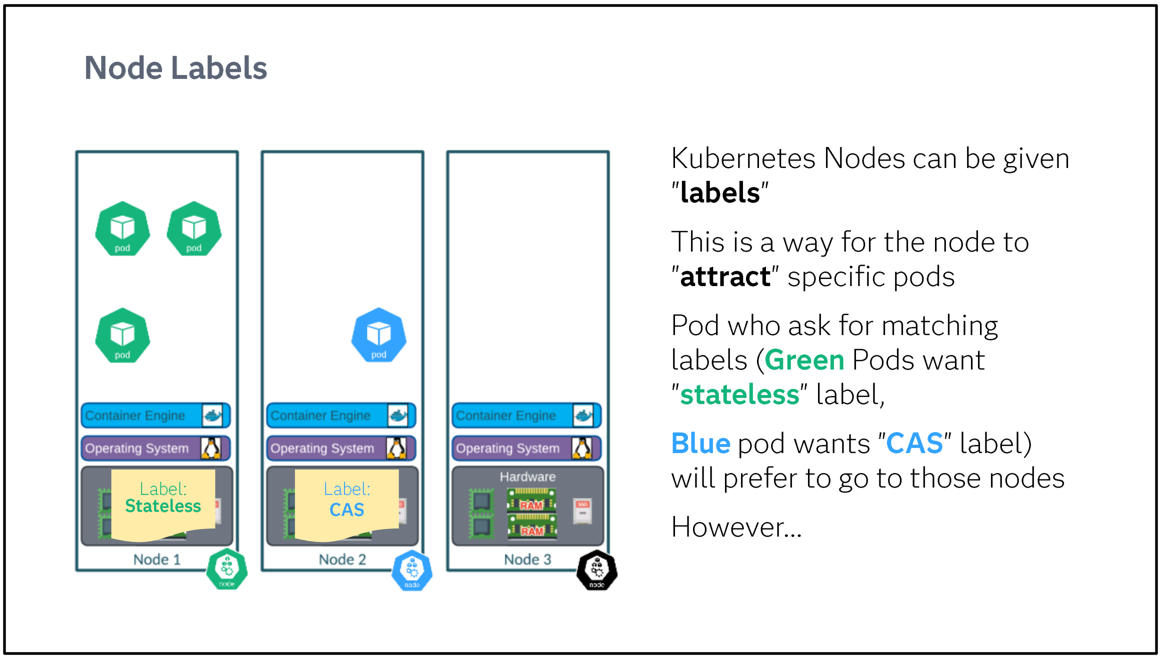
Pod Affinities and Anti-Affinity.

Pods can be configured to "like" or "dislike" each other.

In this example, the purple pods have an affinity for each other. They want to be co-located on the same node, and they are all running on node one.

The Blue pods have an Anti-Affinity for each other. They want to run on different nodes.

Finally, we can see the Green pods. They might have an anti-affinity for each other, and Kubernetes will try and run the pods on different nodes. But with six pods and only three nodes, it simply isn't possible to have one green pod on each node. As shown here, the green pods have ended up being co-located across two nodes.



Node Labels

Kubernetes Nodes can be given "**labels**"

This is a way for the node to "**attract**" specific pods

Pod who ask for matching labels (**Green** Pods want "**stateless**" label,

Blue pod wants "**CAS**" label) will prefer to go to those nodes

However…

Node Labels.

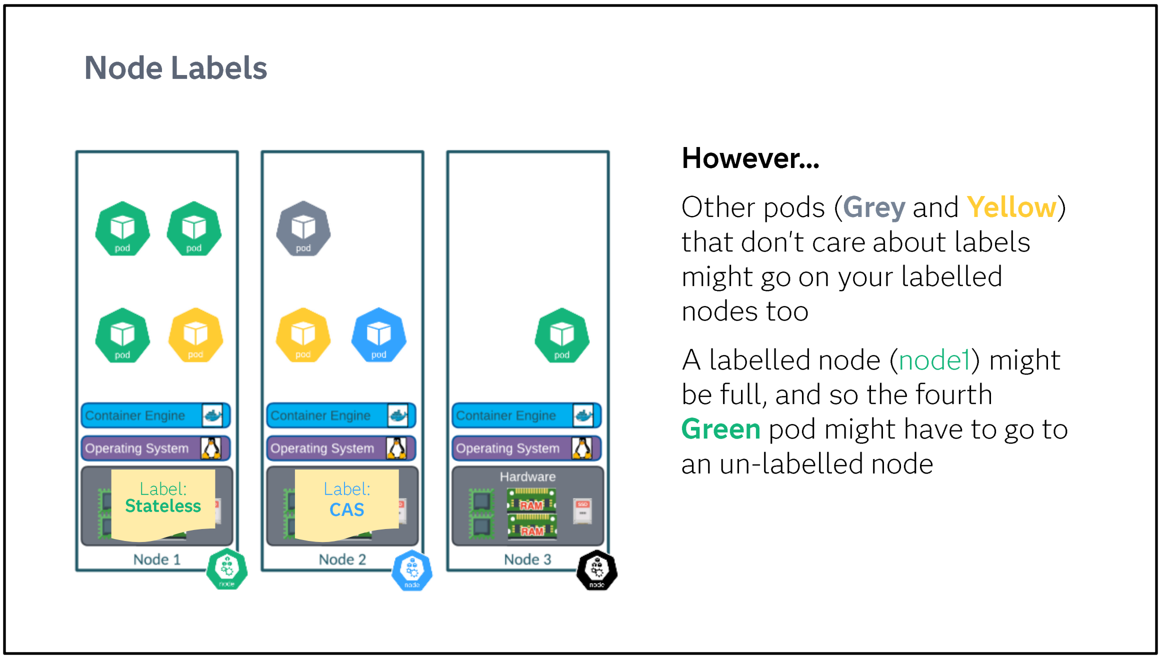
The Kubernetes nodes can be given a label. This provides a way to attract specific pods to a node. The labels are used as part of the node affinity configuration.

Node one has been given the Stateless label, and node two has been given the CAS label.

Pods that have an affinity for a specific label, want to run on a node with that label. Here the green pods want to run on node one.

The Blue pods want to run on a node with the CAS label. In this case, node two.

However, let’s examine this in more detail.



Node Labels

However…

Other pods (**Grey** and **Yellow**) that don't care about labels might go on your labelled nodes too

A labelled node (node1) might be full, and so the fourth **Green** pod might have to go to an un-labelled node

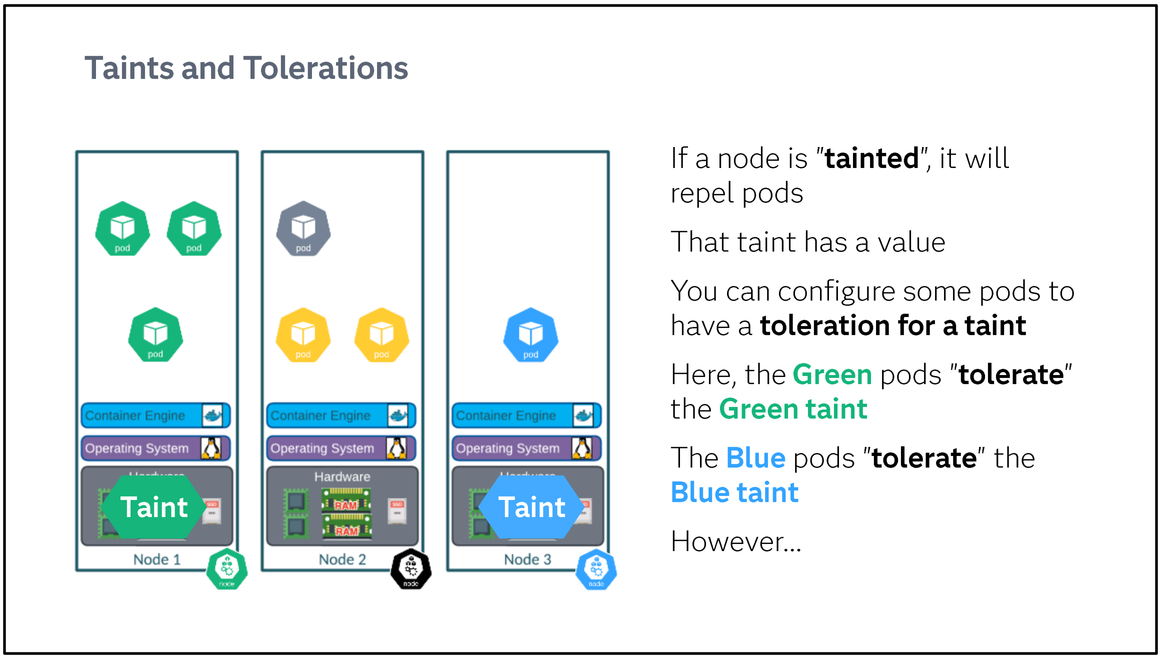
Now, let's extend this example.

Labels don't stop pods from running on a node. Here, the Grey and Yellow pods don't care about the node labels.

We can see that they are running on nodes one and two.

Now assume that node one is full, has no spare capacity. The fourth Green pod needs to be started somewhere. This could be on any of the available nodes.

In this example, node three has no label and Kubernetes started the Green pod on node three.



Taints and Tolerations

If a node is "**tainted**", it will repel pods

That taint has a value

You can configure some pods to have a **toleration for a taint**

Here, the **Green** pods "**tolerate**" the **Green taint**

The **Blue** pods "**tolerate**" the **Blue taint**

However…

We will now look at Taints and Tolerations.

A node can be given a taint. The taint is used to repel pods from running on the node.

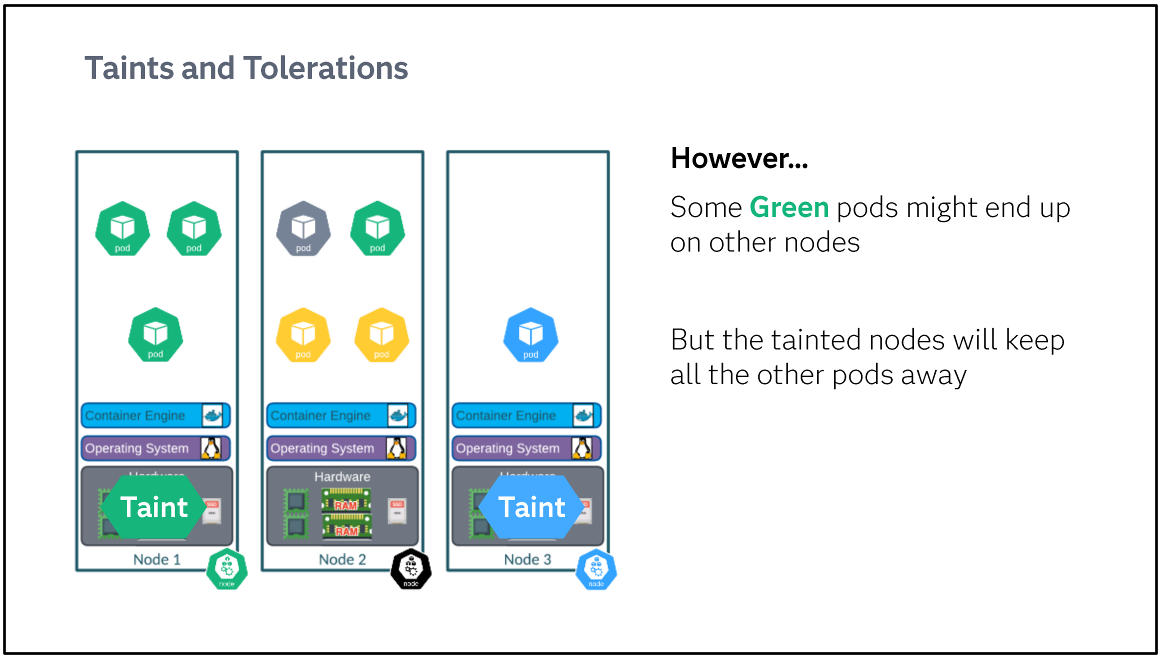
Pods can be configured to “tolerate” a taint, they have a toleration for the taint.

Here you can see that node one has the Green Taint, and node three has the “Blue Taint”. But node two has no taint applied. As the Grey and Yellow pods do not have a toleration for either taint, they are running on node two.

The Green pods have been given a toleration for the Green Taint, and are running on node one.

The Blue pod has a toleration for the Blue Taint, and is running on node three.

Let’s look at this in more detail.



Taints and Tolerations

However…

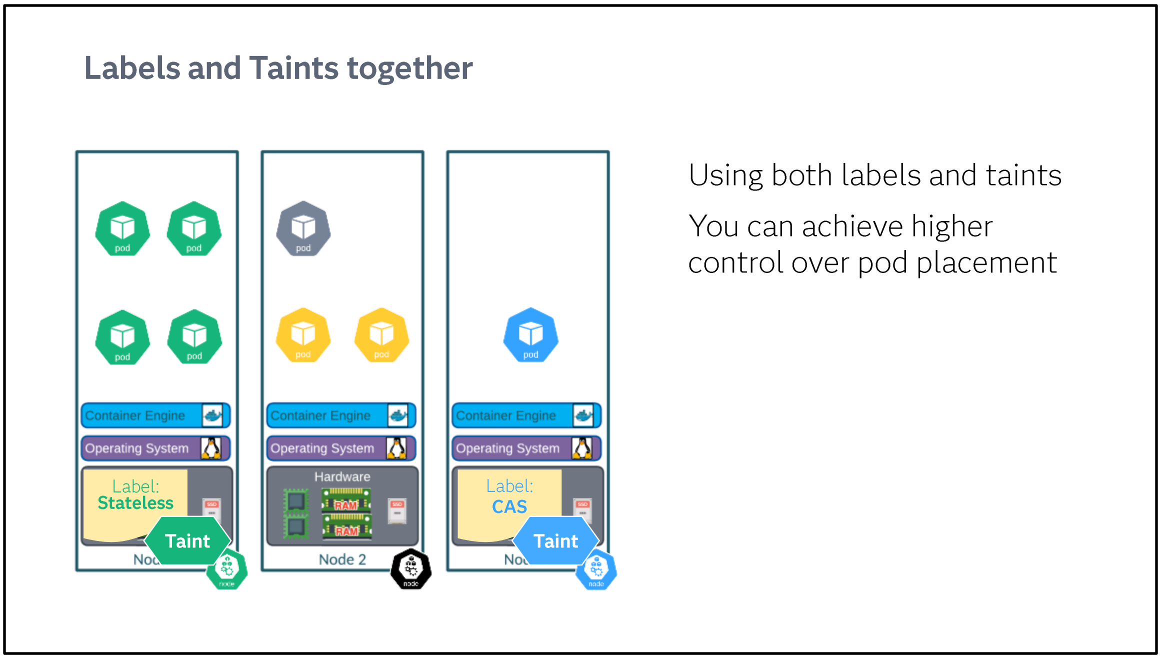
Some **Green** pods might end up on other nodes

But the tainted nodes will keep all the other pods away

The node selection is governed by many factors.

Depending on the node affinity configuration, a pod can still end up running on an un-tainted node. In this example, the fourth Green pod is now running on node two.

But the taints on nodes one and two, have kept the Grey and Yellow pods off these nodes.



Labels and Taints together

Using both labels and taints

You can achieve higher control over pod placement

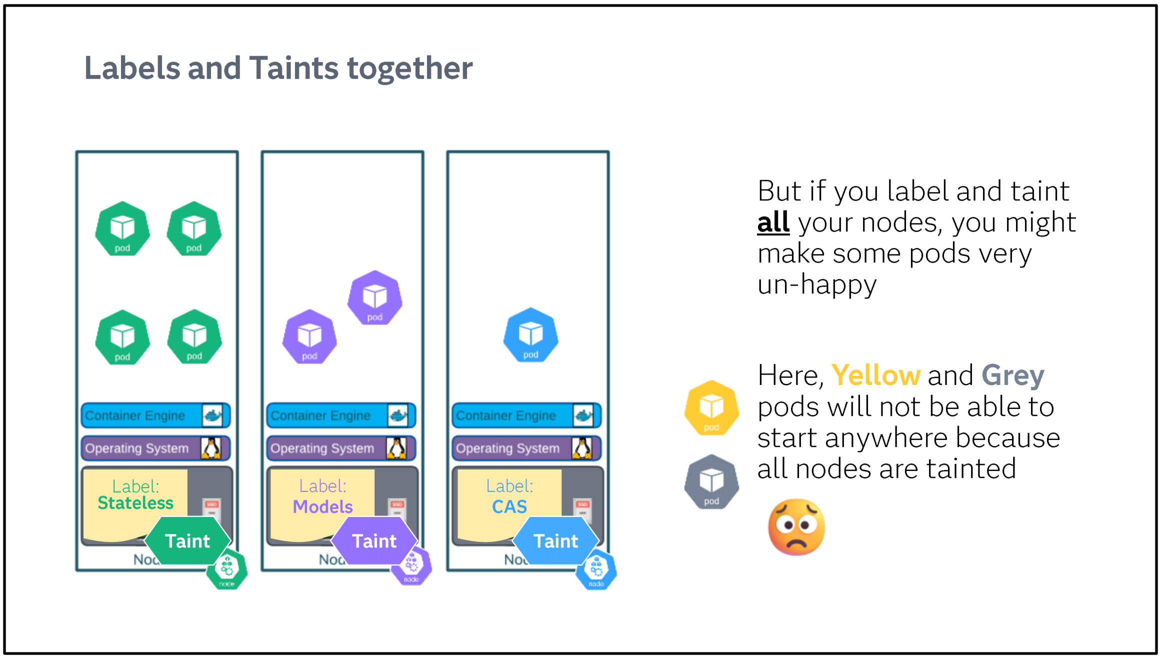
Using both labels and taints.
Nodes one and three have now been given a label and a taint.

By labelling and tainting the nodes you can achieve higher control over pod placement.

The label is used to attract pods, and the taint keeps pods away.

Now we can see that all the Green pods are running on node one, the stateless node. And the Blue pod is running on node three, the CAS node.

While the Yellow and Grey pods end up on node two, as they do not have a toleration for the green or blue taint.



Labels and Taints together

But if you label and taint **all** your nodes, you might make some pods very un-happy

Here, **Yellow** and **Grey** pods will not be able to start anywhere because all nodes are tainted

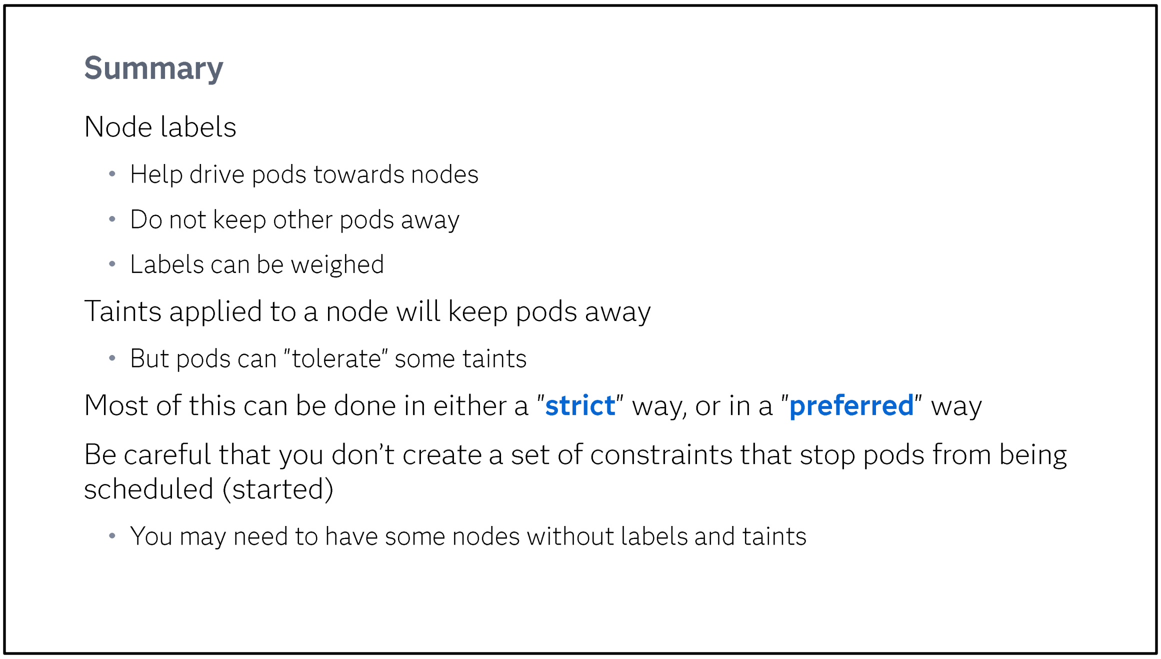
😟

But what happens if you label and taint all the nodes?

We can now see that node two has also been given a taint. The Models taint. Perhaps this node is dedicated to running the SAS Container Runtime pods.

If the pods do not tolerate any of the taints, they have nowhere to run.

In this example, the Yellow and Grey pods do not tolerate any of the taints, and end up in a pending state. They cannot be started on any of the nodes.



Summary

Node labels

- Help drive pods towards nodes
- Do not keep other pods away
- Labels can be weighed

Taints applied to a node will keep pods away

- But pods can "tolerate" some taints

Most of this can be done in either a "**strict**" way, or in a "**preferred**" way

Be careful that you don't create a set of constraints that stop pods from being scheduled (started)

- You may need to have some nodes without labels and taints

To summarize.

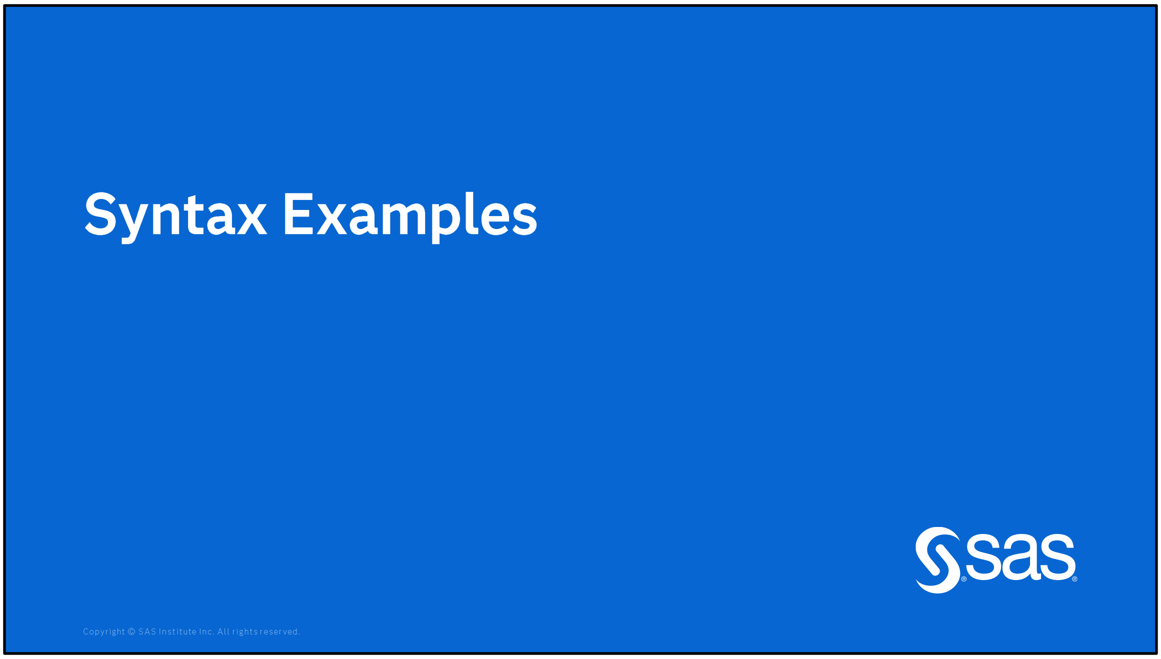
Node labels are used to attract pods to a node, but they do not keep pods away. They can also be given a weighting.

Taints are applied to a node to keep pods away. But a pod can be given a toleration for a taint.

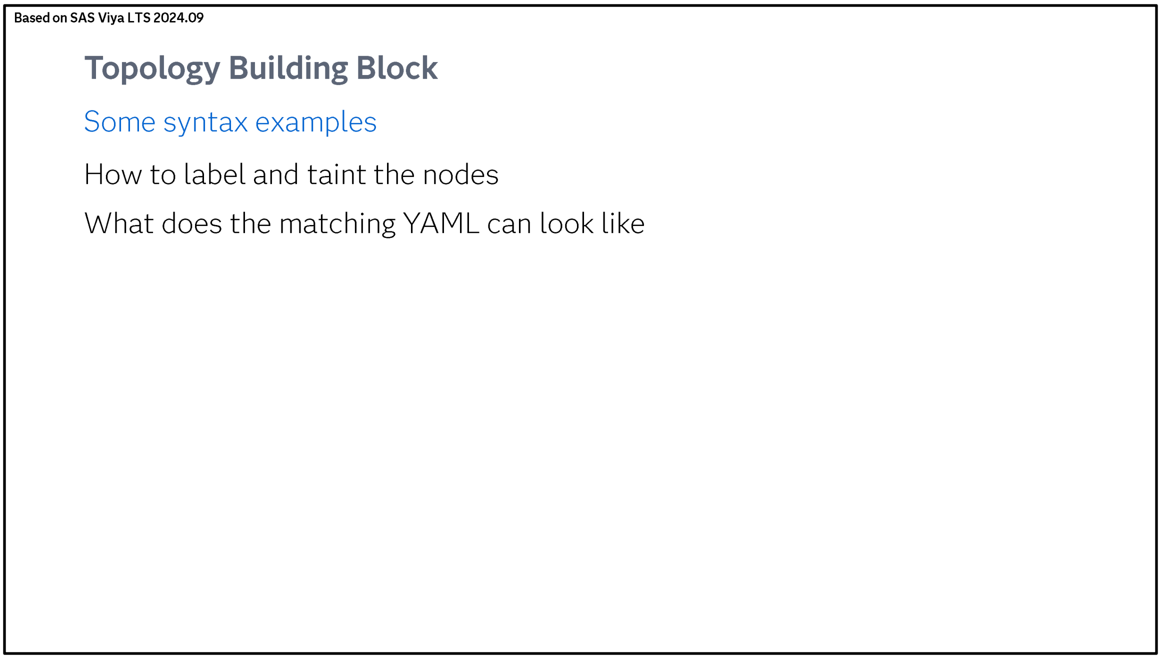
This configuration can be done in a strict or preferred way. Strict scheduling can lead to pods in a pending state, if no nodes are available with capacity to run the pod.

When applying the taints to the nodes and defining the node affinity you need to be careful that you don't create a set of constraints that stop pods from being scheduled, that prevents a pod from being started.

You may need to have some nodes without any labels and taints.



Syntax Examples



Based on SAS Viya LTS 2024.09

Topology Building Block

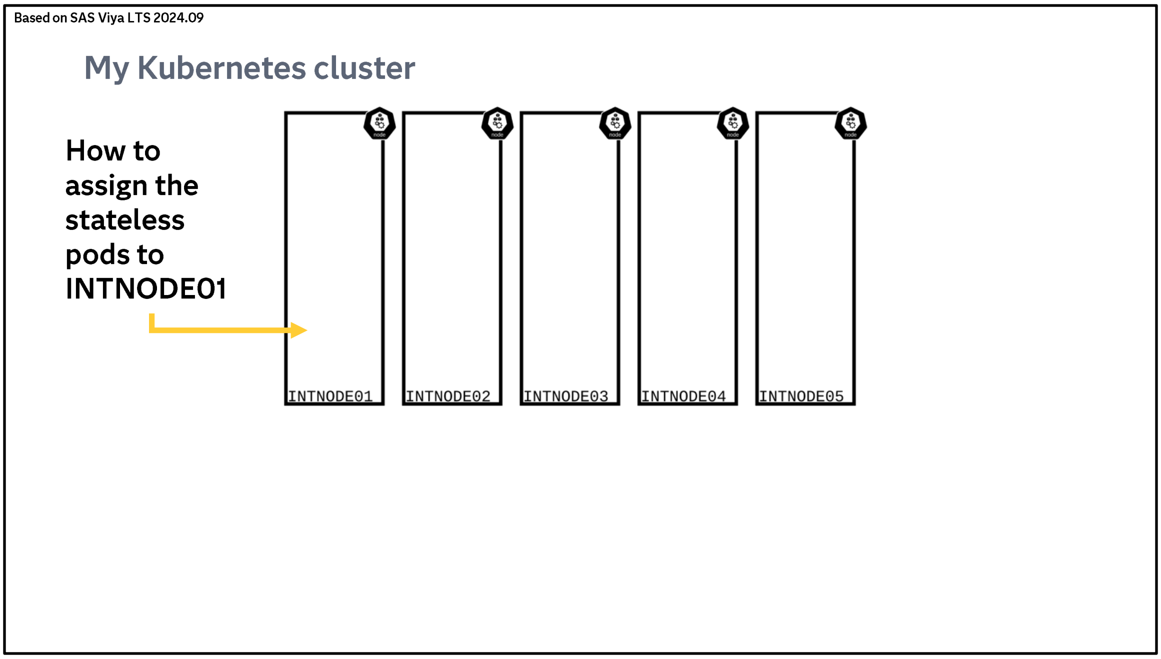
Some syntax examples

How to label and taint the nodes

What does the matching YAML can look like

In this module we are going to look at syntax examples, as part of discussing the Kubernetes topology building blocks.

We will look at how to label and taint the Kubernetes nodes, and what the corresponding YAML would look like.



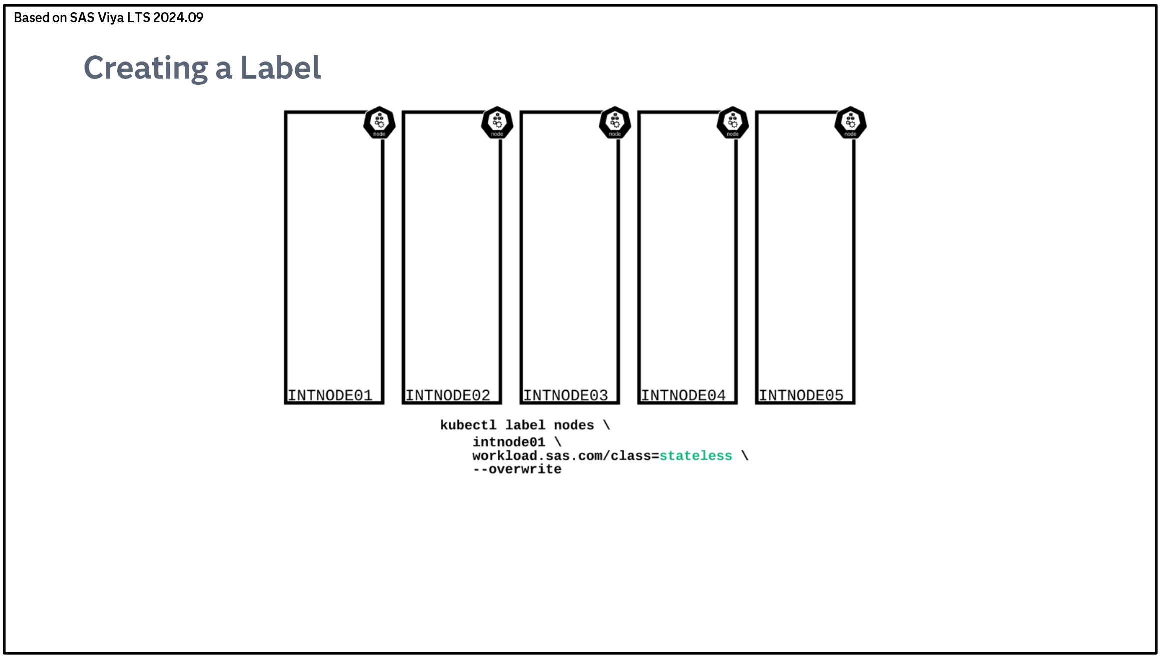
Based on SAS Viya LTS 2024.09

My Kubernetes cluster

How to assign the stateless pods to INTNODE01

Here we can see a Kubernetes cluster, it has five nodes. Int node zero one through to int node zero five.

In the following slides we will look at how to assign the SAS Viya stateless pods to int-node zero one.



Based on SAS Viya LTS 2024.09

Creating a Label

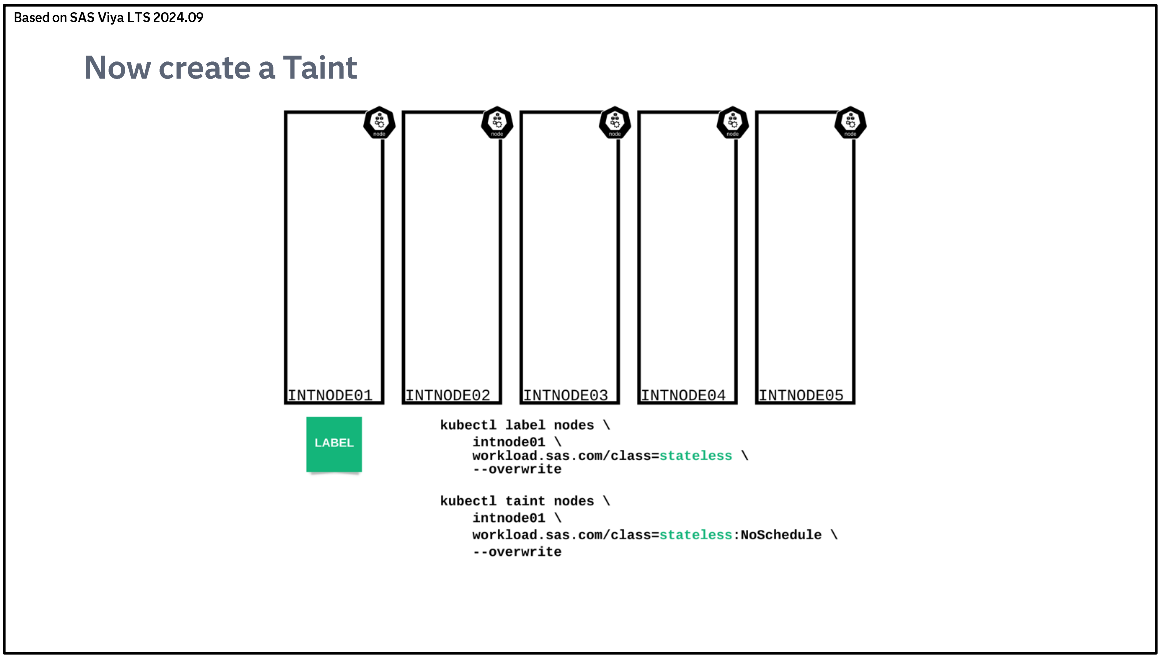
```
kubectl label nodes \
    intnode01 \
    workload.sas.com/class=stateless \
    --overwrite
```

The first step is to label and taint the nodes. We will start by looking at how to label a node.

The kubectl command to label the node is shown here. The label being applied is for the stateless workload class. That is, workload.sas.com/class=stateless.

The dash dash overwrite parameter specifies that any existing label will be replaced.

Running this command applies the label to int node zero one.



Based on SAS Viya LTS 2024.09

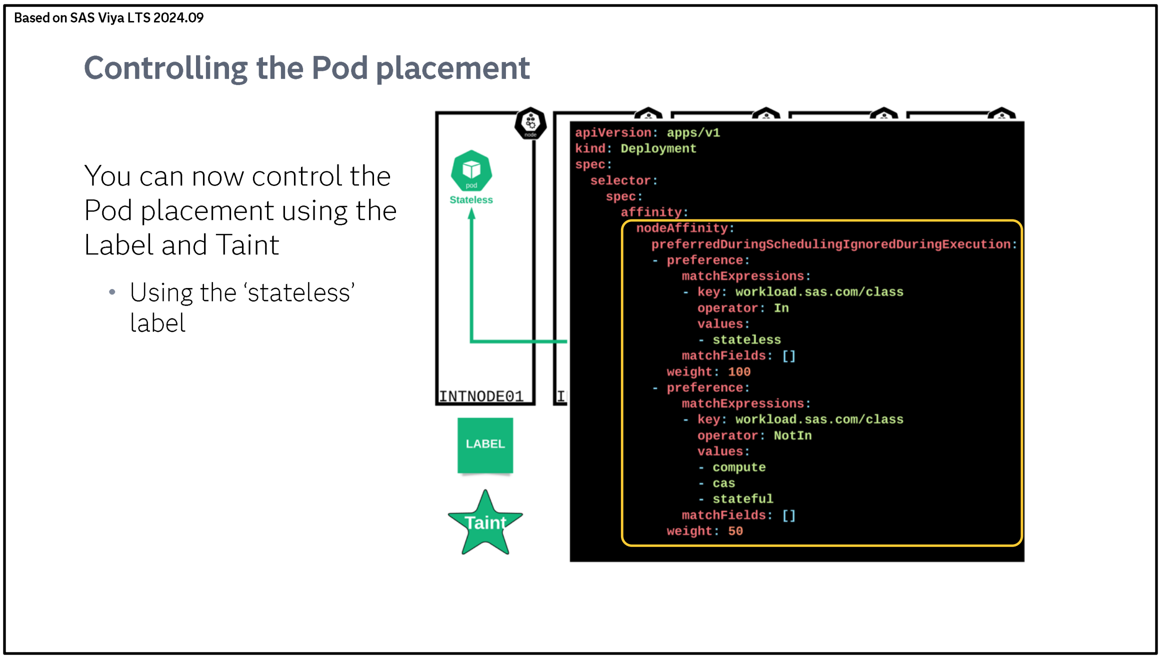
Now create a Taint

```
kubectl label nodes \
    intnode01 \
    workload.sas.com/class=stateless \
    --overwrite

kubectl taint nodes \
    intnode01 \
    workload.sas.com/class=stateless:NoSchedule \
    --overwrite
```

The next step is to taint the node.

Here you can now see the kubectl taint node command. The taint is for the Stateless workload class.

We now have the taint applied to the node. The taint is used to repel pods.



Based on SAS Viya LTS 2024.09

Controlling the Pod placement

You can now control the Pod placement using the Label and Taint

- Using the ‘stateless’ label

```
apiVersion: apps/v1
kind: Deployment
spec:
  selector:
    spec:
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - preference:
              matchExpressions:
              - key: workload.sas.com/class
                operator: In
                values:
                - stateless
              matchFields: []
            weight: 100
          - preference:
              matchExpressions:
              - key: workload.sas.com/class
                operator: NotIn
                values:
                - compute
                - cas
                - stateful
              matchFields: []
            weight: 50
```

Now that the node has a label and taint applied.

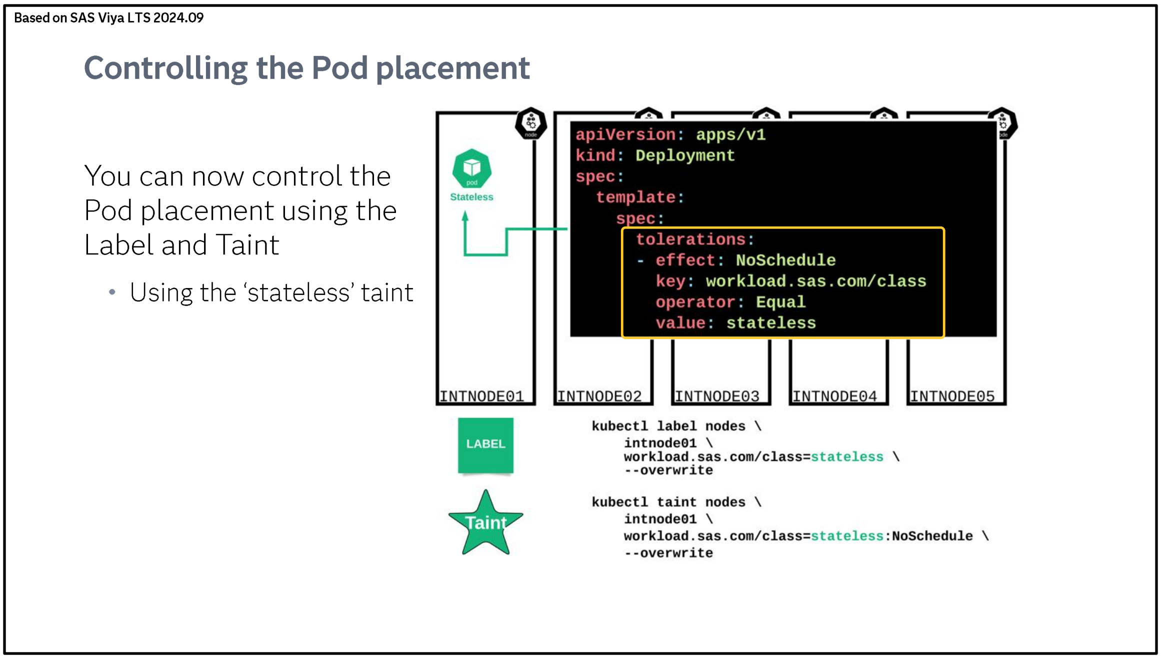
We will now look at how to control the pod placement.

In the code snippet on the right you can see the node affinity definition.

The SAS Viya deployment uses preferred scheduling. This is the preferred during scheduling ignore during execution statement.

The first match expression is for the stateless nodes. The node affinity is for nodes with the stateless workload class label.

The second match expression provides an alternative definition. Looking at the code you can see that the first match expression has a weighting of 100, while the alternate match expression has a weighting of 50.



Based on SAS Viya LTS 2024.09

Controlling the Pod placement

You can now control the Pod placement using the Label and Taint

- Using the 'stateless' taint

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      tolerations:
      - effect: NoSchedule
        key: workload.sas.com/class
        operator: Equal
        value: stateless
```

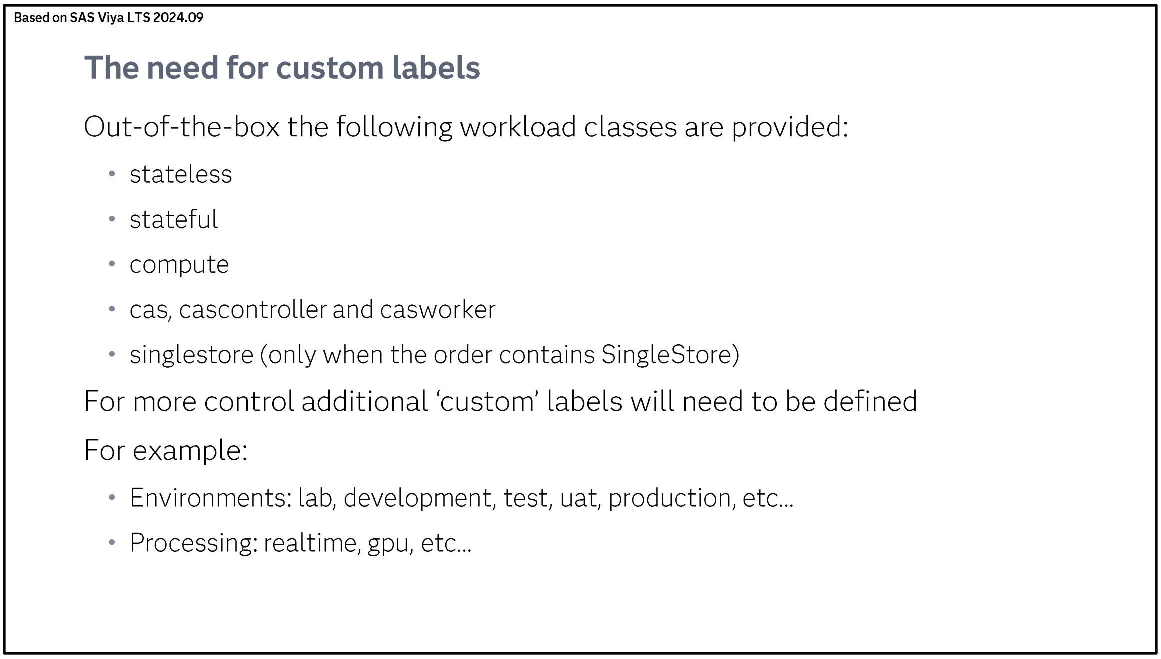
```
kubectl label nodes \
    intnode01 \
    workload.sas.com/class=stateless \
    --overwrite

kubectl taint nodes \
    intnode01 \
    workload.sas.com/class=stateless:NoSchedule \
    --overwrite
```

The next part of the deployment definition to discuss is the toleration for the stateless workload class taint.

As we want the stateless pods to run on the int node zero one node, and a taint has been applied to the node. The stateless pods need a toleration for the taint.

This is shown here.

This example is illustrating the default SAS Viya configuration. The Viya manifest are defined with the node affinities and tolerations as shown here.



Based on SAS Viya LTS 2024.09

The need for custom labels

Out-of-the-box the following workload classes are provided:

- stateless
- stateful
- compute
- cas, cascontroller and casworker
- singlestore (only when the order contains SingleStore)

For more control additional ‘custom’ labels will need to be defined

For example:

- Environments: lab, development, test, uat, production, etc…
- Processing: realtime, gpu, etc…

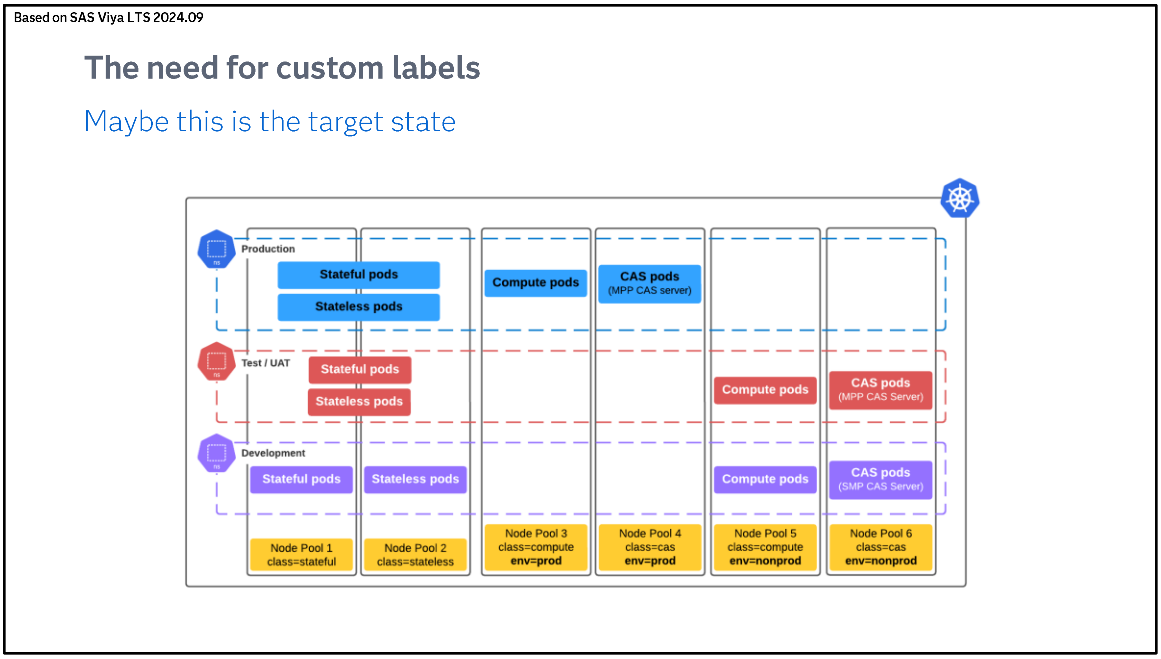
By default, the following workload classes are defined.

If the SAS order includes Single Store, the associated manifests include support for the single store workload class.

For greater control over the deployment topology custom labels will need to be defined. Along with the labels additional node taints may also be required.

For example, an environment label might be needed. Such as, test, UAT, or production.

Or, a label might be needed for the use of specialized nodes. For example, to target nodes with GPUs.



Based on SAS Viya LTS 2024.09

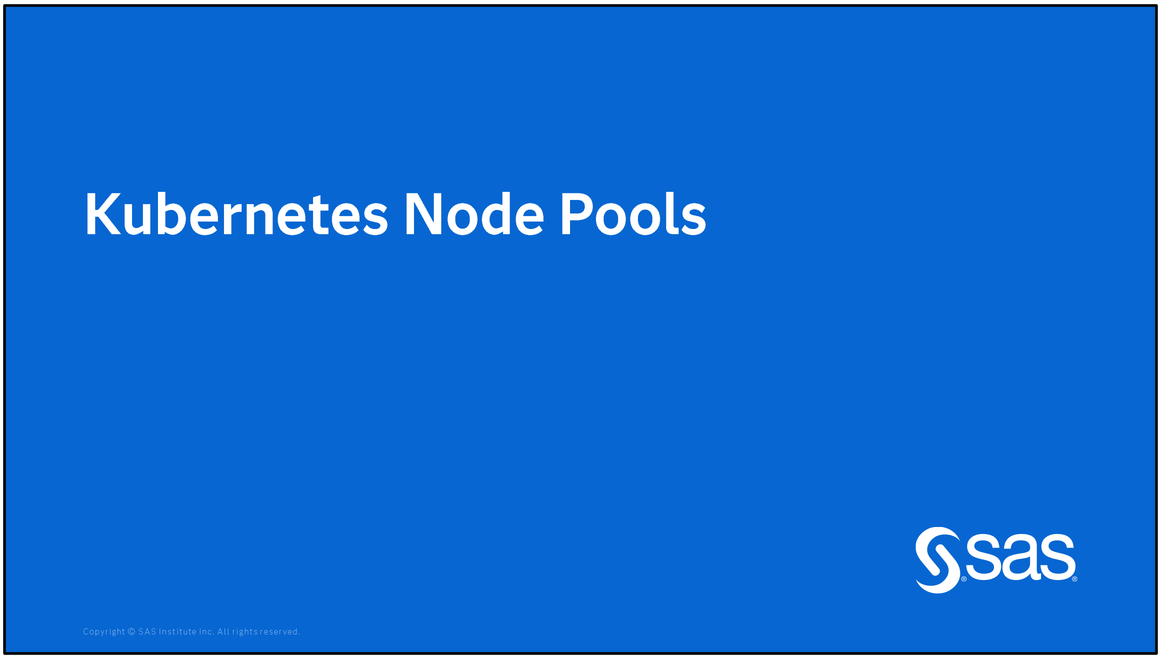
The need for custom labels

Maybe this is the target state

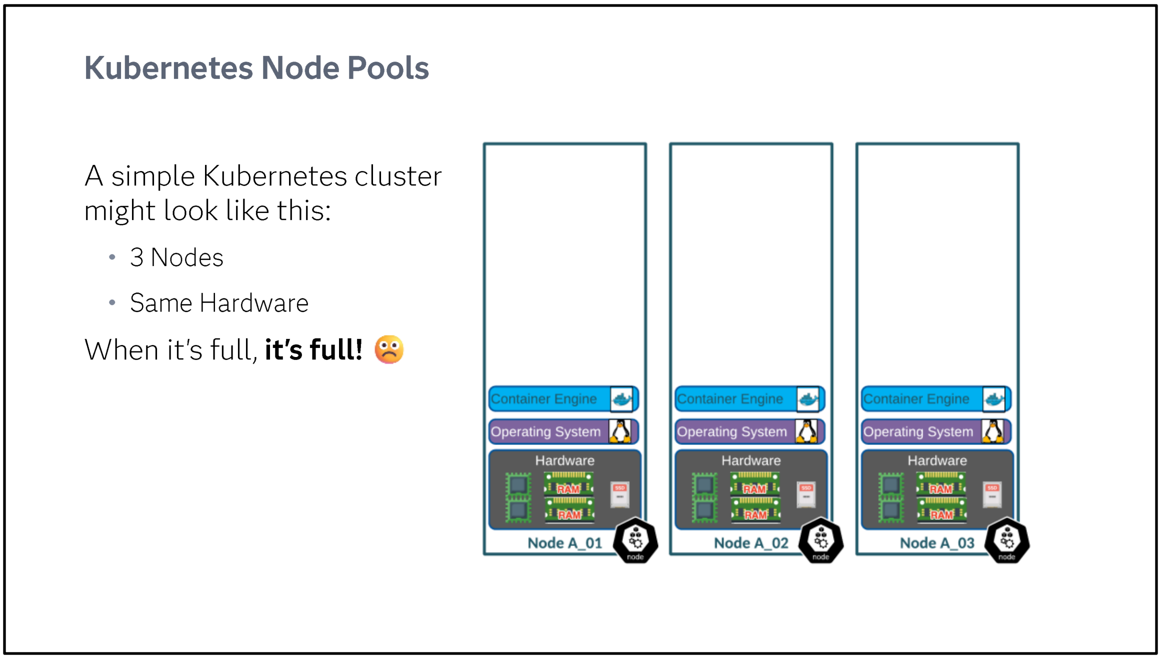
Finally, maybe this is the target topology for a shared Kubernetes cluster, with three SAS Viya deployments, three Viya environments.

Some nodes are shared by all three environments and some nodes might be dedicated to an environment. For example, node pool three and four are to be dedicated to the production environment.

That concludes our look at topology syntax examples. How to control the topology and the default workload classes are discussed in more detail in other modules.



Kubernetes Node Pools



Kubernetes Node Pools

A simple Kubernetes cluster might look like this:

- 3 Nodes
- Same Hardware

When it's full, **it's full!** ☹

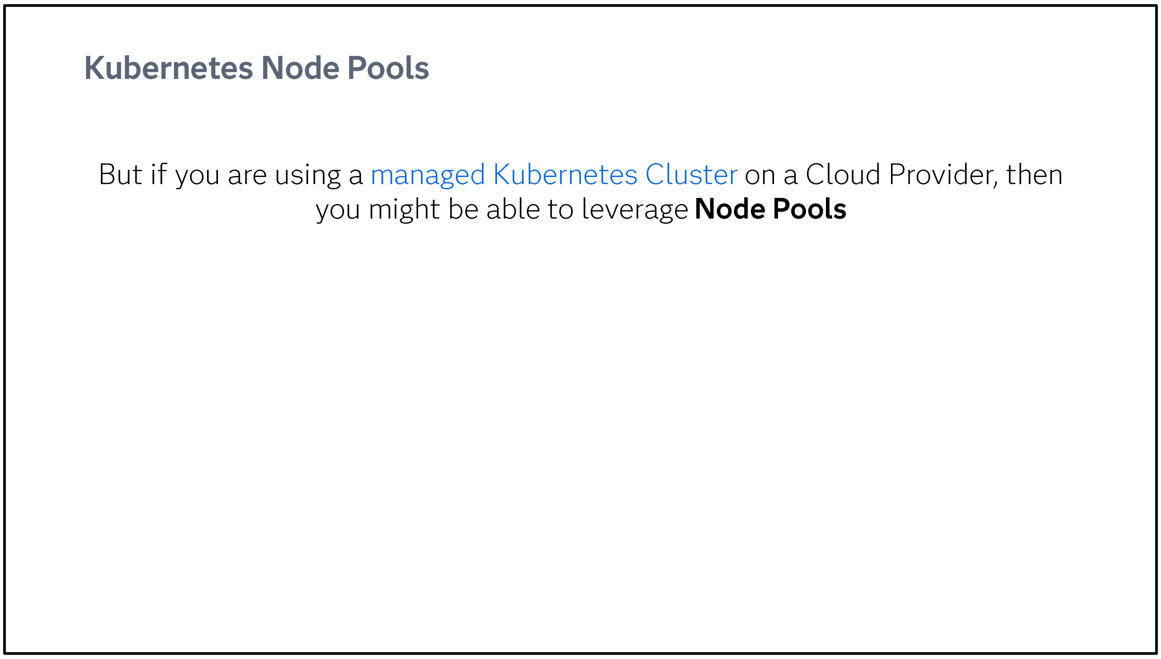
In this module we will discuss Kubernetes node pools. A node pool is a group of nodes with the same configuration.

Let's start by looking at a basic Kubernetes cluster.

Here is my Kubernetes cluster. It has three nodes. They're all the same size, the same hardware type, the same number of CPUs and the same amount of memory.

When the nodes are full, the cluster is full. There is no capacity to run any additional workload.

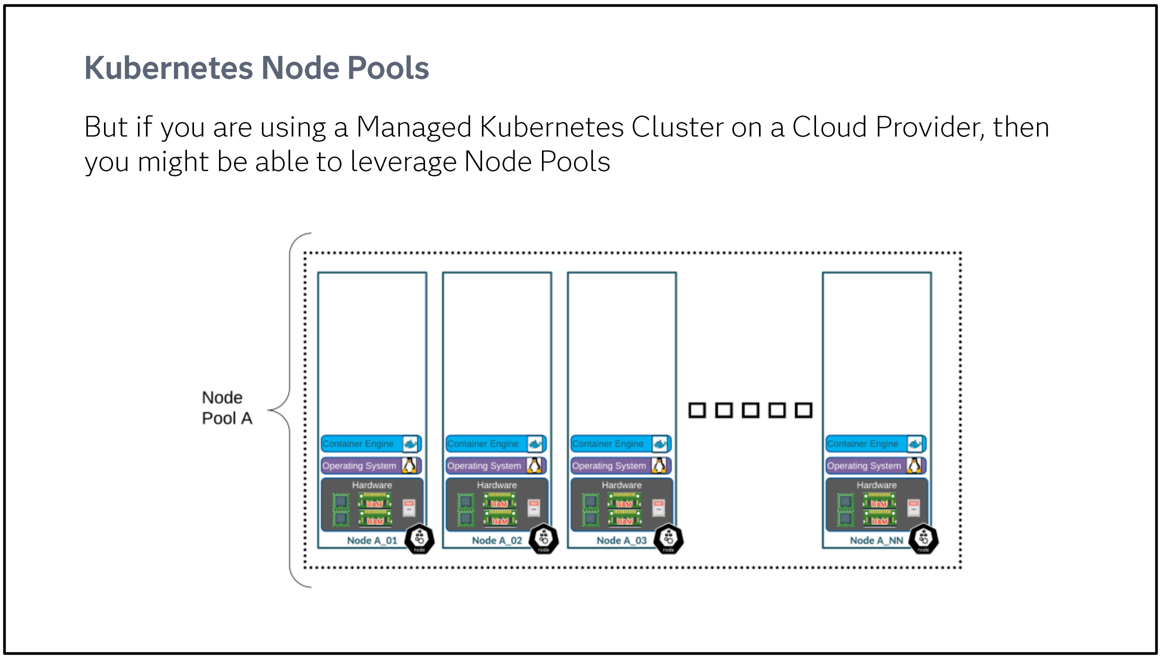
I would have to expand my cluster by adding additional nodes.



Kubernetes Node Pools

But if you are using a managed Kubernetes Cluster on a Cloud Provider, then you might be able to leverage **Node Pools**

But if you're using a managed Kubernetes cluster on one of the cloud providers, then you should be able to leverage node pools.

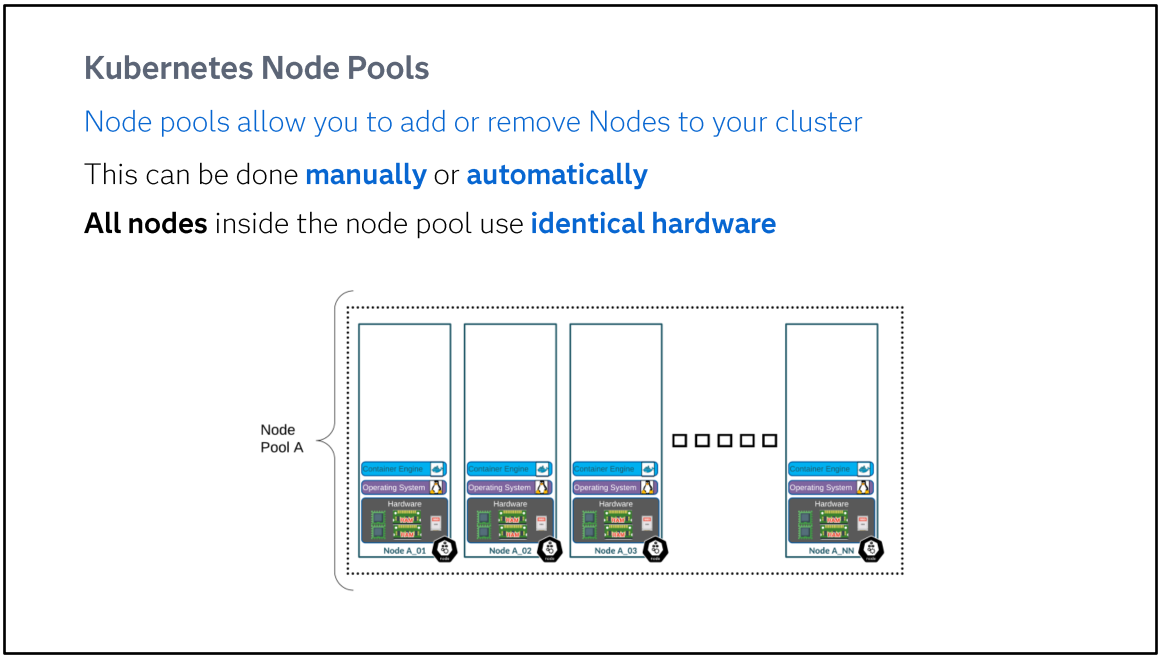


Kubernetes Node Pools

But if you are using a Managed Kubernetes Cluster on a Cloud Provider, then you might be able to leverage Node Pools

Here we can see “Node Poole A”. A node pool is a group of nodes with the same configuration. The node pool provides a template for the nodes.

Rather than being a fixed size, perhaps the node pool can dynamically scale out from three nodes up to N nodes. As shown here.



Kubernetes Node Pools

Node pools allow you to add or remove Nodes to your cluster

This can be done **manually** or **automatically**

All nodes inside the node pool use **identical hardware**

Node pools allow you to add or remove nodes to your cluster.

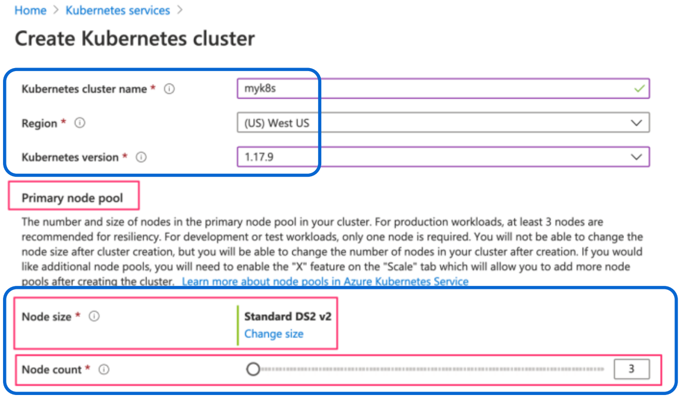
This can be done manually or automatically. When you manually adjust the node pool, you might be using the command line or a GUI like the Azure portal.

When the node pool is scaled automatically, it's because you've defined a scale set for the node pool.

Remember, all nodes inside the node pool use identical hardware. Here we can see node pool A, it has a number of nodes defined, they are all the same instance type.

Kubernetes Node Pools

Example: On Azure AKS



Create Kubernetes cluster

Kubernetes cluster name * ⓘ myk8s ✓

Region * ⓘ (US) West US

Kubernetes version * ⓘ 1.17.9

Primary node pool

The number and size of nodes in the primary node pool in your cluster. For production workloads, at least 3 nodes are recommended for resiliency. For development or test workloads, only one node is required. You will not be able to change the node size after cluster creation, but you will be able to change the number of nodes in your cluster after creation. If you would like additional node pools, you will need to enable the "X" feature on the "Scale" tab which will allow you to add more node pools after creating the cluster. Learn more about node pools in Azure Kubernetes Service

Node size * ⓘ Standard DS2 v2

Node count * ⓘ 3

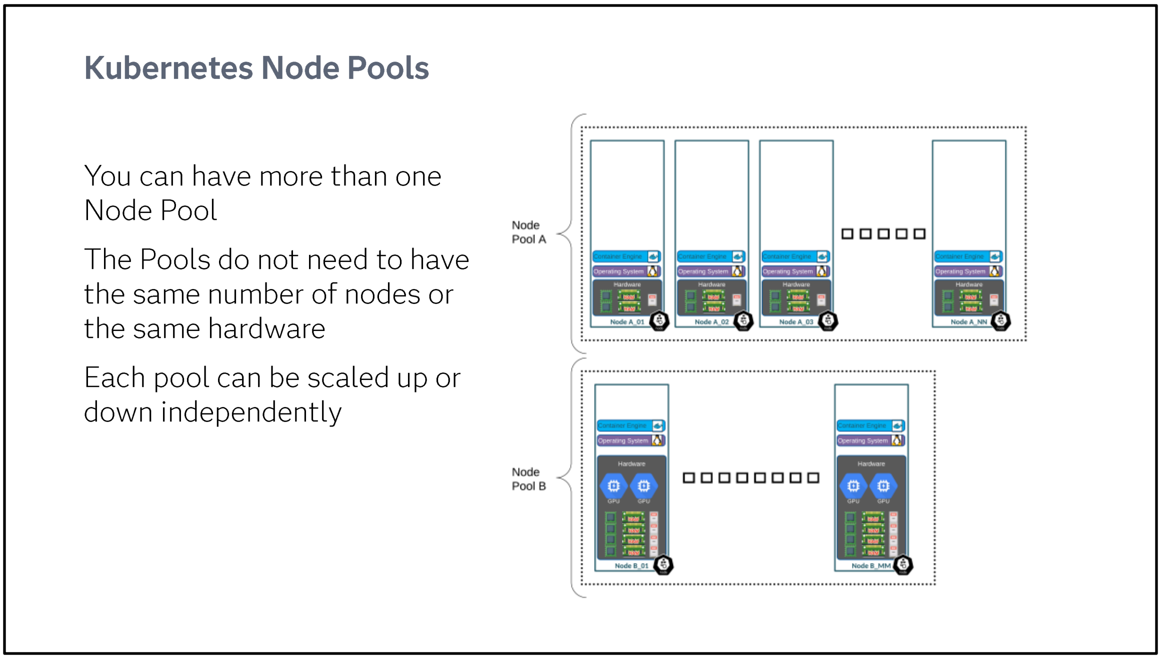
Note, the Kubernetes version shown is no longer supported. The Microsoft Azure Node instance types also change over time.

If we look at an example, for creating a Kubernetes cluster in Azure.

In this example, we can see that we are creating a cluster called “my K8s”. The region that the cluster will be running in, and the Kubernetes version that is going to be used. Note it's a very old Kubernetes version. This is an old screenshot.

The first node pool that is created is the primary, or system node pool. Here we are creating the primary node pool.

When creating the node pool, you get to select the size of the nodes, the instance type. In this example, the instance type is a standard DS2. You also get to set the number of nodes in the node pool, the node count. The node count has been set to three.



Kubernetes Node Pools

You can have more than one Node Pool

The Pools do not need to have the same number of nodes or the same hardware

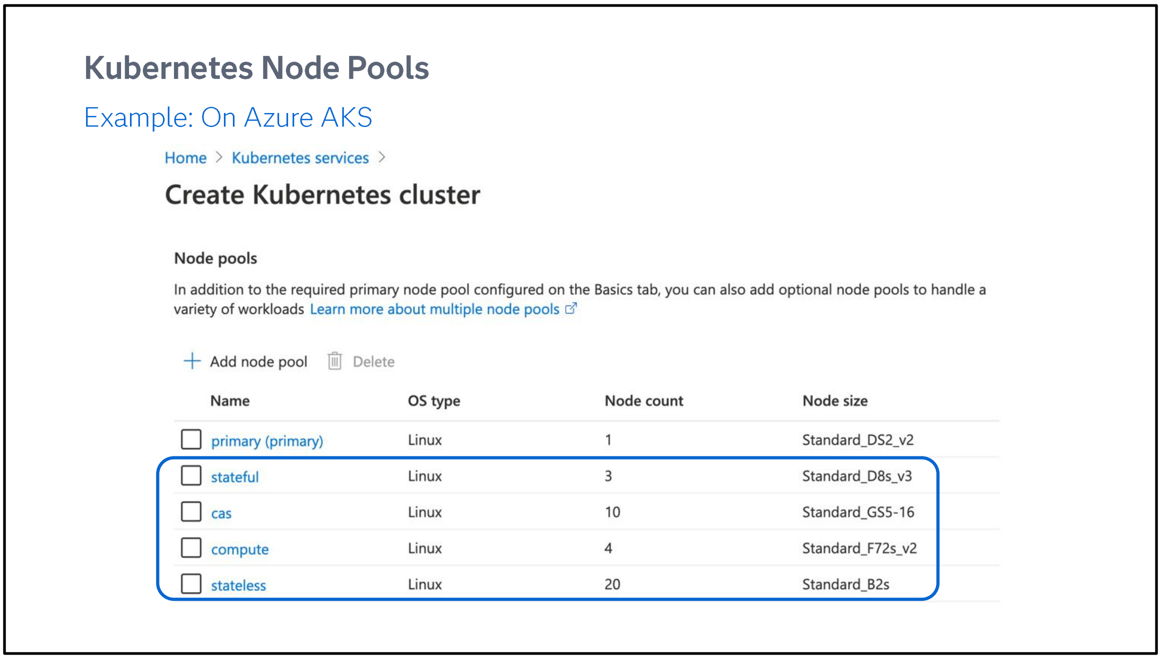
Each pool can be scaled up or down independently

You can add more than one node pool, and the node pools do not need to have the same number of nodes or use the same hardware.
Each node pool can be scaled independently of the other node pools.

Once again, we can see my node pool A, and perhaps this is being used for the SAS Viya stateful and stateless services.

But node pool B is using a different instance type. Here we can see that Node Pool B is using instances with GPUs. Perhaps this is for CAS or the compute server functions.

The use of node pools allows you to optimize the Kubernetes infrastructure, by defining nodes for specific workloads or to meet specific infrastructure requirements. Such as the amount of memory or the use of GPUs.



Kubernetes Node Pools

Example: On Azure AKS

Create Kubernetes cluster

Node pools

In addition to the required primary node pool configured on the Basics tab, you can also add optional node pools to handle a variety of workloads Learn more about multiple node pools

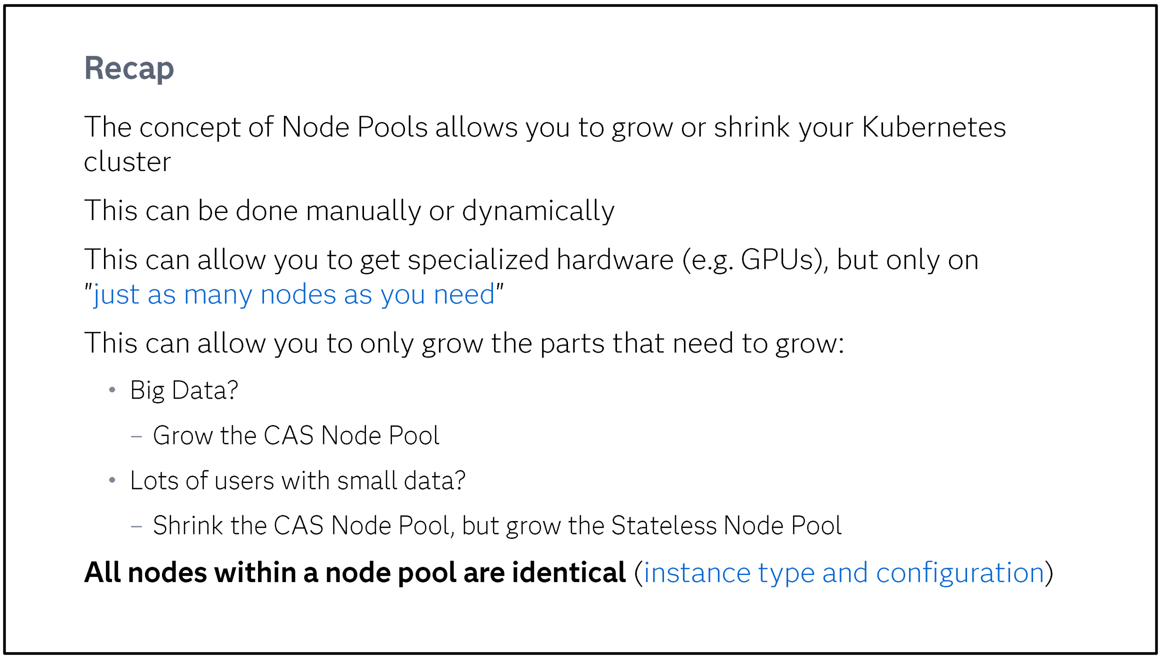
	Name	OS type	Node count	Node size
□	primary (primary)	Linux	1	Standard_DS2_v2
□	stateful	Linux	3	Standard_D8s_v3
□	cas	Linux	10	Standard_GS5-16
□	compute	Linux	4	Standard_F72s_v2
□	stateless	Linux	20	Standard_B2s

Coming back to the Azure example of creating my cluster. Once the primary node pool has been created, you can create additional node pools.

In this example, we can see the following node pools:
Stateful
CAS
Compute, and
Stateless.

You can see the OS type, the number of nodes, and the node size or the instance type that is been used.

As described earlier, this allows us to optimize the hardware, for a particular workload or requirement, for the pods that will be running on those nodes.



Recap

The concept of Node Pools allows you to grow or shrink your Kubernetes cluster

This can be done manually or dynamically

This can allow you to get specialized hardware (e.g. GPUs), but only on "just as many nodes as you need"

This can allow you to only grow the parts that need to grow:

- Big Data?
  - Grow the CAS Node Pool
- Lots of users with small data?
  - Shrink the CAS Node Pool, but grow the Stateless Node Pool

All nodes within a node pool are identical (instance type and configuration)

To recap, the concept of node pools allows you to grow or shrink your Kubernetes cluster.

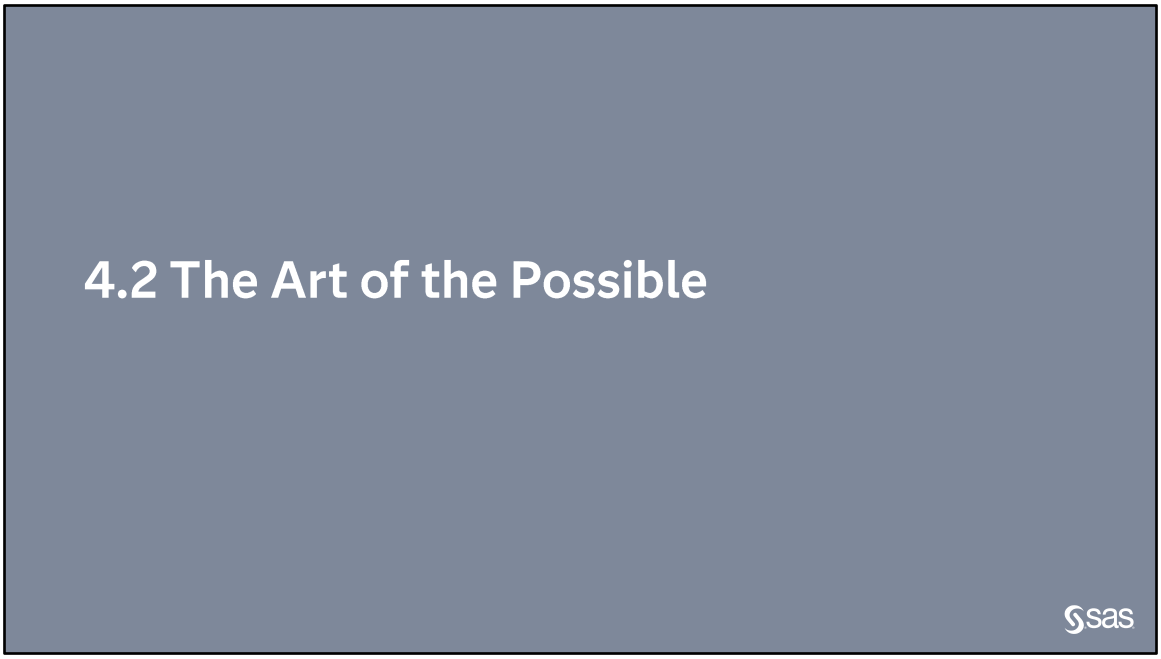
And this can be done manually or dynamically.

This allows you to use specialized hardware, such as instances with GPUs, but only as many as you need. Coming back to this concept that the node pools allow you to optimize that hardware. You don't have to use the same node type for all nodes within the Kubernetes cluster.

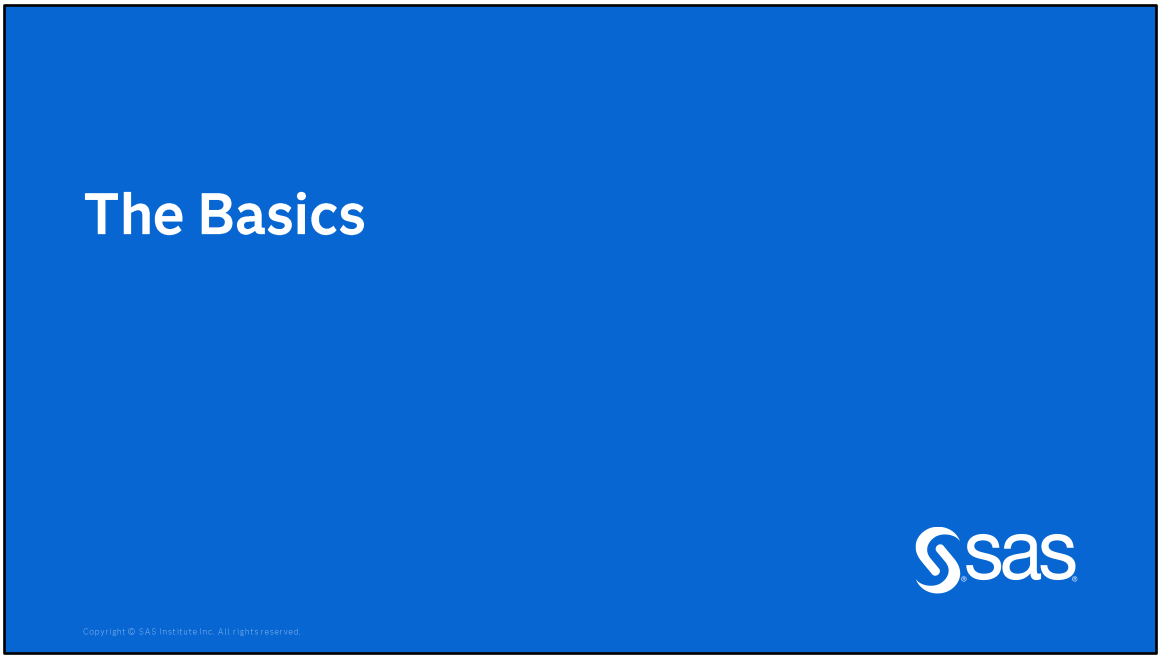
This can allow you to grow the parts that need to grow. For example, if you were dealing with big data, perhaps you would grow out the CAS node pool because you're using a large MPP CAS server, or maybe it's a MPP CAS server with GPUs.

Or perhaps you've got lots of users and a small amount of data, because it's more like a reporting environment. Therefore, you might shrink the CAS node pool, to use an SMP CAS server, but grow out the stateless services as these have to support a large number of users.

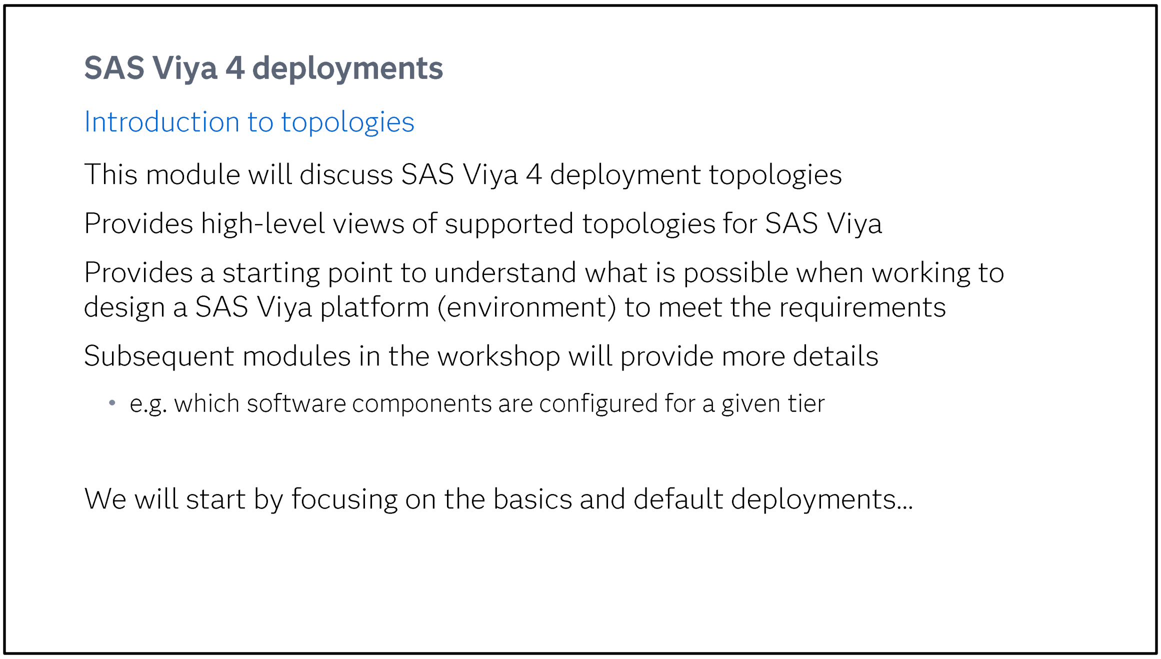
Finally, remember that all nodes within a node pool are identical. They're the same instance type and have the same configuration.



4.2 The Art of the Possible



The Basics



SAS Viya 4 deployments

Introduction to topologies

This module will discuss SAS Viya 4 deployment topologies

Provides high-level views of supported topologies for SAS Viya

Provides a starting point to understand what is possible when working to design a SAS Viya platform (environment) to meet the requirements

Subsequent modules in the workshop will provide more details

- e.g. which software components are configured for a given tier

We will start by focusing on the basics and default deployments…

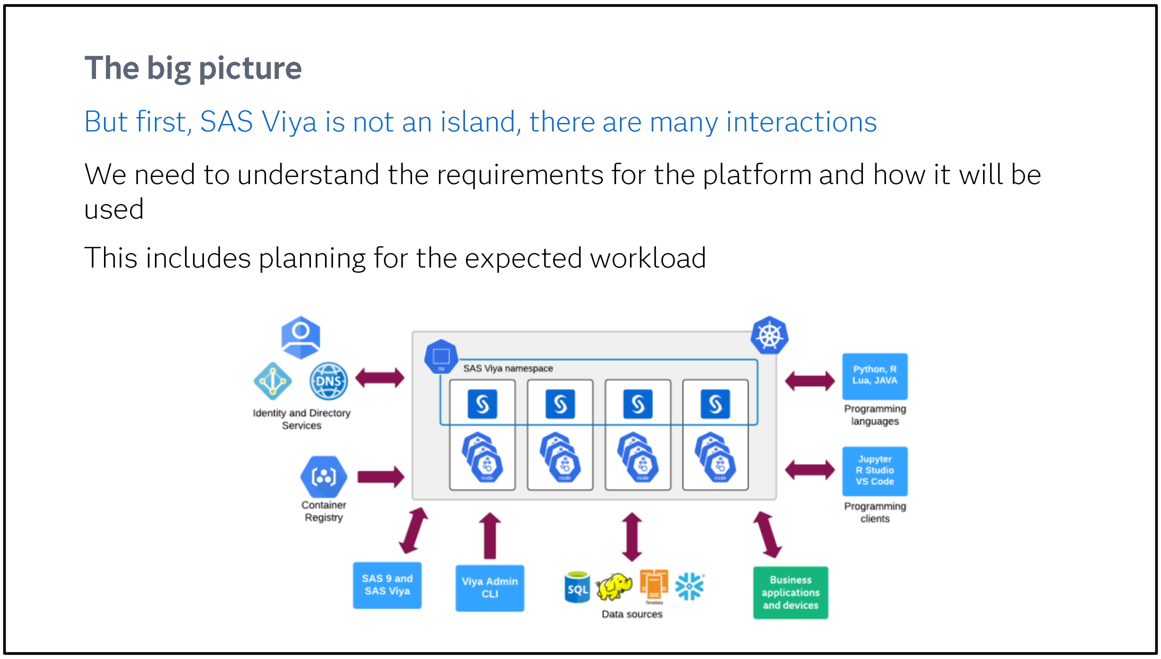
This module will discuss SAS Viya deployment topologies.

It provides a high-level view of the supported topologies for SAS Viya.

And provides a starting point to understand what is possible when working to design the SAS Viya platform, the environment, to meet the requirements.

Subsequent modules in the workshop provide more details. For example, the software components and how they’re configured for a given tier.

We will start by focusing on the basics and the default deployments.



The big picture

But first, SAS Viya is not an island, there are many interactions

We need to understand the requirements for the platform and how it will be used

This includes planning for the expected workload

Before we discuss the deployment topologies in detail, let’s first look at the wider picture, the big picture if you like.

It is important to understand the requirements for the platform and how it will be used. This includes planning for the expected workload.

The image illustrates that SAS Viya runs on Kubernetes and is deployed into a Kubernetes namespace, and that there are many interactions with systems and services that are outside of the Kubernetes cluster.

Starting from the left.

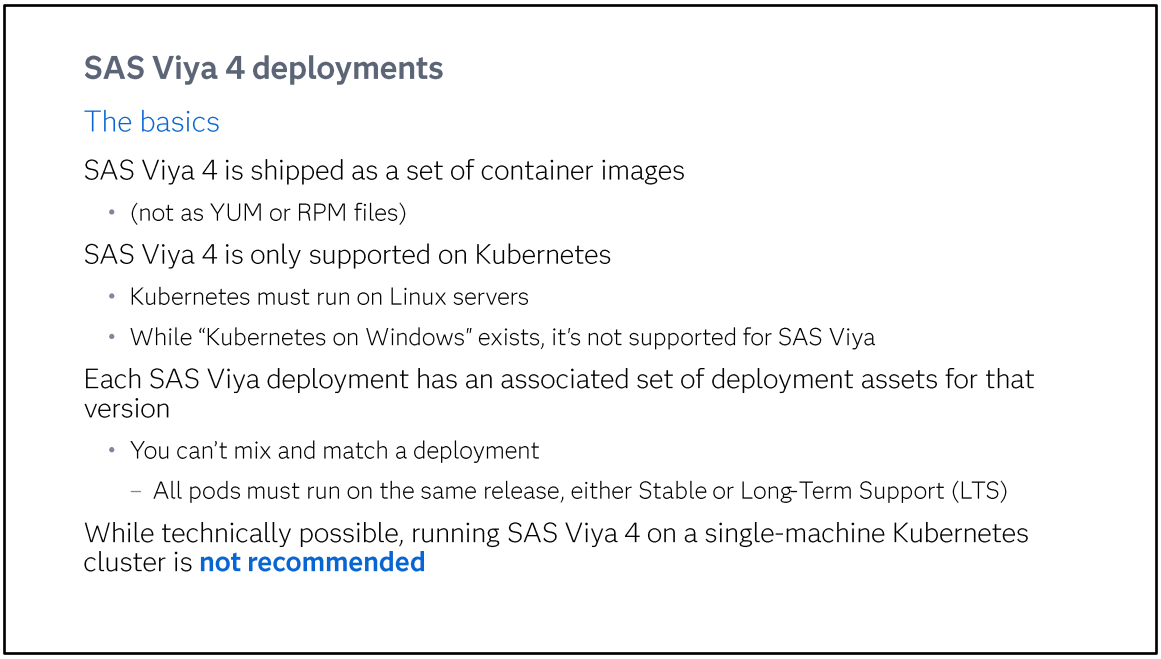
The SAS Viya platform has dependencies on other systems and services. For example, DNS, identity and directory services, a Docker Registry, etc.

The Viya platform might be access by, or integrated with, other SAS Viya platforms and maybe SAS 9. There can also be administrators using the Viya CLI.

The data could be in a variety of data sources, including SAS datasets, relational databases, Hadoop, and Snowflake to name a few sources.

Additionally, the SAS Viya platform could be accessed by other applications. For example, event stream processing and real-time decisioning.

Finally, there could be data scientists that are using a range of programming clients and programming languages to perform their work.



SAS Viya 4 deployments

The basics

SAS Viya 4 is shipped as a set of container images

- (not as YUM or RPM files)

SAS Viya 4 is only supported on Kubernetes

- Kubernetes must run on Linux servers
- While “Kubernetes on Windows” exists, it’s not supported for SAS Viya

Each SAS Viya deployment has an associated set of deployment assets for that version

- You can’t mix and match a deployment
  - All pods must run on the same release, either Stable or Long-Term Support (LTS)

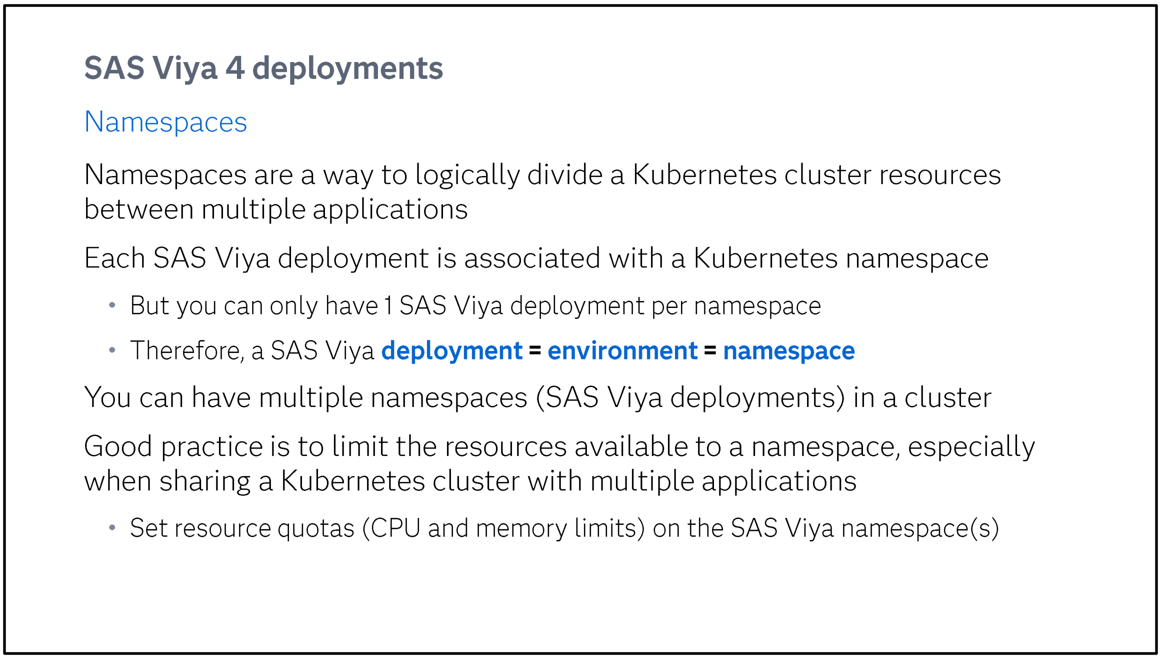
While technically possible, running SAS Viya 4 on a single-machine Kubernetes cluster is **not recommended**

SAS Viya 4 is shipped as a set of container images, not as YUM or RPM files.

SAS Viya is only supported on Kubernetes running on Linux servers. While it is possible to run Kubernetes on Windows servers, it's not supported for SAS Viya.

Each SAS Viya deployment has an associated set of deployment assets for that version, or cadence. You can't mix or match them in a deployment. All pods must run on the same release, either stable or long-term support.

While technically possible, running SAS Viya on a single machine Kubernetes cluster, it is not recommended.



SAS Viya 4 deployments

Namespaces

Namespaces are a way to logically divide a Kubernetes cluster resources between multiple applications

Each SAS Viya deployment is associated with a Kubernetes namespace

- But you can only have 1 SAS Viya deployment per namespace
- Therefore, a SAS Viya **deployment** = **environment** = **namespace**

You can have multiple namespaces (SAS Viya deployments) in a cluster

Good practice is to limit the resources available to a namespace, especially when sharing a Kubernetes cluster with multiple applications

- Set resource quotas (CPU and memory limits) on the SAS Viya namespace(s)

Kubernetes namespaces.

Namespaces are a way to logically divide a Kubernetes cluster resources between multiple applications.

Each SAS Viya deployment is associated with a Kubernetes namespace.

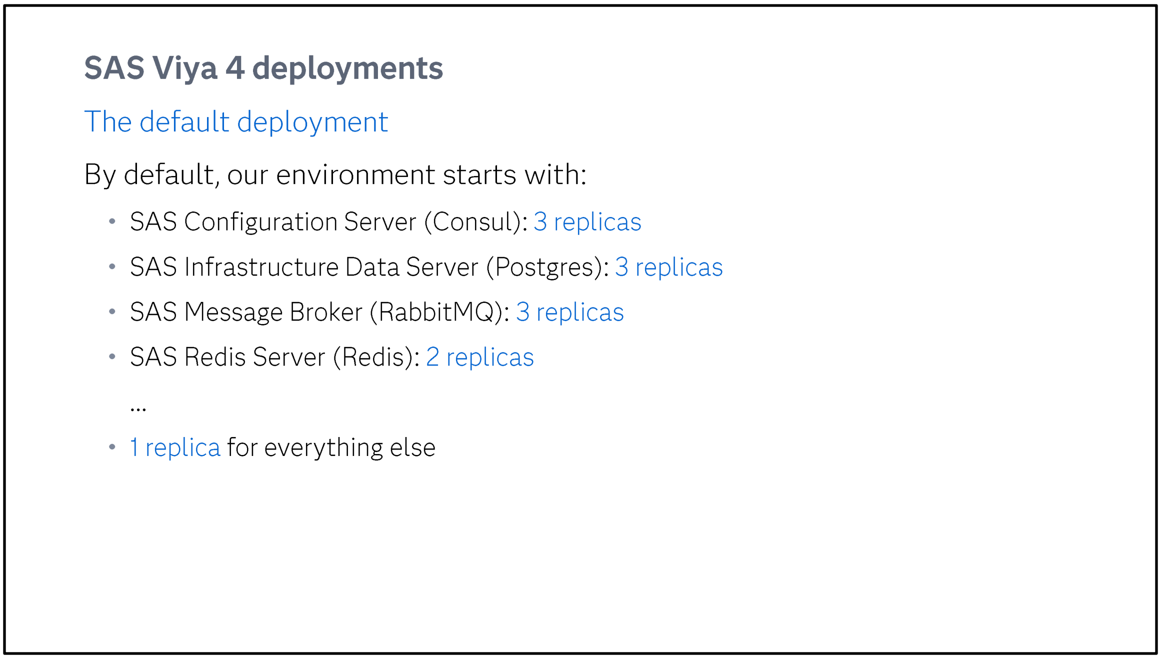
But you can only have 1 SAS Viya deployment per namespace.

Therefore, you could say that a SAS Viya deployment equals an environment, which equals a namespace.

You can have multiple SAS Viya deployments in a Kubernetes cluster. Each running in its own namespace. But you need to make sure that they can coexist together. If running different cadence versions of SAS Viya, there could be different system requirements.

It is good practice to limit the resources available to a namespace, especially when sharing a Kubernetes cluster with multiple applications.

It is possible to set resource quotas on CPU and memory limits, on the SAS Viya namespace.



SAS Viya 4 deployments

The default deployment

By default, our environment starts with:

- SAS Configuration Server (Consul): 3 replicas
- SAS Infrastructure Data Server (Postgres): 3 replicas
- SAS Message Broker (RabbitMQ): 3 replicas
- SAS Redis Server (Redis): 2 replicas

…

- 1 replica for everything else

If we look at the default deployment.

By default, the environment starts with three replicas for the SAS Configuration server.

Three replicas for the SAS Infrastructure Data server, if you're using internal Postgres.

Three replicas of the SAS Message Broker pods. And the SAS Redis Server has two replicas.

And there is one replica of everything else.

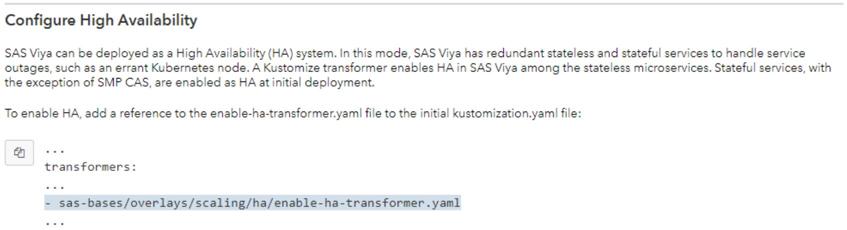
SAS Viya 4 deployments

The default deployment

Therefore, the default deployment provides redundancy for the core services (Consul, PostgreSQL, RabbitMQ, Redis…)

Enabling High Availability (HA)

- There is a 'transformer' to enable HA for the stateless microservices (pods)



Configure High Availability

SAS Viya can be deployed as a High Availability (HA) system. In this mode, SAS Viya has redundant stateless and stateful services to handle service outages, such as an errant Kubernetes node. A Kustomize transformer enables HA in SAS Viya among the stateless microservices. Stateful services, with the exception of SMP CAS, are enabled as HA at initial deployment.

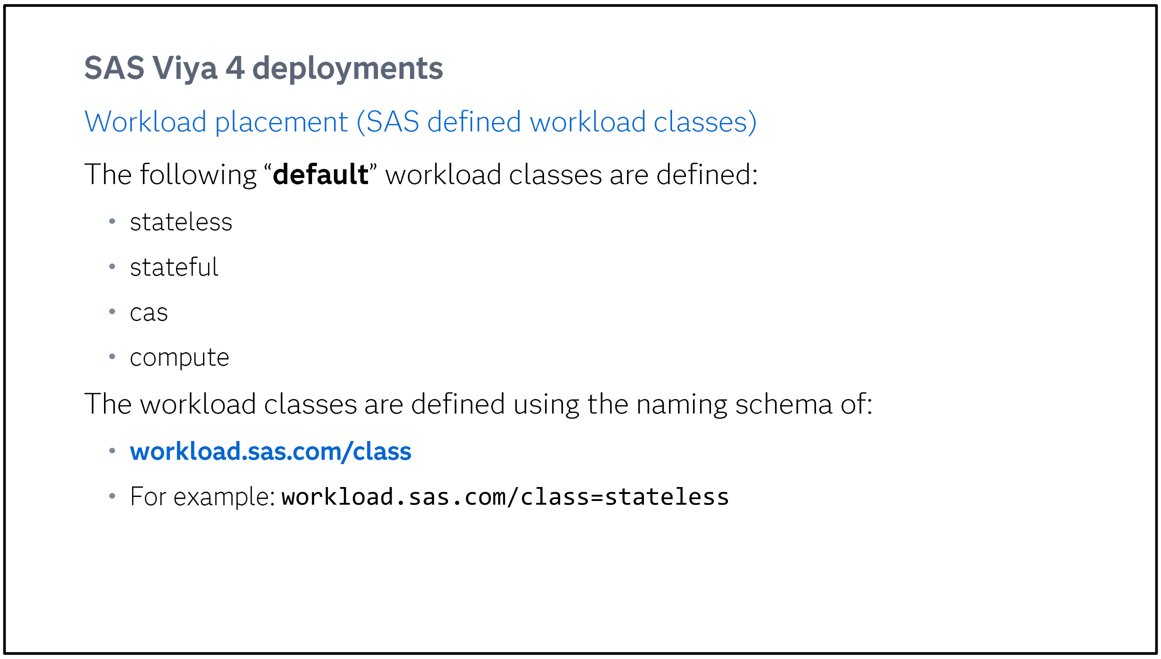
To enable HA, add a reference to the enable-ha-transformer.yaml file to the initial kustomization.yaml file:

```
...
transformers:
...
- sas-bases/overlays/scaling/ha/enable-ha-transformer.yaml
...
```

Therefore, the default deployment provides redundancy for the core services, for the stateful services, console, Postgres, RabbitMQ, et cetera.

You can enable high availability, and there is a patch transformer to enable HA for the stateless microservices. The image is from the Deployment Guide.

To enable HA, you need to include the overlay from the sas bases. The enable HA transformer needs to be added to the transformers block in the kustomization.yaml file.



SAS Viya 4 deployments

Workload placement (SAS defined workload classes)

The following “**default**” workload classes are defined:

- stateless
- stateful
- cas
- compute

The workload classes are defined using the naming schema of:

- **workload.sas.com/class**
- For example: `workload.sas.com/class=stateless`

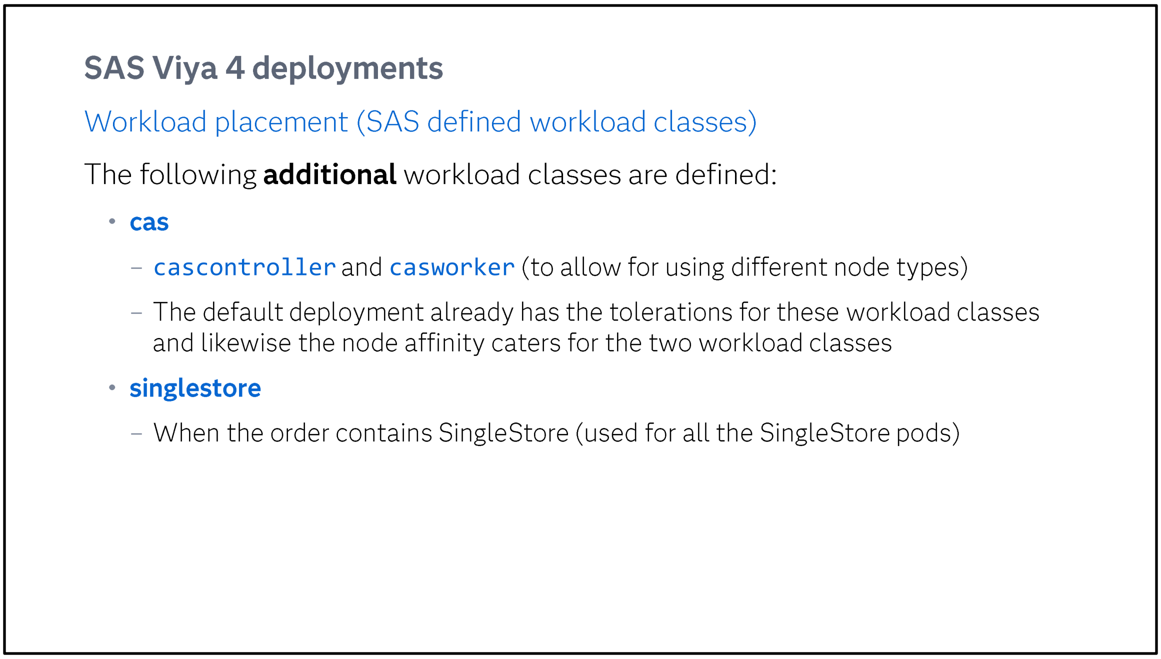
Now let’s discuss workload placement.

There are a number of SAS defined workload classes. The following default workload classes are defined: stateless, stateful, CAS, and compute.

The workload classes are defined using the naming schema of workload.sas.com/class. An example for the stateless workload class is shown here.

SAS recommends that the workload class is used to label and taint the Kubernetes nodes to achieve the default workload placement. This is explained in the SAS Viya Platform Operations guide.

Additional configuration maybe required to achieve specific deployment topologies. For example, when using a shared Kubernetes cluster for multiple SAS Viya deployments, and you wanted to dedicate some nodes to a specific deployment, then additional labels, taints and tolerations would need to be defined.



SAS Viya 4 deployments

Workload placement (SAS defined workload classes)

The following **additional** workload classes are defined:

- **cas**
  - `cascontroller` and `casworker` (to allow for using different node types)
  - The default deployment already has the tolerations for these workload classes and likewise the node affinity caters for the two workload classes
- **singlestore**
  - When the order contains SingleStore (used for all the SingleStore pods)

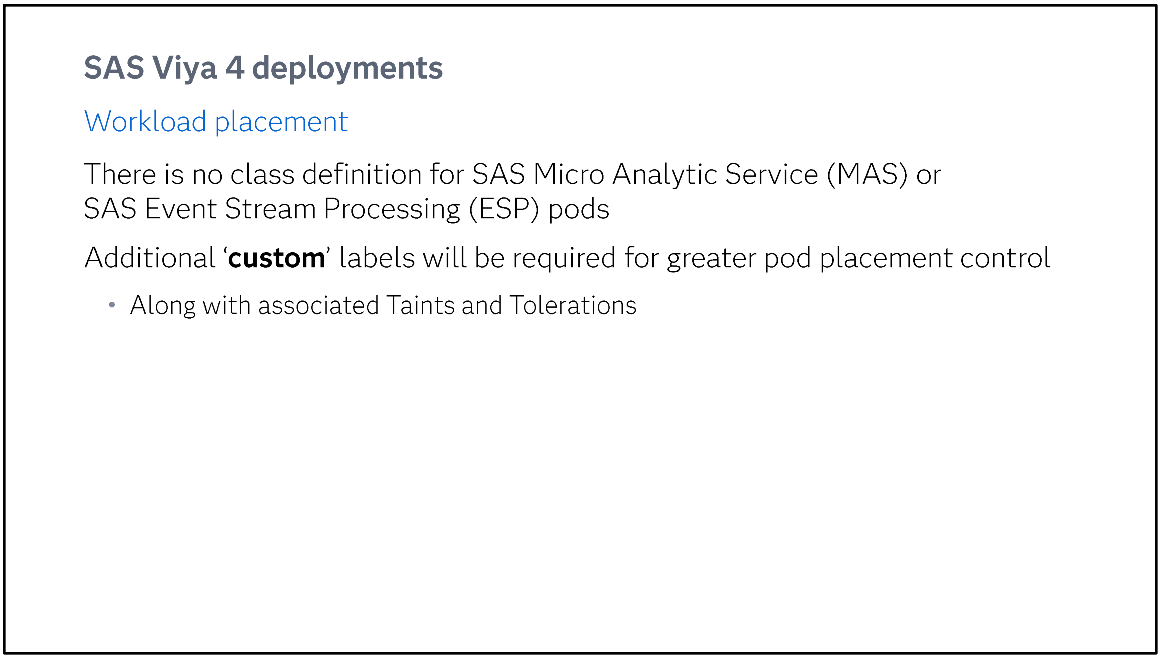
In addition to the default classes described on the previous slide, there are additional workload classes defined for CAS and SingleStore.

For CAS, there is a CAS controller and CAS worker class. This is to allow you to use different node types for the CAS controller and CAS worker pods.

This is a reflection of the different resource consumption or the resource footprint, if you like, for the controller versus the workers.

The default deployment manifest already have the tolerations for these workload classes, and likewise the node affinity caters for the two workload classes.

If the license includes SingleStore, a SAS SpeedyStore license, there is also a workload class for SingleStore. It should be noted that this product was formerly named ‘SAS with SingleStore’.



SAS Viya 4 deployments

Workload placement

There is no class definition for SAS Micro Analytic Service (MAS) or
SAS Event Stream Processing (ESP) pods

Additional ‘**custom**’ labels will be required for greater pod placement control

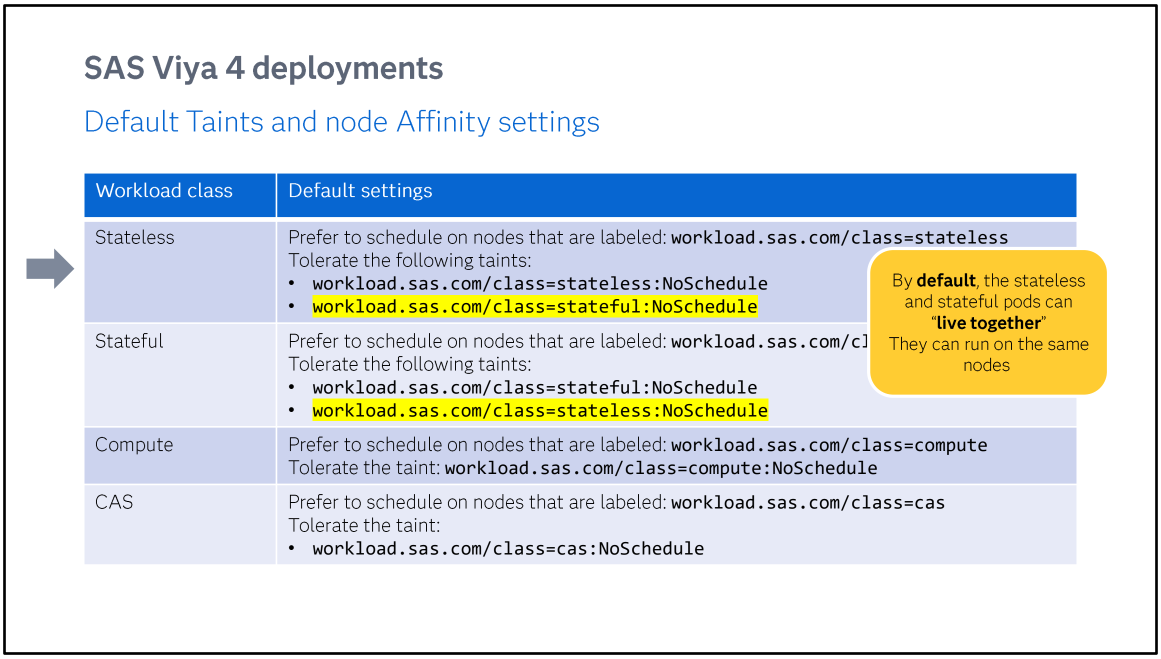
- Along with associated Taints and Tolerations

It is important to note that there is no workload class definition for SAS Micro Analytic Service, sometimes referred to as MAS, or SAS Event Stream Processing.

By default, these are treated as stateless services.

Therefore, additional custom labels and taints would be required for greater control over pod placement. For example, perhaps there is the requirement to dedicate nodes to these functions.

Along with the custom taints, the associated pod tolerations would have to be defined.



SAS Viya 4 deployments

Default Taints and node Affinity settings

Workload class	Default settings
Stateless	Prefer to schedule on nodes that are labeled: `workload.sas.com/class=stateless` Tolerate the following taints: • `workload.sas.com/class=stateless:NoSchedule` • `workload.sas.com/class=stateful:NoSchedule`
Stateful	Prefer to schedule on nodes that are labeled: `workload.sas.com/cl` Tolerate the following taints: • `workload.sas.com/class=stateful:NoSchedule` • `workload.sas.com/class=stateless:NoSchedule`
Compute	Prefer to schedule on nodes that are labeled: `workload.sas.com/class=compute` Tolerate the taint: `workload.sas.com/class=compute:NoSchedule`
CAS	Prefer to schedule on nodes that are labeled: `workload.sas.com/class=cas` Tolerate the taint: • `workload.sas.com/class=cas:NoSchedule`

By **default**, the stateless and stateful pods can "**live together**"
They can run on the same nodes

Now let's look at the default workload class taints and node affinity in more detail.

In the table the left column is showing the workload classes. The second column details the node affinities and tolerations.

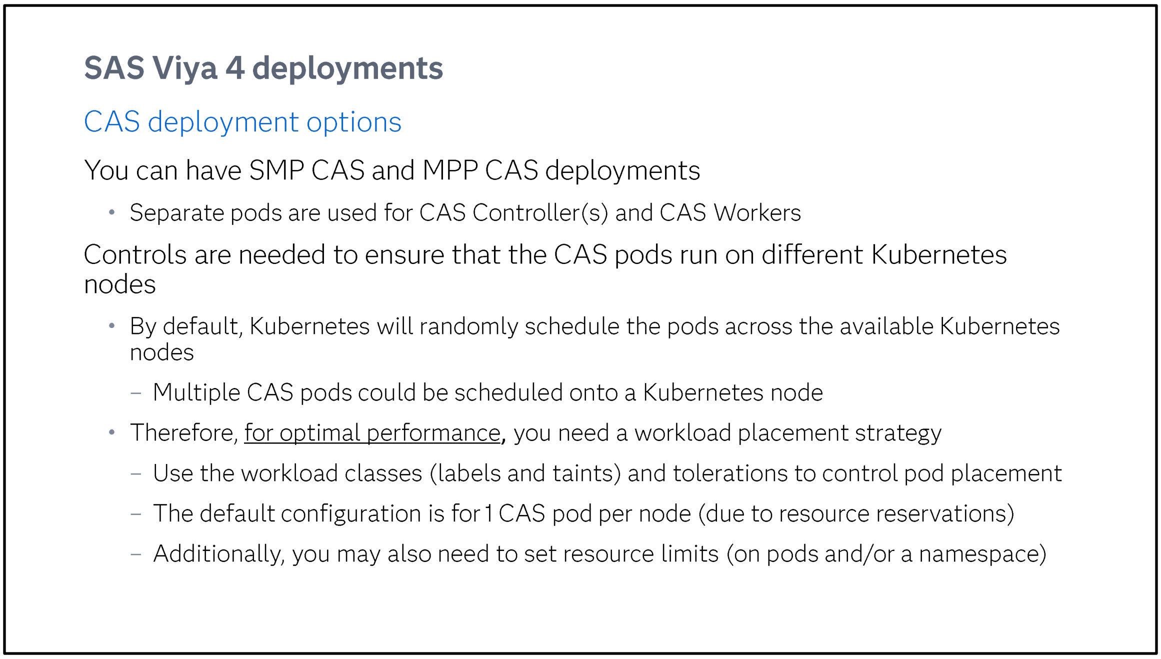
For example, the stateless workload class for the stateless pods, these components prefer to be scheduled on nodes that have the label of workload.sas.com/class=stateless. But they can tolerate both the stateless and the stateful taints.

Likewise, the stateful pods prefer to be scheduled on nodes that have the label of workload.sas.com/class=stateful. But again, they can tolerate both the stateless and the stateful taints.

Therefore, by default, the stateless and stateful pods can live together, they can run on the same nodes.

For our compute workload class, the compute pods, they prefer to be on nodes that have that compute label, and they have a toleration for the compute taint.

Finally, the CAS workload class. The CAS pods prefer to run on nodes with the CAS workload class label, and they have a toleration for the CAS workload taint.



SAS Viya 4 deployments

CAS deployment options

You can have SMP CAS and MPP CAS deployments

- Separate pods are used for CAS Controller(s) and CAS Workers

Controls are needed to ensure that the CAS pods run on different Kubernetes nodes

- By default, Kubernetes will randomly schedule the pods across the available Kubernetes nodes
  - Multiple CAS pods could be scheduled onto a Kubernetes node
- Therefore, <u>for optimal performance</u>, you need a workload placement strategy
  - Use the workload classes (labels and taints) and tolerations to control pod placement
  - The default configuration is for 1 CAS pod per node (due to resource reservations)
  - Additionally, you may also need to set resource limits (on pods and/or a namespace)

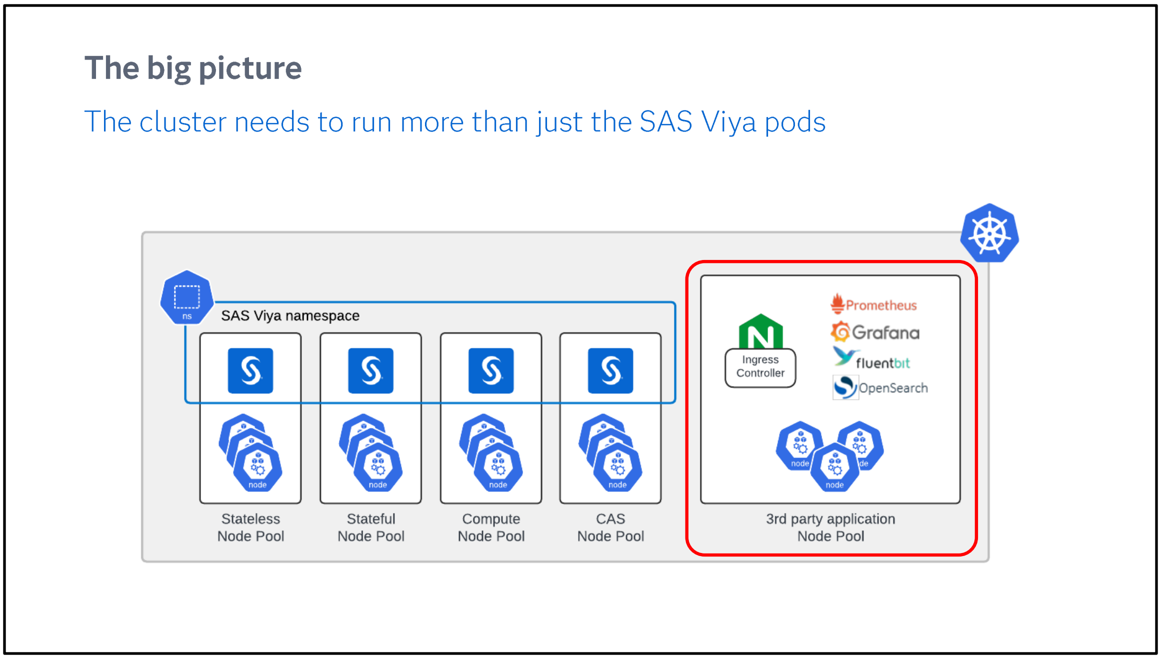
There are a number of CAS deployment options. You can have an SMP CAS server or a MPP CAS server deployment. The MPP CAS server separates the pods used for the CAS controller and the CAS workers.

Controls are needed to ensure that the CAS pods run on different Kubernetes nodes.

By default, Kubernetes will randomly schedule the pods across the available Kubernetes nodes. Without any controls, you could end up with multiple CAS pods scheduled onto the same node.

Therefore, for optimal performance, you need a workload placement strategy. The use of workload classes, labels and taints and tolerations to control the pod placement. This is not just for CAS, it's for all the components. The default configuration for CAS, however, is for one CAS pod per node, and this is due to the resource reservations.

Additionally, you may also need to set resource limits on the pods or a namespace. Setting resource limits on the Viya namespace may be required if sharing the Kubernetes cluster with other applications and or other SAS Viya deployments.



The big picture

The cluster needs to run more than just the SAS Viya pods

If we step back and consider the Kubernetes cluster as a whole.

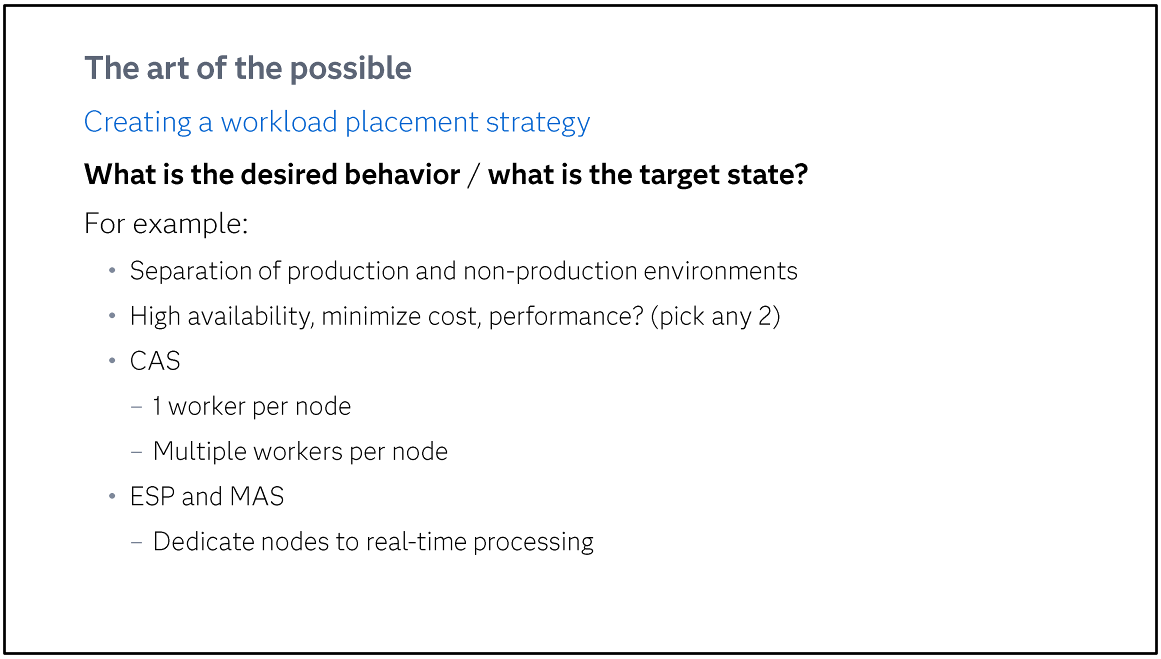
The cluster needs to run more than just the SAS Viya pods. There could be other applications running in the cluster, if it's not dedicated to SAS Viya. But for SAS Viya, there is prerequisites software that also needs to run.

For example, components like the Ingress Controller. Or it could be that you're using the SAS Viya monitoring for Kubernetes tools, this includes Prometheus, Grafana, Fluent-bit, as shown on the right-hand side of the diagram. Here you can see that they are running in a node pool that has been dedicated to the third party applications.

The art of the possible

Creating a workload placement strategy

Let's look at creating that workload placement strategy.



The art of the possible

Creating a workload placement strategy

What is the desired behavior / what is the target state?

For example:

- Separation of production and non-production environments
- High availability, minimize cost, performance? (pick any 2)
- CAS
  - 1 worker per node
  - Multiple workers per node
- ESP and MAS
  - Dedicate nodes to real-time processing

To create a workload placement strategy, or plan, it is important to understand what you are trying to achieve. What is the desired outcome?

What is the desired behavior or the desired state of the platform. The slide illustrates some of the possible drivers.

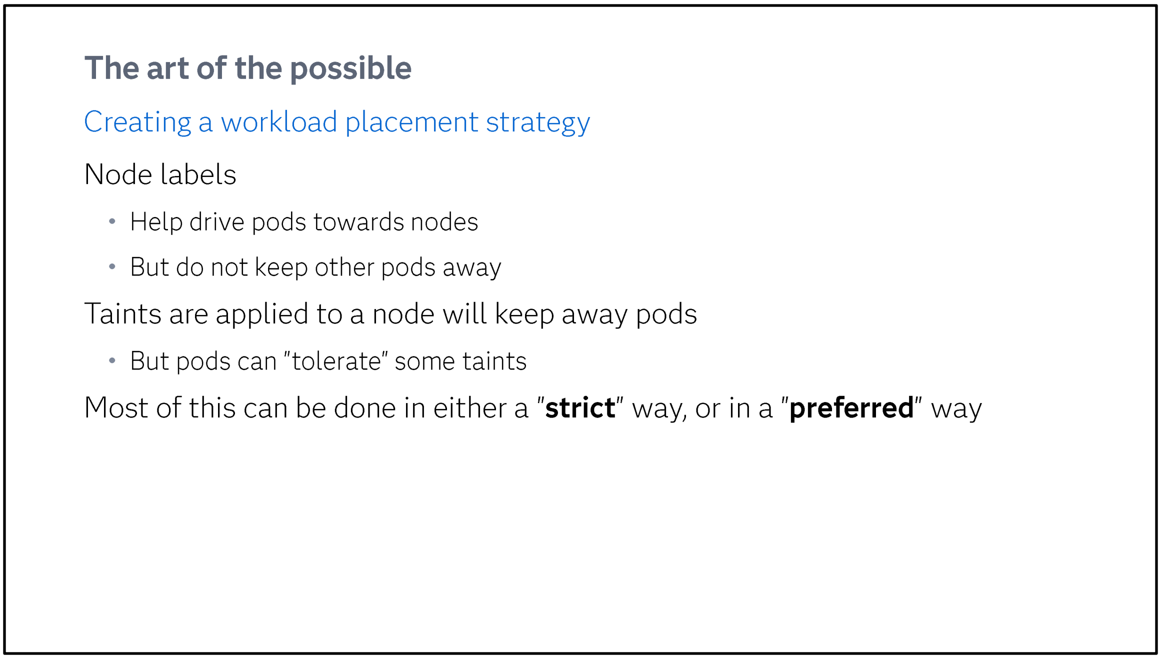
For example, perhaps it is the separation of production and non-production environments.

Or maybe, it's high availability or to minimize cost or a focus on good performance. Pick two, you can't have all three.

For CAS, maybe it's one worker per node or multiple workers on a node.

For ESP and MAS, perhaps it’s to use dedicated nodes for the real-time processing, to protect that processing from other workloads.

In order to create the workload placement plan, or strategy, it is important to understand the requirements and drivers for the SAS Viya platform.



The art of the possible

Creating a workload placement strategy

Node labels

- Help drive pods towards nodes
- But do not keep other pods away

Taints are applied to a node will keep away pods

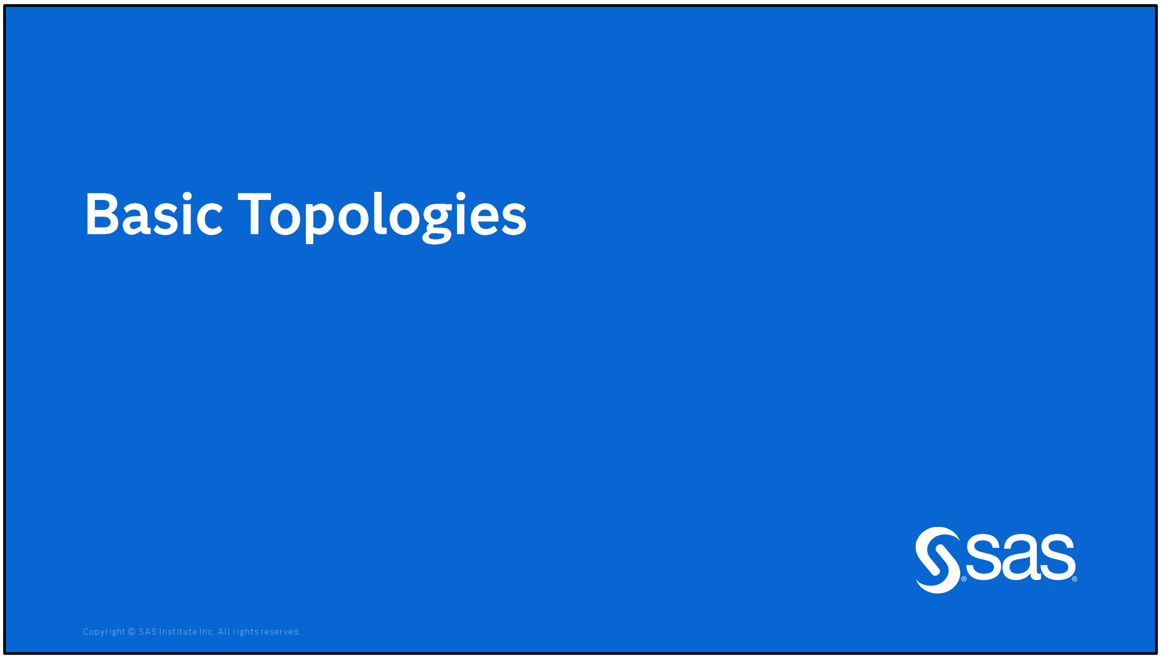
- But pods can "tolerate" some taints

Most of this can be done in either a "**strict**" way, or in a "**preferred**" way

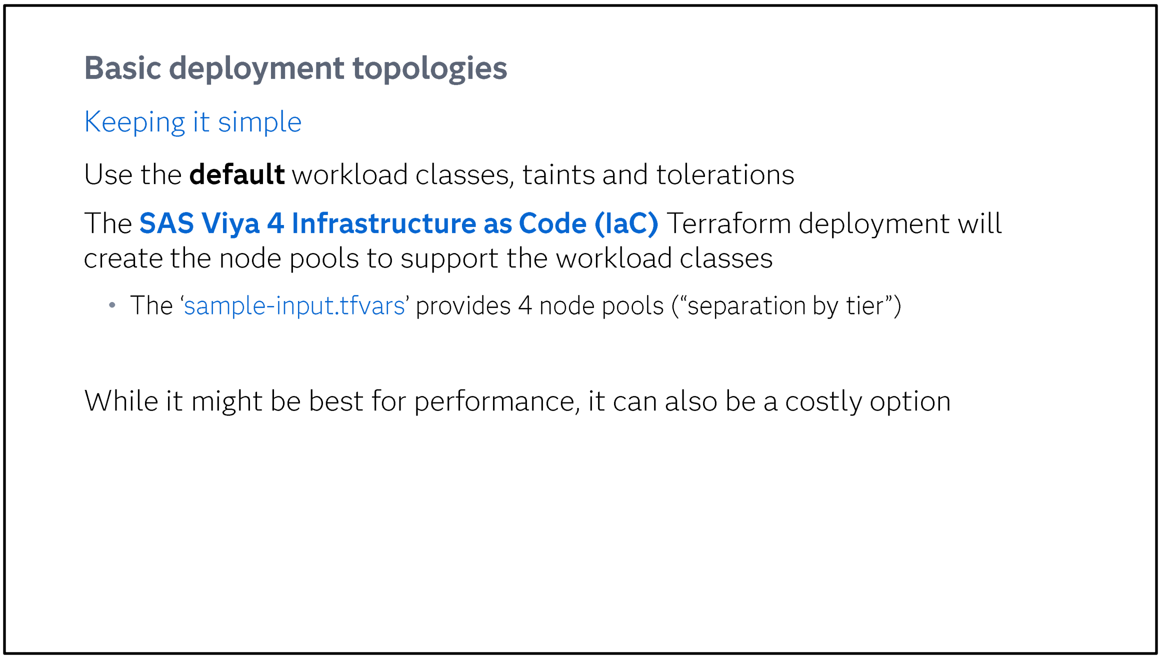
Finally, when you think about creating that workload placement strategy, or plan, remember that node labels help drive pods towards a node. They are used to attract pods, but they do not keep pods away.

Taints are applied to a node, a taint is used to keep pods away, to repel unwanted pods, but a pod can be given a toleration for a taint.

The node affinity can be implemented in either a strict or preferred way.



Basic Topologies



Basic deployment topologies

Keeping it simple

Use the **default** workload classes, taints and tolerations

The **SAS Viya 4 Infrastructure as Code (IaC)** Terraform deployment will create the node pools to support the workload classes

- The 'sample-input.tfvars' provides 4 node pools ("separation by tier")

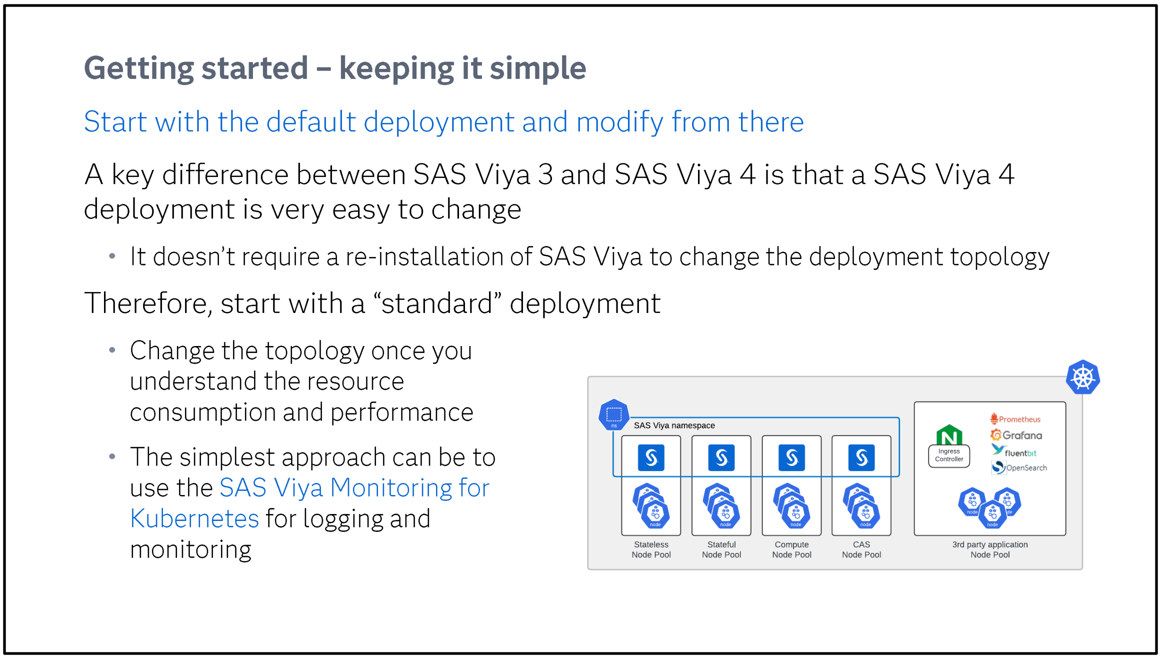
While it might be best for performance, it can also be a costly option

In this module, we're going to look at some basic deployment topologies, using a dedicated Kubernetes cluster with a single SAS Viya deployment.

We'll start by keeping it simple using the default workload classes, taints, and tolerations.

The SAS Viya infrastructure as code Terraform deployment will create the node pools to support the workload classes. The sample inputs Terraform file, called sample dash input dot TF vars provides four node pools. You could call this separation by tier.

It might be a good approach for performance, but it could also be a costly option, from a Kubernetes infrastructure perspective.



Getting started – keeping it simple

Start with the default deployment and modify from there

A key difference between SAS Viya 3 and SAS Viya 4 is that a SAS Viya 4 deployment is very easy to change

- It doesn't require a re-installation of SAS Viya to change the deployment topology

Therefore, start with a "standard" deployment

- Change the topology once you understand the resource consumption and performance
- The simplest approach can be to use the SAS Viya Monitoring for Kubernetes for logging and monitoring

Let's get started by keeping it simple.

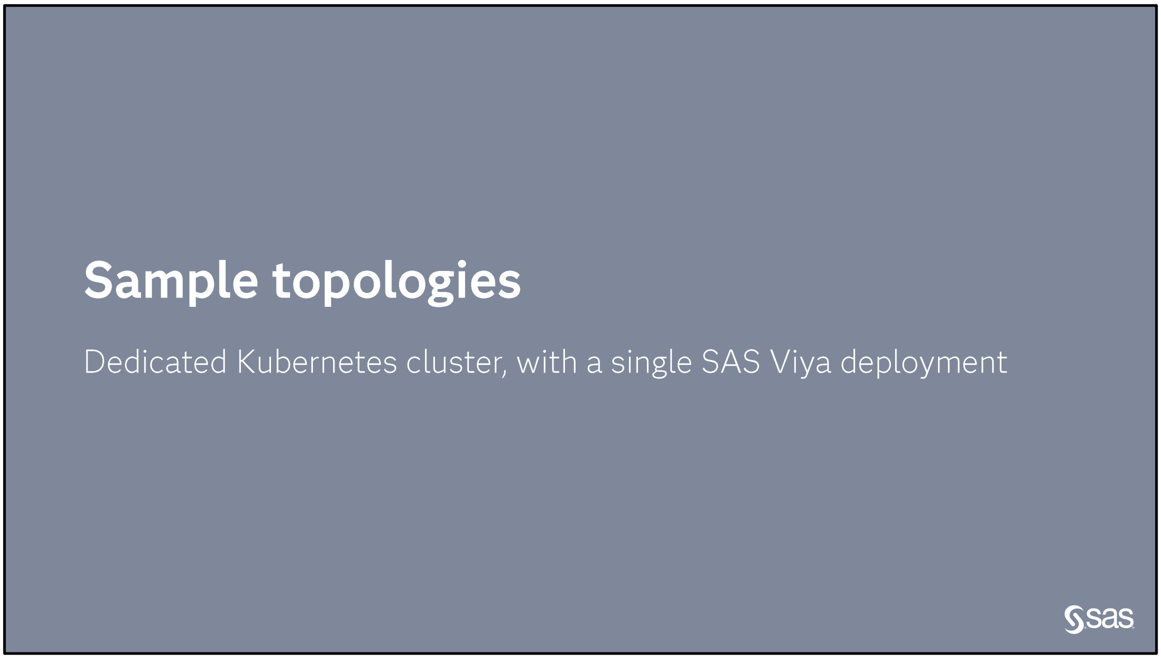
You can start with the default deployment and modify it from there.

A key difference between SAS Viya 3 and SAS Viya 4 is that a SAS Viya 4 deployment is very easy to change. It doesn't require a re-installation of SAS Viya to change the deployment topology. You don't have to uninstall an existing server and start from scratch again.

Therefore, start with a standard deployment and modify from there. Here on the right, you can see we've got my Kubernetes cluster. There are four node pools for the SAS Viya platform, the stateless, stateful, compute, and CAS node pools. And there is a fifth node pool for third-party applications.

So, start with the default topology, and change the topology once you understand the resource consumption and performance. To do this, you'll need to monitor the deployment.

The simplest approach can be to use the SAS Viya 4 monitoring for Kubernetes GitHub project. SAS Viya Monitoring for Kubernetes provides scripts and customization options to deploy metric monitoring, alerts, and log message aggregation for SAS Viya.

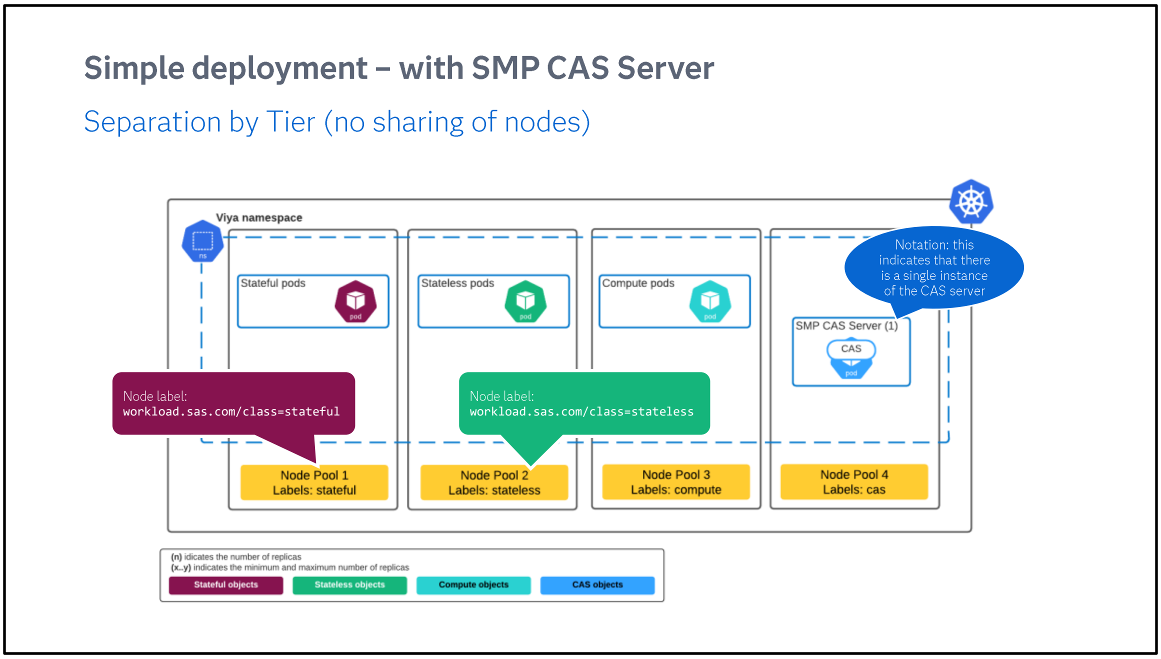


Sample topologies

Dedicated Kubernetes cluster, with a single SAS Viya deployment

Now let's look at some sample topologies using a dedicated Kubernetes cluster with a single SAS Viya deployment.

The following set of slides show some sample topologies based on using the default workload classes.



Simple deployment – with SMP CAS Server

Separation by Tier (no sharing of nodes)

Notation: this indicates that there is a single instance of the CAS server

Node label:
`workload.sas.com/class=stateful`

Node label:
`workload.sas.com/class=stateless`

(n) idicates the number of replicas
(x..y) indicates the minimum and maximum number of replicas

Stateful objects | Stateless objects | Compute objects | CAS objects

We will start by looking at separation by tier with no sharing of nodes. Here you can see my Kubernetes cluster, it has four node pools dedicated to the SAS Viya platform.

Here you can see the stateful pods are all running on the stateful node pool, node pool one. The stateless pods are all running on the stateless node pool. The compute pods are running on node pool 3. And finally, the CAS pods are running on node pool 4.

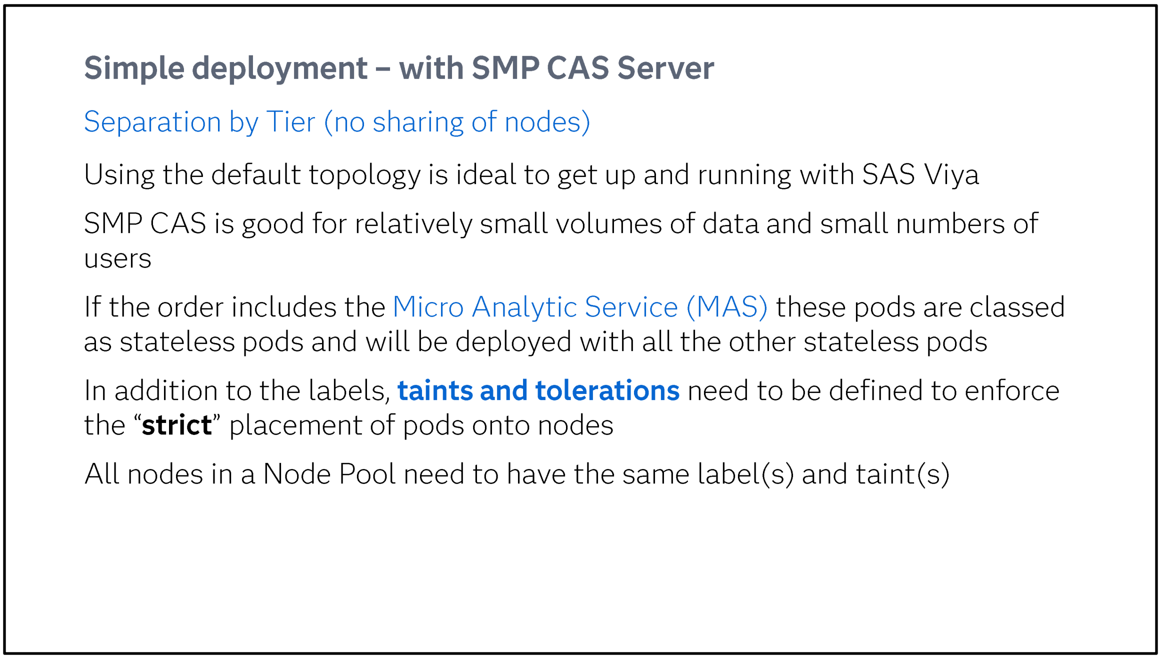
The nodes all have the default SAS workload class labels and taints applied.

For example, node pool one is for the stateful nodes. The workload class label and taint is: workload.sas.com/class=stateful. And, the green box shows the workload class for the stateless nodes.

We will discuss the labels and taints in more detail as we go through this module.

Using the default labels and taints, using the default topology, is a good way to get up and running. The default topology also allows for the stateful and stateless workloads to share nodes, but more on that later.

A final note on notation. The number in the brackets indicates that there is a single instance of the CAS server. You will see this type of notation on many of the slides.



Simple deployment – with SMP CAS Server

Separation by Tier (no sharing of nodes)

Using the default topology is ideal to get up and running with SAS Viya

SMP CAS is good for relatively small volumes of data and small numbers of users

If the order includes the Micro Analytic Service (MAS) these pods are classed as stateless pods and will be deployed with all the other stateless pods

In addition to the labels, **taints and tolerations** need to be defined to enforce the “**strict**” placement of pods onto nodes

All nodes in a Node Pool need to have the same label(s) and taint(s)

As previously stated, using the default topology is ideal to get up and running with SAS Viya.

An SMP CAS server is good for relatively small volumes of data and small numbers of users.

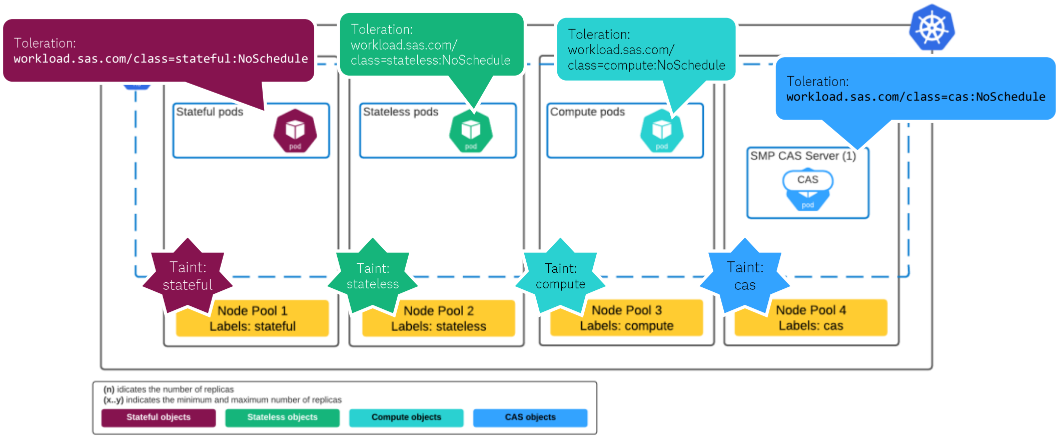
If the order includes the micro-analytic service or MAS, these pods are classed as stateless pods and will be deployed with all the other stateless pods. There is no specific workload class for MAS or real-time processing.

In addition to the labels, taints and tolerations need to be defined to enforce the strict placements of pods onto nodes. The SAS Viya manifests are setup to have node affinities for the default workload classes and tolerations for the default workload taints.

Finally, all nodes in a node pool need to have the same labels and taints applied.

Simple deployment – with SMP CAS Server

Separation by Tier (no sharing of nodes)



Let's look at this example of a deployment using an SMP CAS server. Once again, you can see my Kubernetes cluster with the four node pools.

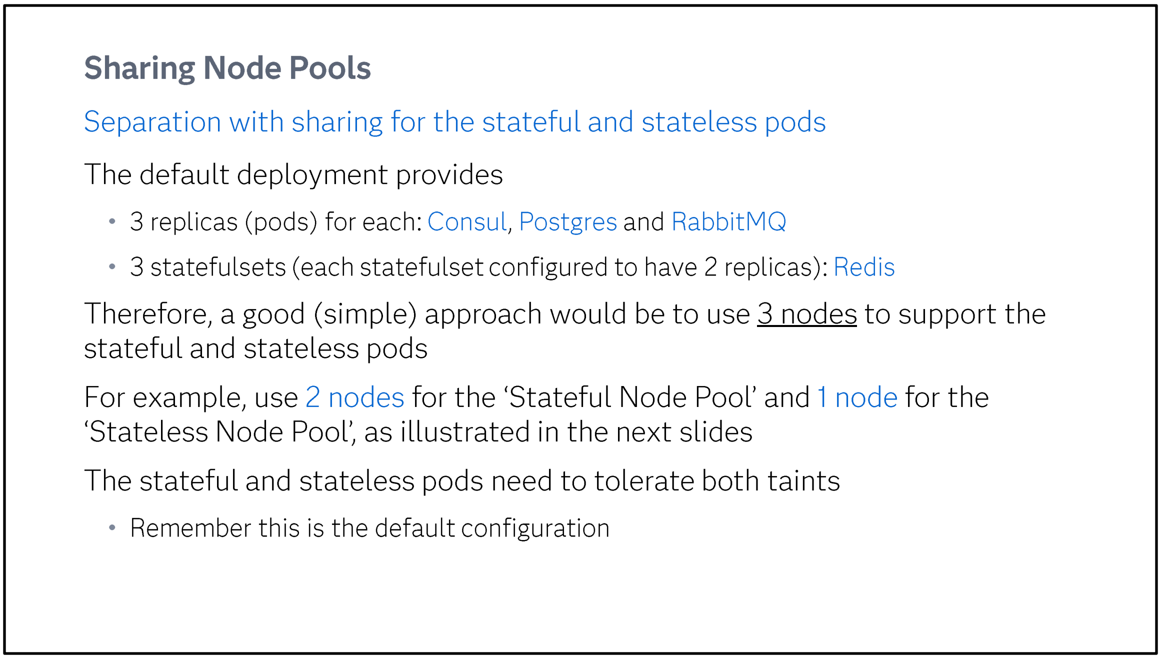
The nodes have all been labeled with SAS workload class labels. For example, the nodes in node pool one have the label: workload.sas.com/class=stateful applied.
The nodes in node pool two have the label: workload.sas.com/class=stateless. Similarly, the compute and CAS nodes have been labeled.

The associated workload taints have been applied to the nodes within each node pool. Remember, labels are used to attract pods and taints are used to repel, or to keep pods away.

Now that the nodes have the taints applied. To implement the default workload topology, the pods need a toleration for the taints. The tolerations are shown in the colored boxes.

The stateful pods have a toleration for the taint, workload.sas.com/class=stateful. Likewise, the stateless pods have a toleration for that stateless taint. The compute pods have a toleration for the compute taint. And finally, the CAS pods have a toleration for the CAS taint.

The SAS Viya manifests are using preferred scheduling, so if there are sufficient nodes available in the stateful and stateless node pools the pods will naturally separate by tier for these components.



Sharing Node Pools

Separation with sharing for the stateful and stateless pods

The default deployment provides

- 3 replicas (pods) for each: Consul, Postgres and RabbitMQ
- 3 statefulsets (each statefulset configured to have 2 replicas): Redis

Therefore, a good (simple) approach would be to use 3 nodes to support the stateful and stateless pods

For example, use 2 nodes for the ‘Stateful Node Pool’ and 1 node for the ‘Stateless Node Pool’, as illustrated in the next slides

The stateful and stateless pods need to tolerate both taints

- Remember this is the default configuration

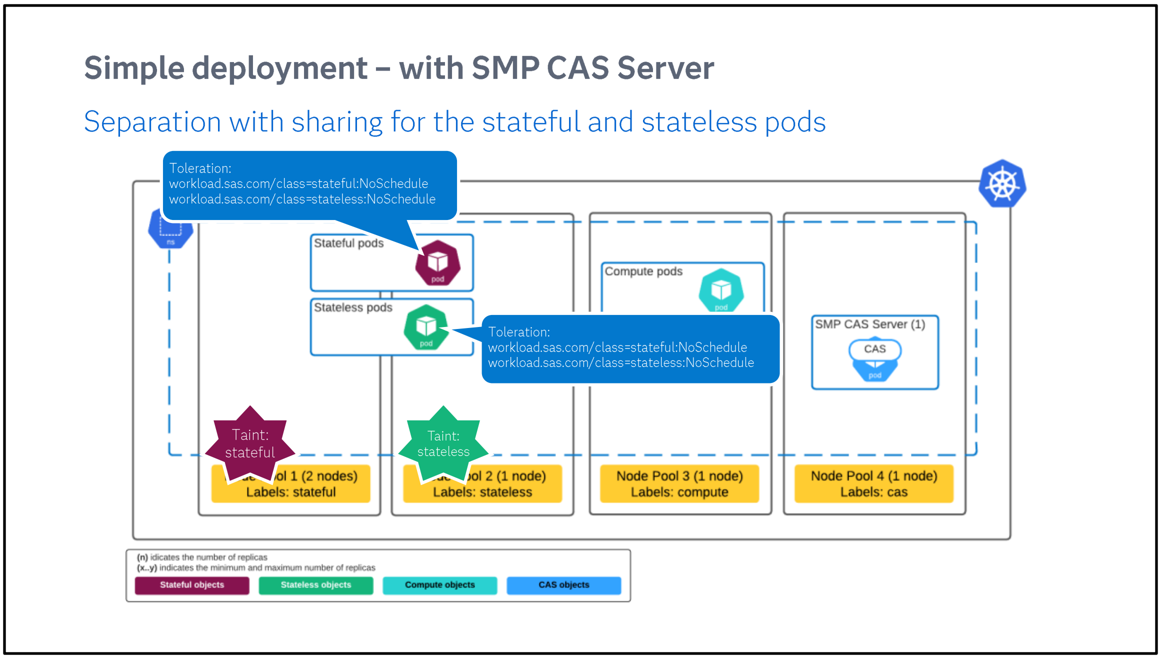
Now let’s look at the case of sharing the stateful and stateless nodes. This is the default configuration in the SAS Viya manifests.

The default deployment provides three replicas for Consul, Postgres, and RabbitMQ. The Red-is server has three stateful-sets. With each stateful-set configured to have two replicas.

Therefore, a good, or simple approach, would be to use a minimum of 3 nodes to support the stateful and stateless pods. This provides some redundancy for these core components.

For example, you could use two nodes for the Stateful Node Pool and one node for the Stateless Node Pool, this is illustrated in the following slides.

To implement this topology, the stateful and stateless pods need to tolerate both taints. This is the default configuration.



Simple deployment – with SMP CAS Server

Separation with sharing for the stateful and stateless pods

Looking at this in more detail.

Once again, you can see my Kubernetes cluster. This time you can see the number of nodes in the node pools. The stateful node pool has two nodes, the stateless, compute and CAS node pools each have one node.

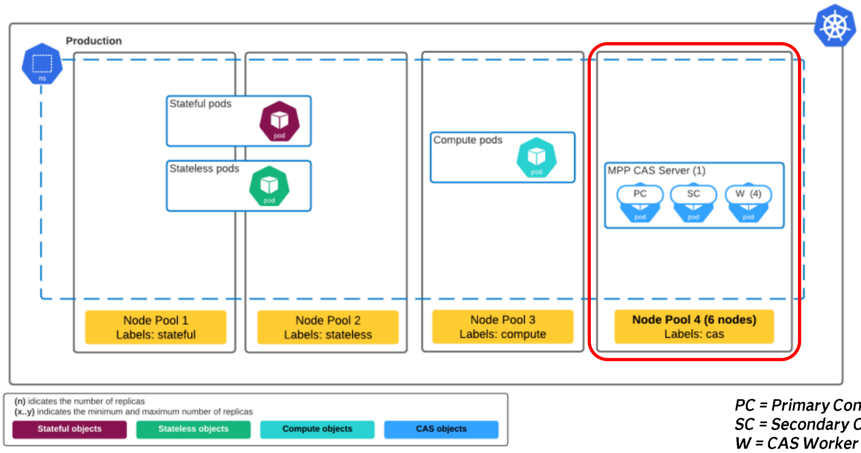
The default workload class labels have been applied to the nodes.

The taints have been applied to all the nodes, but we will just focus on the stateful and stateless nodes for the moment.

We can see that the stateful pods have a toleration for both the stateful and stateless taints. And likewise, the stateless pods have a toleration for the stateful and the stateless taints, which means they can run on either node pool.

Larger deployment – with MPP CAS Server

Separation with sharing for the stateful and stateless pods

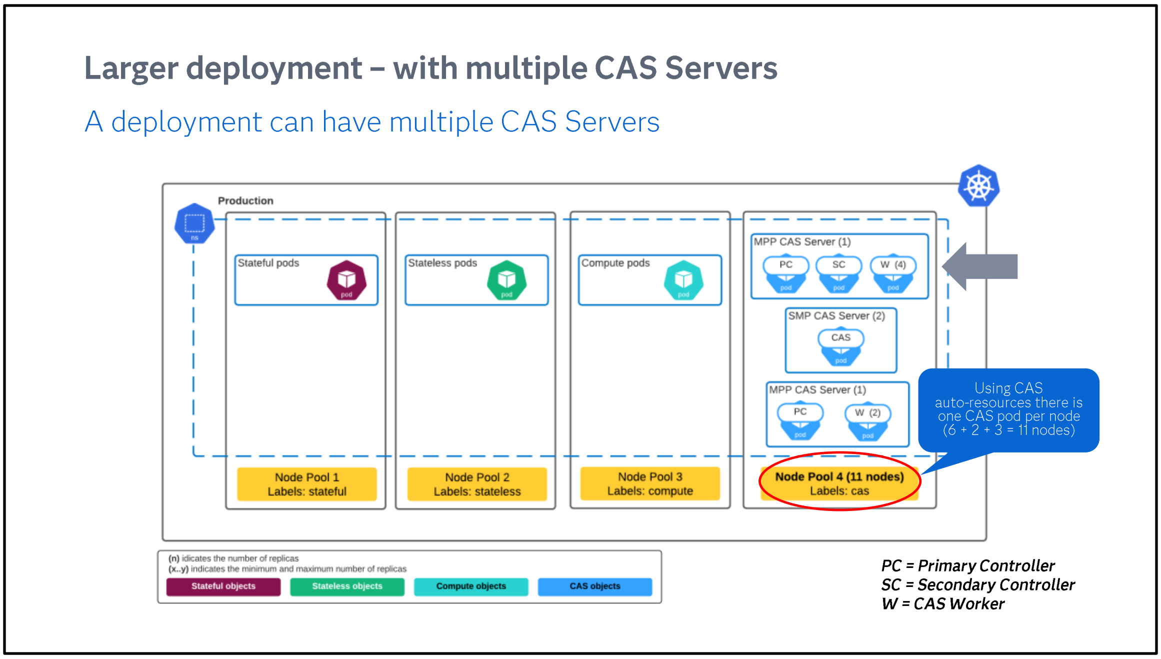


PC = Primary Controller
SC = Secondary Controller
W = CAS Worker

Now looking at the default deployment, this time using a MPP CAS server. For larger deployments, you might use an MPP CAS server, this is shown in node pool four.

This time node pool four has six nodes. Assuming that “CAS auto resources” is configured there will be one CAS pod on each nodes.

As our MPP CAS server has a Primary Controller, a backup or secondary controller and four workers, then the CAS node pool must have six active nodes in order to run this configuration.



Larger deployment – with multiple CAS Servers

A deployment can have multiple CAS Servers

Using CAS auto-resources there is one CAS pod per node (6 + 2 + 3 = 11 nodes)

PC = Primary Controller
SC = Secondary Controller
W = CAS Worker

You can also have multiple CAS servers within your deployment, as shown here. Looking at the diagram you can see that there are two MPP CAS servers and two SMP CAS servers.

Working from top to bottom.

The first MPP CAS server has a Primary Controller, a Secondary Controller and four workers.

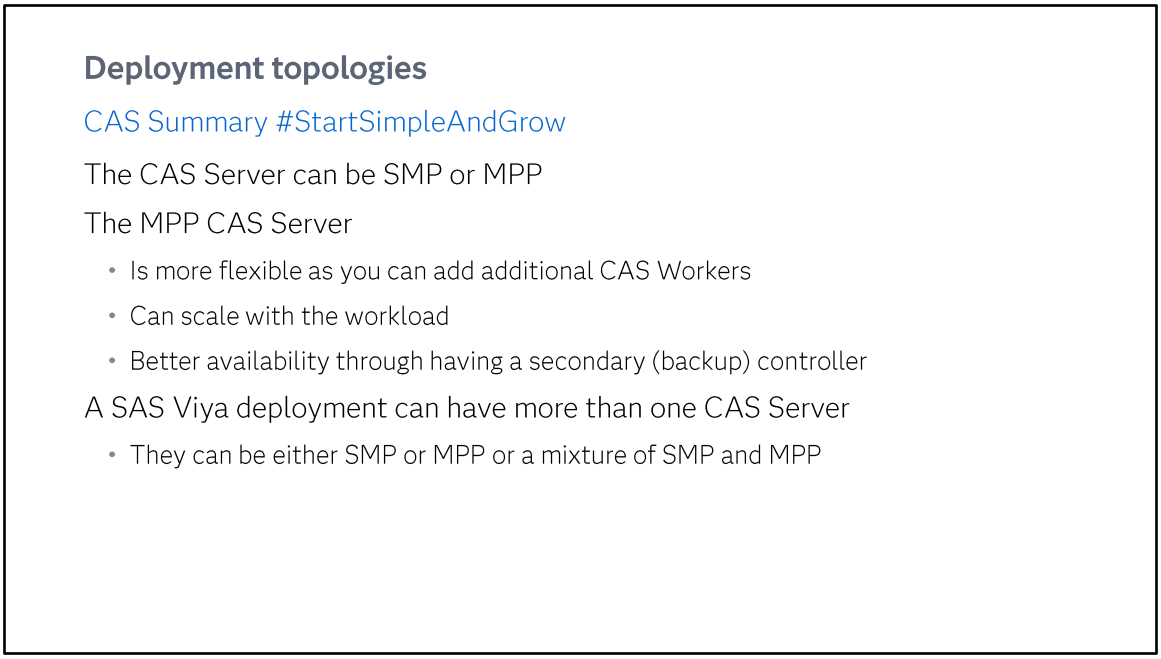
In the middle the notation is showing that there are two SMP CAS servers.

And finally at the bottom there is a MPP CAS server with a Primary Controller and two workers.

This is also illustrating that you can deploy a mixture of SMP and MPP CAS servers.

Again, if CAS auto resources is configured, in this case for all the CAS servers, then eleven nodes are needed in the CAS node pool. As there would be one CAS pod per node.

If we add this up. The first CAS server needs six nodes. We've got two instances of an SMP CAS server, this requires two nodes.
And finally, the second SMP CAS server needs three nodes. That gives us our 11 nodes.



Deployment topologies

CAS Summary #StartSimpleAndGrow

The CAS Server can be SMP or MPP

The MPP CAS Server

- Is more flexible as you can add additional CAS Workers
- Can scale with the workload
- Better availability through having a secondary (backup) controller

A SAS Viya deployment can have more than one CAS Server

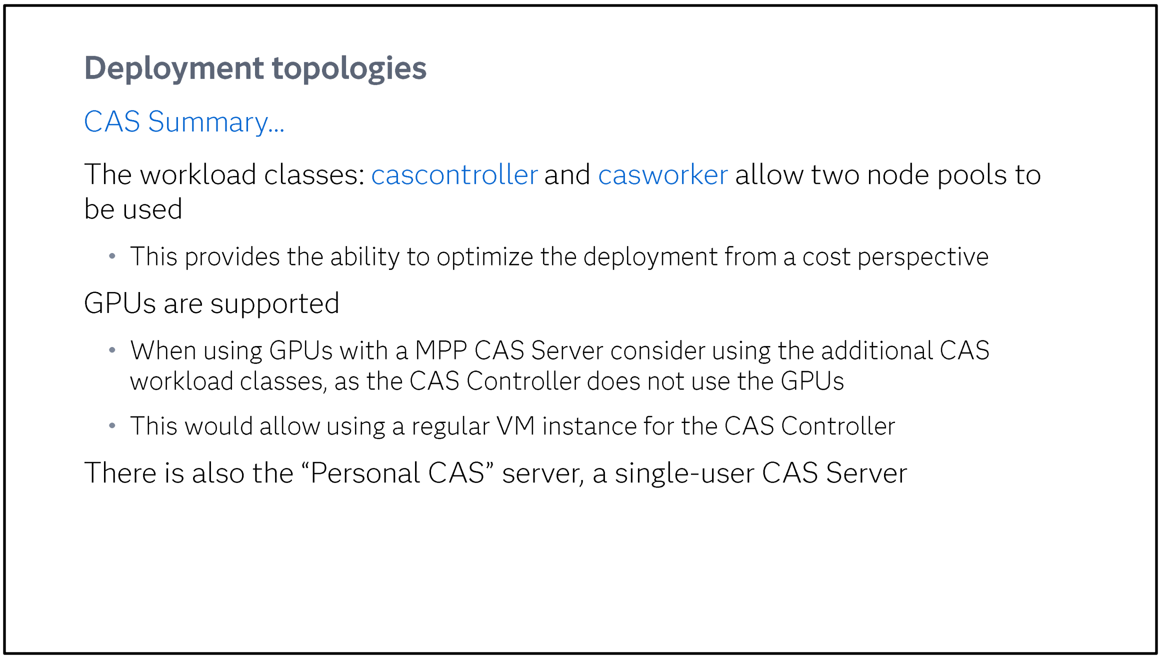
- They can be either SMP or MPP or a mixture of SMP and MPP

In summary, start with a simple deployment and grow.

The CAS server can be SMP or MPP.

For the MPP CAS server:
The MPP CAS server is more flexible as you can add additional workers,
and it can scale with the workload.
It provides better availability through the ability to have a backup controller, as well as the multiple workers.

As we have seen, the SAS Viya deployment can have multiple CAS servers, they can be either SMP or MPP, or a mixture of SMP and MPP CAS servers.



Deployment topologies

CAS Summary…

The workload classes: cascontroller and casworker allow two node pools to be used

- This provides the ability to optimize the deployment from a cost perspective

GPUs are supported

- When using GPUs with a MPP CAS Server consider using the additional CAS workload classes, as the CAS Controller does not use the GPUs
- This would allow using a regular VM instance for the CAS Controller

There is also the “Personal CAS” server, a single-user CAS Server

Continuing the CAS summary.

The workload classes: cascontroller and casworker allow for the use of two node pools for CAS.

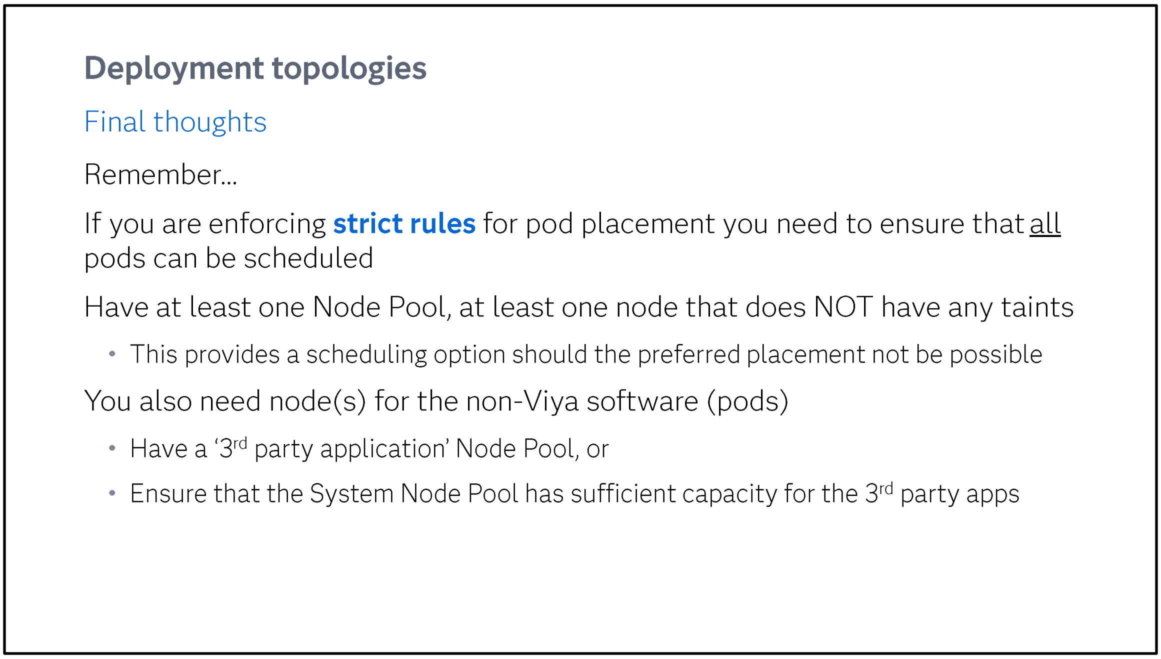
This provides the ability to optimize the deployment from a cost perspective. Allowing for the CAS Controller and CAS Workers to use different instance types.

GPUs are supported for the CAS server deployment.

When using GPUs with an MPP CAS server, consider using the additional CAS workload classes, as the CAS controller does not use GPUs.

As the CAS Controller does not use GPUs, using two node pools for CAS, provides the ability to use a regular VM instance for the CAS Controller.

Lastly, there is also the Personal CAS server, a single-user CAS Server. This is a transient SMP CAS Server, and runs alongside the Compute Server. This provides the ability to work with a CAS server that is local to a users SAS Compute session.



Deployment topologies

Final thoughts

Remember…

If you are enforcing **strict rules** for pod placement you need to ensure that all pods can be scheduled

Have at least one Node Pool, at least one node that does NOT have any taints

- This provides a scheduling option should the preferred placement not be possible

You also need node(s) for the non-Viya software (pods)

- Have a '3rd party application' Node Pool, or
- Ensure that the System Node Pool has sufficient capacity for the 3rd party apps

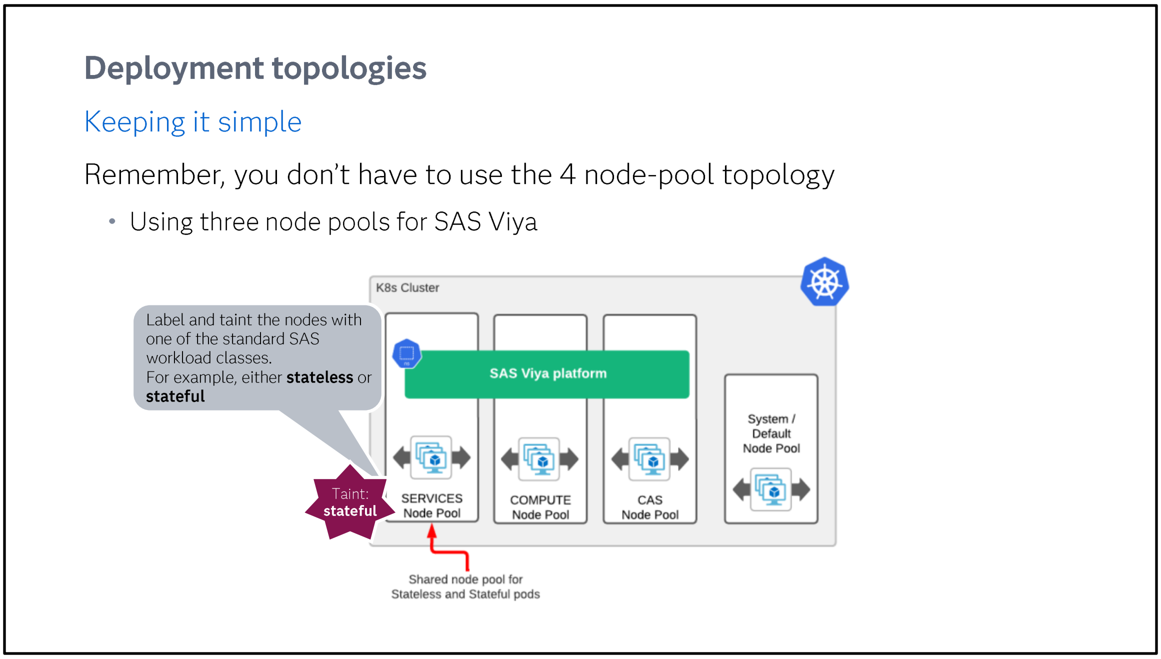
Some final thoughts.

If you are enforcing strict rules for pod placement you need to ensure that all pods can be scheduled. There needs to be sufficient nodes and resources available to be able to schedule all the pods.

When you fully label and taint the nodes for the SAS Viya workload classes you need to ensure that there are still nodes available for other components. For example, the ingress controller or perhaps the monitoring and logging components.

Have at least one Node Pool, at least one node, that does not have any taints. This can also provide a scheduling option should the preferred placement not be possible.

Remember, you always need a node, or nodes, available for the non-Viya software pods.



Deployment topologies

Keeping it simple

Remember, you don't have to use the 4 node-pool topology

- Using three node pools for SAS Viya

Label and taint the nodes with one of the standard SAS workload classes.
For example, either **stateless** or **stateful**

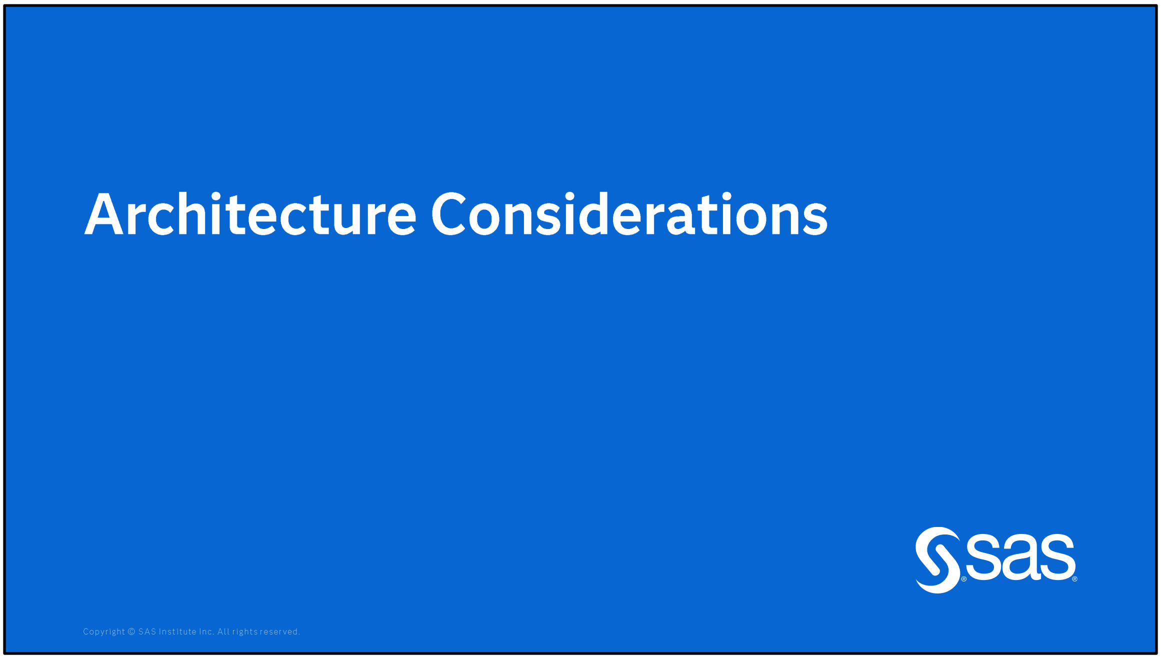
In this module we have looked at the default workload placement, or topology, using four node pools. But it is important to understand that it is not mandatory to use four node pools.

This shows an example of using three node pools for the SAS Viya platform.

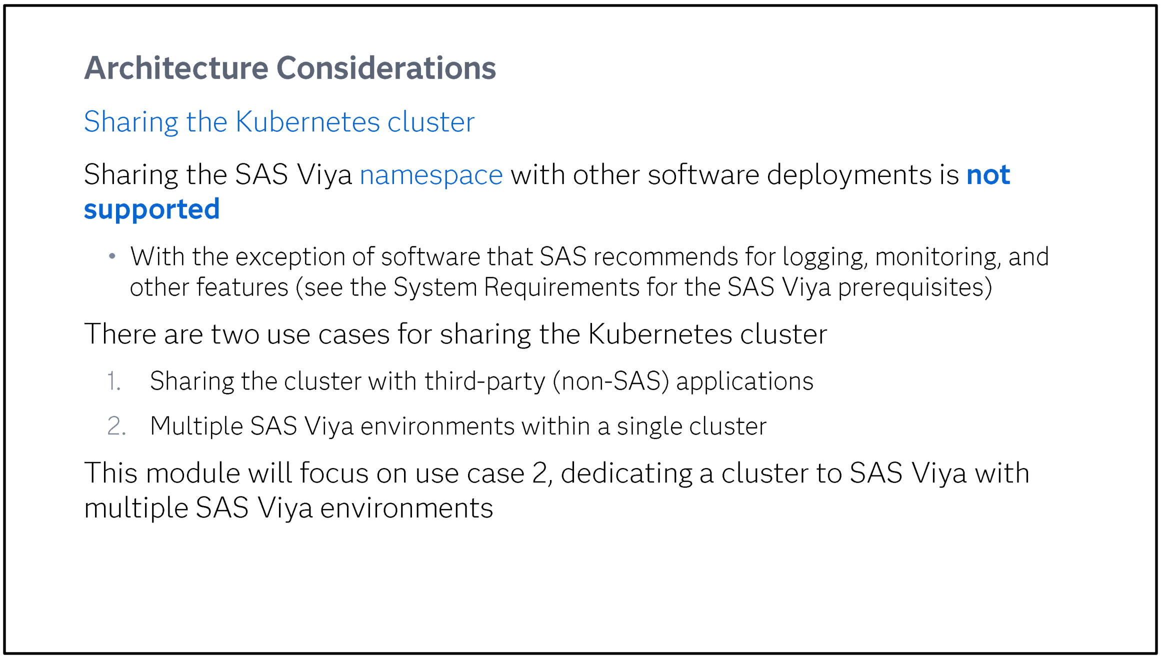
To implement this, you can still use the standard workload labels and taints. For example, the “services” node pool is being used for the stateful and stateless components. You would apply either the stateful or stateless workload label.

Then you would taint the nodes with either the stateless or stateful taint. In this example the stateful label and taint has been applied to the nodes. Remember, the stateless and stateful pods tolerate both taints by default. So, it doesn’t really matter which you use.

That concludes our look at the basic, or default, deployment topologies, with a single SAS Viya deployment.



Architecture Considerations



Architecture Considerations

Sharing the Kubernetes cluster

Sharing the SAS Viya namespace with other software deployments is **not supported**

- With the exception of software that SAS recommends for logging, monitoring, and other features (see the System Requirements for the SAS Viya prerequisites)

There are two use cases for sharing the Kubernetes cluster

1. Sharing the cluster with third-party (non-SAS) applications
2. Multiple SAS Viya environments within a single cluster

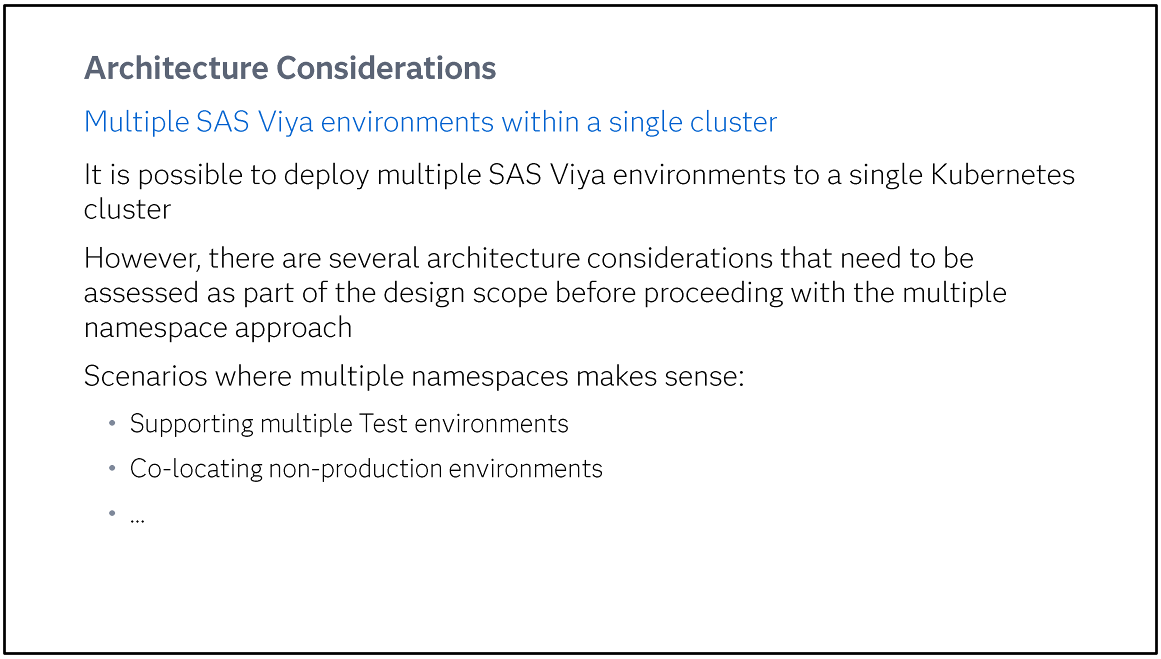
This module will focus on use case 2, dedicating a cluster to SAS Viya with multiple SAS Viya environments

This module will look at the architecture considerations for running multiple SAS Viya environments, multiple SAS Viya deployments in a shared Kubernetes cluster.

It is important to note sharing the SAS Viya namespace with other software deployments is not supported. With the exception of the software that SAS recommends for logging, monitoring, and other features. See the system requirements for more information on the SAS Viya prerequisites.

There are two use cases for sharing the Kubernetes cluster. Sharing the cluster with third-party, non-SAS applications. And, running multiple SAS Viya environments within a single cluster.

This module will focus on the second use case, dedicating a cluster to SAS Viya with multiple SAS Viya environments.



Architecture Considerations

Multiple SAS Viya environments within a single cluster

It is possible to deploy multiple SAS Viya environments to a single Kubernetes cluster

However, there are several architecture considerations that need to be assessed as part of the design scope before proceeding with the multiple namespace approach

Scenarios where multiple namespaces makes sense:

- Supporting multiple Test environments
- Co-locating non-production environments
- ...

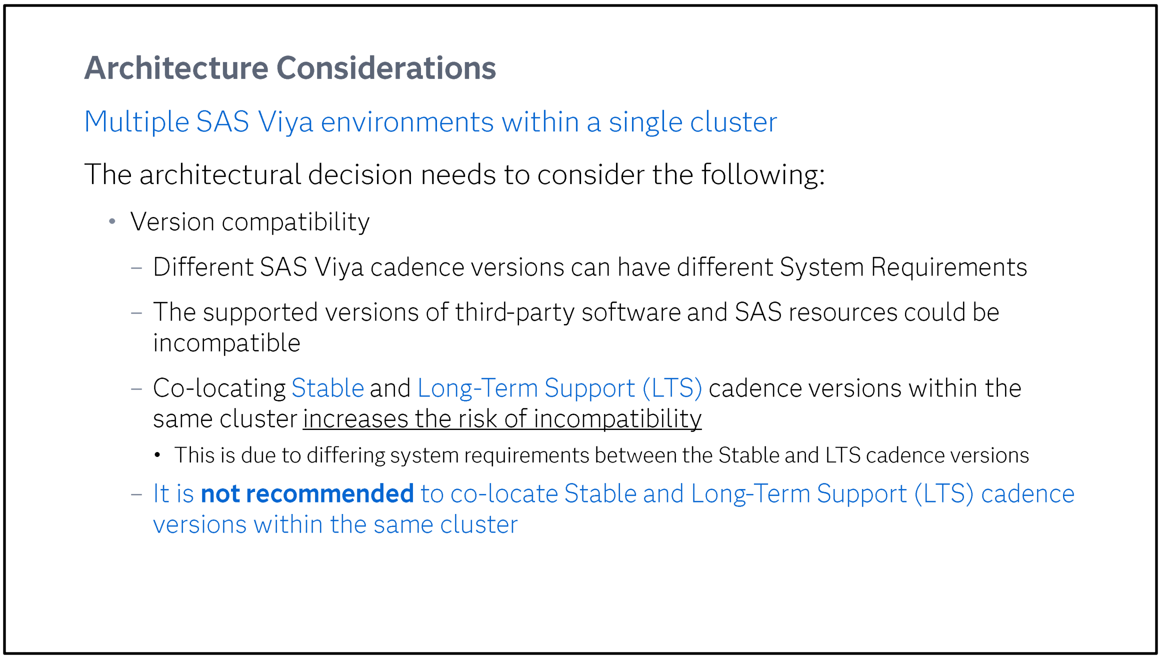
Running Multiple SAS Viya environments within a single cluster.

It is possible to deploy multiple SAS Viya environments to a single Kubernetes cluster. Each environment runs in its own namespace.

However, there are several architecture considerations that need to be assessed as part of the design scope, before proceeding with the multiple namespace approach.

Scenarios where running multiple SAS Viya platforms in a shared cluster make sense, are: supporting multiple test environments, or perhaps co-locating non-production environments. Or co-locating production environments.

I'm sure there are other scenarios as well.



Architecture Considerations

Multiple SAS Viya environments within a single cluster

The architectural decision needs to consider the following:

- Version compatibility
  - Different SAS Viya cadence versions can have different System Requirements
  - The supported versions of third-party software and SAS resources could be incompatible
  - Co-locating Stable and Long-Term Support (LTS) cadence versions within the same cluster increases the risk of incompatibility
    - This is due to differing system requirements between the Stable and LTS cadence versions
  - It is **not recommended** to co-locate Stable and Long-Term Support (LTS) cadence versions within the same cluster

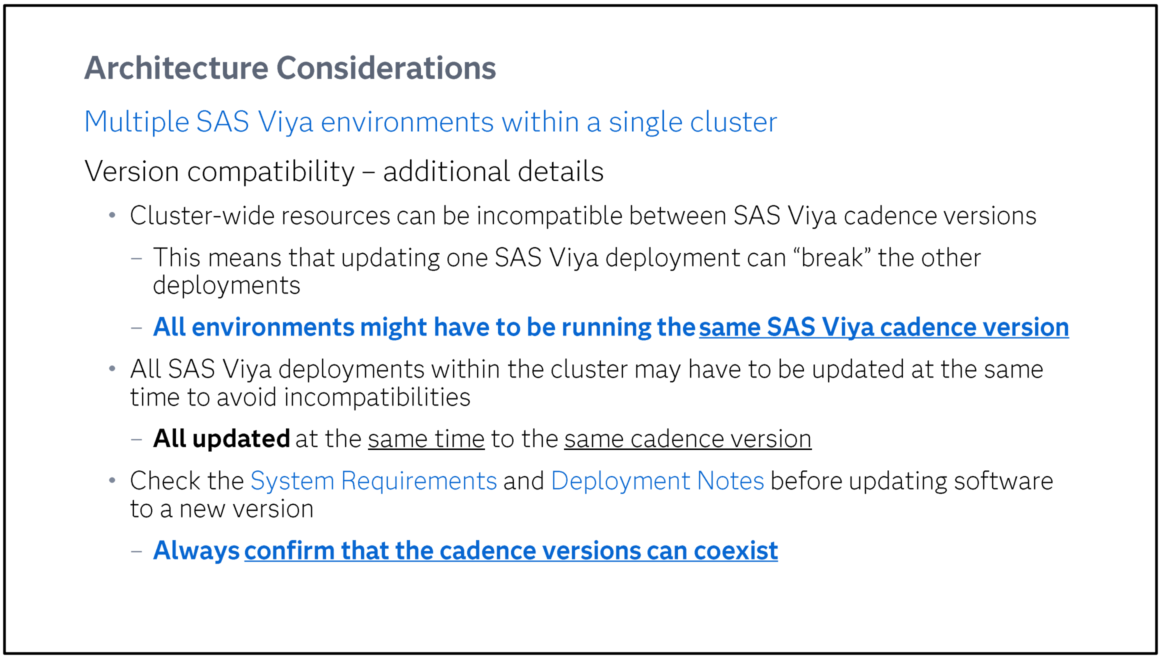
Looking at the architecture considerations. You need to consider the following.

Version compatibility. Different SAS Viya cadence versions can have different System Requirements.

The supported versions of third-party software and SAS resources could be incompatible between different cadence versions.

Co-locating Stable and Long-Term Support cadence versions within the same cluster increases the risk of incompatibility. This is due to differing system requirements between the Stable and LTS cadence versions. For example, supported versions of the ingress controller, or Postgres when using an external instance of the SAS Infrastructure Data Server.

Therefore, it is not recommended to co-locate Stable and Long-Term Support cadence versions within the same cluster.



Architecture Considerations

Multiple SAS Viya environments within a single cluster

Version compatibility – additional details

- Cluster-wide resources can be incompatible between SAS Viya cadence versions
  - This means that updating one SAS Viya deployment can “break” the other deployments
  - **All environments might have to be running the same SAS Viya cadence version**
- All SAS Viya deployments within the cluster may have to be updated at the same time to avoid incompatibilities
  - **All updated** at the same time to the same cadence version
- Check the System Requirements and Deployment Notes before updating software to a new version
  - **Always confirm that the cadence versions can coexist**

Here are some additional details on potential version incompatibilities.

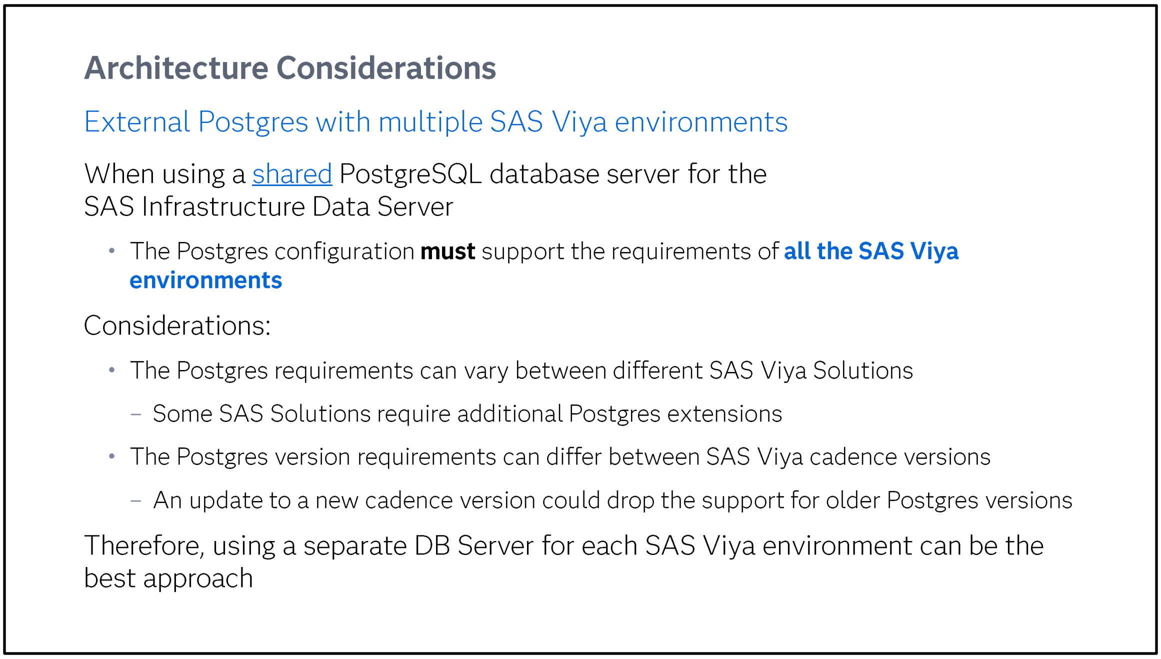
Cluster-wide resources can be incompatible between SAS Viya cadence versions. This means that updating one SAS Viya deployment can, or could, “break” the other deployments.

Therefore, all environments might have to be running the same SAS Viya cadence version.

To avoid incompatibilities, all the SAS Viya deployments within the cluster may have to be updated at the same time. And all updated at the same time to the same cadence version.

It is important to always check the System Requirements and Deployment Notes before updating the SAS Viya software to a new version.

You must always confirm that the different cadence versions can coexist.



Architecture Considerations

External Postgres with multiple SAS Viya environments

When using a shared PostgreSQL database server for the SAS Infrastructure Data Server

- The Postgres configuration **must** support the requirements of **all the SAS Viya environments**

Considerations:

- The Postgres requirements can vary between different SAS Viya Solutions
  - Some SAS Solutions require additional Postgres extensions
- The Postgres version requirements can differ between SAS Viya cadence versions
  - An update to a new cadence version could drop the support for older Postgres versions

Therefore, using a separate DB Server for each SAS Viya environment can be the best approach

Now let’s look at the considerations when using external Postgres with multiple SAS Viya platforms. The considerations for sharing the Postgres database server with multiple SAS Viya deployments.

This is when using a shared Postgres database server for the SAS Infrastructure Data server, and the SAS Common Data Store, the C-D-S Postgres.

The Postgres configuration must support the requirements of all the SAS Viya environments.

The considerations.

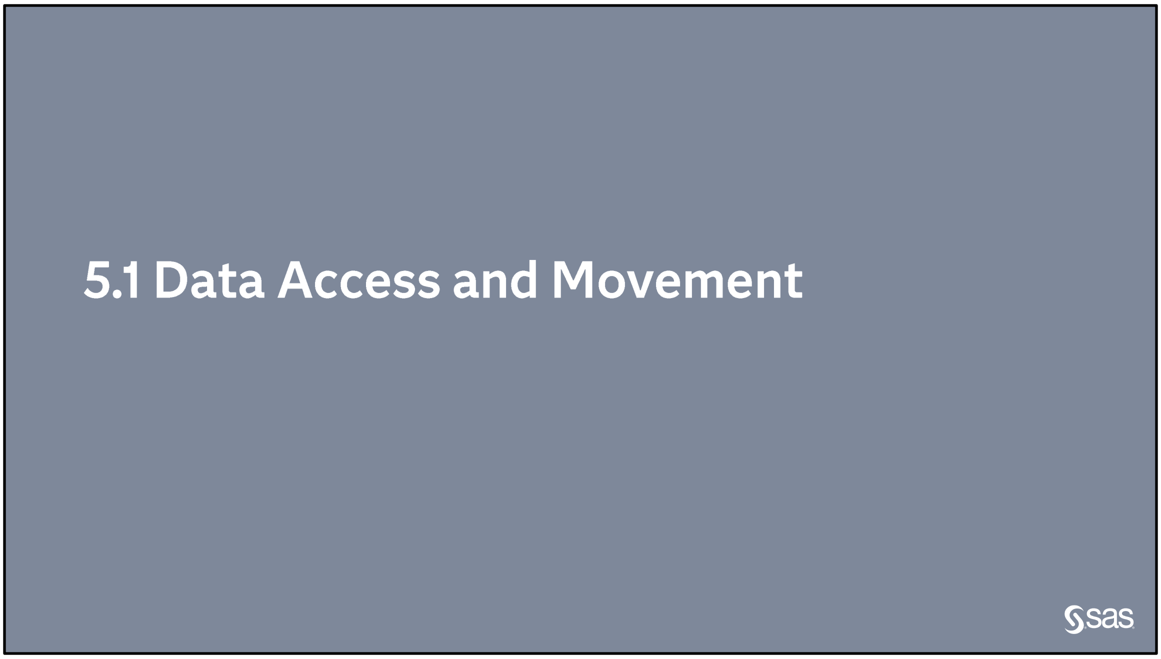
The Postgres requirements can vary between different SAS Viya Solutions. For example, some SAS Solutions require additional Postgres extensions.

The Postgres version requirements can differ between SAS Viya cadence versions. An update to a new cadence version could drop the support for older Postgres versions. Therefore, it is important to confirm the Postgres requirements for the cadence versions, and or the solutions being deployed.

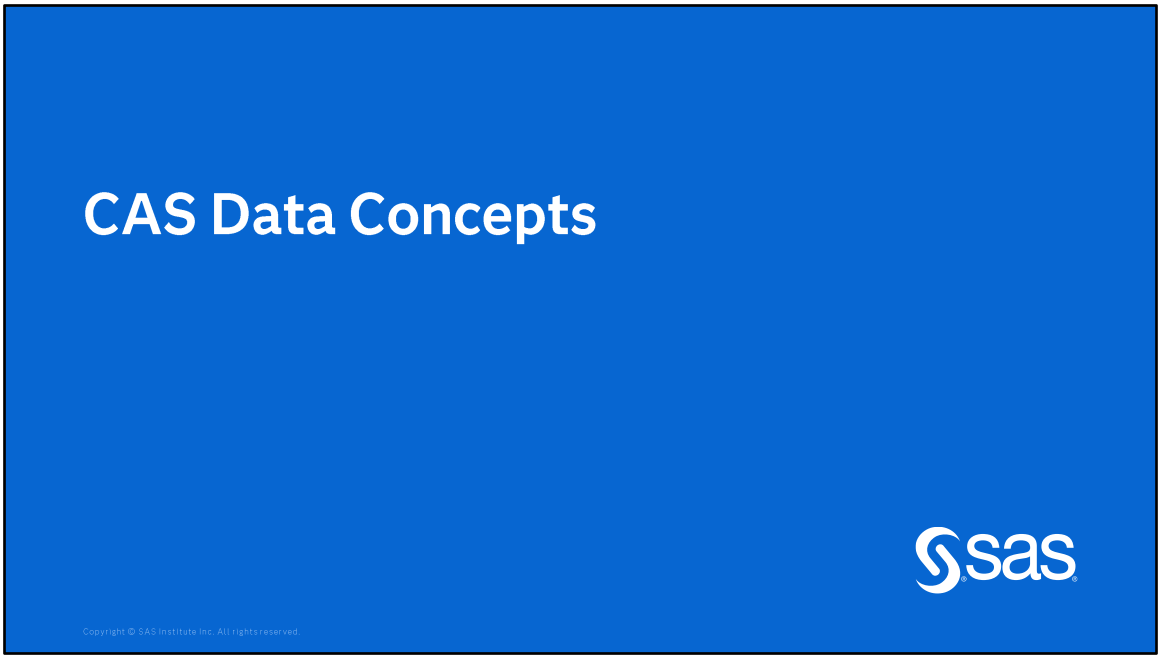
Given the risk of potential incompatibilities, using a separate database server for each SAS Viya environment can be the best approach.

That concludes our review of the architecture considerations for running multiple SAS Viya deployments in a shared Kubernetes cluster.

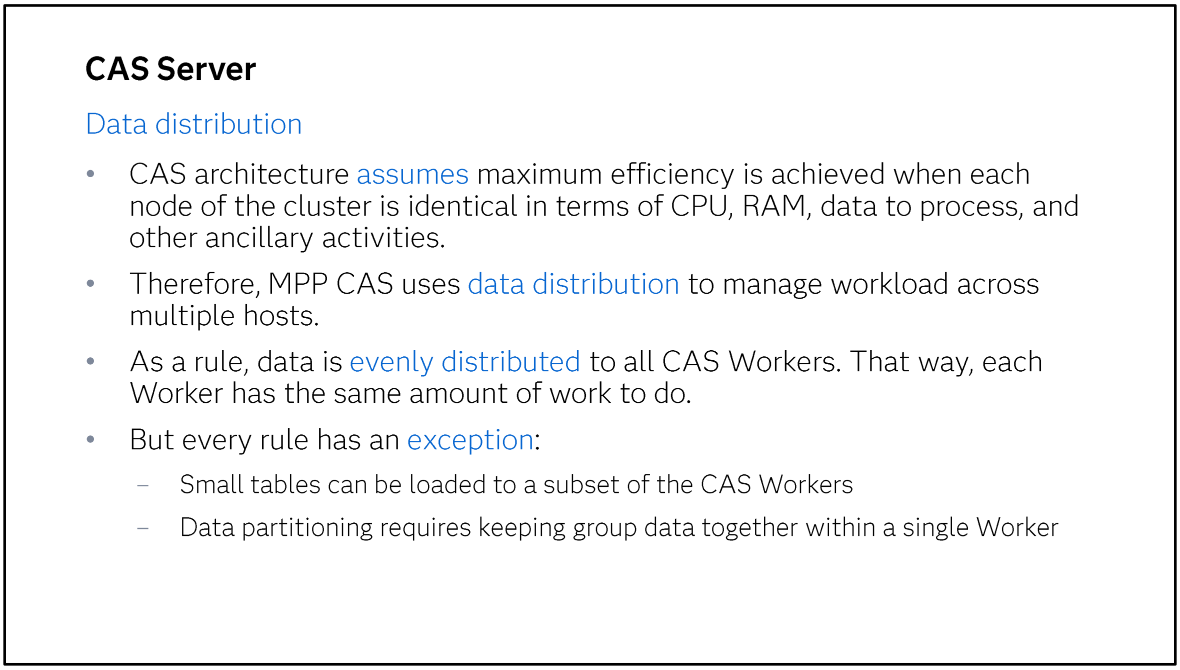
Lesson 5: Storage and Networking



5.1 Data Access and Movement



CAS Data Concepts



CAS Server

Data distribution

- CAS architecture assumes maximum efficiency is achieved when each node of the cluster is identical in terms of CPU, RAM, data to process, and other ancillary activities.
- Therefore, MPP CAS uses data distribution to manage workload across multiple hosts.
- As a rule, data is evenly distributed to all CAS Workers. That way, each Worker has the same amount of work to do.
- But every rule has an exception:
  - Small tables can be loaded to a subset of the CAS Workers
  - Data partitioning requires keeping group data together within a single Worker

The architecture of SAS Cloud Analytic Services – or CAS – assumes maximum efficiency is achieved when each node of the CAS server cluster is identical in terms of CPU, RAM, data to process, and other ancillary activities.

Therefore, MPP CAS or multi-machine CAS uses data distribution to manage workload across multiple hosts.

As a rule, data is evenly distributed to all CAS workers. That way, each worker has the same amount of work to do, and if they have identical CPU, RAM, etc., then they have the same amount of resources with which to do that work.

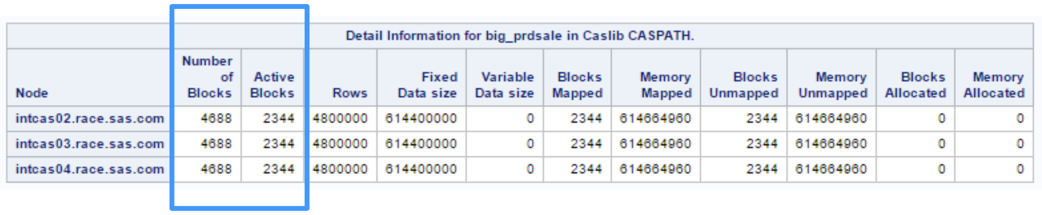
Of course, every rule has an exception. In this case, small tables can be loaded to a subset of CAS workers if you don't want it distributed across all of them. Also, data partitioning requires keeping grouped data together within a single worker.

And so, what that means is if you choose to partition your data, each partition has to live wholly on a single CAS worker, and it's the number of partitions that are distributed evenly. If your data cardinality is such that you have some partitions that are very large and others that are very small, that effectively means that some CAS workers have more work to do than others, so choose your partitioning criteria carefully.

CAS Server

Failure Resilient

- CAS is built to be resilient to failure of a Worker
- By default, CAS maintains 1 original + 1 replicate copy in the event of a node failure (COPIES=1 means data ×2)
- The inactive data blocks will become active if a CAS node goes down (as per their distribution)



Detail Information for big_prdsale in Caslib CASPATH.

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
intcas02.race.sas.com	4688	2344	4800000	614400000	0	2344	614664960	2344	614664960	0	0
intcas03.race.sas.com	4688	2344	4800000	614400000	0	2344	614664960	2344	614664960	0	0
intcas04.race.sas.com	4688	2344	4800000	614400000	0	2344	614664960	2344	614664960	0	0

CAS is also built to be resilient to the failure of a worker.

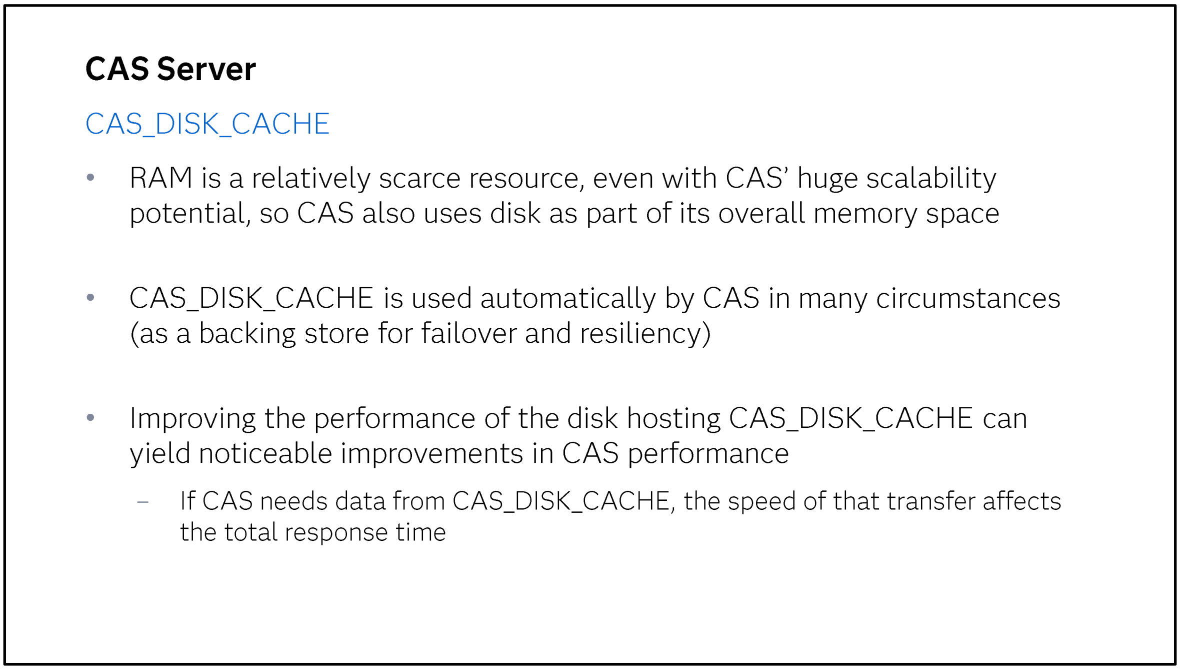
By default, CAS maintains one original plus one replicate copy of the data in event of a node failure. The COPIES data set option specifies the number of redundant copies of blocks stored across the CAS server. And so, if you say COPIES equals one, that means one more copy in addition to the existing data. In other words, copies equals one means twice the data is being stored in your CAS environment.

The inactive blocks, the extra copy, will become active if a CAS node goes down as per their distribution.

The table shown here breaks down the data tracked as blocks across the worker nodes of an MPP CAS server. The first column counts the total number of blocks and the second column counts the number of those that are active, or in-use.

Here we can see three CAS workers (or nodes) appear to have a pretty even data distribution with almost 4,700 blocks per node. And on each of those workers, half the blocks are active.

This aligns with the COPIES equals 1 concept, where each CAS worker is assigned two sets of blocks, one that's actively in-use and a second set that's inactive, ready to be called upon if one of the CAS workers goes offline.



CAS Server

CAS_DISK_CACHE

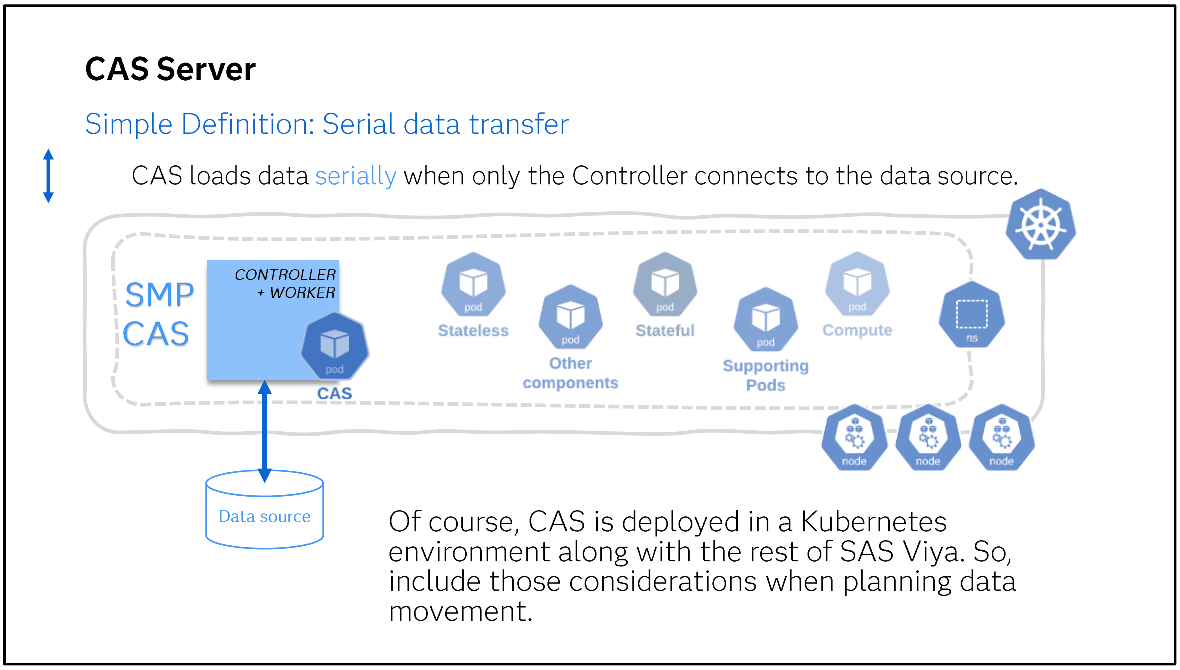
- RAM is a relatively scarce resource, even with CAS' huge scalability potential, so CAS also uses disk as part of its overall memory space
- CAS_DISK_CACHE is used automatically by CAS in many circumstances (as a backing store for failover and resiliency)
- Improving the performance of the disk hosting CAS_DISK_CACHE can yield noticeable improvements in CAS performance
  - If CAS needs data from CAS_DISK_CACHE, the speed of that transfer affects the total response time

RAM is a relatively scarce resource, even with CAS's huge scalability potential. So, CAS uses its disk as part of the overall memory space.

CAS disk cache is then used automatically by CAS in many circumstances as a backing store for failover and resiliency.

Improving the performance of the disk hosting CAS disk cache can yield noticeable improvements in CAS performance.

The idea is that if CAS must often rely on its disk cache to relieve pressure on RAM – like frequently swapping active tables to/from RAM – then the I/O performance of the CAS cache might become an important aspect to consider.



CAS Server

Simple Definition: Serial data transfer

CAS loads data serially when only the Controller connects to the data source.

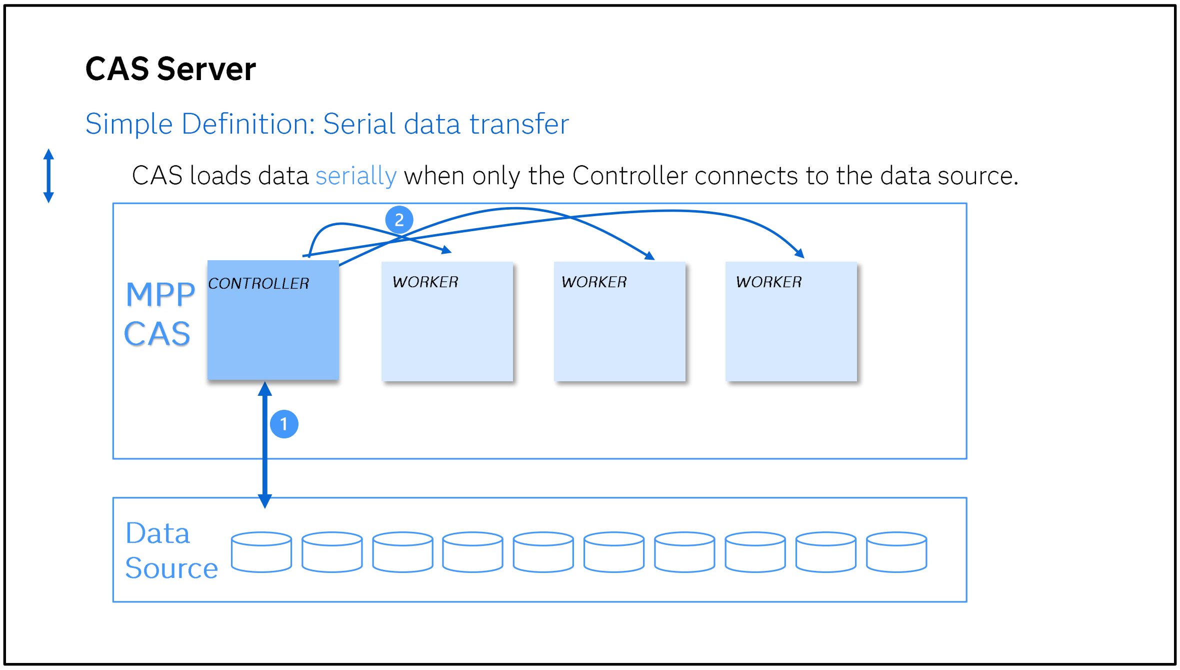
Of course, CAS is deployed in a Kubernetes environment along with the rest of SAS Viya. So, include those considerations when planning data movement.

Let's talk about some of the basic definitions for movement of data with SAS Viya, starting with serial data transfer.

First of all, this simplified illustration shows CAS as part of a SAS Viya deployment in Kubernetes. For this discussion of data movement, the Kubernetes components don't play a role, but we show them here to acknowledge the environment.

CAS, as shown here, is deployed in SMP mode as a single server. Internally, SMP CAS acts as both a CAS controller role and CAS worker role.

Because CAS is just a single node, the data transfer utilizes a single I/O channel which is, of course, serial by definition. That's not very interesting though because it's exactly how you would expect to characterize data transfer in this case.



CAS Server

Simple Definition: Serial data transfer

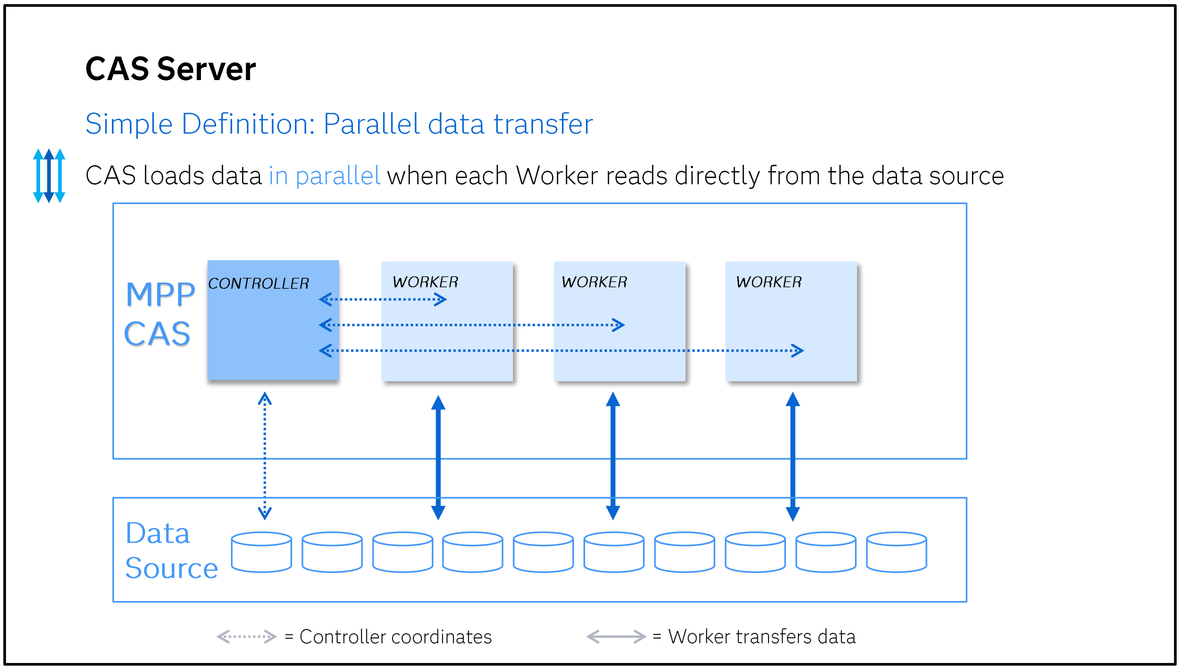
CAS loads data serially when only the Controller connects to the data source.

But we have another definition of serial transfer that gets a lot more interesting when we talk about working with MPP CAS or multi-machine CAS. And here is an illustration of CAS with a controller and multiple workers. The rest of the Kubernetes environment isn't shown in this illustration, but rest assured it's still there. We're just focused on the data transfer at this point.

When it comes to serial data transfer in this case, the CAS controller communicates with the data source to bring the data across a single I/O channel.

And here's the fun part: the CAS controller then distributes the data that it has loaded from the source evenly across all its workers.

Keep this picture in mind for MPP CAS when we talk about serial data transfer going forward.



CAS Server

Simple Definition: Parallel data transfer

CAS loads data in parallel when each Worker reads directly from the data source

= Controller coordinates

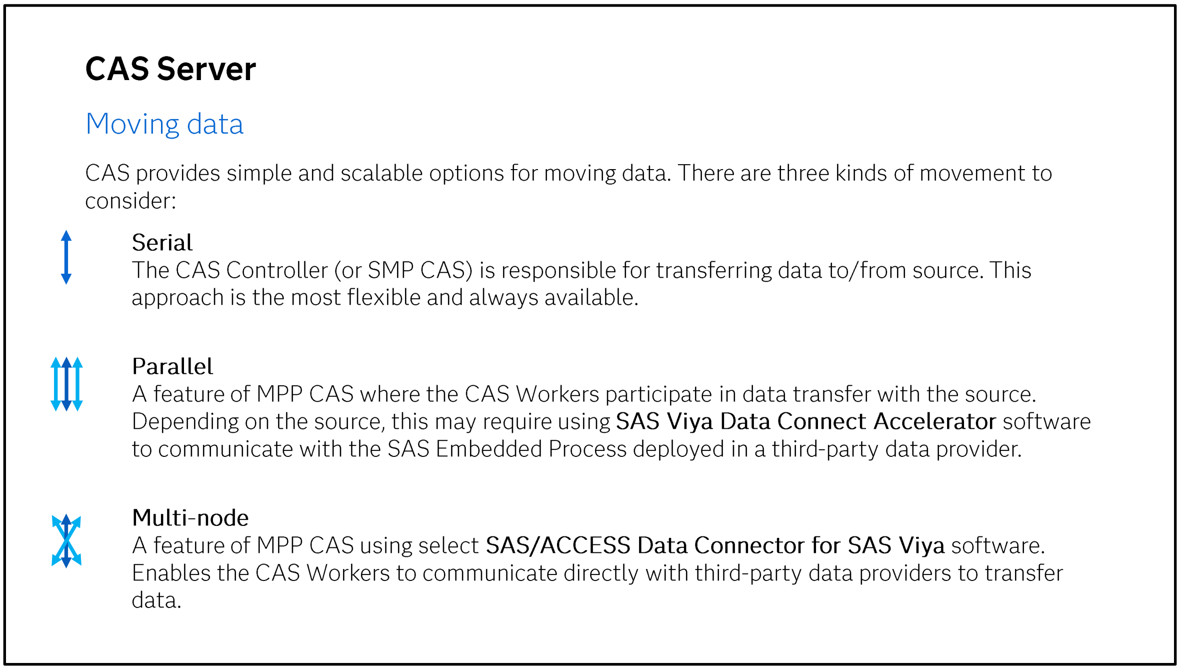
= Worker transfers data

CAS also offers a powerful feature we call parallel data transfer.

Parallel data movement occurs when each worker reads directly from the data source itself. As illustrated here, we can see multiple I/O channels employed which loads data into CAS quickly, assuming the infrastructure can support that type of movement.

The CAS controller coordinates this activity. It will communicate with the data source to get an idea of the size and the shape of the data. And then the controller directs the workers as to which bits of the data they each need to bring across.

The result is that each worker only brings over the data it needs, and it doesn't try to load any data that the controller has destined for other hosts.



CAS Server

Moving data

CAS provides simple and scalable options for moving data. There are three kinds of movement to consider:

Serial
The CAS Controller (or SMP CAS) is responsible for transferring data to/from source. This approach is the most flexible and always available.

Parallel
A feature of MPP CAS where the CAS Workers participate in data transfer with the source. Depending on the source, this may require using **SAS Viya Data Connect Accelerator** software to communicate with the SAS Embedded Process deployed in a third-party data provider.

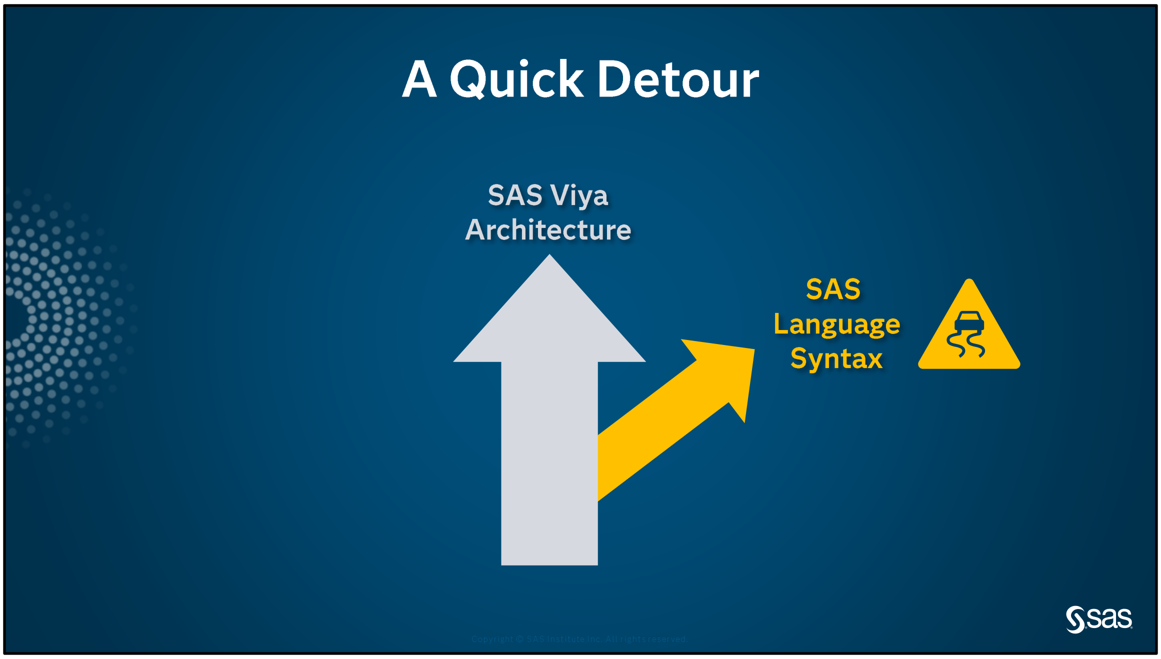
Multi-node
A feature of MPP CAS using select **SAS/ACCESS Data Connector for SAS Viya** software. Enables the CAS Workers to communicate directly with third-party data providers to transfer data.

CAS provides simple and scalable options for moving data. And there's three kinds of movement for us to consider.

Now, we've already talked about serial, which is responsible for data transfer to or from a source. And typically, this approach is very flexible and almost always available in most cases. So, if we can reach the data source and talk to it, chances are serial is going to work for us in some fashion.

Parallel movement is a feature of MPP CAS, where the CAS workers participate in data transfer with the source. Depending on that source, this may require using SAS Viya Data Connect Accelerator software to communicate with the SAS Embedded Process that's been deployed to the third-party data provider. We'll talk a little bit more about that coming up. The goal of parallel data transfer is that each of the workers has its own connection to the source to lift data from disk to RAM much faster than a single serial connection.

We also have this concept of multi-node data transfer. This is a feature of MPP CAS that is provided by the SAS Data Connector technology for SAS Viya software. The data connector enables CAS to communicate directly with third-party data providers to transfer the data. At its best, multi-node transfer is effectively parallel, but it's also possible that there might exist limitations in concurrent I/O between CAS and the data source. The implementation of multi-node by CAS means it can accommodate those limitations and complete the data movement.

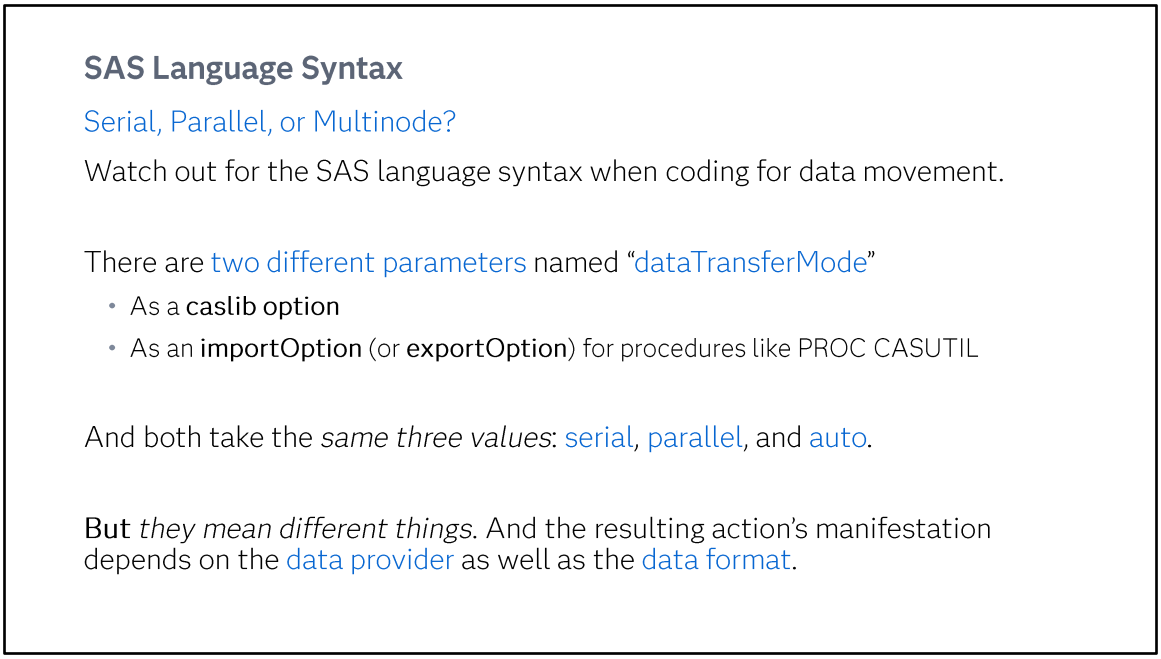


A Quick Detour

SAS Viya Architecture

SAS Language Syntax

Now, at this point, let’s take a real quick detour. We've been talking about the architecture of data movement and stuff, but now we need to spend just a couple minutes talking about SAS language syntax and it’s a little confusing at first glance.



SAS Language Syntax

Serial, Parallel, or Multinode?

Watch out for the SAS language syntax when coding for data movement.

There are two different parameters named “dataTransferMode”

- As a **caslib** option
- As an **importOption** (or **exportOption**) for procedures like PROC CASUTIL

And both take the *same three values*: serial, parallel, and auto.

But *they mean different things*. And the resulting action’s manifestation depends on the data provider as well as the data format.

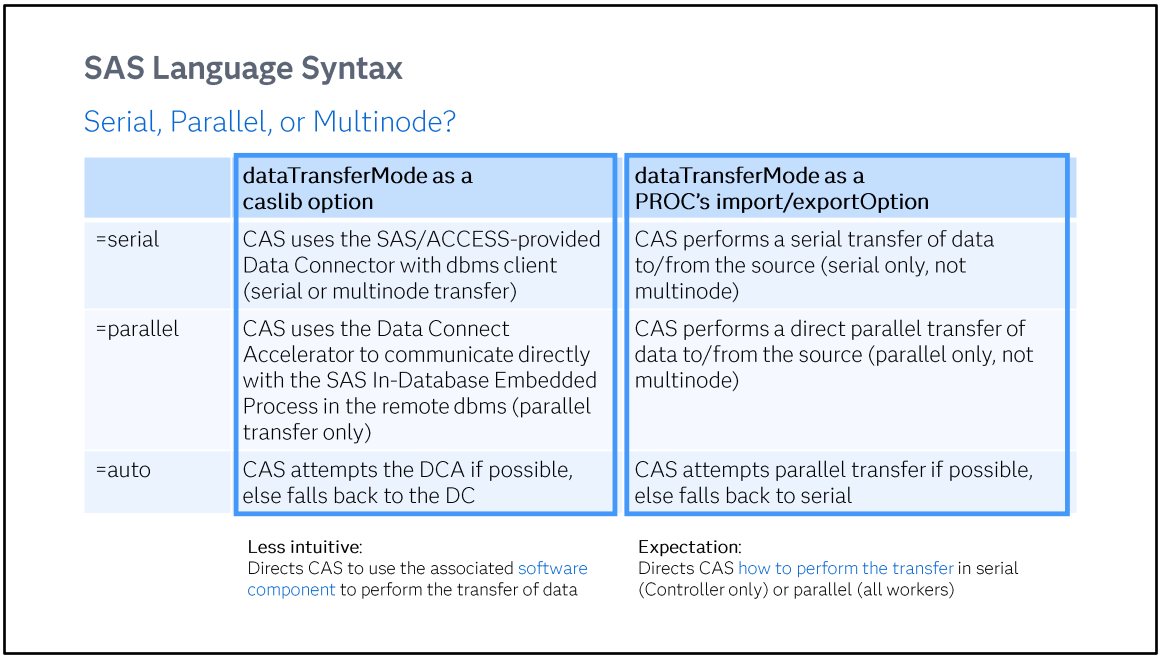
We need to watch out for the SAS language syntax when we're coding for data movement.

As it turns out, there are two different parameters, and they are both named “data-transfer-mode”.

Where you specify the data-transfer-mode option matters as that determines which functionality you’ll get. One place it can be specified is as an option on the caslib statement. The other place is an import-option (or export-option) for SAS procedures like PROC CASUTIL.

Not only are there two different options with the same name, but they both accept the same three values as well: serial, parallel, and auto. Again, the values specified for these options mean different things based on where they’re used in your SAS code. The resulting action’s manifestation depends on the data provider as well as the data format.

They’re not interchangeable either. There are situations where specifying the data-transfer-mode is only appropriate for the caslib or for the importOption. Take care to reference the correct parameter, with the desired value, in the proper place.



SAS Language Syntax

Serial, Parallel, or Multinode?

	dataTransferMode as a caslib option	dataTransferMode as a PROC's import/exportOption
=serial	CAS uses the SAS/ACCESS-provided Data Connector with dbms client (serial or multinode transfer)	CAS performs a serial transfer of data to/from the source (serial only, not multinode)
=parallel	CAS uses the Data Connect Accelerator to communicate directly with the SAS In-Database Embedded Process in the remote dbms (parallel transfer only)	CAS performs a direct parallel transfer of data to/from the source (parallel only, not multinode)
=auto	CAS attempts the DCA if possible, else falls back to the DC	CAS attempts parallel transfer if possible, else falls back to serial

Less intuitive:
Directs CAS to use the associated software component to perform the transfer of data

Expectation:
Directs CAS how to perform the transfer in serial (Controller only) or parallel (all workers)

Let's take a closer look at the two kinds of data-transfer-mode parameters, where they're used, and what their associated values direct CAS to do. Keep in mind that we're mostly contemplating multi-machine or MPP CAS. For single-machine or SMP CAS, only serial data movement is possible, and these SAS programming options don't apply.

First, we'll look at the data-transfer-mode when specified as an import- or export-option for a SAS procedure like PROC CASUTIL.

The first value is "serial". In this case, we'll get the classic serial movement of data as illustrated earlier where the CAS controller will load the data by itself from the source and distribute it across the workers. We note here that multi-node transfer is not included which might seem weird to call out but will make sense in just a bit.

Next up is the value of "parallel". This tells CAS to attempt parallel movement of the data from the source utilizing all its workers such that it makes its own connection to the source and, as coordinated by the controller, transfers its allotment of data directly itself. Parallel in this case means that all CAS workers must participate. Multi-node, again, is not included.

And then if we set the value to "auto", meaning "automatic", then CAS will attempt parallel transfer if possible. But if that's not possible for some reason, that is, something prevents one or more of the CAS workers from participating, then it will fall back to serial.

Taking a moment to reflect at this point, the values of "serial" and "parallel" pretty much mean what we expect, right? When data-transfer-mode is specified as an import- or export-option, then it tells CAS to move the data from source using either the controller alone for serial movement, or all of the workers for parallel.

Now let’s look at something different.

Looking at data-transfer-mode as an option for the caslib statement produces similar outcomes, but through a very different interpretation and approach.

When “serial” is specified, that directs CAS to use its Data Connector software (often in coordination with something like a DBMS client). Back when the Data Connector technology was first introduced for CAS, it was only capable of serial data movement. In the years since, it has evolved additional capabilities beyond serial-only transfer. But for backward compatibility, this value of “serial” is still used today for the data-transfer-mode option on the caslib statement to activate the desired Data Connector.

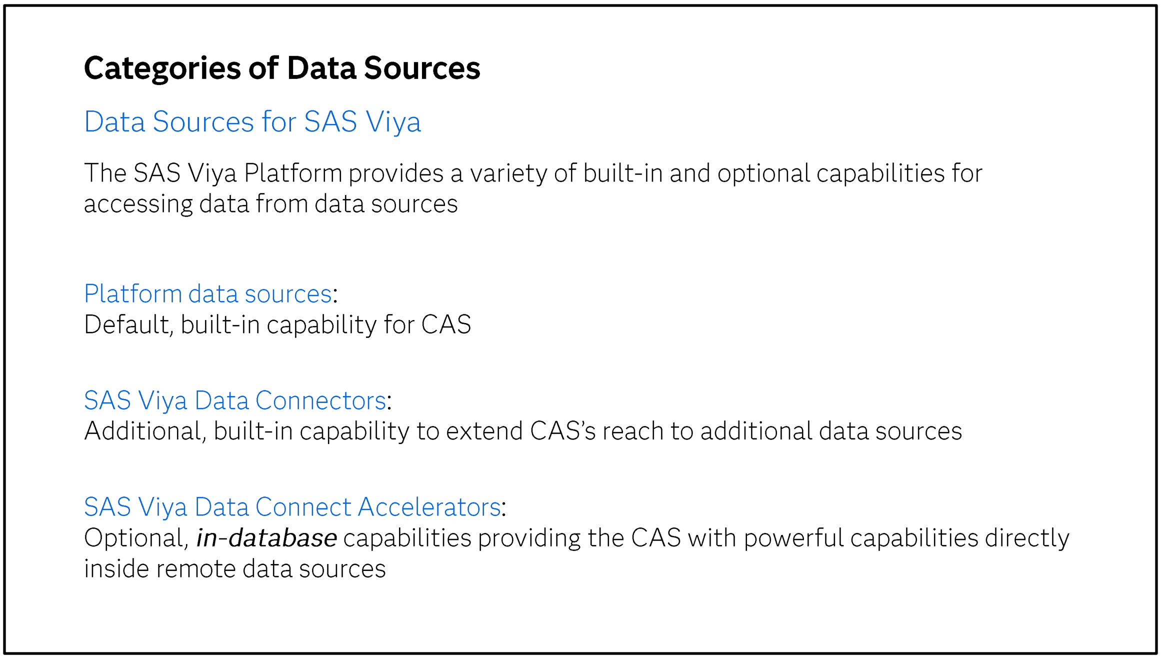
Speaking of additional capabilities, the “serial” value accommodates more than just serial data transfer. It is also used to activate multi-node transfers involving two or more workers of a CAS server. That’s because multi-node functionality is provided by the SAS Data Connector technology.

Next let’s look at “parallel”. And as before, this directs CAS to utilize a specific software, in this case, the Data Connect Accelerator. The Data Connect Accelerator technology enables CAS to talk to the SAS In-Database Embedded Process in a supported data provider. When this happens, CAS workers are coordinated by the controller to talk to the EP directly such that all the CAS workers participate in parallel transfer of the data.

The caslib option also has an “auto” value so that CAS will automatically attempt to use the Data Connect Accelerator if possible. But if for some reason that doesn't work, like maybe there's not an Embedded Process to talk to on the remote side, then CAS will fall back to using its Data Connector software instead. And, as just described, the Data Connector software can perform either serial transfer or multi-node transfer.

Compared to data-transfer-mode for the import- and export-option of PROC CASUTIL, the caslib’s data-transfer-option is less intuitive in meaning because what it’s really doing is directing CAS to use the associated software component to perform the transfer of the data. The resulting data movement will depend on external factors and the ability of the systems to participate.

Data Sources



Categories of Data Sources

Data Sources for SAS Viya

The SAS Viya Platform provides a variety of built-in and optional capabilities for accessing data from data sources

Platform data sources:
Default, built-in capability for CAS

SAS Viya Data Connectors:
Additional, built-in capability to extend CAS's reach to additional data sources

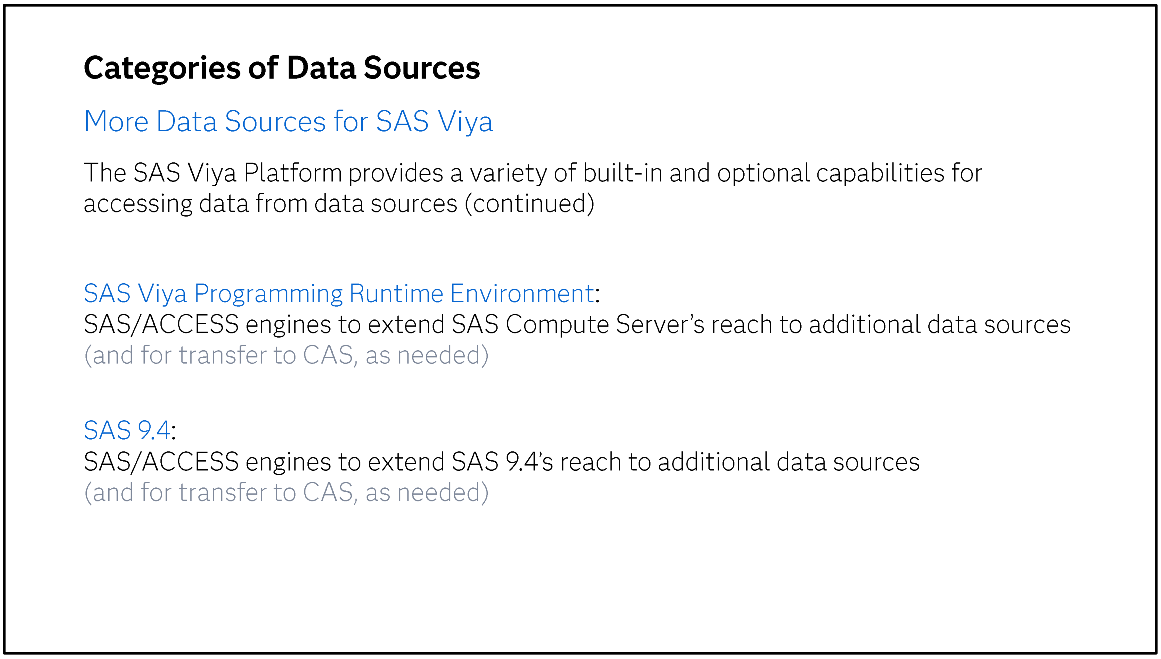
SAS Viya Data Connect Accelerators:
Optional, *in-database* capabilities providing the CAS with powerful capabilities directly inside remote data sources

The SAS Viya platform provides a variety of built-in and optional capabilities for accessing data from various data sources. We're going to break those down here into major groupings.

The first are the the platform data sources. These include the locations and formats which SAS Cloud Analytic Services has the built-in capability to work with natively and directly.

Next, we have the SAS Viya Data Connectors. These extend CAS's reach to work additional data sources. Nearly all data connectors are included with SAS Viya offerings by default.

We also have the SAS Viya Data Connect Accelerators. While the name seems similar to Data Connectors, these are targeted at providing a very specific kind of functionality. They allow CAS to communicate with SAS In-Database technology that's deployed to a select set of remote data sources.



Categories of Data Sources

More Data Sources for SAS Viya

The SAS Viya Platform provides a variety of built-in and optional capabilities for accessing data from data sources (continued)

SAS Viya Programming Runtime Environment:
SAS/ACCESS engines to extend SAS Compute Server's reach to additional data sources
(and for transfer to CAS, as needed)

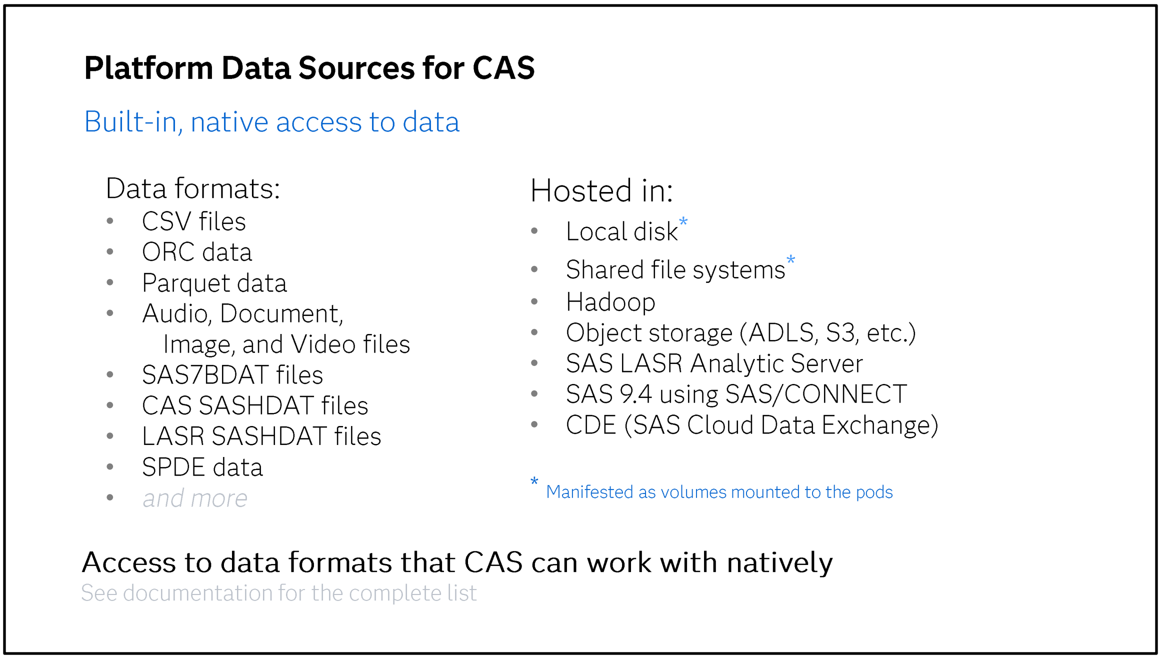
SAS 9.4:
SAS/ACCESS engines to extend SAS 9.4's reach to additional data sources
(and for transfer to CAS, as needed)

If the data you want isn't supported (yet) by a SAS Data Connector, then we can try another route using other SAS runtime engines.

With Viya, we have the SAS Programming Runtime Environment where SAS program code including DATA and PROC steps as well as Macro code is executed. This runtime is exemplified by the SAS Compute Server, SAS Batch Server, and SAS Connect Server in a Viya deployment. SAS/ACCESS products are used to extend the runtime's ability and provide the means to talk directly to a third-party data source (often using its associated DBMS client).

Indeed, if that's not sufficient and your site also has SAS 9.4, then that could also be used to get to external data if there's an appropriate SAS/ACCESS product there for it.

Either way, these SAS runtimes can communicate directly with CAS and so, can be used as a bridge to reach other data sources that CAS might not be able to interact with directly.



Platform Data Sources for CAS

Built-in, native access to data

Data formats:

- CSV files
- ORC data
- Parquet data
- Audio, Document, Image, and Video files
- SAS7BDAT files
- CAS SASHDAT files
- LASR SASHDAT files
- SPDE data
- *and more*

Hosted in:

- Local disk*
- Shared file systems*
- Hadoop
- Object storage (ADLS, S3, etc.)
- SAS LASR Analytic Server
- SAS 9.4 using SAS/CONNECT
- CDE (SAS Cloud Data Exchange)

* Manifested as volumes mounted to the pods

Access to data formats that CAS can work with natively

See documentation for the complete list

Let's look at some of the CAS platform data sources. This is data that CAS knows how to work with right out of the box.

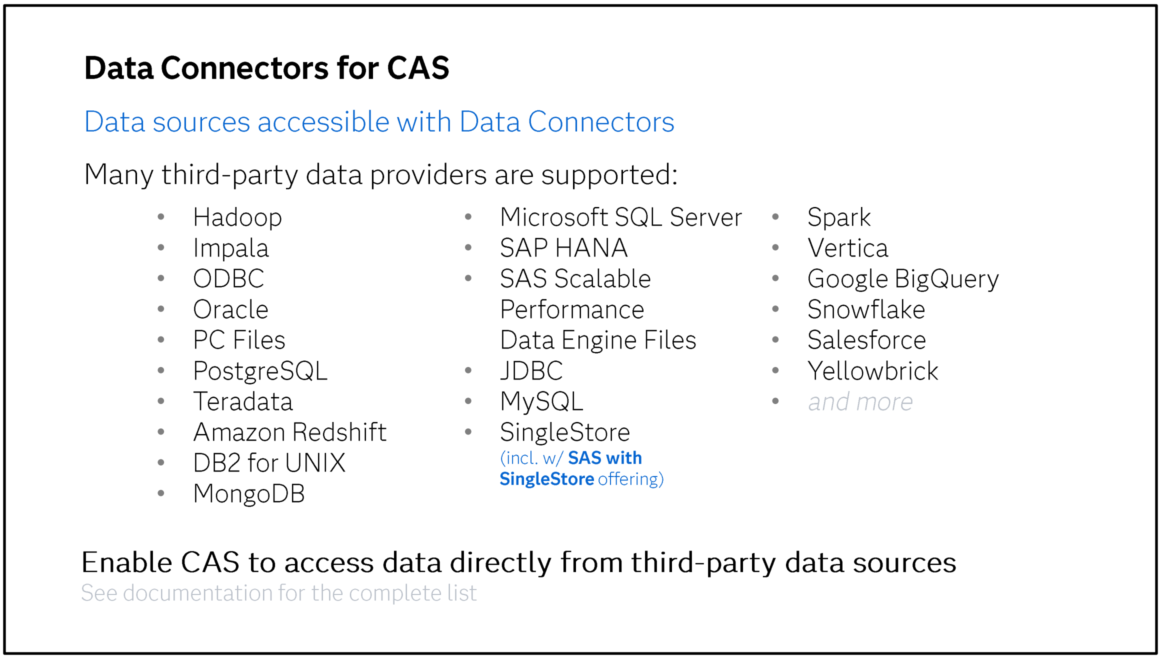
You'll see several standardized data types listed here, including ORC (optimized row columnar), CSV (comma-separated values), and Parquet (another columnar format). CAS also knows how to work with SAS proprietary formats like SAS7BDAT, SASHDAT, and SPDE (SAS Scalable Performance Data Engine) files. Additionally, CAS can process multimedia files for audio, image, and video as well as binary document formats like PDF and Excel workbooks.

Note this slide does not present an exhaustive list so refer to SAS documentation for additional details.

Where the data resides matters to CAS. Some formats are only accessible through certain channels or from supported media. If looking at local disk or other mounted storage like a shared file system, remember that locations must be manifested as volumes mounted to the appropriate pods (usually CAS controller and workers) for SAS Viya.

Beyond local storage, CAS can also interface with hosted systems and services, including Hadoop clusters, object storage like Amazon S3 or Azure Data Lake Storage. CAS also is backward compatible with older SAS technology and can receive data from the SAS LASR Analytic Server as well as through SAS/CONNECT and library services from SAS 9.4.

More recently, SAS has added the SAS Cloud Data Exchange to provide a standardized approach for cloud-based deployments of Viya so that CAS to work with data from on-premise sources securely with efficient, high-volume data transfer. We'll look more closely at that in another module.



Data Connectors for CAS

Data sources accessible with Data Connectors

Many third-party data providers are supported:

- Hadoop
- Impala
- ODBC
- Oracle
- PC Files
- PostgreSQL
- Teradata
- Amazon Redshift
- DB2 for UNIX
- MongoDB
- Microsoft SQL Server
- SAP HANA
- SAS Scalable Performance Data Engine Files
- JDBC
- MySQL
- SingleStore (incl. w/ **SAS with SingleStore** offering)
- Spark
- Vertica
- Google BigQuery
- Snowflake
- Salesforce
- Yellowbrick
- *and more*

Enable CAS to access data directly from third-party data sources

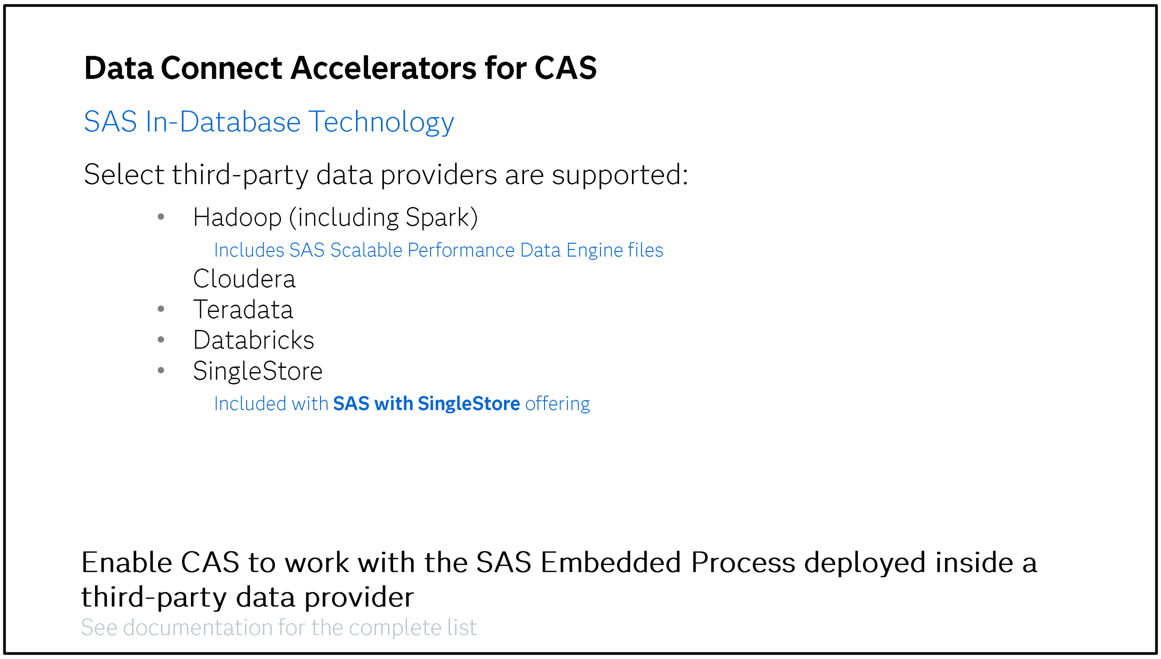
See documentation for the complete list

SAS Data Connectors extend CAS to give it the ability to communicate with third-party providers efficiently. Most SAS Data Connectors are included with the Viya offerings by default.

While the list of supported third-party sources shown on this slide is long, it does not present an exhaustive inventory so refer to SAS documentation for additional details. Still, let’s call out a few notable ones.

Using the appropriate data connectors, CAS can work with data from RDBMS sources like Oracle, PostgreSQL, MySQL, and more. Also, almost every data source supports some level of OBDC or JDBC access. So, if a native data connector for CAS isn’t available to a specific data provider, chances are the data can still be reached through ODBC or JDBC. Depending on where they reside, CAS can also use data connectors to reach files in supported PC file formats, SPDE data, and so on.

SAS currently engages in a strong partnership with SingleStore. Indeed, we even have an offering known as SAS with SingleStore where a SingleStore memSQL database is included and deployed with SAS Viya in Kubernetes. This partnership is significantly increasing Viya’s integration and ability to work with SingleStore natively in new and powerful ways.



Data Connect Accelerators for CAS

SAS In-Database Technology

Select third-party data providers are supported:

- Hadoop (including Spark)
  - Includes SAS Scalable Performance Data Engine files

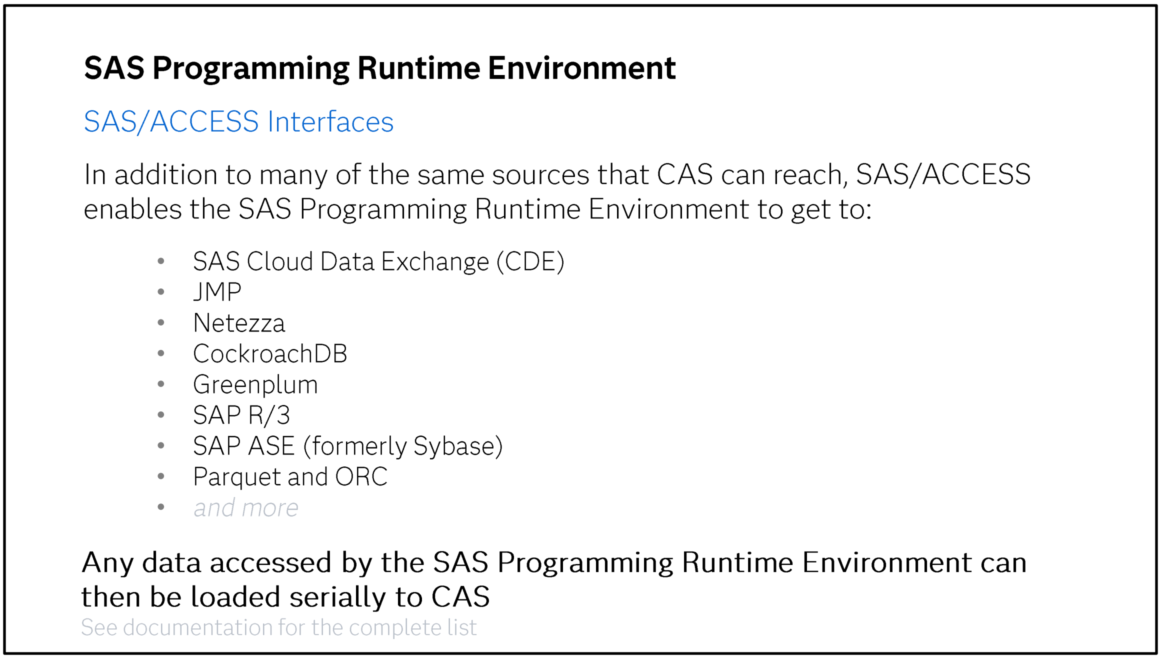
  Cloudera
- Teradata
- Databricks
- SingleStore
  - Included with **SAS with SingleStore** offering

Enable CAS to work with the SAS Embedded Process deployed inside a third-party data provider

See documentation for the complete list

SAS In-Database technology is an optional offering where a site can deploy the SAS Embedded Process into a supported data provider. For Viya, this list is not as long, but the technology is powerful. In-database processing allows SAS to run analytics against the data right where it lives in the source. The results as well as subset data can be streamed back to CAS for additional processing.

As we're discussing here, data movement from the Embedded Process to CAS is very efficient and employs multiple, parallel I/O channels to get the data across fast.



SAS Programming Runtime Environment

SAS/ACCESS Interfaces

In addition to many of the same sources that CAS can reach, SAS/ACCESS enables the SAS Programming Runtime Environment to get to:

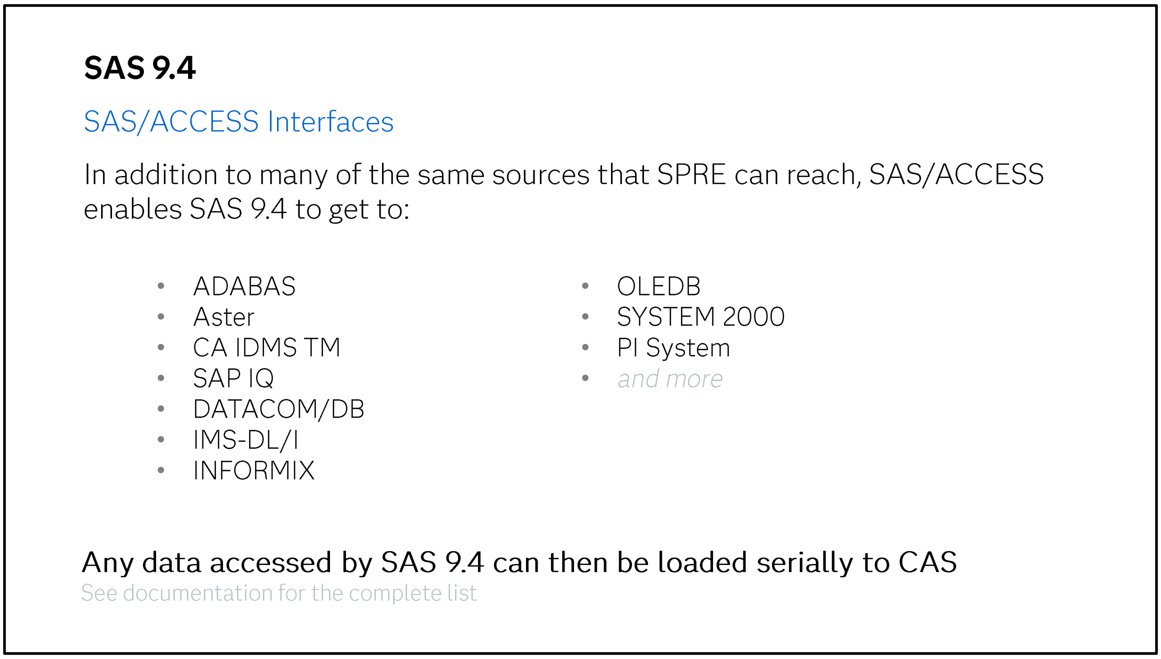
- SAS Cloud Data Exchange (CDE)
- JMP
- Netezza
- CockroachDB
- Greenplum
- SAP R/3
- SAP ASE (formerly Sybase)
- Parquet and ORC
- *and more*

Any data accessed by the SAS Programming Runtime Environment can then be loaded serially to CAS

See documentation for the complete list

Besides CAS in Viya, we also have the SAS Viya Programming Runtime Environment.

The list of SAS/ACCESS products available for the Programming Runtime Environment is much longer than the few shown here, but we're trying to point out that there are some available as SAS/ACCESS products which are not offered as data connectors for CAS. To that end, if your goal is to get data from one of these sources into CAS, then you can use the SAS Programming Runtime Environment as a bridge in the process for loading data to CAS.



SAS 9.4

SAS/ACCESS Interfaces

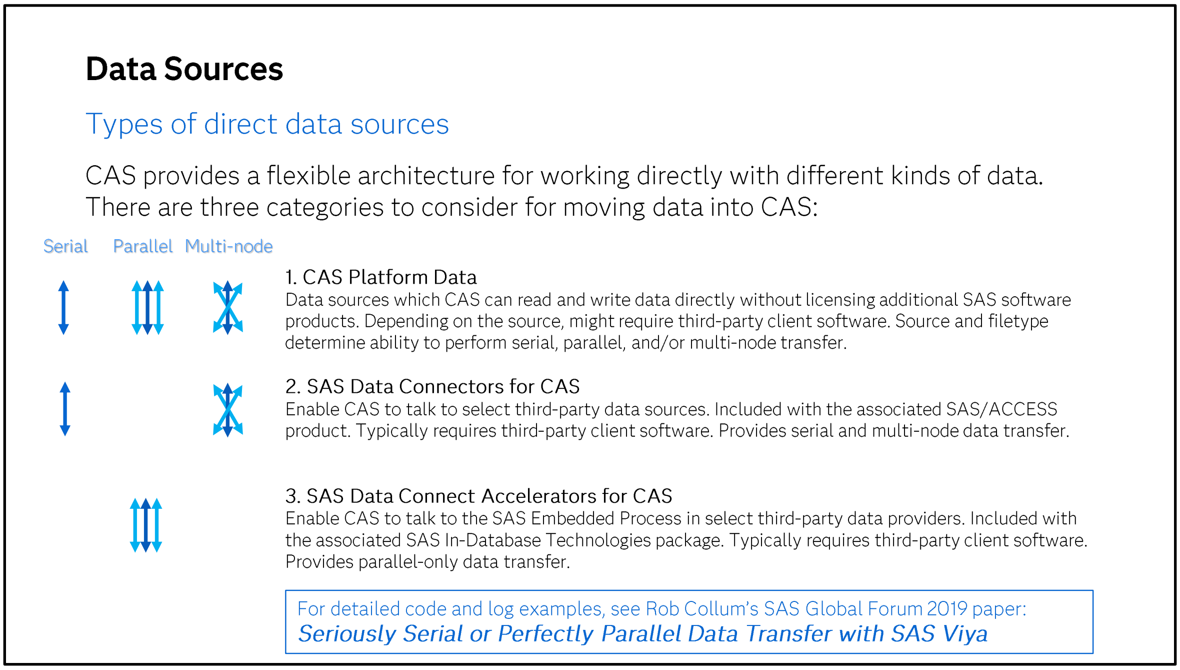
In addition to many of the same sources that SPRE can reach, SAS/ACCESS enables SAS 9.4 to get to:

- ADABAS
- Aster
- CA IDMS TM
- SAP IQ
- DATACOM/DB
- IMS-DL/I
- INFORMIX
- OLEDB
- SYSTEM 2000
- PI System
- *and more*

Any data accessed by SAS 9.4 can then be loaded serially to CAS

See documentation for the complete list

Many sites have been SAS customers for years and still rely on SAS 9.4 while also running with SAS Viya. And again, the list of SAS/ACCESS products for SAS 9.4 is much longer than those shown here, but again, SAS 9.4 can act as a bridge in the process for loading data over to CAS, if that’s your goal.



Data Sources

Types of direct data sources

CAS provides a flexible architecture for working directly with different kinds of data. There are three categories to consider for moving data into CAS:

Serial Parallel Multi-node

1. CAS Platform Data
Data sources which CAS can read and write data directly without licensing additional SAS software products. Depending on the source, might require third-party client software. Source and filetype determine ability to perform serial, parallel, and/or multi-node transfer.

2. SAS Data Connectors for CAS
Enable CAS to talk to select third-party data sources. Included with the associated SAS/ACCESS product. Typically requires third-party client software. Provides serial and multi-node data transfer.

3. SAS Data Connect Accelerators for CAS
Enable CAS to talk to the SAS Embedded Process in select third-party data providers. Included with the associated SAS In-Database Technologies package. Typically requires third-party client software. Provides parallel-only data transfer.

For detailed code and log examples, see Rob Collum's SAS Global Forum 2019 paper:
Seriously Serial or Perfectly Parallel Data Transfer with SAS Viya

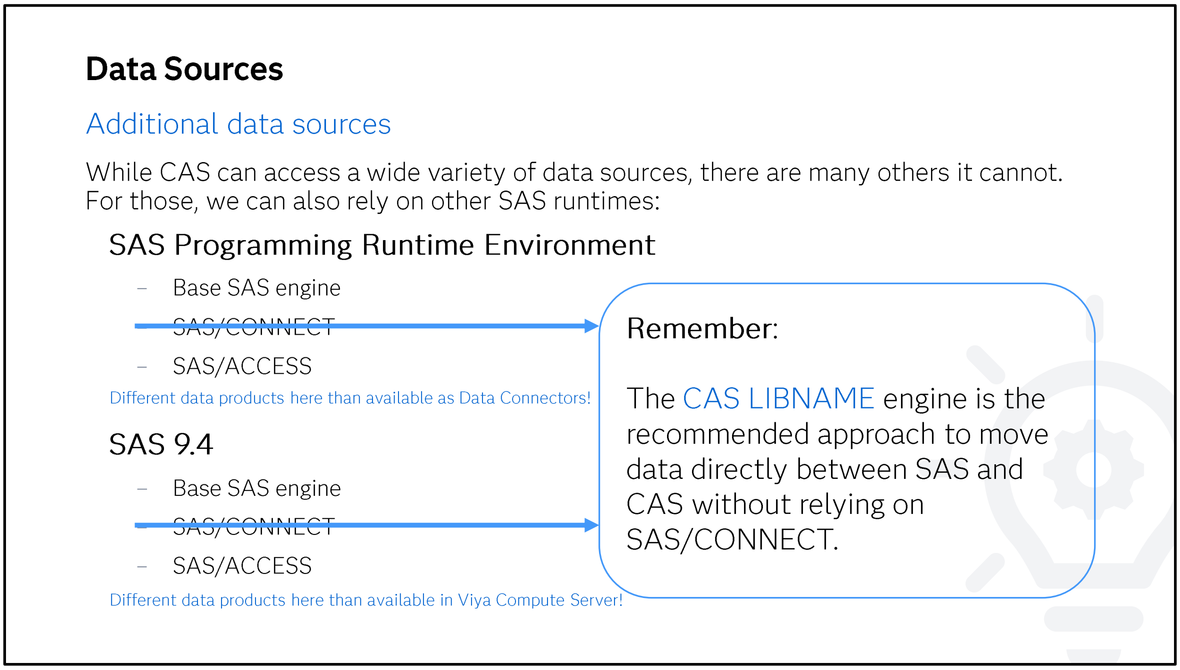
CAS provides a flexible architecture for working directly with different kinds of data. And there are three categories to consider when we're moving data into CAS.

First off, there's the CAS platform data. These re data sources which CAS can read and write directly and we don't need to license any additional SAS software. Depending on the source, it might require third-party client software to be installed and configured. The source and file type will determine CAS's ability to perform serial, parallel, and or multi-node transfer of that data.

The SAS Data Connectors enable CAS to work with third party-data sources. Most of them are included with the SAS Viya product offerings. They typically require some third-party client software. And they give the ability to perform serial movement of data into CAS, as well as multi-node data transfer where multiple CAS workers participate directly in loading data. Noticeably absent here is that the data connectors do not allow CAS to perform what's defined as parallel transfer. From a strict definition, parallel means that all the CAS workers must participate in the data transfer.

Third on the list are the SAS Data Connect Accelerators. These enable CAS to communicate directly with the SAS In-Database Embedded Process in select third-party data providers. Data movement using the data connect accelerator is only parallel – all CAS workers must participate.

For detailed code and log examples when working with serial, parallel, or multi-node, refer to the SAS Global Forum paper from 2019 titled "Seriously Serial or Perfectly Parallel Data Transfer with SAS Viya". This workshop highlights key points, but that paper provides a lot more detailed descriptions. And even though it was written before SAS Viya 4, the syntax, terminology, and most of the behavior has evolved only slightly so the paper itself is still pretty effective in conveying what you need to look out for.



Data Sources

Additional data sources

While CAS can access a wide variety of data sources, there are many others it cannot. For those, we can also rely on other SAS runtimes:

SAS Programming Runtime Environment

- Base SAS engine
- ~~SAS/CONNECT~~
- SAS/ACCESS

Different data products here than available as Data Connectors!

SAS 9.4

- Base SAS engine
- ~~SAS/CONNECT~~
- SAS/ACCESS

Different data products here than available in Viya Compute Server!

Remember:

The CAS LIBNAME engine is the recommended approach to move data directly between SAS and CAS without relying on SAS/CONNECT.

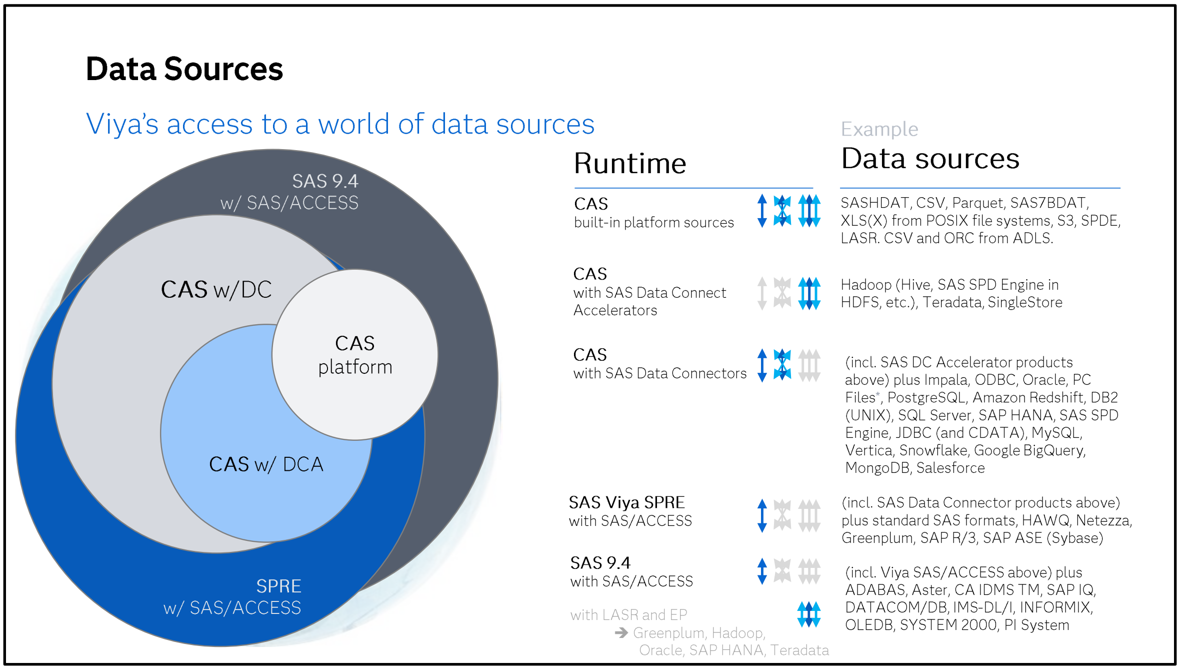
While CAS can access a wide variety of data sources, there are many other data sources that it can’t. And for those, we might rely on other SAS runtimes, like the SAS Programming Runtime Environment in Viya or SAS 9.4. Both have the Base SAS engine and SAS/CONNECT as well as a selection of SAS/ACCESS products.

Indeed, there are different SAS/ACCESS products for the SAS Programming Runtime Environment in Viya than offered for CAS.

Similarly, the same is true for SAS 9.4 compared to the SAS/ACCESS products offered for Viya.

If needed, we can use the SAS/CONNECT technology to transfer data from those SAS runtimes to CAS, but…

We recommend using the CAS type of libname instead. It provides the best and most efficient technique to move data from the SAS runtime over to CAS. The SAS/CONNECT approach still works and if you have legacy code still using it, that’s fine.



Data Sources

Viya's access to a world of data sources

Runtime	Example Data sources
CAS built-in platform sources	SASHDAT, CSV, Parquet, SAS7BDAT, XLS(X) from POSIX file systems, S3, SPDE, LASR. CSV and ORC from ADLS.
CAS with SAS Data Connect Accelerators	Hadoop (Hive, SAS SPD Engine in HDFS, etc.), Teradata, SingleStore
CAS with SAS Data Connectors	(incl. SAS DC Accelerator products above) plus Impala, ODBC, Oracle, PC Files*, PostgreSQL, Amazon Redshift, DB2 (UNIX), SQL Server, SAP HANA, SAS SPD Engine, JDBC (and CDATA), MySQL, Vertica, Snowflake, Google BigQuery, MongoDB, Salesforce
SAS Viya SPRE with SAS/ACCESS	(incl. SAS Data Connector products above) plus standard SAS formats, HAWQ, Netezza, Greenplum, SAP R/3, SAP ASE (Sybase)
SAS 9.4 with SAS/ACCESS	(incl. Viya SAS/ACCESS above) plus ADABAS, Aster, CA IDMS TM, SAP IQ, DATACOM/DB, IMS-DL/I, INFORMIX, OLEDB, SYSTEM 2000, PI System
with LASR and EP → Greenplum, Hadoop, Oracle, SAP HANA, Teradata	

To sum up and illustrate a bit, Viya has access to a world of data sources.

Starting with CAS and its built-in platform sources. They support serial, multi-node, and parallel movement of data depending on its format and its location. A representative list of those sources is shown on the right, but of course, refer to the SAS documentation for the full details.

We also have SAS Data Connect Accelerators for CAS. And that gives us a little bit broader scope. The data connect accelerators only support parallel movement of the data, because they require all CAS workers to talk to the SAS Embedded Process deployed as part of a SAS In-Database environment in a select provider like Hadoop or Teradata or SingleStore.

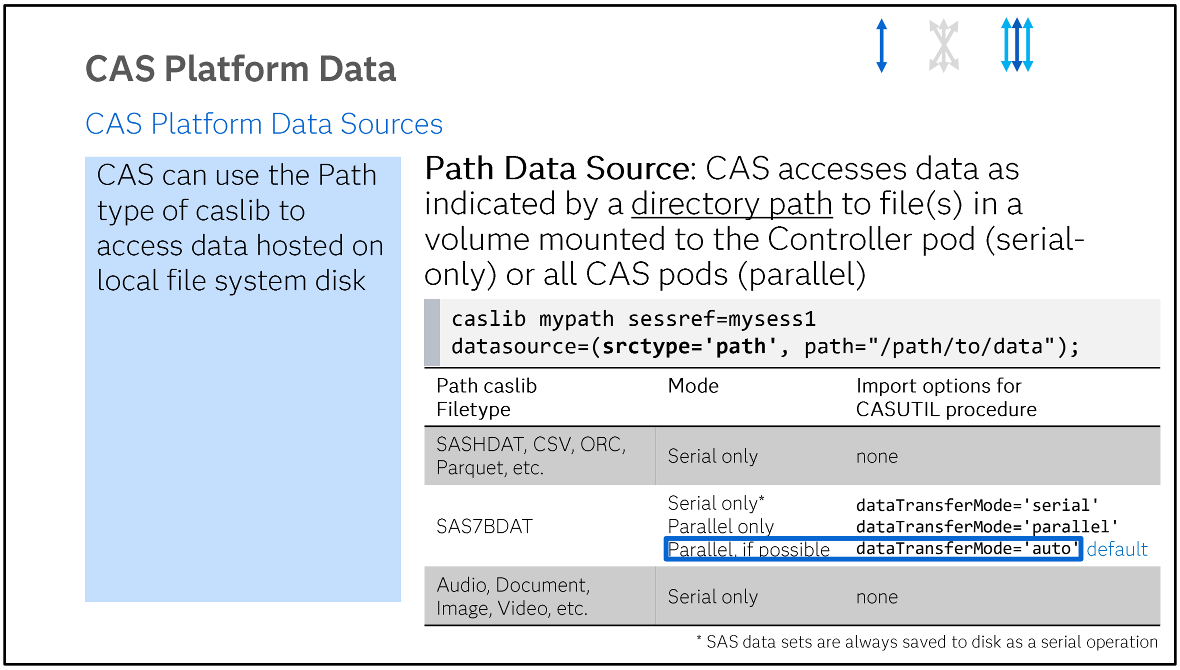
Next are the SAS Data Connectors. Most of those are normally included with the SAS Viya platform. And as you can see, that circle is getting bigger. CAS can access a large number of third-party sources and formats using data connectors using serial, parallel, or multi-node transfer.

Moving on to the SAS Viya Programming Runtime Environment which offers additional SAS/ACCESS products that CAS might not have yet. We can use that as a bridge, if needed, to bring remote data to CAS.

And then SAS 9.4 and its possible SAS/ACCESS products cover everything else. Of course, any data source that's ODBC or JDBC compliant can be reached at any level here.

And lastly, more of a footnote to point out that CAS can directly load data from the SAS LASR Analytic Server which itself could be loaded with data in parallel from several sources.

Platform Data Sources



CAS Platform Data

CAS Platform Data Sources

CAS can use the Path type of caslib to access data hosted on local file system disk

Path Data Source: CAS accesses data as indicated by a directory path to file(s) in a volume mounted to the Controller pod (serial-only) or all CAS pods (parallel)

```
caslib mypath sessref=mysess1
datasource=(srctype='path', path="/path/to/data");
```

Path caslib Filetype	Mode	Import options for CASUTIL procedure
SASHDAT, CSV, ORC, Parquet, etc.	Serial only	none

SAS7BDAT	Serial only*
	Parallel only
	Parallel, if possible
		`dataTransferMode='serial'`
		`dataTransferMode='parallel'`
		`dataTransferMode='auto'` default

Audio, Document, Image, Video, etc.	Serial only	none

* SAS data sets are always saved to disk as a serial operation

First off, CAS can use the PATH type of caslib to access data that's hosted on local file system disk. That means that it's going to refer to a directory path that's in a volume that's mounted to the CAS controller or optionally to all CAS pods.

The caslib statement for PATH is pretty straightforward. After the “caslib” keyword, we provide the following parameters. The first is the name of the caslib, in this case that’s “my path”. For clarity, we also specify the CAS session that this caslib should refer to. Next are the data source parameters – this is where we indicate the type of caslib with the “source type” parameter equal to “path”. And then we specify the path to the volume mount as seen inside the CAS pod (typically defined by a Kubernetes persistent volume claim).

With our path caslib defined, let’s look at the kinds of data it can access.

In the first column of this table, we see the file types supported for the PATH caslib, including SASHDAT, CSV, ORC, and Parquet. There are also SAS datasets as well as various audio, image, video, and document formats.

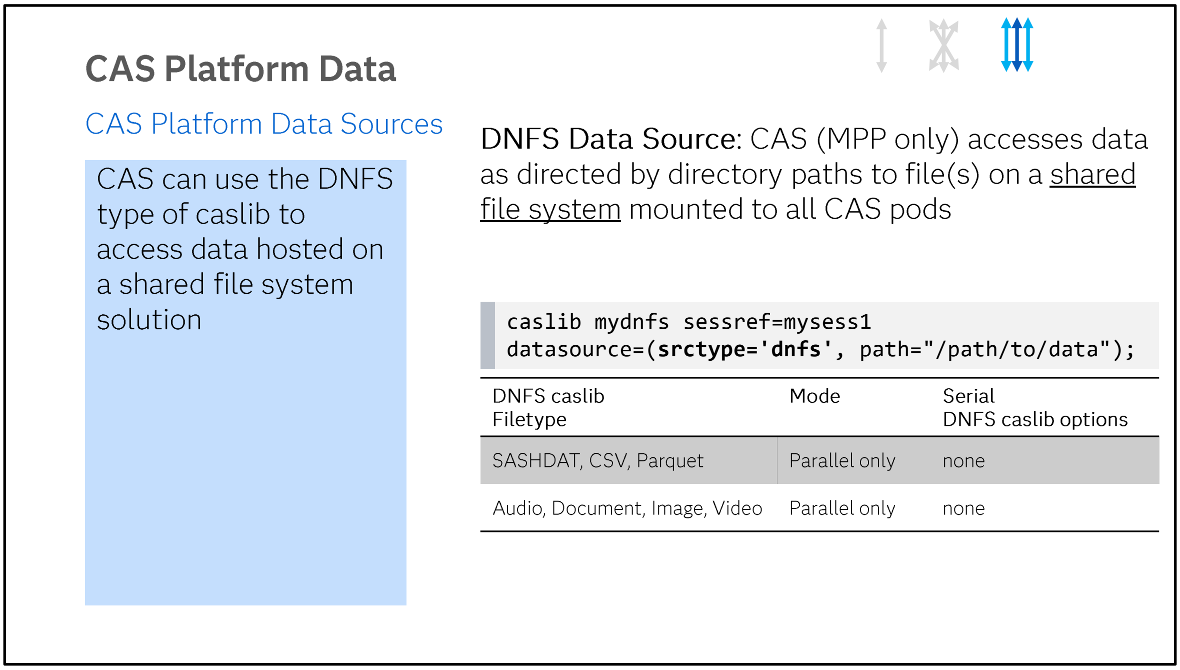
The second column indicates the data transfer mode supported by the PATH caslib for these formats. For most formats, only serial transfer is supported. But there’s one exception: SAS data sets. When using the PATH type of caslib with MPP CAS where the same shared volume is mounted identically to the CAS controller and all CAS workers, then CAS can perform a parallel transfer of the data from the SAS data set such that each CAS worker grabs its allotment of data from the file as directed by the CAS controller.

The third column of the table shows how we control that – by specifying the desired ”data transfer mode” value as an ”import option” in the CASUTIL procedure.

Notice that the data transfer mode shown here does not apply to the caslib itself – but as an import option for CASUTIL specifically.

Finally, unless otherwise specified, the default value of the ”data transfer mode” as an ”import option” for the CASUTIL procedure will be “auto”. This directs CAS to attempt parallel transfer of the data if possible. If there’s any problem with that, then it will automatically fall back to attempt a serial transfer.

One last note to say that SAS data sets are always saved back to disk using the serial data transfer approach.



CAS Platform Data

CAS Platform Data Sources

CAS can use the DNFS type of caslib to access data hosted on a shared file system solution

DNFS Data Source: CAS (MPP only) accesses data as directed by directory paths to file(s) on a shared file system mounted to all CAS pods

```
caslib mydnfs sessref=mysess1
datasource=(srctype='dnfs', path="/path/to/data");
```

DNFS caslib Filetype	Mode	Serial DNFS caslib options
SASHDAT, CSV, Parquet	Parallel only	none
Audio, Document, Image, Video	Parallel only	none

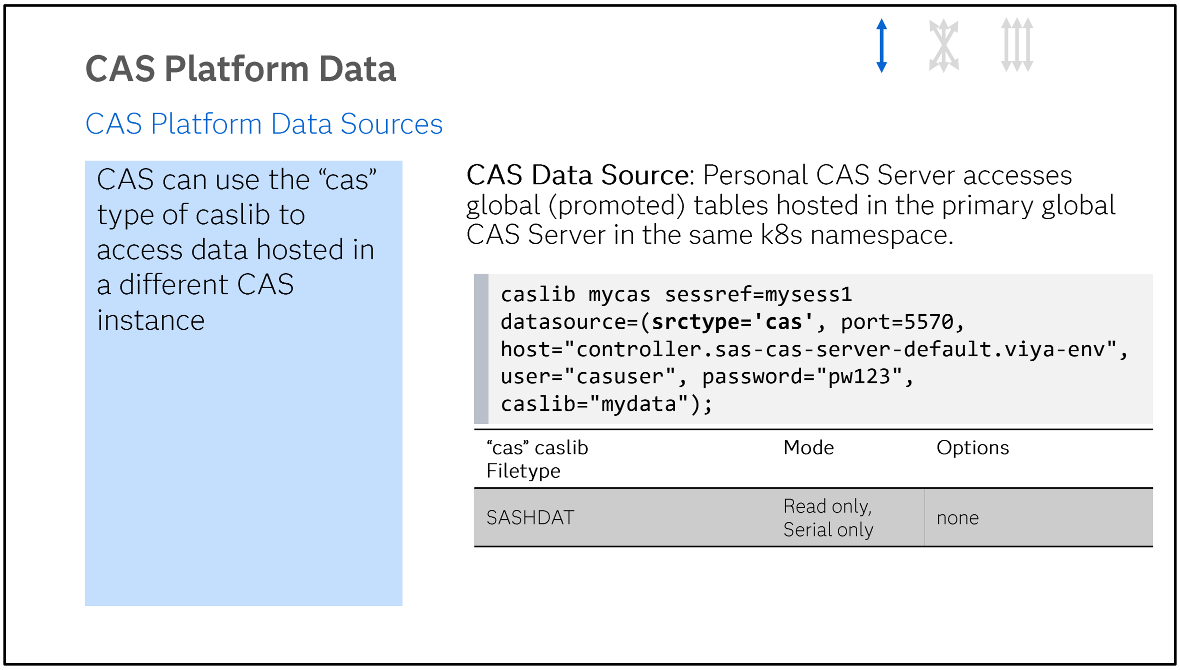
CAS can use the DNFS type of caslib to access data hosted on a shared file system solution. That is, the data must be in a volume that's mounted identically to all CAS pods.

The syntax to define this caslib is very similar to PATH; the only meaningful difference is the "source type".

The DNFS caslib was originally designed for working with SASHDAT files. And in the first column of this table, we see the filetypes supported for the DNFS caslib, including SASHDAT, CSV, and Parquet. There are also various audio, image, video, and document formats.

The second column indicates the data transfer mode and only parallel is offered. In other words, DNFS is synonymous with parallel data movement. If there's a problem in the environment that prevents all of the CAS pods from seeing the data, then DNFS won't be able to execute the transfer.

The third column indicates that there's no option at all for serial movement. It's parallel for DNFS or not at all. This also means that the DNFS type of caslib is not available for SMP CAS (deployed as a single pod in Kubernetes).



CAS Platform Data

CAS Platform Data Sources

CAS can use the “cas” type of caslib to access data hosted in a different CAS instance

CAS Data Source: Personal CAS Server accesses global (promoted) tables hosted in the primary global CAS Server in the same k8s namespace.

```
caslib mycas sessref=mysess1
datasource=(srctype='cas', port=5570,
host="controller.sas-cas-server-default.viya-env",
user="casuser", password="pw123",
caslib="mydata");
```

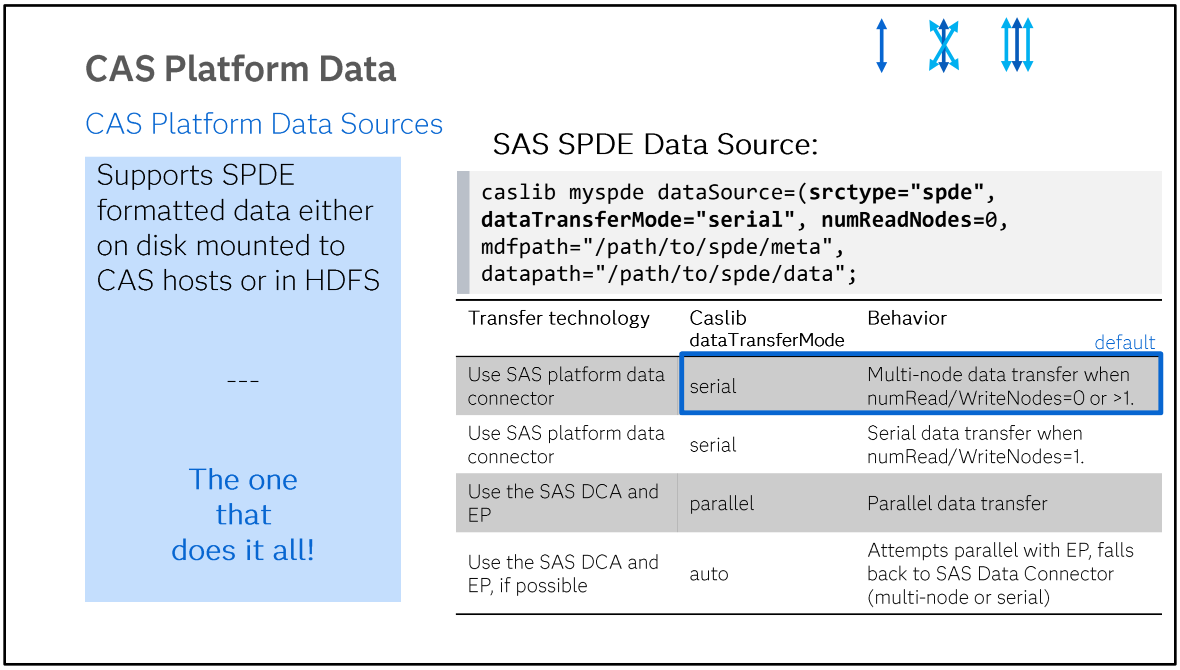
“cas” caslib Filetype	Mode	Options
SASHDAT	Read only, Serial only	none

The next platform data source is accessed using the “cas” type of caslib, which is a little bit self-referential. It was built with one use case in mind: where an individual user has fired up a personal CAS server and wants to access some table that is hosted in the global CAS server. This caslib allows the user to transfer the data down to their playpen and work on it.

In this table, we can see that the “cas” type of caslib works with only one data format: SASHDAT. Remember that all CAS in-memory data is in SASHDAT format, so this makes sense.

And for data transfer, notice that it’s not just serial-only – because a personal CAS server is SMP (runs as a single pod) – but it’s also read-only. That is, it’s a one-way street – this type of caslib can load data but does not save it back “up” to the remote CAS server.

There are no data movement options to specify.



CAS Platform Data

CAS Platform Data Sources

Supports SPDE formatted data either on disk mounted to CAS hosts or in HDFS

The one that does it all!

SAS SPDE Data Source:

```
caslib myspde dataSource=(srctype="spde",
dataTransferMode="serial", numReadNodes=0,
mdfpath="/path/to/spde/meta",
datapath="/path/to/spde/data";
```

Transfer technology	Caslib dataTransferMode	Behavior
		default
Use SAS platform data connector	serial	Multi-node data transfer when numRead/WriteNodes=0 or >1.
Use SAS platform data connector	serial	Serial data transfer when numRead/WriteNodes=1.
Use the SAS DCA and EP	parallel	Parallel data transfer
Use the SAS DCA and EP, if possible	auto	Attempts parallel with EP, falls back to SAS Data Connector (multi-node or serial)

CAS can also work directly with data from the SAS Scalable Performance Data Engine as a source.

SPDE is a predecessor for the large-scale, multi-channel loading of data into SAS that debuted long before CAS or Viya were on the scene. SPDE tables are saved as multiple files that can be loaded in parallel. As such, they can be saved to a performant shared file system or to Hadoop's distributed file system. Either way, cass can read those files directly. And this is where it gets kind of exciting.

That's because the SPDE caslib is the one that does it all. You'll notice at the top of the screen that the icons show all three modes of data transfer are available: serial, parallel, and multi-node. That's very cool.

Looking at the caslib statement shown here, there are a few items of interest to note besides the "source type" of "spde". When the "data transfer mode" is "serial", then that unlocks a new parameter we haven't seen yet called "num read nodes" (and there's an equivalent not shown here called "num write nodes"). There are also some additional parameters unique to working with SPDE files that are specified as well for meta and data files.

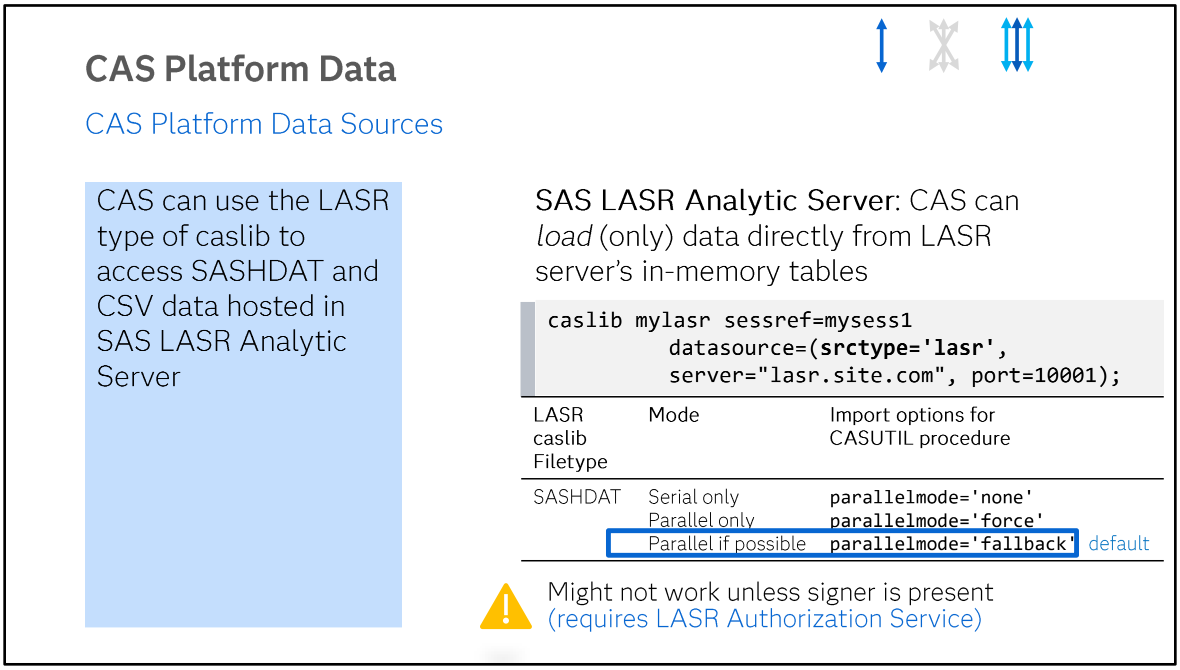
When working with the "data transfer mode" as a caslib parameter (not as an "import option" on the CASUTIL procedure), remember that its meaning is slightly more abstract in that it directs the kind of technology employed, not necessarily the literal meaning of the word.

When the "data transfer mode" value is "serial", that tells CAS to utilize its Data Connector technology to get the data. With a "source type" that accepts the "serial" data transfer mode, then we can also specify the "num read nodes". This new parameter tells CAS how many CAS nodes should participate in the data movement. If "num read nodes" is "1", then the CAS controller handles the transfer from source alone, literally moving the data serially. If "num read nodes" is an integer greater than one, then CAS will use that number of CAS workers to move the data in a technique we refer to as multi-node transfer. And finally, "0" is a special value meaning simply "use all available CAS workers". Negative values for "num read nodes" are not allowed.

Moving on, when the “data transfer mode” value is “parallel”, that tells CAS to use its Data Connect Accelerator technology. Now this assumes that the SAS In-Database Embedded Process has been deployed to the remote data source (in this case, Hadoop). When this is attempted, then all instances of the EP send their data to all CAS workers directly, resulting in parallel data transfer.

The last value for “data transfer mode” is “auto”. This directs CAS to first try using its Data Connect Accelerator technology to get the data. If that fails for some reason, then it will fall back to using its Data Connector technology instead. Depending on how the caslib is defined, this might include attempting multi-node transfer eventually falling all the way back to serial transfer, if needed.

If key parameters specifying the ”data transfer mode” or “num read nodes” are not defined in the caslib statement, then the default behavior you should expect to see will be “serial” – meaning use of the Data Connector technology - with a “num read nodes” value of “0” so that all CAS workers will attempt to participate in the multi-node transfer of data from the source.



CAS Platform Data

CAS Platform Data Sources

CAS can use the LASR type of caslib to access SASHDAT and CSV data hosted in SAS LASR Analytic Server

SAS LASR Analytic Server: CAS can *load* (only) data directly from LASR server's in-memory tables

```
caslib mylasr sessref=mysess1
      datasource=(srctype='lasr',
      server="lasr.site.com", port=10001);
```

LASR caslib Filetype	Mode	Import options for CASUTIL procedure	
SASHDAT	Serial only	parallelmode='none'	
	Parallel only	parallelmode='force'	
	Parallel if possible	parallelmode='fallback'	default

⚠ Might not work unless signer is present (requires LASR Authorization Service)

CAS can load data directly from SAS laser Server.

This situation might come up when working with a legacy SAS 9 environment. CAS can communicate with the laser Server and perform a read-only operation to copy data out of the laser in-memory tables. CAS cannot write data directly back to laser. And laser cannot use CAS as a data source.

When the “source type” is “laser”, then the caslib statement will accept parameters like server and port to establish the connection.

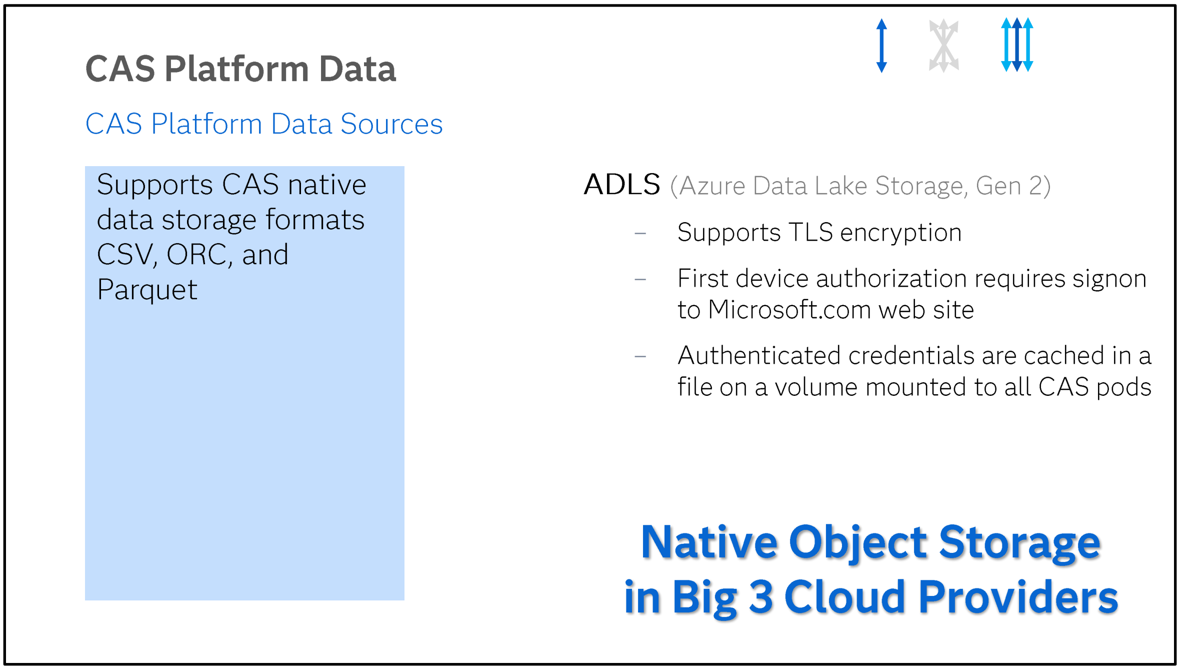
Similar to the ”CAS” type of caslib, the only format that works is SASHDAT. That’s because laser, as the immediate predecessor to CAS, stores in-memory data exclusively in SASHDAT format.

The data transfer mode could be serial or parallel depending on the whether CAS is SMP or MPP or if the laser Server is distributed or non-distributed. Assuming multi-host implementations of both, we can specify the transfer mode with a parameter called “parallel mode”. It accepts three different values than we’ve seen with “data transfer mode”, but it’s easy to understand what they mean.

If “parallel mode” is “none”, then serial movement of the data is performed. If the value is “force”, then parallel movement is attempted using all laser data nodes and all CAS workers. If there’s a problem such that all hosts cannot participate as expected, then the parallel transfer will fail. And finally, there’s the “fallback” value for “parallel mode”. This will direct CAS to attempt parallel transfer first and, if there’s a problem, then it will fall back to attempt serial movement.

If the “parallel mode” parameter is not specified as an “import option” in the CASUTIL procedure, then the default behavior will be “fallback”.

One last point to mention here is that CAS expects the laser Authorization Service to be running as a signer so it can confirm it has appropriate access to the laser tables in memory.



CAS Platform Data

CAS Platform Data Sources

Supports CAS native data storage formats CSV, ORC, and Parquet

ADLS (Azure Data Lake Storage, Gen 2)

- Supports TLS encryption
- First device authorization requires signon to Microsoft.com web site
- Authenticated credentials are cached in a file on a volume mounted to all CAS pods

Native Object Storage in Big 3 Cloud Providers

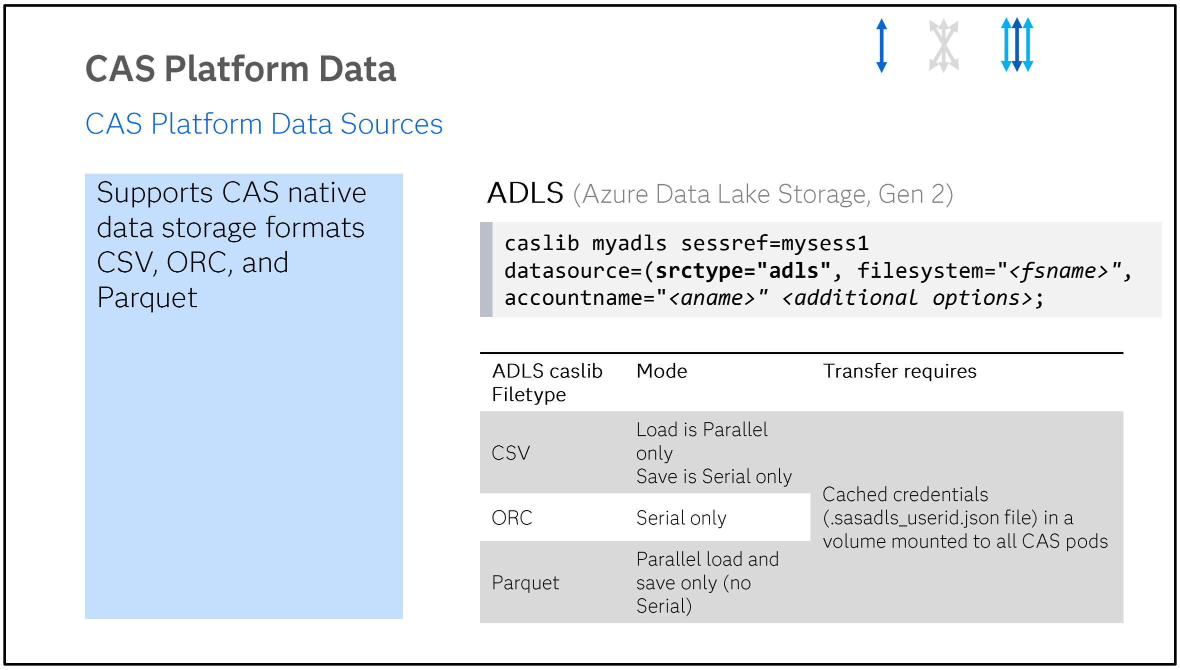
CAS can also directly access data in native storage formats from the big three cloud providers, and we'll take a quick survey of those real fast.

The first one we'll look at is ADLS. That's Azure's Data Lake Storage Gen 2.

It supports TLS encryption, so the movement of data between CAS and ADLS is protected.

The first device authorization requires sign-on to a Microsoft.com website – an increasingly common convention where end-user authentication is moved out-of-band from the application path to generate an auth token.

Then the authenticated credentials are cached in a file on a volume that's mounted to all the CAS pods so that it can be reused until it expires.



CAS Platform Data

CAS Platform Data Sources

Supports CAS native data storage formats CSV, ORC, and Parquet

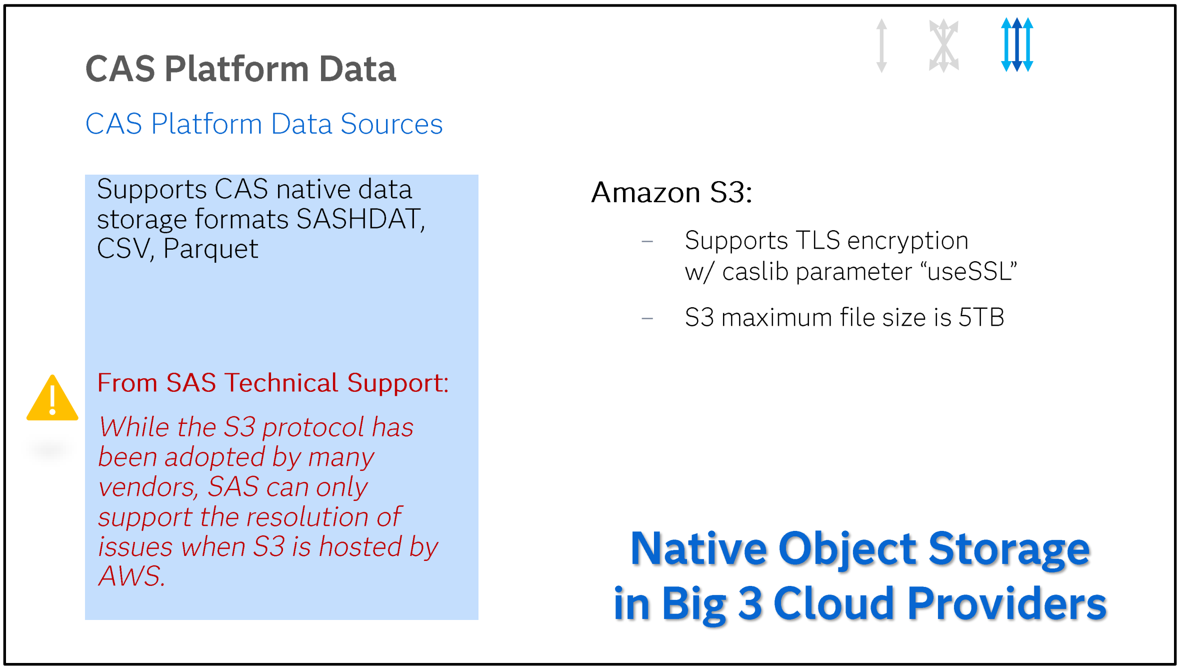
ADLS (Azure Data Lake Storage, Gen 2)

```
caslib myadls sessref=mysess1
datasource=(srctype="adls", filesystem="<fsname>",
accountname="<aname>" <additional options>;
```

ADLS caslib Filetype	Mode	Transfer requires
CSV	Load is Parallel only Save is Serial only	Cached credentials (.sasadls_userid.json file) in a volume mounted to all CAS pods
ORC	Serial only	
Parquet	Parallel load and save only (no Serial)	

As we've seen before with other data sources, the ADLS "source type" has specific parameters and options for communicating with the Azure Data Lake Storage.

The ADLS "source type" supports data in CSV, ORC, and Parquet formats. And interestingly enough, the determination of serial or parallel movement of the data is automatically determined by the file type. For CSV, load is parallel only, but saving back to ADLS is serial only. For ORC, only serial movement is allowed in either direction. And for Parquet, only parallel movement is provided.



CAS Platform Data

CAS Platform Data Sources

Supports CAS native data storage formats SASHDAT, CSV, Parquet

⚠ From SAS Technical Support:

While the S3 protocol has been adopted by many vendors, SAS can only support the resolution of issues when S3 is hosted by AWS.

Amazon S3:

- Supports TLS encryption w/ caslib parameter "useSSL"
- S3 maximum file size is 5TB

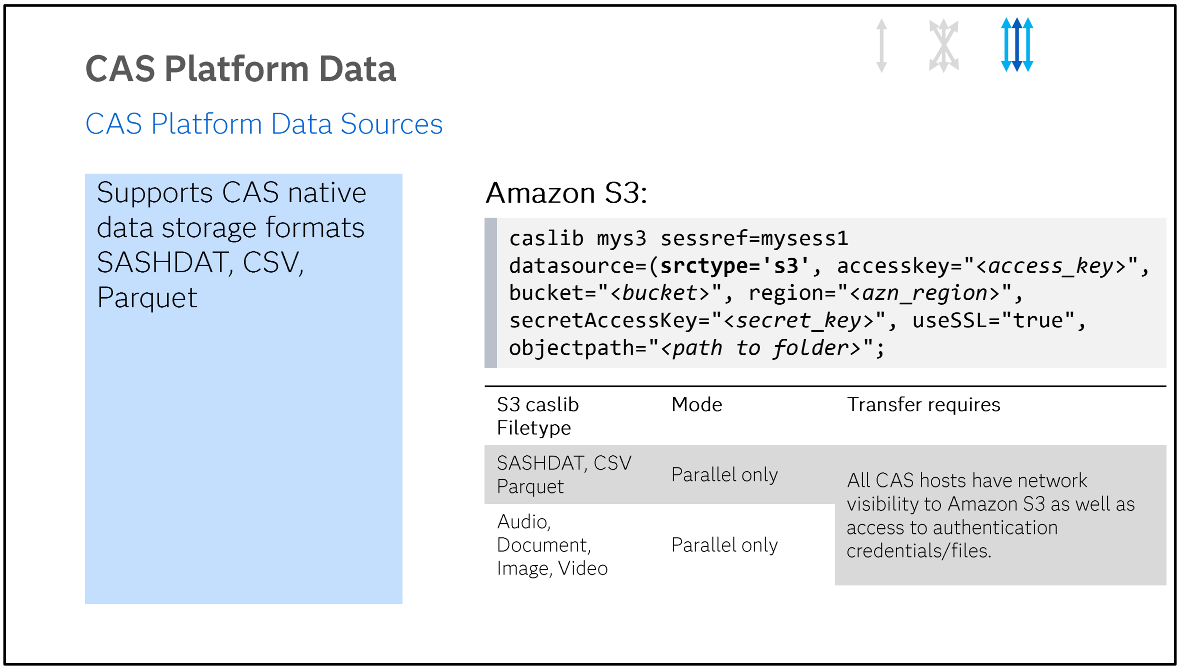
Native Object Storage in Big 3 Cloud Providers

Similarly, we can look at Amazon's S3 – which stands for ”simple storage service” – as another platform data source for CAS.

It also supports TLS encryption to protect communication between the CAS Server and AWS.

And there is a maximum file size limit of 5 terabytes. But understand that that's not a CAS limitation. That’s an S3 restriction that it places on file sizes.

Note that SAS documentation explains, “SAS provides S3 support only for Amazon S3. Third-party compatible providers that implement the public Amazon S3 API might work, but SAS has not validated these providers. As a result, SAS cannot provide direct technical support for other S3-compatible providers.”



CAS Platform Data

CAS Platform Data Sources

Supports CAS native data storage formats SASHDAT, CSV, Parquet

Amazon S3:

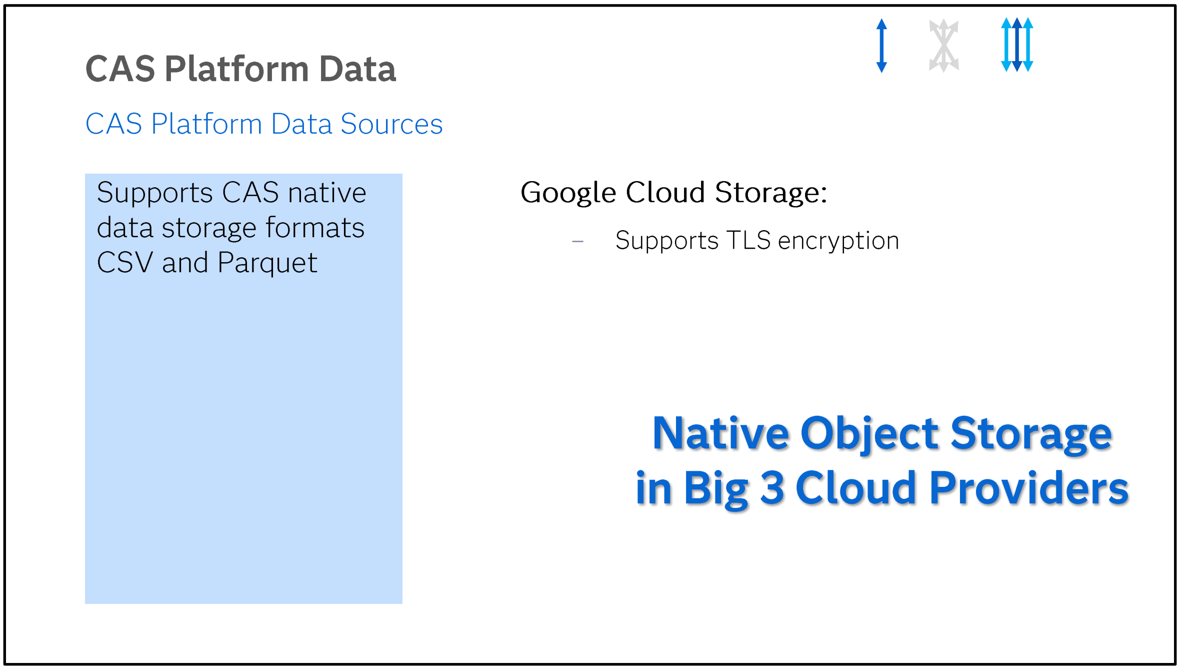
```
caslib mys3 sessref=mysess1
datasource=(srctype='s3', accesskey="<access_key>",
bucket="<bucket>", region="<azn_region>",
secretAccessKey="<secret_key>", useSSL="true",
objectpath="<path to folder>";
```

S3 caslib Filetype	Mode	Transfer requires
SASHDAT, CSV Parquet	Parallel only	All CAS hosts have network visibility to Amazon S3 as well as access to authentication credentials/files.
Audio, Document, Image, Video	Parallel only	

The S3 “source type” has specific parameters and options for communicating with the Amazon’s S3 service.

CAS can use the S3 caslib to work with data saved as SASHDAT, CSV, and Parquet. It also supports binary formats for audio, video, images, and documents (like Excel workbooks).

Data movement of any file type is always parallel when CAS is working with S3. Serial is not an option. Due to the nature of parallel movement with no fall back to serial, ensure that all CAS hosts (controllers and workers) have network visibility to Amazon S3 as well as access to the authentication credentials/files (typically saved to a common volume).



CAS Platform Data

CAS Platform Data Sources

Supports CAS native data storage formats CSV and Parquet

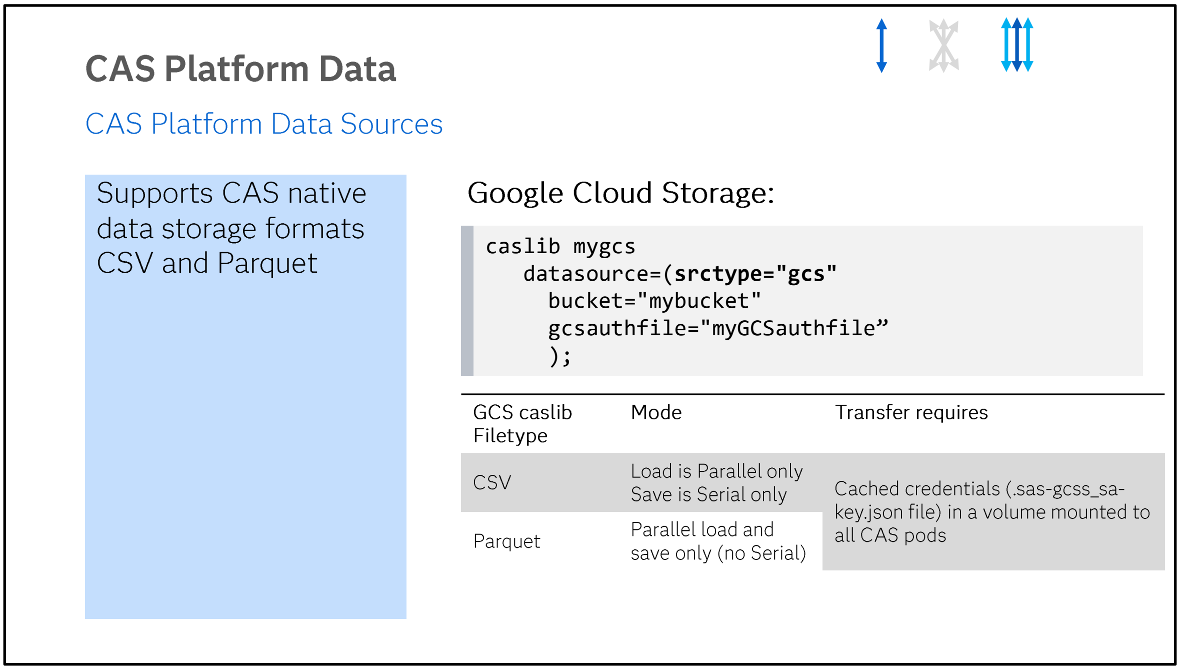
Google Cloud Storage:

- Supports TLS encryption

Native Object Storage in Big 3 Cloud Providers

The last cloud provider we'll look at as a platform data source for CAS is Google Cloud Storage.

Like the others, it supports TLS encryption to protect communication between the CAS Server and GCS.



CAS Platform Data

CAS Platform Data Sources

Supports CAS native data storage formats CSV and Parquet

Google Cloud Storage:

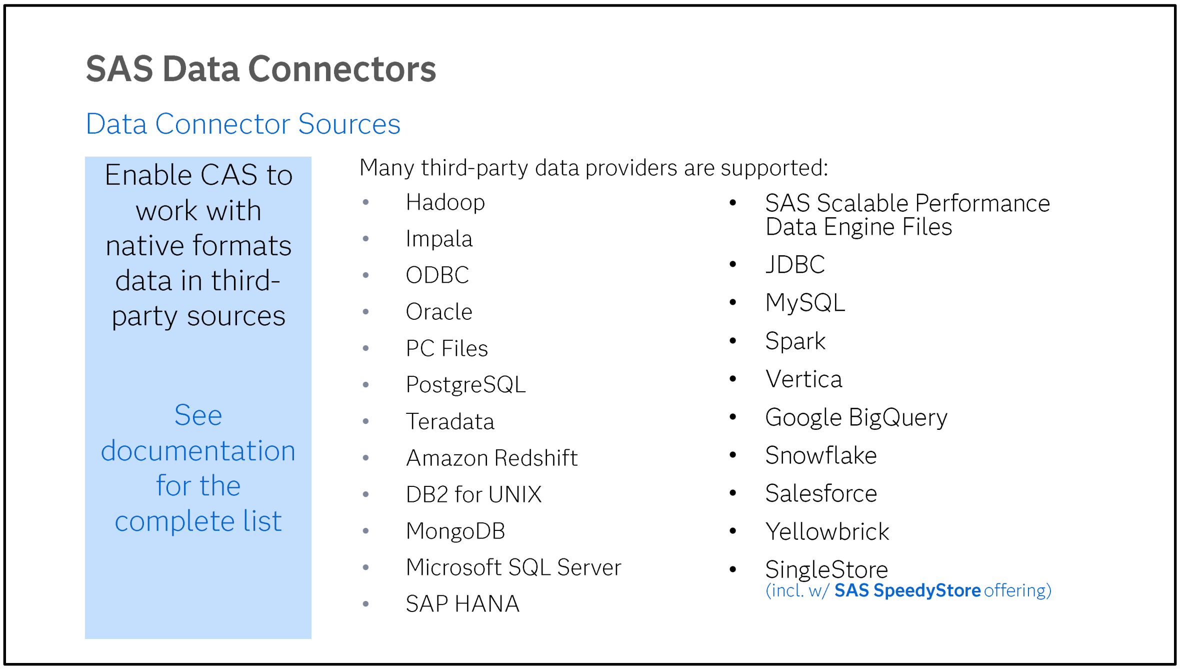
```
caslib mygcs
   datasource=(srctype="gcs"
     bucket="mybucket"
     gcsauthfile="myGCSauthfile”
     );
```

GCS caslib Filetype	Mode	Transfer requires
CSV	Load is Parallel only Save is Serial only	Cached credentials (.sas-gcss_sa-key.json file) in a volume mounted to all CAS pods
Parquet	Parallel load and save only (no Serial)	

The GCS “source type” has specific parameters and options for communicating with Google Cloud Storage.

CAS can use the GCS caslib to work with data saved in either CSV or Parquet files. For CSV files, loading is parallel only and saving is serial only. For Parquet, it's a fully parallel for both load and save, but there is no option for serial transfer.

Data Connector Sources



SAS Data Connectors

Data Connector Sources

Enable CAS to work with native formats data in third-party sources

See documentation for the complete list

Many third-party data providers are supported:

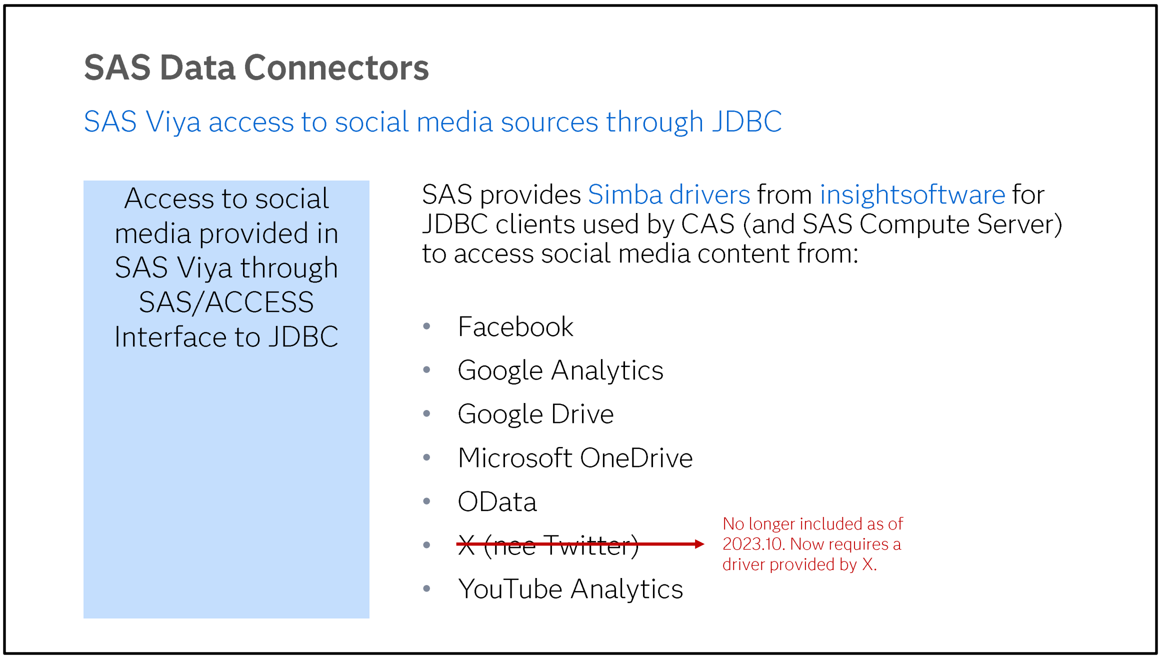
- Hadoop
- Impala
- ODBC
- Oracle
- PC Files
- PostgreSQL
- Teradata
- Amazon Redshift
- DB2 for UNIX
- MongoDB
- Microsoft SQL Server
- SAP HANA
- SAS Scalable Performance Data Engine Files
- JDBC
- MySQL
- Spark
- Vertica
- Google BigQuery
- Snowflake
- Salesforce
- Yellowbrick
- SingleStore
  (incl. w/ **SAS SpeedyStore** offering)

The SAS Viya data connectors enable CAS to work with native format data in third party sources. And here is a pretty good list of the various third-party data providers that are supported by SAS Viya data connectors for CAS.

You'll certainly recognize many of the big names in the list, like Oracle, PostgreSQL, MongoDB, Microsoft SQL Server, SingleStore and others.

Still, make sure you refer to the documentation, which is linked in the notes, for a complete list as the offerings evolve over time.

Also note that if a data source is not listed here, but is ODBC or JDBC compliant, then they can be reached through those data connectors by CAS, respectively.



SAS Data Connectors

SAS Viya access to social media sources through JDBC

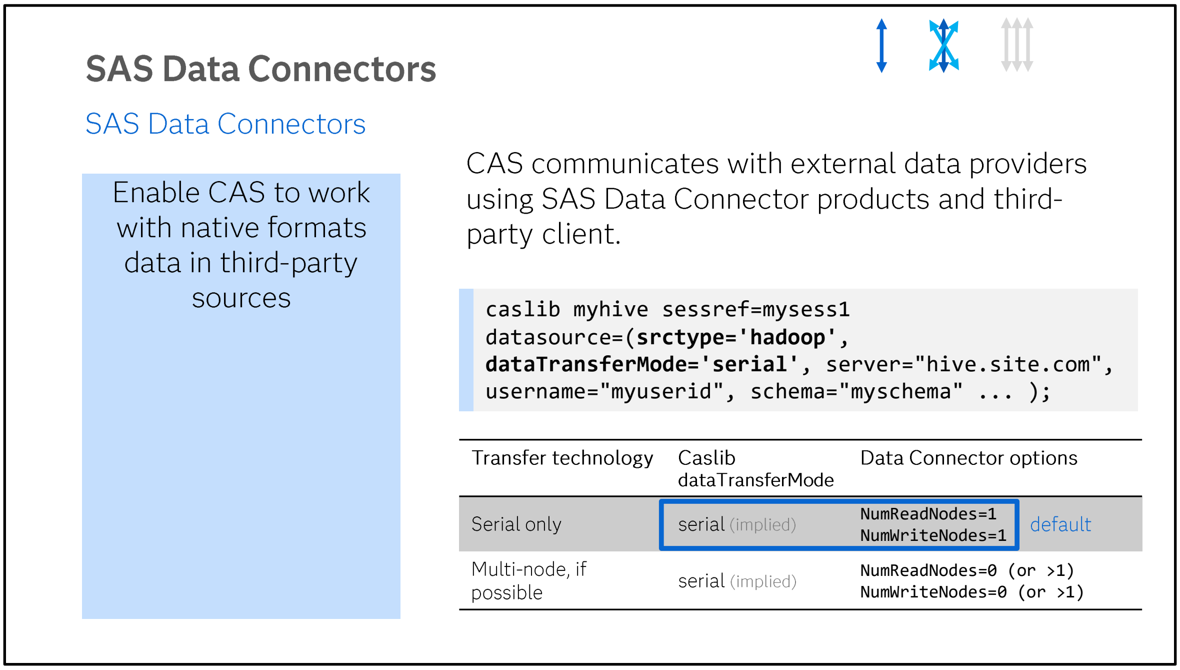
Access to social media provided in SAS Viya through SAS/ACCESS Interface to JDBC

SAS provides Simba drivers from insightsoftware for JDBC clients used by CAS (and SAS Compute Server) to access social media content from:

- Facebook
- Google Analytics
- Google Drive
- Microsoft OneDrive
- OData
- ~~X (nee Twitter)~~
- YouTube Analytics

No longer included as of 2023.10. Now requires a driver provided by X.

SAS provides Simba drivers from insight software for JDBC clients that allow CAS and the SAS Compute server to access social media content from Facebook, Google Analytics, Google Drive, Microsoft, and so on. CAS communicates with external data providers using the data connector products in third party clients.



SAS Data Connectors

SAS Data Connectors

Enable CAS to work with native formats data in third-party sources

CAS communicates with external data providers using SAS Data Connector products and third-party client.

```
caslib myhive sessref=mysess1
datasource=(srctype='hadoop',
dataTransferMode='serial', server="hive.site.com",
username="myuserid", schema="myschema" ... );
```

Transfer technology	Caslib dataTransferMode	Data Connector options	

Serial only	serial (implied)	NumReadNodes=1
		NumWriteNodes=1
			default

Multi-node, if possible	serial (implied)	NumReadNodes=0 (or >1)
		NumWriteNodes=0 (or >1)	

The syntax for defining a caslib that employs a SAS/ACCESS data connector for a third-party database is very similar to what is used for the platform data sources and should look familiar to you at this point.

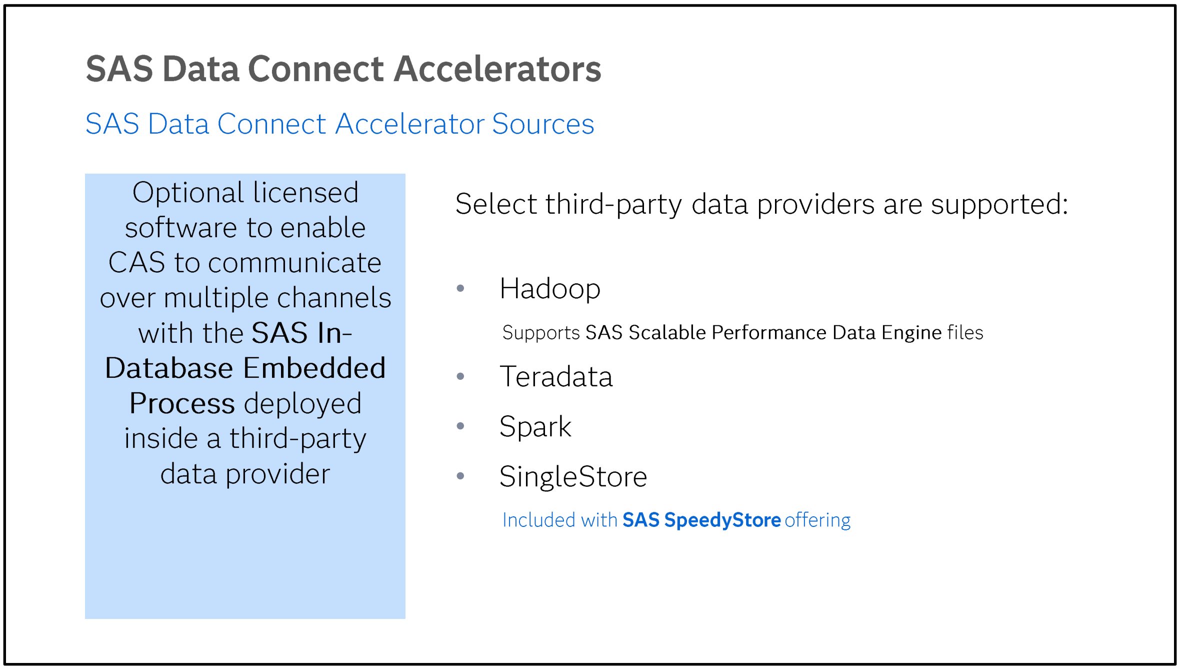
In this example, we're looking at a caslib to reach a Hive database in a Hadoop cluster. The data connector will expect certain parameters that make sense for Hadoop to be included in the caslib definition, like the server name, user name, and schema, among other elements. Other data sources will expect different parameter names and those can be found in the SAS documentation.

Recall one very important item: when we specify the "data transfer mode" in the caslib statement, that's a code-word directing CAS as to which technology to employ. The "serial" value means "use data connector technology" and does not guarantee that serial-only movement will happen because multi-node transfer is also a function of the data connector technology.

Data connectors in general support both serial movement and multi-node movement of the data as core functionality and where the environment (or data source) support it. And we can direct cass as to which type of movement to try.

By default, when using a SAS data connector to a third-party source like Hive in Hadoop, the "num read nodes" (and "num write nodes") parameter in the caslib statement will assume a value of "1". This tells CAS to use only one node to load the data, which means the CAS controller then will attempt a serial transfer.

If we want multi-node transfer using a SAS data connector to a third-party source, then we must specify the "num read nodes" parameter. Remember, a value of "0" here tells CAS to try and use all its workers to make the transfer. Otherwise, a positive integer value will direct CAS to use exactly that number of workers, if possible.



SAS Data Connect Accelerators

SAS Data Connect Accelerator Sources

Optional licensed software to enable CAS to communicate over multiple channels with the **SAS In-Database Embedded Process** deployed inside a third-party data provider

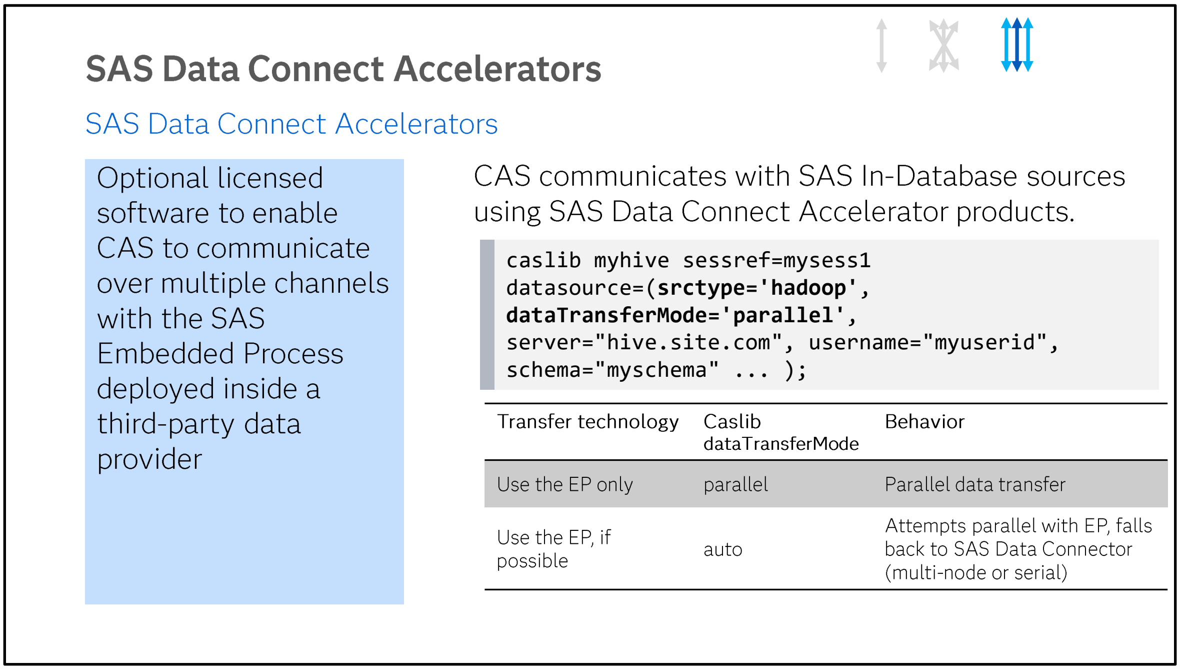
Select third-party data providers are supported:

- Hadoop

  Supports **SAS Scalable Performance Data Engine** files
- Teradata
- Spark
- SingleStore

  Included with **SAS SpeedyStore** offering

We also have our SAS Data Connect Accelerator sources, and they include Hadoop and Teradata and Spark and, of course, SingleStore.



SAS Data Connect Accelerators

SAS Data Connect Accelerators

Optional licensed software to enable CAS to communicate over multiple channels with the SAS Embedded Process deployed inside a third-party data provider

CAS communicates with SAS In-Database sources using SAS Data Connect Accelerator products.

```
caslib myhive sessref=mysess1
datasource=(srctype='hadoop',
dataTransferMode='parallel',
server="hive.site.com", username="myuserid",
schema="myschema" ... );
```

Transfer technology	Caslib dataTransferMode	Behavior
Use the EP only	parallel	Parallel data transfer
Use the EP, if possible	auto	Attempts parallel with EP, falls back to SAS Data Connector (multi-node or serial)

In this scenario, we now want to use the SAS Data Connect Accelerator technology to transfer data from that Hive database in Hadoop. This means that Hadoop already has the SAS In-Database Embedded Process deployed and running there.

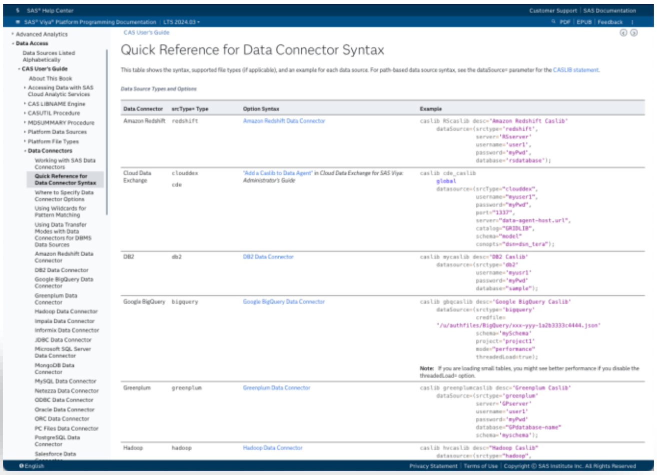
The caslib statement looks similar to before, but with one significant change – the “data transfer mode” is now “parallel”. As you’ll recall, in this syntax, this is directing CAS to implement its data connect accelerator technology so it can communicate with the SAS Embedded Process in Hadoop.

The data connect accelerator technology can only communicate with the SAS Embedded Process. And the SAS Embedded Process is designed to transfer data to CAS using multiple I/O channels to achieve parallel movement of the data.

Alternatively, we can specify a ”data transfer mode” of “auto” that tells CAS to try using its data connect accelerator technology. If there’s a problem that somehow prevents the SAS Embedded Process in the remote source from sending its data to the CAS workers directly, then CAS can fall back and try its data connector technology for serial or multi-node transfer instead.

Notice that we’re not calling out a default mode here because there isn’t one. To use the data connect accelerator technology, you must specify the appropriate “data transfer mode” explicitly on the caslib statement.

Examples of CASLIB syntax by srcType


SAS® Help Center
Customer Support | SAS Documentation
SAS® Viya® Platform Programming Documentation | LTS 2024.03
PDF | EPUB | Feedback
Advanced Analytics
Data Access
Data Sources Listed Alphabetically
CAS User's Guide
About This Book
Accessing Data with SAS Cloud Analytic Services
CAS LIBNAME Engine
CASUTIL Procedure
MDSUMMARY Procedure
Platform Data Sources
Platform File Types
Data Connectors
Working with SAS Data Connectors
Quick Reference for Data Connector Syntax
Where to Specify Data Connector Options
Using Wildcards for Pattern Matching
Using Data Transfer Modes with Data Connectors for DBMS Data Sources
Amazon Redshift Data Connector
DB2 Data Connector
Google BigQuery Data Connector
Greenplum Data Connector
Hadoop Data Connector
Impala Data Connector
Informix Data Connector
JDBC Data Connector
Microsoft SQL Server Data Connector
MongoDB Data Connector
MySQL Data Connector
Netezza Data Connector
ODBC Data Connector
Oracle Data Connector
ORC Data Connector
PC Files Data Connector
PostgreSQL Data Connector
Salesforce Data Connector
English
Privacy Statement | Terms of Use | Copyright © SAS Institute Inc. All Rights Reserved

CAS User's Guide

Quick Reference for Data Connector Syntax

This table shows the syntax, supported file types (if applicable), and an example for each data source. For path-based data source syntax, see the dataSource= parameter for the CASLIB statement.

Data Source Types and Options

Data Connector	srcType= Type	Option Syntax	Example

Amazon Redshift	redshift	Amazon Redshift Data Connector	`caslib RScaslib desc='Amazon Redshift Caslib'`
			`dataSource=(srctype='redshift',`
			`server='RSserver'`
			`username='user1',`
			`password='myPwd',`
			`database='rsdatabase');`

Cloud Data Exchange	clouddex
	cde
		'Add a Caslib to Data Agent' in *Cloud Data Exchange for SAS Viya: Administrator's Guide*	`caslib cde_caslib`
			`global`
			`datasource=(srcType="clouddex",`
			`username="myuser1",`
			`password="myPwd",`
			`port="1337",`
			`server="data-agent-host.url",`
			`catalog="GRIDLIB",`
			`schema="model"`
			`conopts="dsn=dsn_tera");`

DB2	db2	DB2 Data Connector	`caslib mycaslib desc='DB2 Caslib'`
			`datasource=(srctype='db2'`
			`username='myusr1'`
			`password='myPwd'`
			`database="sample");`

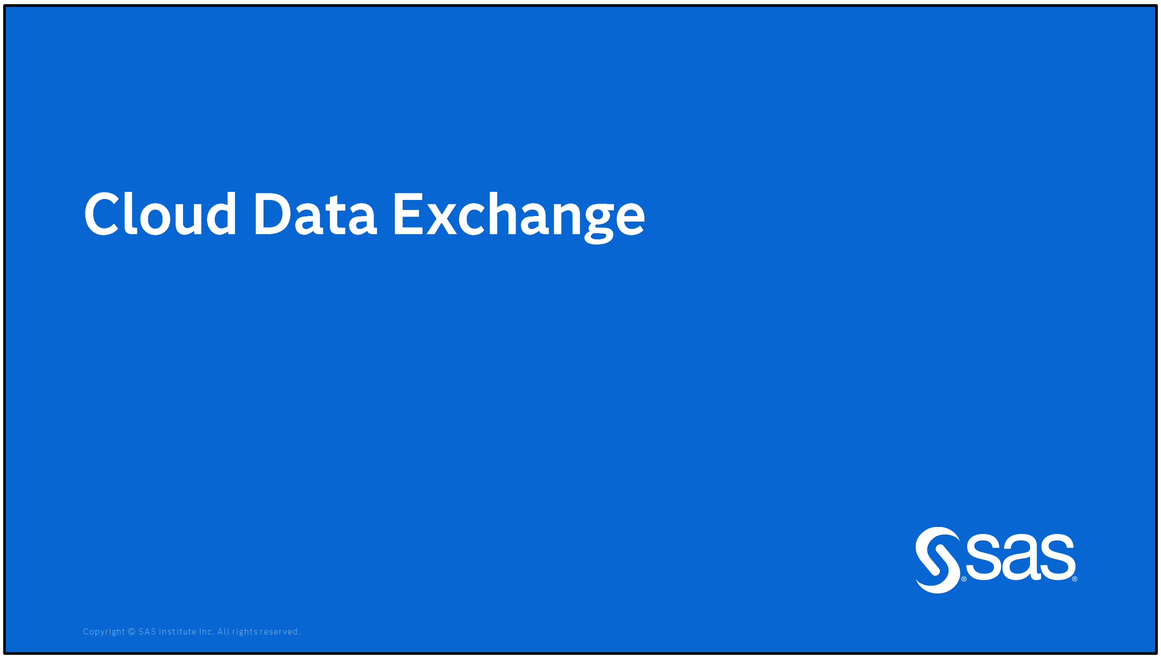
Google BigQuery	bigquery	Google BigQuery Data Connector	`caslib gbqcaslib desc='Google BigQuery Caslib'`
			`dataSource=(srctype='bigquery'`
			`credfile=`
			`'/u/authfiles/BigQuery/xxx-yyy-1a2b3333c4444.json'`
			`schema='mySchema'`
			`project='project1'`
			`mode="performance"`
			`threadedLoad=true);`
			Note: If you are loading small tables, you might see better performance if you disable the threadedLoad= option.

Greenplum	greenplum	Greenplum Data Connector	`caslib greenplumcaslib desc='Greenplum Caslib'`
			`dataSource=(srctype='greenplum'`
			`server='GPserver'`
			`username='user1'`
			`password='myPwd'`
			`database="GPdatabase-name"`
			`schema='myschema');`

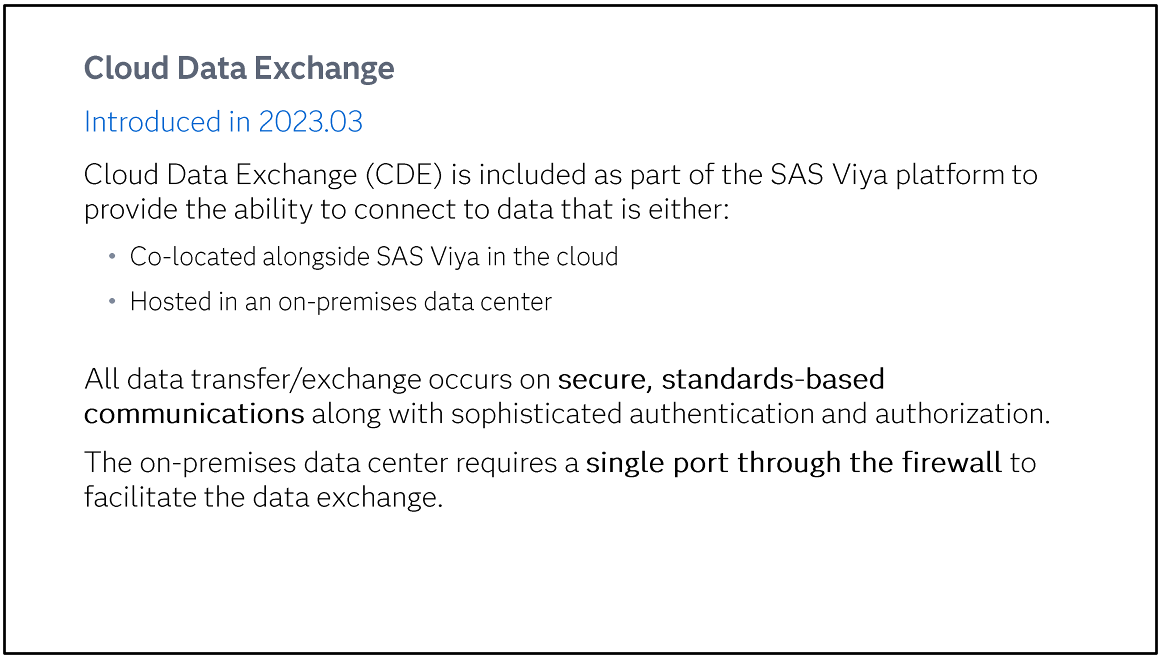
Hadoop	hadoop	Hadoop Data Connector	`caslib hvcaslib desc="Hadoop Caslib"`
			`datasource=(srctype="hadoop",`

See *SAS® Viya® Programming Documentation* > Quick Reference for Data Connector Syntax

In this module, we focused on example caslibs defining connections for CAS to a Hive database in Hadoop. But of course, there are many more data sources and they each have their own set of parameters and considerations to weigh. Those are beyond the scope of this workshop, but the details you'll need can be found in the SAS documentation. A link to this particular document is provided in the slide notes, or you can navigate there referring to the breadcrumb trail at the bottom of the slide shown here.



Cloud Data Exchange



Cloud Data Exchange

Introduced in 2023.03

Cloud Data Exchange (CDE) is included as part of the SAS Viya platform to provide the ability to connect to data that is either:

- Co-located alongside SAS Viya in the cloud
- Hosted in an on-premises data center

All data transfer/exchange occurs on **secure, standards-based communications** along with sophisticated authentication and authorization.

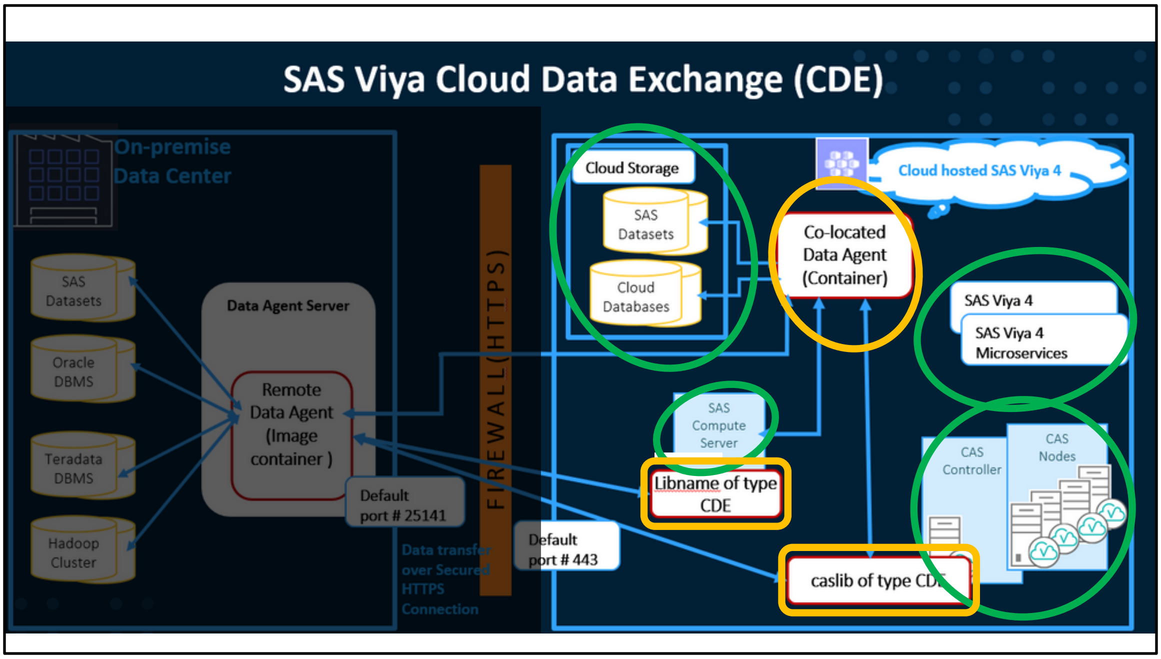
The on-premises data center requires a **single port through the firewall** to facilitate the data exchange.

Cloud Data Exchange (or CDE) was introduced in early 2023 and is included as part of the SAS Viya platform. It provides the ability to connect to data that exists nearby, alongside SAS Viya in the cloud or to data that is hosted in a site's on-premise data center that is physically separate from the SAS Viya location.

The latter is where things get interesting because securing cloud-originated communication to resources on-premise is something that most IT organizations take very seriously.

SAS addresses those concerns by ensuring that data exchange occurs using secure, standards-based communications along with sophisticated authentication and authorization capabilities.

One way this is made clear is that Cloud Data Exchange does not utilize proprietary protocols for communication. Instead, communication is encrypted using TLS as the exchange occurs using a single TCP port. That port is usually the standard 443, however, CDE can be configured to send TLS communication over an alternate port, if preferred.



SAS Viya Cloud Data Exchange (CDE)

On-premise Data Center

SAS Datasets

Oracle DBMS

Teradata DBMS

Hadoop Cluster

Data Agent Server

Remote Data Agent (Image container)

Default port # 25141

FIREWALL(HTTPS)

Data transfer over Secured HTTPS Connection

Default port # 443

Cloud Storage

SAS Datasets

Cloud Databases

Cloud hosted SAS Viya 4

Co-located Data Agent (Container)

SAS Viya 4

SAS Viya 4 Microservices

SAS Compute Server

Libname of type CDE

CAS Controller

CAS Nodes

caslib of type CDE

This next slide shows the SAS Viya platform deployed to the cloud and implementing the Cloud Data Exchange technology to access data that is also hosted in the cloud as well as remote data from a site's on-premise data center.

Let's look a little closer at what's going on here.

Starting on the far-right side, we have SAS Viya and its constituent components

including the usual complement of SAS Viya infrastructure services and microservices.

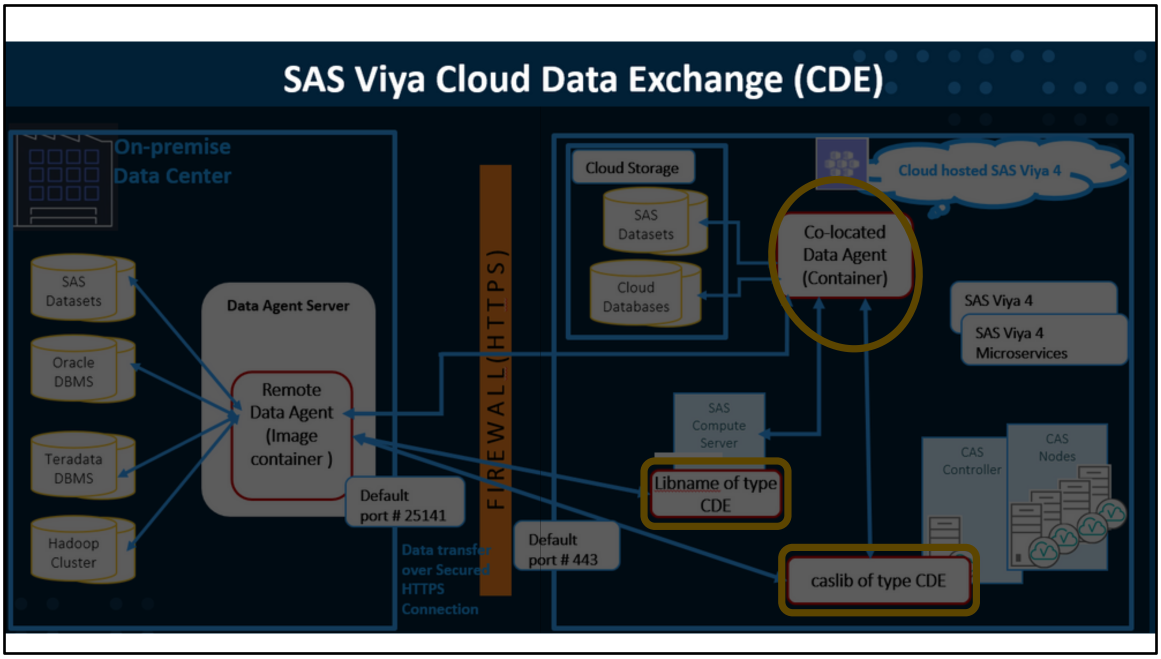
We can also see a CAS server running on multiple hosts and a SAS Compute Server as well.

In that same cloud's storage exist some SAS data sets as well as cloud-hosted data bases.

A key component for Cloud Data Exchange to work is its data agent. We can deploy a co-located data agent (typically run as a Docker or OCI-compliant container) in the cloud and configure it to access the various data sources.

Then we can define a data library for SAS Compute and a caslib for CAS to communicate with the data agent to get to that data.

Although it's slightly more work to setup initially, there are several benefits to using a CDE data agent when compared to allowing CAS or Compute to directly access this data nearby in the cloud. They include keeping the database communication configuration limited to the data agent, not having to be reproduced and maintained across all library definitions in SAS code. The data agent can also be placed into perimeter network zones (like a DMZ) to help support isolation and security of resources while still providing a fully encrypted channel for data movement back to the SAS analytic engines.



SAS Viya Cloud Data Exchange (CDE)

On-premise Data Center

SAS Datasets

Oracle DBMS

Teradata DBMS

Hadoop Cluster

Data Agent Server

Remote Data Agent (Image container)

Default port # 25141

FIREWALL (HTTPS)

Data transfer over Secured HTTPS Connection

Default port # 443

Cloud Storage

SAS Datasets

Cloud Databases

Cloud hosted SAS Viya 4

Co-located Data Agent (Container)

SAS Viya 4

SAS Viya 4 Microservices

SAS Compute Server

Libname of type CDE

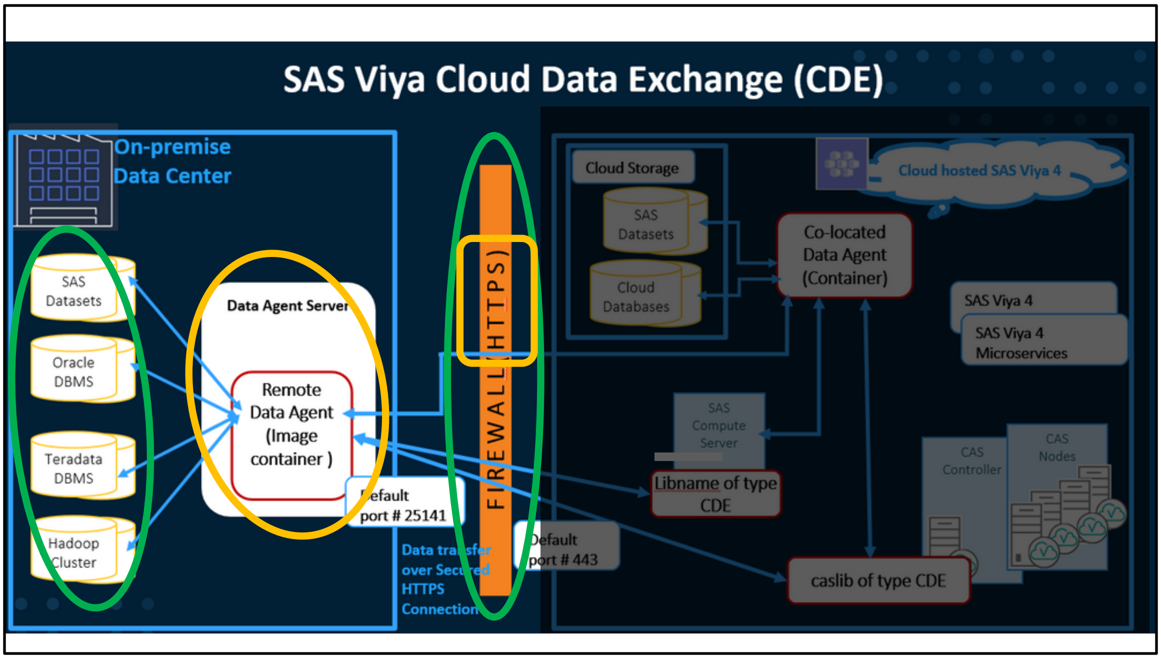
CAS Controller

CAS Nodes

caslib of type CDE

This slide shows another scenario as well.

Outside of the cloud environment, we can see there’s an on-premise data center illustrated on the left side of this slide.



First let’s look at what’s separating the cloud envrionment from the on-premise data center.

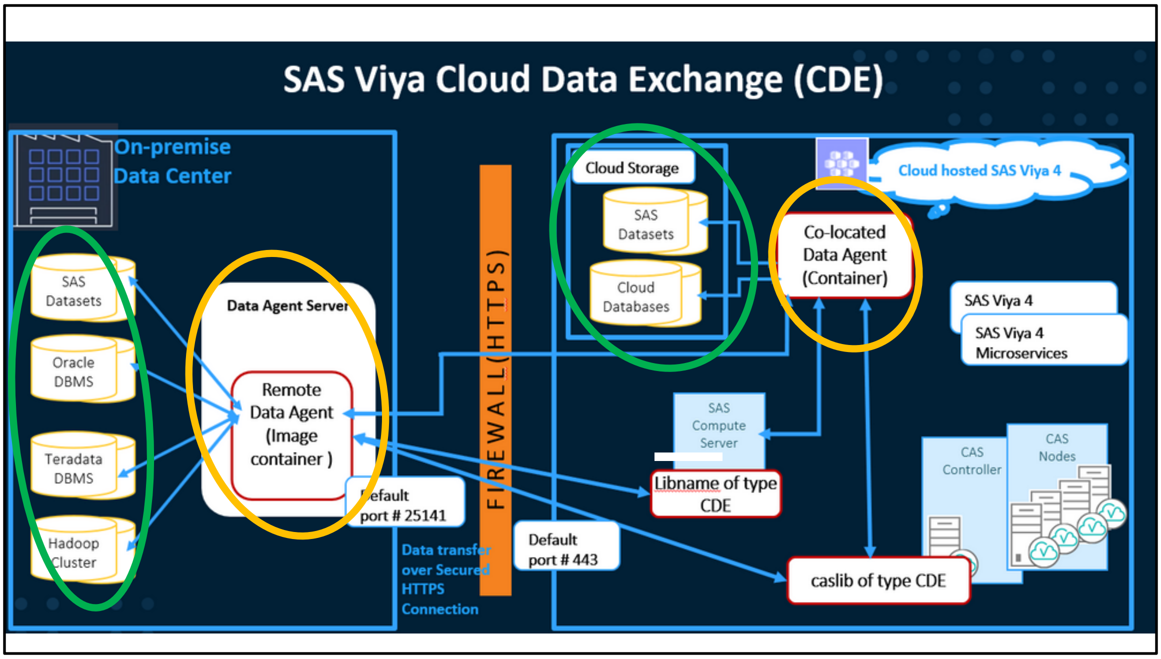
There are likely one or more firewall implementations to protect each environment.

In the on-prem data center, we see several interesting sources with data that we want to run analytics on.

For those, we can also deploy a remote data agent (again, typically run as a Docker or OCI-compliant container). It will be configured to access the various sources and

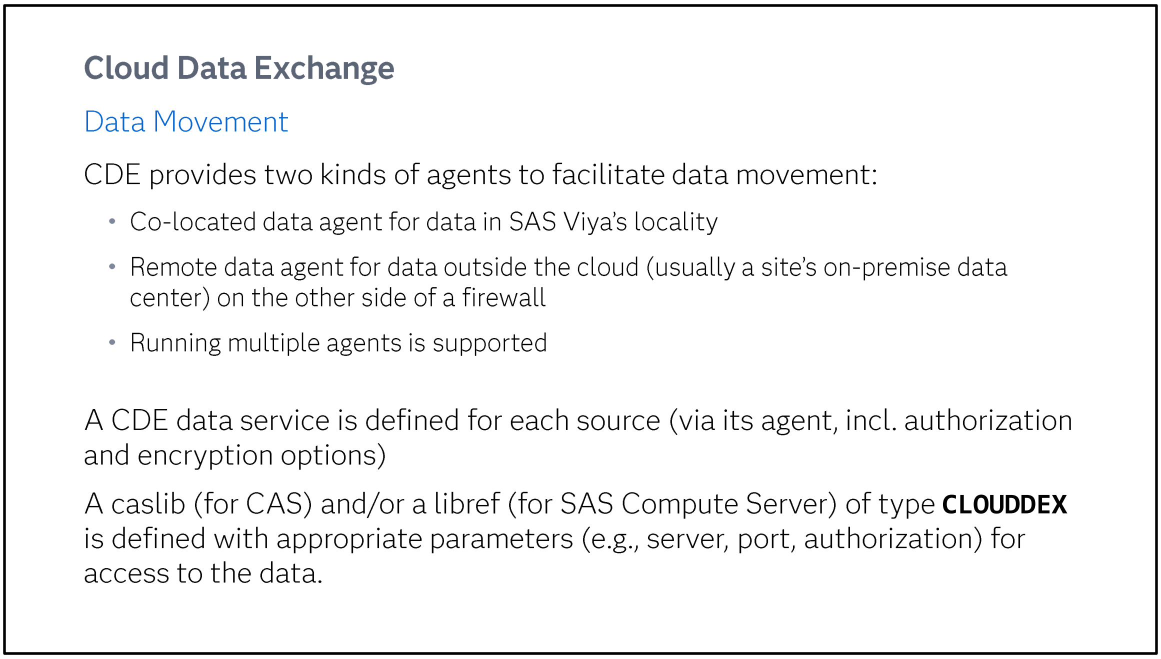
it can relay the requested data back to the Viya analytics over HTTPS (or TLS) encrypted channel. To do this requires only one port opening through the firewall and rules can be enhanced to match on trusted source and destination IP addresses, if needed.

The benefits of a remote data agent as shown in this scenario are similar to that of the co-located data agent in scope, but greater in impact. It’s technically possible for SAS Compute or the CAS server to communicate directly with the various sources in the data center assuming a viable route is provided. That’s a big assumption. And it carries a lot of considerations in terms of opening up the firewall in numerous places to allow for proprietary protocols to establish communication. Further, additional technical challenges might show up, like latency over long distances might not be well handled in some protocols.



Finally

we show only two Cloud Data Exchange data agents here, but each is talking to numerous data sources. We can extend this concept as far as needed when new data sources become available in other environments by deploying data agents to those, too.



Cloud Data Exchange

Data Movement

CDE provides two kinds of agents to facilitate data movement:

- Co-located data agent for data in SAS Viya's locality
- Remote data agent for data outside the cloud (usually a site's on-premise data center) on the other side of a firewall
- Running multiple agents is supported

A CDE data service is defined for each source (via its agent, incl. authorization and encryption options)

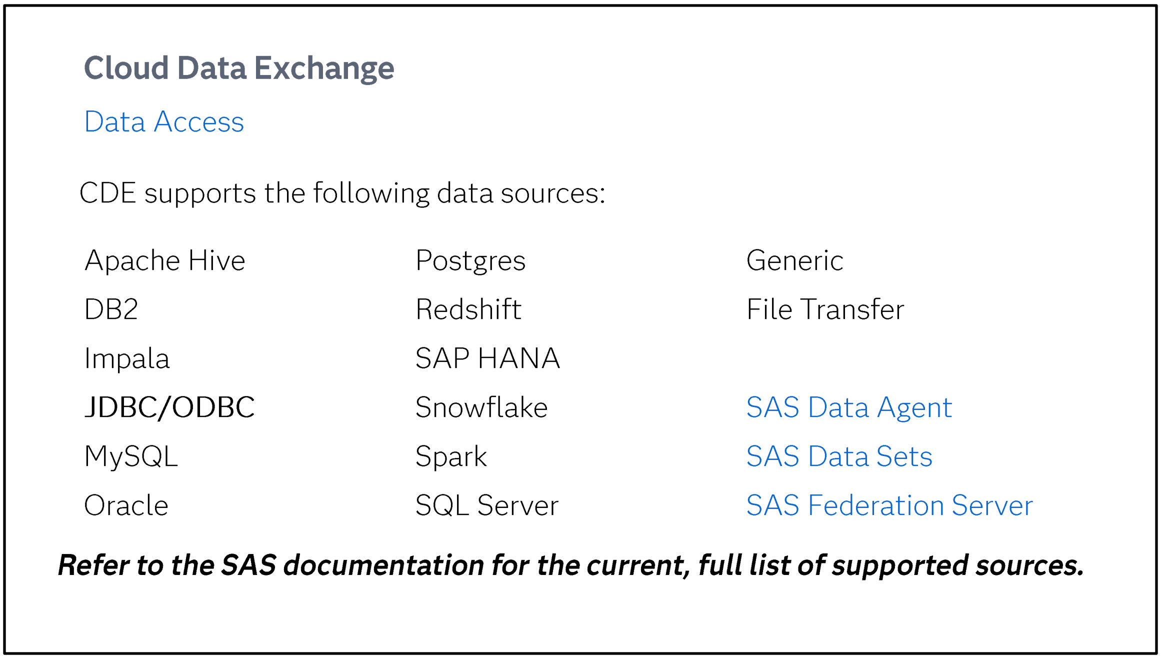
A caslib (for CAS) and/or a libref (for SAS Compute Server) of type **`CLOUDDEX`** is defined with appropriate parameters (e.g., server, port, authorization) for access to the data.

As we just showed, CDE provides two kinds of data agents to facilitate movement of data. There's the co-located data agent that lives in the SAS Viya's locality. It's accessing data that's somewhere nearby in that same hosted environment. We also have the remote data agent which is deployed to an environment that is physically separate from the Viya location.

It is possible to run multiple data agents to support the data movement needs of your Viya deployment.

One data agent can work with multiple data sources. This is accomplished by defining a data service for each source. That data service definition is setup using the sas-viya CLI to direct the agent about how to access the data, how to encrypt it for transfer, and so on).

Once the data agent knows about the various sources it'll work with, then we can define a caslib for CAS or a libref for SAS Compute with the Cloud Data Exchange source type. The library statement will include connection parameters for reaching the desired data agent such as its server and port and further parameters to select the desired data source with authorization details.



Cloud Data Exchange

Data Access

CDE supports the following data sources:

- Apache Hive
- DB2
- Impala
- JDBC/ODBC
- MySQL
- Oracle
- Postgres
- Redshift
- SAP HANA
- Snowflake
- Spark
- SQL Server
- Generic
- File Transfer
- SAS Data Agent
- SAS Data Sets
- SAS Federation Server

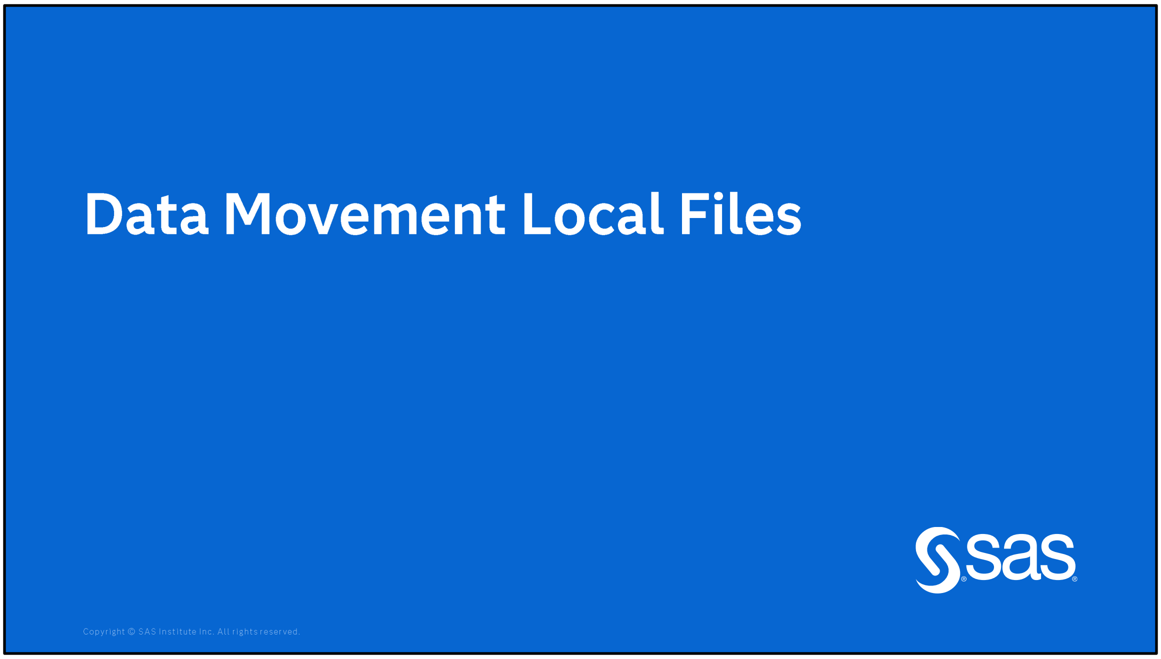
Refer to the SAS documentation for the current, full list of supported sources.

Cloud Data Exchange supports the following data sources.

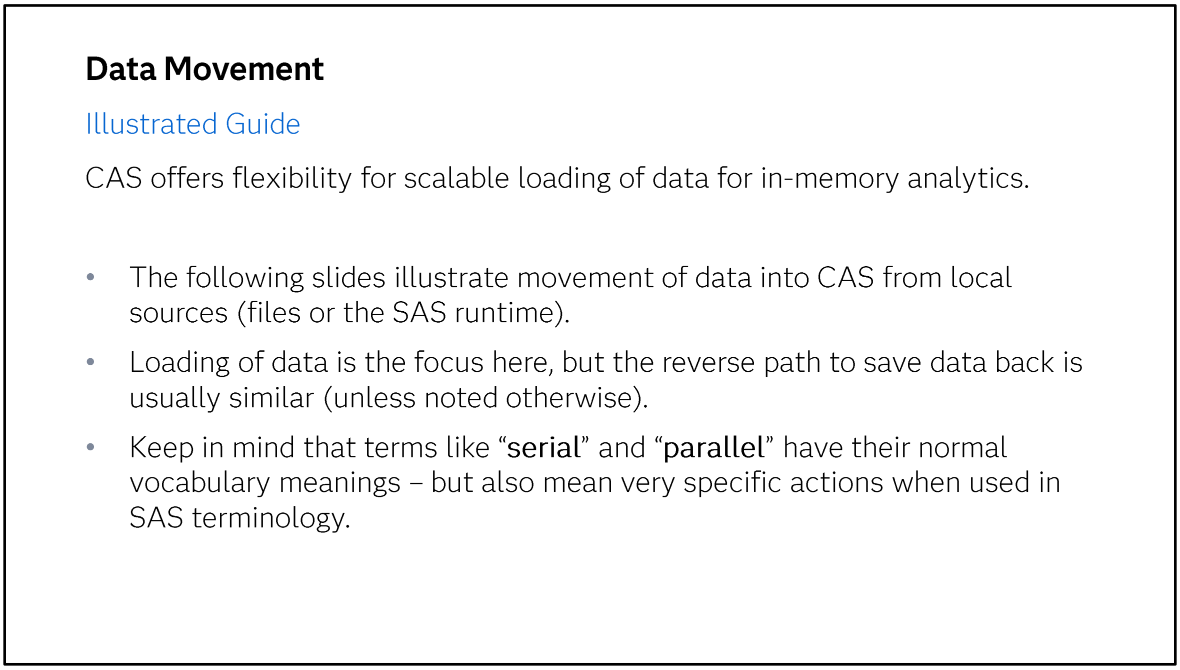
You probably recognize many of these data sources. And this list will continue to grow over time, but let’s call out a few of interest here.

First, note that JDBC and ODBC are supported. So, if the data agent doesn’t support native communication with a data source, then we can still use JDBC or ODBC to reach many not shown here.

Also, one data agent can communicate with and request data directly from another data agent. You can effectively daisy-chain them together if that’s necessitated by the network topology or other rules in your environment. Data agents can also access SAS data sets directly as well as fetch data from the SAS Federation Server, too.



Data Movement Local Files



Data Movement

Illustrated Guide

CAS offers flexibility for scalable loading of data for in-memory analytics.

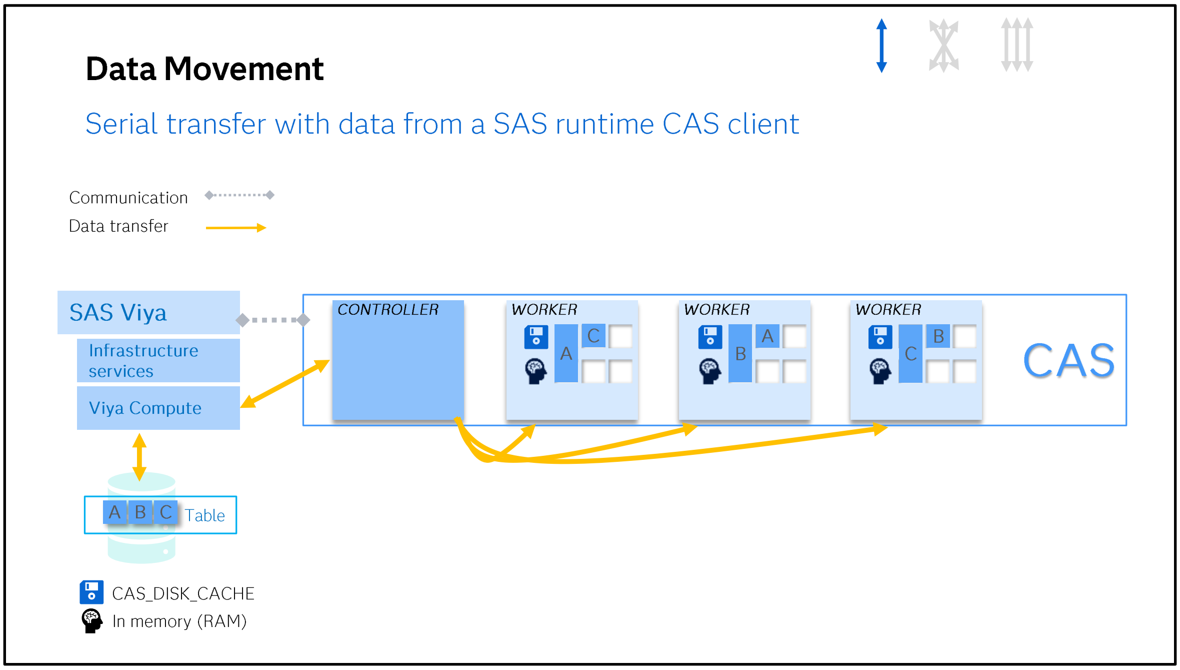
- The following slides illustrate movement of data into CAS from local sources (files or the SAS runtime).
- Loading of data is the focus here, but the reverse path to save data back is usually similar (unless noted otherwise).
- Keep in mind that terms like “**serial**” and “**parallel**” have their normal vocabulary meanings – but also mean very specific actions when used in SAS terminology.

CAS offers flexibility for scalable loading of data for in-memory analytics.

And the following slides will illustrate the movement of data into CAS from local sources like files or an instance of the SAS runtime, like a SAS Compute Server.

Loading of data is the focus here, but the reverse path to save data back is usually similar, unless noted otherwise.

Keep in mind that terms like “serial” and “parallel” have their normal vocabulary meanings. But as we've shown earlier, they can also have very specific actions and interpretations when used in SAS programming terminology.



Data Movement

Serial transfer with data from a SAS runtime CAS client

Communication

Data transfer

SAS Viya

Infrastructure services

Viya Compute

A B C Table

CONTROLLER

WORKER A C

WORKER B A

WORKER C B

CAS

CAS_DISK_CACHE

In memory (RAM)

Let's look at what happens when a SAS runtime, like the SAS Viya Compute Server, transfers data over to CAS.

Shown here, we can see a simplified illustration of a SAS Viya deployment. Notice that we have an MPP CAS server with a controller and three workers. To the left of that we have the rest of SAS Viya summed up in just a couple of small boxes, including the SAS Compute Server. Not shown is the Kubernetes cluster nor the cloud infrastructure, but it's all there, too. Just below the Compute server is a local file with data in it. To keep things simple, that data is represented here as blocks labeled "A", "B", and "C".

Looking back inside the CAS boxes, notice we have some empty slots shown in the workers. The legend at the bottom tells us those slots correspond to space in the host's memory and in CAS disk cache.

Alright – so, the objective here is to get that data table into CAS using the SAS Compute Server.

The Compute server reads that data in using a libref or fileref as needed. From a Kubernetes perspective, the Compute server pod likely accesses that table as a file through a mounted volume from a POSIX-compliant source. That file could be a SAS data set, CSV or other delimited file, or unstructured data. If we look beyond individual files, any source that Compute server might reach could be used as well, like data from a remote source such as an RDBMS.

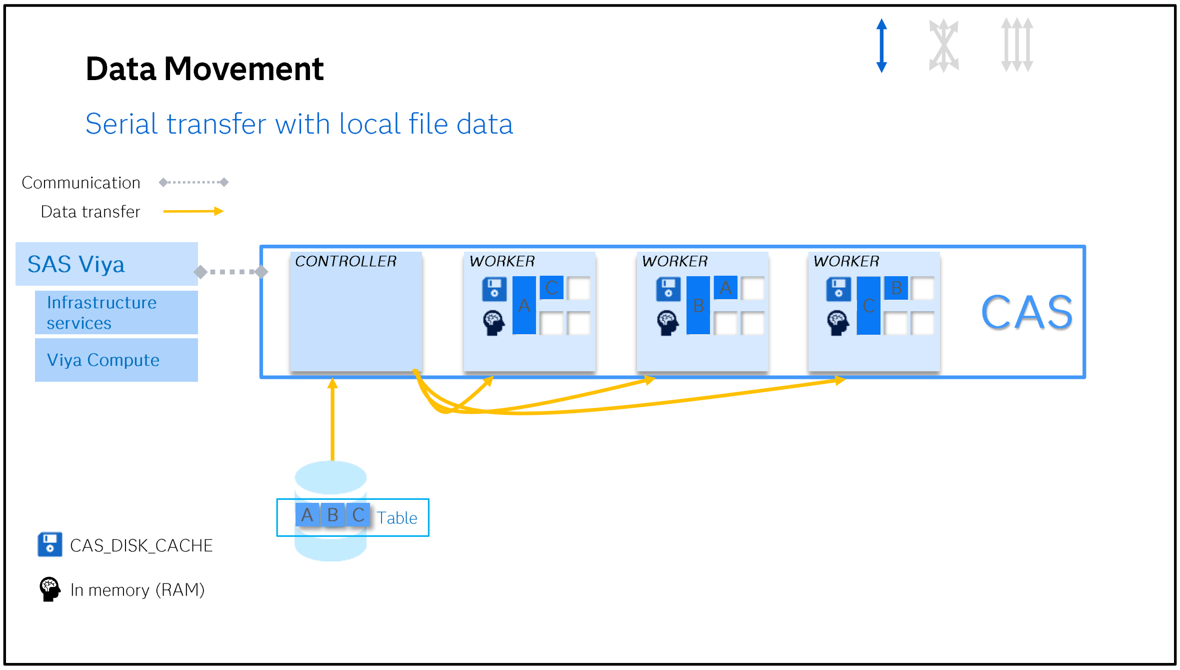
Then, using a DATA step or other technique, the Compute server sends that data to the CAS Controller. This is serial movement of the data because the SAS Compute Server is only talking to the CAS Controller – none of the CAS workers participate here.

Once the CAS Controller has the data, it distributes it evenly across the workers, completing the serial loading process.

Next up is to take a closer look at what the CAS workers actually do with their allotment of the data. Anything CAS keeps in memory must be mapped to a backing store. The backing store in this example is the CAS disk cache. Shown here, the blocks of data have been distributed evenly across the workers – just note in real life that the number of blocks will usually count in the hundreds or thousands or more, not just three. And each block is present in RAM with a memory map defined to a location in that worker's local CAS disk cache. Very simply, the memory map allows the OS to move the data into and out of RAM on demand as a kind of virtual memory scheme.

CAS takes one more step with the data and places an additional copy in the CAS disk cache of the workers for redundancy. Notice that the copied blocks do not duplicate what's already there in RAM. This ensures that if one node goes down, then the remaining two nodes still have a complete set of data for the table.

We won't show all possible variations on this theme, but you could substitute the Compute server with any other SAS programming runtime like a SAS Batch Server, SAS Connect Server, or even a SAS 9 Workspace Server. This same illustration can be extended similarly for other SAS engines, like the Event Stream Processing Server.



Data Movement

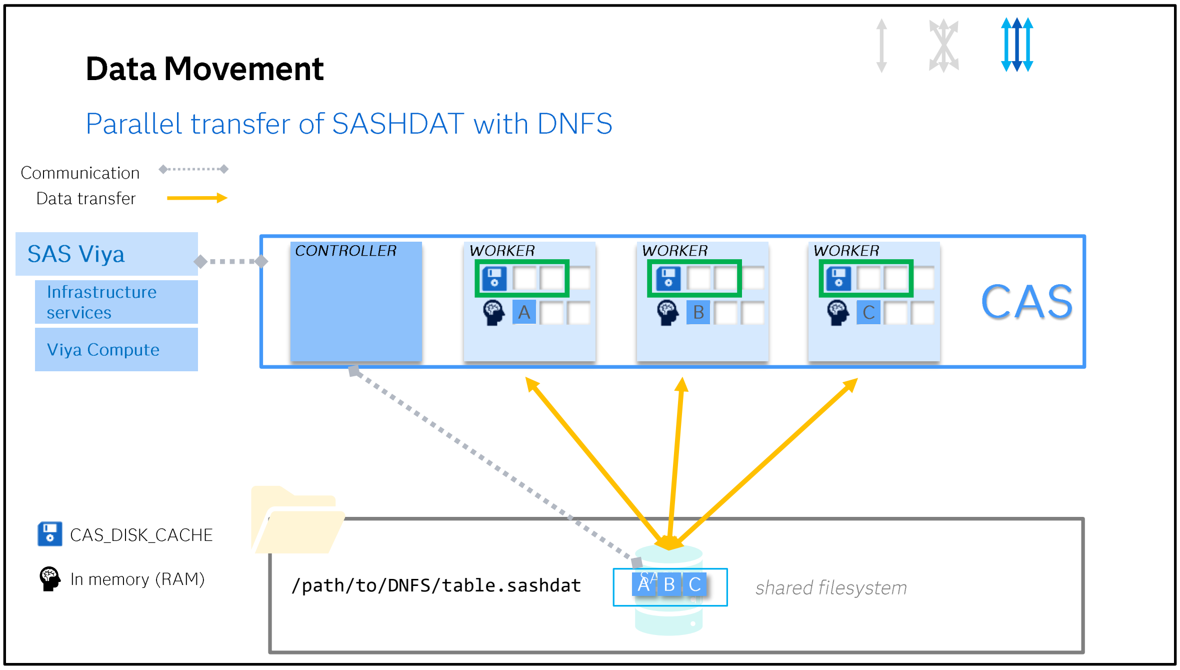
Serial transfer with local file data

This example is slightly different than before. In this case, the CAS Controller has direct access to the source table. And remember, from a Kubernetes perspective, the CAS Controller pod likely accesses that table as a file through a mounted volume from a POSIX-compliant source or block storage. That file could be a SAS data set, CSV or other delimited file, SASHDAT, audio, video, document formats or anything else that CAS knows how to work with.

The goal here is for CAS to load that data in itself.

As illustrated here, only the CAS Controller can access this file, so it reads in the data. This is serial movement again because only the Controller is accessing the file.

Once the CAS Controller has the data, it distributes it evenly across the workers, completing the serial loading process. As before, data loaded into CAS memory is mapped to CAS disk cache as the backing store. And redundant blocks are distributed for protection against the loss of a CAS worker.



Data Movement

Parallel transfer of SASHDAT with DNFS

This example is a little bit more fun where we can see parallel movement of data.

In this case, we have a shared file system that's mounted as an RWX (or ROX) volume to all the CAS pods at the same local path. And here, the data is stored in a SASHDAT file so that we can use the DNFS type of caslib to achieve parallel loading.

The goal here is for all the CAS workers to participate in loading that data.

When directed to load the data, CAS first looks at the SASHDAT metadata stored in the file header to get a clear understanding of the structure and volume of the data. This helps ensure a very efficient load with no guesswork.

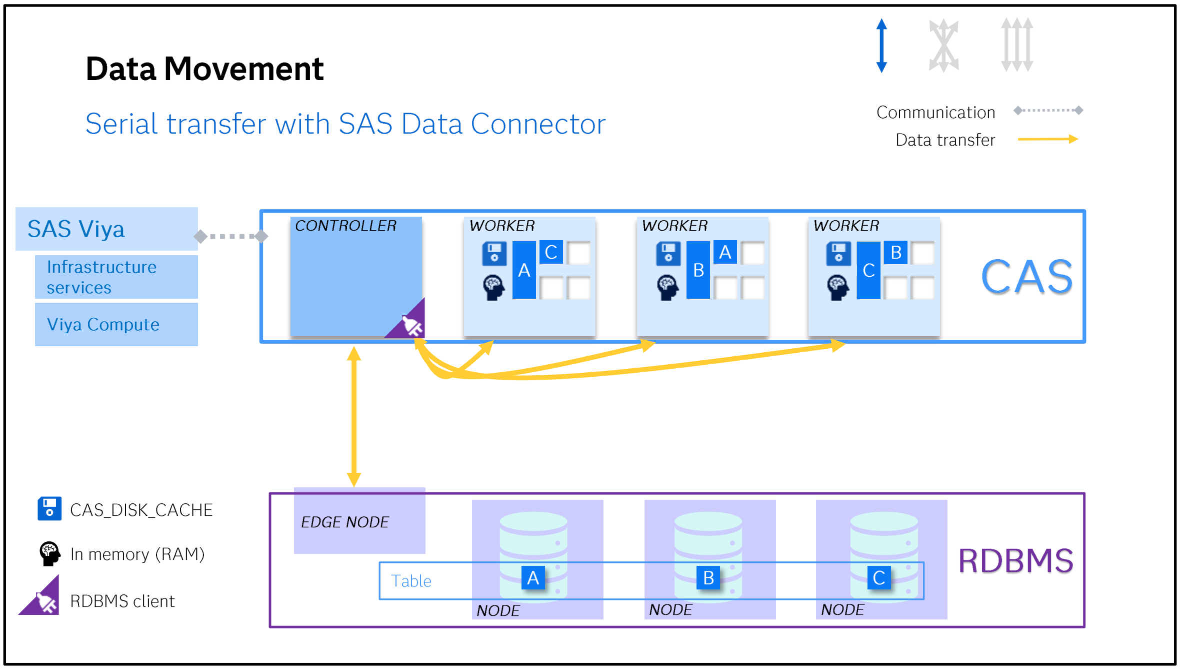
The CAS Controller then directs the CAS workers to find and load their allotment of data from the source. We call this a parallel load because all the workers are doing their part to get the data. However, understand it's possible there's a serial bottleneck at some point along the way. For example, a basic NFS mount to the file without any kind of RAID redundancy or other parallel-supporting infrastructure is effectively just a serial transfer at the lowest level (even if CAS does not perceive it that way).

Notice that the behavior with the data in memory is different here, too. This data is not mapped to CAS disk cache. Nor are there any replicated blocks for failover. Why is that?

Because we're demonstrating the DNFS type of caslib, then that means our source data is already SASHDAT. When we tell CAS to load data from this source, it doesn't actually copy the data over. Instead, it creates a memory map directly to the SASHDAT source and reports the load is complete. This is called a "lazy load" because CAS hasn't actually loaded the data yet. The first time that data is referenced by a job, CAS will then access the memory location to get it. This is where the memory map comes in. The OS is responsible for handling the memory map activity and that's when the data physically transfers from the source on disk to the RAM in the CAS workers.

Remember earlier when I said, “Anything CAS keeps in memory must be mapped to a backing store”? In this example, the backing store is not CAS disk cache, but instead the original SASHDAT source on disk. That’s also why there are no replicated blocks stored in CAS disk cache – because all CAS workers already have direct access to the source data and so if one or more CAS workers goes offline, the remaining workers can still access the complete table. The DNFS source provides the expected redundancy.

Data Movement SAS Data Connector



Now we'll discuss what happens when we direct CAS to use its data connector technology to serially load data from a remote source.

Shown here, we can see our familiar simplified illustration of a SAS Viya deployment with an MPP CAS server sporting a controller and three workers. Below that is an external data source, shown as a large deployment of an RDBMS with multiple nodes and an edge node for communication.

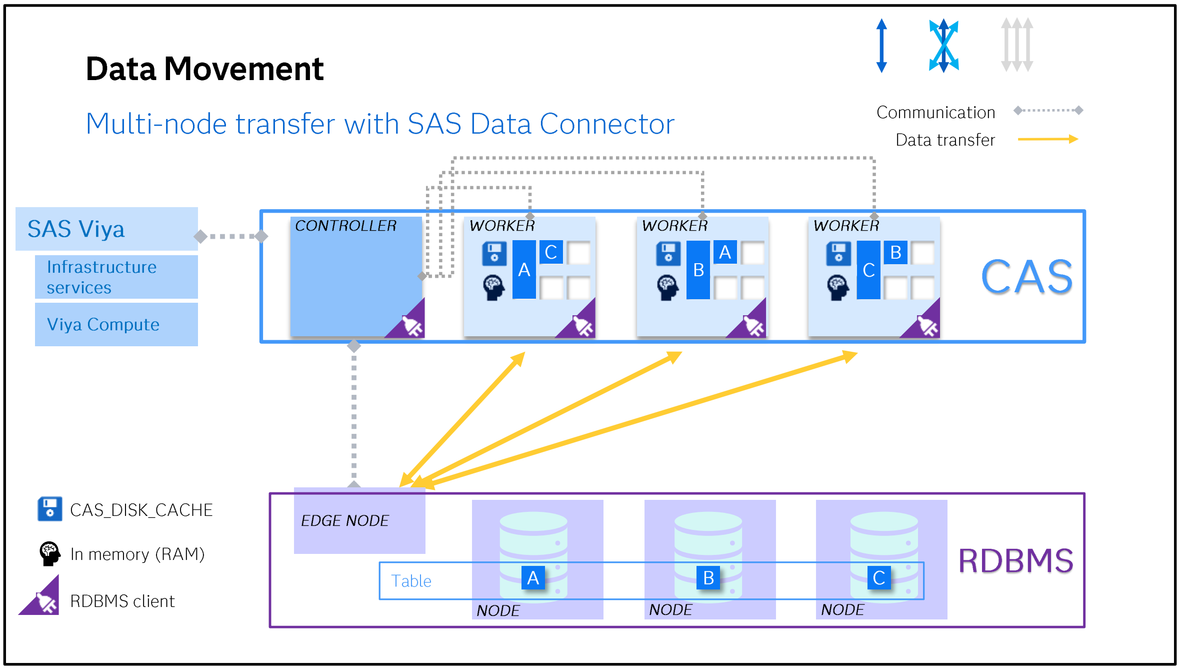
The data we want in the RDBMS shown here is represented again as blocks labeled “A”, ”B”, and “C”. In real life, of course, the actual number of blocks would be much higher, but this is just a simple illustration.

The objective is for CAS to read the data from the RDBMS. To do that, the SAS Data Connectors use the same approach that you might already be familiar with from SAS 9 ACCESS engines – a dedicated RDBMS client is deployed for use by the CAS controller. That's represented here with the purple plug icon on the CAS controller. From a Kubernetes perspective, this client software is likely placed in a volume that is already configured, tested, and mounted to the CAS controller pod.

With all these pieces in place, let's get that data.

As we've seen before for serial transfer, the CAS controller will reach out to the source and request the data. Once the data has come across, then the CAS controller will distribute the data evenly across the CAS workers.

Because the source data is not in native SASHDAT format, then the data in memory will map to use the CAS disk cache as a backing store. And replicate blocks are created in case of CAS worker failure.



Data Movement

Multi-node transfer with SAS Data Connector

In this next slide, we’ll look at multi-node transfer using the SAS Data Connectors.

To accomplish a multi-node transfer of data to CAS, then the CAS workers must be able to participate in communicating with the data source.

So, first of all, we need to get the the RDBMS client software hooked up on the CAS workers just as it was for the CAS controller. Notice that we now have purple plug icons on all CAS hosts. From a Kubernetes perspective, this client software is likely placed in a volume that is already configured, tested, and mounted to identical mount points on all CAS pods.

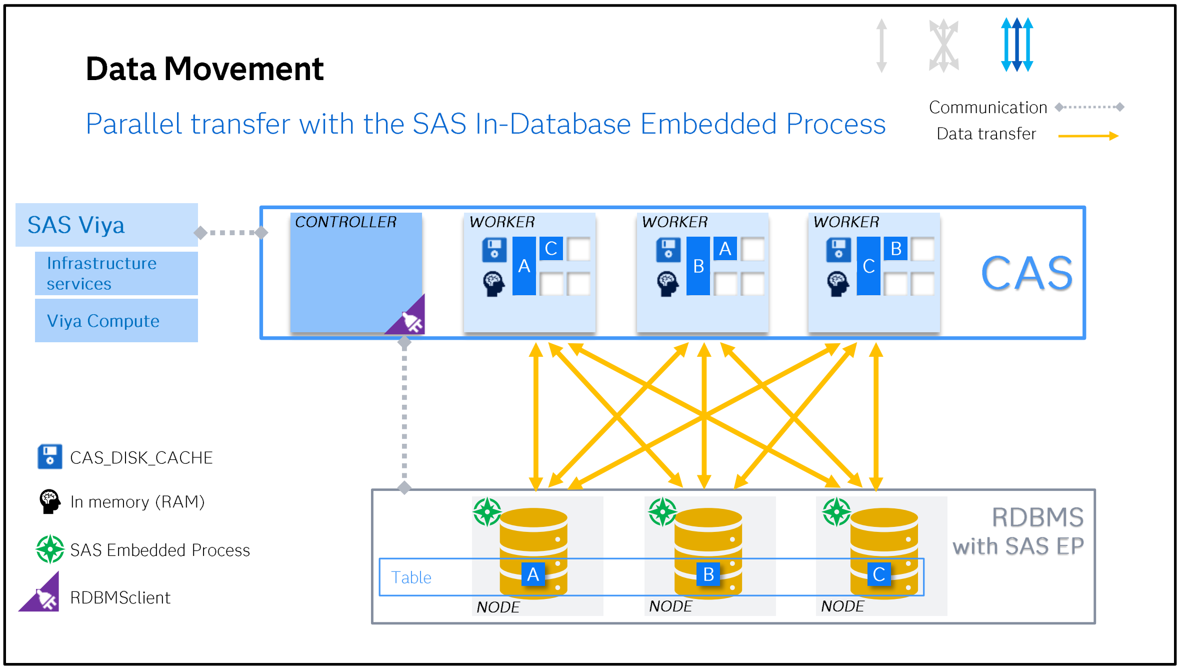
When directed to load the data, the CAS controller reaches out to the RDBMS to get an idea of the size and shape of the data. Then it informs the CAS workers as to the queries each should use to pull over its allotment of data.

Each CAS worker then performs its own query to get its assigned rows of data from the source.

As shown here, this RDMS provides only a single edge node. However, some RDBMS can provide multiple edge nodes to to help parallelize and distribute the large-scale movement of data across multiple IO channels. This helps explain why we use the term “multi-node” instead of “parallel” for this kind of scenario. For CAS, we’ll definitely try to use multiple nodes, however, the data source may or may not have sufficient infrastructure in place to perform fully parallel movement of the data. The term “multi-node” is meant to capture this potential variability.

Finally, CAS is resilient in its use of multi-node data transfer so that it can complete data movement even if one or more CAS workers is unable to participate in the transfer.

Data Movement Data Connect Accelerator



Data Movement

Parallel transfer with the SAS In-Database Embedded Process

Communication

Data transfer

SAS Viya

Infrastructure services

Viya Compute

CONTROLLER

WORKER

A C

WORKER

B A

WORKER

C B

CAS

CAS_DISK_CACHE

In memory (RAM)

SAS Embedded Process

RDBMSclient

Table

A

B

C

NODE

NODE

NODE

RDBMS with SAS EP

Now we'll discuss what happens when we direct CAS to use its data connect accelerator technology to parallel load data from a remote source where the SAS In-Database Embedded Process has been deployed.

Shown here, we again see our familiar simplified illustration of a SAS Viya deployment with an MPP CAS server sporting a controller and three workers. Below that is an external data source, shown as a large RDBMS that's compatible with SAS In-Database technology for Viya.

As established earlier, in order for CAS to communicate with a remote data provider, it will use an RDBMS client - shown here with the purple plug icon on the CAS controller – that we know from a Kubernetes perspective, has been placed in a volume that is already configured, tested, and mounted to the CAS controller pod.

We also want to show that the SAS Embedded Process has been deployed to the remote data source, typically running an instance on each node of that cluster.

The data we want in the RDBMS shown here is represented again as blocks labeled "A", "B", and "C". In real life, of course, the actual number of blocks would be much higher, but we're keeping it simple here.

Now that we know where everything is, let's get that data loaded into CAS.

Things work a little different here than we've seen earlier. In this case, CAS reaches out to the Embedded Process and requests the desired data. It also gives the EP a list of its worker nodes' hostnames as the destination for the requested table.

Now something happens that we haven't seen before. The EP instances on each node initiate new network connections over to the CAS workers to send the data across. In other transfers, we've seen CAS as the network connection originator with the data source simply replying back along the same communication channel – think of like CAS is "pulling" the data over. But as we're showing here, the EP instances are acting as the network connection originator to “push” the data over to the CAS workers. This has implications from a network management perspective beyond the scope of our discussion here, but useful to understand in general.

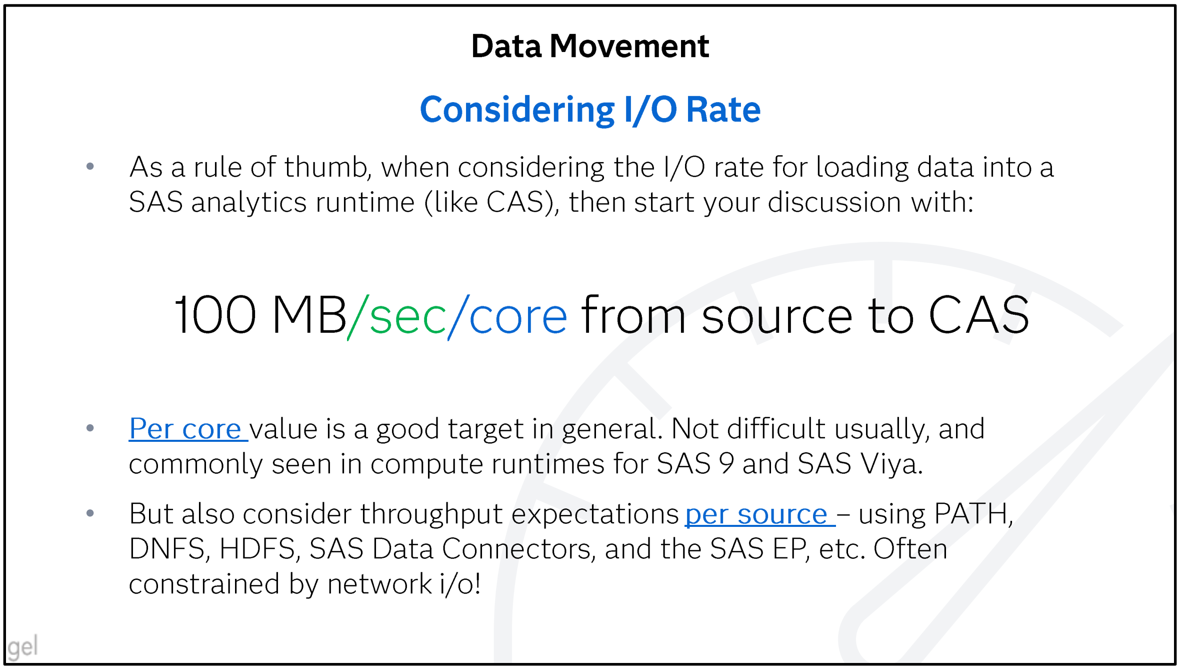
Also, let’s point out that while this simple diagram shows three CAS workers and three RDBMS nodes, there’s no expectation that the number of nodes must match in any way.

We can also look at little closer at the data movement and see that each EP is actually distributing the data it has access to over to all the CAS workers. This massively parallel movement of the data can provide very fast loads assuming the network infrastructure is up to the challenge. Some sites will prefer to implement a dedicated network fabric for very large-scale movement of data to separate the traffic and avoid over-saturating the primary network.

In general, because the source data is not in native SASHDAT format, then the data in memory will map to use the CAS disk cache as a backing store. And replicate blocks are created in case of CAS worker failure.

From SAS’ perspective, parallel loading implies that all the CAS workers are participating in the data movement. If any one of them cannot, then the parallel transfer will fail. Specify the appropriate caslib and importOptions to define a possible fallback methodology if desired.

Data Movement Throughput Rate



Data Movement

Considering I/O Rate

- As a rule of thumb, when considering the I/O rate for loading data into a SAS analytics runtime (like CAS), then start your discussion with:

100 MB/sec/core from source to CAS

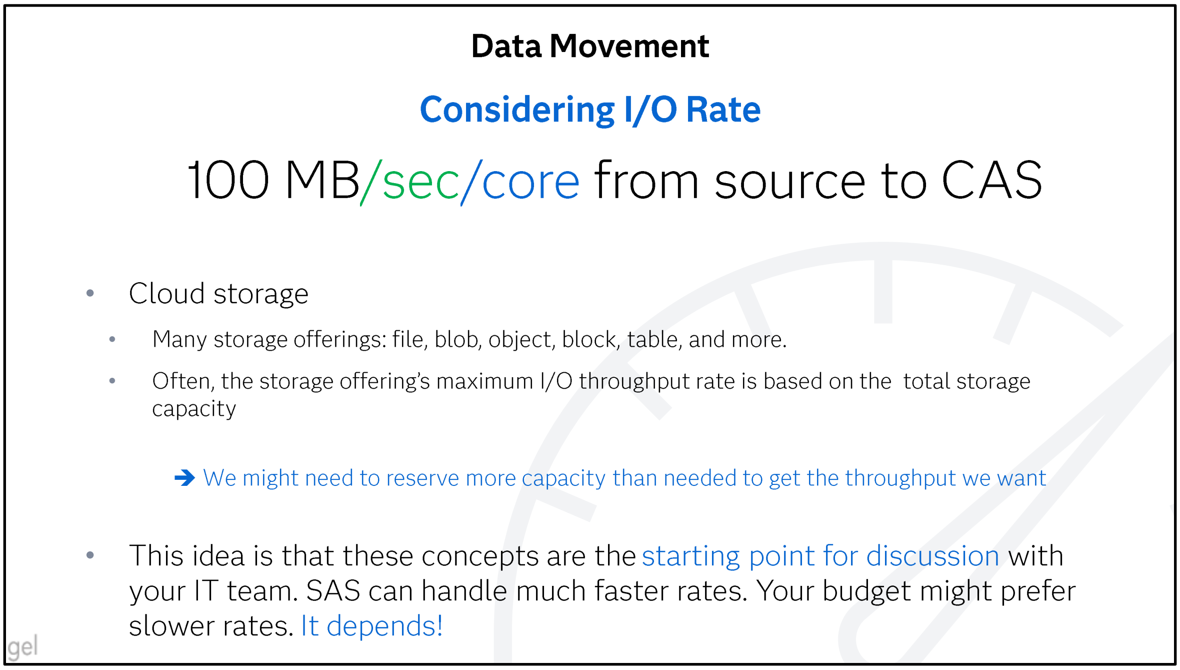
- Per core value is a good target in general. Not difficult usually, and commonly seen in compute runtimes for SAS 9 and SAS Viya.
- But also consider throughput expectations per source – using PATH, DNFS, HDFS, SAS Data Connectors, and the SAS EP, etc. Often constrained by network i/o!

When it comes to moving data around into and out of CAS or other SAS Viya runtimes, the throughput rate for our IO is very important.

As a rule of thumb, when considering the IO rate for loading data into a SAS Analytics runtime like CAS, then we should plan on starting our discussions with a speed around 100 megabytes per second per core from the source to CAS.

The core count is based on the host or hosts where CAS is running. Per core as a value is a good target in general. Typically, it's not too difficult these days and is a requirement that we’ve historically targeted for compute runtimes in SAS 7, SAS 8, and SAS 9 as well as SAS Viya.

We also must consider the throughput expectations per source. So, when we're using the PATH, DNFS, HDFS, data connectors, and the SAS embedded process, we are often constrained by network IO. Each of these can rely on different IO routes that must be planned for to ensure good movement to satisfy user expectations and service level agreements.



Data Movement

Considering I/O Rate

100 MB/sec/core from source to CAS

- Cloud storage
  - Many storage offerings: file, blob, object, block, table, and more.
  - Often, the storage offering's maximum I/O throughput rate is based on the total storage capacity

➔ We might need to reserve more capacity than needed to get the throughput we want

- This idea is that these concepts are the starting point for discussion with your IT team. SAS can handle much faster rates. Your budget might prefer slower rates. It depends!

gel

100 megabytes per second per core from the source to CAS is something that we need to ensure is possible from cloud providers and their various storage offerings. There are many ways to get data in to (and out of) CAS and the other SAS analytic runtime engines. Direct-attached storage is great, of course, but many other storage offerings are delivered over a network connection. Whichever route is taken, we need to ensure sufficient throughput for SAS to work efficiently as intended and not sitting idle waiting while the data trickles in.

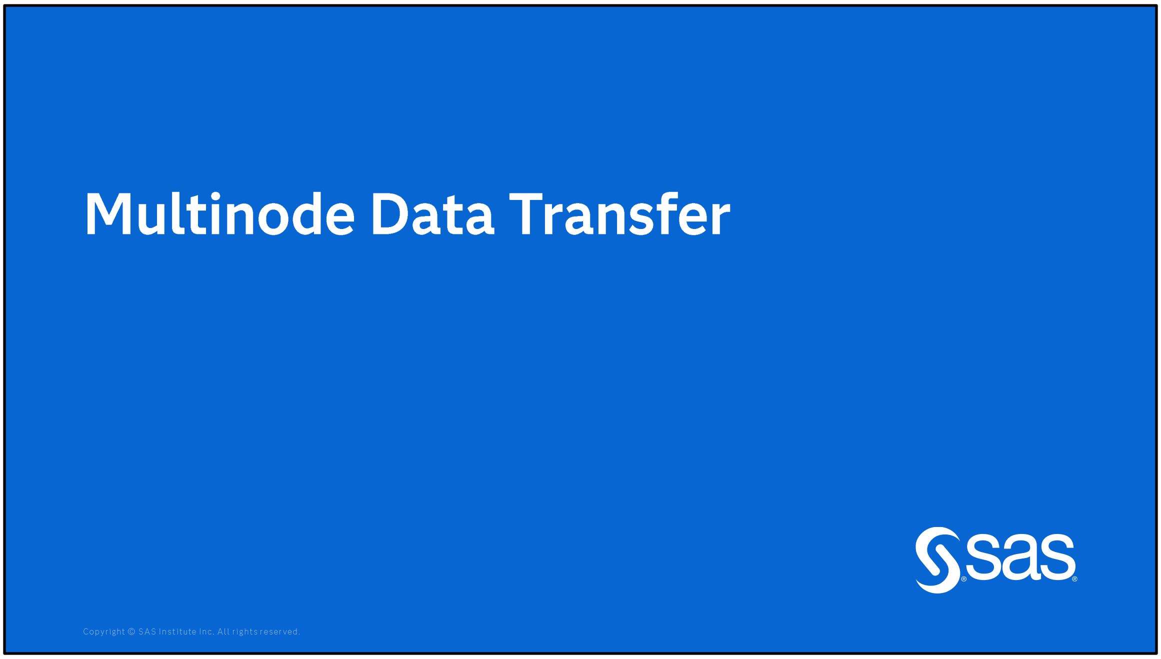
Storage in the cloud takes many forms. There's file storage, blob and object storage, block storage, storage for tables, and more.

For many cloud providers, we often see the storage offering's IO throughput rate is based on total storage capacity.

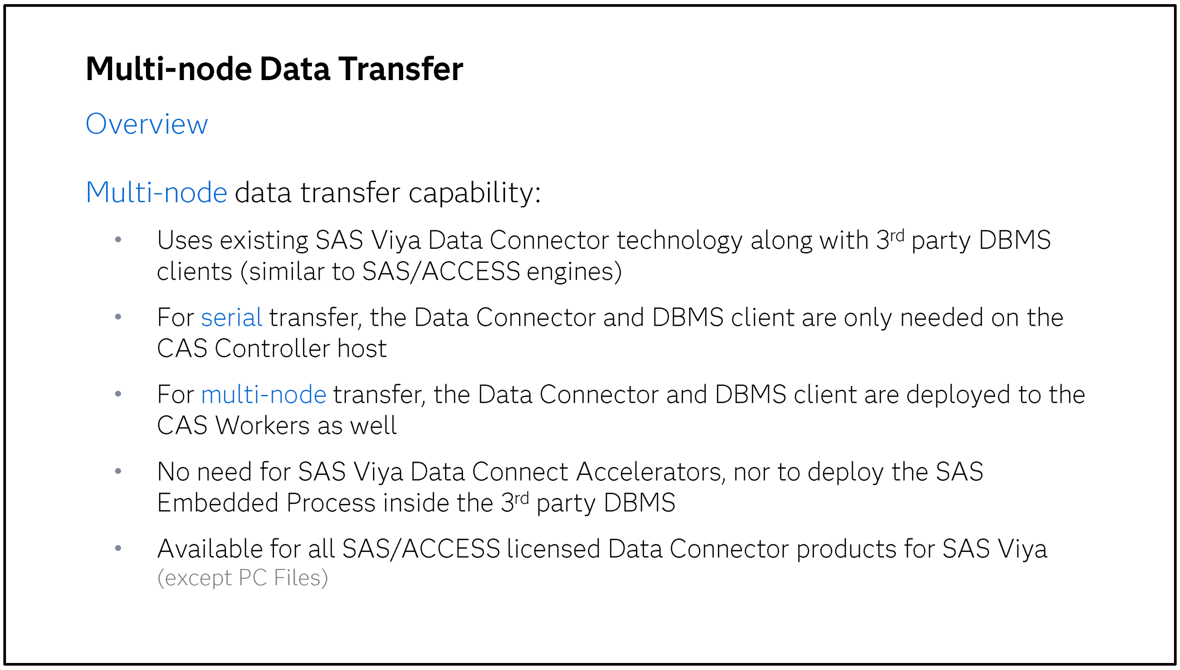
This means that a site may need to provision more total storage volume than they plan to use in order to achieve the IO throughput rate that's needed to meet performance targets.

The idea with these concepts is they are a starting point for discussion with your IT team. SAS analytic engines can handle much, much faster I/O throughput rates. 100 megabytes per second per core is not a ceiling value. It's really a floor value. On the other hand, your budget might prefer slower rates depending on what your objectives are.

So, what's your target I/O throughput rate with storage? Ultimately, the answer is, "It depends." The main takeaway should be that it's important, it matters, and it must be planned for appropriately for the data volumes and overall workload.



Multinode Data Transfer



Multi-node Data Transfer

Overview

Multi-node data transfer capability:

- Uses existing SAS Viya Data Connector technology along with 3rd party DBMS clients (similar to SAS/ACCESS engines)
- For serial transfer, the Data Connector and DBMS client are only needed on the CAS Controller host
- For multi-node transfer, the Data Connector and DBMS client are deployed to the CAS Workers as well
- No need for SAS Viya Data Connect Accelerators, nor to deploy the SAS Embedded Process inside the 3rd party DBMS
- Available for all SAS/ACCESS licensed Data Connector products for SAS Viya (except PC Files)

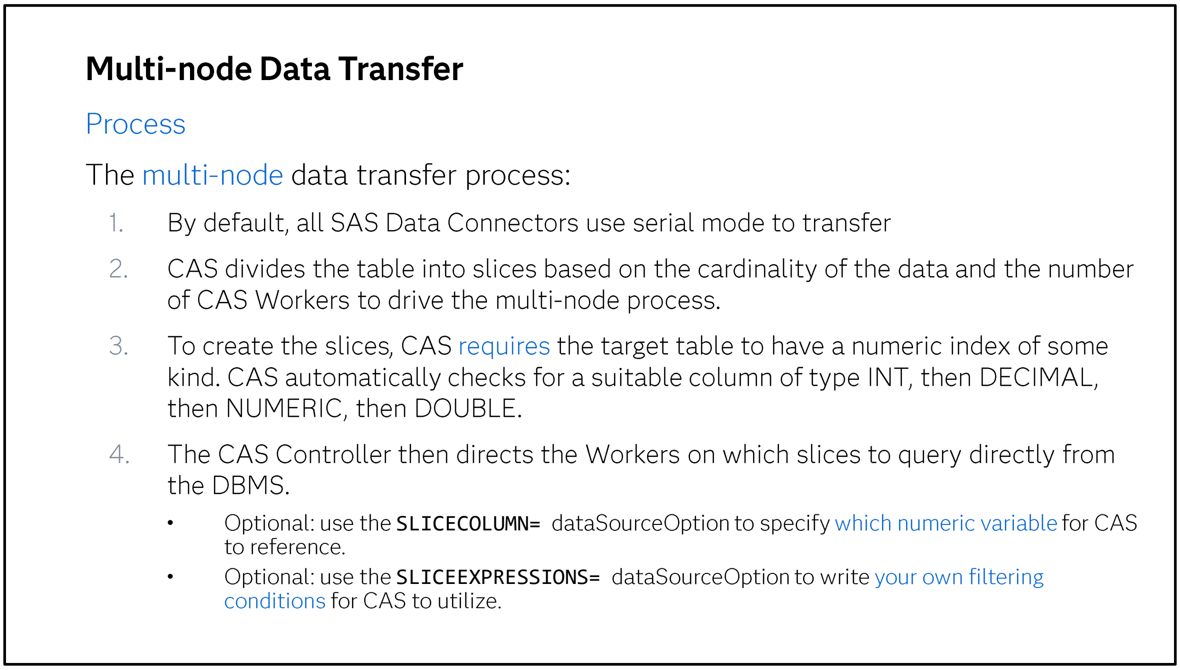
The multi-node data transfer capability uses existing SAS Viya data connector technology along with a third-party database management system client, similar to what we have experienced with SAS/ACCESS engines in the past.

For serial movement of the data, the data connector and the DBMS client are only needed on the CAS controller host.

For multi-node transfer, the data connector and DBMS client are deployed to the CAS workers as well.

There's no need for SAS Viya data connect accelerators at this point, nor do we need to deploy the SAS In-Database Embedded Process inside the third-party data provider.

Multi-node data transfer capability is available for all SAS/ACCESS licensed data connector products for SAS Viya, with the notable exception being PC files, because a data connector with multi-node transfer doesn't really make sense in that case.



Multi-node Data Transfer

Process

The multi-node data transfer process:

1. By default, all SAS Data Connectors use serial mode to transfer
2. CAS divides the table into slices based on the cardinality of the data and the number of CAS Workers to drive the multi-node process.
3. To create the slices, CAS requires the target table to have a numeric index of some kind. CAS automatically checks for a suitable column of type INT, then DECIMAL, then NUMERIC, then DOUBLE.
4. The CAS Controller then directs the Workers on which slices to query directly from the DBMS.
   - Optional: use the **`SLICECOLUMN=`** dataSourceOption to specify which numeric variable for CAS to reference.
   - Optional: use the **`SLICEEXPRESSIONS=`** dataSourceOption to write your own filtering conditions for CAS to utilize.

The multi-node data transfer process takes several steps.

Number one, by default, the SAS data connectors use serial mode to transfer data. That is, that's where the CAS controller will get the data from the source and then distribute it across all the workers evenly.

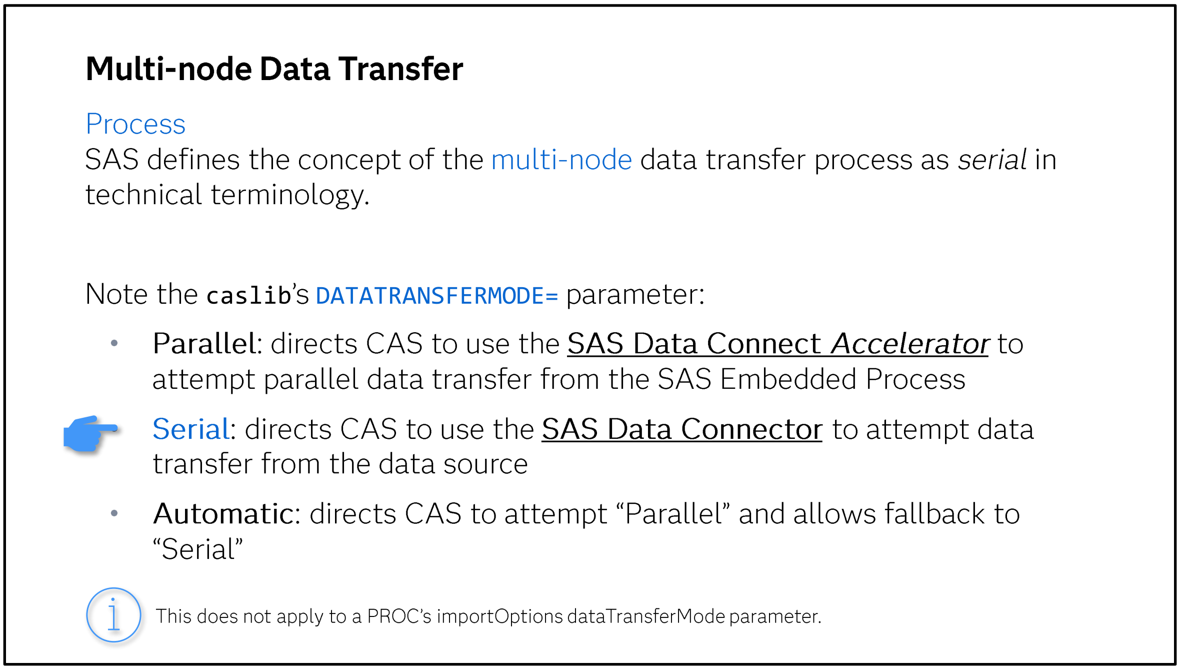
When we have multi-node data transfer, CAS will divide the table into slices based on the cardinality of the data and the number of CAS workers to drive the multi-node process.

To create those slices, CAS requires the target table to have a numeric index of some kind. CAS automatically checks for a suitable column of type int, then decimal, then numeric, and then double.

The CAS controller will then direct the workers on which slices they are each to query directly from the remote data provider. We also have some additional options that we can specify instead of allowing CAS to decide for itself how to slice up the data.

We have the option to specify the slice column as a data source option, and that tells CAS which numeric variable we want it to reference so it doesn't pick the wrong one and get poor slice decisions.

We can also optionally specify slice expressions as a data source option. That allows you to write your own filtering conditions for CAS to utilize so that you can make the process even more efficient if necessary.



Multi-node Data Transfer

Process

SAS defines the concept of the multi-node data transfer process as *serial* in technical terminology.

Note the `caslib`’s `DATATRANSFERMODE=` parameter:

- Parallel: directs CAS to use the SAS Data Connect *Accelerator* to attempt parallel data transfer from the SAS Embedded Process
- Serial: directs CAS to use the SAS Data Connector to attempt data transfer from the data source
- Automatic: directs CAS to attempt “Parallel” and allows fallback to “Serial”

This does not apply to a PROC’s importOptions dataTransferMode parameter.

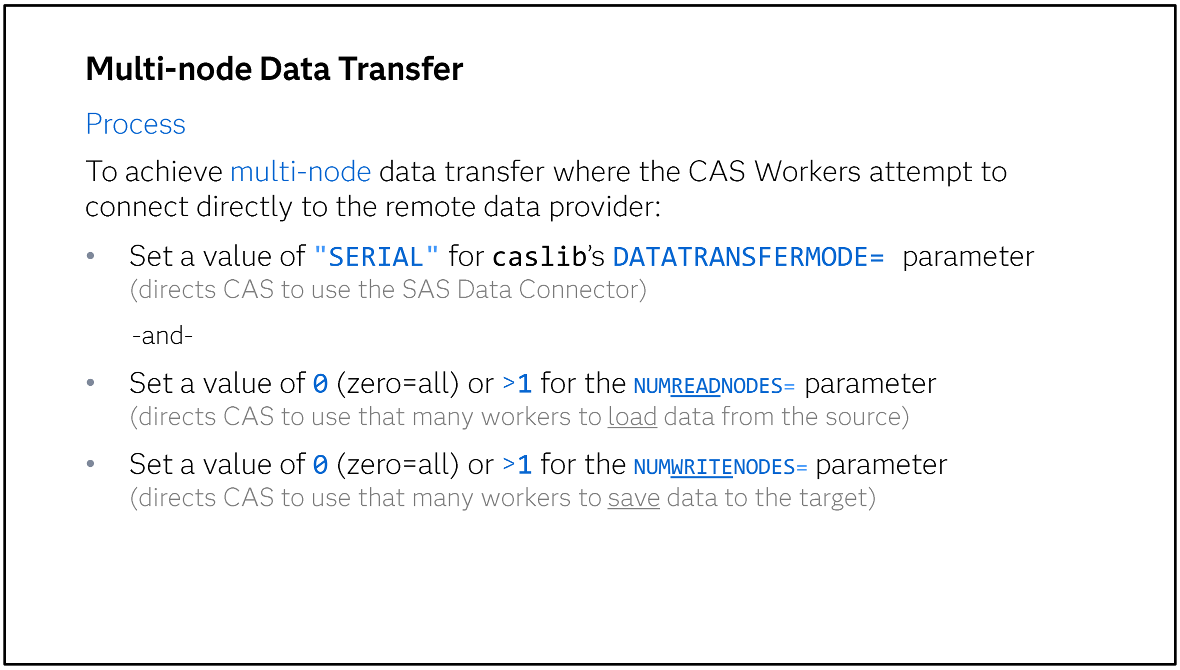
SAS defines the concept of multi-node data transfer process as a “serial” operation in our technical terminology. That is, the data transfer mode option in a caslib statement doesn’t define the movement of data directly. Instead, it’s meant to tell CAS which technology it should use to get to the data.

When we are referring to that data transfer mode parameter on the caslib statement, the value of “parallel” directs CAS to use the SAS Data Connect Accelerator technology to attempt parallel data transfer from the SAS In-Database Embedded Process. That's not something we are interested in here for multi-node data transfer.

The “serial” value is what directs CAS to use its data connector technology to attempt data transfer from the source. Data connectors can move the data in serial fashion as well as multi-node.

We also have a value of “automatic” that directs CAS to attempt parallel – that is, use the Data Connect Accelerator – and if for some reason that doesn't work, then it will fall back to serial, which is directing CAS to use its data connector technology to get to the data.

Again, we've covered this earlier, but just want to point out that the data transfer mode parameter discussed here is for the caslib statement and does not apply to the data transfer mode parameter that's specified as an import- or export-option for SAS procedures like PROC CASUTIL.



Multi-node Data Transfer

Process

To achieve multi-node data transfer where the CAS Workers attempt to connect directly to the remote data provider:

- Set a value of `"SERIAL"` for `caslib`'s `DATATRANSFERMODE=` parameter
  (directs CAS to use the SAS Data Connector)

  -and-

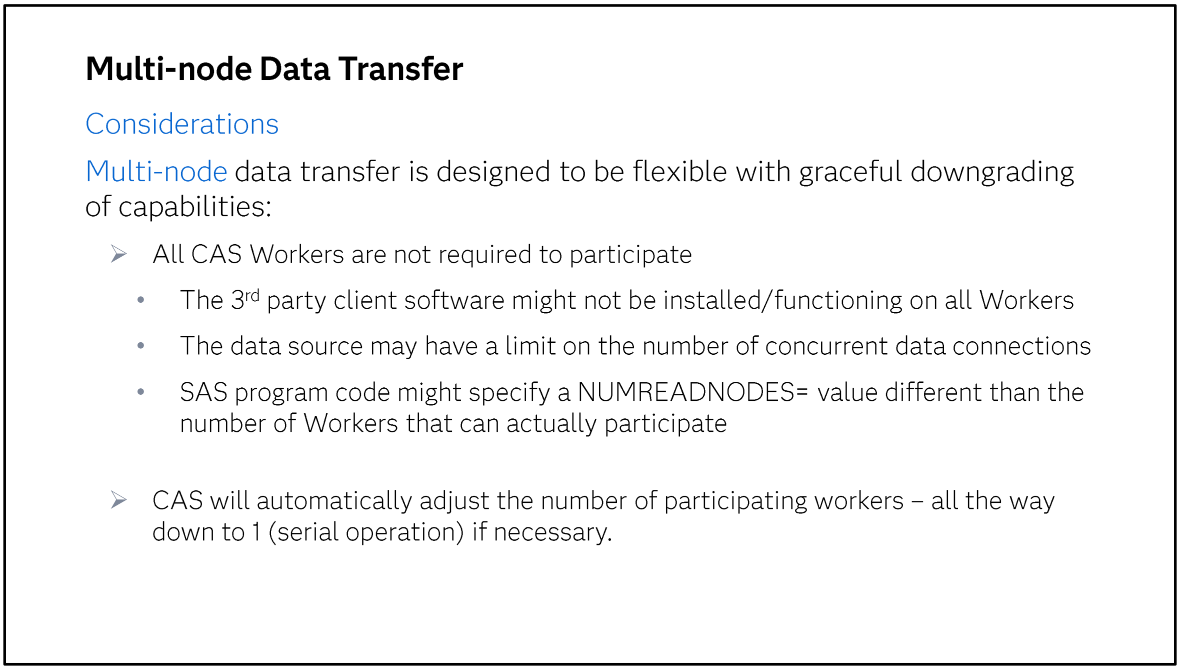
- Set a value of `0` (zero=all) or `>1` for the `NUMREADNODES=` parameter
  (directs CAS to use that many workers to load data from the source)
- Set a value of `0` (zero=all) or `>1` for the `NUMWRITENODES=` parameter
  (directs CAS to use that many workers to save data to the target)

To achieve multi-node data transfer where the CAS workers attempt to connect directly to the remote data provider, we usually need to specify a few parameters.

First is to set a value of serial for the caslib's data transfer mode parameter so that CAS will use its data connector technology to get at the data.

Then we need to specify the numReadNodes parameter on the caslib statement. The default is usually "1", so we can set numReadNodes to zero to tell CAS to employ all its workers in reading the data from source. Alternatively, we can specify an integer values greater than one to indicate a specific number of CAS workers to engage.

And if we want to write the data back to source using multi-node transfer, we can use the same values for the numWriteNodes parameter on the caslib statement.



Multi-node Data Transfer

Considerations

Multi-node data transfer is designed to be flexible with graceful downgrading of capabilities:

- All CAS Workers are not required to participate
  - The 3rd party client software might not be installed/functioning on all Workers
  - The data source may have a limit on the number of concurrent data connections
  - SAS program code might specify a NUMREADNODES= value different than the number of Workers that can actually participate
- CAS will automatically adjust the number of participating workers – all the way down to 1 (serial operation) if necessary.

Multi-node data transfer is designed to be flexible with graceful downgrading of capabilities.

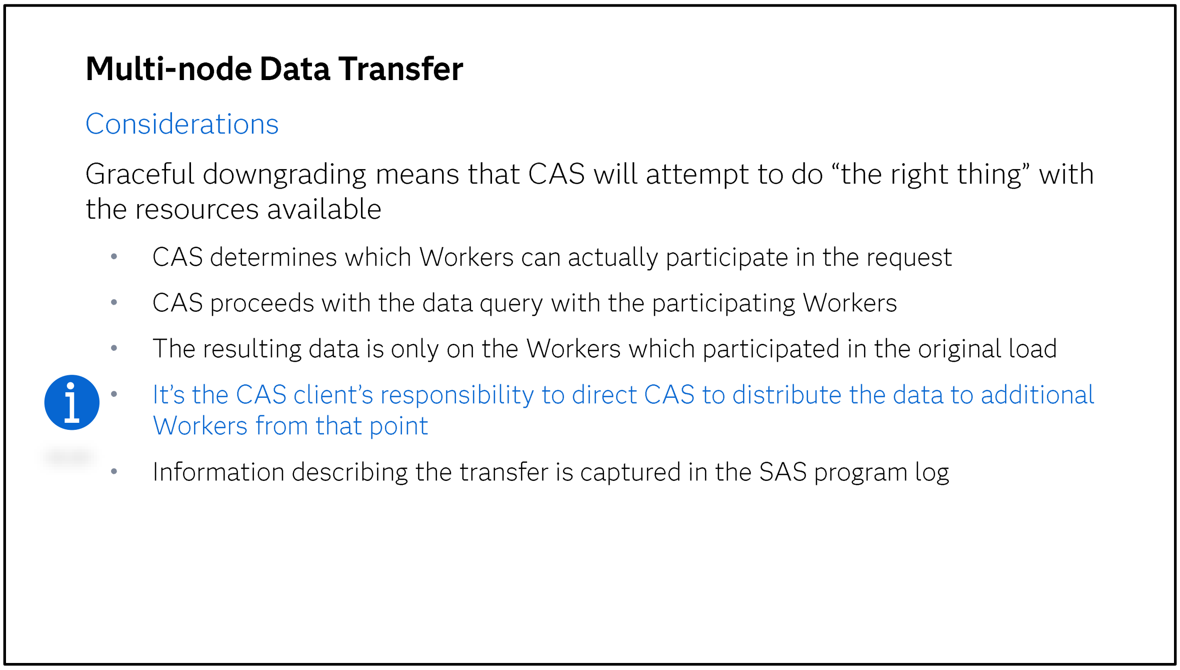
For one thing, all CAS workers are not required to participate in a multi-node data transfer. There are reasons why this might be, including:

The third-party client software that’s necessary for a given data source might not be installed or functioning on all CAS workers. If only half your workers have the RDBMS client installed, then they will be the only ones that can participate in multi-node transfer.

Another reason why all the CAS workers might not participate is that the data source itself may have a limit on the number of concurrent connections it will allow. This is implemented on some systems so that they don't get overloaded with too much work, or there might be other licensing constraints or limits.

And noted a moment ago, SAS program code might specify an integer greater than one for the numReadNodes parameter. And if that number is different than the total number of workers that CAS has, then it will limit to that number of participant workers.

CAS will automatically adjust the number of participating workers all the way down to one for serial operation through the CAS controller if necessary. So, even if you specify a number of workers that's too many as mentioned in the examples above, CAS is flexible to automatically adjust to fewer workers to get the job done.



Multi-node Data Transfer

Considerations

Graceful downgrading means that CAS will attempt to do “the right thing” with the resources available

- CAS determines which Workers can actually participate in the request
- CAS proceeds with the data query with the participating Workers
- The resulting data is only on the Workers which participated in the original load
- It’s the CAS client’s responsibility to direct CAS to distribute the data to additional Workers from that point
- Information describing the transfer is captured in the SAS program log

Graceful downgrading means that CAS will attempt to do the right thing with the resources that are available to it.

First, CAS determines which workers can actually participate in the request.

Then CAS proceeds with the data query using those workers.

The resulting data is only on the workers which participated in the original load.

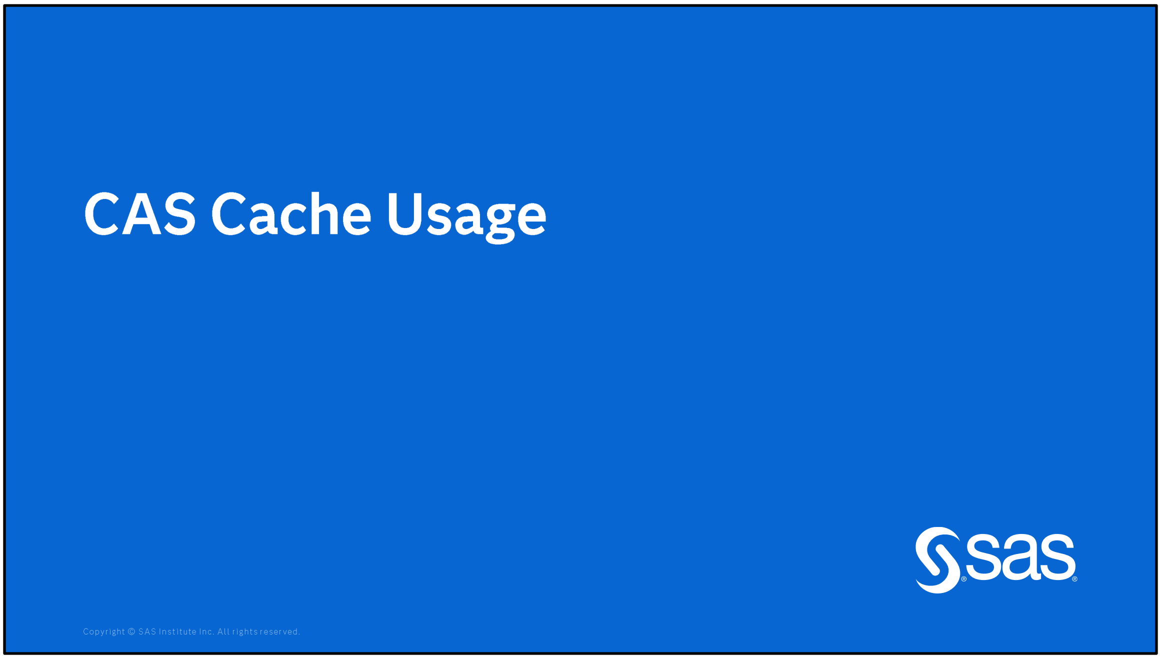
It's then the CAS client's responsibility to direct CAS to distribute the data to additional workers from that point. Let’s dig into this a little bit.

A CAS client is whatever is talking to CAS. That could be some SAS program code you’ve written that’s running in a SAS Compute Server. Or it might be some Python code, R code, Java code, or other open-source programming you’ve employed. There are other CAS clients as well. The SAS Visual Analytics offering is a point-and-click UI that is a CAS client. The list goes on.

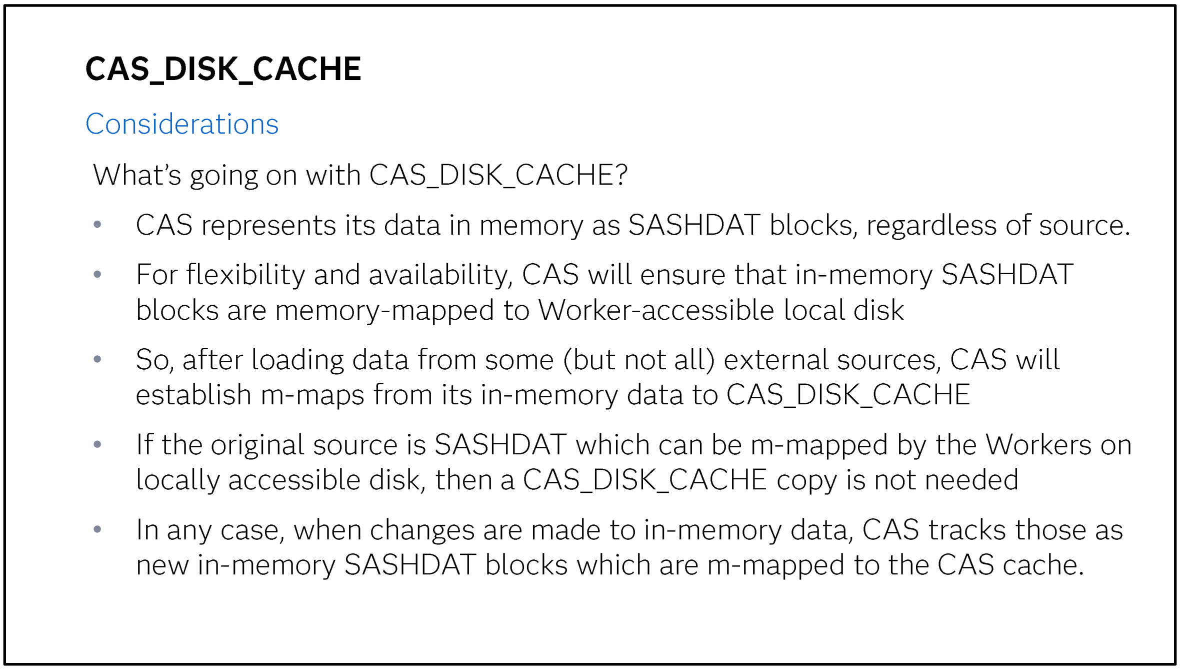
Imagine a scenario where you’ve got MPP CAS running with six workers and your SAS program code employed multi-node transfer to load data into CAS. For some reason, only three CAS workers participated in the load. Those three have all the data shared between them with none on the other three workers that were left out. If you want that data distributed across all six CAS workers in this situation, then you need to submit some additional code telling CAS to do that.

Now, instead of your own SAS program code, imagine you’re using a point-and-click user interface like SAS Visual Analytics or SAS Environment Manager. That is, offerings which don’t provide direct access to their SAS code when it runs behind the scenes. If, for some reason, the full data is only loaded to a subset of CAS workers, then CAS won’t just automatically rebalance the data across all its workers. And if the visual interface software doesn’t detect that, then you might need to take additional steps manually if you want it done.

Once the data has been transferred, information that describes that transfer can be captured in the SAS program log if you want to double check and see that you've got the behavior that you're expecting.



CAS Cache Usage



CAS_DISK_CACHE

Considerations

What’s going on with CAS_DISK_CACHE?

- CAS represents its data in memory as SASHDAT blocks, regardless of source.
- For flexibility and availability, CAS will ensure that in-memory SASHDAT blocks are memory-mapped to Worker-accessible local disk
- So, after loading data from some (but not all) external sources, CAS will establish m-maps from its in-memory data to CAS_DISK_CACHE
- If the original source is SASHDAT which can be m-mapped by the Workers on locally accessible disk, then a CAS_DISK_CACHE copy is not needed
- In any case, when changes are made to in-memory data, CAS tracks those as new in-memory SASHDAT blocks which are m-mapped to the CAS cache.

What's going on with CAS disk cache?

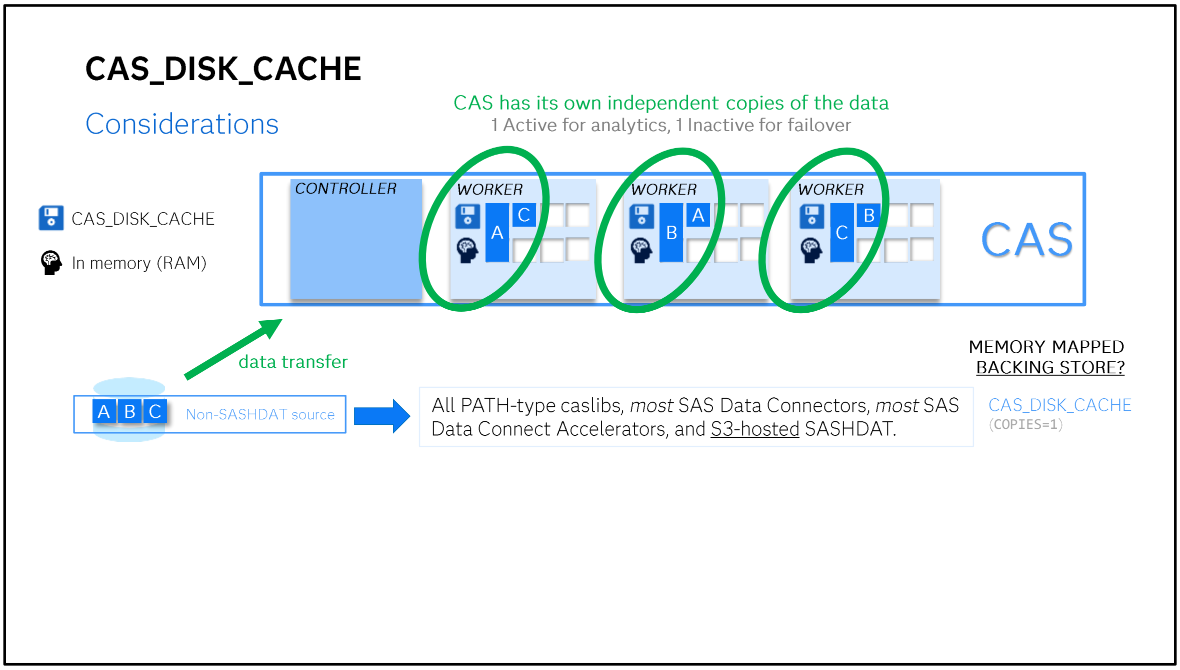
CAS represents its data in memory as SASHDAT blocks, regardless of the source.

For flexibility and availability, CAS will ensure that in-memory SASHDAT blocks are memory mapped to worker accessible local disk.

So, after loading data from some (but not all) external sources, CAS will establish memory maps from its in-memory data to the CAS disk cache.

If the original source is SASHDAT and that source can be memory mapped by the workers on locally accessible disk directly, then CAS disk cache copy is not needed.

In any case, when changes are made to in-memory data, CAS tracks those changes as new in-memory SASHDAT blocks, which are then memory mapped to the CAS cache.



CAS_DISK_CACHE

Considerations

CAS has its own independent copies of the data
1 Active for analytics, 1 Inactive for failover

💾 CAS_DISK_CACHE

In memory (RAM)

CONTROLLER

WORKER A C

WORKER B A

WORKER C B

CAS

data transfer

A B C Non-SASHDAT source

All PATH-type caslibs, *most* SAS Data Connectors, *most* SAS Data Connect Accelerators, and S3-hosted SASHDAT.

MEMORY MAPPED
BACKING STORE?

CAS_DISK_CACHE
(COPIES=1)

To help explain when the CAS disk cache is used, we're going to do another illustration to make it all a little bit clearer.

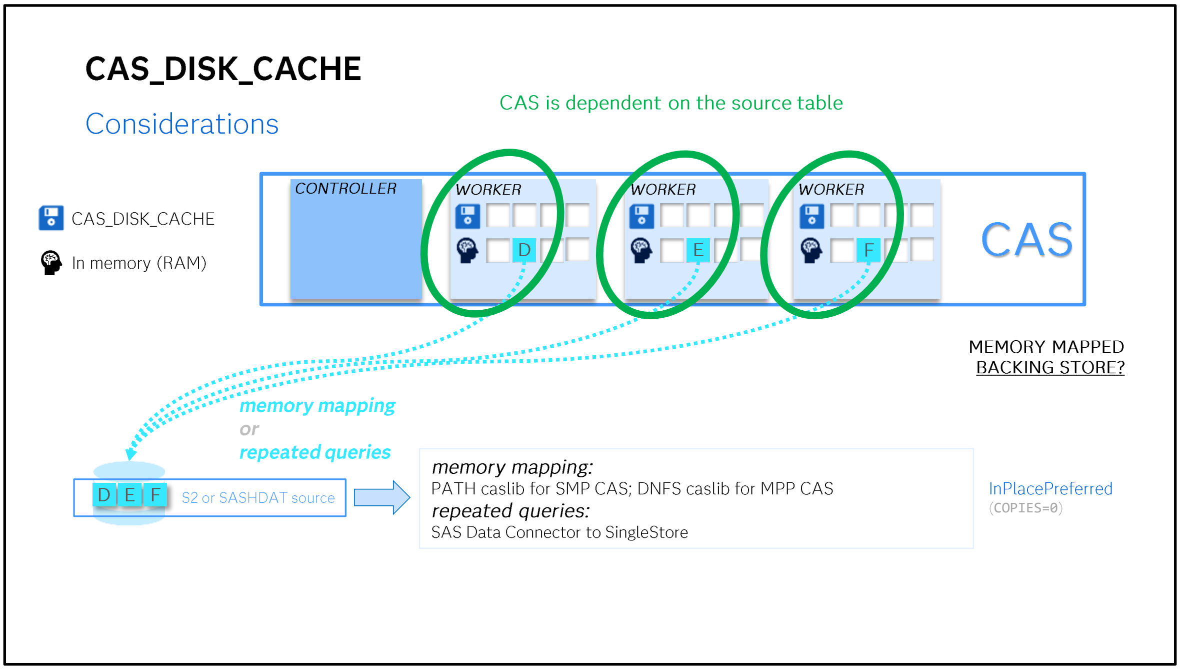
And in this illustration, we have a representation of multi-machine CAS with a controller and three workers similar to what we've seen before. We've got slots for data that's in-memory represented by the brain and other slots represented for data that is stored in the CAS disk cache represented by a floppy disk icon.

And when we talk about data that's coming from a non-SASHDAT source, whether it's from a remote data provider or if it's a CSV file or a SAS data set or SPDE data from a Hadoop cluster, doesn't matter as long as it's a non-SASHDAT source. Again, we're using blocks shown simply as A, B, and C to represent the data being worked with. The actual number of blocks may be in the hundreds, or thousands, or more.

And depending on the source type, when we access this data using the appropriate kinds of caslibs, like PATH, most of the SAS data connectors, most of the SAS data connect accelerators, as well as also including SASHDAT that's hosted in Amazon's S3, then we see the behavior illustrated here.

This data is loaded into memory and memory-mapped to the CAS disk cache as the backing store. Another aspect of this load is that the data is replicated with copies equals one behavior so that if we lose a worker, we still have a complete copy of the table to work with.

In other words, when the CAS disk cache is the backing store, this means that CAS has its own independent copy of the data to work with. Changes made at the source after the initial load do not affect what CAS has in memory.



CAS_DISK_CACHE

Considerations

CAS is dependent on the source table

memory mapping:
PATH caslib for SMP CAS; DNFS caslib for MPP CAS
repeated queries:
SAS Data Connector to SingleStore

InPlacePreferred
(COPIES=0)

Let’s look at a different situation where we have data that’s either stored in a SingleStore database or is a SASHDAT source that the CAS nodes have direct file access to. These are two very different places to keep your data, but they share some common behaviors exhibited by CAS.

Getting to the data depends on what’s going on in the environment. For SASHDAT, it could be accessed using the PATH type of caslib from an SMP (that is, single machine) CAS server, or by using the DNFS type of caslib for MPP (that is, multi-machine) CAS server. Note that we’re being specific here – the PATH type of caslib is available to MPP CAS, but will not exhibit this behavior. And due to how the DNFS type of caslib is defined, it’s not an option for SMP CAS.

We also include data that CAS accesses in SingleStore.

Specifically, we get a different default behavior than shown on the previous slide for non-SASHDAT sources. It’s called InPlacePreferred.

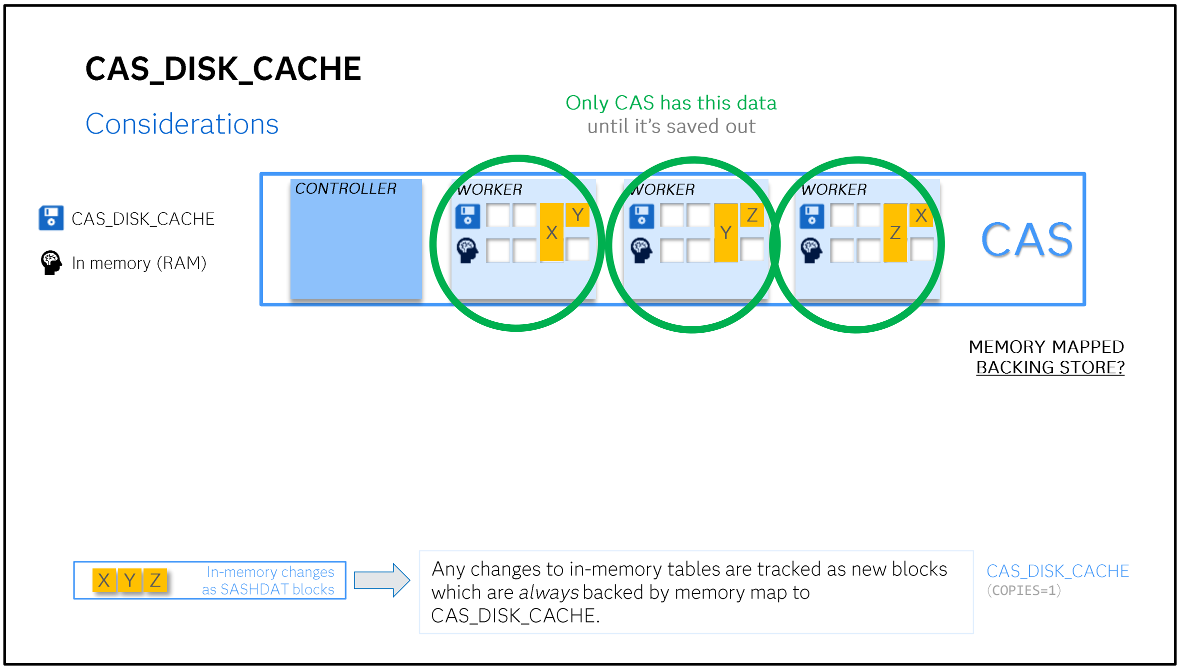
For SASHDAT, when we tell CAS to load the data, it doesn’t actually do that. Not yet anyway. Instead it creates a memory map from the workers’ RAM to the source data where it lives on disk. That only takes a fraction of a second and when the maps exist, CAS sends a message to the SAS program log that the load is complete.

This process is referred to as a “lazy load”. The first time CAS attempts to access the data for analytics purposes, then the OS on the CAS workers will notice that the data is not actually there yet. It will follow the memory map to fetch the data and that’s when the actual load first occurs.

For data in SingleStore, the movement is different, but the result is basically the same. Because a memory map is not possible – data in SingleStore is not SASHDAT format nor directly accessible on disk local to the CAS workers – then CAS will simply make repeated queries as needed to get the data.

Either way, CAS is dependent on the table at the source. If the SASHDAT file changes on disk, that can invalidate the memory mappings as pointers get shuffled around. A table in SingleStore will be requeried, so any changes to the source might come in for the next CAS analytics task. It's important to plan for this behavior.

There are benefits to the InPlacePreferred approach. It can limit data movement, data versions, etc. as well as provide a means for working with frequently updated tables. Just remember that we're describing the default behavior here. It can be overridden to achieve different outcomes and efficiencies as needed.



CAS_DISK_CACHE

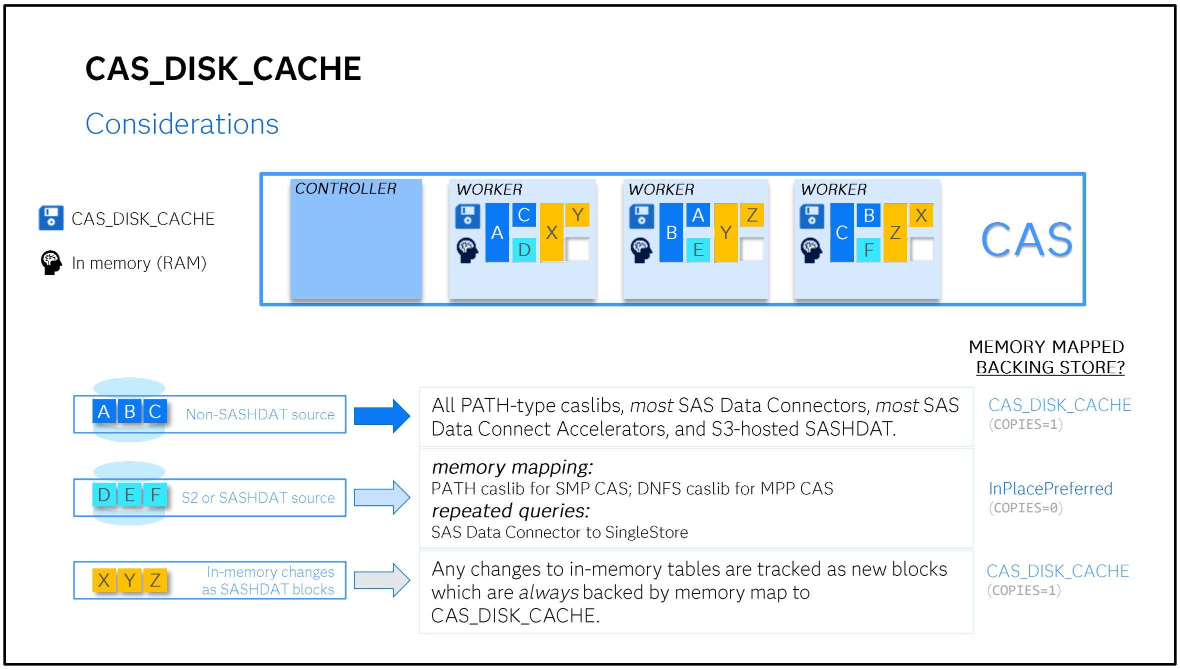
Considerations

Any changes to in-memory tables are tracked as new blocks which are *always* backed by memory map to CAS_DISK_CACHE.

CAS_DISK_CACHE
(COPIES=1)

And finally, let's look at what happens when we make changes to an in-memory table. Those changes are tracked as new blocks and kept in-memory with a memory map to CAS disk cache as the backing store along with COPIES=1 for failover replication.

When it comes to changes, only CAS has this data until it's explicitly saved out somewhere.



CAS_DISK_CACHE

Considerations

CAS_DISK_CACHE

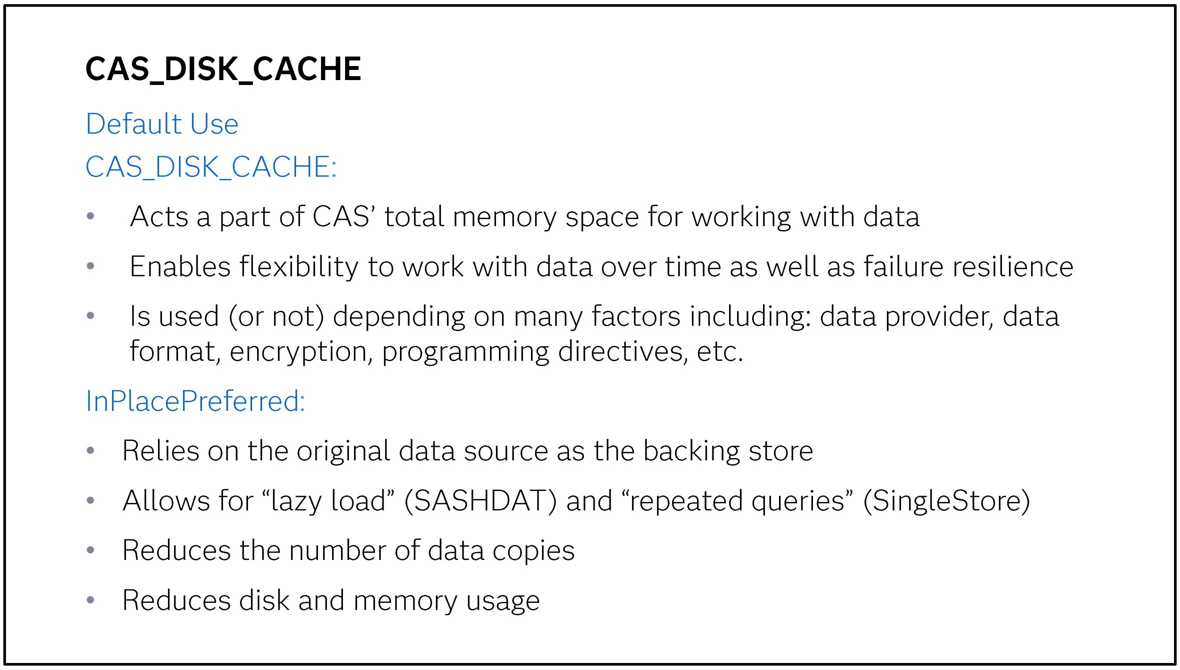
In memory (RAM)

		MEMORY MAPPED BACKING STORE?
A B C Non-SASHDAT source	All PATH-type caslibs, *most* SAS Data Connectors, *most* SAS Data Connect Accelerators, and S3-hosted SASHDAT.	CAS_DISK_CACHE (COPIES=1)
D E F S2 or SASHDAT source	*memory mapping:* PATH caslib for SMP CAS; DNFS caslib for MPP CAS *repeated queries:* SAS Data Connector to SingleStore	InPlacePreferred (COPIES=0)
X Y Z In-memory changes as SASHDAT blocks	Any changes to in-memory tables are tracked as new blocks which are *always* backed by memory map to CAS_DISK_CACHE.	CAS_DISK_CACHE (COPIES=1)

Altogether now, we can see this behavior in a single CAS server if it's loaded with data from various sources.

In summary, CAS makes default choices about the backing store location and technique that it will use depending on the data source and type.

Understanding these defaults as well as their optional overrides can help achieve certain efficiencies when working with the data.



CAS_DISK_CACHE

Default Use

CAS_DISK_CACHE:

- Acts a part of CAS' total memory space for working with data
- Enables flexibility to work with data over time as well as failure resilience
- Is used (or not) depending on many factors including: data provider, data format, encryption, programming directives, etc.

InPlacePreferred:

- Relies on the original data source as the backing store
- Allows for "lazy load" (SASHDAT) and "repeated queries" (SingleStore)
- Reduces the number of data copies
- Reduces disk and memory usage

The CAS disk cache is often used as the backing store for in-memory data, but not always. Default behaviors per data source and type have been established.

When CAS disk cache is used, then it acts as a part of CAS's total memory space for working with the data as a form of virtual memory.

That gives us the flexibility to work with data over time as well as failure resilience.

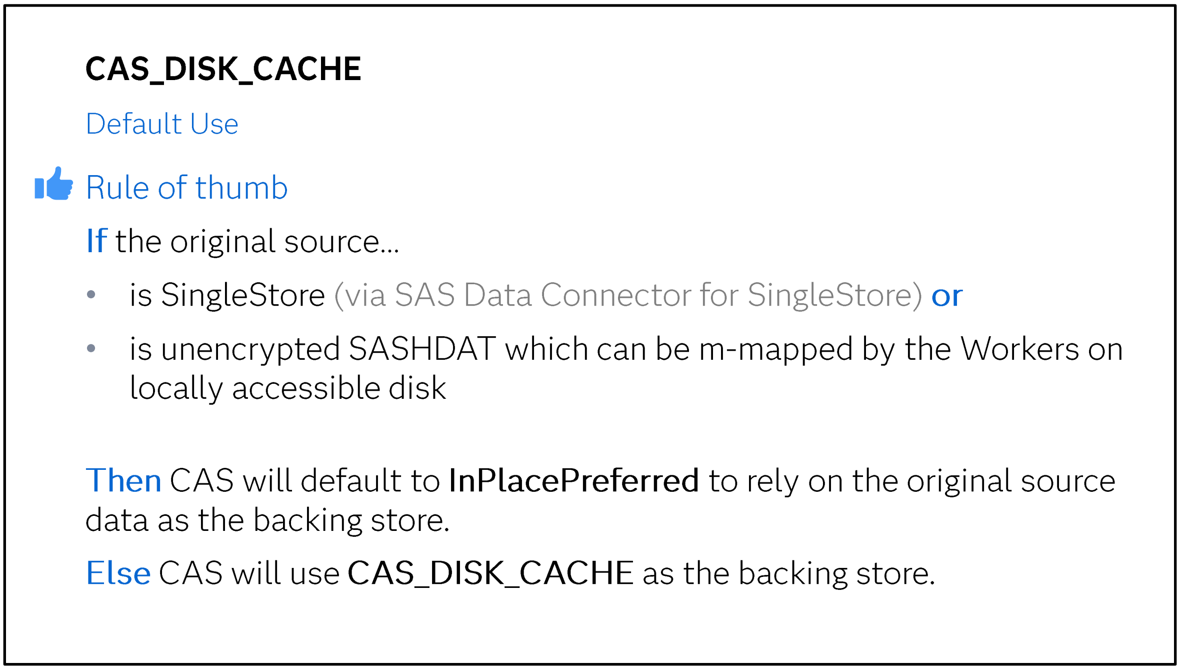
Whether the CAS disk cache is used or not depends on many factors, including the data provider, the data format, encryption, programming directives, et cetera.

When we go with the inPlacePreferred type of approach for a backing store, that relies on the original data source to act as the backing store.

With inPlacePreferred, CAS will implement a lazy load approach for unencrypted SASDHAT on directly-accessible disk. Or, for SingleStore data, it will perform repeated queries of the data.

In-place preferred also reduces the number of copies of the data that exist. For some sites, this is a key feature to focus on one version of the data for consistent reporting.

And it also reduces disk and memory usage in CAS, because we don't have to rely on the CAS disk cache as much.



CAS_DISK_CACHE

Default Use

Rule of thumb

If the original source…

- is SingleStore (via SAS Data Connector for SingleStore) or
- is unencrypted SASHDAT which can be m-mapped by the Workers on locally accessible disk

Then CAS will default to **InPlacePreferred** to rely on the original source data as the backing store.

Else CAS will use **CAS_DISK_CACHE** as the backing store.

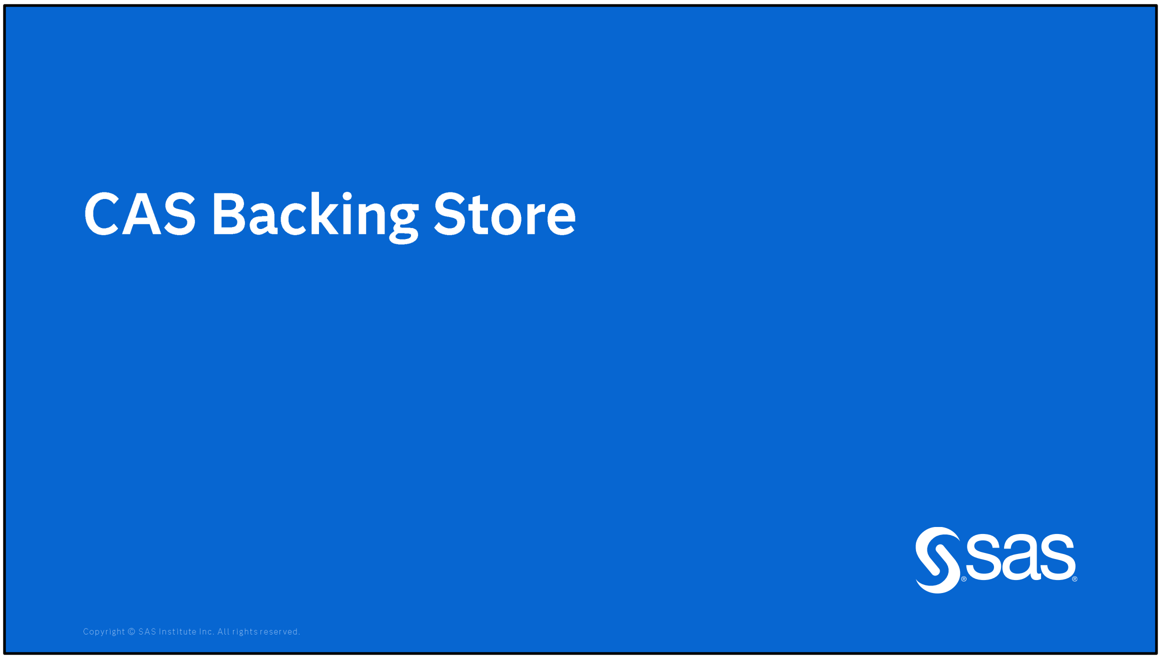
As a rule of thumb…

If the original source is SingleStore, and we're using the SAS Data Connector for SingleStore, or…

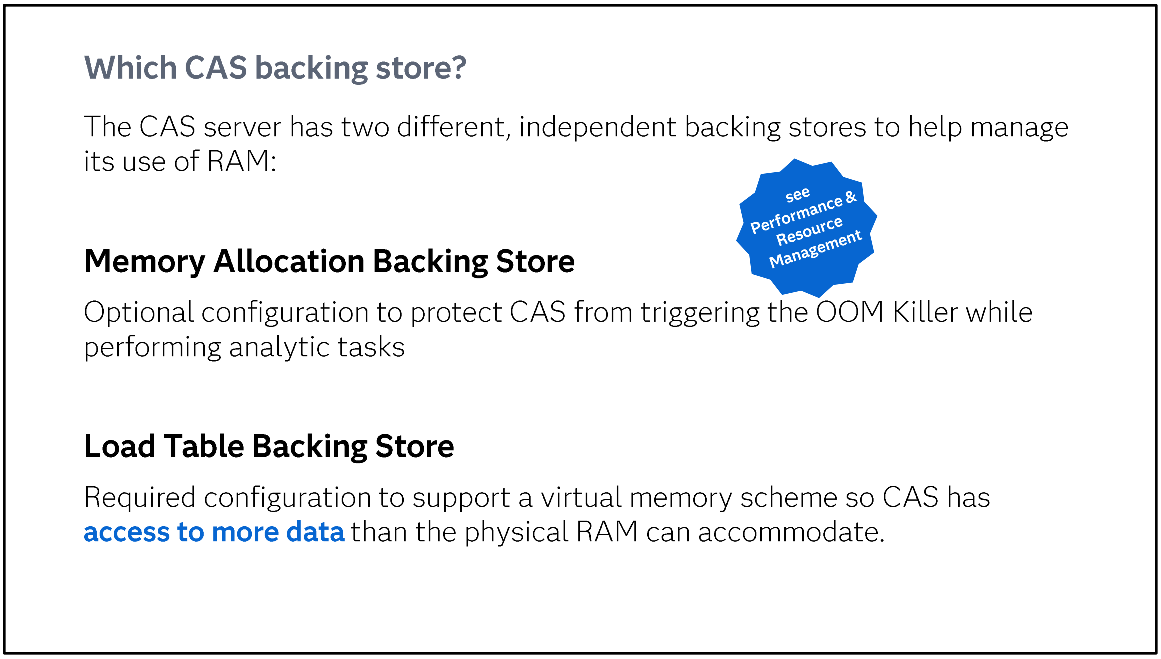
Is unencrypted SASHDAT, which could be memory mapped by the workers on locally accessible disk…

Then CAS will default to the inPlacePreferred approach to rely on the original source data as the backing store.

Else, in any other data situation, CAS will rely on its CAS_DISK_CACHE to act as the backing store for in-memory data.



CAS Backing Store



Which CAS backing store?

The CAS server has two different, independent backing stores to help manage its use of RAM:

see Performance & Resource Management

Memory Allocation Backing Store

Optional configuration to protect CAS from triggering the OOM Killer while performing analytic tasks

Load Table Backing Store

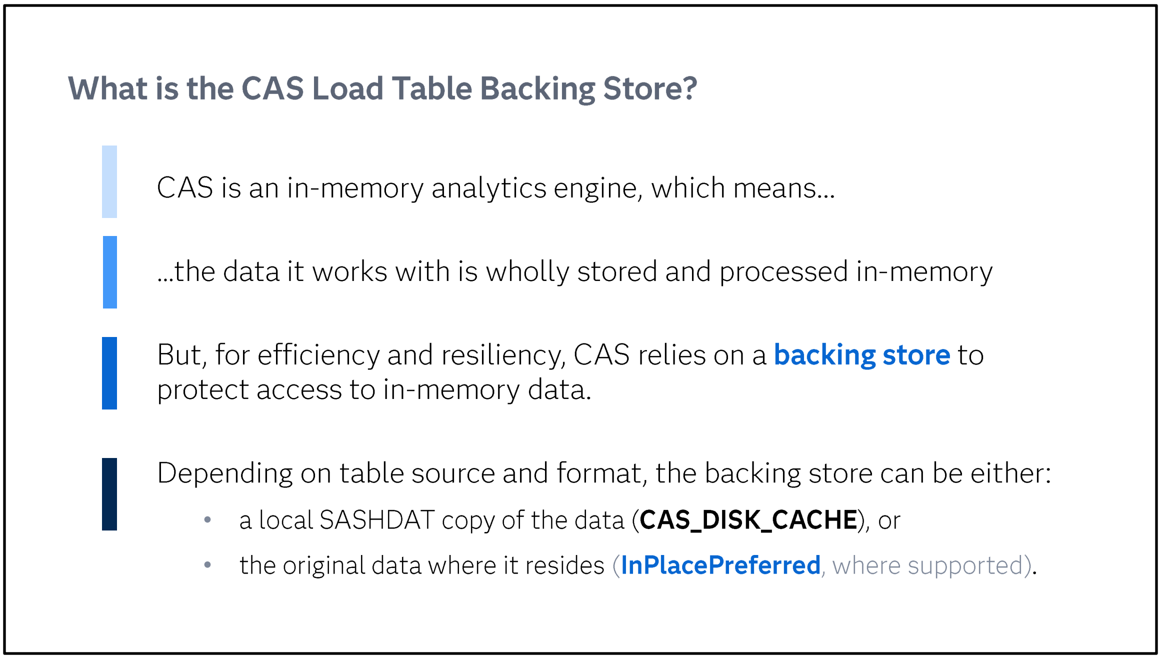
Required configuration to support a virtual memory scheme so CAS has **access to more data** than the physical RAM can accommodate.

Let's look at the CAS backing store. As of the 2024.09 release of SAS Viya, we now need to distinguish between two different backing stores.

The first is the memory allocation backing store. This is a recent feature added to CAS. It provides an option for you to configure CAS to protect its use of RAM during analytic processing operations. Specifically, since CAS is an in-memory analytics engine, then it needs to use a lot of RAM for calculations and storing interim results. However, Kubernetes has some rigid controls on how much ad-hoc memory can be utilized and the downside is that it's possible the out-of-memory killer process could inadvertently shut down CAS completely.

We cover more about this in the Performance and Resource Management content.

Next is the load table backing store. This is a required configuration you specify that lets CAS use local disk as part of a virtual memory scheme so that it has the access it needs to volumes of data that might be larger than the physical RAM offered on the host machines. That's what we'll talk about in this module.



What is the CAS Load Table Backing Store?

CAS is an in-memory analytics engine, which means…

…the data it works with is wholly stored and processed in-memory

But, for efficiency and resiliency, CAS relies on a **backing store** to protect access to in-memory data.

Depending on table source and format, the backing store can be either:

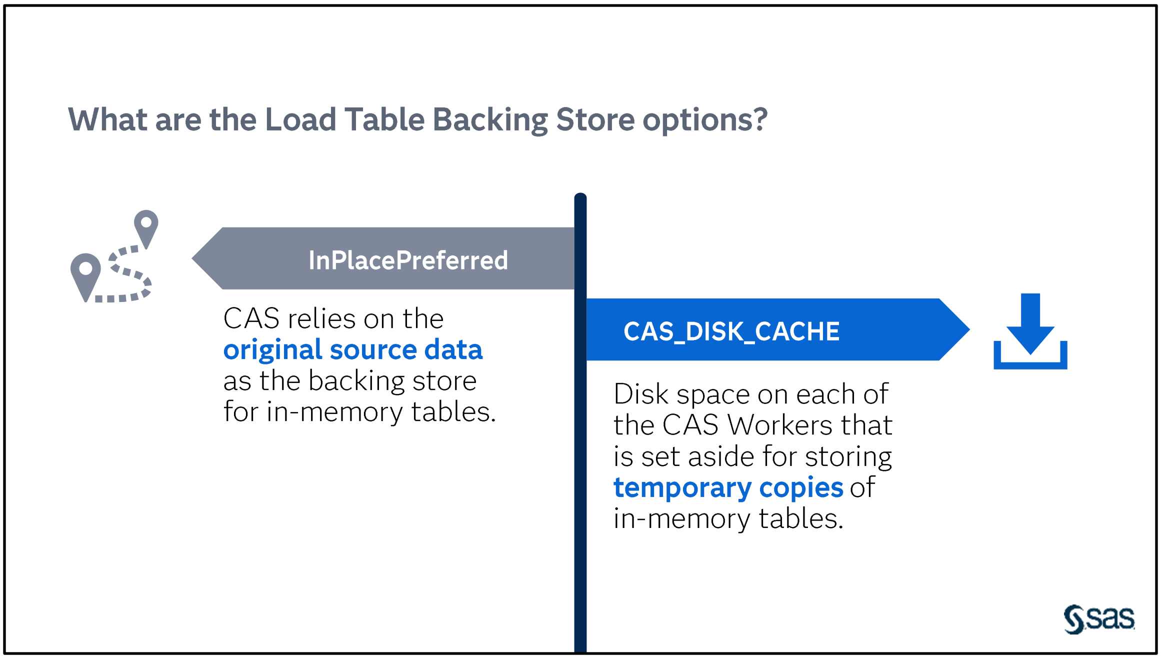
- a local SASHDAT copy of the data (**CAS_DISK_CACHE**), or
- the original data where it resides (**InPlacePreferred**, where supported).

CAS is an in-memory analytics engine, which means...

…the data it works with is wholly stored and processed in-memory

But, for efficiency and resiliency, CAS relies on a backing store to protect access to in-memory data.

Depending on table source and format, the backing store can be either a local SASHDAT copy of the data (using CAS_DISK_CACHE) or the original data where it resides (known as InPlacePreferred, where it’s supported).



What are the Load Table Backing Store options?

InPlacePreferred

CAS relies on the **original source data** as the backing store for in-memory tables.

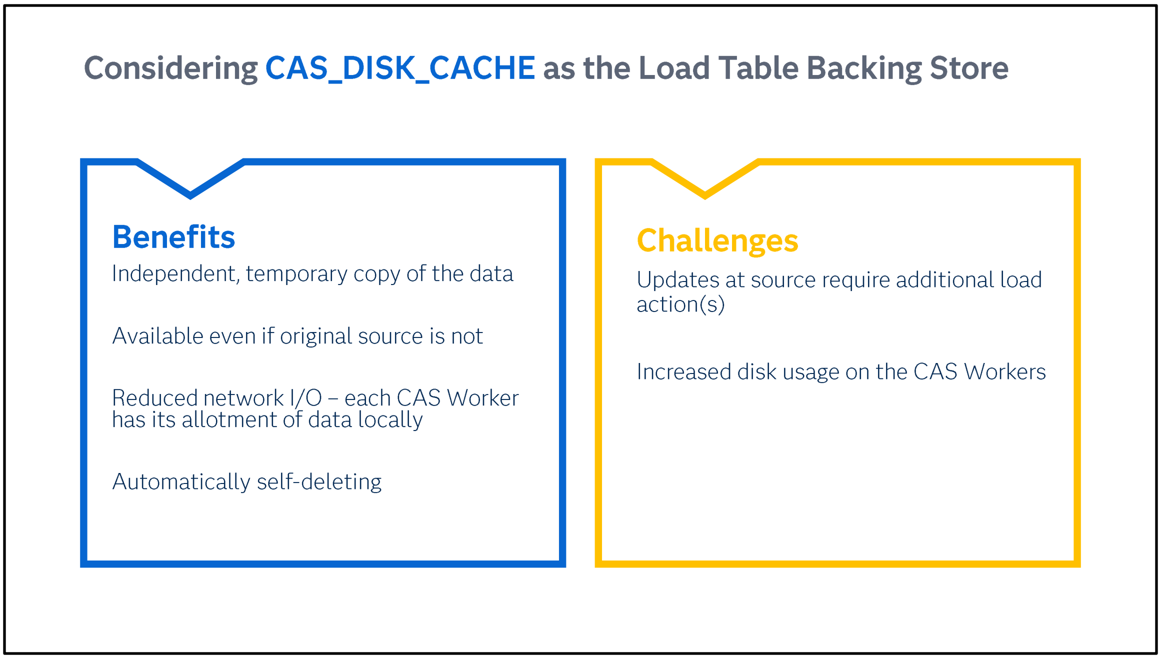
CAS_DISK_CACHE

Disk space on each of the CAS Workers that is set aside for storing **temporary copies** of in-memory tables.

Let’s look a little closer at these two options for the backing store.

First, there’s InPlacePreferred. When this approach to a backing store is selected, then CAS will rely on the original source data as the backing store for the tables it has in memory.

Else CAS can use its disk cache. We’ve described the CAS_DISK_CACHE approach earlier. It’s where each CAS Worker has a dedicated local disk set aside for storing temporary copies of its in-memory tables.



Considering CAS_DISK_CACHE as the Load Table Backing Store

Benefits

Independent, temporary copy of the data

Available even if original source is not

Reduced network I/O – each CAS Worker has its allotment of data locally

Automatically self-deleting

Challenges

Updates at source require additional load action(s)

Increased disk usage on the CAS Workers

When using the CAS_DISK_CACHE as the backing store, we can realize certain benefits.

The copy of data in-memory and stored as a copy in the disk cache is independent of the source. This means that the source can update yet CAS will still see the data in the state it was when originally loaded. Data stored in the CAS_DISK_CACHE is temporary and isn't stored permanently in this location.

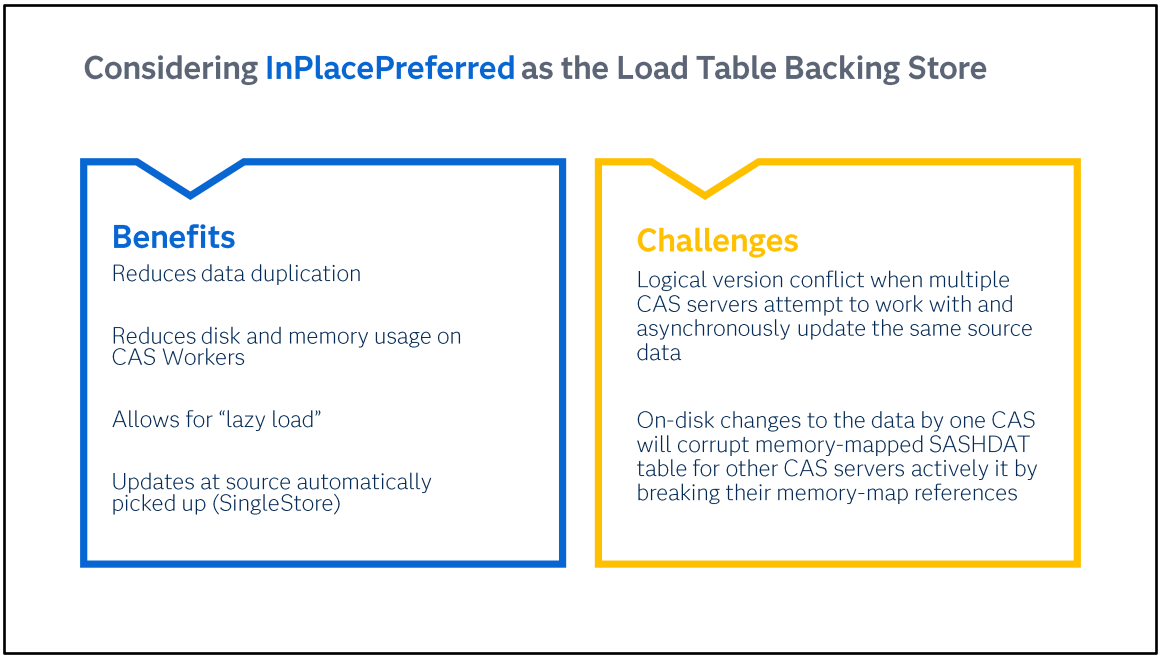
Of course, this means CAS has a copy of the data even if the source disappears for some reason and can continue to work.

If CAS needs to reference the cached copy of the data, then it likely won't use network i/o since we typically recommend mounting the volumes for CAS_DISK_CACHE as disk local to the CAS Worker node.

The data in CAS_DISK_CACHE is only useful for a limited lifetime. When CAS places data in its disk cache, it preemptively marks the file as deleted with the OS but keeps the file handle open. That way, CAS can continue to work with the data as it's needed. But when CAS shuts down and releases the file handle, then that disk space is treated as free and available by the OS.

On the other hand, there are a couple of other attributes to bear in mind when using the CAS_DISK_CACHE.

If you're working with source data that updates frequently, then you'll need to ensure CAS reloads the changed data as desired for your process. Also, it's important to provide sufficient disk volume to store the copies of data stored by CAS in its disk cache as well as ensure that the I/O throughput of that disk is fast enough to allow performant and efficient operation.



Considering InPlacePreferred as the Load Table Backing Store

Benefits

Reduces data duplication

Reduces disk and memory usage on CAS Workers

Allows for “lazy load”

Updates at source automatically picked up (SingleStore)

Challenges

Logical version conflict when multiple CAS servers attempt to work with and asynchronously update the same source data

On-disk changes to the data by one CAS will corrupt memory-mapped SASHDAT table for other CAS servers actively it by breaking their memory-map references

When using the InPlacePreferred as the backing store,
there are different benefits.

There’s no explicit data duplication for caching purposes. And in some ways, InPlacePreferred reduces disk and memory usage on the CAS Workers.

For example, InPlacePreferred allows for lazy loading of tables which is a trick where CAS simply creates a memory-map from RAM to the source table on disk. This takes a tiny fraction of time compared to explicitly transferring the entire table completely. When CAS is directed to analyze the table later, then the OS will utilize the memory map to transfer the data from disk to RAM as required.

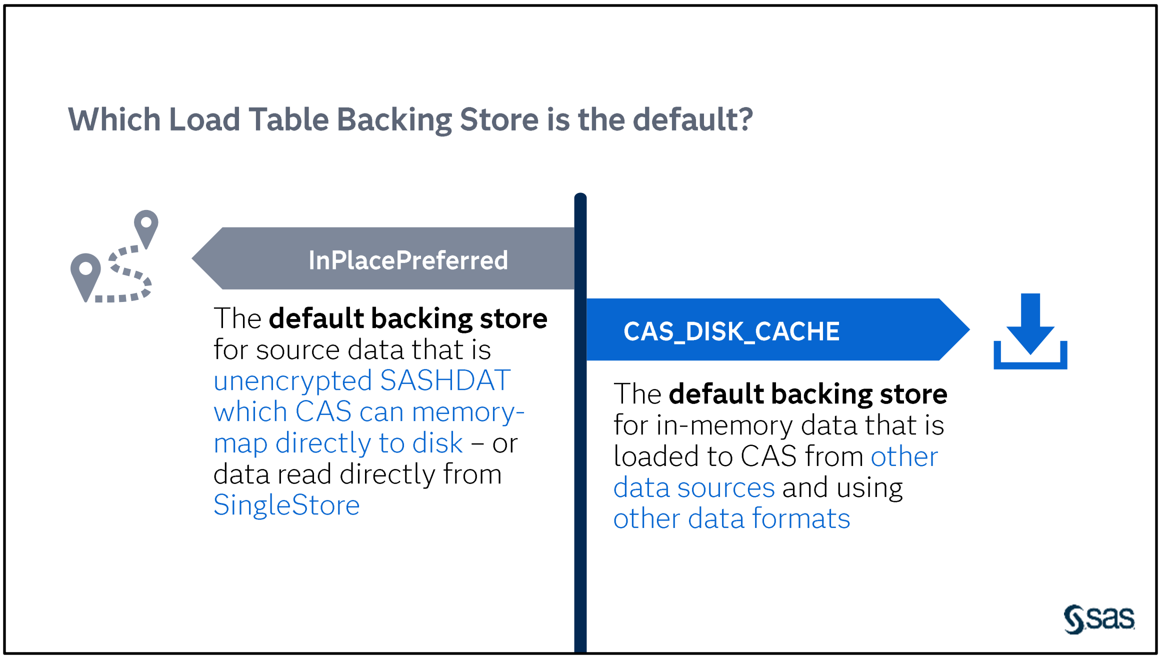
When source data is referenced from a SingleStore database, then another approach to InPlacePreferred is seen. Instead of memory-mapping, CAS will repeatedly use queries to bring the data over from the SingleStore database each time it’s referenced.

And finally, another benefit is that changes made to a source table in SingleStore are automatically picked up due to the repeated queries approach.

There are challenges to manage with the InPlacePreferred approach to the CAS backing store, too.

If there are other actors besides CAS working with the source data, then it’s possible to encounter a logical conflict where the version in CAS isn’t what you intend to work with. Those other actors might make changes you’re not anticipating or could write-back their changes on top of yours. So, take care to understand if the source is locked for your use, and if not, then take necessary steps to avoid conflicts.

Also, if InPlacePreferred is used for SASHDAT tables on disk with memory-mapping, then ensure that other CAS servers do not access and modify that data while you’re working with it. If that happens, then the memory maps established by your CAS server will be broken if the data is altered, deleted, or otherwise changed on disk.



Which Load Table Backing Store is the default?

InPlacePreferred

The **default backing store** for source data that is unencrypted SASHDAT which CAS can memory-map directly to disk – or data read directly from SingleStore

CAS_DISK_CACHE

The **default backing store** for in-memory data that is loaded to CAS from other data sources and using other data formats

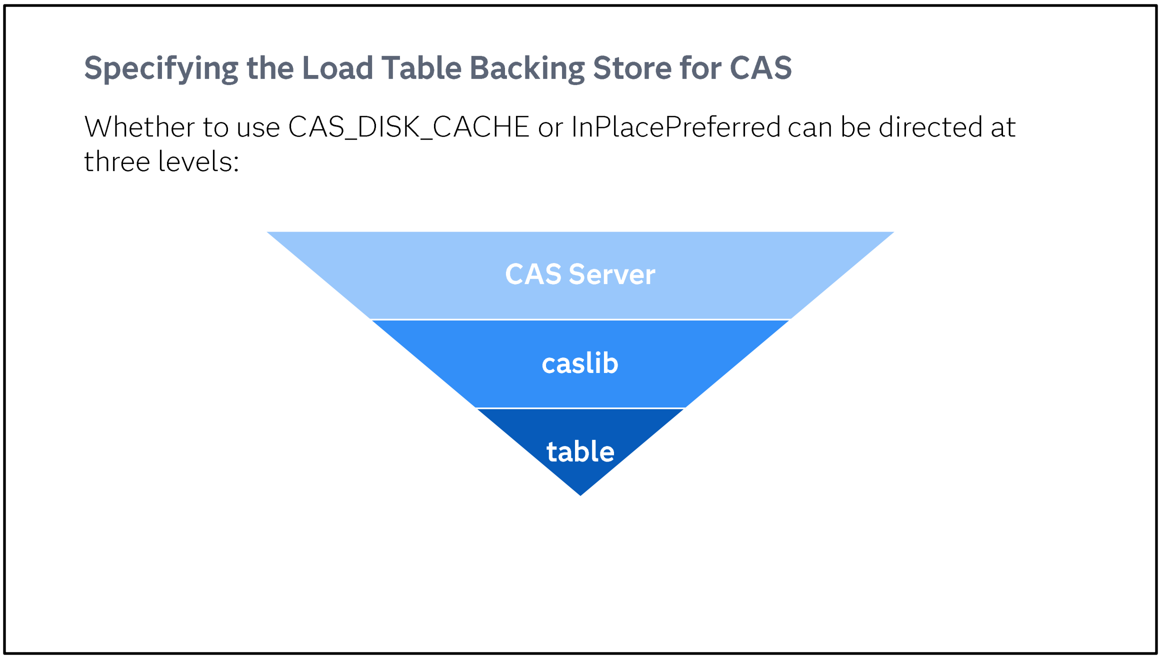
Which backing store approach is the default? Well, it depends. CAS will always implement a backing store for its in-memory data, but the approach selected by default will vary by source and format.

CAS will default to use the InPlacePreferred approach when the source data is unencrypted SASHDAT that CAS can memory-map directly to the source disk. By definition, this excludes SASHDAT accessed directly using the Amazon S3 protocol, for example, since that’s not a POSIX-compliant disk location that can be memory-mapped.

CAS will also default to use InPlacePreferred when data is accessed in SingleStore. This is because SAS and SingleStore have embarked on a partnership to realize performance and efficiency improvements by working closely together.

CAS will default to use its CAS_DISK_CACHE as the backing store for data that’s loaded from other data sources or using other data formats not mentioned for InPlacePreferred.

Put another way, CAS_DISK_CACHE is the default in most situations, but there are significant exceptions where InPlacePreferred is used as the default instead.



Specifying the Load Table Backing Store for CAS

Whether to use CAS_DISK_CACHE or InPlacePreferred can be directed at three levels:

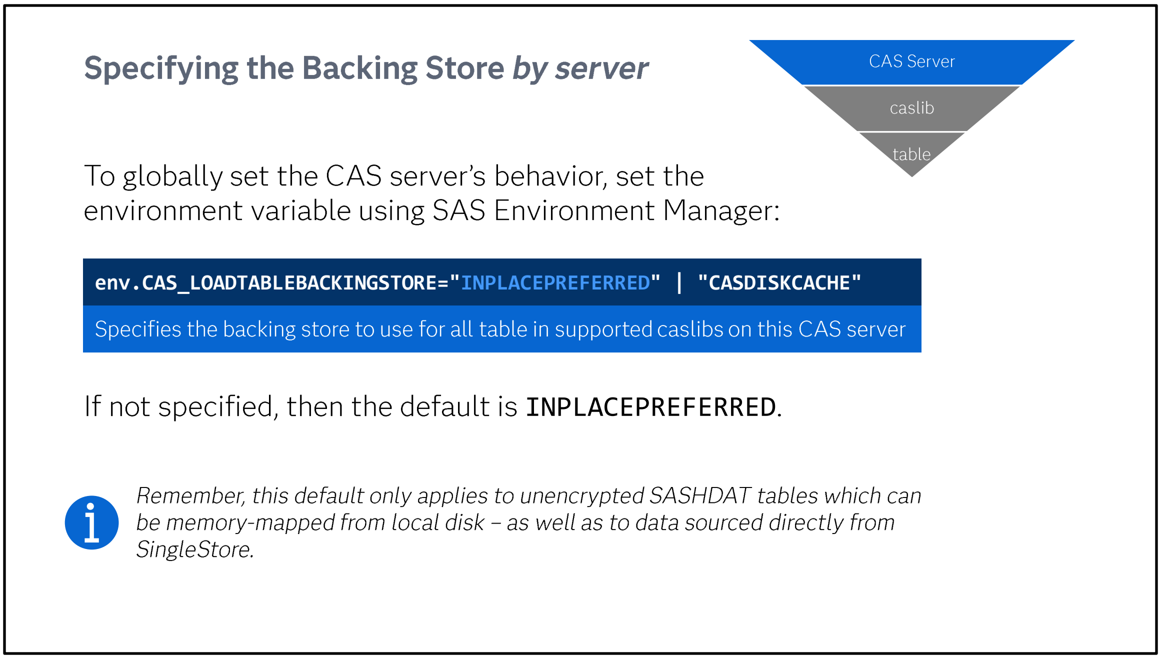
CAS Server

caslib

table

Now that we have been introduced to the backing store options and the default approach that is selected for data sources, let’s look at overriding the default and specifying the backing store we prefer.

We can specify the desired backing store at three levels: for the CAS Server as a whole, for an entire caslib, or per table.



Specifying the Backing Store *by server*

CAS Server
caslib
table

To globally set the CAS server’s behavior, set the environment variable using SAS Environment Manager:

```
env.CAS_LOADTABLEBACKINGSTORE="INPLACEPREFERRED" | "CASDISKCACHE"
```

Specifies the backing store to use for all table in supported caslibs on this CAS server

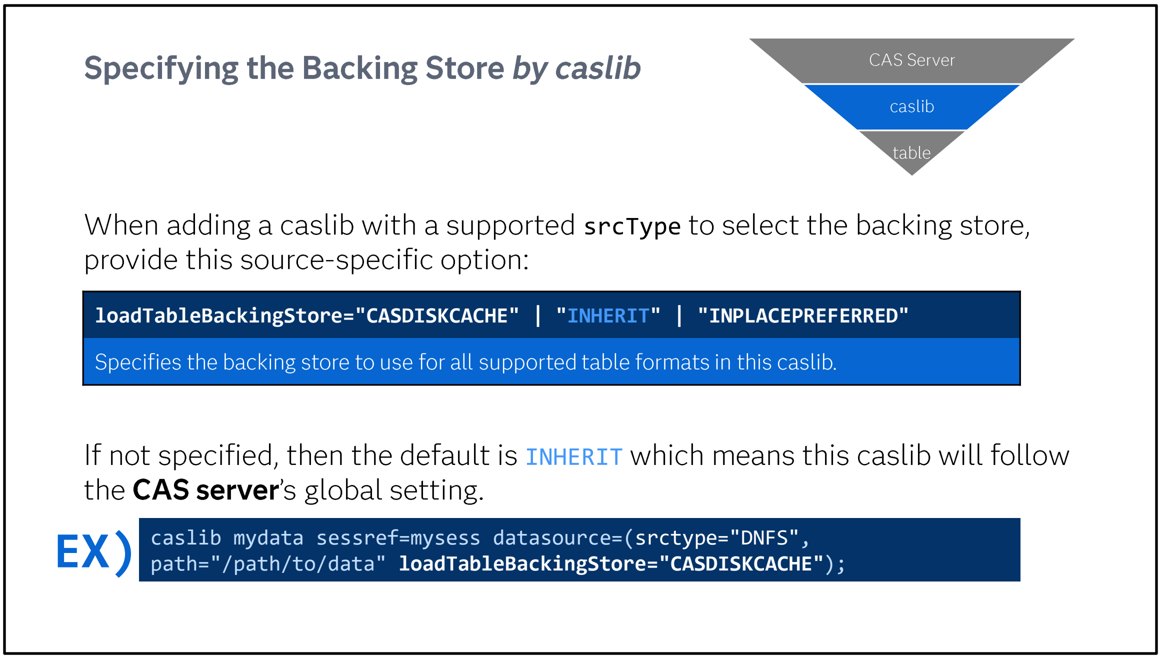
If not specified, then the default is `INPLACEPREFERRED`.

Remember, this default only applies to unencrypted SASHDAT tables which can be memory-mapped from local disk – as well as to data sourced directly from SingleStore.

To globally set the CAS server's behavior, you can set an environment variable using the SAS Environment Manager. The variable is named env.CAS_LoadTableBackingStore and it takes two values: InPlacePreferred and CASDiskCache.

The InPlacePreferred value is the default globally for the CAS Server if it has not been explicitly specified.

This is where there’s a little bit of a twist. The term “default” in this case must be caveated to remind you that InPlacePreferred only applies to unencrypted SASHDAT data that can be memory-mapped from locally-accessible disk or for data in SingleStore. Otherwise, CAS_DISK_CACHE will act as the backing store for data coming from anywhere else or in a different format.



CAS Server
caslib
table

Specifying the Backing Store *by caslib*

When adding a caslib with a supported `srcType` to select the backing store, provide this source-specific option:

`loadTableBackingStore="CASDISKCACHE" | "INHERIT" | "INPLACEPREFERRED"`

Specifies the backing store to use for all supported table formats in this caslib.

If not specified, then the default is `INHERIT` which means this caslib will follow the **CAS server**'s global setting.

EX)

```
caslib mydata sessref=mysess datasource=(srctype="DNFS",
path="/path/to/data" loadTableBackingStore="CASDISKCACHE");
```

Now that we know what backing store approach the CAS Server will default to , we can override it to specify the backing store for a single caslib with the LoadTableBackingStore option on the caslib statement. It takes two values we've already seen - CASDiskCache and InPlacePreferred – plus a third value, Inherit.

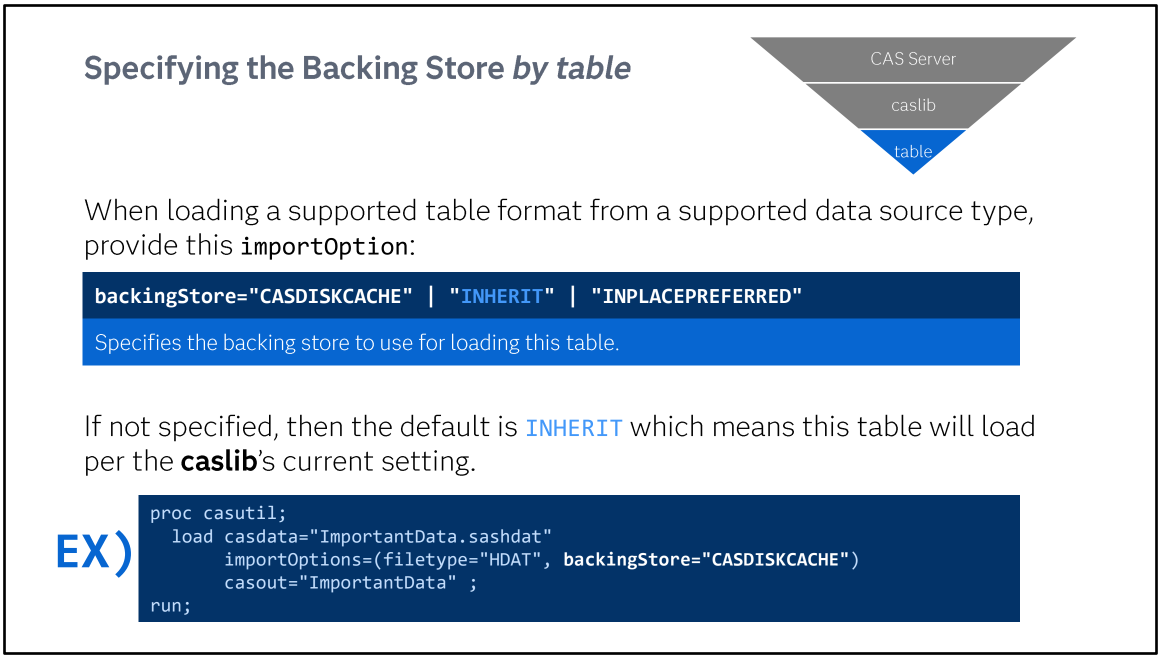
If this option isn't specified on the caslib statement, then it will default to the Inherit value where it will use the same backing store that the CAS Server specifies.

Shown here is a simple caslib statement example with the LoadTableBackingStore option selecting the CASDiskCache approach.

This is a good place to point out the source Type shown in the caslib statement. Now that we understand that InPlacePreferred is really only available in very certain circumstances, then we know it can only work with three source Types. First is DNFS (as shown here) which is only available on MPP CAS Servers to access SASHDAT directly from the Workers.

Next, we can also access SASHDAT using the PATH type of caslib. But, thinking it through, InPlacePreferred is only available on SMP CAS Servers. Remember, it has to memory-map from the CAS Worker to the SASHDAT file on disk and even though MPP CAS can use PATH to load SASHDAT, doing so is exclusively through the CAS Controller. With SMP CAS, it's a single-machine instance of CAS acting as both Controller and Worker, so that's why it can use InPlacePreferred.

And third is when the source Type specified is SingleStore, as noted earlier.



Specifying the Backing Store *by table*

CAS Server

caslib

table

When loading a supported table format from a supported data source type, provide this `importOption`:

backingStore="CASDISKCACHE" | "INHERIT" | "INPLACEPREFERRED"

Specifies the backing store to use for loading this table.

If not specified, then the default is `INHERIT` which means this table will load per the **caslib**’s current setting.

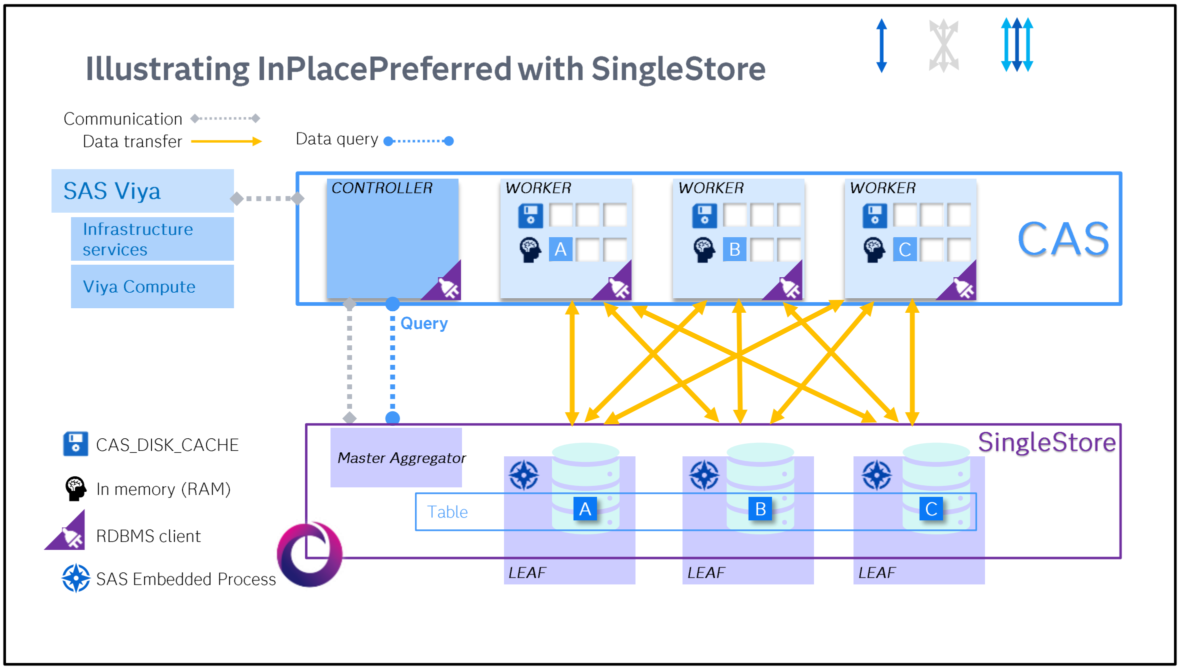
EX)

```
proc casutil;
  load casdata="ImportantData.sashdat"
       importOptions=(filetype="HDAT", backingStore="CASDISKCACHE")
       casout="ImportantData" ;
run;
```

Similar to caslib, we can also specify the backing store approach for each table that’s loaded, if we want. The BackingStore option is specified as an ImportOption in the CASUTIL procedure.

Again, the default here is Inherit which means the table load will inherit the same backing store as specified by the caslib.

And here’s a simple example showing CASDiskCache specified as the backing store approach for this table when loaded. Assuming proper architecture and other defaults, this table otherwise would have InPlacePreferred as the backing store such that the CAS Workers would have implemented a lazy load.



Illustrating InPlacePreferred with SingleStore

Now we'll discuss what happens when we direct CAS to access data from SingleStore.

Shown here, we again see our familiar simplified illustration of a SAS Viya deployment with an MPP CAS server sporting a controller and three workers. Below that is an external data source, shown as a large deployment of SingleStore with SAS In-Database technology for Viya.

As established earlier, in order for CAS to communicate with a remote data provider, it will use an RDBMS client - shown here with the purple plug icon on the CAS controller and workers – that we know from a Kubernetes perspective, has been placed in a volume that is already configured, tested, and mounted to the CAS controller and worker pods.

We also want to show that the SAS Embedded Process has been deployed to the SingleStore data source, typically running an instance on each leaf of that cluster.

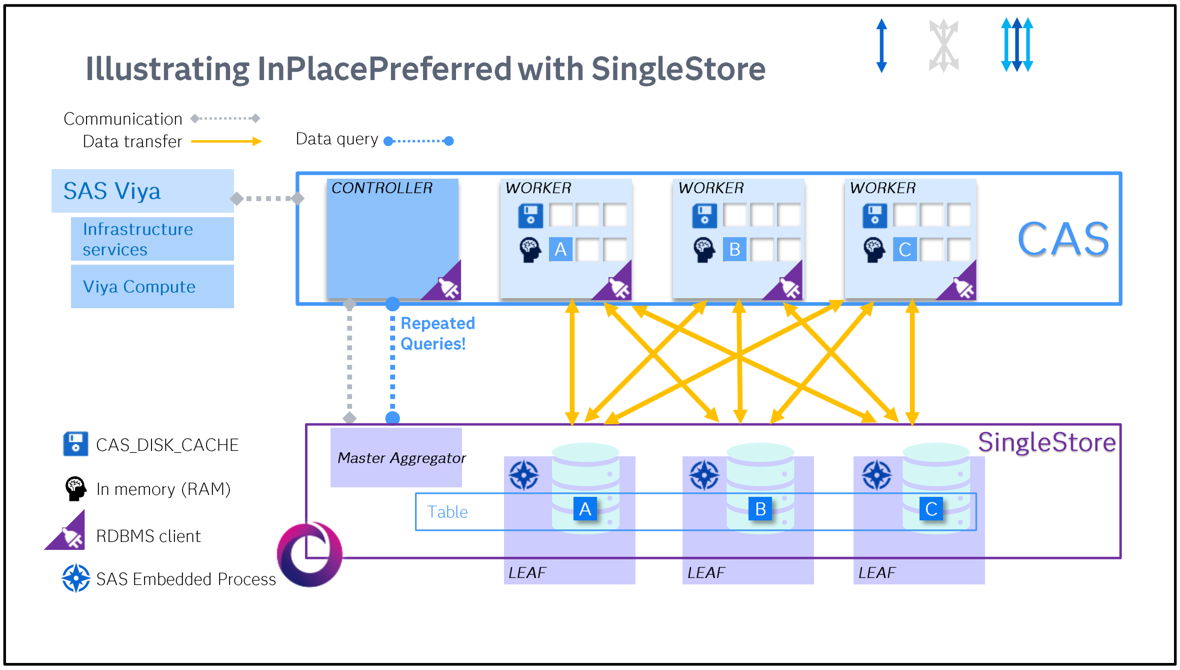
The data we want in the SingleStore database shown here is represented again as blocks labeled "A", "B", and "C". In real life, of course, the actual number of blocks would be much higher, but we're keeping it simple here.

Now that we know where everything is, let's get that data loaded into CAS.

Things work a little different here than we've seen earlier. For the SingleStore data source, CAS will default to use the InPlacePreferred approach to rely on SingleStore as the backing store. When CAS needs the data in SingleStore, it will query the database to fetch it for that action.

With the SAS Embedded Process present, we get the multi-channel parallel data transfer that's fast and efficient.

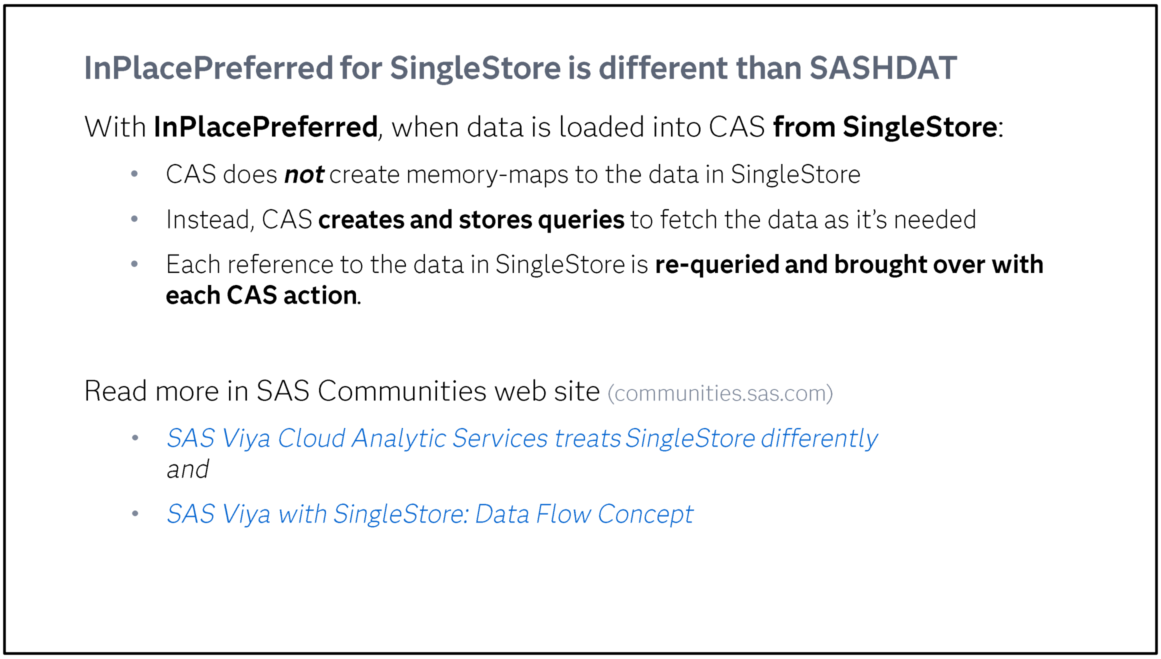
Notice the data blocks appear in memory on the CAS workers, but is not shown as utilizing the CAS_DISK_CACHE.



Once the data transfer has finished and the subsequent CAS action is complete, then…

CAS releases the data in memory from SingleStore that it was working with.

For any subsequent CAS actions, CAS will repeat the query and fetch the data again as needed.



InPlacePreferred for SingleStore is different than SASHDAT

With **InPlacePreferred**, when data is loaded into CAS **from SingleStore**:

- CAS does ***not*** create memory-maps to the data in SingleStore
- Instead, CAS **creates and stores queries** to fetch the data as it's needed
- Each reference to the data in SingleStore is **re-queried and brought over with each CAS action**.

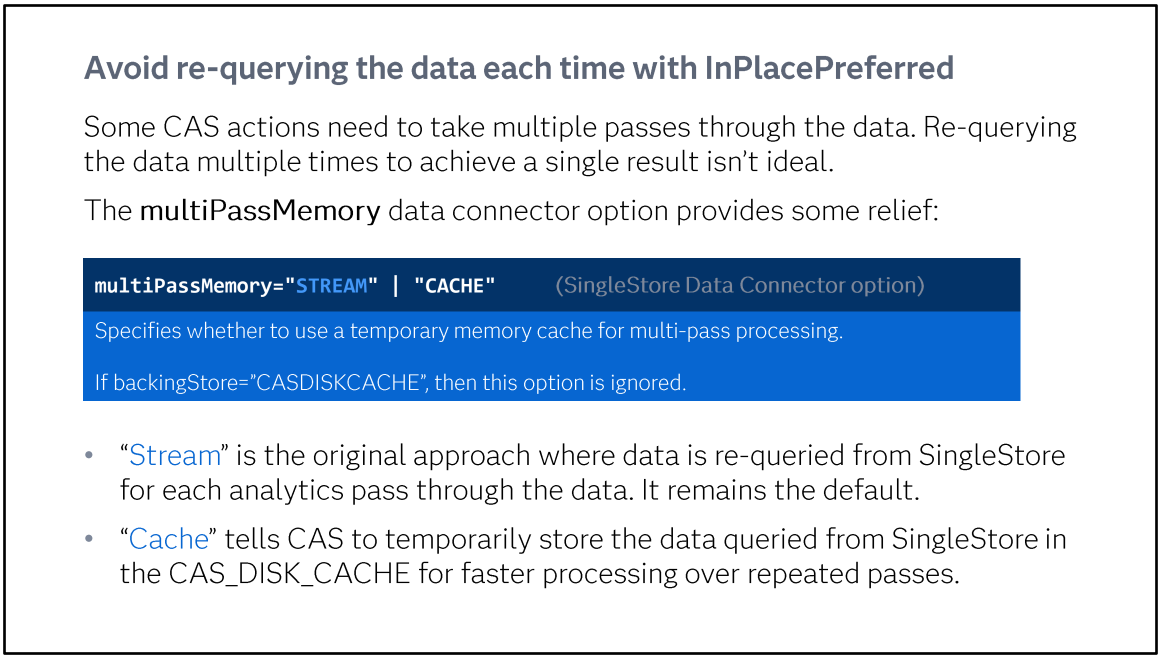
Read more in SAS Communities web site (communities.sas.com)

- *SAS Viya Cloud Analytic Services treats SingleStore differently* *and*
- *SAS Viya with SingleStore: Data Flow Concept*

The previous illustration showed how InPlacePreferred for SingleStore is different than memory-mapping with SASHDAT files.

With InPlacePreferred, when data is loaded into CAS from SingleStore, then CAS does not create memory-maps to the data in SingleStore. Instead, CAS creates and stores queries to fetch the data as it's needed. Each reference to the data in SingleStore is re-queried and brought over again with each CAS action.

For additional details about this behavior, you can read more in the SAS Communities web site at communities.sas.com. Look for posts titled, SAS Viya Cloud Analytic Services treats SingleStore differently by Rob Collum and SAS Viya with SingleStore: Data Flow Concept by Nicolas Robert.



Avoid re-querying the data each time with InPlacePreferred

Some CAS actions need to take multiple passes through the data. Re-querying the data multiple times to achieve a single result isn't ideal.

The **multiPassMemory** data connector option provides some relief:

```
multiPassMemory="STREAM" | "CACHE"
```

(SingleStore Data Connector option)

Specifies whether to use a temporary memory cache for multi-pass processing.

If backingStore="CASDISKCACHE", then this option is ignored.

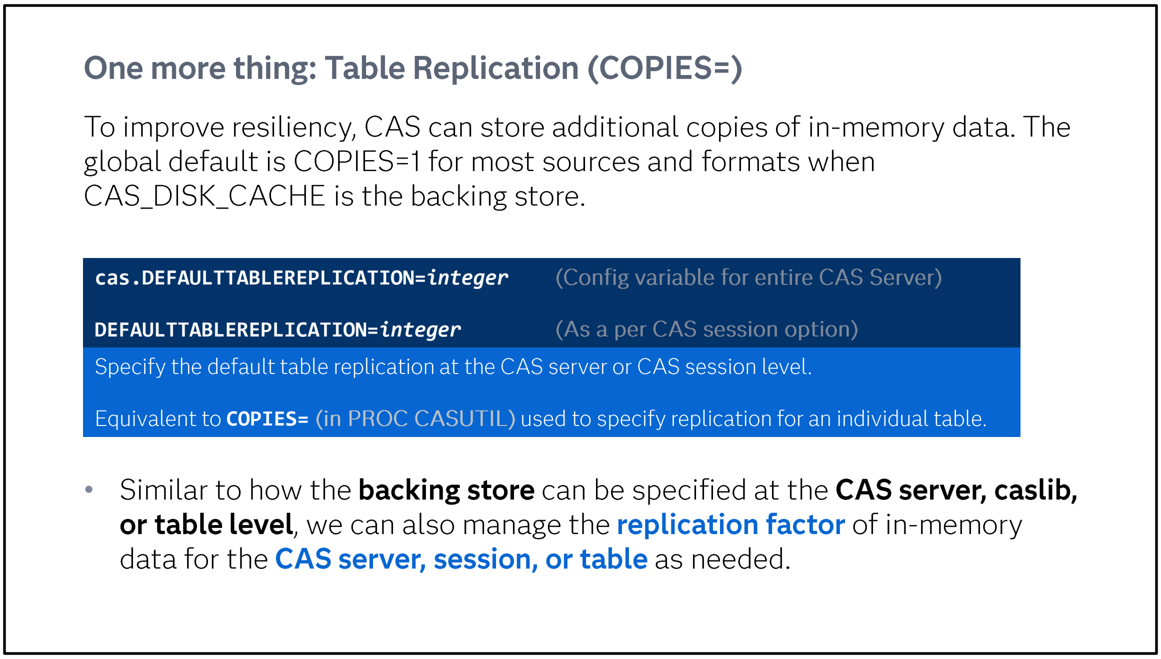
- "Stream" is the original approach where data is re-queried from SingleStore for each analytics pass through the data. It remains the default.
- "Cache" tells CAS to temporarily store the data queried from SingleStore in the CAS_DISK_CACHE for faster processing over repeated passes.

One advantage of the InPlacePreferred approach is that any changes to the source data in SingleStore are automatically picked up during the next CAS action execution, thanks to the use of repeated queries. However, some CAS actions require multiple passes over the data to produce a single result. In these cases, re-querying can be inefficient and may introduce errors if the data changes between passes. To address this, when using a caslib with a source type of SingleStore, you can enable the MultiPassMemory option to optimize query behavior and reduce unnecessary data re-access.

The default value is Stream which we've described generally as repeated queries.

Alternatively, a value of Cache will direct CAS to temporarily store the query results from SingleStore in CAS_DISK_CACHE for faster processing of analytics that take multiple passes at the data.

Note that this option is ignored if the backing store is already CASDISKCACHE for the table in question.



One more thing: Table Replication (COPIES=)

To improve resiliency, CAS can store additional copies of in-memory data. The global default is COPIES=1 for most sources and formats when CAS_DISK_CACHE is the backing store.

`cas.DEFAULTTABLEREPLICATION=`*`integer`* (Config variable for entire CAS Server)

`DEFAULTTABLEREPLICATION=`*`integer`* (As a per CAS session option)

Specify the default table replication at the CAS server or CAS session level.

Equivalent to **`COPIES=`** (in PROC CASUTIL) used to specify replication for an individual table.

- Similar to how the **backing store** can be specified at the **CAS server, caslib, or table level**, we can also manage the **replication factor** of in-memory data for the **CAS server, session, or table** as needed.

While we’re talking about the backing store, there’s one more topic to touch on quickly: table replication.

When the CAS_DISK_CACHE is acting as the backing store, then CAS defaults to creating an additional copy of the table in the cache as a safeguard against Worker failure. That way, if one of the CAS Workers crashes, suffers a network disconnection, or otherwise appears offline, the remaining workers will still have a complete copy of the table to work with – so you don’t have to reload it from scratch.

Usually, 1 additional copy protects against the loss of 1 worker, 2 copies for 2 workers, and so on. While any integer is technically acceptable, realistically this value is most often 0, 1, or 2. CAS is designed to work with very large volumes of data, so keep that in mind when setting the replication factor.

As we saw for specifying the backing store, you can control table replication at three levels: for the entire CAS server, for a specific CAS session (and all tables therein), or for an individual table.

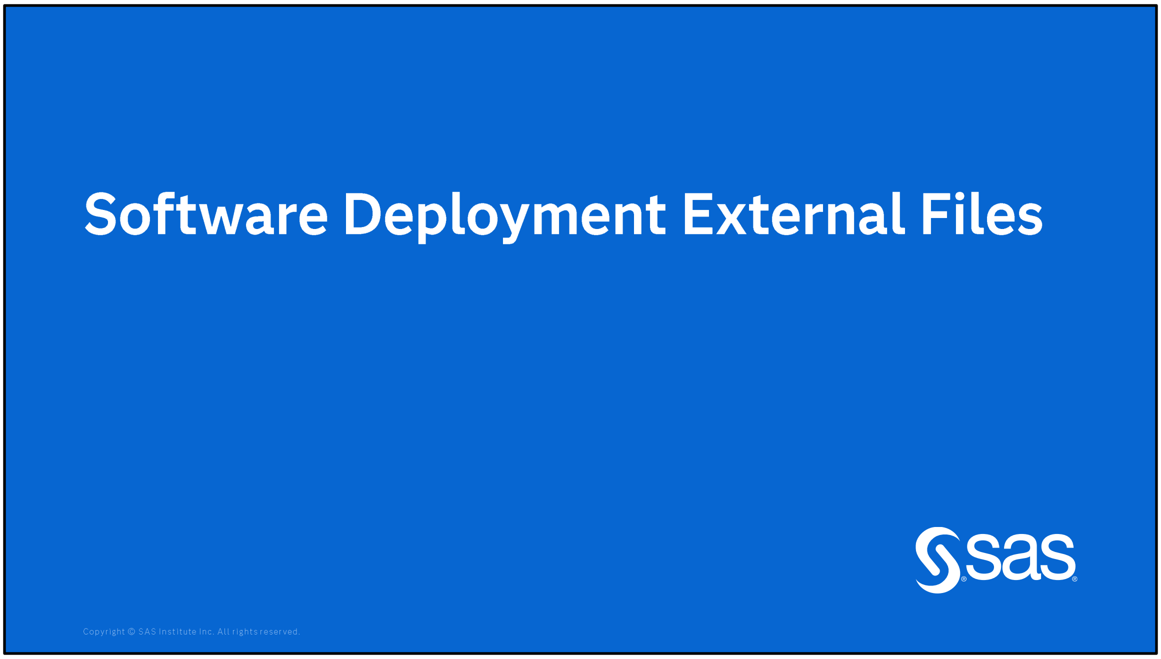
This is helpful because some tables are short-lived, small, or otherwise very temporary so maybe they don’t need replication. Other tables might require a lot of work to get into their final form, so perhaps they need extra failure protection.

The cas.DEFAULTTABLEREPLICATION configuration variable can be set using SAS Environment Manager to establish the default replication for the entire CAS Server. The default value is typically 1.

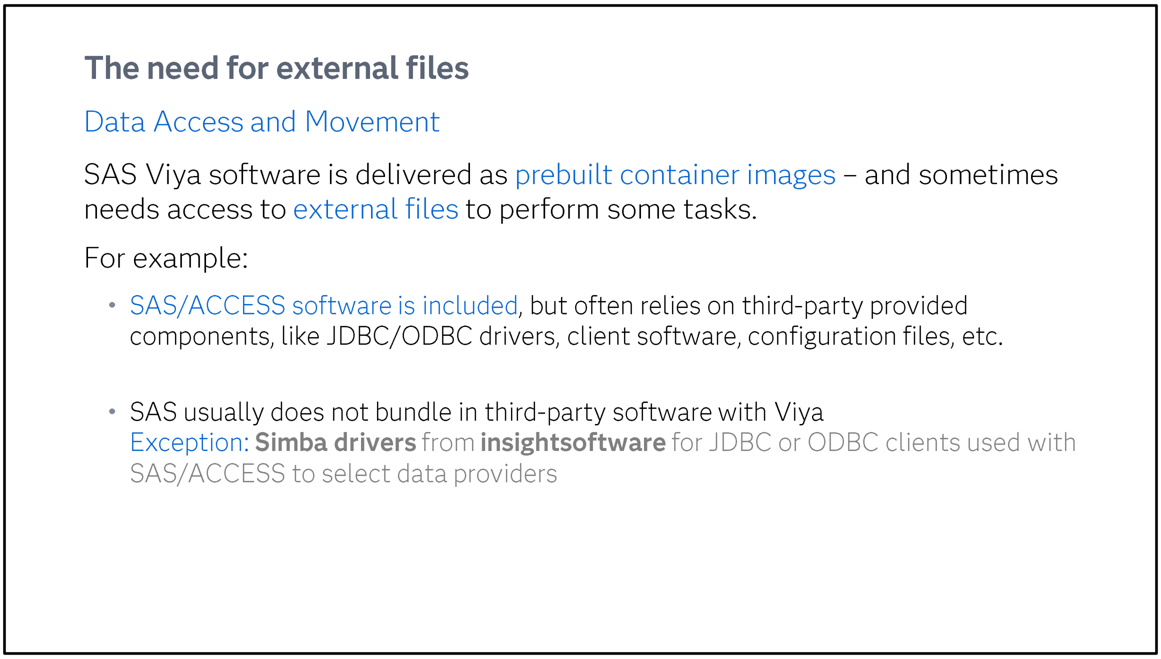
To set table replication for a given CAS session, specify the DEFAULTTABLEREPLICATION session option.

And finally, you can control the replication factor for any specific table using the COPIES= option in

PROC CASUTIL.



Software Deployment External Files



The need for external files

Data Access and Movement

SAS Viya software is delivered as prebuilt container images – and sometimes needs access to external files to perform some tasks.

For example:

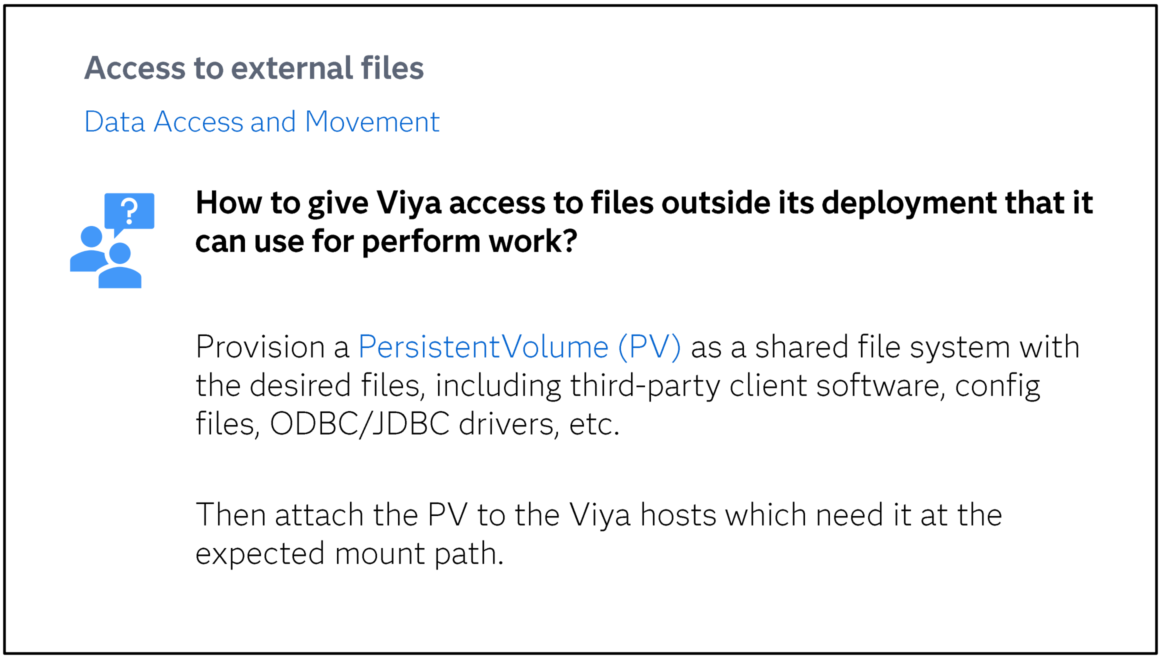
- SAS/ACCESS software is included, but often relies on third-party provided components, like JDBC/ODBC drivers, client software, configuration files, etc.

- SAS usually does not bundle in third-party software with Viya
  Exception: **Simba drivers** from **insightsoftware** for JDBC or ODBC clients used with SAS/ACCESS to select data providers

SAS Viya software is delivered as prebuilt container images – and sometimes needs access to external files to perform some tasks.

For example, SAS/ACCESS software is included but often relies on third-party provided components, like JDBC and ODBC drivers, client software, configuration files, and so on.

Combine this with the information that SAS usually does not bundle in third-party software with Viya with the notable exception that we do include the associated Simba drivers from Insight Software for JDBC or ODBC clients used with SAS/ACCESS software to select data providers.



Access to external files

Data Access and Movement

How to give Viya access to files outside its deployment that it can use for perform work?

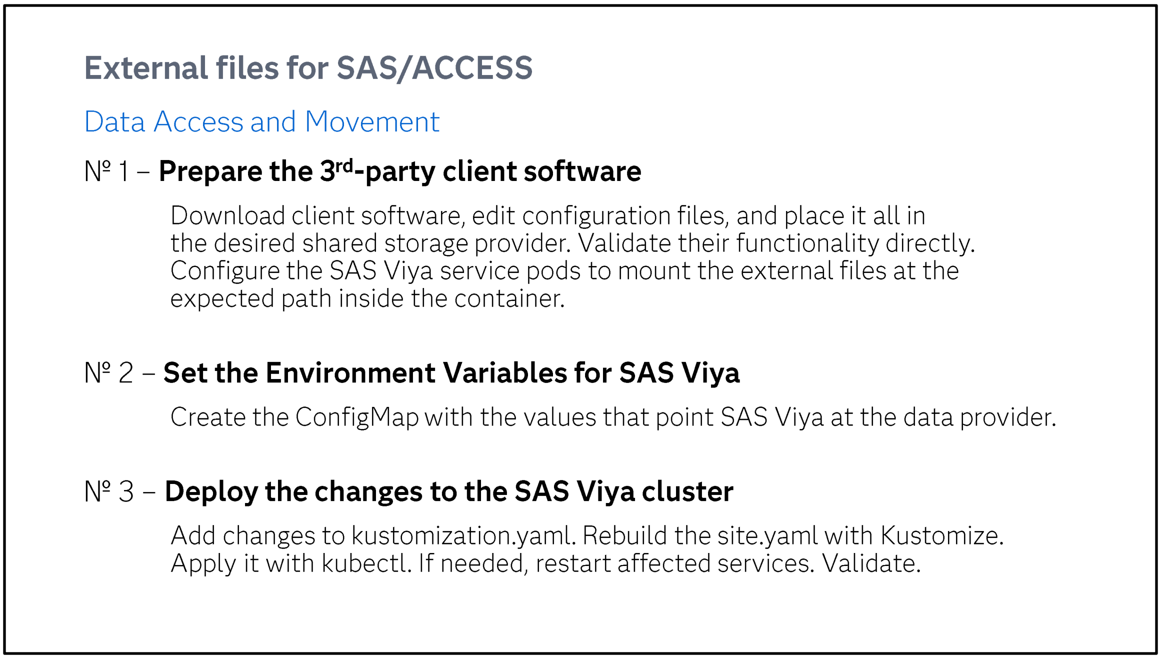
Provision a PersistentVolume (PV) as a shared file system with the desired files, including third-party client software, config files, ODBC/JDBC drivers, etc.

Then attach the PV to the Viya hosts which need it at the expected mount path.

Realizing that there might be files outside the Viya deployment that it needs, how can we set up Viya to reach them?

One common approach is to provision a persistent volume (or PV) as a shared file system with the desired files, including third-party client software, config files, ODBC and JDBC drivers, and anything else that's needed.

Then configure Viya to attach the persistent volume to the Viya hosts which need it at the expected mount path.



External files for SAS/ACCESS

Data Access and Movement

№ 1 – **Prepare the 3rd-party client software**

Download client software, edit configuration files, and place it all in the desired shared storage provider. Validate their functionality directly. Configure the SAS Viya service pods to mount the external files at the expected path inside the container.

№ 2 – **Set the Environment Variables for SAS Viya**

Create the ConfigMap with the values that point SAS Viya at the data provider.

№ 3 – **Deploy the changes to the SAS Viya cluster**

Add changes to kustomization.yaml. Rebuild the site.yaml with Kustomize. Apply it with kubectl. If needed, restart affected services. Validate.

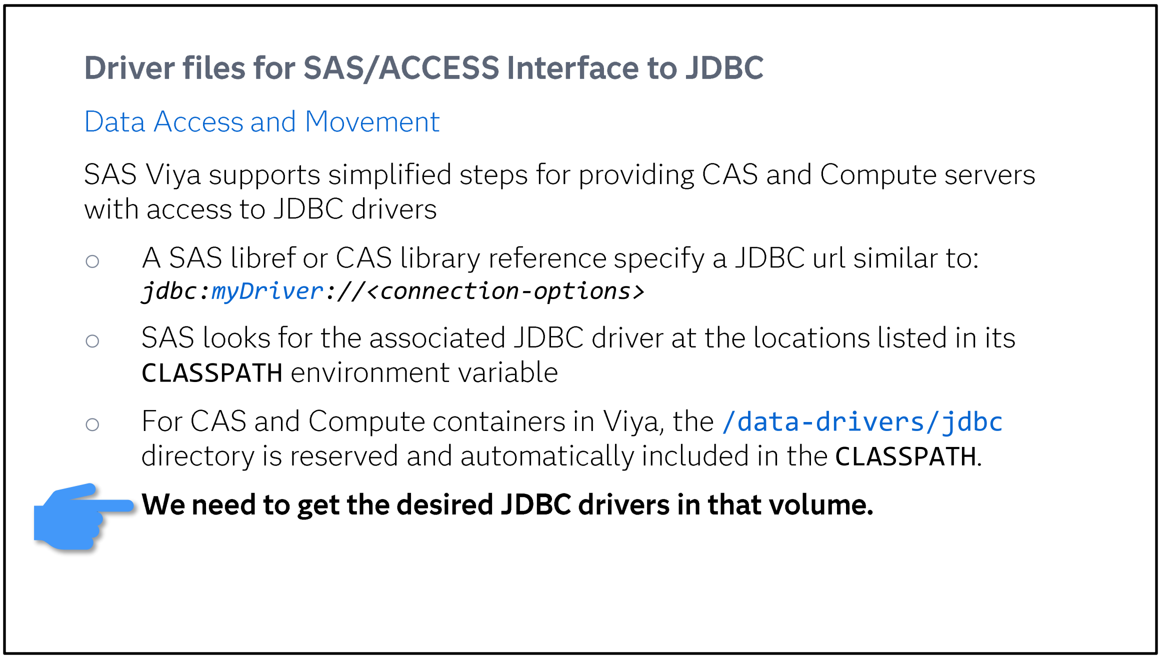
To make this point, let's look at the example of attaching external files to Viya in support of the SAS/ACCESS software.

The first thing to do is prepare the third-party client software. In other words, download client software, edit its configuration files, and place it all in the desired shared storage provider. You'll want to validate the functionality of the third-party client directly – that is a lot simpler than trying to figure out what's wrong with SAS running on top of it. When you've got the third-party components working together correctly, then configure the SAS Viya service pods to mount the external files at the expected path inside the relevant Viya containers.

Another step might be to configure Viya with additional environment variables, if needed. That's typically managed by creating a config map with the values that point SAS Viya at the data provider per its requirements.

The third step here is to apply these changes to Viya in the Kubernetes cluster. That usually consists of making changes to kustomization.yaml. Rebuilding the site.yaml with Kustomize. And applying the updated site manifest with kubectl. If needed, restart affected services to pickup these changes. Then validate that Viya is operating as intended.

That's pretty high-level so far, so let's look at a more specific use-case next.



Driver files for SAS/ACCESS Interface to JDBC

Data Access and Movement

SAS Viya supports simplified steps for providing CAS and Compute servers with access to JDBC drivers

- A SAS libref or CAS library reference specify a JDBC url similar to: `jdbc:myDriver://<connection-options>`
- SAS looks for the associated JDBC driver at the locations listed in its `CLASSPATH` environment variable
- For CAS and Compute containers in Viya, the `/data-drivers/jdbc` directory is reserved and automatically included in the `CLASSPATH`.

We need to get the desired JDBC drivers in that volume.

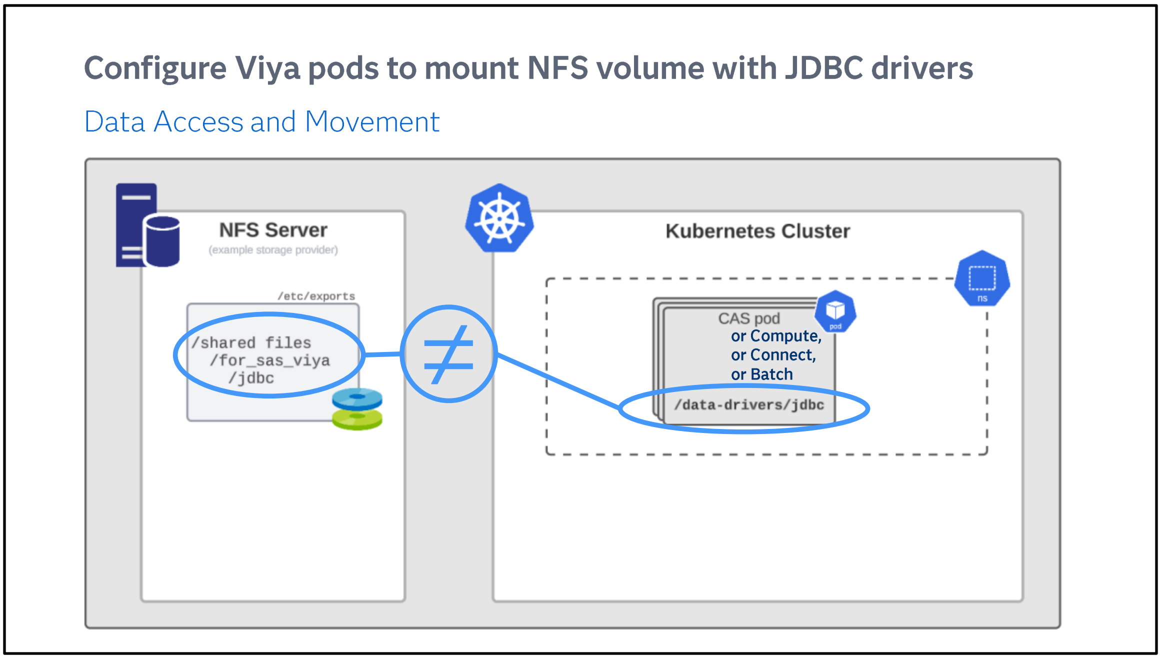
SAS Viya supports simplified steps for providing CAS and Compute servers with access to JDBC drivers.

The scenario we're considering is to support a SAS programmer who uses JDBC references in their code as part of their SAS libref or CAS library reference definitions.

Based on the url specified, SAS looks for the associated JDBC driver at the locations listed in its CLASSPATH environment variable.

For CAS and Compute containers in Viya, the "/data-drivers/jdbc" directory is reserved and automatically included in the CLASSPATH.

So, we need to get the desired JDBC drivers in that volume.



Configure Viya pods to mount NFS volume with JDBC drivers

Data Access and Movement

In this illustration, we show an NFS Server on the left and Kubernetes with SAS Viya on the right. On the Viya side, we're focused on a group of CAS pods – the controllers and workers of an MPP CAS Server.

Note that this example also works for the Compute,

Connect, and

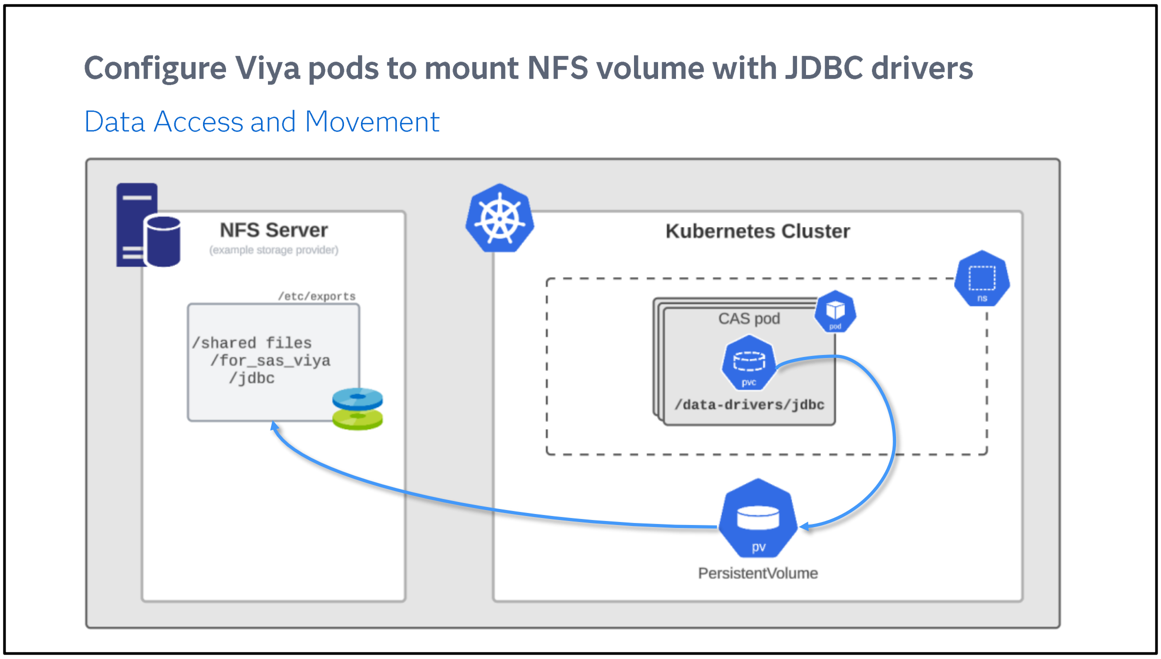
Batch Server pods as well. But for now, we'll focus on CAS.

Notice that the path "/data-drivers/jdbc" is defined in the CAS pod – or more specifically, the CAS container in the CAS pod. This applies both to CAS controllers as well as CAS workers.

Looking back at the NFS Server – which acts as a service to share files to multiple hosts at the same time – we have defined a path to store some JDBC drivers. To be clear, an NFS Server is not the only way to share files with multiple pods – but it's a common use-case.

Notice as well that those paths don't have to match each other.

But we do need to connect them together so that the CAS process - or Compute Server process - can find the JDBC drivers it needs.



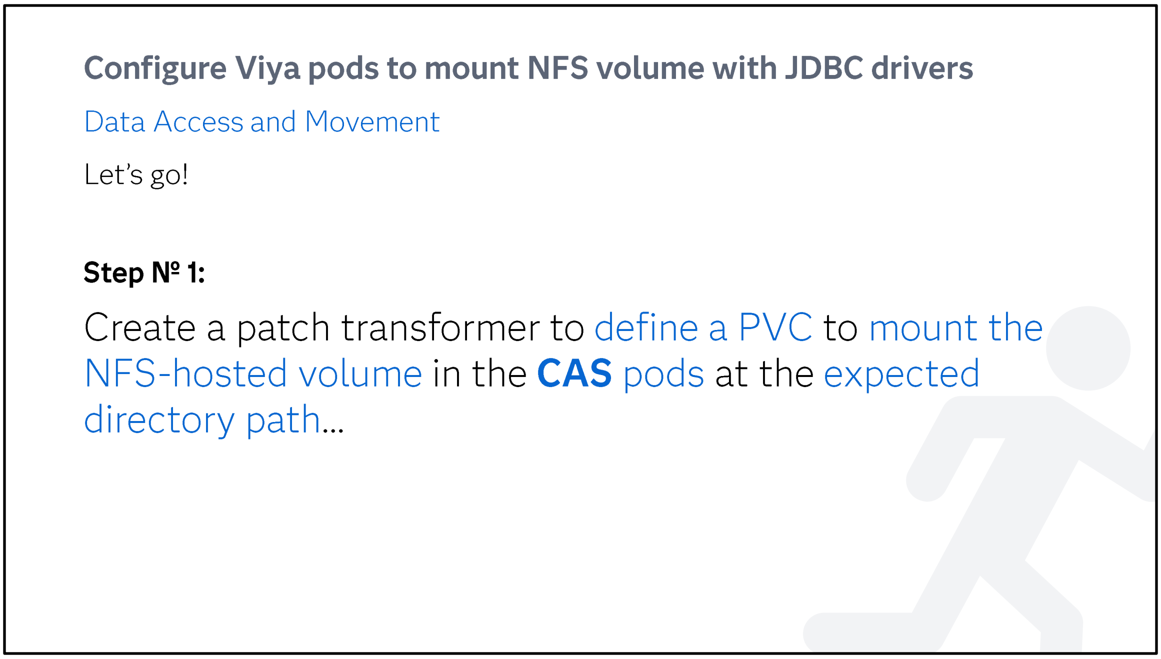
Configure Viya pods to mount NFS volume with JDBC drivers

Data Access and Movement

Kubernetes provides numerous mechanisms to connect pods with different storage options.

In this example, we'll define a persistent volume claim in the CAS pod which invokes a persistent volume.

The persistent volume references the location of our JDBC driver storage over on the NFS Server.



Configure Viya pods to mount NFS volume with JDBC drivers

Data Access and Movement

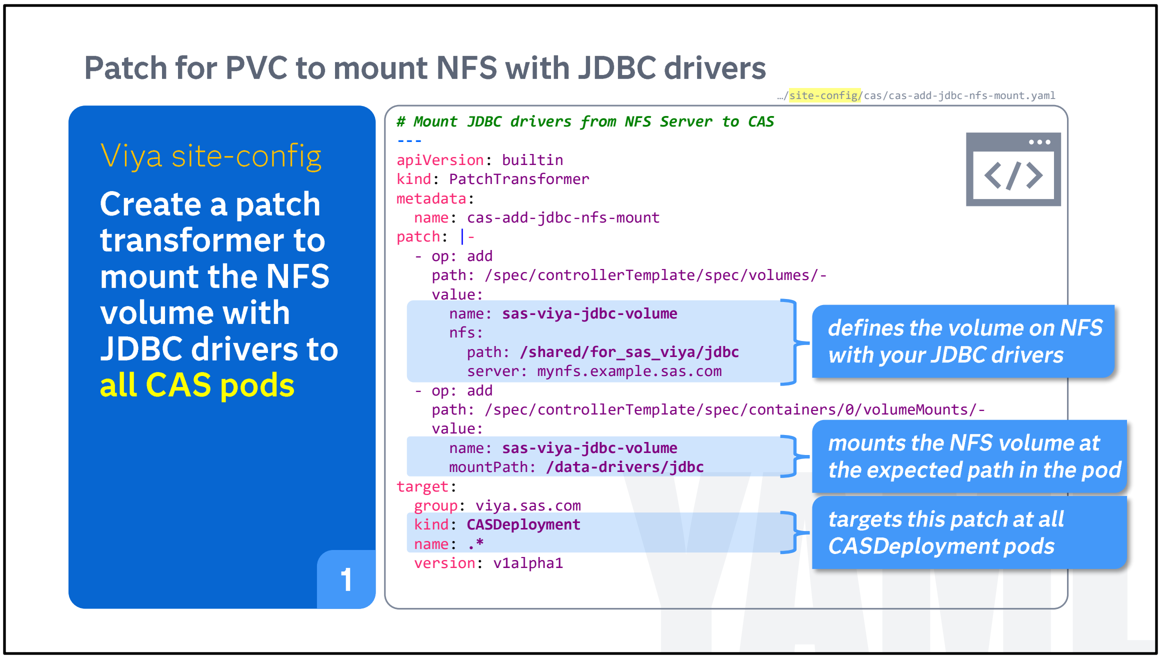
Let’s go!

Step № 1:

Create a patch transformer to define a PVC to mount the NFS-hosted volume in the **CAS** pods at the expected directory path…

To accomplish this goal, we need to run through a few steps of configuration and deployment.

The first step is to create a patch transformer to define a PVC to mount the NFS-hosted volume in the CAS pods at the expected directory path. We’ll do that by creating a patch transformer that Kustomize will use to build or update Viya’s site manifest file to tell Kubernetes what to provision in the environment.



Patch for PVC to mount NFS with JDBC drivers

Viya site-config

Create a patch transformer to mount the NFS volume with JDBC drivers to all CAS pods

1

…/site-config/cas/cas-add-jdbc-nfs-mount.yaml

```
# Mount JDBC drivers from NFS Server to CAS
---
apiVersion: builtin
kind: PatchTransformer
metadata:
  name: cas-add-jdbc-nfs-mount
patch: |-
  - op: add
    path: /spec/controllerTemplate/spec/volumes/-
    value:
      name: sas-viya-jdbc-volume
      nfs:
        path: /shared/for_sas_viya/jdbc
        server: mynfs.example.sas.com
  - op: add
    path: /spec/controllerTemplate/spec/containers/0/volumeMounts/-
    value:
      name: sas-viya-jdbc-volume
      mountPath: /data-drivers/jdbc
target:
  group: viya.sas.com
  kind: CASDeployment
  name: .*
  version: v1alpha1
```

defines the volume on NFS with your JDBC drivers

mounts the NFS volume at the expected path in the pod

targets this patch at all CASDeployment pods

Shown here is an example patch transformer file to update CAS pods. As you can see, it’s written in YAML format.

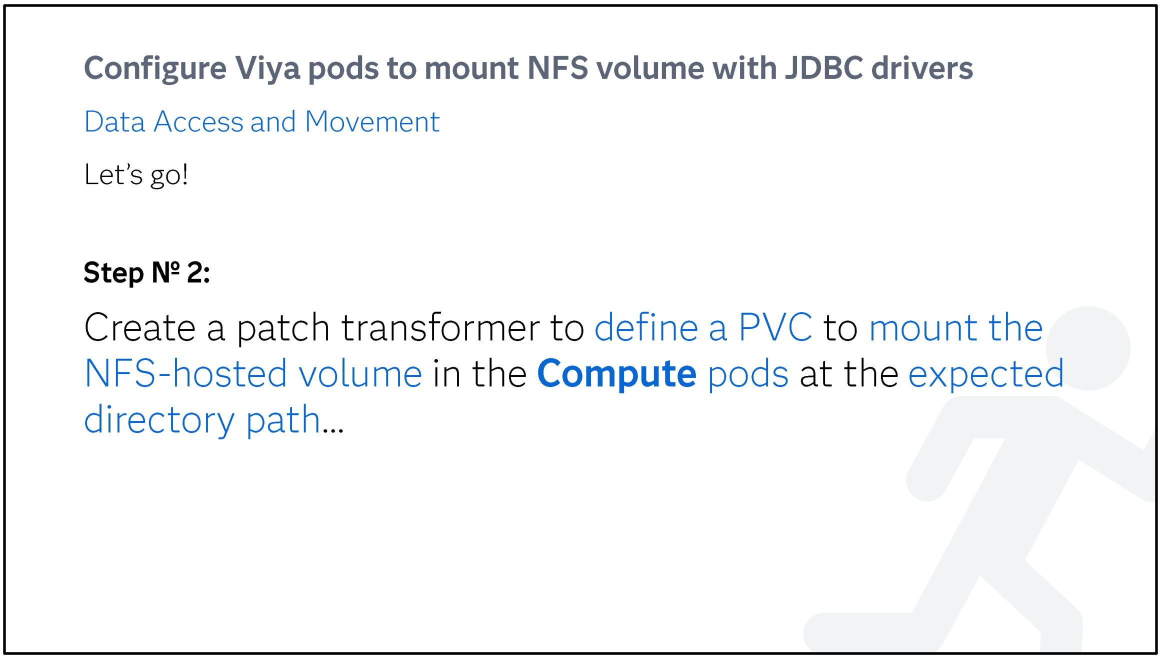
Our convention for configuring Viya deployments is to place files like this in the “site-config” directory on the host where our Viya deployment assets are kept.

We won’t go through this line-by-line, but there are some key areas to note.

First is the definition of the volume on the NFS Server. We give it a name – SAS Viya JDBC Volume – and define it as an NFS volume on the machine specified at the path where the JDBC drivers are being stored.

Next is the location in the CAS container where we want to mount that SAS Viya JDBC Volume. Notice it’s at that reserved path “/data-drivers/jdbc”.

And last is where we direct Kustomize to target this patch to apply to all pods of kind “CAS Deployment”. This will include all CAS controllers and all CAS workers.



Configure Viya pods to mount NFS volume with JDBC drivers

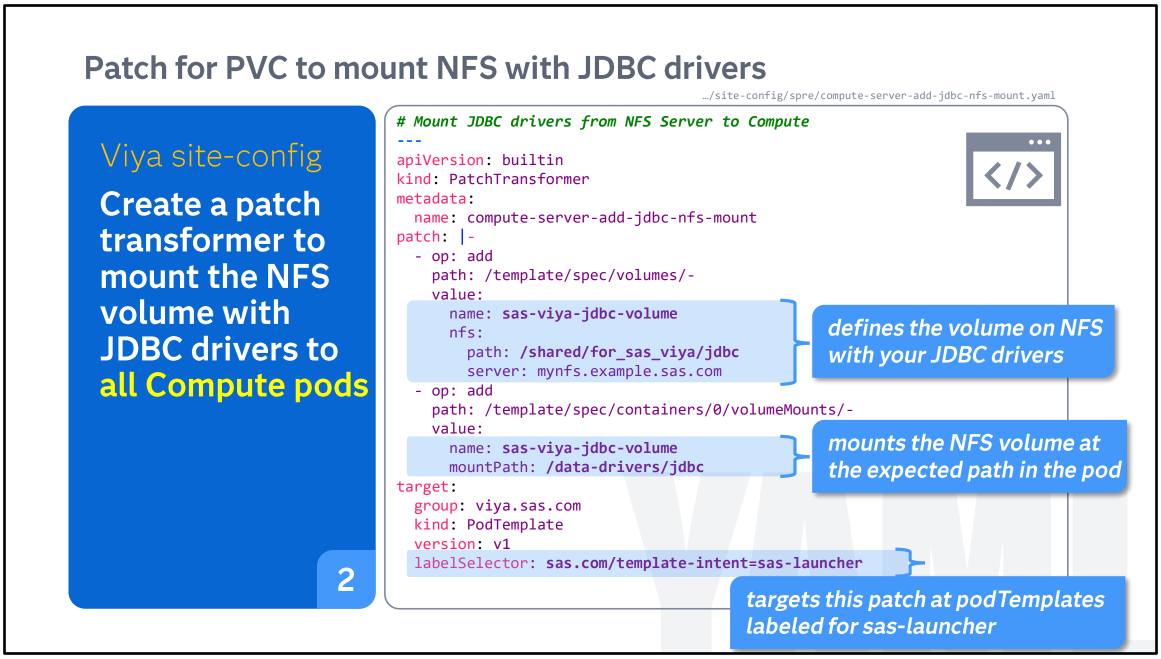
Data Access and Movement

Let’s go!

Step № 2:

Create a patch transformer to define a PVC to mount the NFS-hosted volume in the **Compute** pods at the expected directory path…

The next step is mostly the same as the first. But now, instead of targeting CAS, we will create a patch transformer to define a PVC to mount the NFS-hosted volume in the Compute pods at the expected directory path.



Patch for PVC to mount NFS with JDBC drivers

Viya site-config

Create a patch transformer to mount the NFS volume with JDBC drivers to all Compute pods

2

…/site-config/spre/compute-server-add-jdbc-nfs-mount.yaml

```
# Mount JDBC drivers from NFS Server to Compute
---
apiVersion: builtin
kind: PatchTransformer
metadata:
  name: compute-server-add-jdbc-nfs-mount
patch: |-
  - op: add
    path: /template/spec/volumes/-
    value:
      name: sas-viya-jdbc-volume
      nfs:
        path: /shared/for_sas_viya/jdbc
        server: mynfs.example.sas.com
  - op: add
    path: /template/spec/containers/0/volumeMounts/-
    value:
      name: sas-viya-jdbc-volume
      mountPath: /data-drivers/jdbc
target:
  group: viya.sas.com
  kind: PodTemplate
  version: v1
  labelSelector: sas.com/template-intent=sas-launcher
```

defines the volume on NFS with your JDBC drivers

mounts the NFS volume at the expected path in the pod

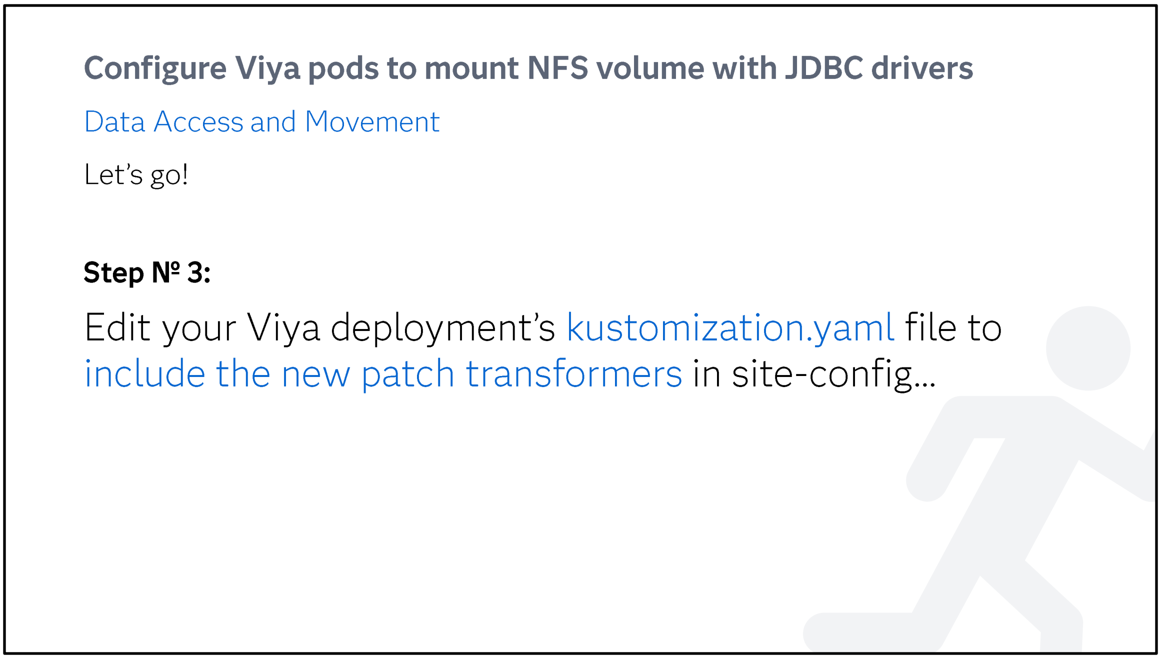
targets this patch at podTemplates labeled for sas-launcher

This is very similar to what we just created for CAS, but with some slight differences for Compute Servers.

First is the definition of the volume on the NFS Server. We give it a name – SAS Viya JDBC Volume – and define it as an NFS volume on the machine specified at the path where the JDBC drivers are being stored.

Next is the location in the Compute Server container where we want to mount that SAS Viya JDBC Volume. Notice it’s at that reserved path “/data-drivers/jdbc” – same as for CAS.

And third is where we direct Kustomize to target this patch to apply to pod templates that reference the SAS Launcher label shown here.



Configure Viya pods to mount NFS volume with JDBC drivers

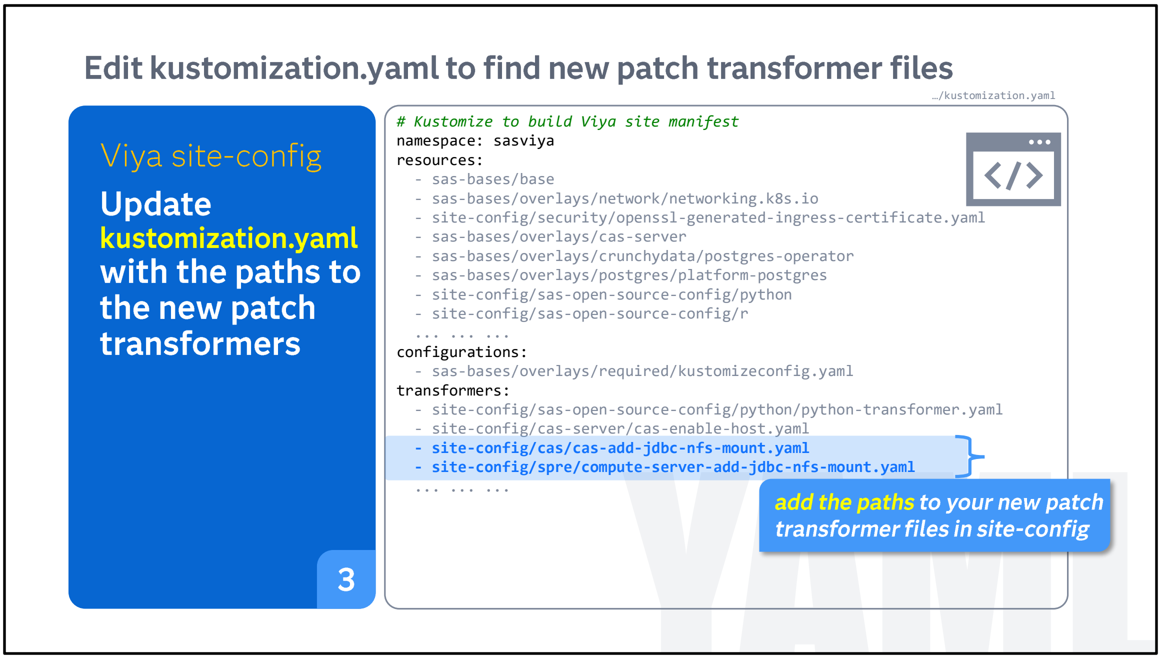
Data Access and Movement

Let’s go!

Step № 3:

Edit your Viya deployment’s kustomization.yaml file to include the new patch transformers in site-config…

Step three is to edit your Viya deployment’s kustomization.yaml file to include the new patch transformer files that have been placed in the site-config directory.



Edit kustomization.yaml to find new patch transformer files

Viya site-config

Update kustomization.yaml with the paths to the new patch transformers

3

…/kustomization.yaml

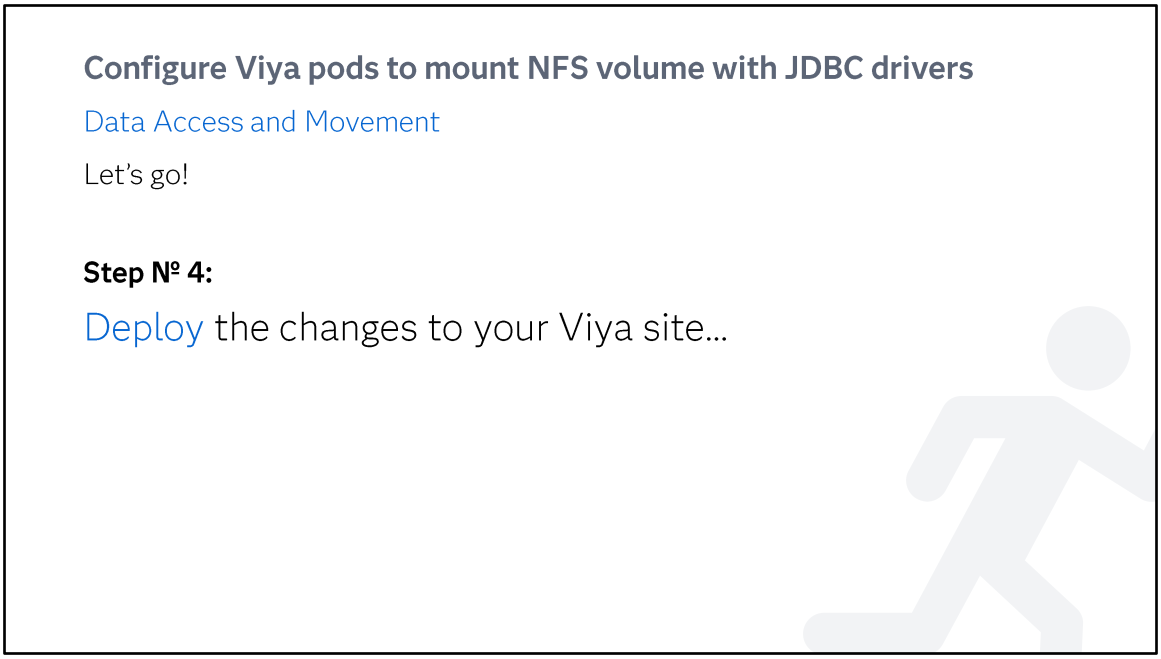
```
# Kustomize to build Viya site manifest
namespace: sasviya
resources:
  - sas-bases/base
  - sas-bases/overlays/network/networking.k8s.io
  - site-config/security/openssl-generated-ingress-certificate.yaml
  - sas-bases/overlays/cas-server
  - sas-bases/overlays/crunchydata/postgres-operator
  - sas-bases/overlays/postgres/platform-postgres
  - site-config/sas-open-source-config/python
  - site-config/sas-open-source-config/r
  ... ... ...
configurations:
  - sas-bases/overlays/required/kustomizeconfig.yaml
transformers:
  - site-config/sas-open-source-config/python/python-transformer.yaml
  - site-config/cas-server/cas-enable-host.yaml
  - site-config/cas/cas-add-jdbc-nfs-mount.yaml
  - site-config/spre/compute-server-add-jdbc-nfs-mount.yaml
  ... ... ...
```

add the paths to your new patch transformer files in site-config

Here we're looking at an abbreviated portion of the kustomization.yaml for a SAS Viya deployment. There are multiple sections to a kustomization file including directives to specify the applicable namespace, resources, configurations, and more.

For this scenario, we want to update the transformers section of kustomization.yaml.

At the bottom, we append two new lines – one for the patch transformer to add the NFS mount to the CAS pods and another line to do the same for the Compute Server pods.



Configure Viya pods to mount NFS volume with JDBC drivers

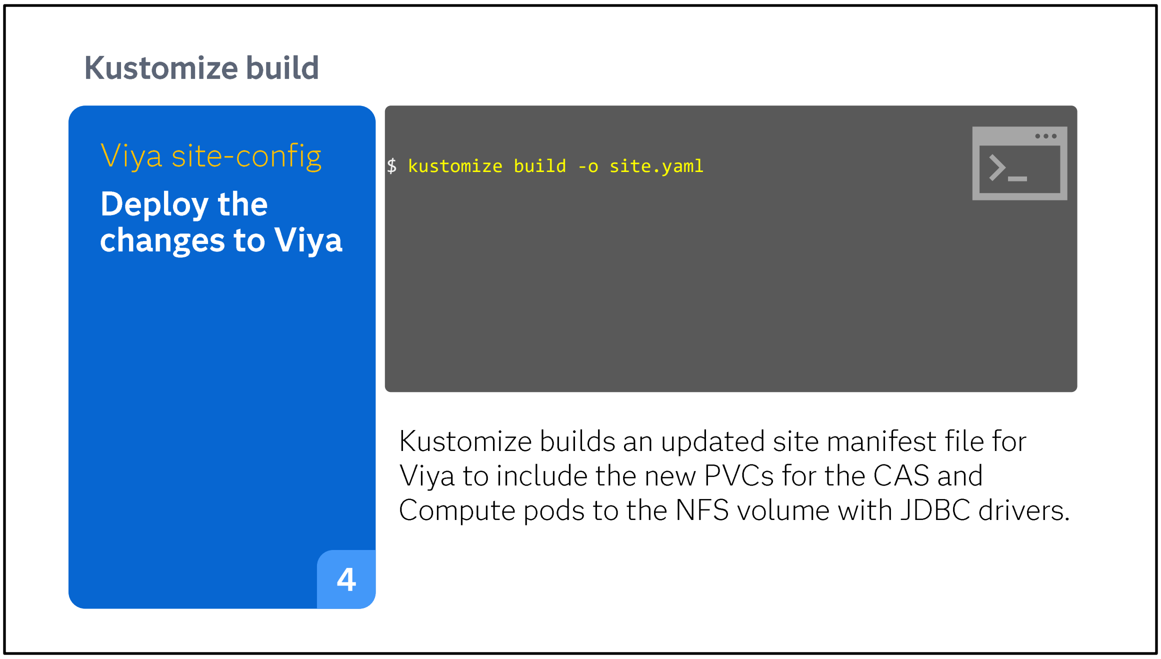
Data Access and Movement

Let’s go!

Step № 4:

Deploy the changes to your Viya site…

Now that we’ve placed JDBC drivers in an NFS server and created patch transformers to mount the associated volume to our CAS and Compute Server pods, we’re ready to deploy these changes to the SAS Viya environment.



Kustomize build

Viya site-config

Deploy the changes to Viya

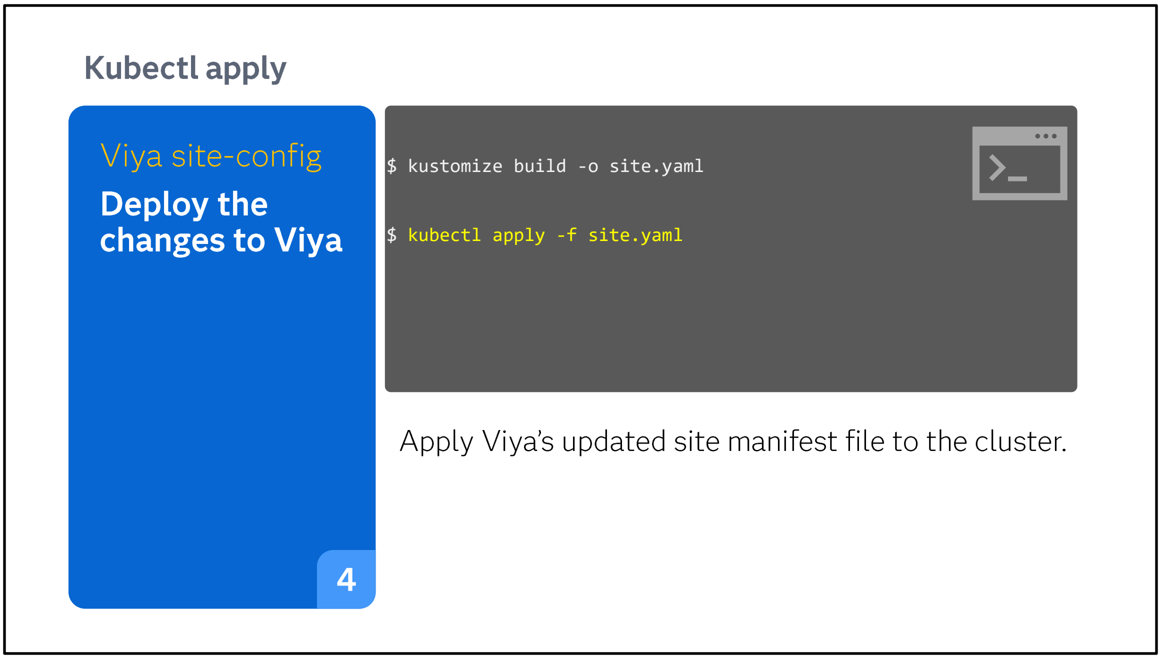
```
$ kustomize build -o site.yaml
```

Kustomize builds an updated site manifest file for Viya to include the new PVCs for the CAS and Compute pods to the NFS volume with JDBC drivers.

4

Instead of simply editing files, now we move on to issuing commands to put it all into play.

As you’ll recall, we need to run the Kustomize command with the build directive so that it will read in the updated kustomization.yaml file and output a new site manifest file for SAS Viya to include the new PVCs for the CAS and Compute pods to the NFS volume with JDBC drivers.



Kubectl apply

Viya site-config

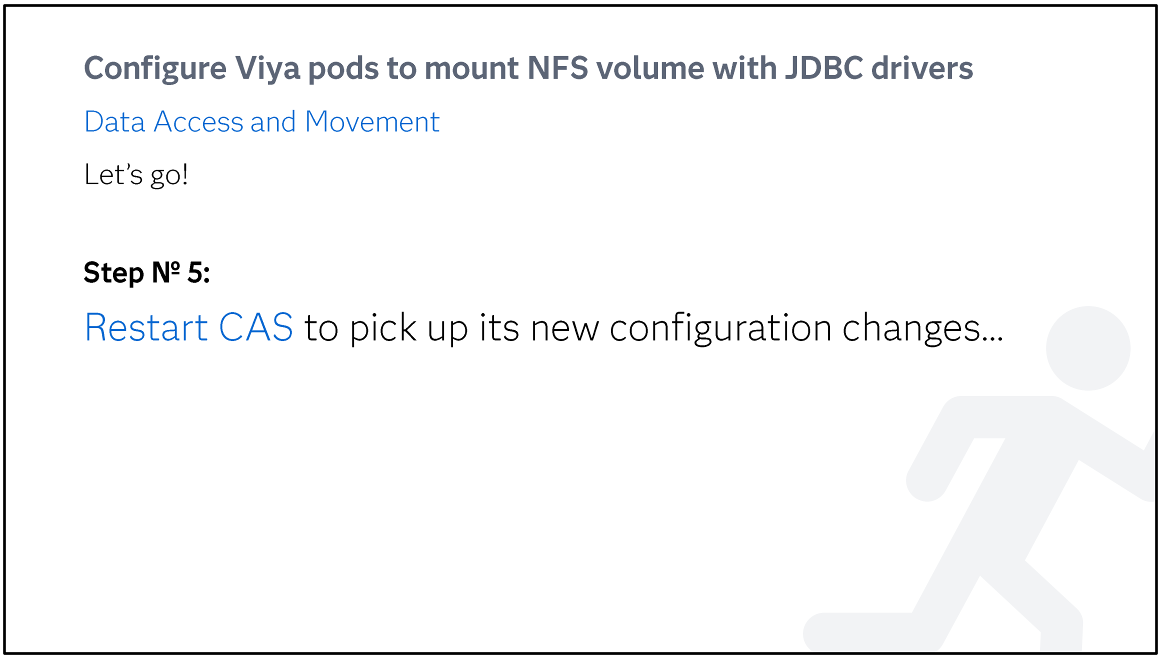
Deploy the changes to Viya

4

```
$ kustomize build -o site.yaml

$ kubectl apply -f site.yaml
```

Apply Viya’s updated site manifest file to the cluster.

Now that we have an updated site manifest for Viya, we can direct kubectl to apply it to the Kubernetes cluster.



Configure Viya pods to mount NFS volume with JDBC drivers

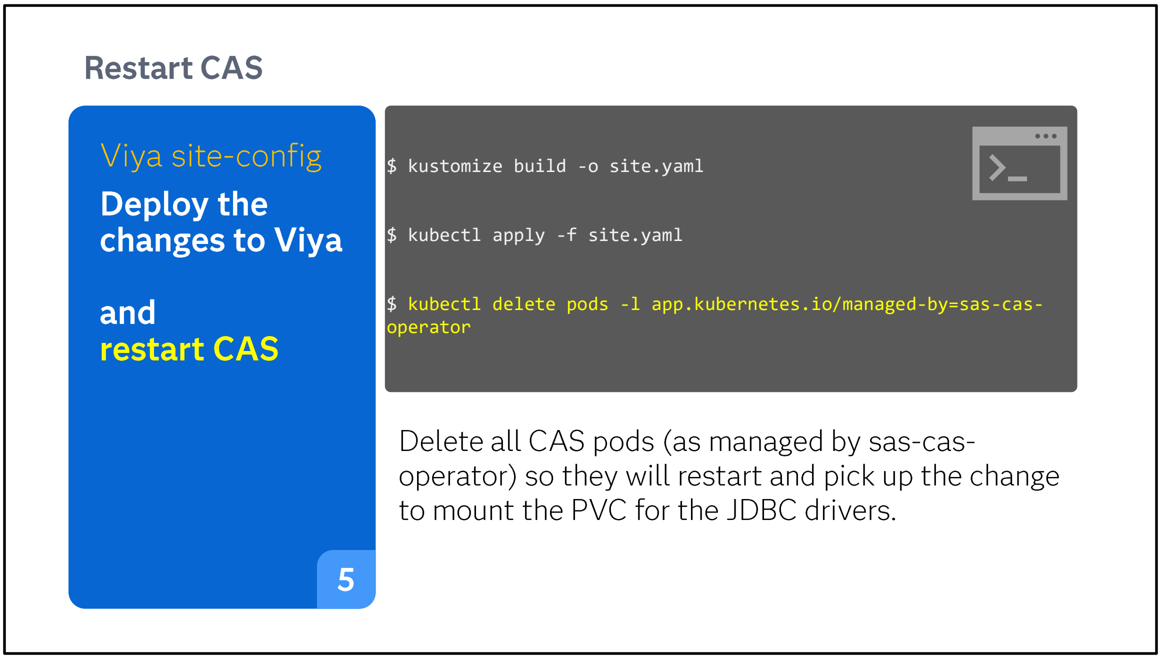
Data Access and Movement

Let’s go!

Step № 5:

Restart CAS to pick up its new configuration changes…

We’ve got one more task to perform. Even though the new configuration has been pushed to the Kubernetes cluster, the current instance of CAS that’s running isn’t aware of it. This is the kind of configuration that CAS processes when it’s first starting up. So we need to restart CAS.



Restart CAS

Viya site-config

Deploy the changes to Viya

and restart CAS

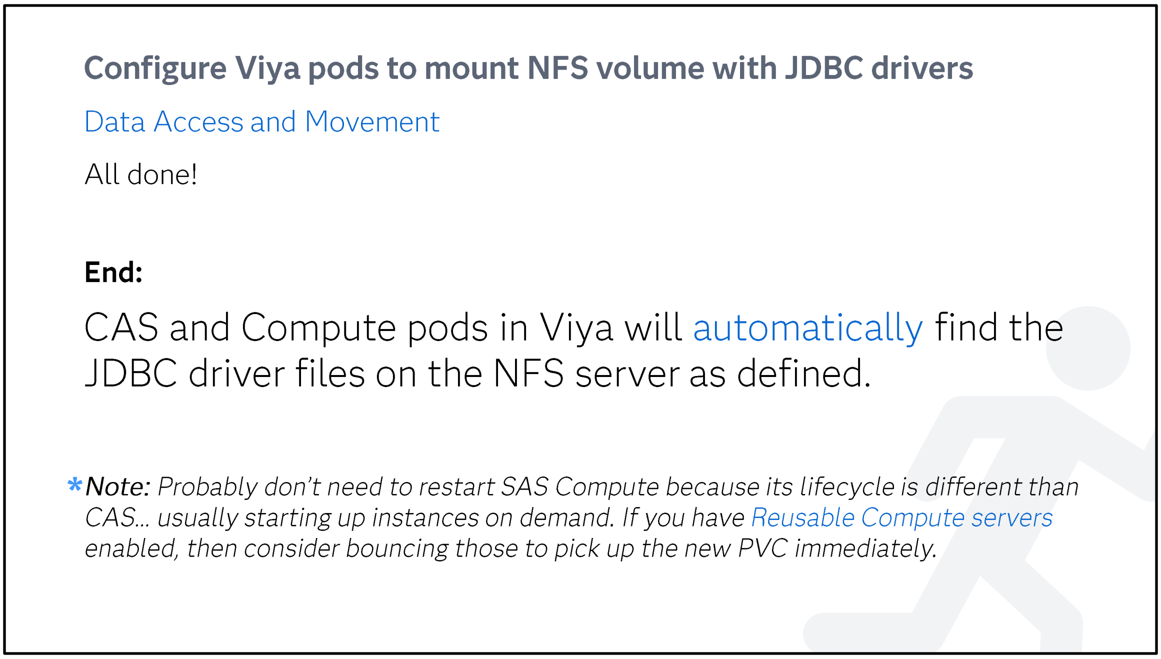
```
$ kustomize build -o site.yaml

$ kubectl apply -f site.yaml

$ kubectl delete pods -l app.kubernetes.io/managed-by=sas-cas-operator
```

Delete all CAS pods (as managed by sas-cas-operator) so they will restart and pick up the change to mount the PVC for the JDBC drivers.

5

This command tells Kubernetes to delete all pods managed by the CAS operator. This will effectively cause the CAS Server to restart from scratch.

Take care when planning this step. This wipes out all in-memory data and will interrupt any in-flight activities your users might have going on. So, treat this as an outage that should be planned to minimize interruption.



Configure Viya pods to mount NFS volume with JDBC drivers

Data Access and Movement

All done!

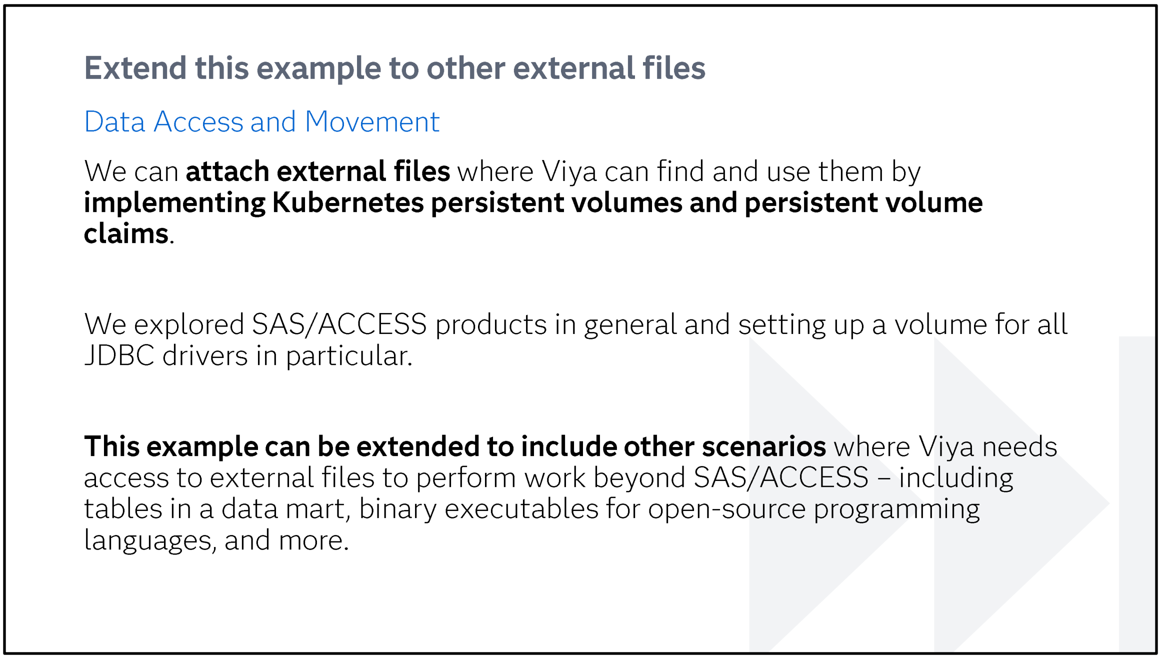
End:

CAS and Compute pods in Viya will automatically find the JDBC driver files on the NFS server as defined.

* *Note: Probably don't need to restart SAS Compute because its lifecycle is different than CAS… usually starting up instances on demand. If you have Reusable Compute servers enabled, then consider bouncing those to pick up the new PVC immediately.*

All done! At this point, CAS and Compute pods in Viya should automatically find the JDBC driver files at their location on the NFS server.

Notice that you probably won't need to restart SAS Compute pods as we did for CAS because its lifecycle is different. By default, Compute Servers usually start up instances on demand when requested. However, if you have configured Viya to enable Reusable Compute servers, then consider bouncing those to pick up the new PVC immediately instead of waiting for their eventual idle timeout.



Extend this example to other external files

Data Access and Movement

We can **attach external files** where Viya can find and use them by **implementing Kubernetes persistent volumes and persistent volume claims**.

We explored SAS/ACCESS products in general and setting up a volume for all JDBC drivers in particular.

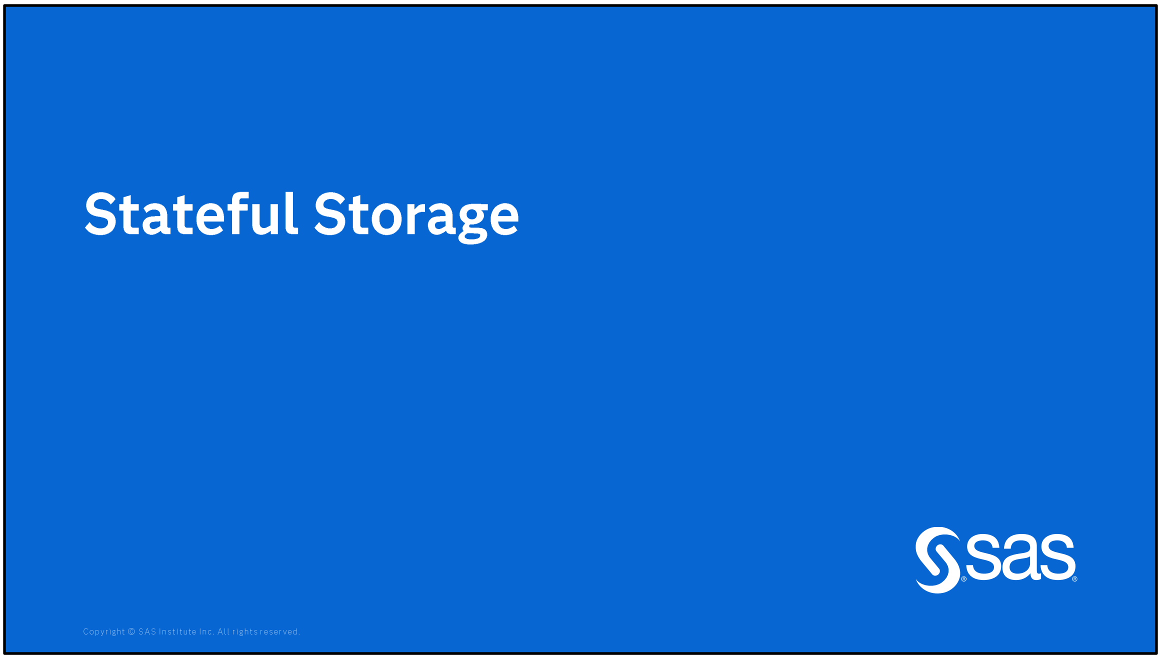
This example can be extended to include other scenarios where Viya needs access to external files to perform work beyond SAS/ACCESS – including tables in a data mart, binary executables for open-source programming languages, and more.

As shown, we can attach external files where Viya can find and use them by implementing Kubernetes persistent volumes and persistent volume claims.

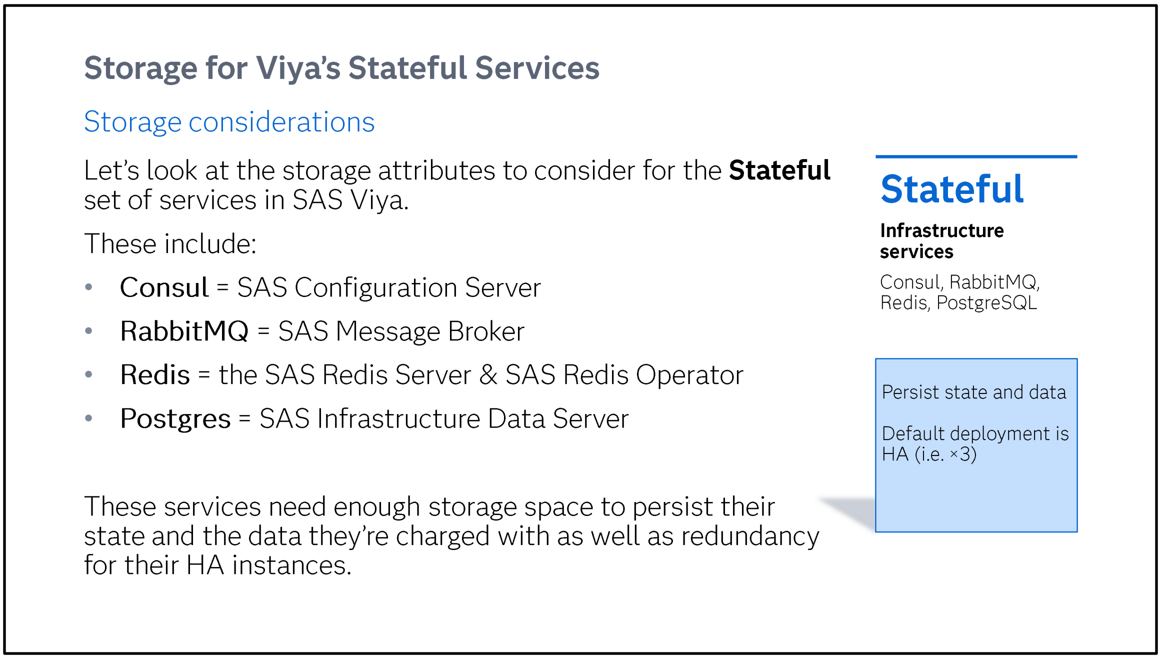
We explored this concept with SAS/ACCESS products in general and by setting up an NFS-hosted volume for all JDBC drivers in particular.

Beyond the SAS/ACCESS example demonstrated here, this process can be extended to include other scenarios where Viya needs access to external files to perform work – including tables in a data mart, binary executables for open-source programming languages, and more.

5.2 Storage Architecture



Stateful Storage



Storage for Viya’s Stateful Services

Storage considerations

Let’s look at the storage attributes to consider for the **Stateful** set of services in SAS Viya.

These include:

- Consul = SAS Configuration Server
- RabbitMQ = SAS Message Broker
- Redis = the SAS Redis Server & SAS Redis Operator
- Postgres = SAS Infrastructure Data Server

These services need enough storage space to persist their state and the data they’re charged with as well as redundancy for their HA instances.

Stateful

Infrastructure services

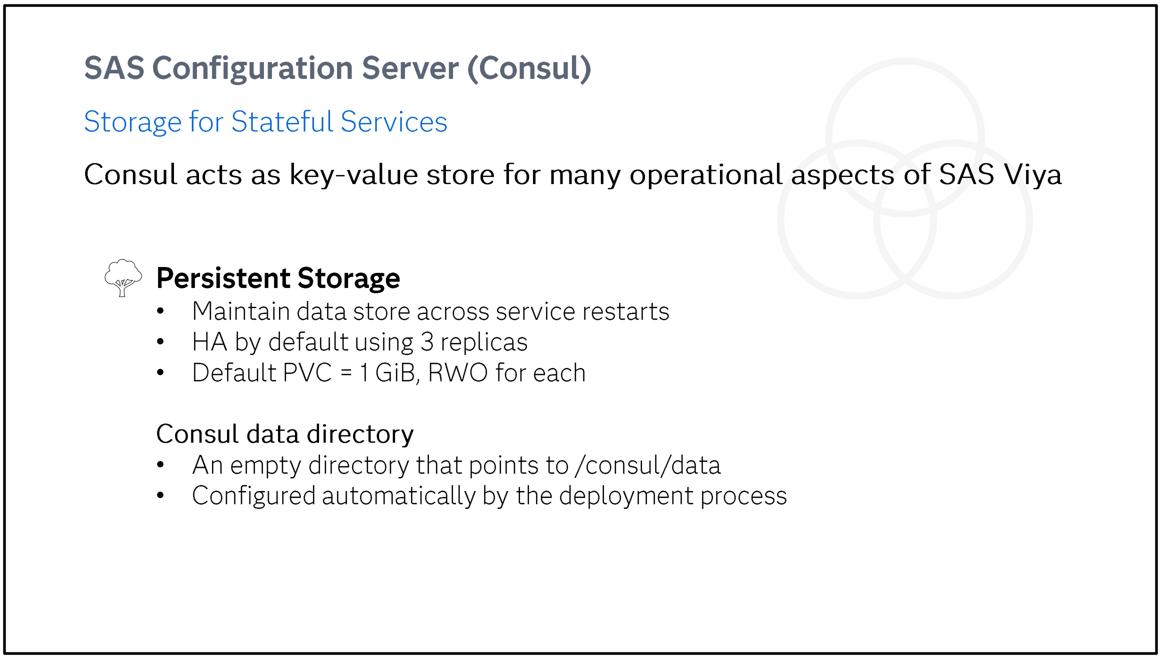
Consul, RabbitMQ, Redis, PostgreSQL

Persist state and data

Default deployment is HA (i.e. ×3)

Let’s look at the storage attributes to consider for the Stateful set of services in SAS Viya. These include Consul as part of the SAS Configuration Server, RabbitMQ as the SAS Message Broker, the SAS Redis Server, and Postgres for the SAS Infrastructure Data Server.

These services need enough storage space to persist their state and the data they’re charged with as well as redundancy for their HA instances.



SAS Configuration Server (Consul)

Storage for Stateful Services

Consul acts as key-value store for many operational aspects of SAS Viya

Persistent Storage

- Maintain data store across service restarts
- HA by default using 3 replicas
- Default PVC = 1 GiB, RWO for each

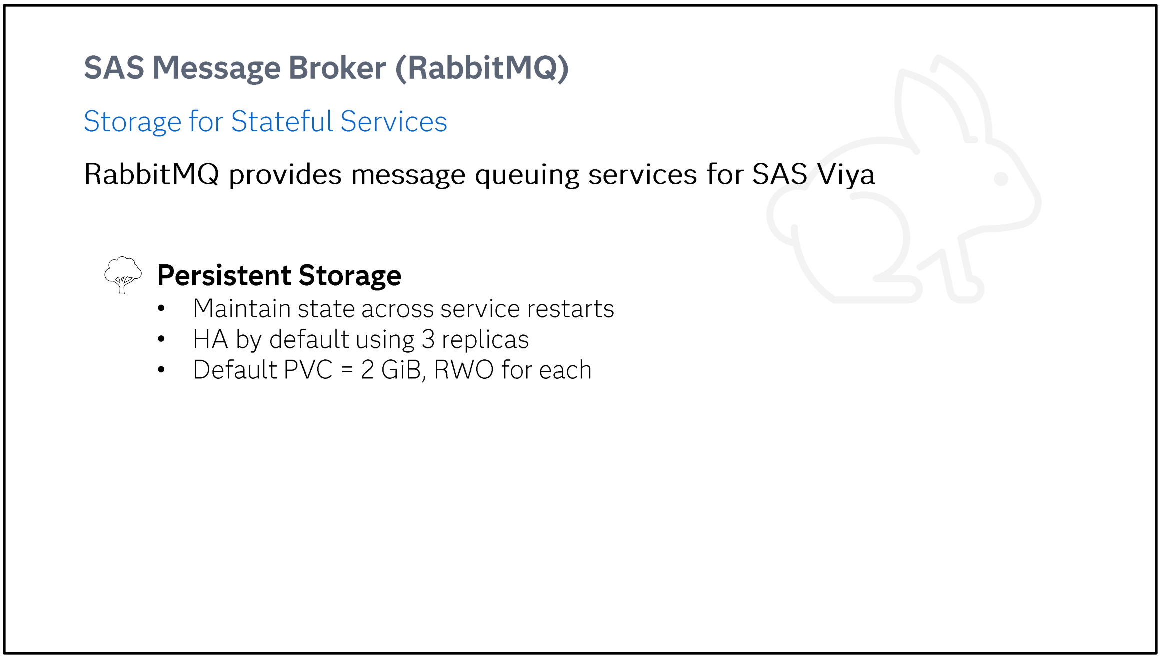
Consul data directory

- An empty directory that points to /consul/data
- Configured automatically by the deployment process

Consul acts as key-value store for many operational aspects of SAS Viya. It needs a place to store persistent data about the Viya configuration that can survive service restarts.

By default, Viya deploys Consul for high availability with three replicas running. Each replica will need its own storage volume to keep an independent copy of its own data. By default, Viya is configured to define a persistent volume claim of 1 gibabyte mounted RWO because the Consul replicas don’t share the physical disk storage. Note that 1 gibabyte per PVC is intended as a conservative size to start with and more storage might be needed for your site.

Consul also needs an emptyDir volume and mounts that inside the container at /consul/data as configured automatically by Viya’s deployment process.



SAS Message Broker (RabbitMQ)

Storage for Stateful Services

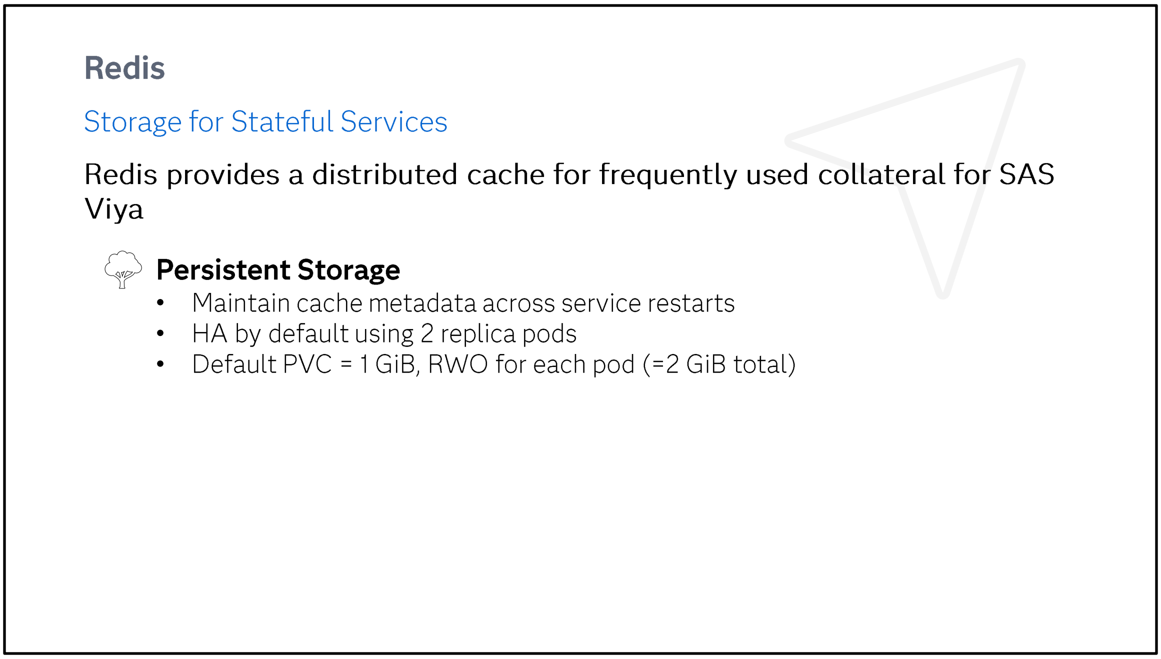
RabbitMQ provides message queuing services for SAS Viya

Persistent Storage

- Maintain state across service restarts
- HA by default using 3 replicas
- Default PVC = 2 GiB, RWO for each

RabbitMQ provides message queuing services for SAS Viya.

This is another service that’s deployed for high availability by default and also relies on three replicas. Each instance references a PVC with a default size of 2 gibabytes mounted RWO as each replica instance keeps an independent copy of the data.



Redis

Storage for Stateful Services

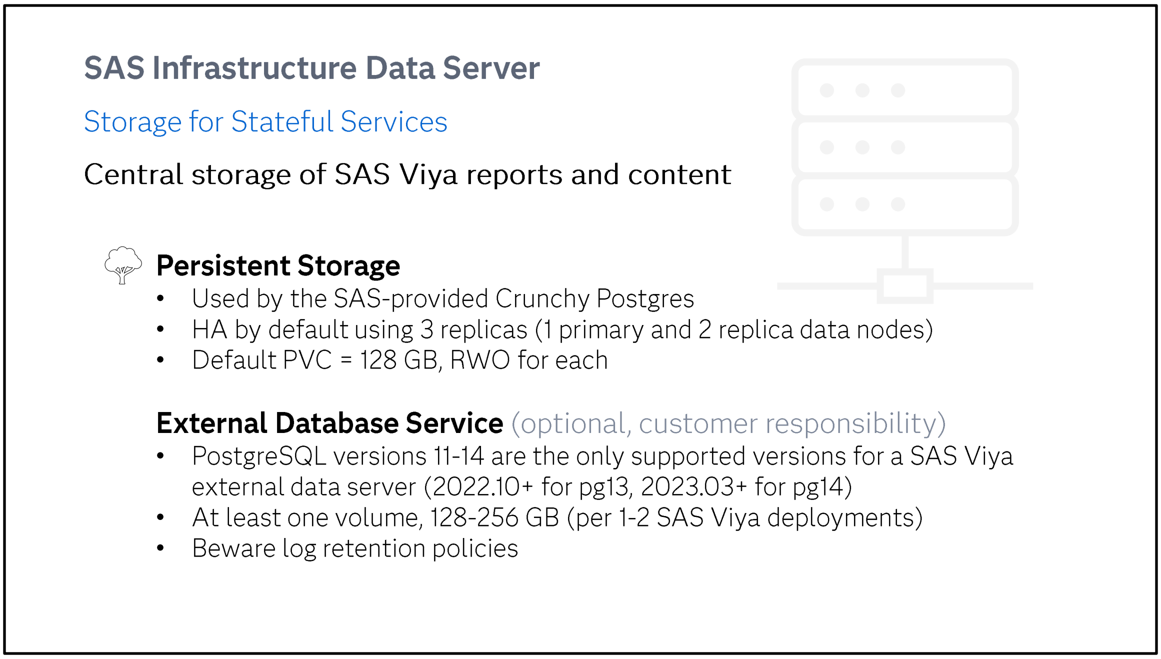
Redis provides a distributed cache for frequently used collateral for SAS Viya

Persistent Storage

- Maintain cache metadata across service restarts
- HA by default using 2 replica pods
- Default PVC = 1 GiB, RWO for each pod (=2 GiB total)

Redis provides a distributed cache for frequently used collateral for SAS Viya.

For high availability, it relies on two replica pods. Each pod references a PVC with a default size of 1 gibabyte mounted RWO for independent copies. Note this means 2 gibabytes total is set aside for Redis.



SAS Infrastructure Data Server

Storage for Stateful Services

Central storage of SAS Viya reports and content

Persistent Storage

- Used by the SAS-provided Crunchy Postgres
- HA by default using 3 replicas (1 primary and 2 replica data nodes)
- Default PVC = 128 GB, RWO for each

External Database Service (optional, customer responsibility)

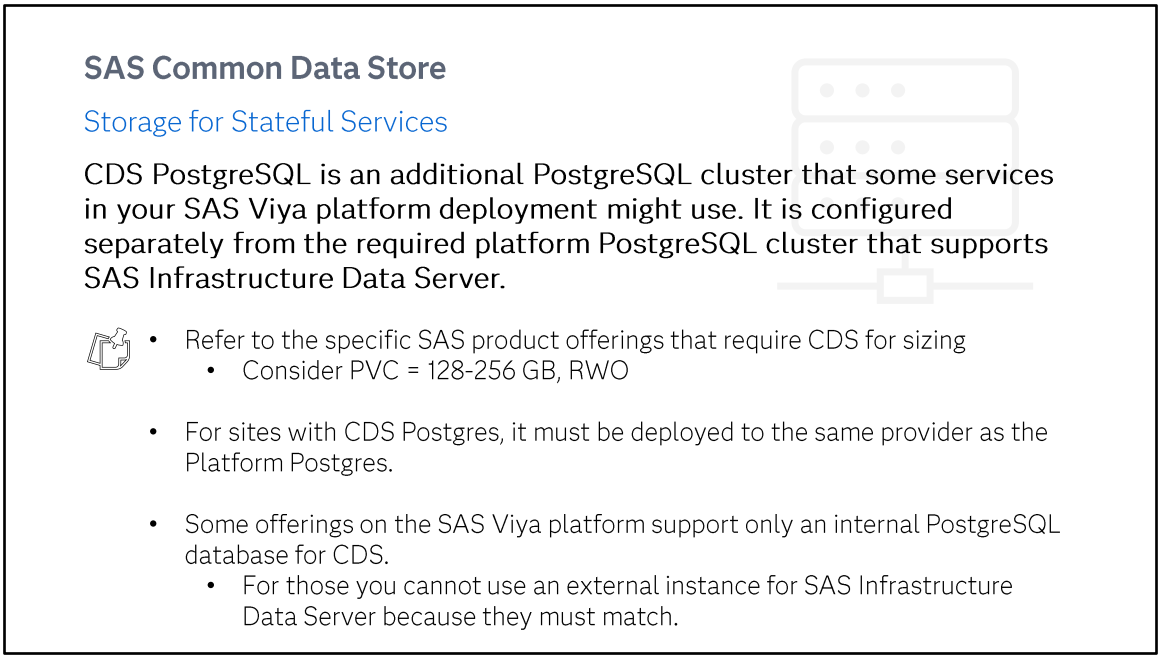
- PostgreSQL versions 11-14 are the only supported versions for a SAS Viya external data server (2022.10+ for pg13, 2023.03+ for pg14)
- At least one volume, 128-256 GB (per 1-2 SAS Viya deployments)
- Beware log retention policies

Next up is the SAS Infrastructure Data Server as backed by PostgreSQL. Indeed, the choice of PostgreSQL is up to you within the constraints of Viya's requirements. Viya can automatically deploy an internal Crunchy Postgres server or Viya can be optionally configured to work with an external Postgres server that is setup and maintained by your site's IT team.

The internal Crunchy Postgres is deployed as highly available by default which consists of one primary plus two replica data nodes. Each node will reference a PVC with a default 128 gigabytes mounted RWO for independent copies.

Alternatively, if you site elects to use an external Postgres that fits within the versions supported for Viya, then we recommend starting with at least one volume that's 128 to 256 gigabytes in size per Viya deployment. Of course, this is just a starting point – depending on usage, your site's IT team should monitor storage usage and adjust over time.

Also take care to monitor logs and the retention policies to avoid accidentally filling up disk space too quickly.



SAS Common Data Store

Storage for Stateful Services

CDS PostgreSQL is an additional PostgreSQL cluster that some services in your SAS Viya platform deployment might use. It is configured separately from the required platform PostgreSQL cluster that supports SAS Infrastructure Data Server.

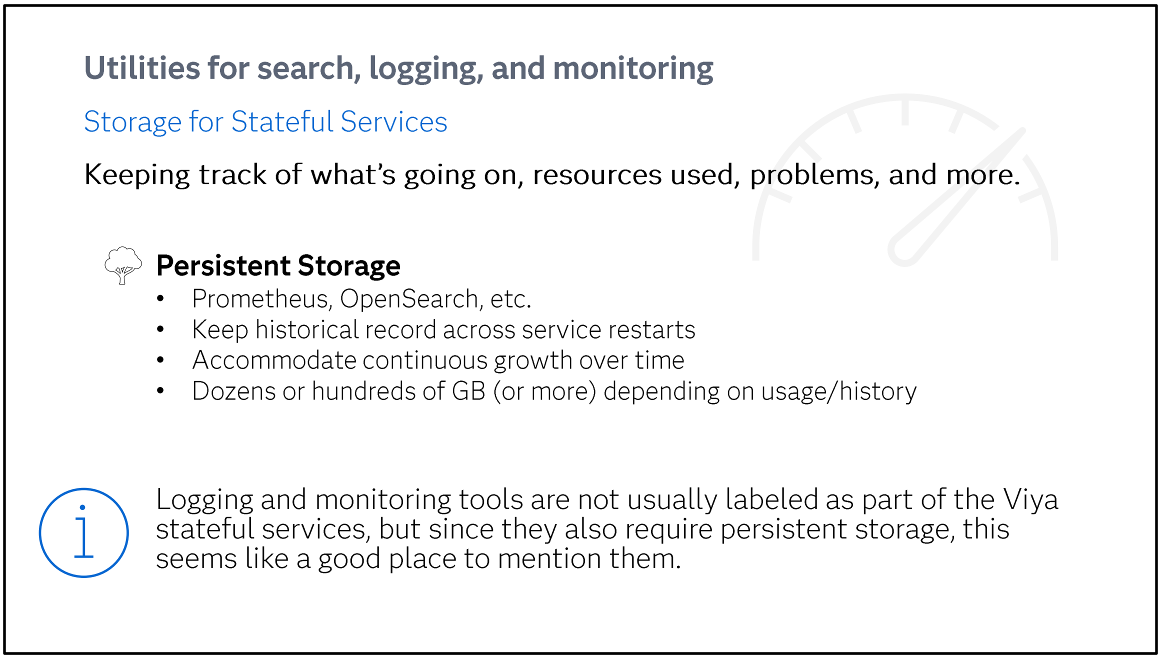
- Refer to the specific SAS product offerings that require CDS for sizing
  - Consider PVC = 128-256 GB, RWO
- For sites with CDS Postgres, it must be deployed to the same provider as the Platform Postgres.
- Some offerings on the SAS Viya platform support only an internal PostgreSQL database for CDS.
  - For those you cannot use an external instance for SAS Infrastructure Data Server because they must match.

The SAS Common Data Store, or CDS, is an additional PostgreSQL cluster that some services in your SAS Viya platform deployment might use. It is configured separately from the required platform PostgreSQL cluster that supports SAS Infrastructure Data Server.

You’ll need to refer to the specific SAS product offerings that require CDS for sizing. Expect to start with one persistent storage volume between 128 and 256 gigabytes in size for CDS.

For sites with CDS Postgres, it must be deployed to the same provider as the primary Platform Postgres.

Note also that some solution offerings on the SAS Viya platform support only an internal PostgreSQL database for CDS. This means that you cannot use an external instance for SAS Infrastructure Data Server for those because they must match.



Utilities for search, logging, and monitoring

Storage for Stateful Services

Keeping track of what's going on, resources used, problems, and more.

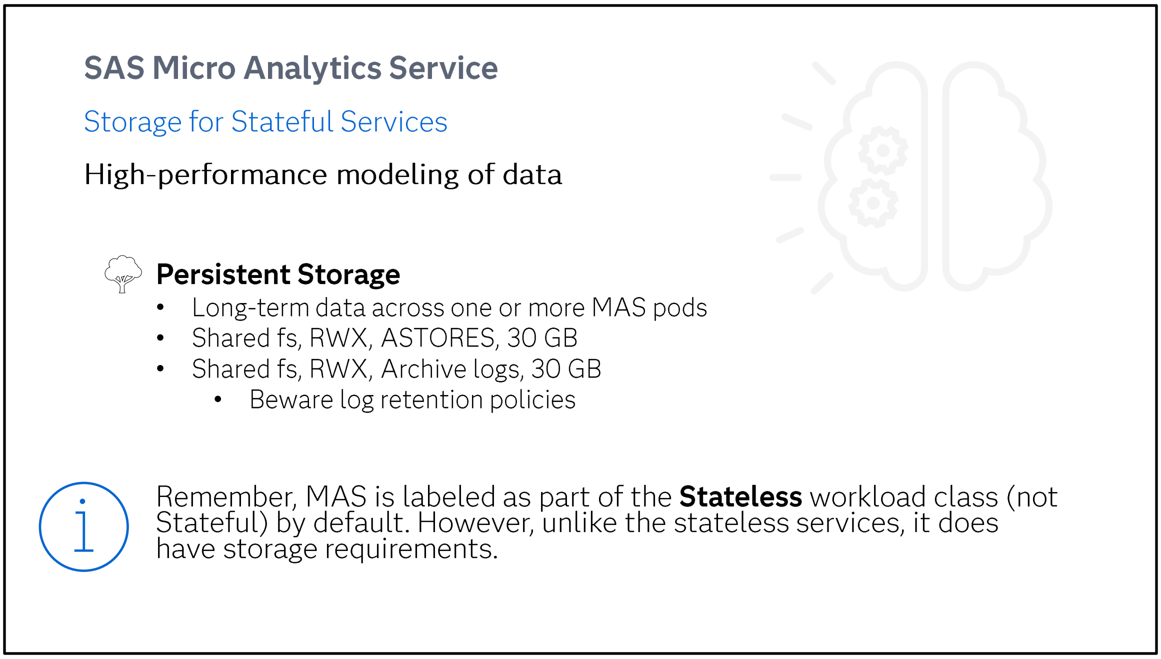
Persistent Storage

- Prometheus, OpenSearch, etc.
- Keep historical record across service restarts
- Accommodate continuous growth over time
- Dozens or hundreds of GB (or more) depending on usage/history

Logging and monitoring tools are not usually labeled as part of the Viya stateful services, but since they also require persistent storage, this seems like a good place to mention them.

Additional utilities can be deployed to the Kubernetes cluster to extend functionality and add administrative insights. Projects like OpenSearch, Prometheus, Grafana, and more are useful to keep historical record. The data they store can be referenced over time and grows continuously measuring dozens or hundreds of gigabytes, so consider implementing a retention policy that protects against excessive disk usage.

While not technically labeled as part of Viya's stateful services, this seems like a good place to mention the storage considerations of these utilities.



SAS Micro Analytics Service

Storage for Stateful Services

High-performance modeling of data

Persistent Storage

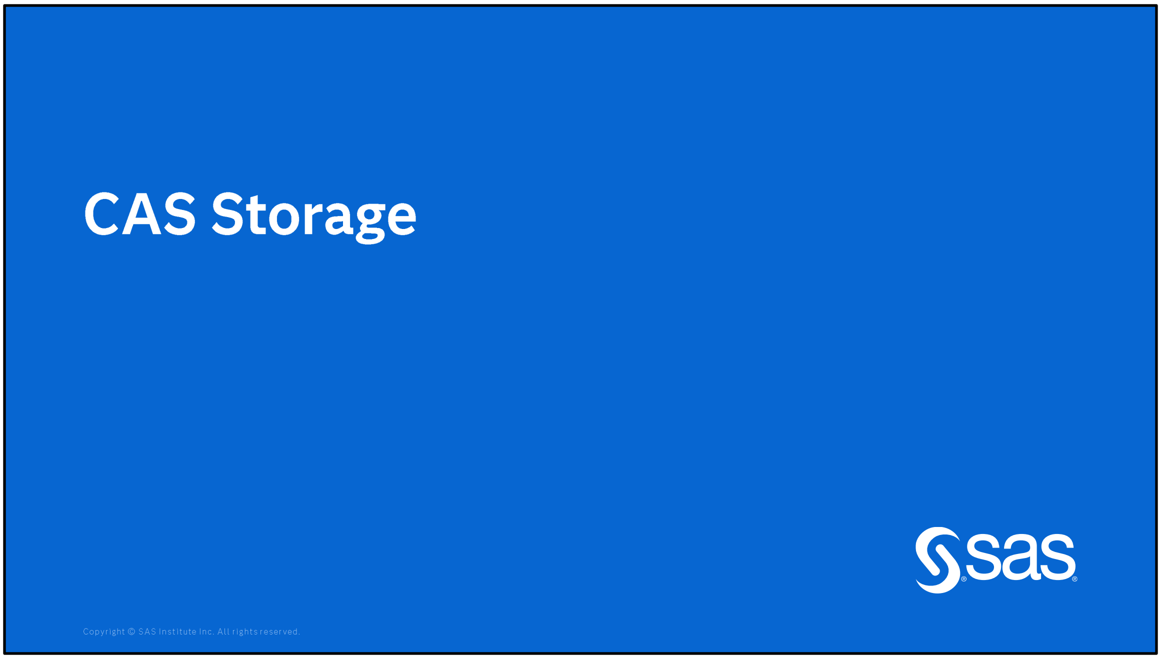
- Long-term data across one or more MAS pods
- Shared fs, RWX, ASTORES, 30 GB
- Shared fs, RWX, Archive logs, 30 GB
  - Beware log retention policies

Remember, MAS is labeled as part of the **Stateless** workload class (not Stateful) by default. However, unlike the stateless services, it does have storage requirements.

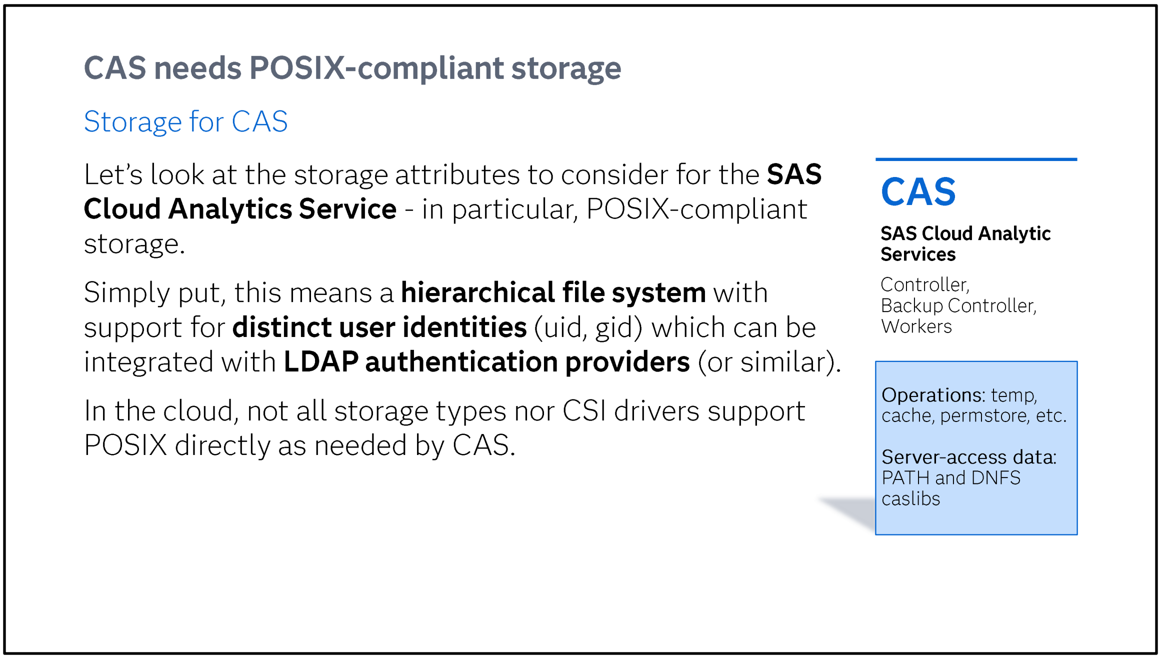
And finally, let’s talk about the SAS Micro Analytics Service, or MAS.

You might recall that it’s labeled as a stateless Viya service by default, but that’s mostly to ensure it doesn’t run on Kubernetes nodes that are set aside for heavy analytic or infrastructure duties. If your site makes substantial use of MAS, it can be directed to run on dedicated nodes as needed.

In any case, unlike the real stateless services, it does make use of persistent storage and expects two volumes in particular. One for A-STORES starting with a PVC around 30 gigabytes in size and another 30 gigabytes PVC for its archive logs. Note that both volumes are expected to mount RWX on shared file systems. This means that multiple instances of MAS can run and they’ll each mount the same shared file systems to access and update A-STORES and archive logs.



CAS Storage



CAS needs POSIX-compliant storage

Storage for CAS

Let’s look at the storage attributes to consider for the **SAS Cloud Analytics Service** - in particular, POSIX-compliant storage.

Simply put, this means a **hierarchical file system** with support for **distinct user identities** (uid, gid) which can be integrated with **LDAP authentication providers** (or similar).

In the cloud, not all storage types nor CSI drivers support POSIX directly as needed by CAS.

CAS

SAS Cloud Analytic Services

Controller,
Backup Controller,
Workers

Operations: temp, cache, permstore, etc.

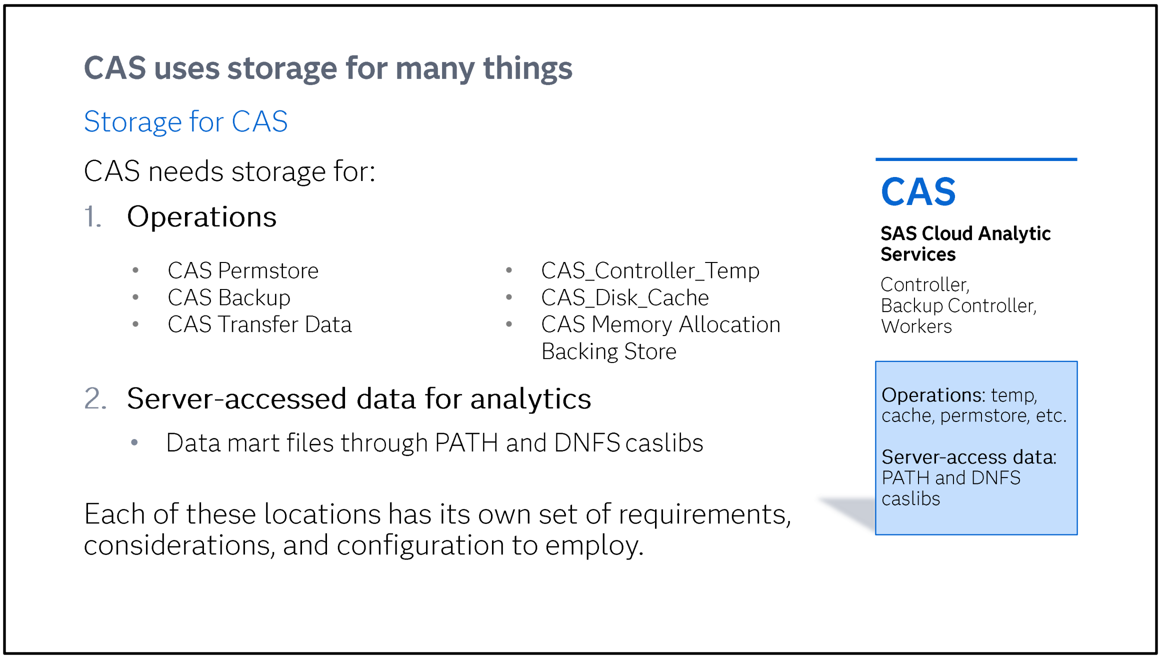
Server-access data: PATH and DNFS caslibs

Let’s look at the storage considerations for CAS – the SAS Cloud Analytics Service.

As a reminder, CAS can be deployed SMP mode running on a single host or in MPP mode where it’s split to run across multiple host machines. In MPP mode, CAS will have at least two Workers and one CAS Controller. Optionally, we can add more Workers as needed as well as a Secondary or Backup CAS Controller for high availability.

CAS requires POSIX-compliant storage to provide a hierarchical file system with support for distinct user identities (in POSIX parlance referring to uid and gid) that can integrate with LDAP authentication providers.

Take care with your cloud vendor. Not all storage types nor CSI drivers (including some that claim full POSIX compliance) actually support the necessary POSIX functionality needed by CAS.



CAS uses storage for many things

Storage for CAS

CAS needs storage for:

1. Operations
   - CAS Permstore
   - CAS Backup
   - CAS Transfer Data
   - CAS_Controller_Temp
   - CAS_Disk_Cache
   - CAS Memory Allocation Backing Store
2. Server-accessed data for analytics
   - Data mart files through PATH and DNFS caslibs

Each of these locations has its own set of requirements, considerations, and configuration to employ.

CAS

SAS Cloud Analytic Services

Controller,
Backup Controller,
Workers

Operations: temp, cache, permstore, etc.

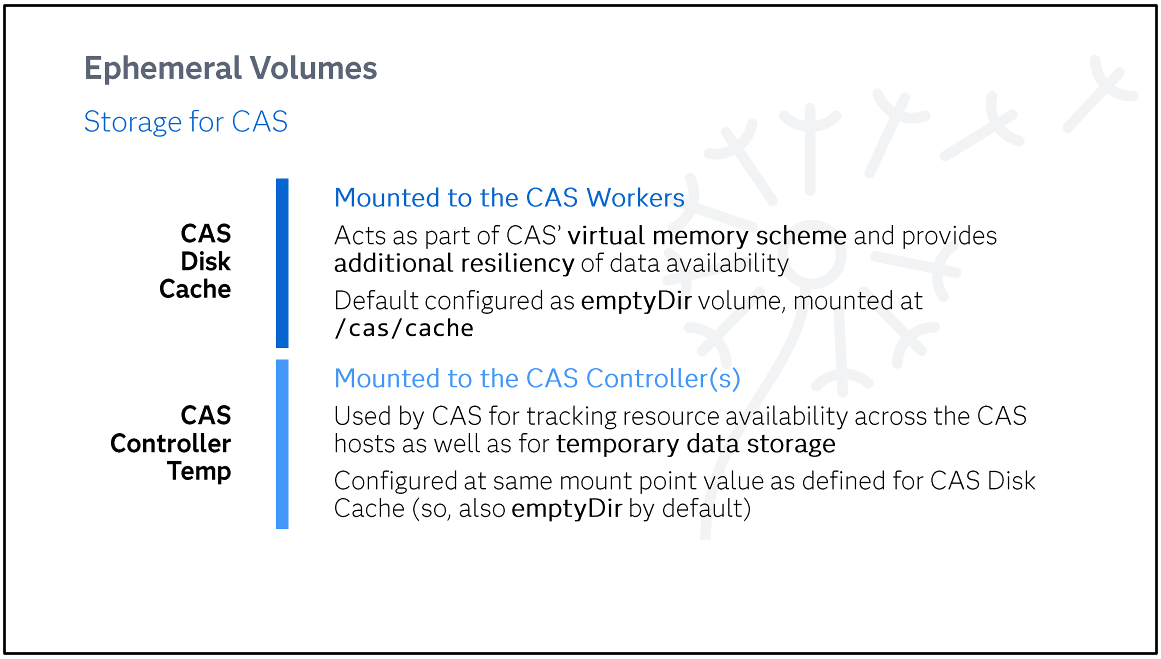
Server-access data: PATH and DNFS caslibs

There are two broad classifications for the kinds of things that CAS uses storage for: Operations and Data.

Operations covers storage in support of backend activity for CAS to process things efficiently. For example, CAS Permstore store permissions metadata for tables in CAS. The Backup location is used for staging backup data as part of the larger Viya backup processing. And the State Transfer functionality, if enabled, needs space to place files in process. We'll look more closely at the CAS Disk Cache, CAS Controller Temp, and the CAS Memory Allocation Backing Store in a moment.

The Data storage is optionally enabled when you want CAS to have direct disk-access to data mart files stored in supported formats like SASHDAT, CSV, SAS7BDAT, Parquet, and more. CAS can access that data through the use of PATH and DNFS type of caslibs as suits the CAS deployment and data type. Of course, if CAS only needs data from external databases, then this disk space might not be necessary.

Keep in mind that each of these locations has its own set of requirements, considerations, and configuration you might need to employ.



Ephemeral Volumes

Storage for CAS

CAS Disk Cache

Mounted to the CAS Workers

Acts as part of CAS' **virtual memory scheme** and provides **additional resiliency** of data availability

Default configured as **emptyDir** volume, mounted at `/cas/cache`

CAS Controller Temp

Mounted to the CAS Controller(s)

Used by CAS for tracking resource availability across the CAS hosts as well as for **temporary data storage**

Configured at same mount point value as defined for CAS Disk Cache (so, also **emptyDir** by default)

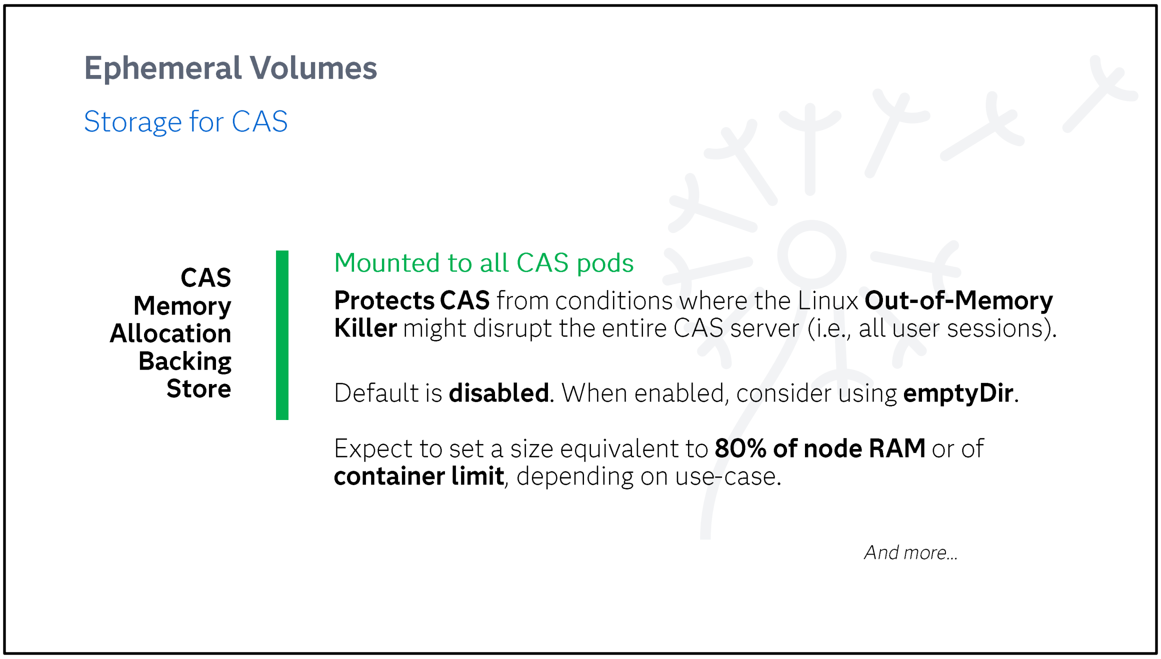
Of the volumes in support of CAS Operations, two in particular are typically defined as ephemeral volumes.

The CAS Disk Cache plays a major role in the operations of CAS as it provides space in support of a virtual memory scheme which allows CAS to load more data than physical RAM can handle. It also additional resiliency to protect against the unexpected loss of host machines for the CAS workers.

By default, CAS is configured to use a Kubernetes emptyDir as its volume.

The CAS Controller also gets its own temp space with the CAS Controller Temp volume. This space is used by CAS to track resource availability across the CAS hosts as well as for staging data temporarily as needed to support certain operations.

The CAS Controller and CAS Workers should always run on independent nodes of the Kubernetes cluster – that is, we don't expect to see a CAS Controller pod on the same host as a CAS Worker pod. With that in mind, the default value for CAS Controller Temp is to use the same value as specified for CAS Disk Cache. They can be configured to use different values, but usually that's not necessary. Just don't be surprised if you see CAS Controller Temp mounted a /cas/cache.



Ephemeral Volumes

Storage for CAS

CAS Memory Allocation Backing Store

Mounted to all CAS pods

Protects CAS from conditions where the Linux **Out-of-Memory Killer** might disrupt the entire CAS server (i.e., all user sessions).

Default is **disabled**. When enabled, consider using **emptyDir**.

Expect to set a size equivalent to **80% of node RAM** or of **container limit**, depending on use-case.

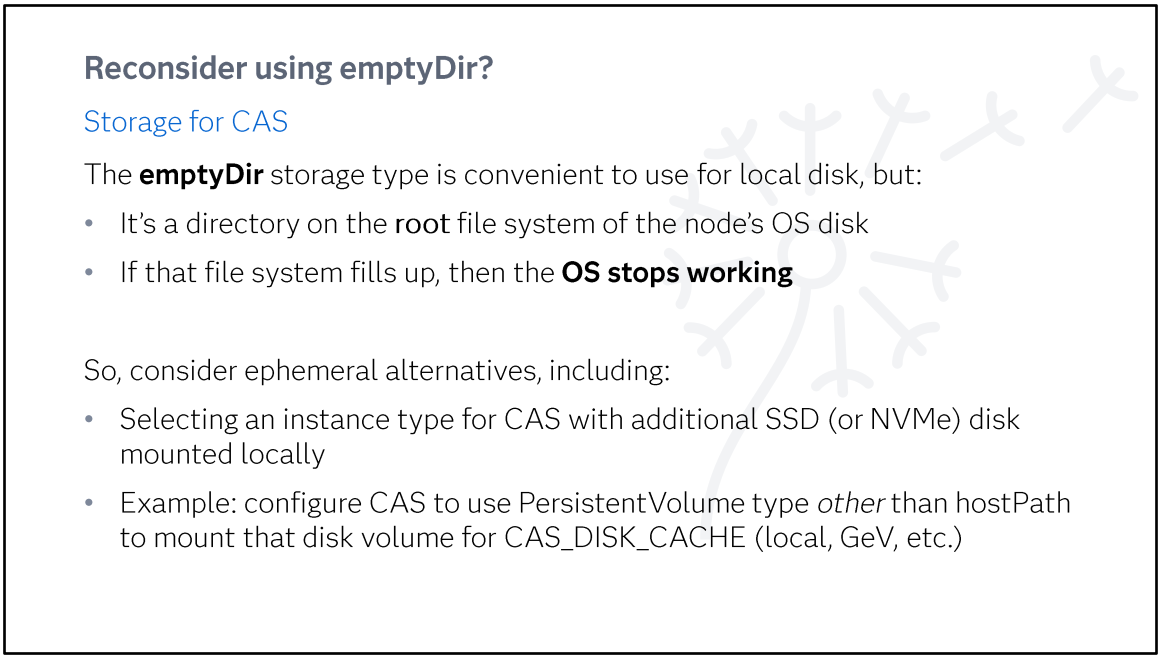
And more…

Next there's a new feature for CAS introduced in twenty-twenty-four-oh-nine that might require its own ephemeral volume, too.

The CAS Memory Allocation Backing Store is intended to guard against certain system protections found in the Kubernetes environment – specifically the Linux Out-of-Memory Killer process. For certain configurations of CAS, we need protection to ensure CAS isn't inadvertently taken out of service by the O-O-M-Killer. We explain more about the technical details elsewhere in this workshop.

The out-of-box configuration for CAS does not enable the CAS Memory Allocation Backing Store. But when it is enabled, we expect that an emptyDir volume will provide the space and should be set to 80% of the effective RAM available, either by host node or container, depending on the use-case.

Watch this configuration in future releases of the Viya platform as it might become enabled by default in a near release.



Reconsider using emptyDir?

Storage for CAS

The **emptyDir** storage type is convenient to use for local disk, but:

- It's a directory on the **root** file system of the node's OS disk
- If that file system fills up, then the **OS stops working**

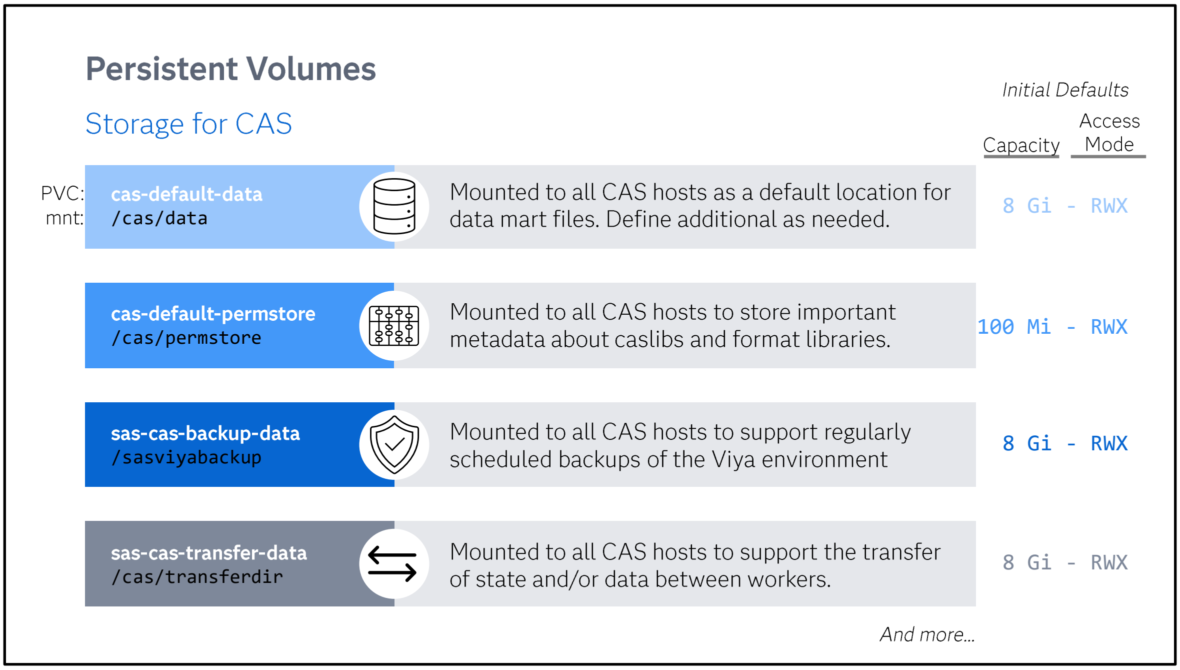
So, consider ephemeral alternatives, including:

- Selecting an instance type for CAS with additional SSD (or NVMe) disk mounted locally
- Example: configure CAS to use PersistentVolume type *other* than hostPath to mount that disk volume for CAS_DISK_CACHE (local, GeV, etc.)

The emptyDir storage type is convenient to use for local disk since we can all but guarantee it will exist in any Kubernetes environment, but there are some important additional factors to consider. For one, it's a directory on the root (or top-level) file system of the node's OS disk. This challenge here is that if the root file system fills up, then the OS will stop working. Without a functioning OS, then the node stops working as well. Of course, this has a negative effect on any pods there, including CAS workers or controllers.

We recommend taking steps to provide alternative ephemeral storage for CAS. Note that most cloud providers offer some instance types which include additional SSD (or NVMe) disk mounted locally and that we can reference for CAS Disk Cache.

The conventional approach in Kubernetes historically to get to local, ephemeral disk on a host was to use a hostPath type of volume. However, significant security problems with that approach means that most sites don't allow hostPath volumes for production use. That's okay as Kubernetes also supports the use a "local" type of PersistentVolume to mount that disk volume for CAS_DISK_CACHE. Generic Ephemeral Volumes can work as well.



Persistent Volumes

Storage for CAS

Initial Defaults

PVC: mnt:		Capacity	Access Mode
cas-default-data `/cas/data`	Mounted to all CAS hosts as a default location for data mart files. Define additional as needed.	`8 Gi` -	`RWX`
cas-default-permstore `/cas/permstore`	Mounted to all CAS hosts to store important metadata about caslibs and format libraries.	`100 Mi` -	`RWX`
sas-cas-backup-data `/sasviyabackup`	Mounted to all CAS hosts to support regularly scheduled backups of the Viya environment	`8 Gi` -	`RWX`
sas-cas-transfer-data `/cas/transferdir`	Mounted to all CAS hosts to support the transfer of state and/or data between workers.	`8 Gi` -	`RWX`

And more…

Shifting gears, let’s look at some persistent volumes for CAS that will persist beyond the lifecycle of the current pods.

The other volumes for CAS are typically set up as PersistentVolumes as they contain data that needs to survive the shutdown and restart of CAS.

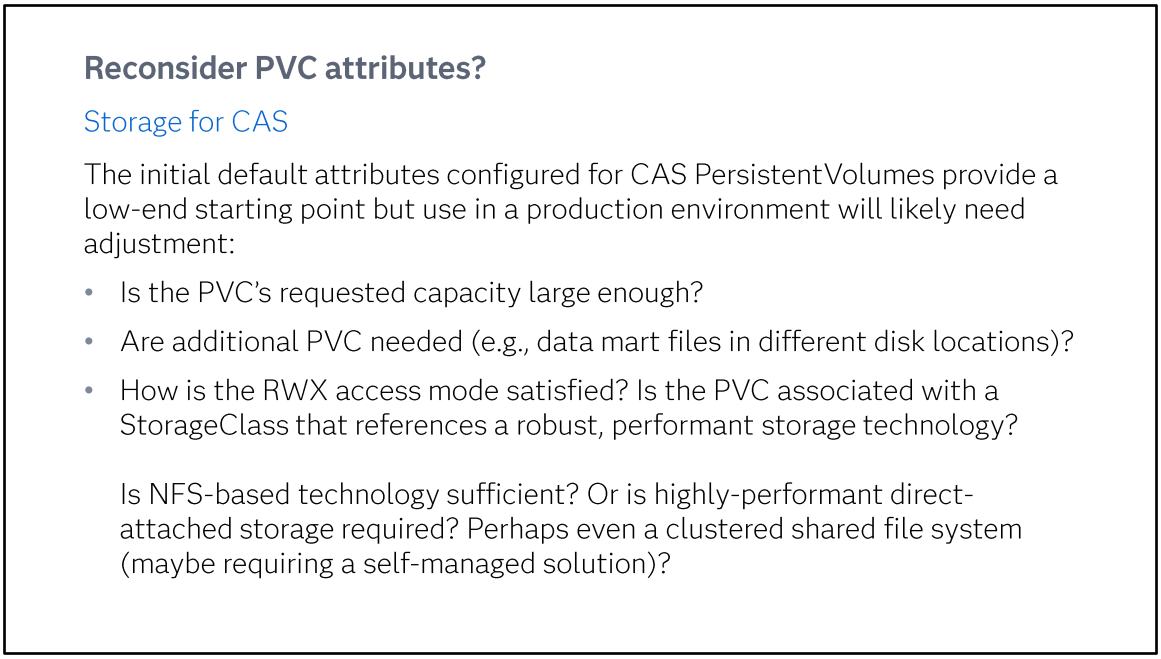
The first one we show here is the PVC referred to as CAS Default Data that’s mounted in the CAS Controller and Workers pods at the /cas/data path. It can be used as a starting place for storing data mart files that CAS can access using the PATH or DNFS type of caslibs. If additional volumes are desired for more data mart locations on disk, then use this one as an example.

To be clear, it’s a single volume that multiple machines access concurrently so its access mode is defined as RWX. The PVC spec defaults to requesting 8 gibibytes of space, but of course that is configurable to suit your needs.

Next up is the CAS Default Permstore PVC. It’s mounted RWX on all CAS hosts – Controllers and Workers – to store metadata about caslibs and format libraries with an initial capacity of 100 mebibytes.

The volume for the SAS CAS Backup Data PVC is mounted RWX to all CAS hosts to support backups of the Viya environment. It defaults to 8 gibibytes in size which should be adjusted to suit the needs of your environment over time.

If you’ve enabled the State Transfer functionality for CAS, then the SAS CAS Transfer Data PVC is used to support the transfer of state data between CAS workers.



Reconsider PVC attributes?

Storage for CAS

The initial default attributes configured for CAS PersistentVolumes provide a low-end starting point but use in a production environment will likely need adjustment:

- Is the PVC's requested capacity large enough?
- Are additional PVC needed (e.g., data mart files in different disk locations)?
- How is the RWX access mode satisfied? Is the PVC associated with a StorageClass that references a robust, performant storage technology?

  Is NFS-based technology sufficient? Or is highly-performant direct-attached storage required? Perhaps even a clustered shared file system (maybe requiring a self-managed solution)?

The initial default attributes configured for CAS PersistentVolumes provide a low-end starting point but will likely need adjustment for production use at your site, so consider the following:

Is the PVC's requested capacity large enough?

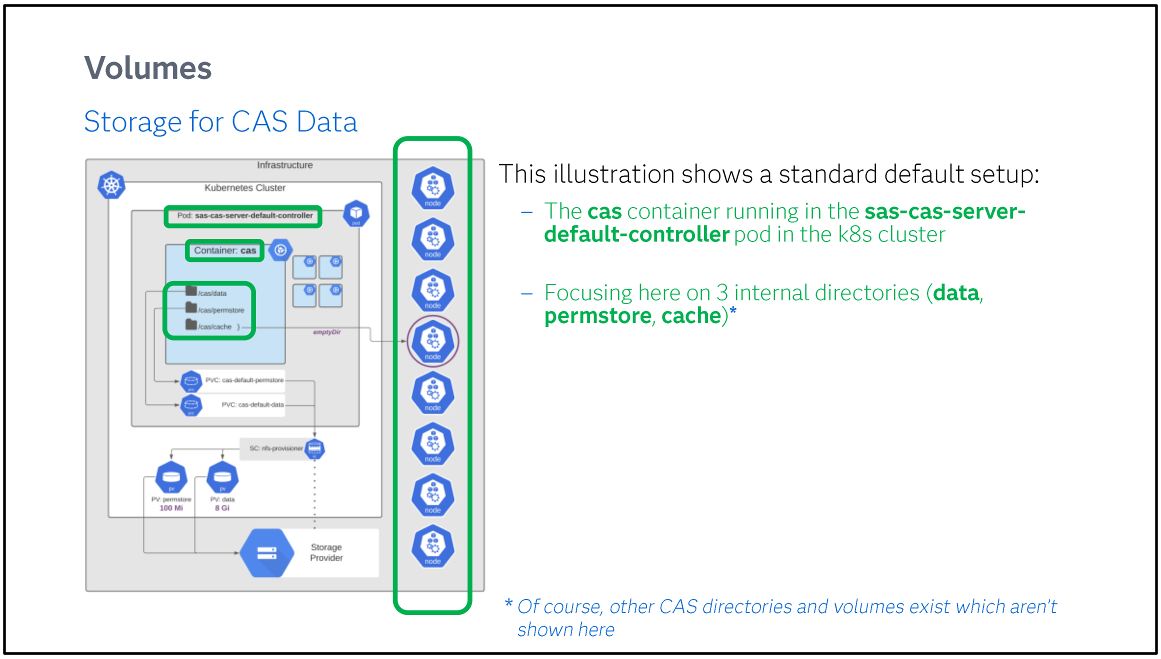
CAS is designed to tackle to the largest analytic workloads. That means a lot of data might be fed into the system. PVC sizing will depend in large part on the number of tables active in CAS and their sizes.

Are additional PVC needed (e.g., data mart files in different disk locations)?

The CAS Default Data PVC is provided as a starting point. But your site might elect to place data files on different volumes which then needed to get mounted where CAS can access them.

How is the RWX access mode satisfied? Is the PVC associated with a StorageClass that references a robust, performant storage technology?

All shared storage options are not the same. NFS-based technologies rely on your host's network capacity to ship data around. Without sufficient network bandwidth, the ability to work with large volumes or other heavy activity might cause contention for limited network resources. Growing beyond NFS-based offerings include using direct-attached storage options or even clustered file shares which can mean substantial additional costs.



Volumes

Storage for CAS Data

This illustration shows a standard default setup:

- The **cas** container running in the **sas-cas-server-default-controller** pod in the k8s cluster
- Focusing here on 3 internal directories (**data**, **permstore**, **cache**)*

** Of course, other CAS directories and volumes exist which aren't shown here*

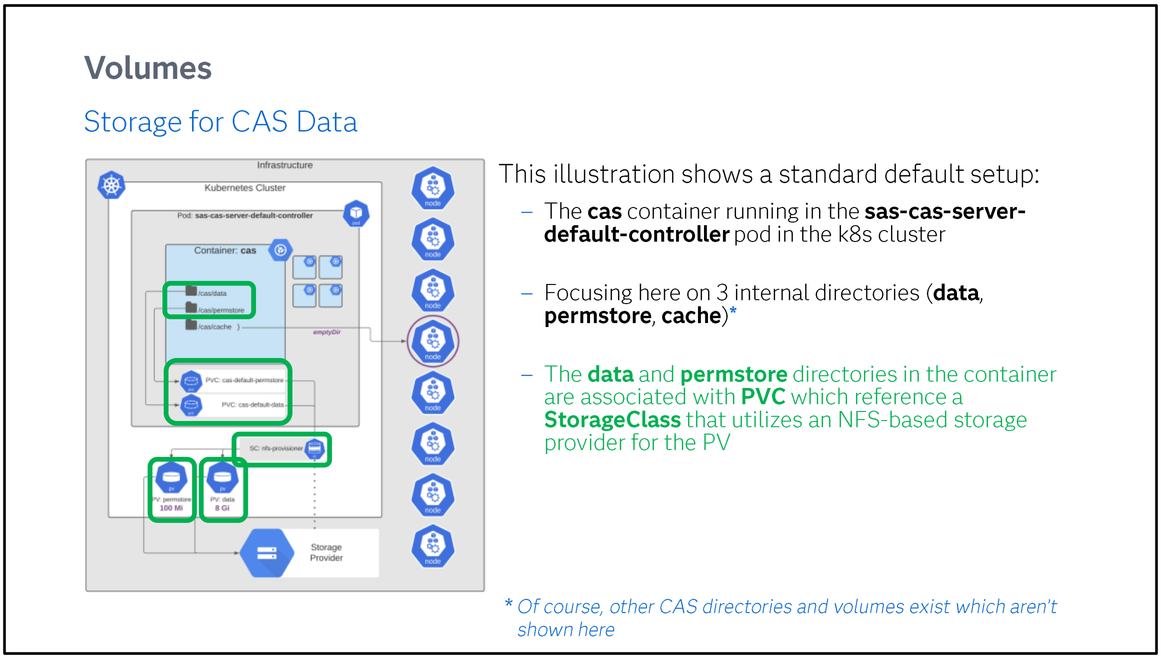
This is a simple illustration to help show the relationship of the various volumes we encounter with CAS.

Working our way in from the outside, we've got infrastructure that includes 8 nodes for our Kubernetes cluster.

In Kubernetes, we're looking at the SAS CAS Server Default Controller pod…

…and we're zoomed into the CAS container of that pod.

This illustration shows a standard, default setup for storage as described earlier. In the CAS container, we're looking at three internal directories as mount points for CAS Data, CAS Permstore, and CAS Cache. Of course, other CAS directories and volumes exist which aren't shown here.



Volumes

Storage for CAS Data

This illustration shows a standard default setup:

- The **cas** container running in the **sas-cas-server-default-controller** pod in the k8s cluster
- Focusing here on 3 internal directories (**data**, **permstore**, **cache**)*
- The **data** and **permstore** directories in the container are associated with **PVC** which reference a **StorageClass** that utilizes an NFS-based storage provider for the PV

** Of course, other CAS directories and volumes exist which aren't shown here*

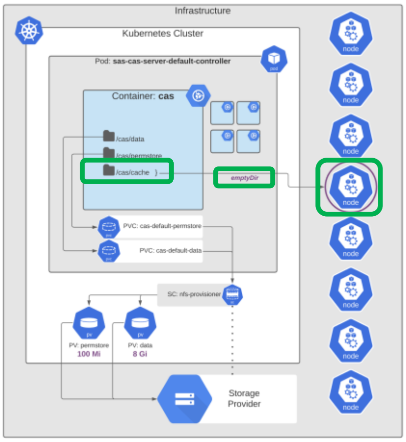
Following the mount points for the Data and Permstore directories…

…we see them linked to PVC that are associated with the NFS Provisioner storage class.

Having a storage class allows Kubernetes to work with a storage provisioner to dynamically provision volumes as needed.

Volumes

Storage for CAS Data



This illustration shows a standard default setup:

- The **cas** container running in the **sas-cas-server-default-controller** pod in the k8s cluster
- Focusing here on 3 internal directories (**data**, **permstore**, **cache**)*
- The **data** and **permstore** directories in the container are associated with **PVC** which reference a **StorageClass** that utilizes an NFS-based storage provider for the PV
- The **cache** directory is associated with an **emptyDir** for an ephemeral volume located in the root file system of the node that's hosting the pod – recall on the Controller this is used for CAS_Controller_Temp.

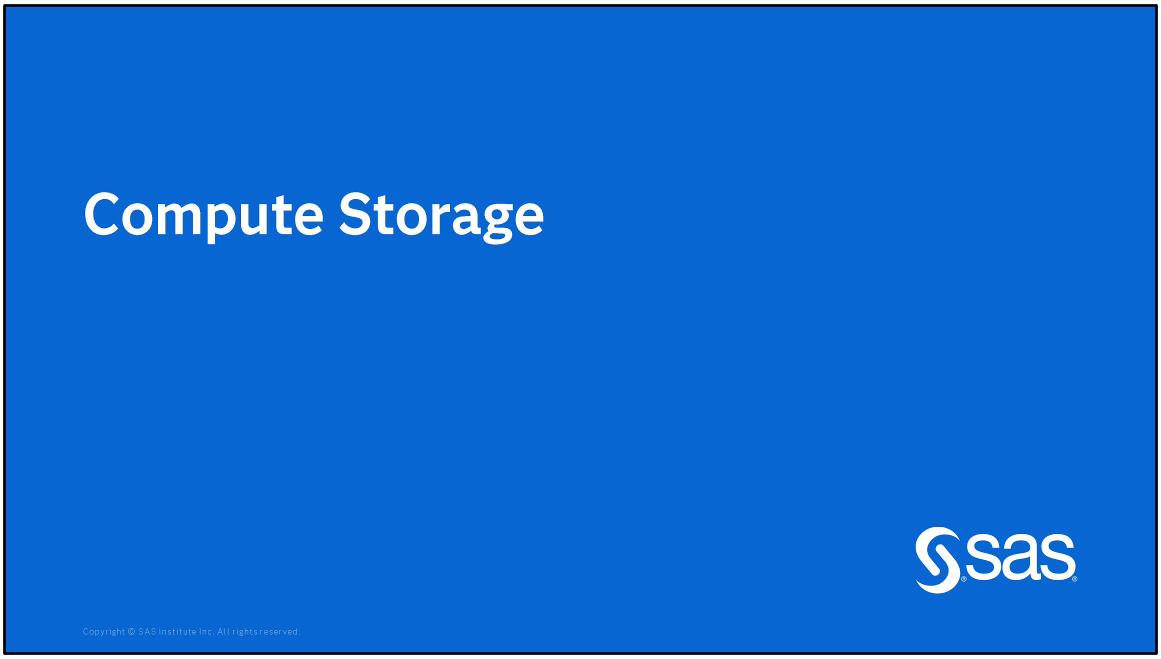
** Of course, other CAS directories and volumes exist which aren't shown here*

The CAS Cache directory, on the other hand, isn't defined as a PVC by default, but instead…

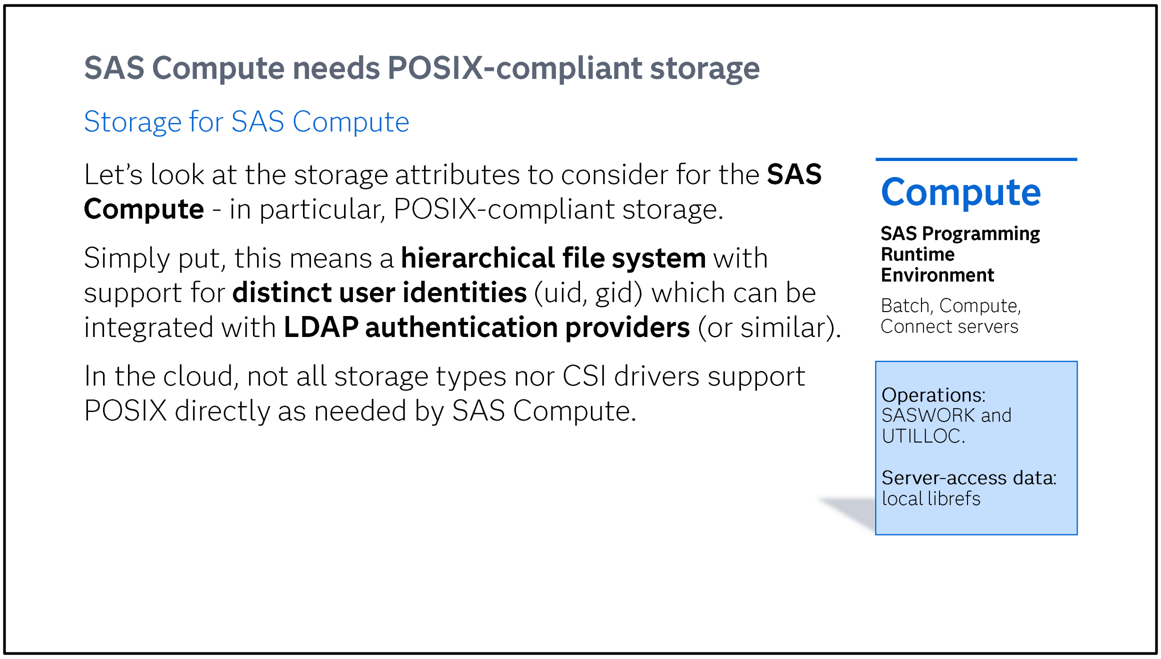
relies on an emptyDir volume…

…on the host where the CAS pod is assigned to run.

In this example, we're looking at a CAS Controller pod and recall that it relies on the cache directory for the CAS Controller Temp storage purpose. On a CAS Worker, this path would actually be used for CAS Disk Cache to support a virtual memory scheme as well as resiliency of data in case of the loss of another Worker.



Compute Storage



SAS Compute needs POSIX-compliant storage

Storage for SAS Compute

Let’s look at the storage attributes to consider for the **SAS Compute** - in particular, POSIX-compliant storage.

Simply put, this means a **hierarchical file system** with support for **distinct user identities** (uid, gid) which can be integrated with **LDAP authentication providers** (or similar).

In the cloud, not all storage types nor CSI drivers support POSIX directly as needed by SAS Compute.

Compute

SAS Programming Runtime Environment

Batch, Compute, Connect servers

Operations: SASWORK and UTILLOC.

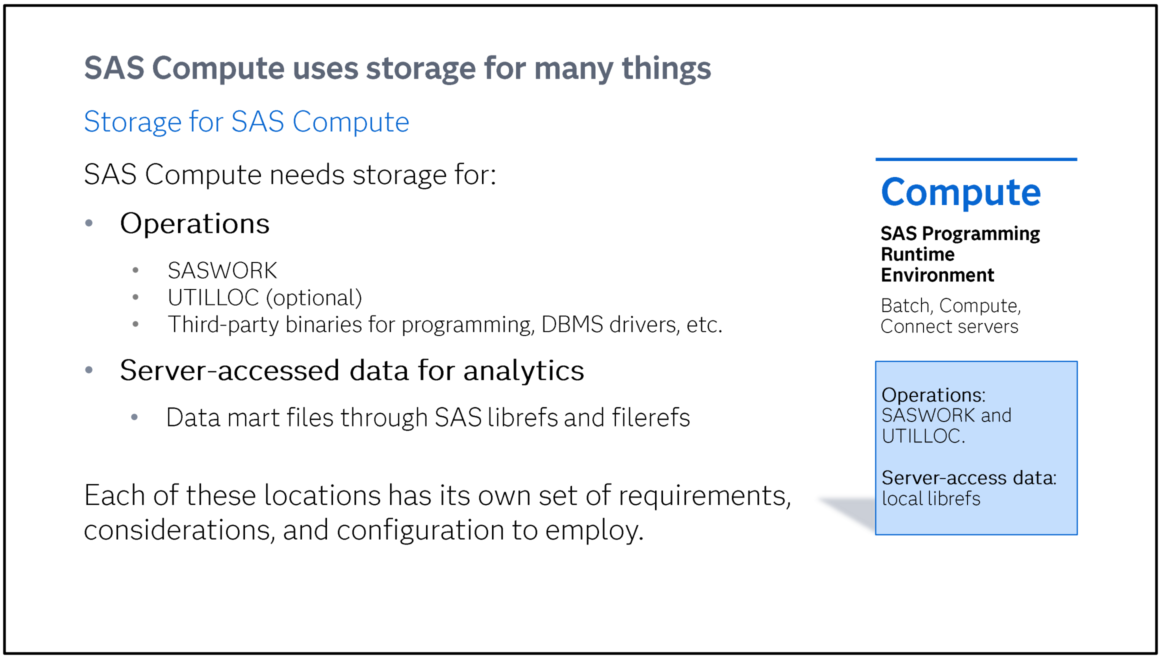
Server-access data: local librefs

Let’s look at the storage considerations SAS Compute.

That is, for the Compute engine of the SAS Programming Runtime Environment as instantiated by SAS Batch, SAS Compute, and SAS Connect servers in Viya.

Similar to what we saw for CAS, SAS Compute requires POSIX-compliant storage to provide a hierarchical file system with support for distinct user identities (in POSIX parlance referring to uid and gid) that can integrate with LDAP authentication providers.

Take care with your cloud vendor. Not all storage types nor CSI drivers (including some that claim full POSIX compliance) actually support the necessary POSIX functionality needed by SAS Compute.



SAS Compute uses storage for many things

Storage for SAS Compute

SAS Compute needs storage for:

- Operations
  - SASWORK
  - UTILLOC (optional)
  - Third-party binaries for programming, DBMS drivers, etc.
- Server-accessed data for analytics
  - Data mart files through SAS librefs and filerefs

Each of these locations has its own set of requirements, considerations, and configuration to employ.

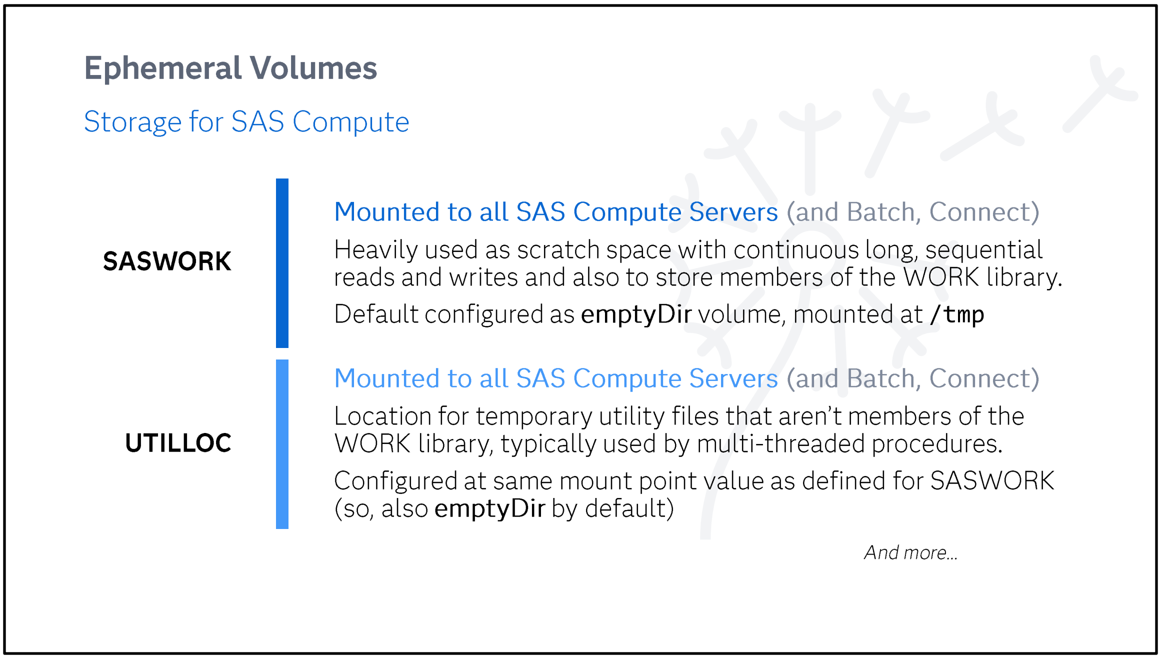
Compute
SAS Programming Runtime Environment
Batch, Compute, Connect servers
Operations: SASWORK and UTILLOC.
Server-access data: local librefs

There are two broad classifications for the kinds of things that SAS Compute uses storage for: Operations and Data.

Operations covers storage in support of backend activity for SAS Compute to process things efficiently. For example, SASWORK is a critical component of the SAS runtime's ability to work with data. Additionally, we might also need to mount volumes in support of enabling functionality provided by third-party software, like open-source programming languages, DBMS drivers to support SAS/ACCESS products, and more.

The Data storage is optionally enabled when you want the SAS runtime to have direct disk-access to data mart files stored in supported formats like SAS data sets, CSV, and more. Of course, if SAS only needs data from external databases, then this disk space might not be necessary.

Keep in mind that each of these locations has its own set of requirements, considerations, and configuration you might need to employ.



Ephemeral Volumes

Storage for SAS Compute

SASWORK

Mounted to all SAS Compute Servers (and Batch, Connect)

Heavily used as scratch space with continuous long, sequential reads and writes and also to store members of the WORK library.

Default configured as **emptyDir** volume, mounted at `/tmp`

UTILLOC

Mounted to all SAS Compute Servers (and Batch, Connect)

Location for temporary utility files that aren't members of the WORK library, typically used by multi-threaded procedures.

Configured at same mount point value as defined for SASWORK (so, also **emptyDir** by default)

And more…

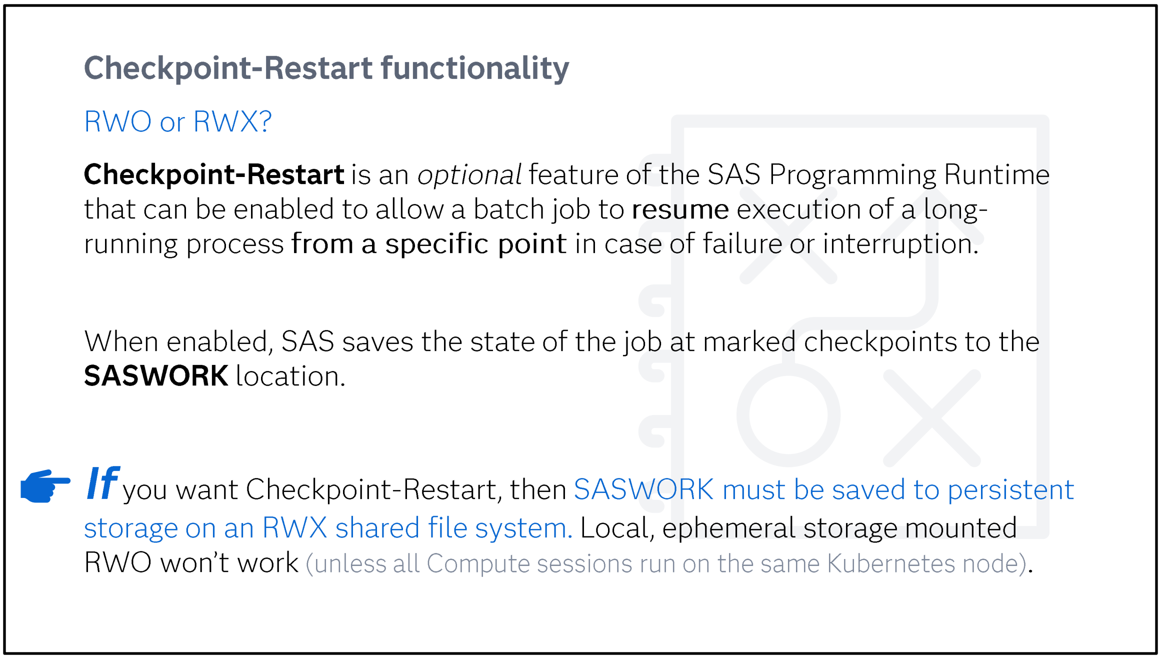
Let's talk about ephemeral volumes.

Of the volumes in support of the SAS Programming Runtime Environment, two in particular are typically defined as ephemeral volumes.

SASWORK is used as scratch space with continuous long, sequential reads and writes and also to store members of the WORK library.

By default, SASWORK is configured to use a Kubernetes emptyDir as its volume.

If needed, we can also specify a separate location for UTILLOC for scratch files that are generated by multi-threaded procedures, like PROC SORT. By default, it uses the same mount point as SASWORK, but it can be separated if having an additional I/O channel helps improve performance.



Checkpoint-Restart functionality

RWO or RWX?

Checkpoint-Restart is an *optional* feature of the SAS Programming Runtime that can be enabled to allow a batch job to **resume** execution of a long-running process **from a specific point** in case of failure or interruption.

When enabled, SAS saves the state of the job at marked checkpoints to the **SASWORK** location.

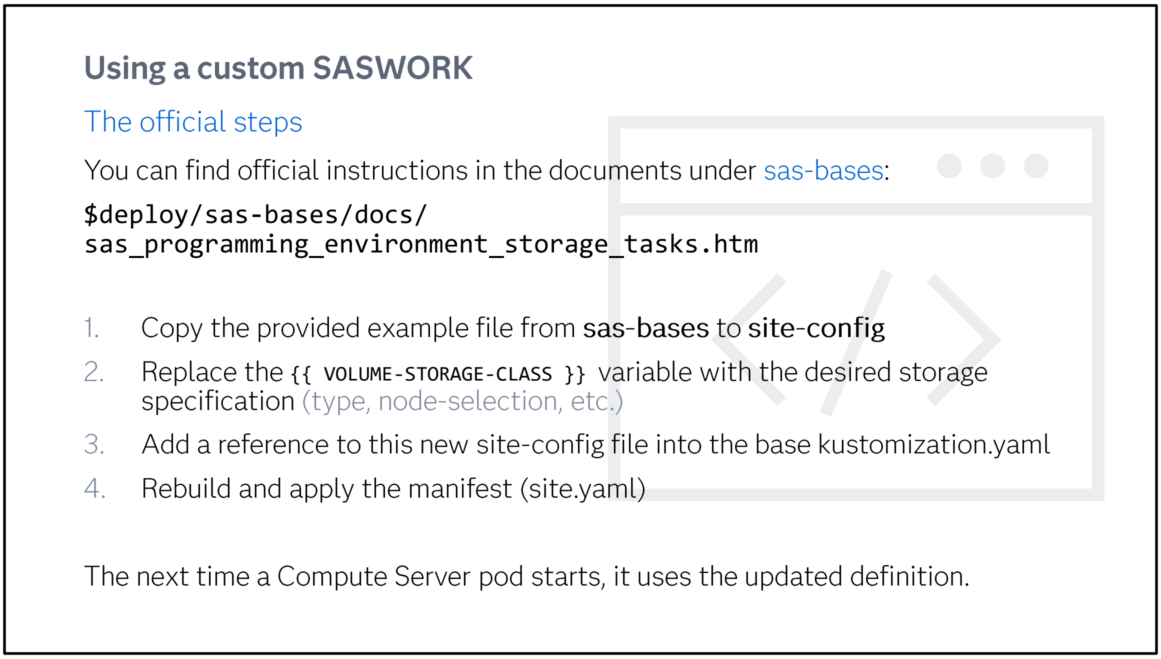
If you want Checkpoint-Restart, then SASWORK must be saved to persistent storage on an RWX shared file system. Local, ephemeral storage mounted RWO won't work (unless all Compute sessions run on the same Kubernetes node).

Checkpoint-Restart is an optional feature of the SAS Programming Runtime that can be enabled to allow a batch job to resume execution of a long-running process from a specific point in case of failure or interruption.

When enabled, SAS saves the state of the job at marked checkpoints to the SASWORK location.

This means that if you want Checkpoint-Restart, then SASWORK must be saved to storage on an RWX shared file system. Local, ephemeral storage mounted RWO won't work for Checkpoint-Restart.

That's a big "if". Implementing shared RWX storage that's sufficient for SASWORK's high I/O throughput expectations to support long, sequential reads and writes of the data multiplied across concurrent SAS Compute sessions is a significant architectural consideration.



Using a custom SASWORK

The official steps

You can find official instructions in the documents under sas-bases:

```
$deploy/sas-bases/docs/
sas_programming_environment_storage_tasks.htm
```

1. Copy the provided example file from **sas-bases** to **site-config**
2. Replace the `{{ VOLUME-STORAGE-CLASS }}` variable with the desired storage specification (type, node-selection, etc.)
3. Add a reference to this new site-config file into the base kustomization.yaml
4. Rebuild and apply the manifest (site.yaml)

The next time a Compute Server pod starts, it uses the updated definition.

Configuring the location of SASWORK and UTILLOC are explained in the SAS documentation, specifically in the README files included with your SAS Viya order assets.

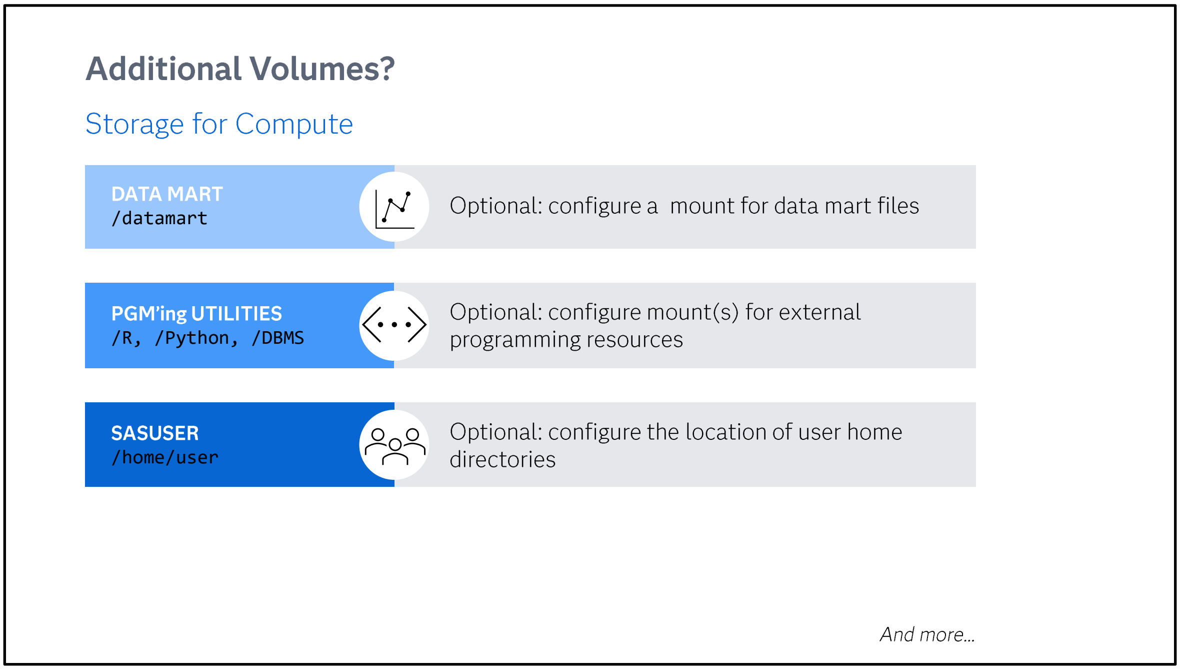
The idea is to start by copying the example file to your Viya deployment's site-config directory.

Then replace the Volume Storage Class placeholder with a specification that describes your desired storage solution.

Then we edit the base kustomization file to include a reference to this new YAML file in site-config.

And then we can use kustomize to rebuild the site manifest, followed by kubectl to apply it to the cluster. This change can be made at time of initial deployment or later as needed.

New instances of the SAS Compute Server (and Batch and Connect) will pick up the updated definition when they start up.



Additional Volumes?

Storage for Compute

DATA MART
`/datamart`

Optional: configure a mount for data mart files

PGM'ing UTILITIES
`/R, /Python, /DBMS`

Optional: configure mount(s) for external programming resources

SASUSER
`/home/user`

Optional: configure the location of user home directories

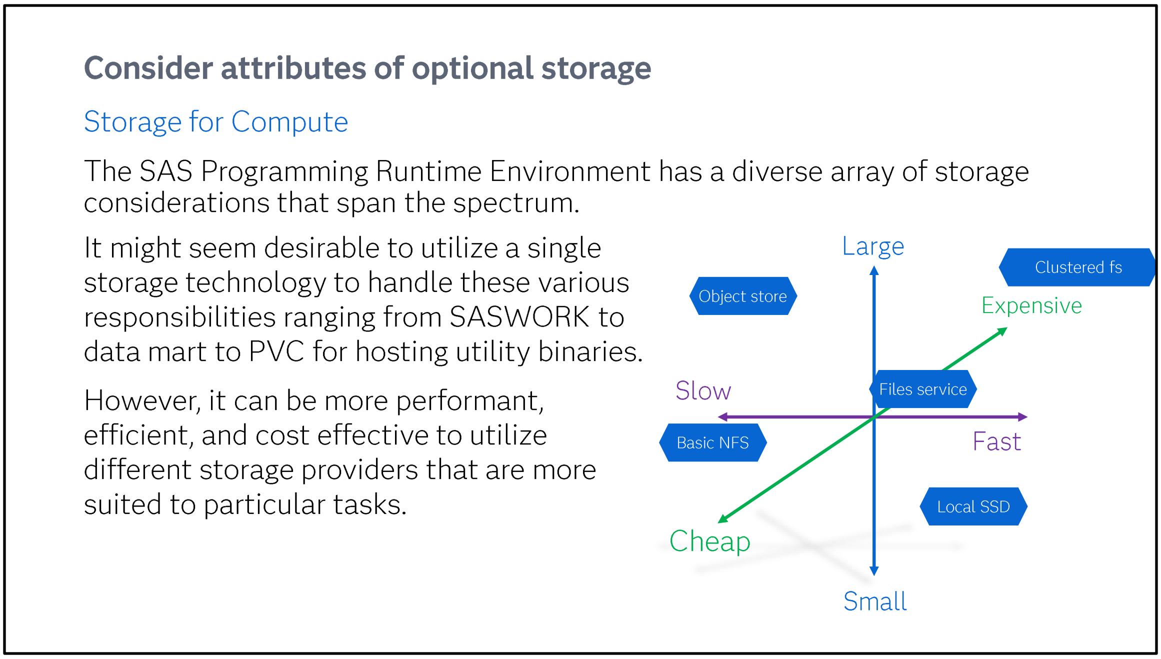
And more…

Besides SASWORK, there are several other storage volumes that SAS Compute might need to operate fully at your site. We'll look at a few here.

First off is the possibility of data mart files. That is, files containing data for SAS Compute to access directly. Typically, they'd be SAS data sets or CSV files: data in formats that SAS can work with natively. The location of these files could be almost anywhere – co-located in the cloud using a Files-based managed service or shared over NFS from on-premise servers or something else.

Next up is whether your Viya environment is set up to integrate with programming utilities like Python and the R programming language, as well as drivers to enable SAS/ACCESS to connect to external databases. Often, the binaries for these utilities are installed onto persistent volumes that are then attached to the SAS Compute pods followed by additional configuration to fully integrate with SAS Viya.

In SAS, the SASUSER library is typically mapped to the user's home directory on the host system. For Kubernetes-based environments like SAS Viya, this is not enabled by default. However, it can be setup through the use of persistent volumes along with configuration of the authentication mechanisms to get it working.



Consider attributes of optional storage

Storage for Compute

The SAS Programming Runtime Environment has a diverse array of storage considerations that span the spectrum.

It might seem desirable to utilize a single storage technology to handle these various responsibilities ranging from SASWORK to data mart to PVC for hosting utility binaries.

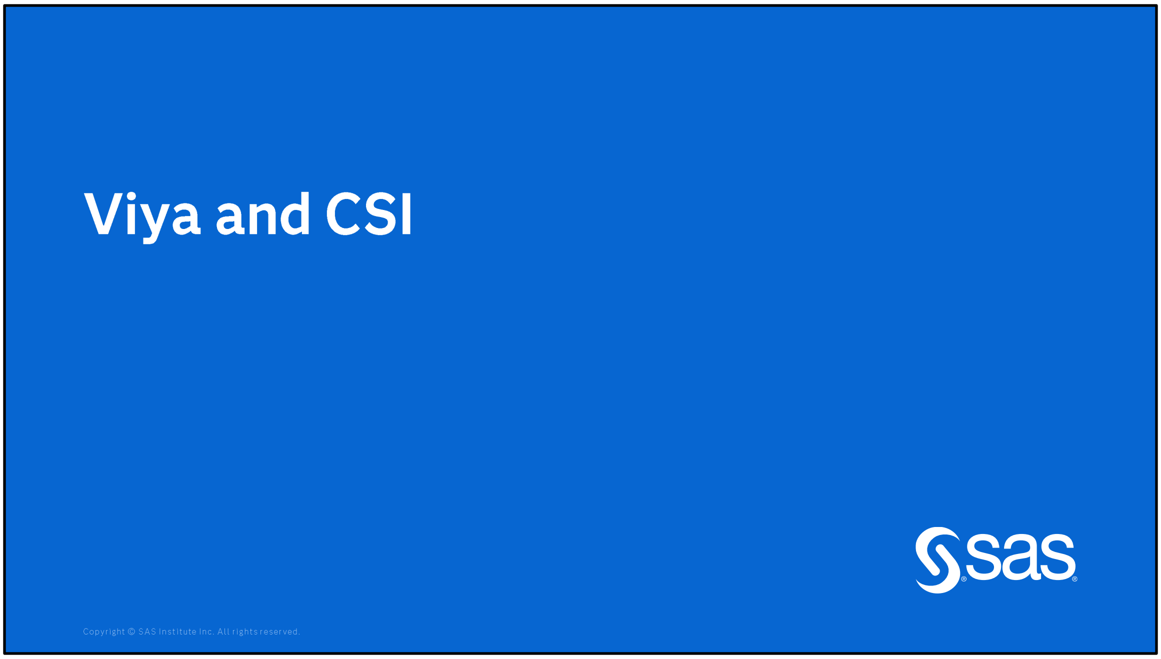
However, it can be more performant, efficient, and cost effective to utilize different storage providers that are more suited to particular tasks.

It’s good to understand the tradeoffs when making storage decisions.

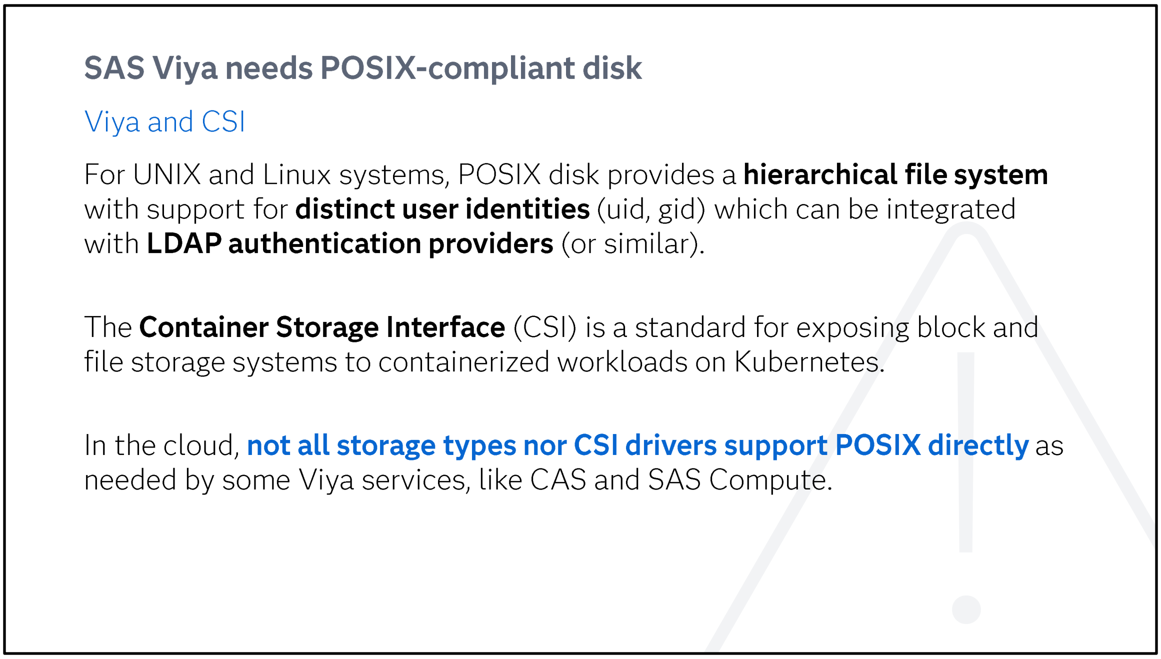
The SAS Programming Runtime Environment has a diverse array of storage considerations that span the spectrum of size, cost, and performance.

For simplicity, it might seem desirable to utilize a single storage technology to handle these various responsibilities ranging from SASWORK to data mart to PVC for hosting utility binaries.

However, it can be more performant, efficient, and cost effective to utilize different storage providers that are more suited to particular tasks.



Viya and CSI



SAS Viya needs POSIX-compliant disk

Viya and CSI

For UNIX and Linux systems, POSIX disk provides a **hierarchical file system** with support for **distinct user identities** (uid, gid) which can be integrated with **LDAP authentication providers** (or similar).

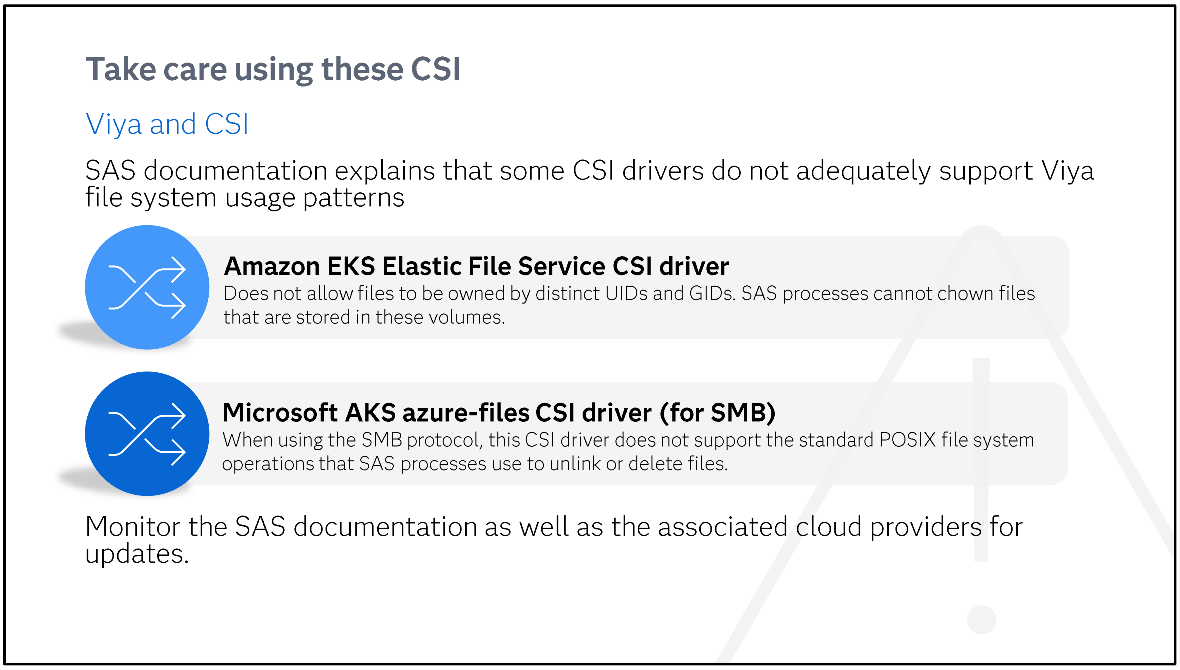
The **Container Storage Interface** (CSI) is a standard for exposing block and file storage systems to containerized workloads on Kubernetes.

In the cloud, **not all storage types nor CSI drivers support POSIX directly** as needed by some Viya services, like CAS and SAS Compute.

For UNIX and Linux systems, POSIX disk provides a hierarchical file system with support for distinct user identities, referenced as UID and GID, which can be integrated with LDAP authentication providers.

The Container Storage Interface (or CSI) is a standard for exposing block and file storage systems to containerized workloads on Kubernetes.

In the cloud, not all storage types nor CSI drivers support POSIX directly as needed by some Viya services, like CAS and SAS Compute.



Take care using these CSI

Viya and CSI

SAS documentation explains that some CSI drivers do not adequately support Viya file system usage patterns

Amazon EKS Elastic File Service CSI driver
Does not allow files to be owned by distinct UIDs and GIDs. SAS processes cannot chown files that are stored in these volumes.

Microsoft AKS azure-files CSI driver (for SMB)
When using the SMB protocol, this CSI driver does not support the standard POSIX file system operations that SAS processes use to unlink or delete files.

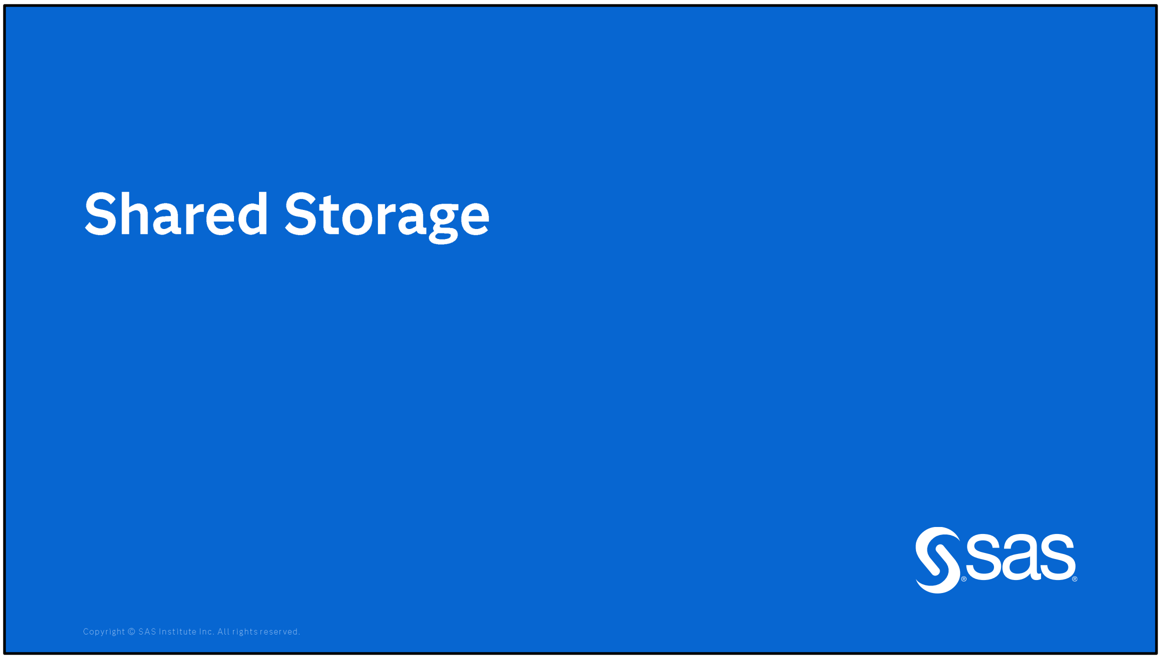
Monitor the SAS documentation as well as the associated cloud providers for updates.

SAS documentation explains that some CSI drivers do not adequately support Viya file system usage patterns. Two in particular are mentioned.

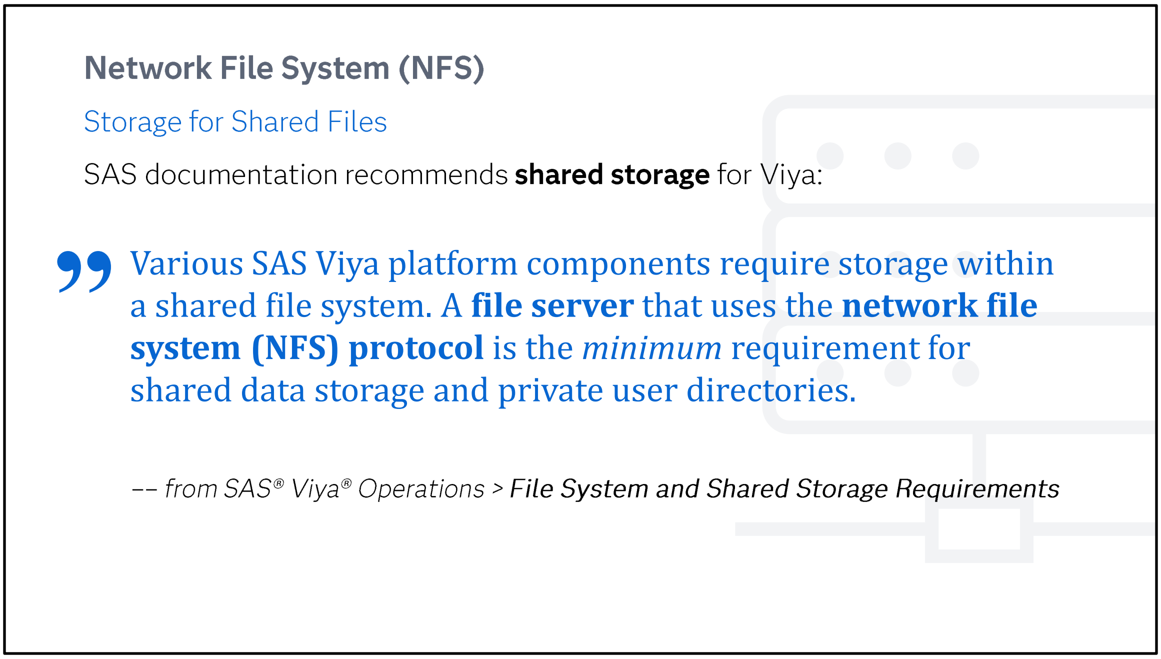
The Amazon EKS Elastic File Service CSI driver does not allow files to be owned by distinct UIDs and GIDs. This means that Viya services like CAS and the SAS Programming Runtime Environment cannot chown files that are stored in these volumes.

Also, the Microsoft AKS azure-files CSI driver has a problem when using the SMB protocol, this CSI driver does not support the standard POSIX file system operations that SAS services use to unlink or delete files. Note that this problem is for SMB only which is typically used for Windows-based OS. It’s fully POSIX compliant using the NFS protocol on Linux systems.

As always, monitor the SAS documentation and confirm the latest information with the associated cloud providers. While these CSI can be used for Viya in some places, you’ll want to avoid them for CAS Disk Cache as well as for SAS Compute volumes like SASWORK and UTILLOC. It’s possible there might be other areas to avoid as well.



Shared Storage



Network File System (NFS)

Storage for Shared Files

SAS documentation recommends **shared storage** for Viya:

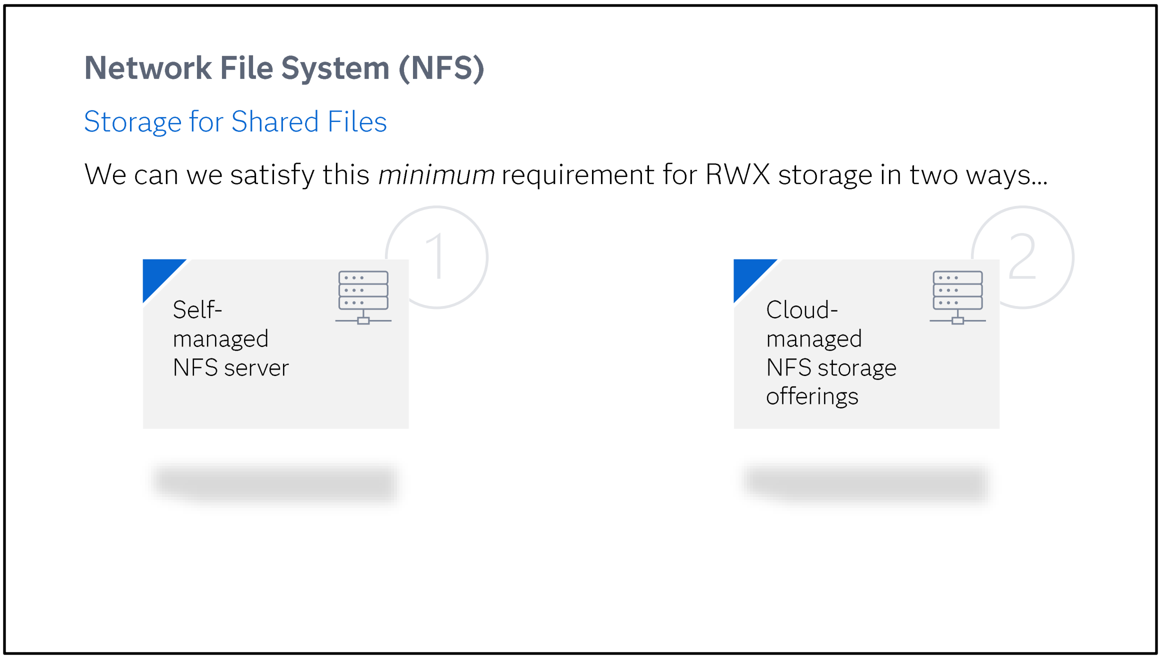
> Various SAS Viya platform components require storage within a shared file system. A **file server** that uses the **network file system (NFS) protocol** is the *minimum* requirement for shared data storage and private user directories.

–– from SAS® Viya® Operations > File System and Shared Storage Requirements

The SAS documentation recommends shared storage for Viya – that is, storage with access mode RWX meaning it can be mounted read-write to multiple pods.

Various SAS Viya platform components require storage within a shared file system. A file server that uses the network file system (or NFS) protocol is the minimum requirement for shared data storage and private user directories.

There are many ways where the NFS protocol can be implemented to support shared disk. The SAS documentation mentions using a “file server” as a minimum, but that’s just one approach.



Network File System (NFS)

Storage for Shared Files

We can we satisfy this *minimum* requirement for RWX storage in two ways…

1

Self-managed NFS server

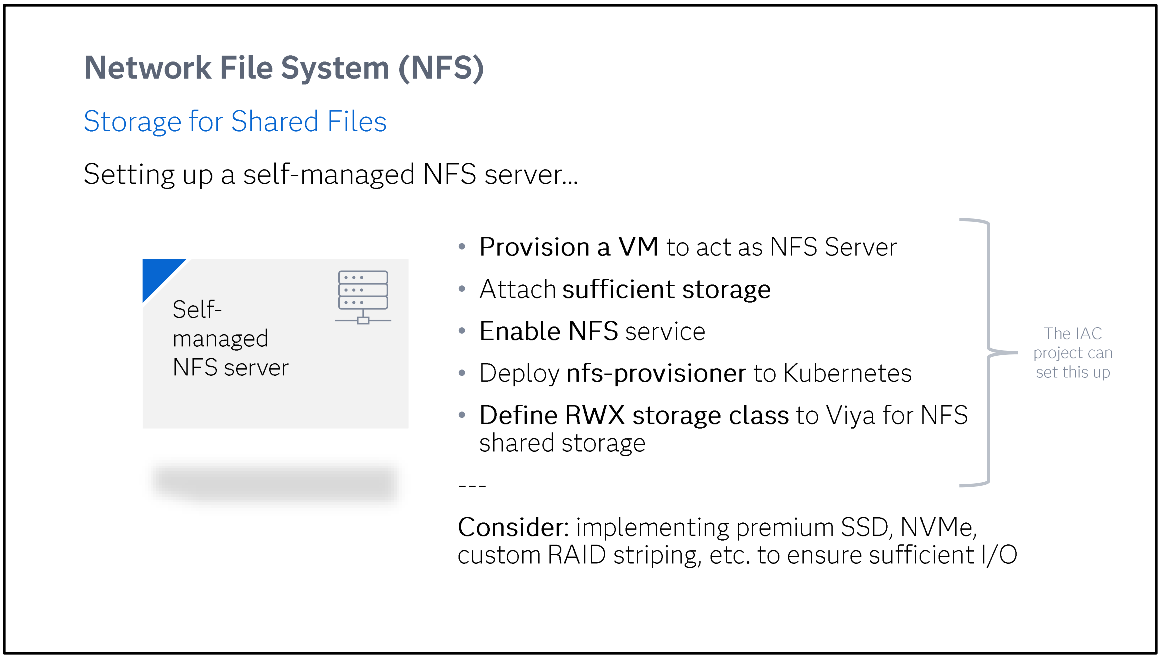
2

Cloud-managed NFS storage offerings

If we focus on setting up storage through the use of the NFS protocol, then there are a couple of ways we can provide RWX storage for Viya.

The first approach might be to stand up a dedicated host to act as an NFS server. Typically, this is something that would operate outside of your Kubernetes cluster and would be maintained directly by your site's IT staff.

Else we could look at the managed storage solutions offered by your cloud provider. These vary by cloud in terms of their exact implementation, but usually offer high availability, robust storage protections, and typically rely on the NFS protocol to mount the volumes to your Viya pods.



Network File System (NFS)

Storage for Shared Files

Setting up a self-managed NFS server…

Self-managed NFS server

- **Provision a VM** to act as NFS Server
- Attach **sufficient storage**
- **Enable NFS** service
- Deploy **nfs-provisioner** to Kubernetes
- **Define RWX storage class** to Viya for NFS shared storage

The IAC project can set this up

Consider: implementing premium SSD, NVMe, custom RAID striping, etc. to ensure sufficient I/O

To set up a standalone NFS server means you need to:

Provision a VM instance to act as an NFS server in your environment

Attach sufficient storage to the instance that provides enough space and can support the I/O throughput expected.

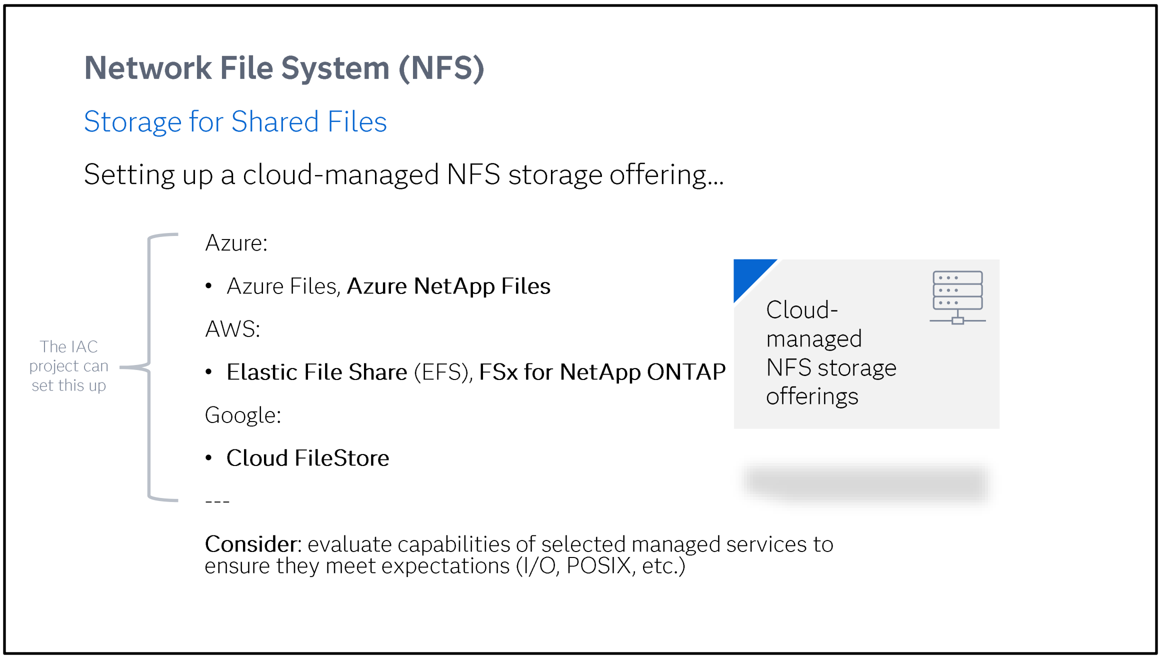
Enable the NFS service daemon in the instance

Over in your Kubernetes cluster, you'll need to install something like the Kubernetes NFS Subdir External Provisioner.

And define a storage class with RWX access mode that the Viya pods can reference for their shared storage needs.

Note that the SAS Infrastructure-as-Code project can set up an NFS server if you choose that configuration option.

Depending on expected number of users, concurrent workloads, data volumes, and other considerations, then you'll want to evaluate the disk technologies available to your site. Premium SSD or NVMe drives can handle much higher throughput rates than conventional spinning hard-disk drives. To further expand volumes or increase throughput, also consider implementing RAID striping or similar technologies as well.



Network File System (NFS)

Storage for Shared Files

Setting up a cloud-managed NFS storage offering…

Azure:

- Azure Files, **Azure NetApp Files**

AWS:

- **Elastic File Share** (EFS), **FSx for NetApp ONTAP**

Google:

- **Cloud FileStore**

Consider: evaluate capabilities of selected managed services to ensure they meet expectations (I/O, POSIX, etc.)

The IAC project can set this up

Cloud-managed NFS storage offerings

Instead of standing up and managing your own NFS-based storage, you could opt to use the shared storage technologies offered by your cloud provider.

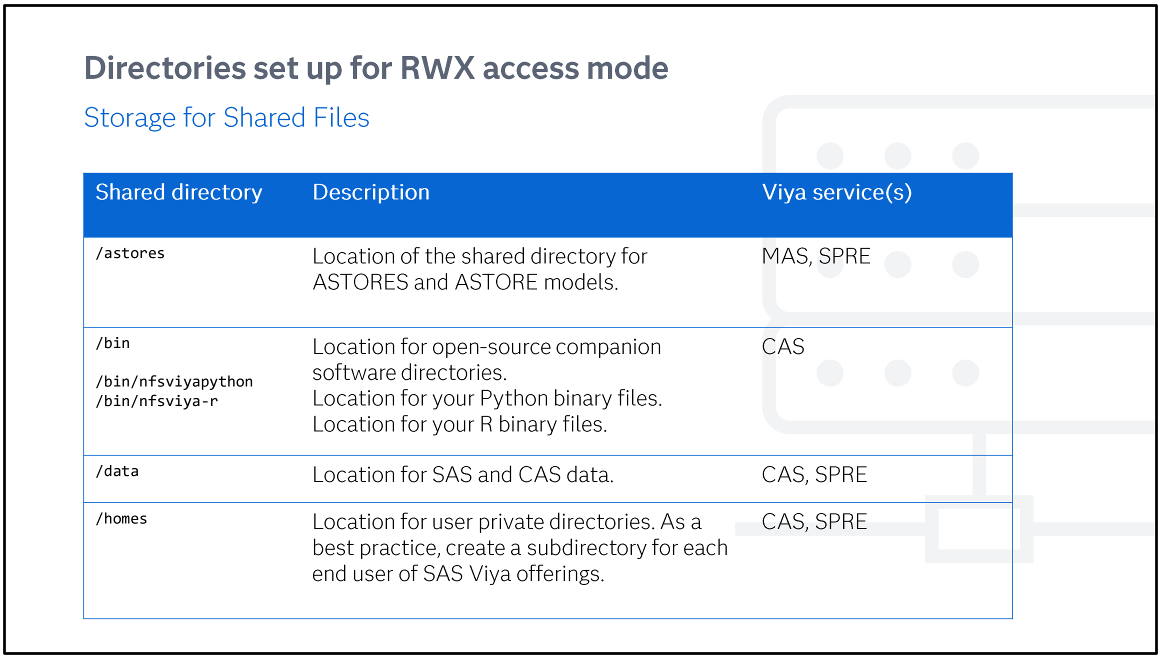
Azure offers very low-cost Azure Files, but that might not be sufficient in critical production applications. Instead, you'll probably want to first look at using Azure NetApp Files which have more capabilities to better support the use-cases for Viya.

Amazon offers a pretty good starting point with its Amazon Elastic File Share, or EFS. Additional performance can be found in other offerings like Amazon FSx for Lustre, Amazon FSx for NetApp ONTAP, or similar high-performance storage solutions.

Google offers its Cloud FileStore solution for shared storage, too.

Note that the SAS Infrastructure-as-Code project can set up a cloud-managed block storage for RWX volumes if you select that configuration option.

Also, remember to evaluate the capabilities of selected managed services to ensure they meet expectations in terms of sustained I/O throughput, POSIX compliance, and so on.



Directories set up for RWX access mode

Storage for Shared Files

Shared directory	Description	Viya service(s)
`/astores`	Location of the shared directory for ASTORES and ASTORE models.	MAS, SPRE

`/bin`
`/bin/nfsviyapython`
`/bin/nfsviya-r`
	Location for open-source companion software directories.
	Location for your Python binary files.
	Location for your R binary files.
		CAS

`/data`	Location for SAS and CAS data.	CAS, SPRE
`/homes`	Location for user private directories. As a best practice, create a subdirectory for each end user of SAS Viya offerings.	CAS, SPRE

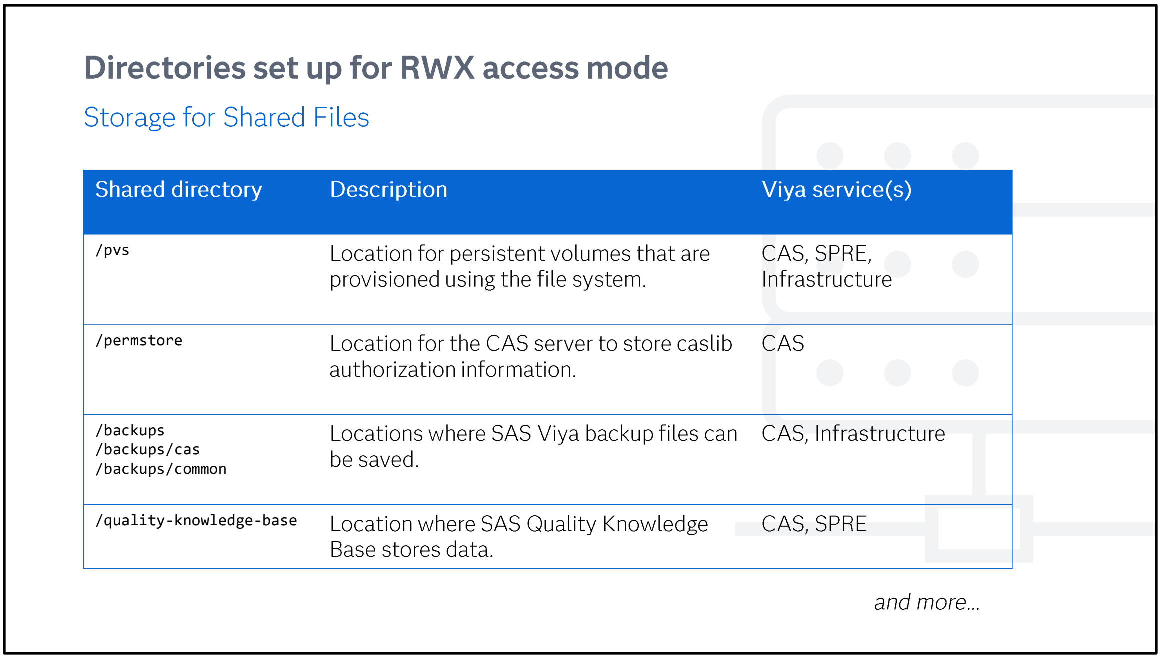
If we crack open the pod specifications for some of the Viya services, then we can find the directories which are used as mount points to connect to the shared RWX storage volumes. We'll look at several of them here.

The first is the “a-stores” directory which is home to analytic stores and score resources used by the SAS Micro Analytics Service and Compute service jobs.

We also have directories under “/bin” used by CAS to access third-party open-source programming files like Python and R.

Another directory that's commonly configured is “/data” as a default location for data mart files for use by CAS and the Compute services.

We can also configure Viya to access user home directories, if needed. By default, these mount at the “/homes” path on CAS and Compute services pods.



Directories set up for RWX access mode

Storage for Shared Files

Shared directory	Description	Viya service(s)
`/pvs`	Location for persistent volumes that are provisioned using the file system.	CAS, SPRE, Infrastructure
`/permstore`	Location for the CAS server to store caslib authorization information.	CAS

`/backups`
`/backups/cas`
`/backups/common`
	Locations where SAS Viya backup files can be saved.	CAS, Infrastructure

`/quality-knowledge-base`	Location where SAS Quality Knowledge Base stores data.	CAS, SPRE

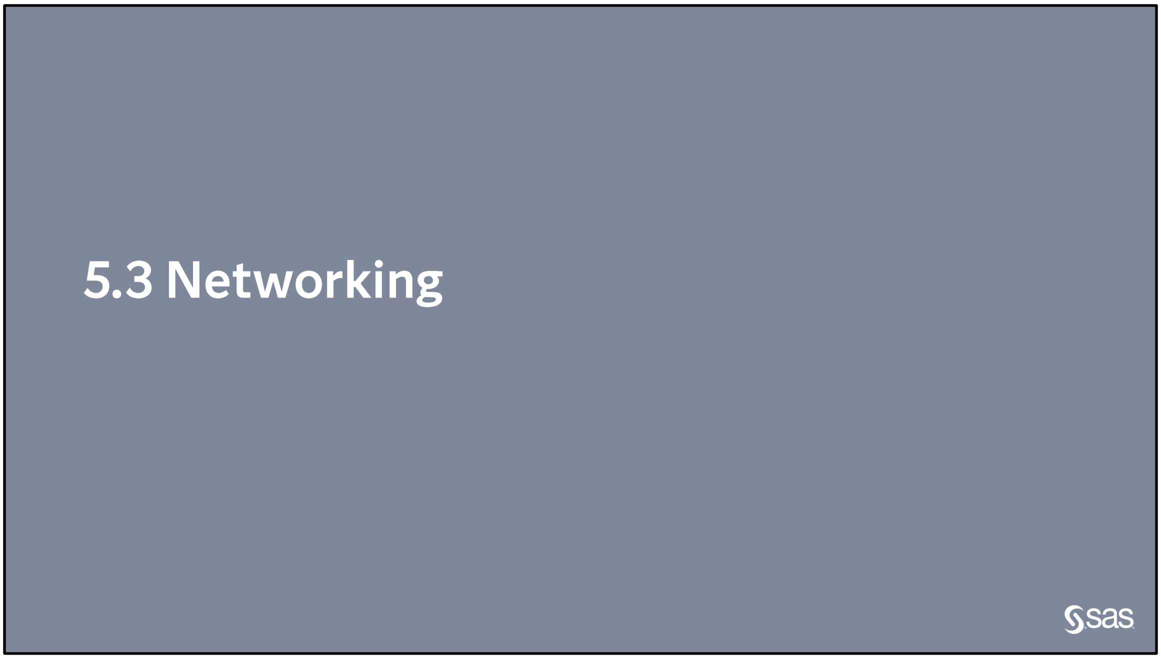
and more…

Continuing on with more examples of shared storage volumes, we can provide a top-level directory called “/pvs” to be the location for persistent volumes that are provisioned using the file system. This might get used by many of the Viya components including CAS, Compute, and other Viya services.

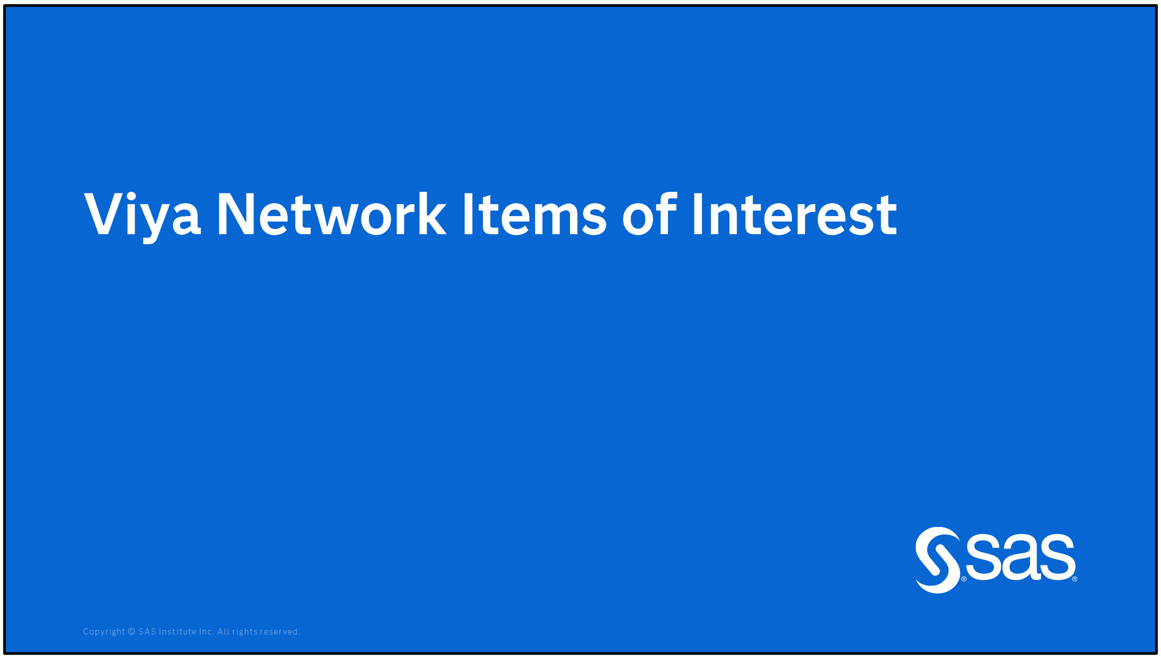
The “/permstore” location is used by CAS to store authorization (or permission) information about data.

And we also can set up locations for backups processing as well as for the Quality Knowledge Base, if in use at your site.

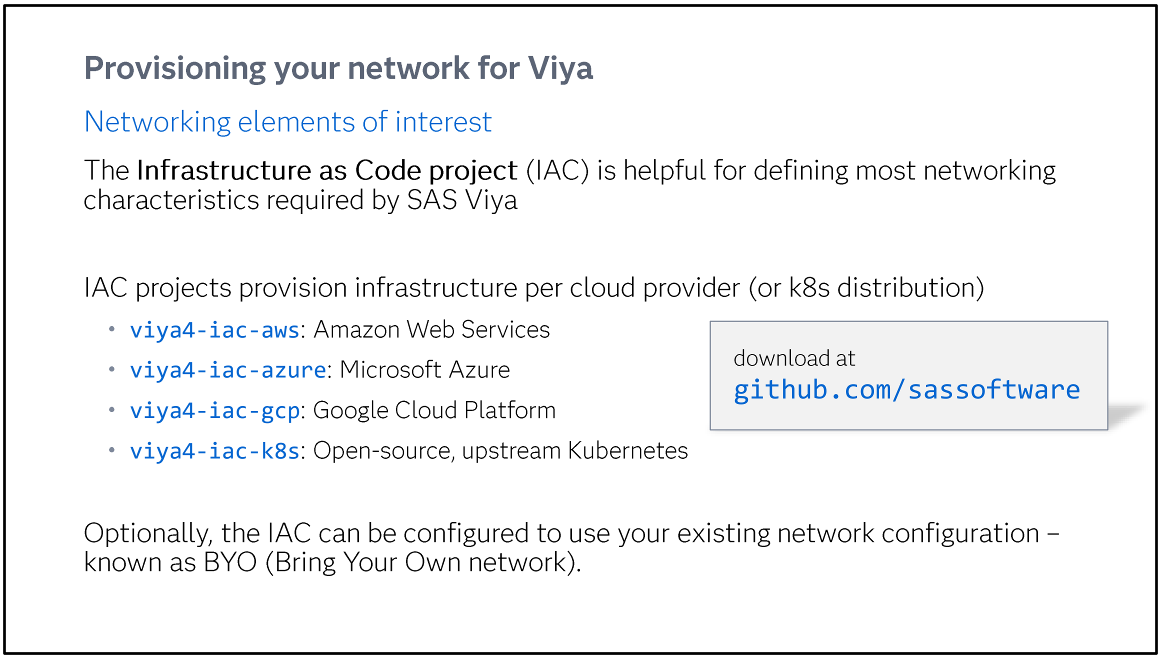
There are more locations that just those shown here – but this helps set the stage for understanding some of the shared storage considerations that Viya requires.



5.3 Networking



Viya Network Items of Interest



Provisioning your network for Viya

Networking elements of interest

The **Infrastructure as Code project** (IAC) is helpful for defining most networking characteristics required by SAS Viya

IAC projects provision infrastructure per cloud provider (or k8s distribution)

- `viya4-iac-aws`: Amazon Web Services
- `viya4-iac-azure`: Microsoft Azure
- `viya4-iac-gcp`: Google Cloud Platform
- `viya4-iac-k8s`: Open-source, upstream Kubernetes

download at
`github.com/sassoftware`

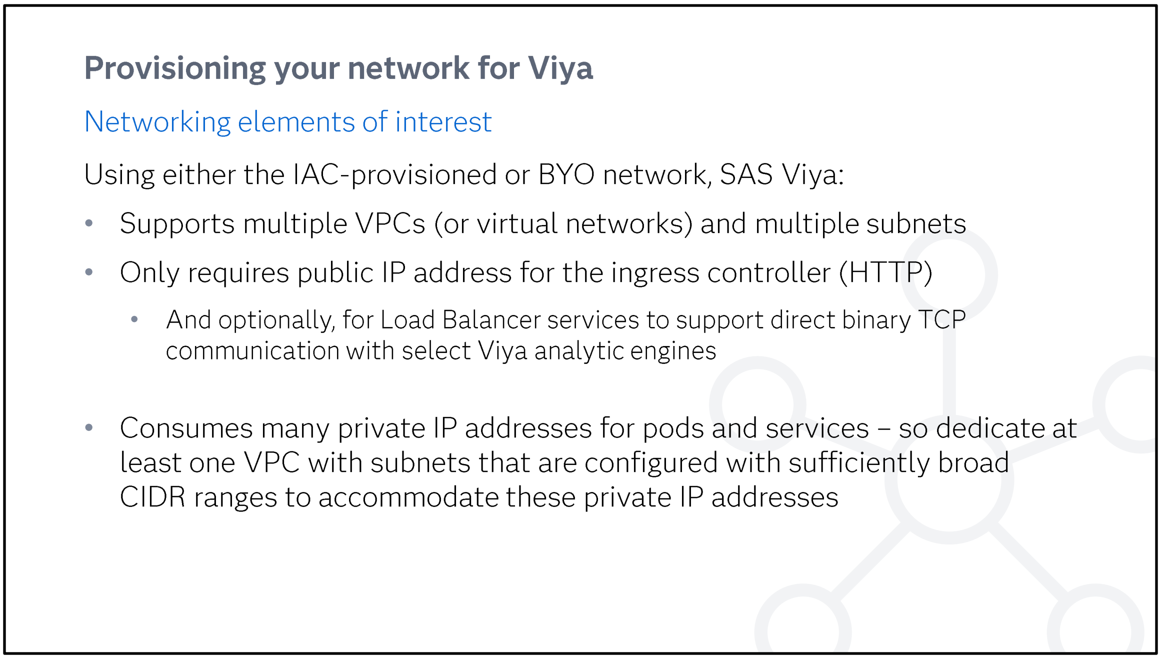
Optionally, the IAC can be configured to use your existing network configuration – known as BYO (Bring Your Own network).

SAS offers prescriptive guidance for many aspects of infrastructure provisioning, including network characteristics and setup, through the Infrastructure as Code project.

The network is a major component of your site's infrastructure to support SAS Viya's deployment to Kubernetes. By default, the Infrastructure as Code project will configure the network to meet Viya's minimum requirements.

The Infrastructure as Code project actually consists of four different projects broken out by cloud provider as shown here. Each one is optimized to provision infrastructure as suits the operational requirements of the associated provider. As a reminder, the Infrastructure as Code project is publicly available for use and is hosted at github dot com in the SAS software repository.

Note that if your site has established rules, conventions, or otherwise prefers to manage the network configuration directly, that's accommodated as well. In the Infrastructure as Code project, we refer to that as bring your own network. That's fine as long as your site's network configuration satisfies Viya's requirements and consumption patterns.



Provisioning your network for Viya

Networking elements of interest

Using either the IAC-provisioned or BYO network, SAS Viya:

- Supports multiple VPCs (or virtual networks) and multiple subnets
- Only requires public IP address for the ingress controller (HTTP)
  - And optionally, for Load Balancer services to support direct binary TCP communication with select Viya analytic engines
- Consumes many private IP addresses for pods and services – so dedicate at least one VPC with subnets that are configured with sufficiently broad CIDR ranges to accommodate these private IP addresses

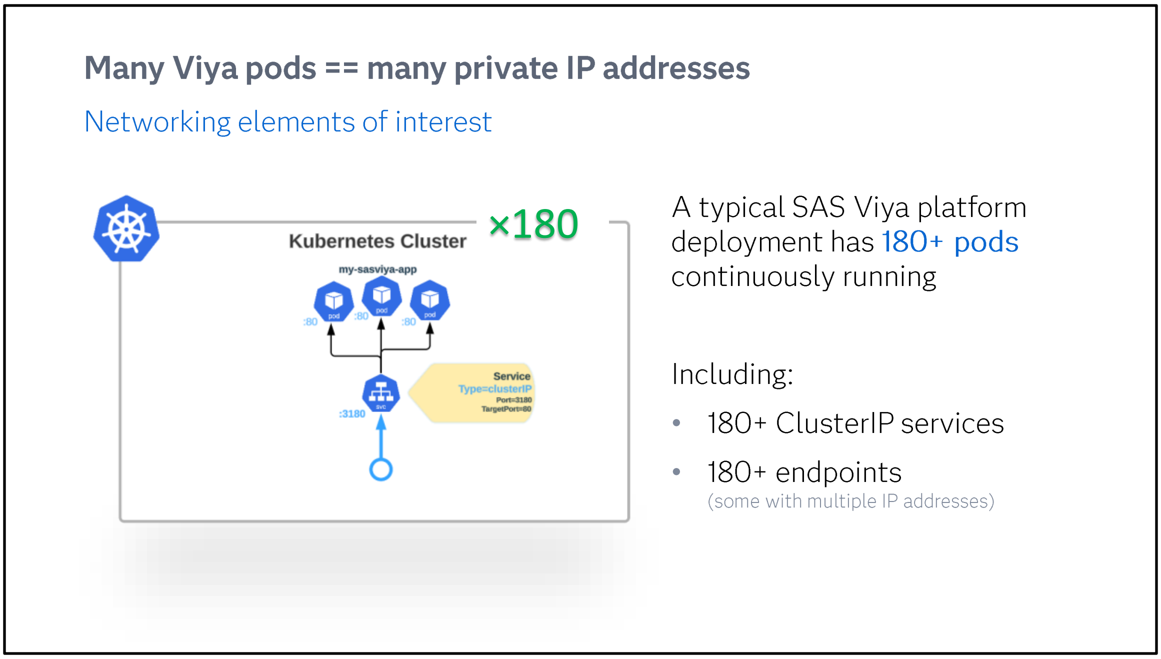
Whether you're using the networking that's set up by the Infrastructure as Code project or you're bringing your own network configuration, the Viya platform supports running across multiple VPCs or virtual networks as well as multiple subnets per deployment.

Access to Viya services over HTTP is satisfied by the use of Ingresses. So only the Ingress Controller requires a public IP address for HTTP traffic to reach Viya from outside the Kubernetes cluster.

Note that the term public in this usage means that the IP address is routable outside the internal VNet. Exposing the public IP address on the world-wide Internet is not required. However, Viya clients that are positioned both inside and outside the Viya platform in Kubernetes must be able to find a network route to the cluster’s ingress endpoint.

If direct-access to the native, binary ports of Viya’s analytic engines is needed – that is, not HTTP - then optional Load Balancer services can be established to support that as well.

And finally, inside the cluster Kubernetes needs a large enough private IP space to accommodate the many pods and services that provide the Viya platform. Expect to dedicate at least one VPC with subnets that provide CIDR ranges broad enough to support the planned number of private IP addresses.



Many Viya pods == many private IP addresses

Networking elements of interest

A typical SAS Viya platform deployment has 180+ pods continuously running

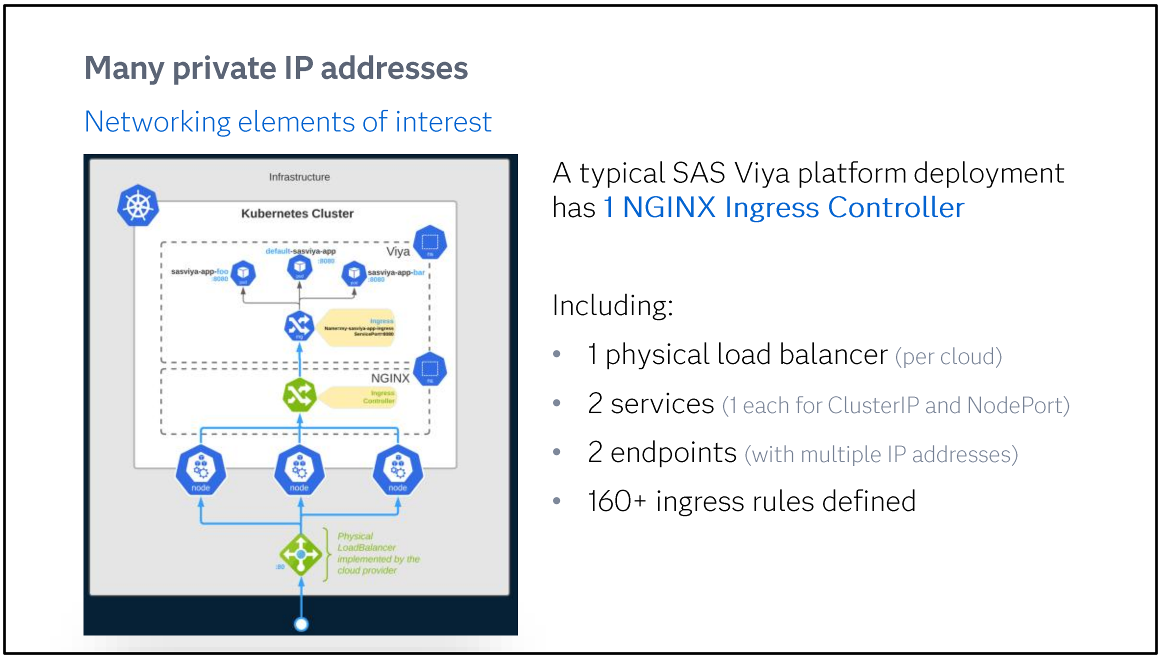
Including:

- 180+ ClusterIP services
- 180+ endpoints
  (some with multiple IP addresses)

This over-simplified illustration calls to attention that the Viya platform employs a number of software services in the Kubernetes cluster. Depending on the technologies in play, some of those services might be replicated once, twice, many times over, or not all, just running as singletons.

If we deploy a typical Viya platform to Kubernetes and let it sit idle, it could be running over 180 pods. That number could climb much higher depending on availability and performance requirements.

The point is that many Viya pods means many service endpoints – and some of those endpoints will have multiple IP addresses. Depending on your infrastructure, these private IP addresses can significantly eat into the available CIDR range at some sites. Usually there are multiple approaches to dealing with this, from VPC and subnet placement to employing alternative Container Network Interface frameworks for dynamic network resource allocation.



Many private IP addresses

Networking elements of interest

A typical SAS Viya platform deployment has 1 NGINX Ingress Controller

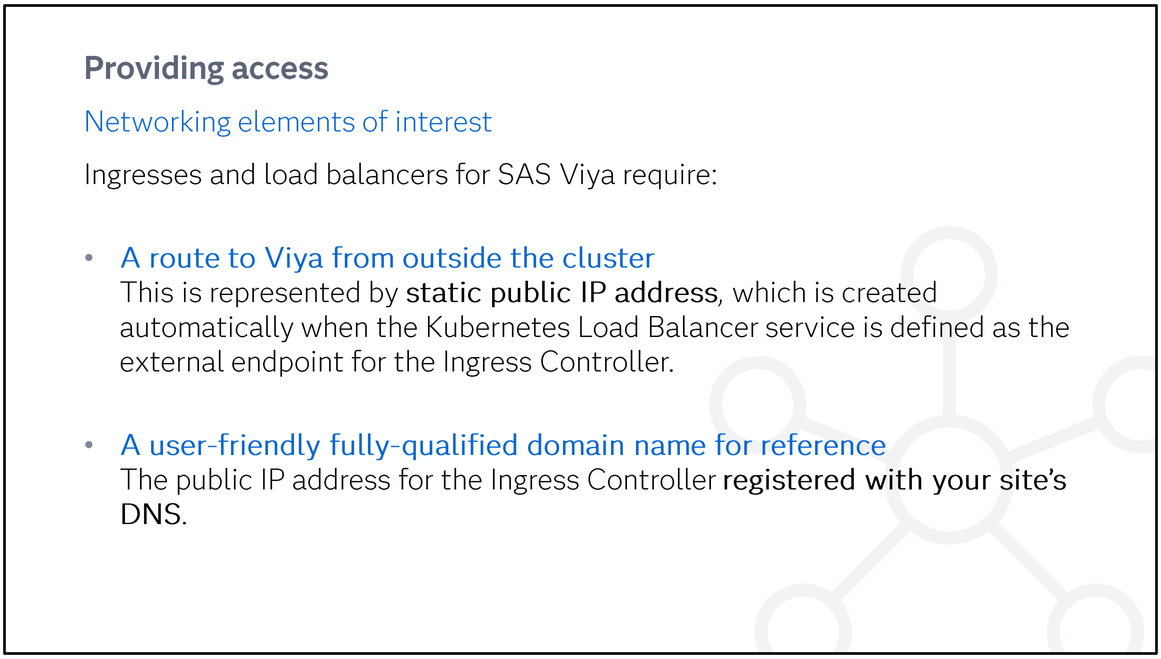
Including:

- 1 physical load balancer (per cloud)
- 2 services (1 each for ClusterIP and NodePort)
- 2 endpoints (with multiple IP addresses)
- 160+ ingress rules defined

A typical SAS Viya platform deployment only needs one Nginx Ingress Controller.

That will include one physical load balancer as implemented by the infrastructure provider along with Kubernetes Cluster IP and Node Port services so it will have two endpoints with multiple IP addresses.

The point is that with a single point of contact, we can expose a very large number of Viya services through a well-established and secure communication protocol.



Providing access

Networking elements of interest

Ingresses and load balancers for SAS Viya require:

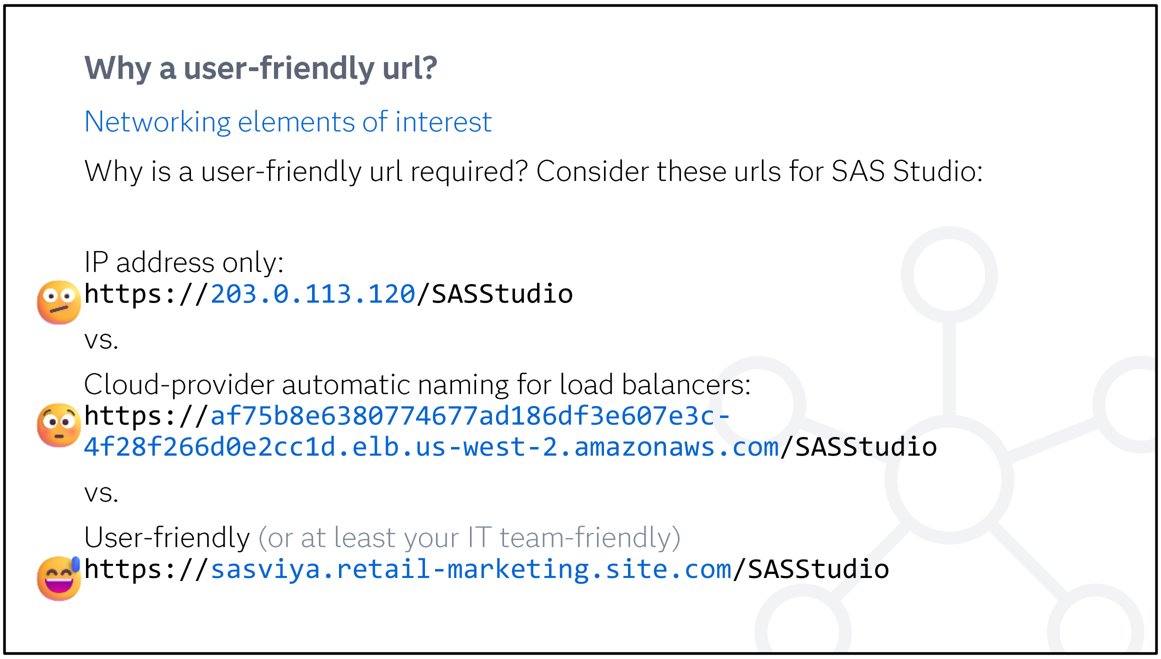
- A route to Viya from outside the cluster
  This is represented by **static public IP address**, which is created automatically when the Kubernetes Load Balancer service is defined as the external endpoint for the Ingress Controller.

- A user-friendly fully-qualified domain name for reference
  The public IP address for the Ingress Controller **registered with your site's DNS**.

Ingresses and load balancers that provide support to the SAS Viya platform will require a couple of items.

The first is a route to reach Viya services from outside the cluster. Typically, this is satisfied by the steps taken to provide a static public IP address. The IP address is created automatically when a Kubernetes load balancer is defined as the external endpoint for the Ingress Controller. Ensuring its routable for use by end users is a function of the network infrastructure provisioning and configuration.

We also expect a user-friendly fully qualified domain name for users to reference. The FQDN will be registered with your site's DNS provider to redirect to the public IP address of the Ingress Controller. This is not a step that the Infrastructure as Code project can perform – but is usually handled by your site's IT team.



Why a user-friendly url?

Networking elements of interest

Why is a user-friendly url required? Consider these urls for SAS Studio:

IP address only:
https://203.0.113.120/SASStudio

vs.

Cloud-provider automatic naming for load balancers:
https://af75b8e6380774677ad186df3e607e3c-4f28f266d0e2cc1d.elb.us-west-2.amazonaws.com/SASStudio

vs.

User-friendly (or at least your IT team-friendly)
https://sasviya.retail-marketing.site.com/SASStudio

Why is a user-friendly url required? Consider these urls for SAS Studio.

The first shown here just relies on the static IP address of the load balancer fronting the Ingress Controller used by the Viya platform. This works fine. But we use domain names for a reason – because IP addresses aren’t meaningful to human beings besides a few folks on the IT team who know the various subnet masks.

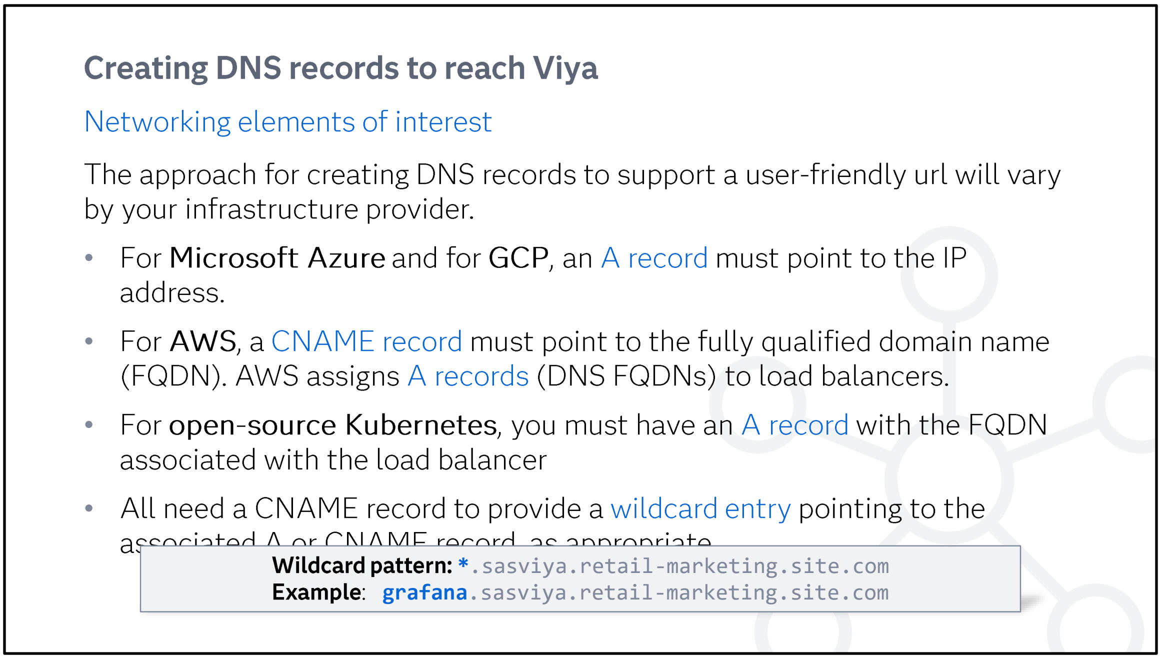
Some cloud providers – like AWS – will automatically generate a random alpha-numeric domain name for the load balancer with the goal of guaranteed uniqueness. It works – and you could bookmark it and never have to type it in – but again, this isn’t user-friendly at all.

Another problem that both of these approaches have is that they’re static. If the resource behind these references goes away and a new one comes up to replace it, then it will likely get a new IP address or ridiculous unique hostname. Then all your users will need to update the bookmark in their browsers to access Viya through the new resource.

This last one is the kind of conventional looking url that users expect. It can be defined to follow the IT conventions in use at your site with specific subdomains and structure as needed.

This domain name would get entered into your site’s DNS provider to resolve it to the backend IP address. That’s a task that your site’s IT team is responsible for.

The nice thing about this approach is that if we consider the problem where a new resource comes up to replace the old one such that the backend IP address changes, your site IT team can update the DNS. Then the next time a user's web browser accesses the friendly name, they will get routed automatically to the new resource - ideally, they won't even realize a change to network addressing even occurred.



Creating DNS records to reach Viya

Networking elements of interest

The approach for creating DNS records to support a user-friendly url will vary by your infrastructure provider.

- For **Microsoft Azure** and for **GCP**, an A record must point to the IP address.
- For **AWS**, a CNAME record must point to the fully qualified domain name (FQDN). AWS assigns A records (DNS FQDNs) to load balancers.
- For **open-source Kubernetes**, you must have an A record with the FQDN associated with the load balancer
- All need a CNAME record to provide a wildcard entry pointing to the associated A or CNAME record, as appropriate.

Wildcard pattern: `*.sasviya.retail-marketing.site.com`
Example: `grafana.sasviya.retail-marketing.site.com`

The approach for creating DNS records to support a user-friendly url will vary by your infrastructure provider. The SAS documentation explains the following.

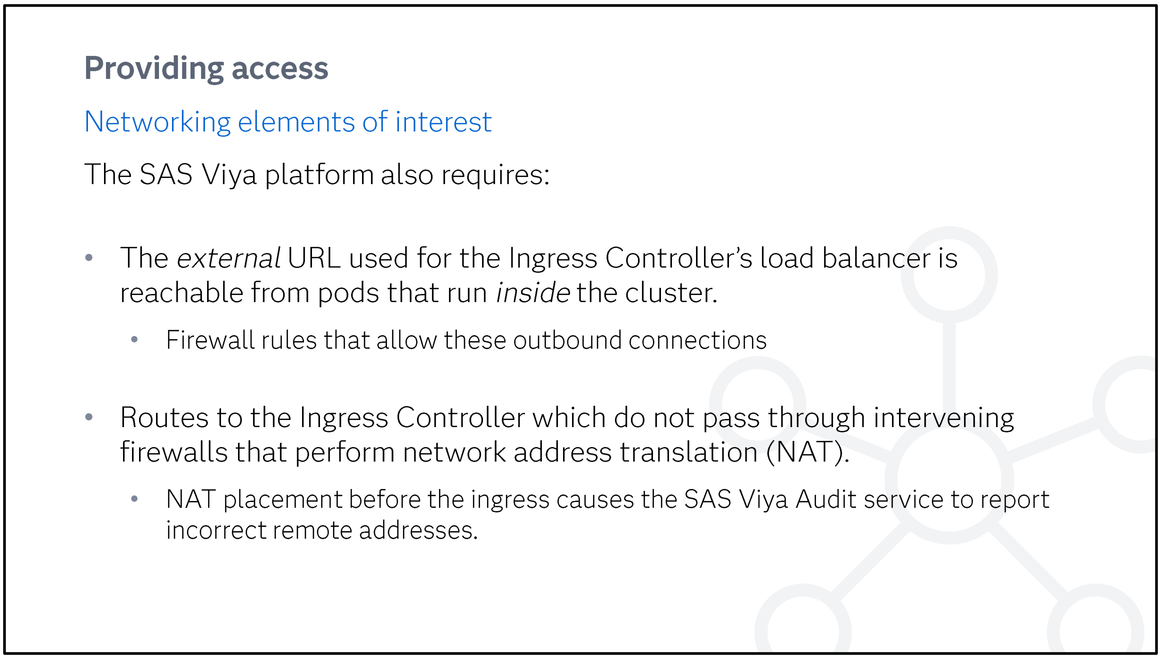
For Microsoft Azure and for GCP, a DNS A record must point to the IP address of the load balancer for the Ingress Controller.

For AWS, a CNAME record must point to the fully qualified domain name as an alias in lieu of an A record - thanks to that crazy long unique domain name that AWS defines for elastic load balancers. For its part, AWS takes the responsibility to assign A records to load balancers IP addresses.

For open-source Kubernetes, you must have an A record with the FQDN associated with the load balancer.

And then lastly, you should also define a wildcard alias as a CNAME record that references the previous A record or CNAME record, as appropriate.

What is a wildcard alias? It’s a simple pattern-matching approach to handle domain names which include arbitrary prefixes. Some third-party software deployed alongside the Viya platform employ prefixes that lead in front of the domain name as opposed to paths that you typically see following the domain name. Shown here, notice that the wildcard in this case is the asterisk. So, if the DNS gets a request for any domain name that ends with the example here, regardless of the prefix, it will route to the same Ingress Controller used by Viya – and from there get directed by the appropriate rule to the specified resource running in Kubernetes.



Providing access

Networking elements of interest

The SAS Viya platform also requires:

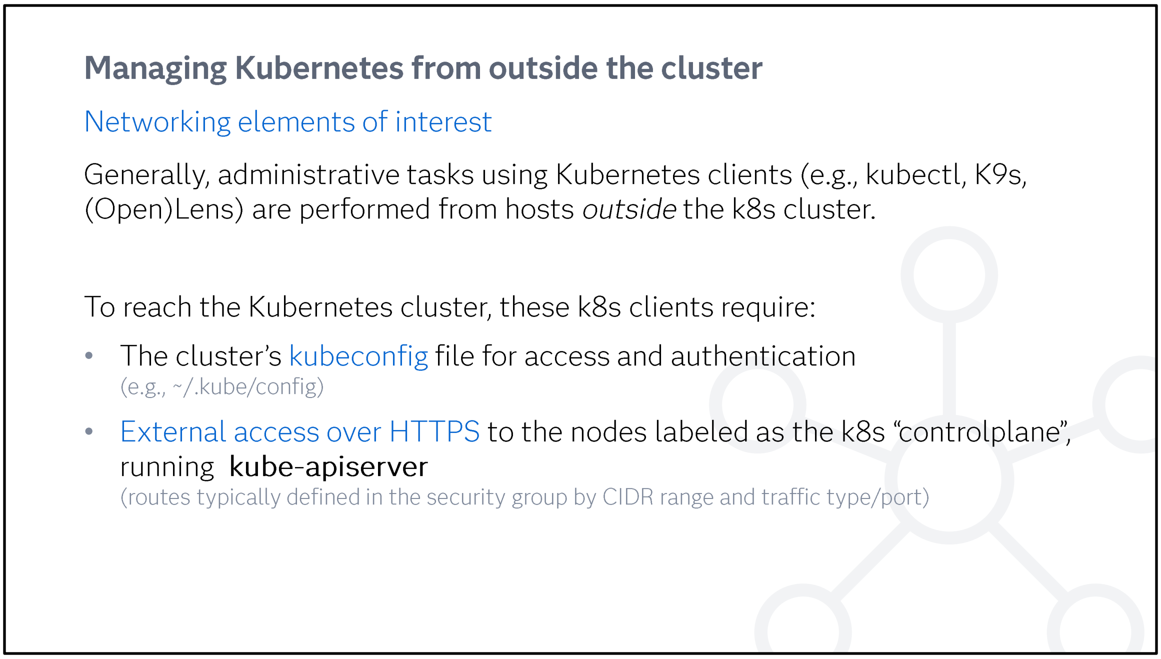
- The *external* URL used for the Ingress Controller's load balancer is reachable from pods that run *inside* the cluster.
  - Firewall rules that allow these outbound connections
- Routes to the Ingress Controller which do not pass through intervening firewalls that perform network address translation (NAT).
  - NAT placement before the ingress causes the SAS Viya Audit service to report incorrect remote addresses.

The Viya platform also requires that the external url is reachable from pods running inside the cluster. Generally, this kind of network routing is not ideal. It's similar in concept to exiting your house and re-entering through the front door instead of just going from one room to the adjacent one. However, there are circumstances where this is needed and must be satisfied.

One item to watch for is to ensure that the security group rules allow that kind of outbound traffic from the Kubernetes cluster.

Another challenge to avoid is the use of network address translation on the route to the Ingress Controller.

If network address translation is in play, then services like the SAS Viya Audit service will incorrectly report the wrong IP addresses for remote clients.



Managing Kubernetes from outside the cluster

Networking elements of interest

Generally, administrative tasks using Kubernetes clients (e.g., kubectl, K9s, (Open)Lens) are performed from hosts *outside* the k8s cluster.

To reach the Kubernetes cluster, these k8s clients require:

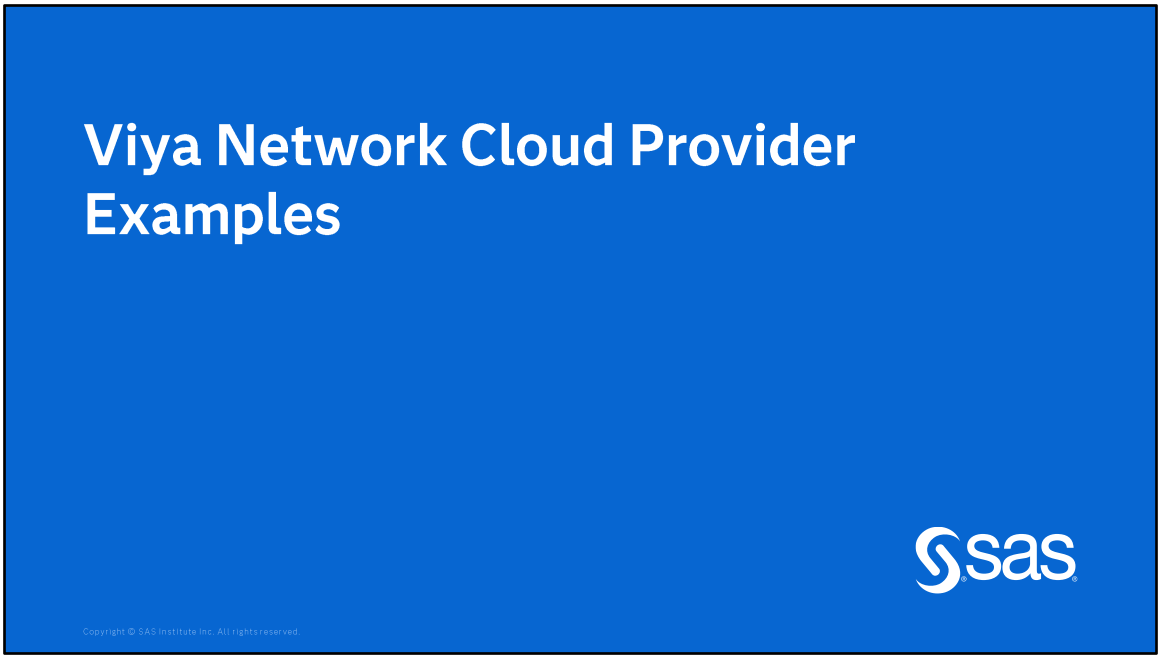
- The cluster's kubeconfig file for access and authentication
  (e.g., ~/.kube/config)
- External access over HTTPS to the nodes labeled as the k8s "controlplane", running `kube-apiserver`
  (routes typically defined in the security group by CIDR range and traffic type/port)

Kubernetes administrators generally operate their Kubernetes clients on hosts outside of the cluster – like their own PC or possibly a jumpbox near the cluster.

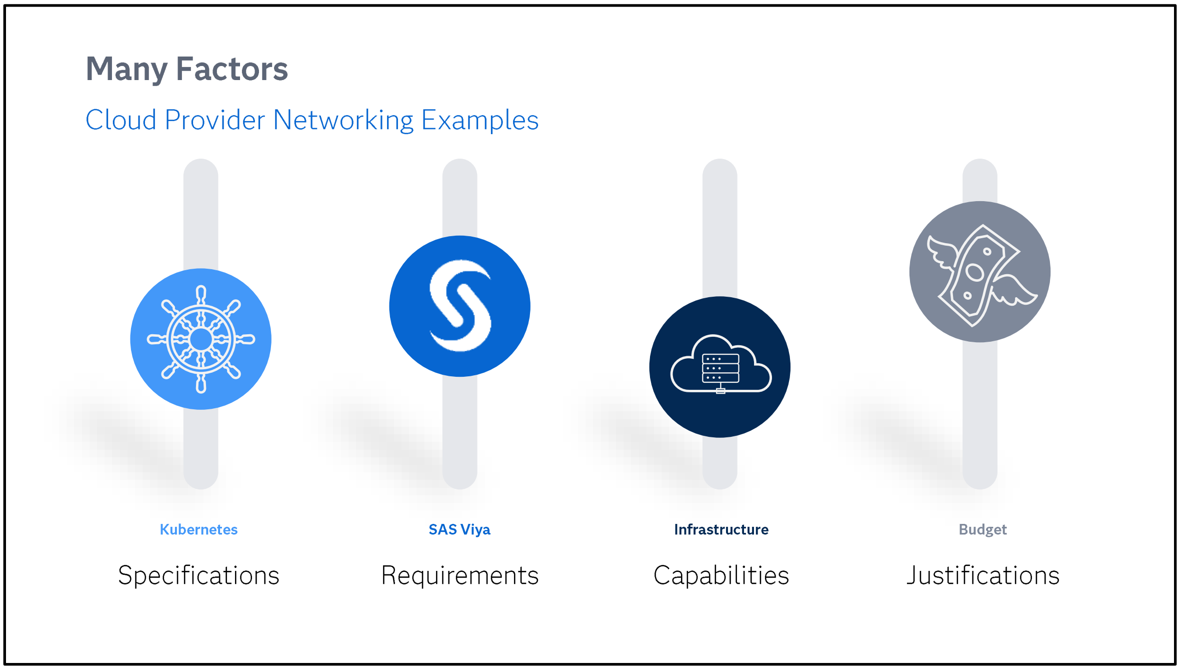
Kubernetes clients like kubectl, K9s, and Open Lens have a couple of requirements to communicate with the cluster.

First of all, they need the kubeconfig file which is provided by the site's Kubernetes administrator. This file explains to the client where to find the Kubernetes cluster on the network and how to communicate with it. It also includes authentication credentials to identify your role and level of access.

The clients also need a network route from the host where they're running to the nodes acting as the Kubernetes control plane. This communication is TLS encrypted as HTTPS traffic with the kube API server listening at port four four three.



Viya Network Cloud Provider Examples



Many Factors

Cloud Provider Networking Examples

Kubernetes

Specifications

SAS Viya

Requirements

Infrastructure

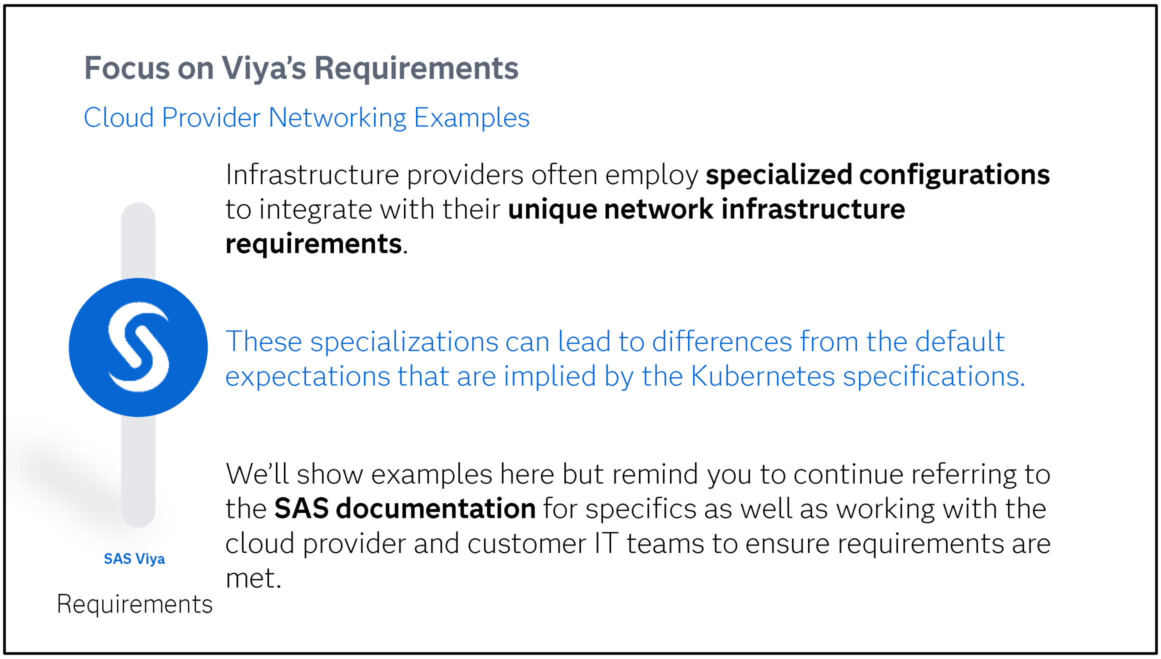
Capabilities

Budget

Justifications

There are many, many factors to weigh when planning the architecture of SAS Viya. Even just focusing on the networking considerations requires looking at numerous factors and controls that will affect the footprint, cost, and administration of the Viya platform.

In this module, we'll touch on some examples where these factors overlap and influence each other. In particular, we'll explore where some infrastructure defaults can play a significant role in managing your Viya platform's deployment.



Focus on Viya's Requirements

Cloud Provider Networking Examples

Infrastructure providers often employ **specialized configurations** to integrate with their **unique network infrastructure requirements**.

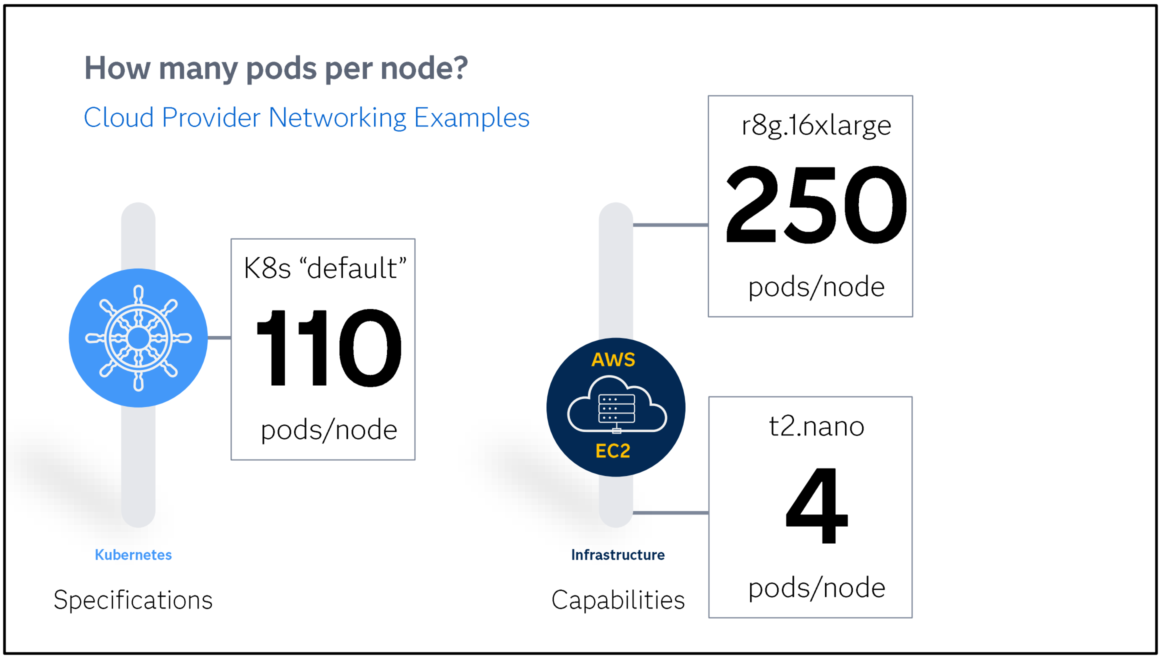
These specializations can lead to differences from the default expectations that are implied by the Kubernetes specifications.

We'll show examples here but remind you to continue referring to the **SAS documentation** for specifics as well as working with the cloud provider and customer IT teams to ensure requirements are met.

Infrastructure providers often employ specialized configurations to integrate with their unique network infrastructure requirements.

These specializations can lead to differences from the default expectations that are implied by the Kubernetes specifications.

We'll show examples here but remind you to continue referring to the SAS documentation for specifics as well as working with the cloud provider and customer IT teams to ensure requirements are met.



How many pods per node?

Cloud Provider Networking Examples

Referring to the Kubernetes documentation, a common takeaway is each Kubernetes node can support up to 110 pods. But that’s too simple and doesn’t take a lot of external influences into account.

While there are many factors that drive the effective number of pods on a node, a more realistic starting point is to look at the number of network interfaces that a machine instance provides. That kind of attribute is something that the infrastructure provider has already made a decision about in terms of how the hardware racks up in their environment.

For example, let’s consider the instance types offered by Amazon Web Services as part of their Elastic Cloud Computing service.

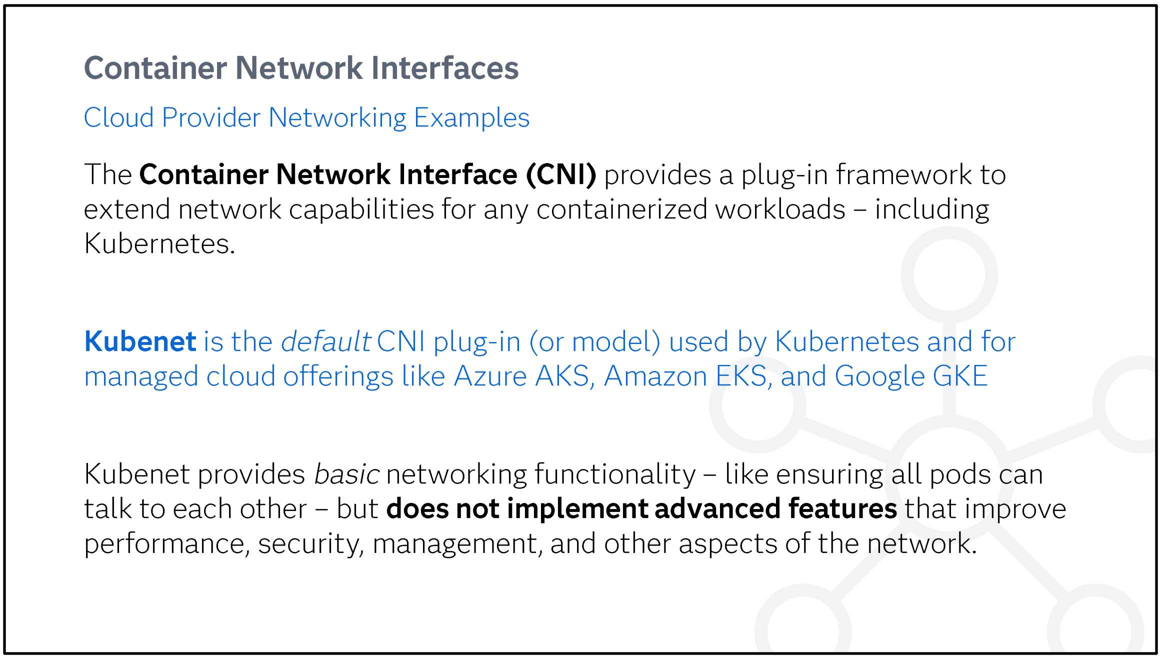
One of the smallest EC2 instance types offered by AWS is the tee two nano. It only has 1 vCPU and half a gig of RAM. But more importantly, that Amazon Machine Image definition only defines four elastic network interfaces for that instance by default. Assuming each ENI is assigned one IP address and realizing that each pod in Kubernetes gets its own IP address, then Amazon calculates the tee two nano as supporting only 4 pods when it’s a node in the EKS cluster.

At the other end of the spectrum is the much larger R eight G dot sixteen extra large. Its default definition allows it to theoretically support up to 737 elastic network interfaces, but for larger machines of this scale – that is, with more than thirty vCPU – AWS caps the recommendation as supporting up to 250 pods.

There are many different instance types ranging in size and specialty between these two – and a range in the number of pods supported along the way as well. For smaller machines with fewer than 30 vCPU, AWS will cap its recommended number of pods at 110, in alignment with the Kubernetes specification.

To be clear, we're not suggesting running over 250 pods – or even 700 pods! – on a Kubernetes node. That's unlikely to happen except for highly specialized workloads. The purpose of this slide is to call out some of the initial decision factors – like the number of network interfaces and associated IP addresses – that can affect workload placement of the Viya platform in your Kubernetes environment.

Partner up with the SAS World-Wide Sizing Team – either through SAS Technical Support or your site's SAS Account Representative – to get started with a proper sizing effort to determine instance types, the number of nodes, initial tuning requirements, workload placement, and more.



Container Network Interfaces

Cloud Provider Networking Examples

The **Container Network Interface (CNI)** provides a plug-in framework to extend network capabilities for any containerized workloads – including Kubernetes.

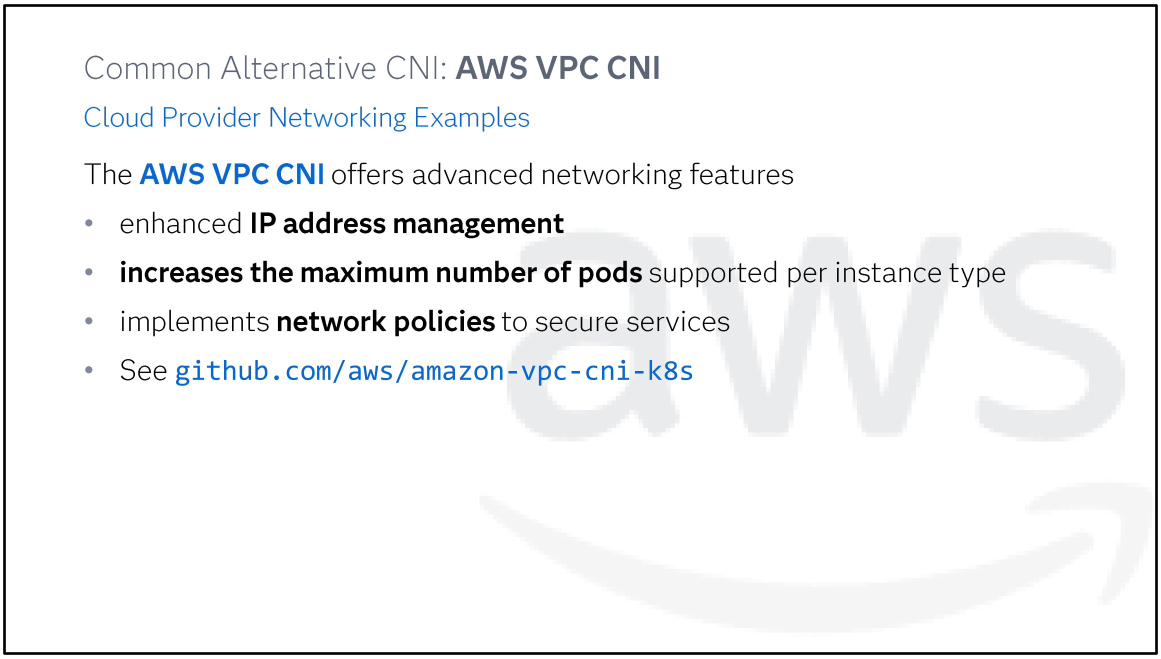
Kubenet is the *default* CNI plug-in (or model) used by Kubernetes and for managed cloud offerings like Azure AKS, Amazon EKS, and Google GKE

Kubenet provides *basic* networking functionality – like ensuring all pods can talk to each other – but **does not implement advanced features** that improve performance, security, management, and other aspects of the network.

While the implementation details around networking can vary from one infrastructure provider to another, they all support a plug-in framework to extend network capabilities and performance for Kubernetes called the Container Network Interface, or CNI.

Kubernetes provides its own default CNI model, or plug-in, known as Kubenet. It's also the default CNI for many cloud-managed Kubernetes offerings, including Azure Kubernetes Services, Amazon Elastic Kubernetes Services, and Google Kubernetes Engine.

Kubenet provides basic networking functionality – like ensuring all pods can talk to each other in a simple and systematic fashion. However, it doesn't implement advanced features that improve scalability and performance, security, administration, and other aspects of the network.



Common Alternative CNI: **AWS VPC CNI**

Cloud Provider Networking Examples

The **AWS VPC CNI** offers advanced networking features

- enhanced **IP address management**
- **increases the maximum number of pods** supported per instance type
- implements **network policies** to secure services
- See `github.com/aws/amazon-vpc-cni-k8s`

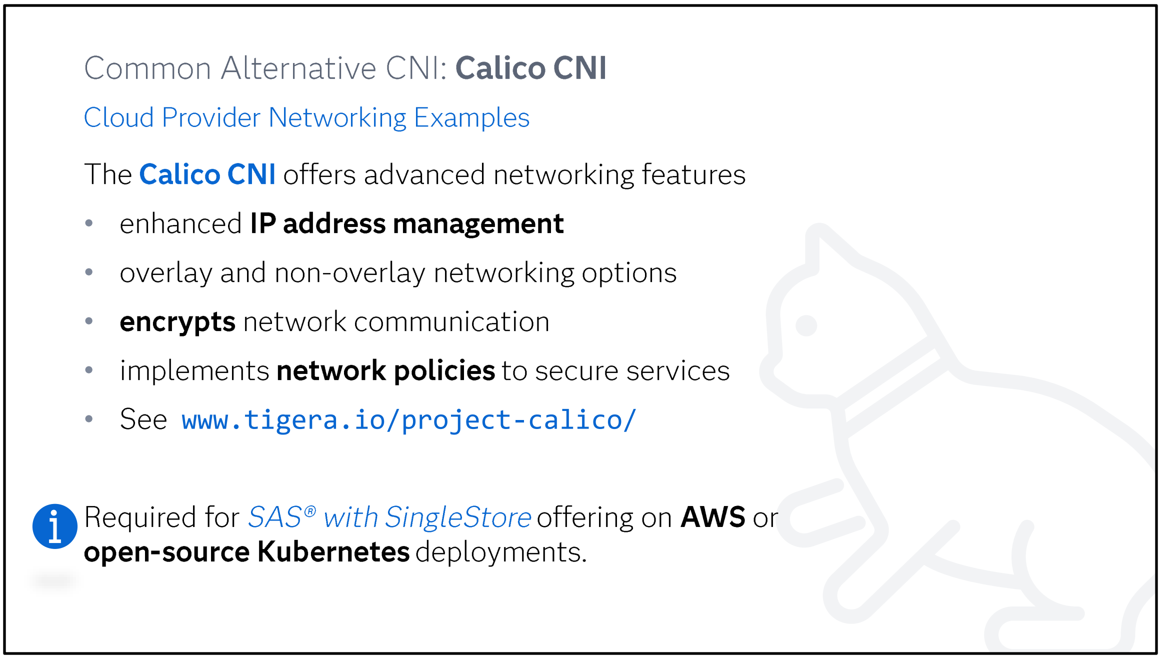
Since we just looked at an example where the number of pods is limited in AWS depending on the instance types of the EKS nodes, then let's look at some of the advanced networking features offered by the AWS VPC CNI first.

The AWS VPC CNI provides enhanced controls and management of IP address space for the pods in the cluster.

When implemented with prefix delegation, it can increase the number of IP addresses that each node can handle, effectively raising one of the lower limits on the number of pods that a node can host.

It also implements network policies which provide a form of network security. By default, Kubernetes allows all pods to talk to all other pods as well as accepting all traffic that comes in from outside the cluster. Network policies provide the Kubernetes administrator fine-grain control to manage those communication interactions – as opposed to relying exclusively on the infrastructure's firewall rules and security groups to implement only a partial solution.

For more information about features and configuration, check out the AWS VPC CNI open-source project hosted at github.com.



Common Alternative CNI: **Calico CNI**

Cloud Provider Networking Examples

The **Calico CNI** offers advanced networking features

- enhanced **IP address management**
- overlay and non-overlay networking options
- **encrypts** network communication
- implements **network policies** to secure services
- See `www.tigera.io/project-calico/`

ℹ Required for *SAS® with SingleStore* offering on **AWS** or **open-source Kubernetes** deployments.

Another CNI that we commonly see is Calico.

It also offers a number of enhanced controls and schemes for management of the Kubernetes IP address space.

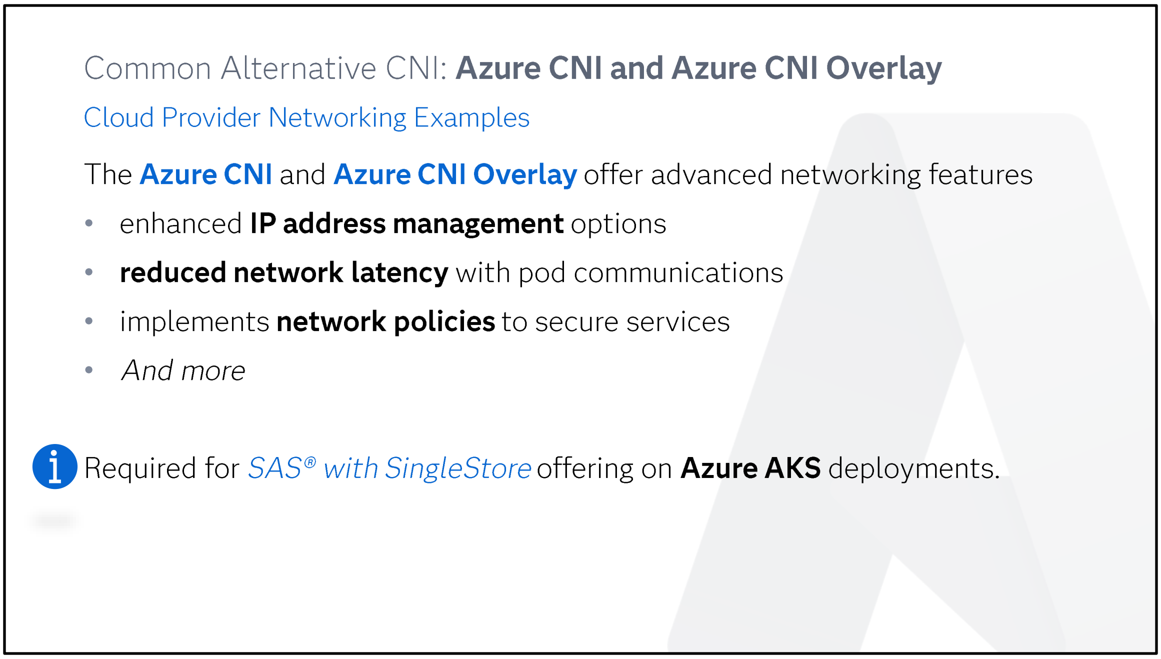
It has the option of either overlay or non-overlay implementation which is a choice on whether to use the same subnet IP address range as the nodes themselves. The overlay option reserves a separate private subnet for the pods similar to Kubenet.

Calico has built-in encryption for data in movement across the cluster.

And it also implements network policies to secure pod services in the cluster, too.

There’s a lot more to Calico so refer to its website and project repository on github.com for more information.

If deploying the SAS with SingleStore add-on to your Viya deployment in either AWS or using open-source Kubernetes, then you’ll likely need to setup the Calico CNI in your cluster as well.



Common Alternative CNI: **Azure CNI and Azure CNI Overlay**

Cloud Provider Networking Examples

The **Azure CNI** and **Azure CNI Overlay** offer advanced networking features

- enhanced **IP address management** options
- **reduced network latency** with pod communications
- implements **network policies** to secure services
- *And more*

ℹ Required for *SAS® with SingleStore* offering on **Azure AKS** deployments.

For Azure, there are two CNI we'll look at briefly – one is the Azure CNI and the other is Azure CNI Overlay.

Together, they offer a number of enhanced controls and schemes for management of the Kubernetes IP address space.

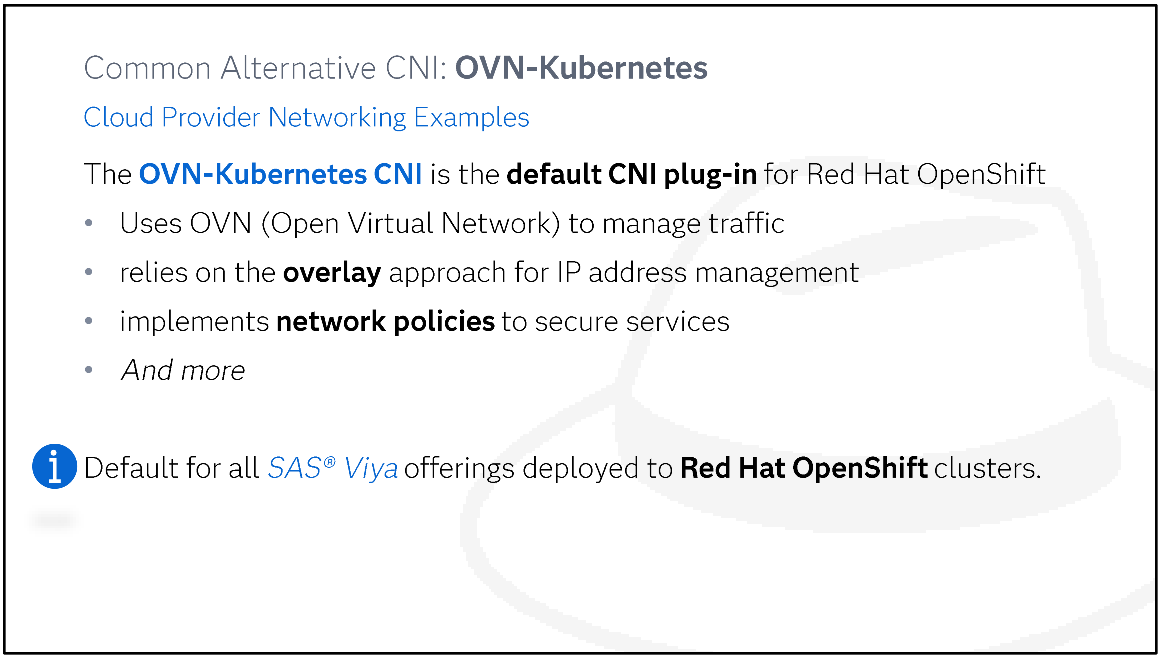
Selecting between overlay or non-overlay implementation is a choice on whether the pods will use the same subnet IP address range as the nodes themselves. Because the non-overlay option places the pods in the same IP address range as the nodes, then it will require a large, non-fragmented VNet IP address space to accommodate all the Viya pods.

One benefit of the non-overlay option is a flatter network scheme that realizes reduced latency with pod communications.

And it also implements network policies to secure pod services in the cluster.

The Azure CNI isn't as feature-rich as the Calico CNI, but there a lot more to it as well than mentioned briefly here, so refer to Microsoft's documentation for more information.

If deploying the SAS with SingleStore add-on to your Viya deployment in the Azure cloud, then you'll likely need to setup the Azure CNI or Azure CNI Overlay in your cluster as well.



Common Alternative CNI: **OVN-Kubernetes**

Cloud Provider Networking Examples

The **OVN-Kubernetes CNI** is the **default CNI plug-in** for Red Hat OpenShift

- Uses OVN (Open Virtual Network) to manage traffic
- relies on the **overlay** approach for IP address management
- implements **network policies** to secure services
- *And more*

ℹ Default for all *SAS® Viya* offerings deployed to **Red Hat OpenShift** clusters.

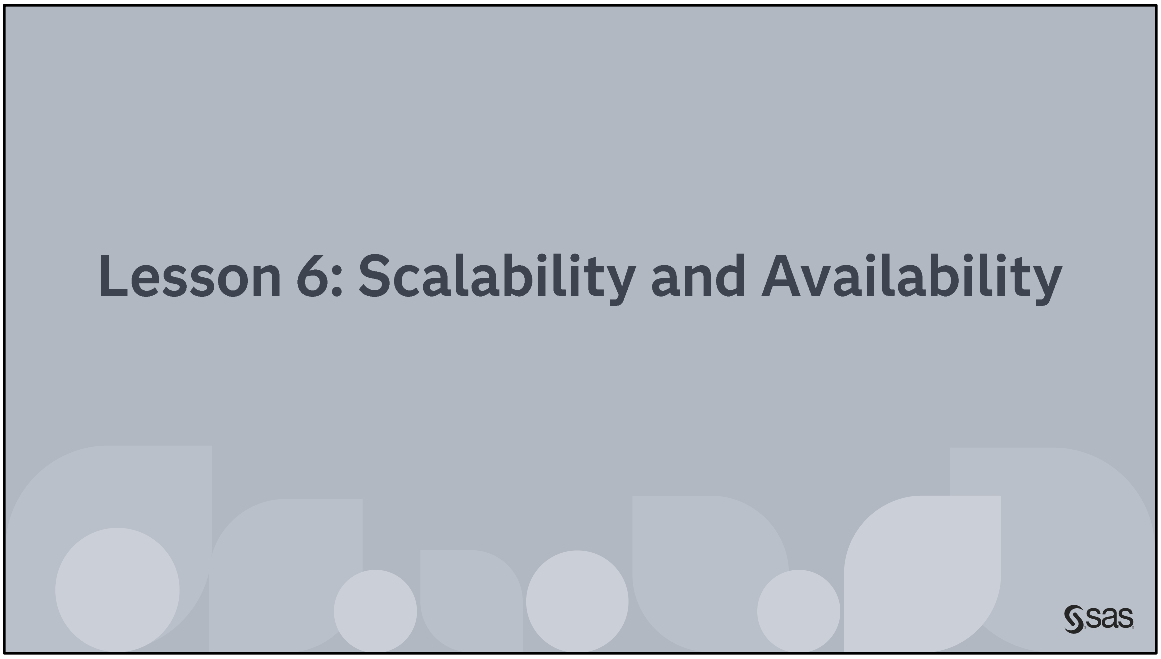
For Kubernetes based in Red Hat's OpenShift Open Container Platform, the default CNI plug-in is the OVN Kubernetes CNI. It's based on the Open Virtual Network project to provide the underlying management of traffic flows.

Like kubenet and some of the other CNI, it implements an overlay approach to IP address management for pods in the cluster.

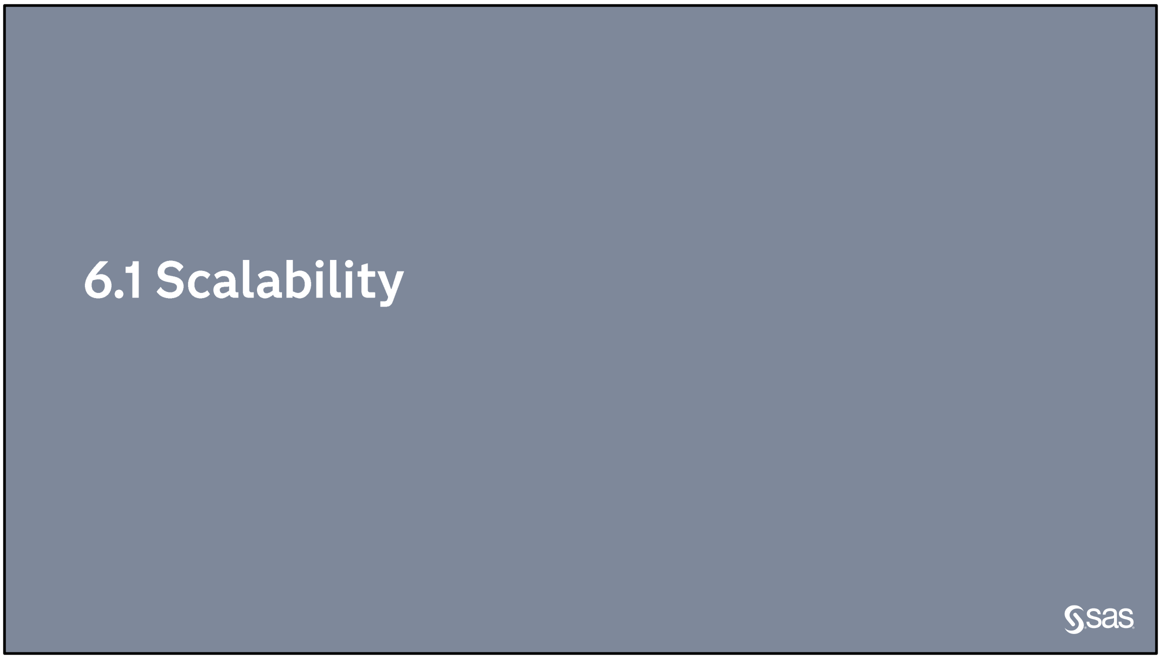
It also supports the use of network policies to secure pod services in the cluster.

For SAS Viya platform deployments to Red Hat OpenShift Container Platform, expect to see the OVN-Kubernetes CNI handling the network configuration.

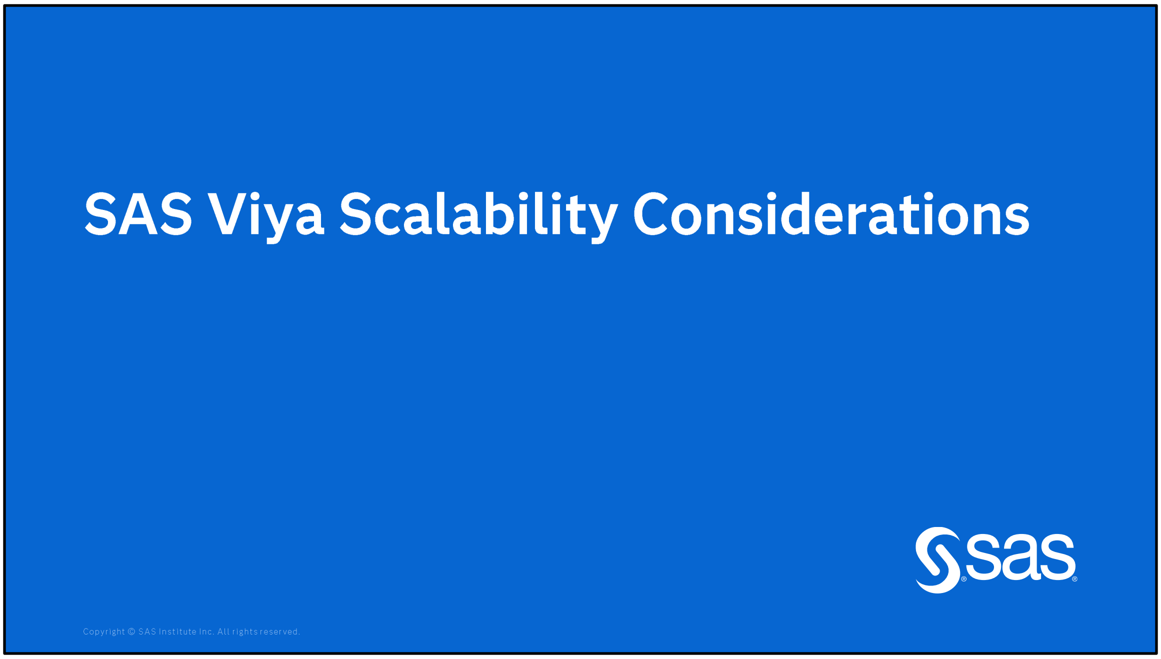
Expect to find the OVN-Kubernetes plug-in as the default CNI in use for Red Hat OpenShift clusters hosting SAS Viya.



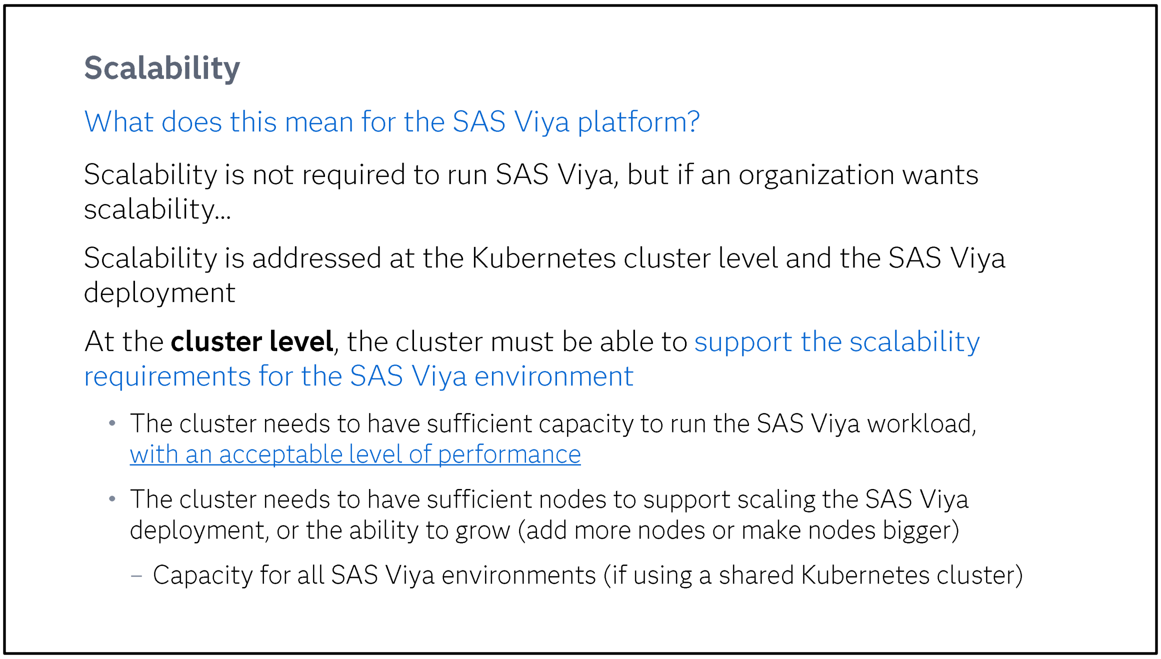
Lesson 6: Scalability and Availability



6.1 Scalability



SAS Viya Scalability Considerations



Scalability

What does this mean for the SAS Viya platform?

Scalability is not required to run SAS Viya, but if an organization wants scalability…

Scalability is addressed at the Kubernetes cluster level and the SAS Viya deployment

At the **cluster level**, the cluster must be able to support the scalability requirements for the SAS Viya environment

- The cluster needs to have sufficient capacity to run the SAS Viya workload, with an acceptable level of performance
- The cluster needs to have sufficient nodes to support scaling the SAS Viya deployment, or the ability to grow (add more nodes or make nodes bigger)
  - Capacity for all SAS Viya environments (if using a shared Kubernetes cluster)

In this module we will look at SAS Viya scalability considerations. What does scalability mean for the SAS Viya platform?

Scalability is not required to run SAS Viya, but if an organization wants scalability, scalability is addressed at two levels.

Scalability is addressed at the Kubernetes cluster level and to meet the requirements for the SAS Viya platform.

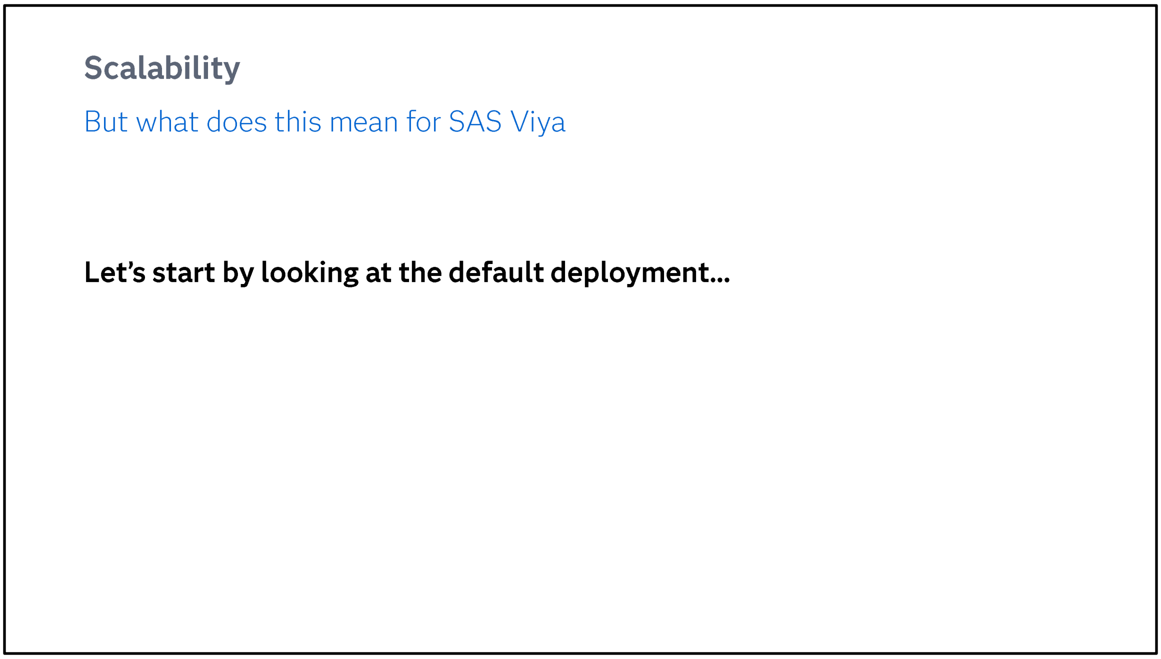
At the cluster level, the cluster must be able to support the scalability requirements for the SAS Viya environment.

The cluster needs to have sufficient capacity to run the SAS Viya workload with an acceptable level of performance.

The cluster needs to have sufficient nodes to support scaling the SAS Viya deployment, or the ability to grow, to add more nodes, or perhaps make the nodes larger.

If you are using a Kubernetes cluster for multiple SAS Viya deployments, the cluster needs to have capacity for all the environments.

Capacity planning is important, even when running on a Kubernetes platform.

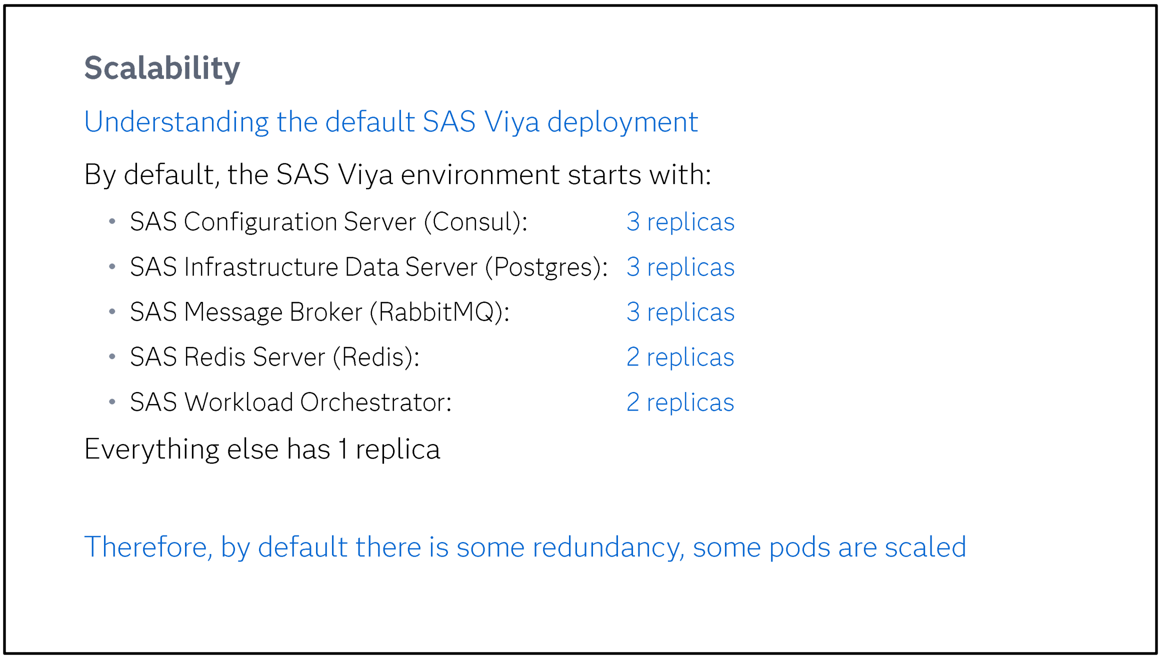


Scalability

But what does this mean for SAS Viya

Let's start by looking at the default deployment…

Let's start by looking at the default SAS Viya deployment.



Scalability

Understanding the default SAS Viya deployment

By default, the SAS Viya environment starts with:

SAS Configuration Server (Consul):	3 replicas
SAS Infrastructure Data Server (Postgres):	3 replicas
SAS Message Broker (RabbitMQ):	3 replicas
SAS Redis Server (Redis):	2 replicas
SAS Workload Orchestrator:	2 replicas

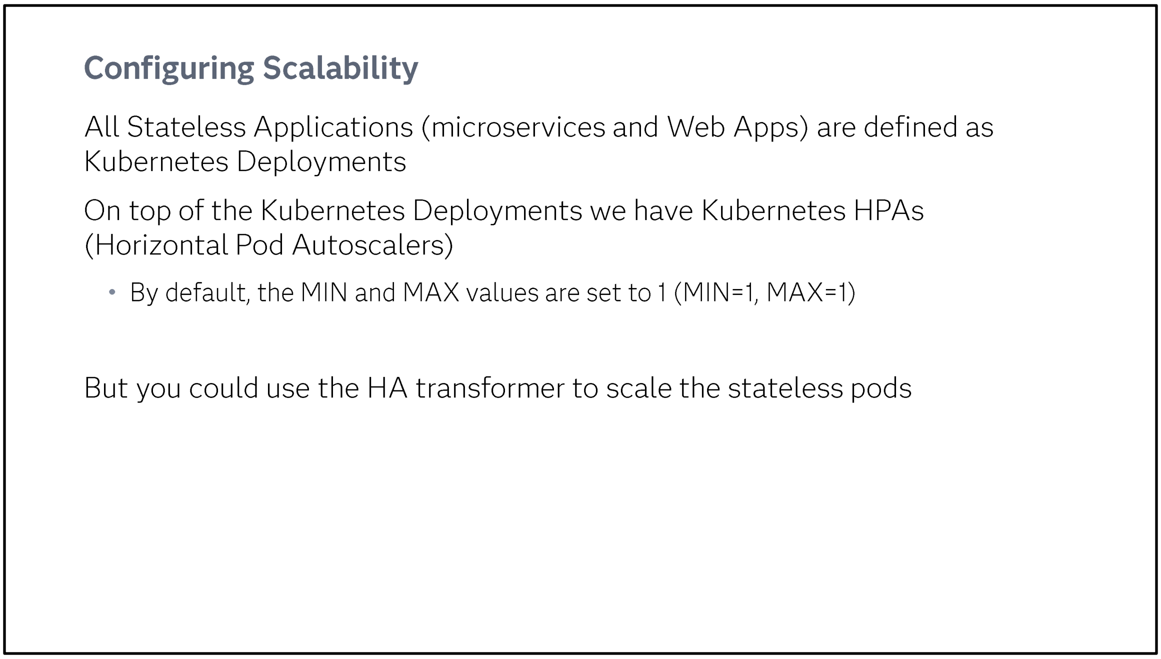
Everything else has 1 replica

Therefore, by default there is some redundancy, some pods are scaled

The SAS Viya platform starts with the following. There are three replicas of Consul, Postgres and Rabbit MQ. The SAS Redis Server has two replicas.

Finally, the SAS Workload Orchestrator has two replicas. Everything else is deployed with one pod replica.

Therefore, by default, there is some redundancy, some pods are scaled.



Configuring Scalability

All Stateless Applications (microservices and Web Apps) are defined as Kubernetes Deployments

On top of the Kubernetes Deployments we have Kubernetes HPAs (Horizontal Pod Autoscalers)

- By default, the MIN and MAX values are set to 1 (MIN=1, MAX=1)

But you could use the HA transformer to scale the stateless pods

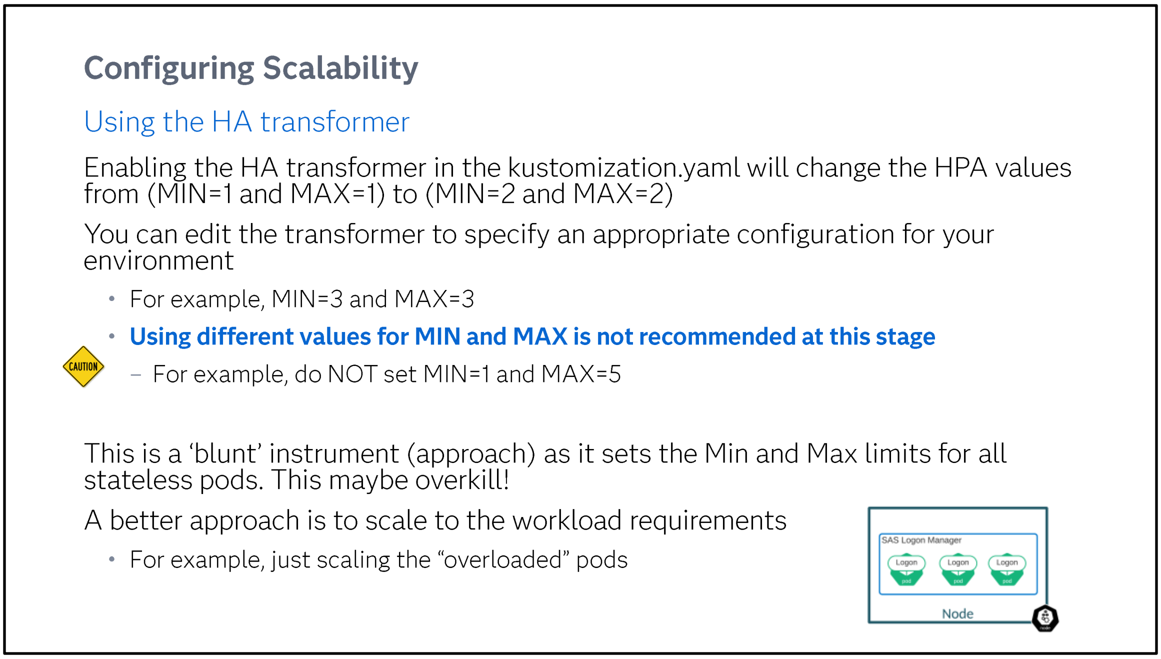
Looking at the default deployment in more detail.

All the stateless applications, the microservices and Web apps, are defined as Kubernetes Deployments.

On top of the Kubernetes Deployments we have Kubernetes Horizontal Pod Autoscalers, the HPA definitions.

By default, the MIN and MAX values are set to one.

But you could use the HA transformer to scale the stateless pods. We will look at this in more detail.



Configuring Scalability

Using the HA transformer

Enabling the HA transformer in the kustomization.yaml will change the HPA values from (MIN=1 and MAX=1) to (MIN=2 and MAX=2)

You can edit the transformer to specify an appropriate configuration for your environment

- For example, MIN=3 and MAX=3
- **Using different values for MIN and MAX is not recommended at this stage**
  - For example, do NOT set MIN=1 and MAX=5

This is a 'blunt' instrument (approach) as it sets the Min and Max limits for all stateless pods. This maybe overkill!

A better approach is to scale to the workload requirements

- For example, just scaling the "overloaded" pods

Using the HA transformer.

Enabling the HA transformer in the kustomization.yaml will change the HPA values from a MIN and MAX value of one. To a MIN and MAX value of two.

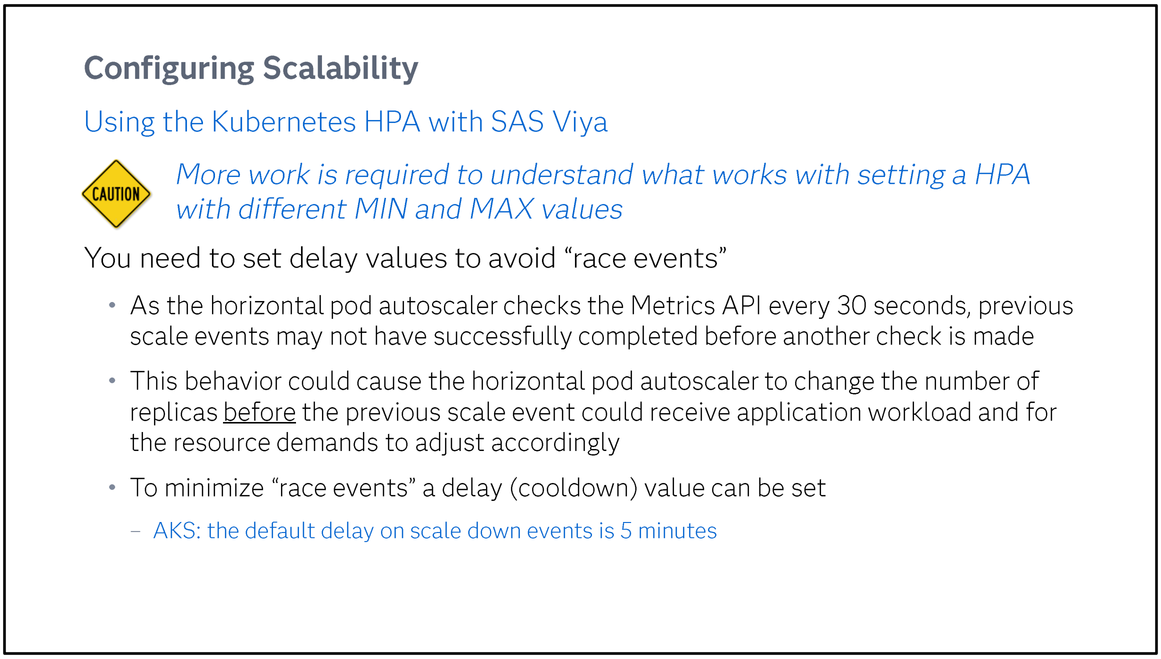
You can edit the transformer to specify an appropriate configuration for your environment. For example, MIN and MAX set to three.

A word of caution. Using different values for MIN and MAX is not recommended at this stage. For example, do NOT set MIN equal to one and MAX equal to five.

However, this is a 'blunt' instrument, it could be viewed as a blunt approach, as it sets the Min and Max limits for all stateless pods, and this maybe overkill!

A better approach is to scale to the workload requirements.

It is better to just focus on the overloaded pods. For example, perhaps there is a large number of users, and to handle the peak loads the SAS Logon Manager application needs three replicas.



Configuring Scalability

Using the Kubernetes HPA with SAS Viya

CAUTION *More work is required to understand what works with setting a HPA with different MIN and MAX values*

You need to set delay values to avoid “race events”

- As the horizontal pod autoscaler checks the Metrics API every 30 seconds, previous scale events may not have successfully completed before another check is made
- This behavior could cause the horizontal pod autoscaler to change the number of replicas before the previous scale event could receive application workload and for the resource demands to adjust accordingly
- To minimize “race events” a delay (cooldown) value can be set
  - AKS: the default delay on scale down events is 5 minutes

Now let’s look in more detail at using Kubernetes HPA’s with SAS Viya.

But first another word of caution. More work is required to understand what works with setting a HPA with different MIN and MAX values. For this, we need to understand the Kubernetes HPA’s.

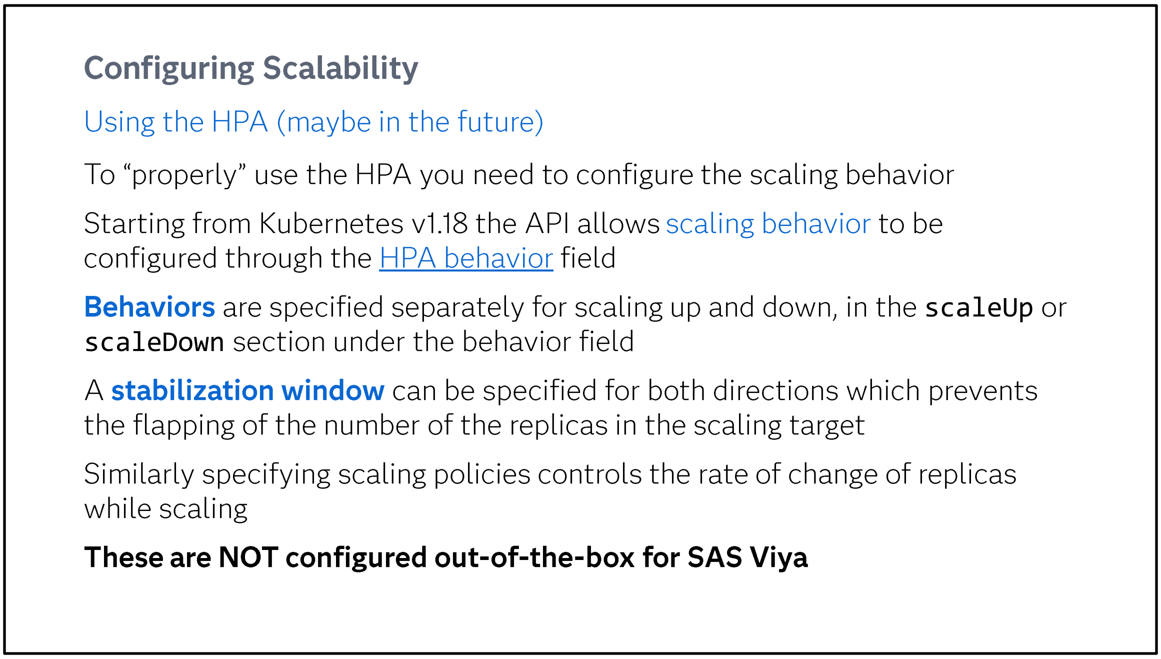
To effectively use the HPA’s you need to set delay values to avoid what is known as race events.

A race event is triggered when a previous scale event has not successfully completed before the next event is triggered. Let’s look at this in more detail.

The Kubernetes horizontal pod auto-scaler checks the metrics API every 30 seconds. This is used to determine if a scale event should be triggered, either up or down. But the previous scale events may not have successfully completed before another check is made.

This behavior could cause the horizontal pod auto-scaler to change the number of replicas before the previous scale event could receive application workload and for the resource demands to adjust accordingly.

To minimize the “race events” a delay, or cool down value can be set.



Configuring Scalability

Using the HPA (maybe in the future)

To “properly” use the HPA you need to configure the scaling behavior

Starting from Kubernetes v1.18 the API allows scaling behavior to be configured through the HPA behavior field

Behaviors are specified separately for scaling up and down, in the `scaleUp` or `scaleDown` section under the behavior field

A **stabilization window** can be specified for both directions which prevents the flapping of the number of the replicas in the scaling target

Similarly specifying scaling policies controls the rate of change of replicas while scaling

These are NOT configured out-of-the-box for SAS Viya

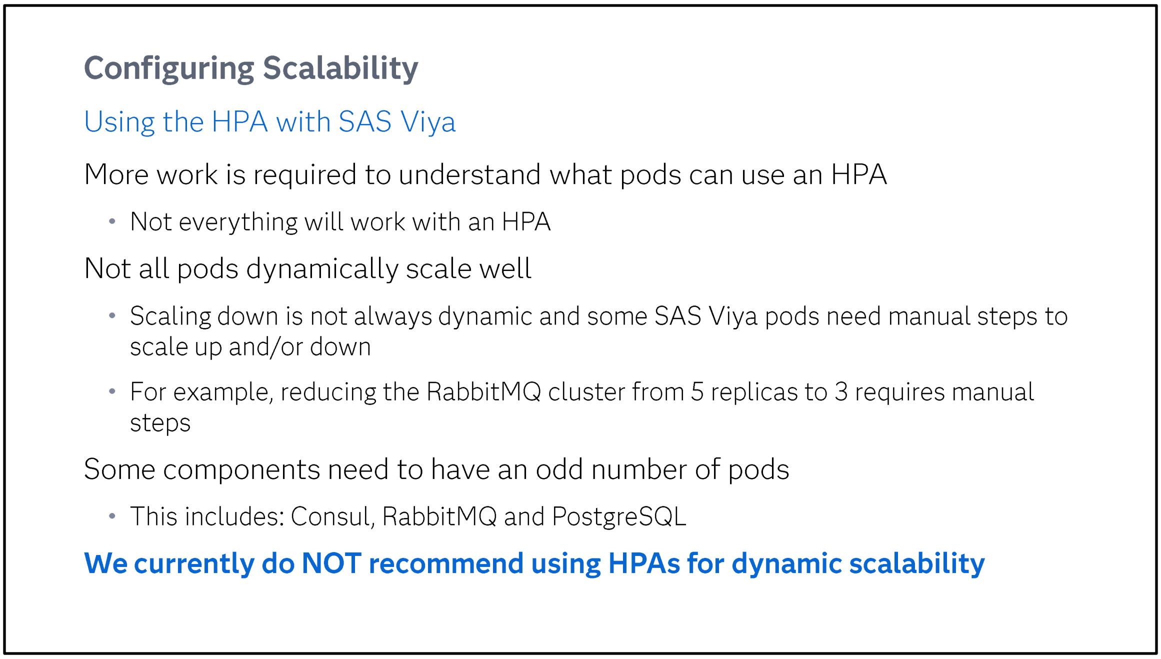
To properly use the HPA you need to configure the scaling behavior. Starting with Kubernetes version 1.18, the API allows scaling behavior to be configured through the HPA behavior field.

Behaviors are specified separately for scaling up and down. This is configured using the scale-up and scale-down sections under the HPA behavior field.

As part of this configuration, a stabilization window can be specified, for both directions, which prevents the flapping, or yo-yoing, of the number of the replicas in the scaling target. It specifies a period of time, a window, to wait for changes to stabilize.

Scaling policies can also be set to control the rate of change, of replicas while scaling. For example, should a pod be scaled one pod at a time, or by more than one pod.

It is important to note, the behavior fields are not configured out-of-the-box for SAS Viya. Hence, the need to specify the same values for MIN and MAX.



Configuring Scalability

Using the HPA with SAS Viya

More work is required to understand what pods can use an HPA

- Not everything will work with an HPA

Not all pods dynamically scale well

- Scaling down is not always dynamic and some SAS Viya pods need manual steps to scale up and/or down
- For example, reducing the RabbitMQ cluster from 5 replicas to 3 requires manual steps

Some components need to have an odd number of pods

- This includes: Consul, RabbitMQ and PostgreSQL

We currently do NOT recommend using HPAs for dynamic scalability

Using the HPA with SAS Viya.

More work is required to understand what pods can use an H-P-A, and how the behaviors should be configured. The HPA's are best suited for the stateless services. As not all components will work with an HPA.

Particularly when thinking about the stateful services, not all pods dynamically scale well. Scaling down is not always dynamic and some SAS Viya pods need manual steps to scale up, and/or down.

For example, reducing the Rabbit MQ cluster from 5 replicas to 3 requires manual steps.

Additionally, some components need to have an odd number of pods. This includes: Consul, RabbitMQ and Postgres. We will discuss this in more detail in other modules.

We currently do not recommend using HPA's for dynamic scalability.

Configuring Scalability

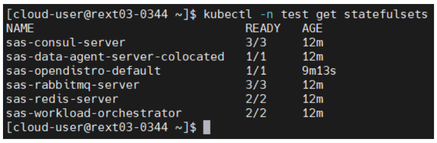
Scaling down to a minimum footprint

If HA is not a requirement, the SAS Viya platform can be configured to have 1 replica of the Stateful services

A PatchTransformer is provided under sas-bases/overlays

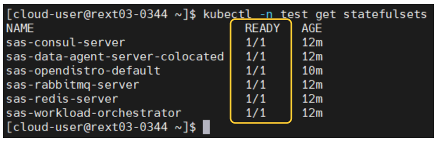
- `sas-bases/overlays/scaling/single-replica/transformer.yaml`

For example:



```
[cloud-user@rext03-0344 ~]$ kubectl -n test get statefulsets
NAME                              READY   AGE
sas-consul-server                 3/3     12m
sas-data-agent-server-colocated   1/1     12m
sas-opendistro-default            1/1     9m13s
sas-rabbitmq-server               3/3     12m
sas-redis-server                  2/2     12m
sas-workload-orchestrator         2/2     12m
[cloud-user@rext03-0344 ~]$
```

Applying the patch



```
[cloud-user@rext03-0344 ~]$ kubectl -n test get statefulsets
NAME                              READY   AGE
sas-consul-server                 1/1     12m
sas-data-agent-server-colocated   1/1     12m
sas-opendistro-default            1/1     10m
sas-rabbitmq-server               1/1     12m
sas-redis-server                  1/1     12m
sas-workload-orchestrator         1/1     12m
[cloud-user@rext03-0344 ~]$
```

Scaling down to a minimum footprint.

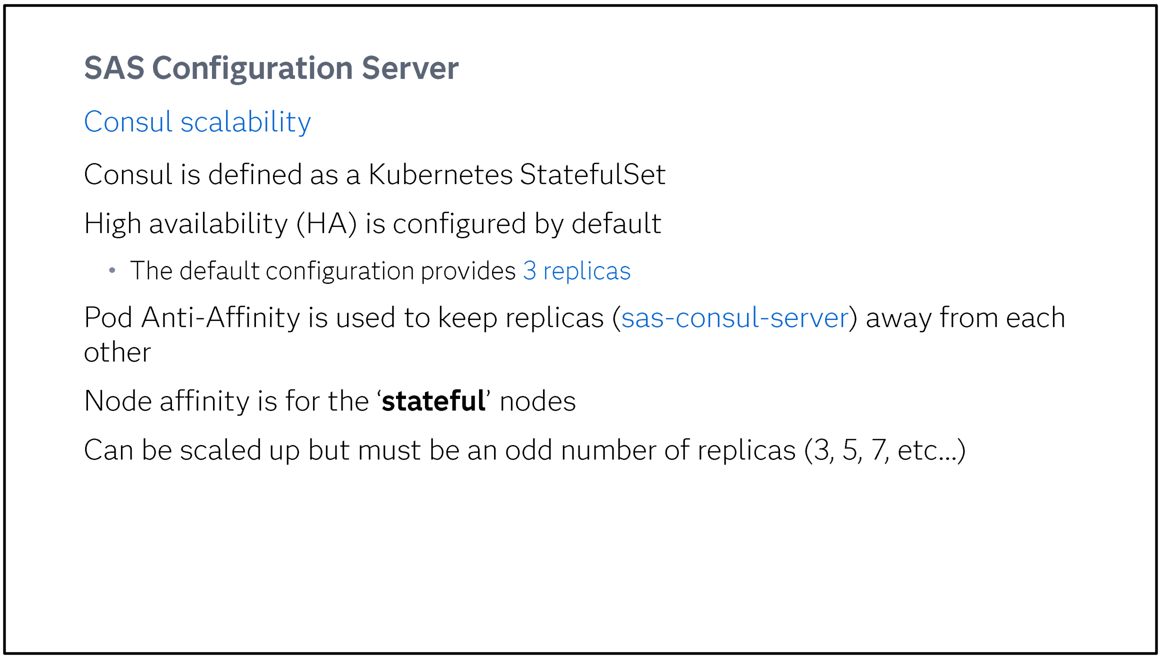
If HA is not a requirement, the SAS Viya platform can be configured to have 1 replica of the Stateful services.

A Patch Transformer is provided under the sas-bases overlays directory. This is located in the path shown here. The transformer needs to be added to the transformers block in your base kustomization.yaml file.

Here is an example of applying the patch transformer. The screenshot on the left is from a deployment without the patch transformer applied. And on the right, we can see the result of using the transformer.

On the right, you can now see that each of the StatefulSets now has one replica.

SAS Viya Platform Scalability



SAS Configuration Server

Consul scalability

Consul is defined as a Kubernetes StatefulSet

High availability (HA) is configured by default

- The default configuration provides 3 replicas

Pod Anti-Affinity is used to keep replicas (sas-consul-server) away from each other

Node affinity is for the ‘**stateful**’ nodes

Can be scaled up but must be an odd number of replicas (3, 5, 7, etc…)

In this module, we're going to look at SAS Viya platform scalability in more detail.

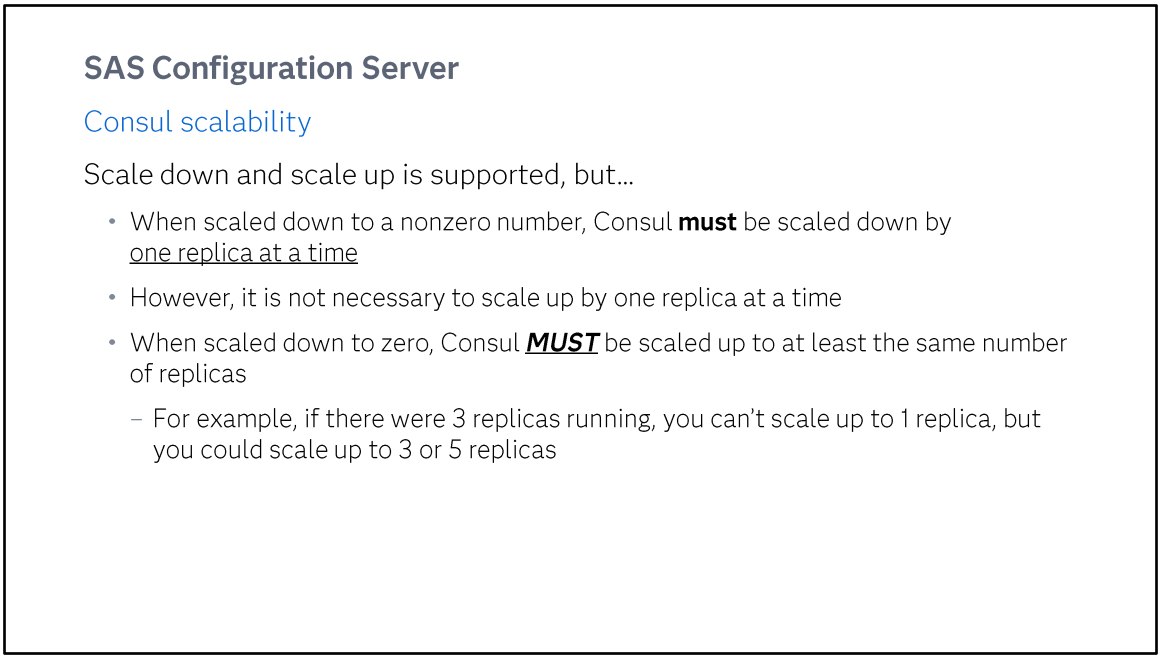
We'll start by looking at the SAS configuration server, console scalability. Console is defined as a Kubernetes StatefulSet.

High availability is configured by default. The default configuration provides three replicas of the console pods.

Pod anti-affinity is used to keep the replicas, the SAS console server pods, away from each other.

Node affinity is for the Stateful nodes. The nodes with the stateful workload class label.

Consul can be scaled up, but it must be to an odd number of replicas. Scaled to 3, 5, or 7 pods, etc.



SAS Configuration Server

Consul scalability

Scale down and scale up is supported, but…

- When scaled down to a nonzero number, Consul **must** be scaled down by one replica at a time
- However, it is not necessary to scale up by one replica at a time
- When scaled down to zero, Consul ***MUST*** be scaled up to at least the same number of replicas
  - For example, if there were 3 replicas running, you can't scale up to 1 replica, but you could scale up to 3 or 5 replicas

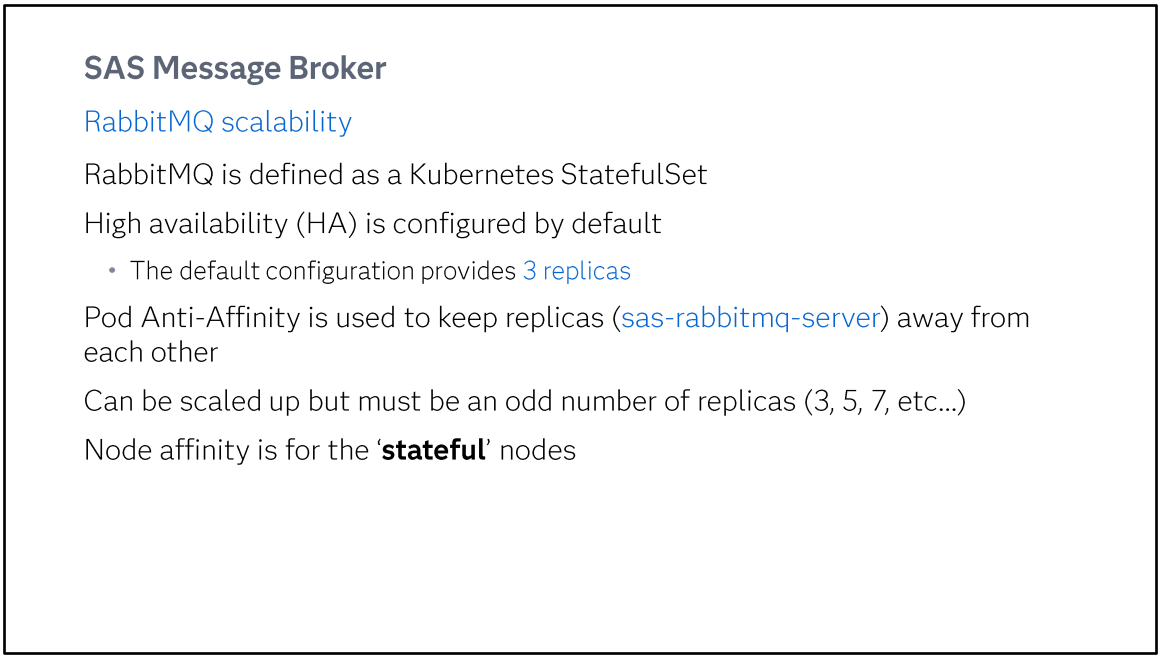
Scaling up and down is supported, but you need to understand some details.

When scaling down to a non-zero number of replicas, console must be scaled down by one replica at a time. For example, from three to two, to one.

However, it is not necessary to scale up by one replica at a time.

When scaling down to zero, console must be scaled up to at least the same number of replicas.

For example, if there were three replicas running, you can't scale up to one replica, but you could scale up to three, or five replicas.



SAS Message Broker

RabbitMQ scalability

RabbitMQ is defined as a Kubernetes StatefulSet

High availability (HA) is configured by default

- The default configuration provides 3 replicas

Pod Anti-Affinity is used to keep replicas (sas-rabbitmq-server) away from each other

Can be scaled up but must be an odd number of replicas (3, 5, 7, etc…)

Node affinity is for the '**stateful**' nodes

Scalability for the SAS Message Broker. RabbitMQ scalability.

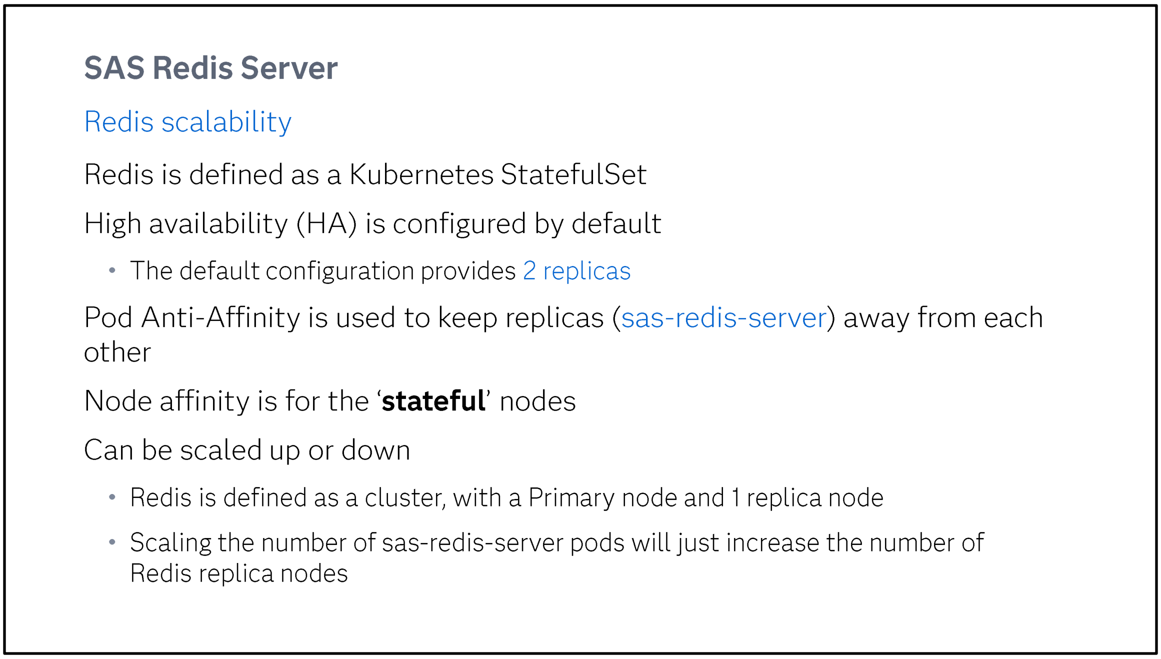
RabbitMQ is defined as a Kubernetes StatefulSet

High availability is configured by default. The default configuration provides three replicas of the RabbitMQ server pods.

Pod anti-affinity is used to keep the replicas, the SAS RabbitMQ server pods, away from each other.

The RabbitMQ server pods can be scaled up, but it must be to an odd number of replicas. Scaled to 3, 5, or 7 pods.

The node affinity is for the stateful nodes. Again, the nodes with the stateful workload class label.



SAS Redis Server

Redis scalability

Redis is defined as a Kubernetes StatefulSet

High availability (HA) is configured by default

- The default configuration provides 2 replicas

Pod Anti-Affinity is used to keep replicas (sas-redis-server) away from each other

Node affinity is for the '**stateful**' nodes

Can be scaled up or down

- Redis is defined as a cluster, with a Primary node and 1 replica node
- Scaling the number of sas-redis-server pods will just increase the number of Redis replica nodes

Scalability for the SAS Redis Server.

Redis is defined as a Kubernetes StatefulSet

High availability is configured by default. The default configuration provides two replicas of the Redis server pods.

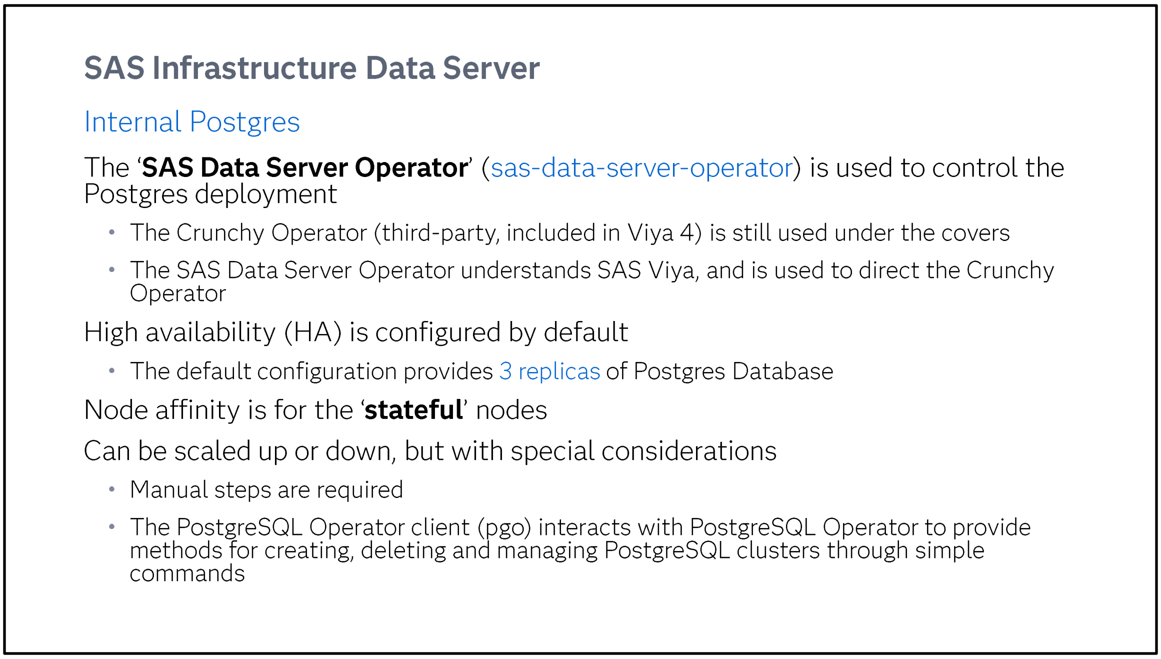
Pod anti-affinity is used to keep the replicas, the SAS Redis server pods, away from each other.

The node affinity is for the stateful nodes. Again, the nodes with the stateful workload class label.

Redis is defined as a single cluster, with a primary Redis node and 1 replica node.

From a scalability perspective, the default deployment can be scaled down to a single Redis Server pod, to a single Redis node. To scale up or out, is possible, but this will only increase the number of Redis replica nodes.

Therefore, rather than increasing the number of replicas, the number of Redis nodes. The most likely change for a busy system would be to adjust the pod memory. A patch transform is provided to modify the Redis memory, in the sas-bases examples.



SAS Infrastructure Data Server

Internal Postgres

The ‘**SAS Data Server Operator**’ (sas-data-server-operator) is used to control the Postgres deployment

- The Crunchy Operator (third-party, included in Viya 4) is still used under the covers
- The SAS Data Server Operator understands SAS Viya, and is used to direct the Crunchy Operator

High availability (HA) is configured by default

- The default configuration provides 3 replicas of Postgres Database

Node affinity is for the ‘**stateful**’ nodes

Can be scaled up or down, but with special considerations

- Manual steps are required
- The PostgreSQL Operator client (pgo) interacts with PostgreSQL Operator to provide methods for creating, deleting and managing PostgreSQL clusters through simple commands

We will now look at the SAS Infrastructure Data Server, when using the internal Postgres server.

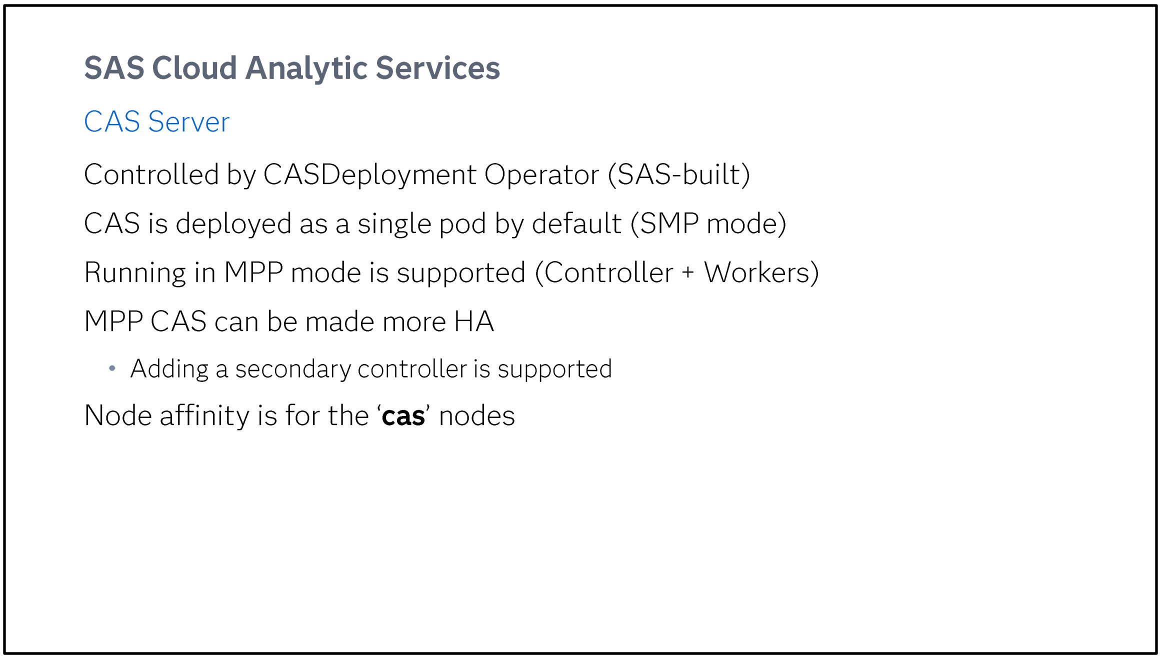
The SAS data server operator is used to control the Postgres deployment, and the name of that operator pod is shown in blue.

The crunchy operator, which is a third-party operator, which is included within the SAS Viya deployment, is still used under the covers. The SAS Data Server operator understands SAS Viya and is used to direct the crunchy operator.

High availability is configured by default. The default configuration provides 3 replicas of the Postgres database.

Node affinity is for the stateful nodes.

The configuration can be scaled up, or down, but there are some special considerations.
Manual steps are required
The Postgres Operator client, PGO, interacts with the Postgres Operator to provide methods for creating, deleting and managing the Postgres cluster.



SAS Cloud Analytic Services

CAS Server

Controlled by CASDeployment Operator (SAS-built)

CAS is deployed as a single pod by default (SMP mode)

Running in MPP mode is supported (Controller + Workers)

MPP CAS can be made more HA

- Adding a secondary controller is supported

Node affinity is for the '**cas**' nodes

We will now look at the CAS Server.

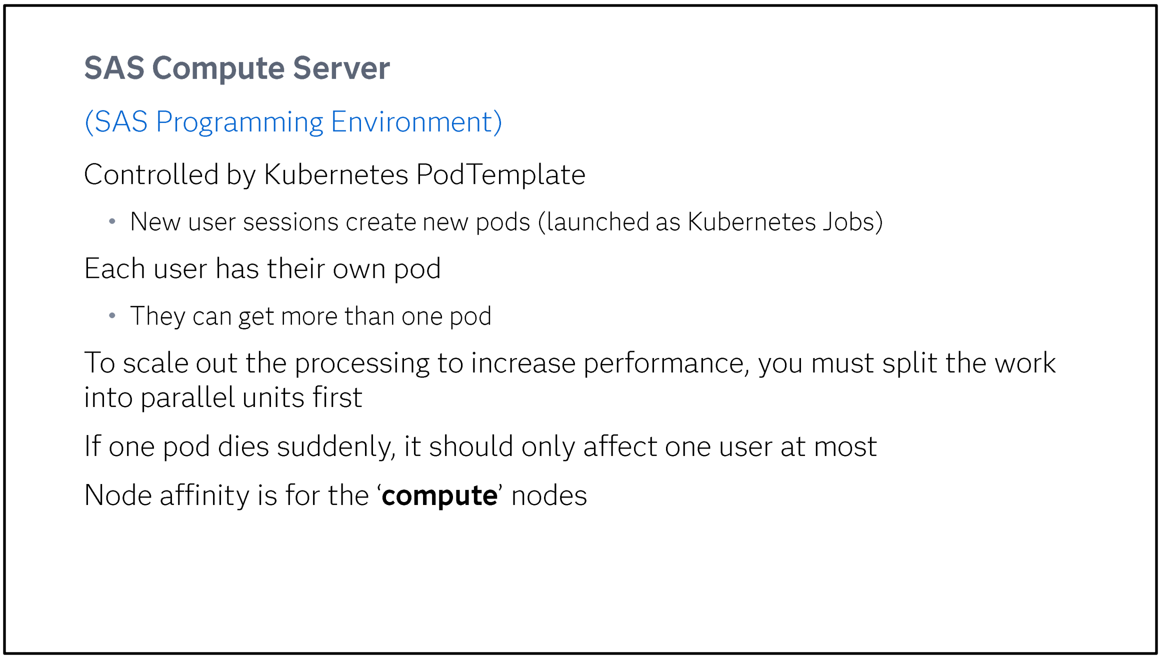
The CAS server is controlled by the CAS Deployment operator, which is a SAS built Kubernetes operator.

CAS is deployed as a single pod by default, running in SMP mode.

The CAS Server can be configured to run in MPP mode. This scales the CAS Server horizontally to have a Controller, and one or more Workers.

The MPP CAS Server can be configured for high availability. A secondary, or backup, controller can be configured. The use of multiple workers provides scalability, but also helps to improve availability of the CAS Server.

Node affinity is for the CAS nodes. Nodes with the CAS workload class label.



SAS Compute Server

(SAS Programming Environment)

Controlled by Kubernetes PodTemplate

- New user sessions create new pods (launched as Kubernetes Jobs)

Each user has their own pod

- They can get more than one pod

To scale out the processing to increase performance, you must split the work into parallel units first

If one pod dies suddenly, it should only affect one user at most

Node affinity is for the '**compute**' nodes

Finally, we will discuss the SAS programming environment.

We will start by looking at the SAS Compute Server.

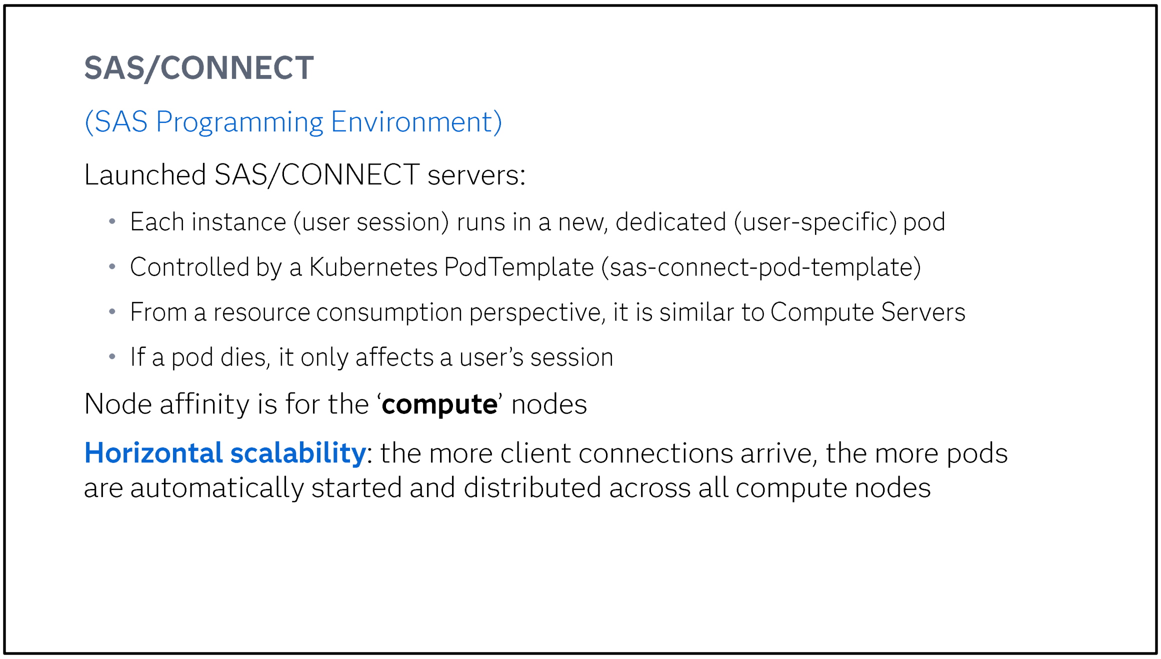
The SAS compute server, the SAS programming environment, is controlled by a Kubernetes pod template. New user sessions create new pods, which are launched as Kubernetes jobs.

Each user has their own compute server pod, and they can have more than one pod. For example, if the user had multiple SAS Studio sessions running. Each session has its own compute server pod.

To scale out the processing, to increase performance, you must split the work, the code, into parallel units.

If one pod dies, it should only affect one user at most. The user for that compute server session.

Node affinity is for the compute nodes. For nodes with the Compute workload class label.



SAS/CONNECT

(SAS Programming Environment)

Launched SAS/CONNECT servers:

- Each instance (user session) runs in a new, dedicated (user-specific) pod
- Controlled by a Kubernetes PodTemplate (sas-connect-pod-template)
- From a resource consumption perspective, it is similar to Compute Servers
- If a pod dies, it only affects a user's session

Node affinity is for the '**compute**' nodes

Horizontal scalability: the more client connections arrive, the more pods are automatically started and distributed across all compute nodes

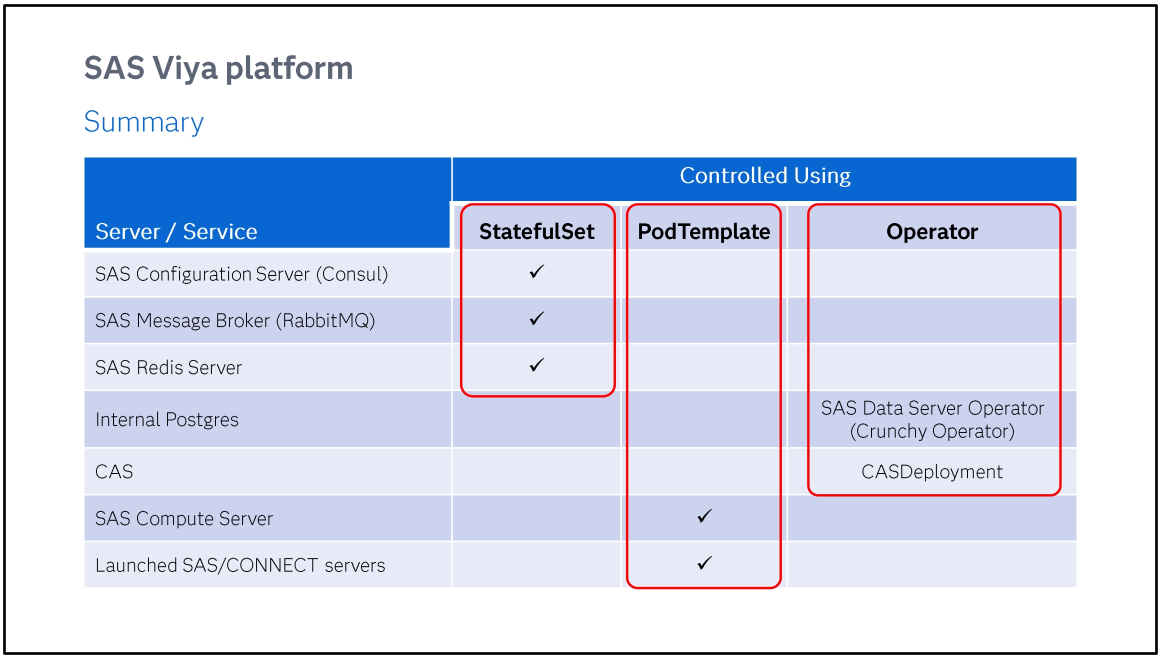
SAS Connect is part of the SAS programming environment.

Here, we will discuss the launched SAS Connect servers.

Each instance, or user session, runs in a dedicated user-specific pod.
The launching of the pods is controlled through the use of a Kubernetes pod template.
From a resource consumption perspective, it is very similar to the compute servers.
Thinking about availability, if one pod dies, it only affects that one user session.

Node affinity is for the compute nodes.

Horizontal scalability is achieved through each session running in its own pod. As the client connections arrive, the new pods are automatically started and distributed across all compute nodes.



SAS Viya platform

Summary

	Controlled Using		
Server / Service	StatefulSet	PodTemplate	Operator
SAS Configuration Server (Consul)	✓		
SAS Message Broker (RabbitMQ)	✓		
SAS Redis Server	✓		
Internal Postgres			SAS Data Server Operator (Crunchy Operator)
CAS			CASDeployment
SAS Compute Server		✓	
Launched SAS/CONNECT servers		✓	

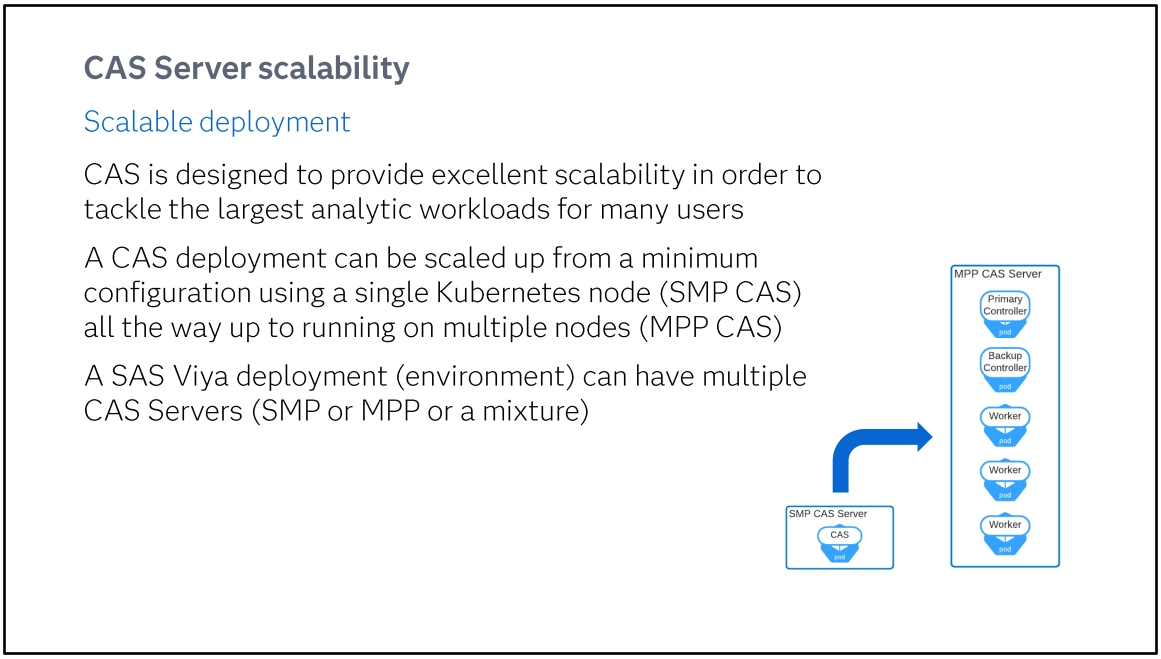
To summarize the SAS Viya platform configuration.

Stateful-sets are used to control the SAS Configuration Server, the SAS Message Broker, and the SAS Redis Server.

As you can see from the table, a Kubernetes pod template is used for the SAS Compute Server, and launched SAS connect servers.

Finally, we can see the components that are controlled using a Kubernetes operator.

CAS



CAS Server scalability

Scalable deployment

CAS is designed to provide excellent scalability in order to tackle the largest analytic workloads for many users

A CAS deployment can be scaled up from a minimum configuration using a single Kubernetes node (SMP CAS) all the way up to running on multiple nodes (MPP CAS)

A SAS Viya deployment (environment) can have multiple CAS Servers (SMP or MPP or a mixture)

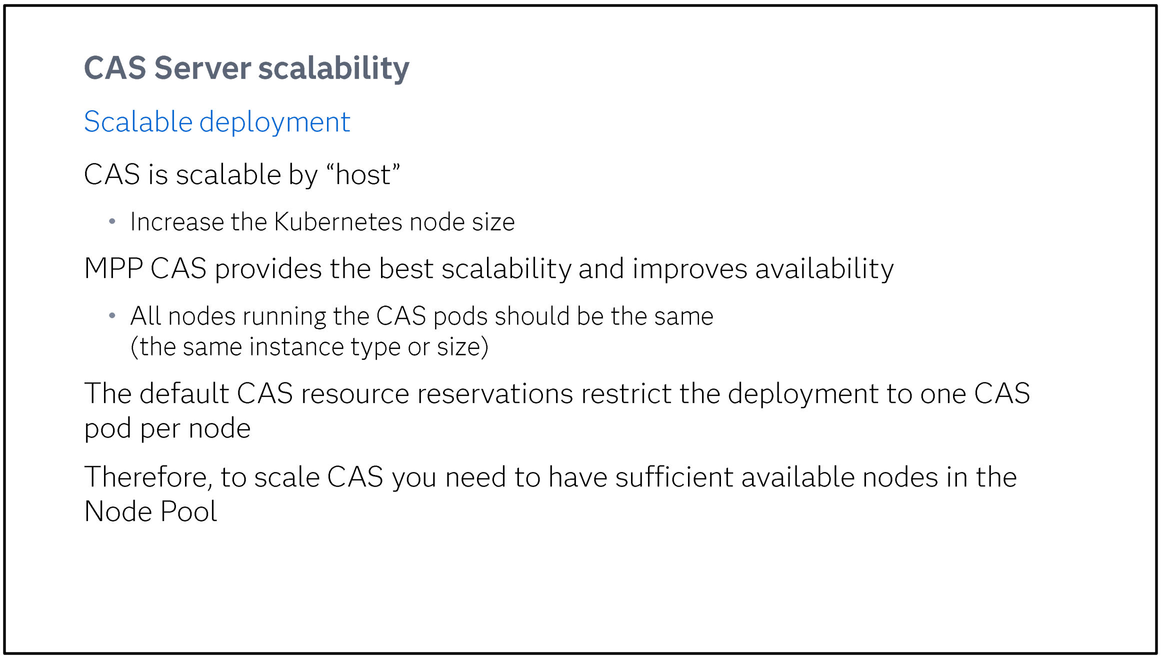
This module provides more details on scaling the SAS Cloud Analytic Services server, the CAS Server.

CAS is designed to provide excellent scalability in order to tackle the largest analytic workloads for many users.

A CAS deployment can be scaled up from a minimum configuration, using a single Kubernetes node, a single pod. This is known as SMP mode, an SMP CAS server. Through to running on multiple nodes. This is MPP mode, or an MPP CAS server.

As shown in the illustration on the right. The SMP CAS server can be scaled to a MPP CAS server. In this example, the MPP CAS Server has a primary controller, backup controller, and three workers.

A SAS Viya deployment, a Viya platform, can have multiple CAS Servers. SMP, MPP, or a mixture of both.



CAS Server scalability

Scalable deployment

CAS is scalable by “host”

- Increase the Kubernetes node size

MPP CAS provides the best scalability and improves availability

- All nodes running the CAS pods should be the same (the same instance type or size)

The default CAS resource reservations restrict the deployment to one CAS pod per node

Therefore, to scale CAS you need to have sufficient available nodes in the Node Pool

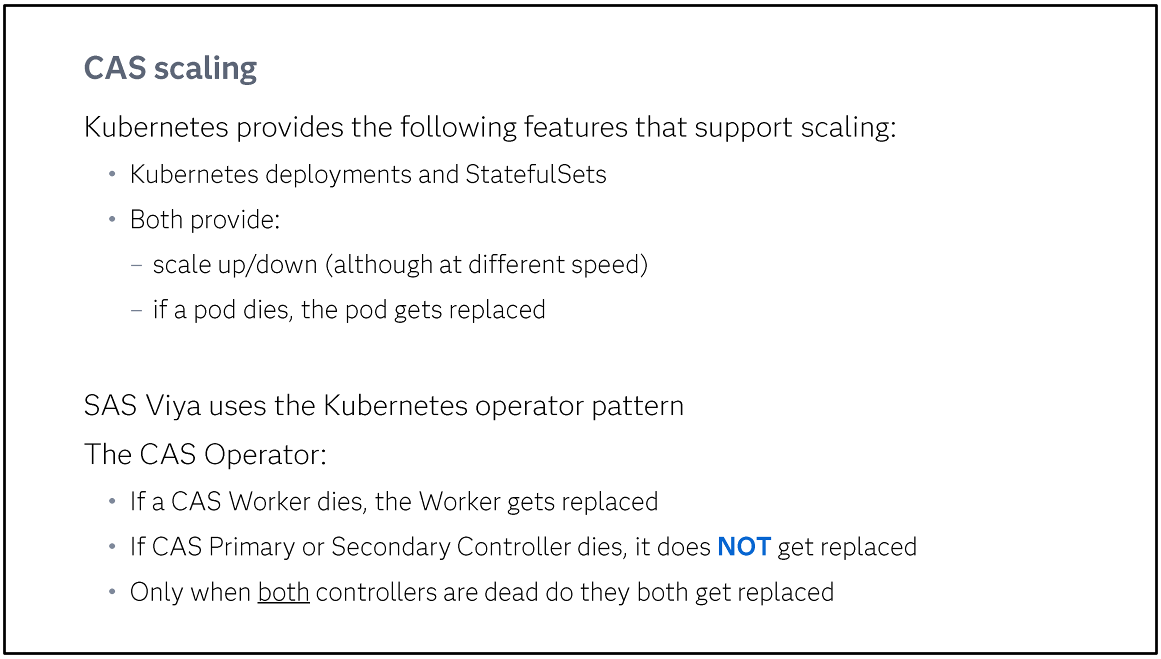
Understanding CAS Server scalability.

CAS is scalable by host. This is done in terms of increasing the Kubernetes node size that is being used for the CAS pods.

The CAS server can also be scaled horizontally, this is done using an MPP CAS server. The MPP CAS server provides the best scalability and improves availability. To optimize the CAS Server performance, for maximum efficiency, all nodes running the CAS pods should be the same. The same instance type, or size. The CAS architecture assumes that all the nodes are identical.

The default CAS resource reservations restrict the deployment to one CAS pod per node.

Therefore, to scale CAS, you need to have sufficient nodes available in the CAS node pool. The node pool needs to support the scalability requirements.



CAS scaling

Kubernetes provides the following features that support scaling:

- Kubernetes deployments and StatefulSets
- Both provide:
  - scale up/down (although at different speed)
  - if a pod dies, the pod gets replaced

SAS Viya uses the Kubernetes operator pattern

The CAS Operator:

- If a CAS Worker dies, the Worker gets replaced
- If CAS Primary or Secondary Controller dies, it does **NOT** get replaced
- Only when both controllers are dead do they both get replaced

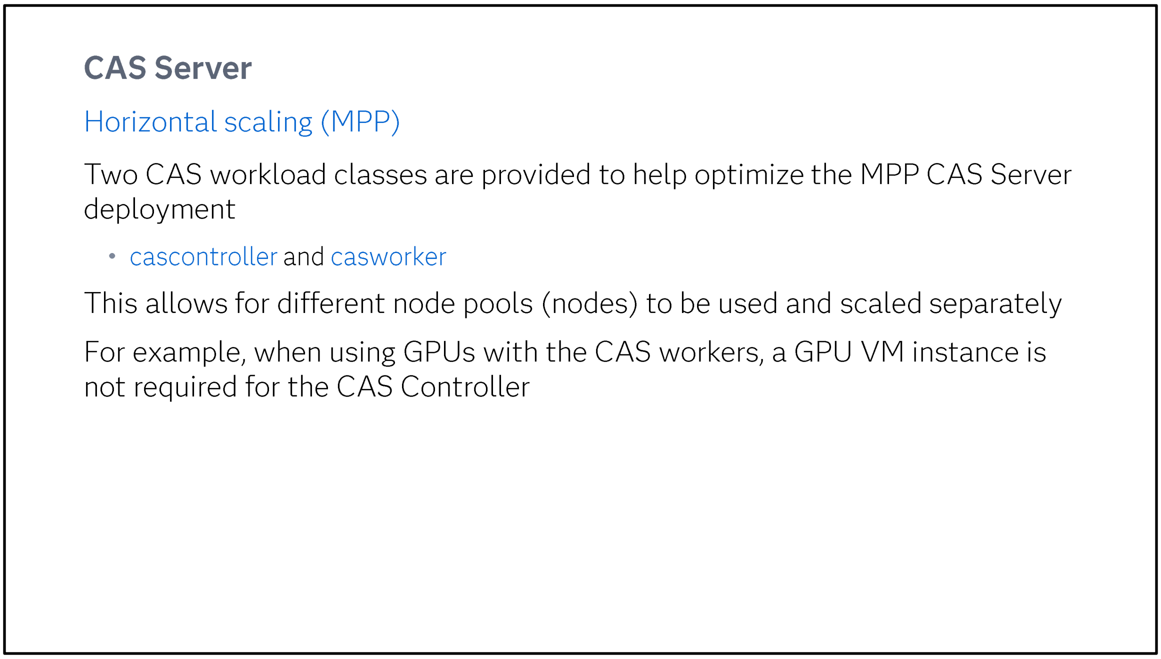
Let’s look at CAS scaling in more detail.

Kubernetes provides the following features that support scaling:
Kubernetes deployments,
and Stateful-Sets.

Both provide scaling up and down, although at different rates. If a pod dies, the pod gets replaced.

SAS Viya uses the Kubernetes operator pattern, to control the CAS server. The CAS Deployment operator is a SAS built operator.

The CAS Operator controls the CAS deployment. With the following behavior:
If a CAS Worker dies, the Worker gets replaced.
If the CAS Primary, or Secondary, Controller dies, it does NOT get immediately replaced.
It is only when both controllers are dead, that they are replaced. A new instance of the controllers is started at that point.



CAS Server

Horizontal scaling (MPP)

Two CAS workload classes are provided to help optimize the MPP CAS Server deployment

- cascontroller and casworker

This allows for different node pools (nodes) to be used and scaled separately

For example, when using GPUs with the CAS workers, a GPU VM instance is not required for the CAS Controller

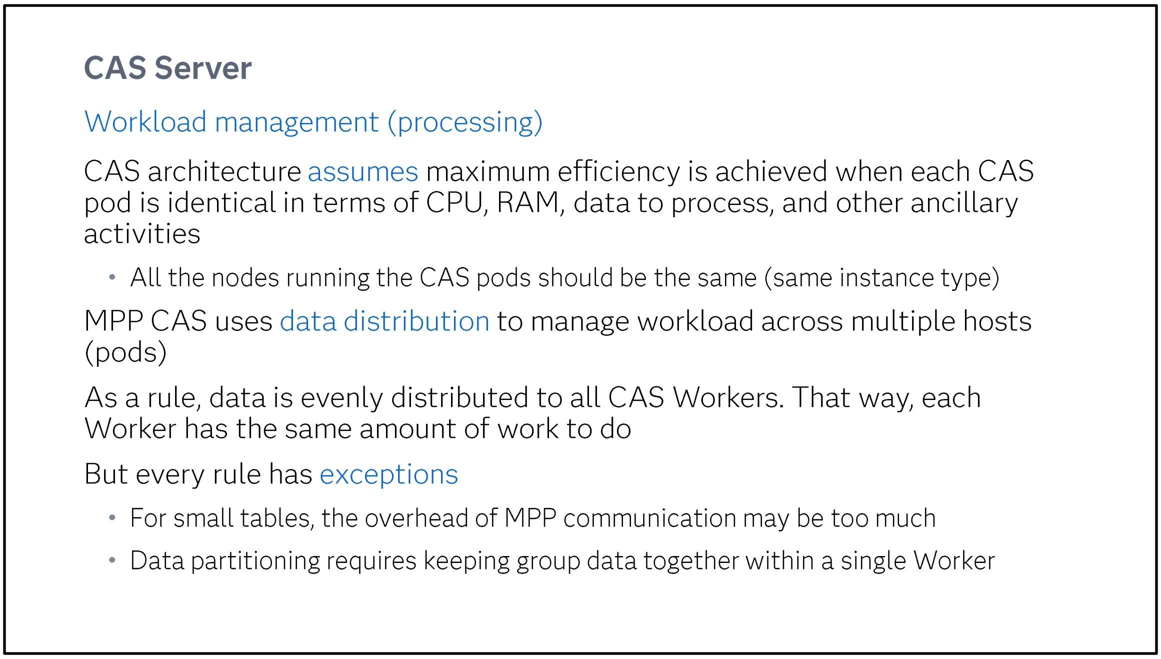
Horizontal scaling.

In addition to the default CAS workload class. Two CAS workload classes are provided to help optimize the MPP CAS Server deployment.

They are the CAS-Controller and CAS-Worker workload classes.

This allows for different node pools, different node types, to be used for the Controller and Workers. And allows for the Workers to be scaled separately from the Controller, or Controllers.

Another advantage to using the two node pools is the ability to use specialize instance types for the CAS Workers. For example, when using GPUs with the CAS workers, a GPU VM instance is not required for the CAS Controller. This helps to optimize the CAS deployment.



CAS Server

Workload management (processing)

CAS architecture assumes maximum efficiency is achieved when each CAS pod is identical in terms of CPU, RAM, data to process, and other ancillary activities

- All the nodes running the CAS pods should be the same (same instance type)

MPP CAS uses data distribution to manage workload across multiple hosts (pods)

As a rule, data is evenly distributed to all CAS Workers. That way, each Worker has the same amount of work to do

But every rule has exceptions

- For small tables, the overhead of MPP communication may be too much
- Data partitioning requires keeping group data together within a single Worker

Understanding “cas workload management”.

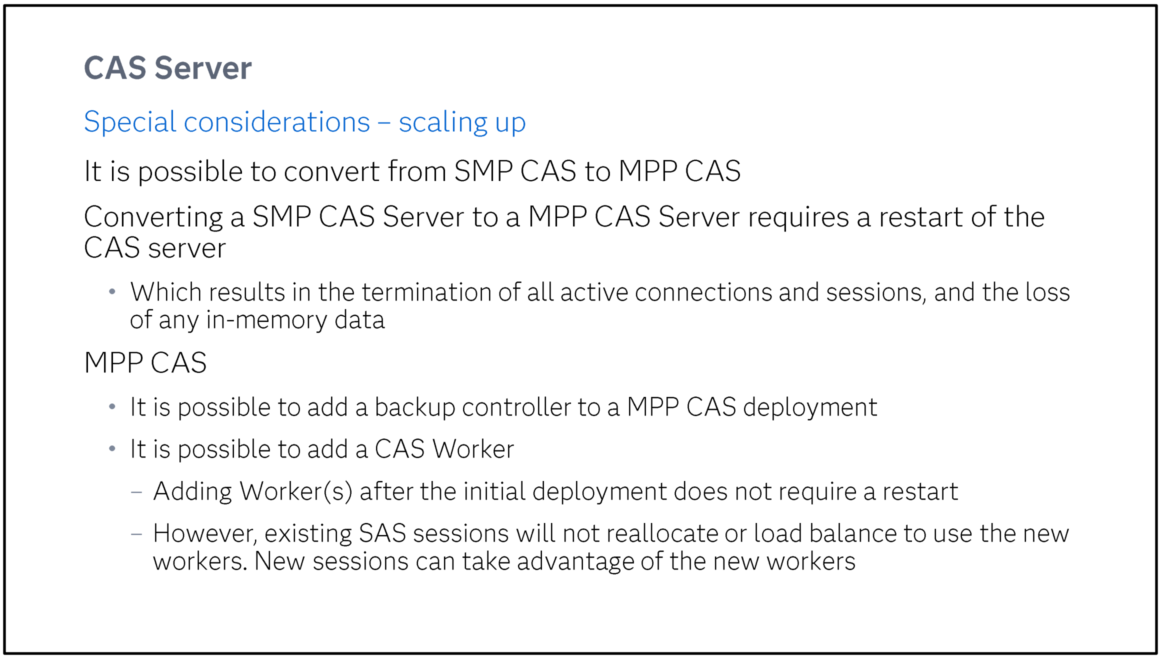
As previously explained, all the nodes running the CAS pods should be the same, the same instance type.

The CAS architecture assumes that maximum efficiency is achieved when each CAS pod is identical. In terms of CPU, RAM, data to process, and other ancillary activities.

MPP CAS uses data distribution to manage workload across multiple hosts, using multiple worker pods.

As a rule, data is evenly distributed to all CAS Workers. That way, each Worker has the same amount of work to do, the same amount of data to process.

But for every rule, there are exceptions. For small tables, the overhead of MPP communication may be too great. And, data partitioning requires keeping grouped data together within a single CAS worker.



CAS Server

Special considerations – scaling up

It is possible to convert from SMP CAS to MPP CAS

Converting a SMP CAS Server to a MPP CAS Server requires a restart of the CAS server

- Which results in the termination of all active connections and sessions, and the loss of any in-memory data

MPP CAS

- It is possible to add a backup controller to a MPP CAS deployment
- It is possible to add a CAS Worker
  - Adding Worker(s) after the initial deployment does not require a restart
  - However, existing SAS sessions will not reallocate or load balance to use the new workers. New sessions can take advantage of the new workers

Now let’s discuss the scaling considerations. Scaling up.

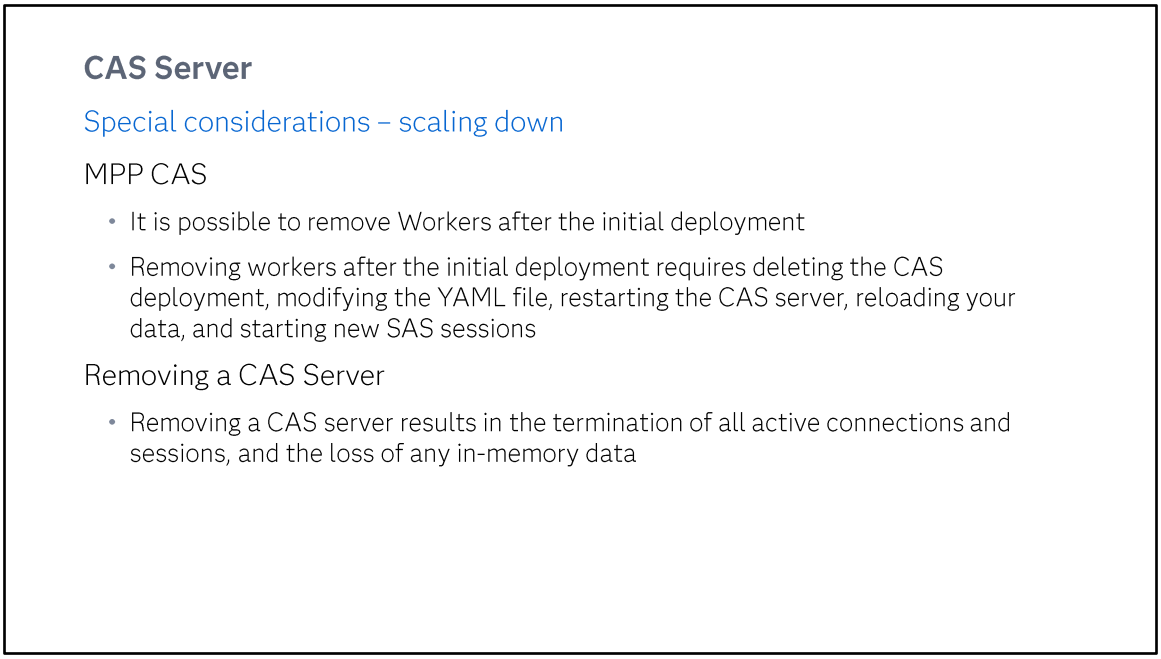
It is possible to convert from a SMP CAS server to a MPP CAS server.

Converting a SMP CAS Server to a MPP CAS Server requires a restart of the CAS server. The restart and reconfiguration of the CAS server, will result in the termination of all active connections and sessions, and the loss of any in-memory data.

For a MPP CAS server it is possible to add a backup controller.

It is possible to add additional CAS Workers. Adding Workers after the initial deployment does not require a restart of the CAS server.
However, existing SAS sessions will not reallocate, or load balance to use the new workers. New sessions can take advantage of the new workers.

By default, any existing loaded data is not automatically rebalanced across the new workers. This behavior is discussed in more detail in another module.



CAS Server

Special considerations – scaling down

MPP CAS

- It is possible to remove Workers after the initial deployment
- Removing workers after the initial deployment requires deleting the CAS deployment, modifying the YAML file, restarting the CAS server, reloading your data, and starting new SAS sessions

Removing a CAS Server

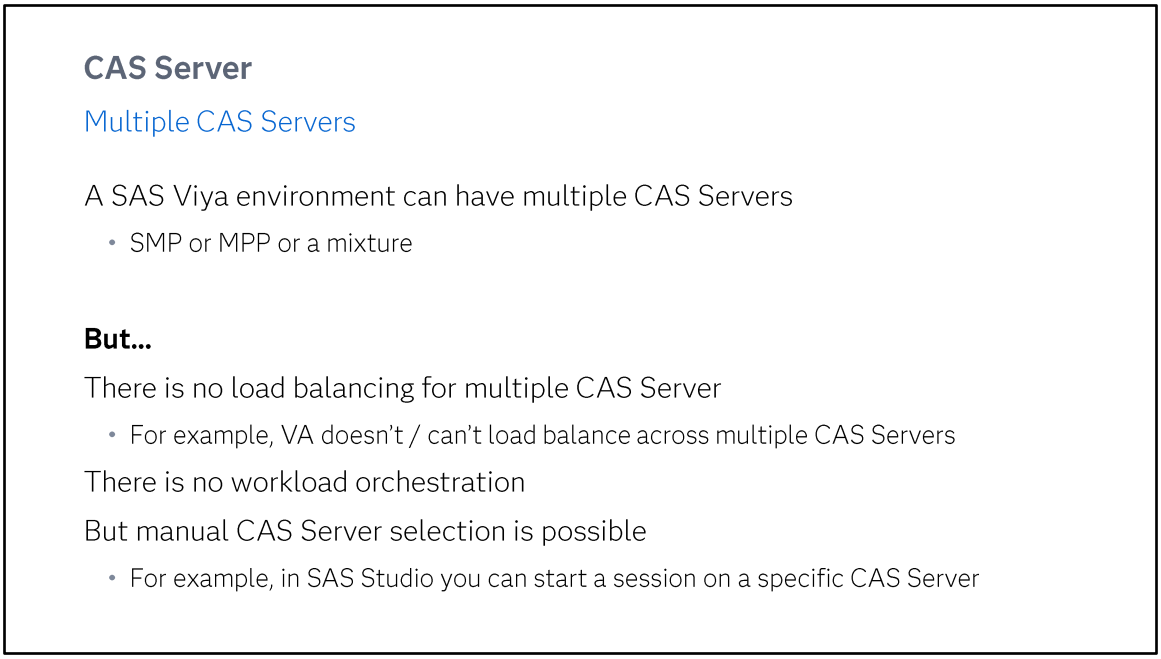
- Removing a CAS server results in the termination of all active connections and sessions, and the loss of any in-memory data

Considerations when scaling down a CAS server.

For an MPP CAS server, it is possible to remove Workers after the initial deployment.

Removing workers after the initial deployment requires deleting the CAS deployment, modifying the CAS configuration, then restarting the CAS server. After the CAS server restart, you need to reload the data, and the users will have to start a new SAS session.

If you have multiple CAS servers defined, it is possible to remove a CAS server. As you would expect, removing a CAS server results in the termination of all active sessions, and the loss of any in-memory data.



CAS Server

Multiple CAS Servers

A SAS Viya environment can have multiple CAS Servers

- SMP or MPP or a mixture

But…

There is no load balancing for multiple CAS Server

- For example, VA doesn't / can't load balance across multiple CAS Servers

There is no workload orchestration

But manual CAS Server selection is possible

- For example, in SAS Studio you can start a session on a specific CAS Server

As stated, it is possible to have multiple CAS servers within the SAS Viya deployment. However, there are some considerations.

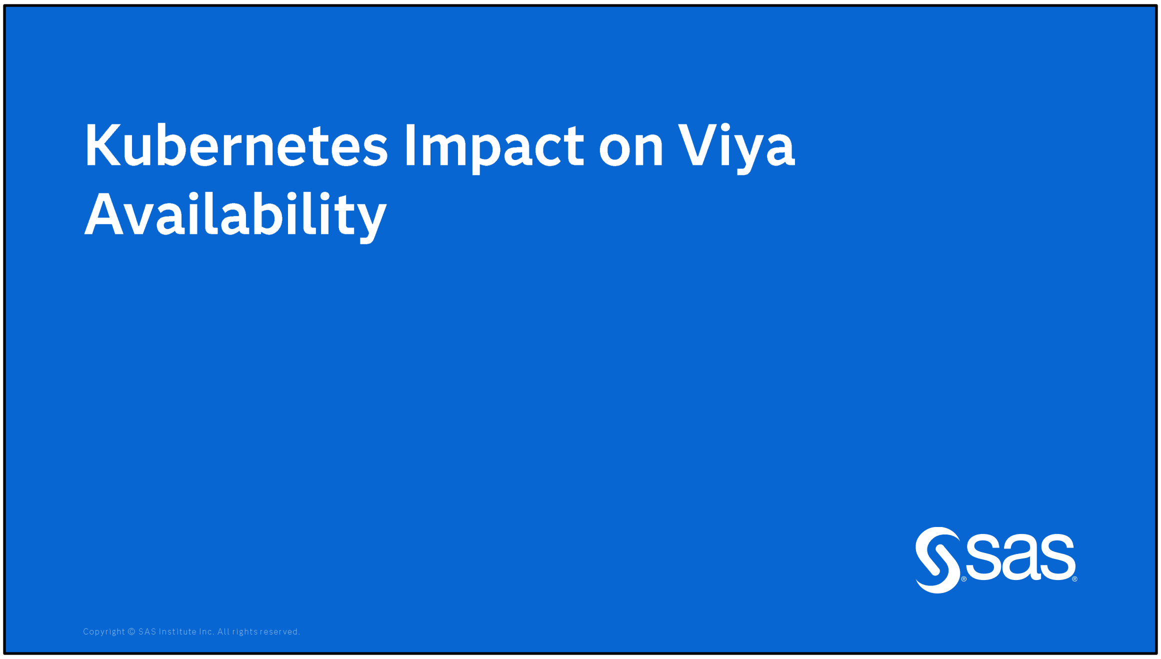
There is no load balancing across the multiple CAS Servers. For example, VA doesn't, or can't, load balance across multiple CAS Servers.

There is no workload orchestration across multiple CAS servers.

But, manual CAS Server selection is possible. For example, if you're programming in SAS Studio you can start a session on a specific CAS Server.

That concludes this look at CAS server scalability.

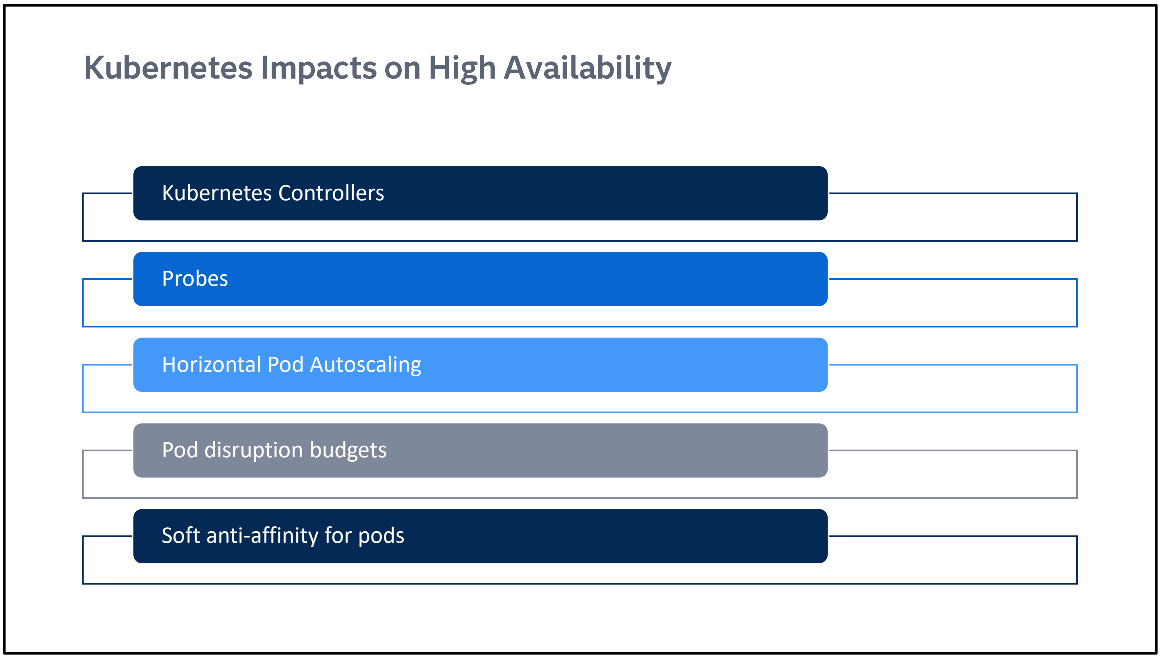
6.2 Availability



Kubernetes Impact on Viya Availability

Kubernetes Impacts on SAS Viya Availability.

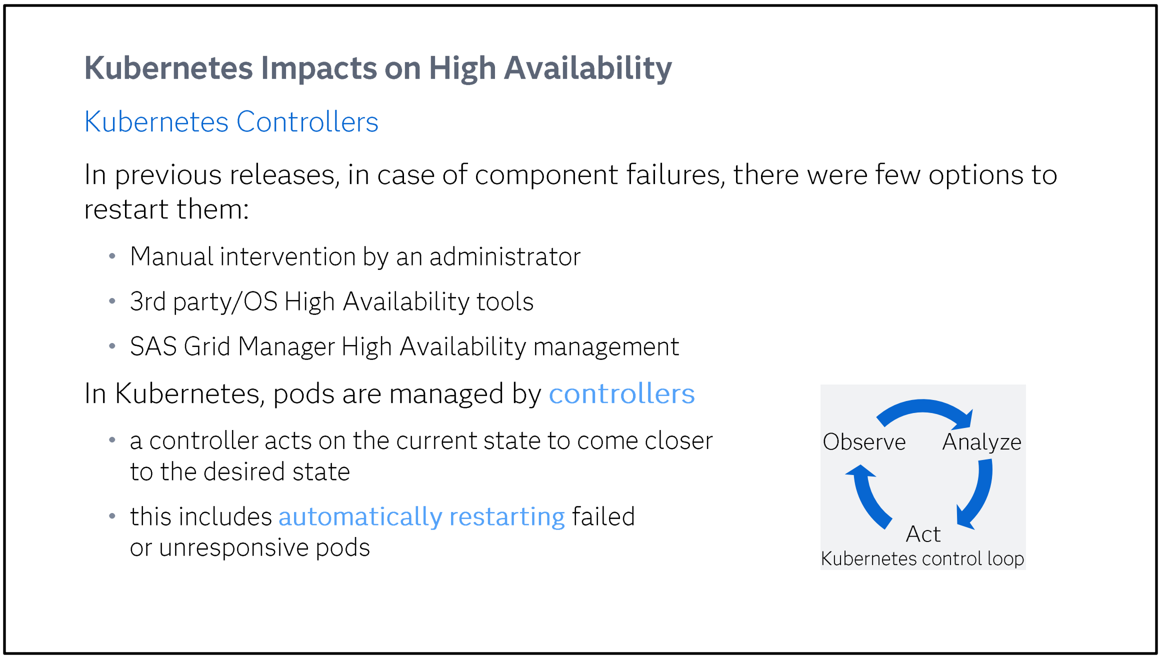
Objectives of this module are to understand the technologies provided by Kubernetes that impact availability, and how SAS Viya components leverage them.



Kubernetes Impacts on High Availability

- Kubernetes Controllers
- Probes
- Horizontal Pod Autoscaling
- Pod disruption budgets
- Soft anti-affinity for pods

Kubernetes provides different constructs, and some of these impact SAS Viya availability. There are controllers, probes, horizontal pod autoscalers, pod disruption budgets, and anti-affinity for pods.



Kubernetes Impacts on High Availability

Kubernetes Controllers

In previous releases, in case of component failures, there were few options to restart them:

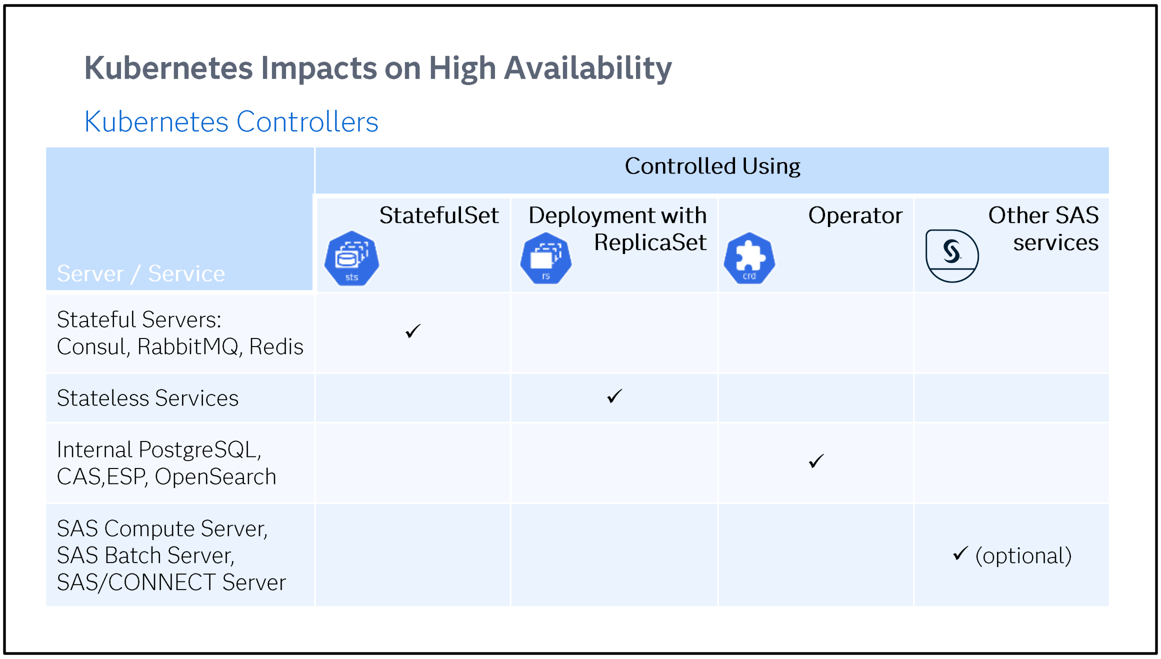
- Manual intervention by an administrator
- 3rd party/OS High Availability tools
- SAS Grid Manager High Availability management

In Kubernetes, pods are managed by controllers

- a controller acts on the current state to come closer to the desired state
- this includes automatically restarting failed or unresponsive pods

In previous releases, in case of a component failure, there were a few options to resume service operation.
An administrator could manually intervene and do something, such as log on to the hosting server, rerun a script to restart the failed component, and verify the success of the operation.
Or you could have deployed third-party tools, or configured tools provided by the operating system, to monitor SAS services and automatically restart failed ones.
Another option was to use SAS Grid Manager. SAS Grid Manager capabilities not only include workload management, but also automatic service monitoring and management to provide higher availability.

With Kubernetes, none of the above is required anymore. Kubernetes pods are managed by controllers, which are control loops that watch the state of your cluster, then make changes where needed. Each controller tries to move the current cluster state closer to the desired state. Usually, the desired state for pods is that you want it to be up and running, so the controller monitors the environment and restarts failed pod.



Kubernetes Impacts on High Availability

Kubernetes Controllers

Server / Service	Controlled Using			
	StatefulSet	Deployment with ReplicaSet	Operator	Other SAS services
Stateful Servers: Consul, RabbitMQ, Redis	✓			
Stateless Services		✓		
Internal PostgreSQL, CAS,ESP, OpenSearch			✓	
SAS Compute Server, SAS Batch Server, SAS/CONNECT Server				✓ (optional)

In this table you can see some SAS Viya components, classified in macro-categories, and the Kubernetes controllers that are used to monitor and manage their pods.

Most stateful servers are managed by a Kubernetes stateful set. By default, the stateful set restarts the pods that is managing.

Stateless services are managed by deployments through replica sets and, again, the replica sets monitor and restart the pods as needed.

Some other servers, including PostgreSQL, CAS, ESP, and OpenSearch, are managed by custom operators. Operators are another Kubernetes construct implemented by pods that embed the business logic required to manage the entities under their control. One of the aspect they manage, is how to keep the controlled pods alive and running.

Finally, backend SAS compute sessions and jobs, such as compute, batch and connect servers, are usually run directly as pods without any controller. They can be configured to be managed by other SAS services. Compute and batch servers can be pre-started, so that their controlling services always keep a minimum number of instances up and running. Batch jobs can be configured to be auto-restarted, in case of failures, by SAS Workload Manager.

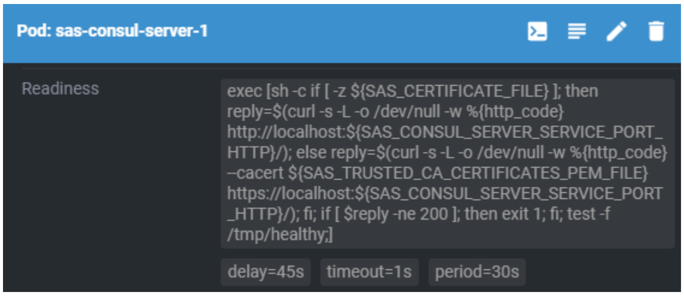
Kubernetes Impacts on High Availability

Probes

Kubernetes can monitor the lifecycle of applications via **probes**.

Probes can take different forms

- check a TCP endpoint
- check an HTTP endpoint
- execute a process



Readiness

```
exec [sh -c if [ -z ${SAS_CERTIFICATE_FILE} ]; then
reply=$(curl -s -L -o /dev/null -w %{http_code}
http://localhost:${SAS_CONSUL_SERVER_SERVICE_PORT_
HTTP}/); else reply=$(curl -s -L -o /dev/null -w %{http_code}
--cacert ${SAS_TRUSTED_CA_CERTIFICATES_PEM_FILE}
https://localhost:${SAS_CONSUL_SERVER_SERVICE_PORT
_HTTP}/); fi; if [ $reply -ne 200 ]; then exit 1; fi; test -f
/tmp/healthy;]
```

delay=45s timeout=1s period=30s

How can Kubernetes monitor pods, so that the controllers can act on them? By using probes.

Probes are declared by the pod developer. Each pod can implement its probes in a custom way, tailored to the business logic of the processes running inside it.

Kubernetes accepts different kind of probes. A probe can be a simple check to see if a TCP endpoint is responsive, or, similarly, if an HTTP endpoint is answering HTTP requests.
More complex probes are implemented by declaring a custom script. Kubernetes periodically executes the script and checks the return code.

In this example we can see a custom probe declared to monitor consul.

Kubernetes Impacts on High Availability

Three Types of Probes

livenessProbe: is the container running?

Status running, ready

- If it fails, Kubernetes kills the container, and the container is subjected to its restart policy.
- Default state : Success.

readinessProbe: is the container ready to respond to requests?

- If it fails, Kubernetes removes the Pod's IP address from the endpoints of all matching Services.
- Default state : initially, Failure. After a delay, Success.

startupProbe: is the application within the container started?

- All other probes are disabled until it succeeds.
- If it fails, Kubernetes kills the container, and the container is subjected to its restart policy.
- Default state : Success.

Probes monitor the main container in each pod. There are different kinds of probes. If a probe is not explicitly declared for a pod, Kubernetes assumes a default state; mostly, it assumes a success.

There are probes for liveness, which are used to check if the container is up and running. If this probe fails, Kubernetes will kill the container. Even if a container is there, but its liveness probe fails, Kubernetes kills it. This is to prevent “zombie” containers that are still there, but processes inside them may be dead or unresponsive. After the container is killed, it is automatically restarted according to its configured policy (which, by default, is “yes, restart”).

Readiness probes are used to check not only that the container is up and running, but also that it can perform its business function. It is used by Kubernetes to decide whether to route incoming request to that specific pod or not. If a readiness probe fails, Kubernetes will not kill the pod, but it will disassociate the pod IP address from the service that is used to reach it on the network. The idea is that, if a process is alive but cannot answer to external calls, let's not even route those requests to its container. This is important when you have a cluster with multiple instances of a microservice. Only pods that are ready to service incoming calls are configured to receive them.

Startup Probes are important, as the name implies, during system startup. If a startup probe is configured and initially false, Kubernetes will not even attempt to run any liveness or readiness probe on that pod. This is a way to inform Kubernetes that the pod is still initializing, so there is no need to check anything else. After the startup is declared successful, then Kubernetes will start to check the liveness and readiness probes. If startup fails, then Kubernetes act just like with liveness probes: Kubernetes will kill the container and start a new one.
Startup probes are important for SAS processes, because they can take some minutes to initialize. Many SAS Viya microservices perform multiple steps before being ready. Without a proper probe, Kubernetes

would kill a pod multiple times while it’s still trying to start.

Kubernetes Impacts on High Availability

Horizontal Pod Autoscalers - HPA

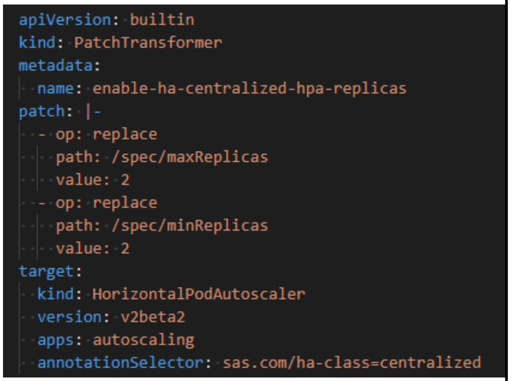
Horizontal Pod Autoscalers automatically scale the Stateless Services based on conditions

- By default, the MIN and MAX values are set to **1**

Use the HA transformer to scale the stateless pods

- By default, the MIN and MAX values are set to **2**

Not everything is controlled by an HPA



```
apiVersion: builtin
kind: PatchTransformer
metadata:
  name: enable-ha-centralized-hpa-replicas
patch: |-
  - op: replace
    path: /spec/maxReplicas
    value: 2
  - op: replace
    path: /spec/minReplicas
    value: 2
target:
  kind: HorizontalPodAutoscaler
  version: v2beta2
  apps: autoscaling
  annotationSelector: sas.com/ha-class=centralized
```

Horizontal Pod Autoscalers are another Kubernetes construct. It is used by Kubernetes to know how to scale a pod based on conditions. With SAS Viya, Horizontal Pod Autoscalers are declared for most stateless services. In a default deployment, they are configured with both min and max properties set to one. This, in practice, disables scaling and clustering, because Kubernetes starts one instance for each microservice.

SAS provides an optional transformer, highlighted in this screenshot, that can be used to configure all SAS-managed HPAs to have min and max set to two. When this happens, Kubernetes creates a second copy of each pod, and it keeps it running. Note that setting min and max to the same value means there is no dynamic scalability, only fixed clustering.

Not everything is controlled by an HPA. A few microservices are always configured to run with a single instance.

Kubernetes Impacts on High Availability

Pod Disruption Budgets

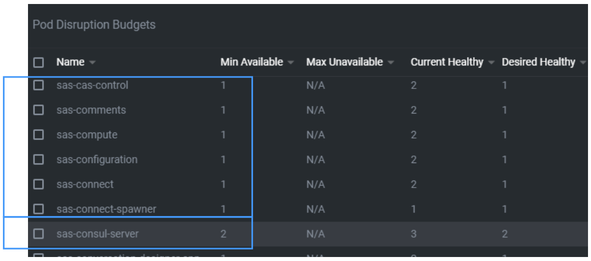
A PDB limits the number of Pods of a replicated application that are down simultaneously from voluntary disruptions.

Quorum-based applications (Consul, RabbitMQ):

- ensure that the number of running replicas is never below the number needed for a quorum.

Everything else:

- ensure that at least one replica is always running.



Pod Disruption Budgets

☐	Name	Min Available	Max Unavailable	Current Healthy	Desired Healthy
☐	sas-cas-control	1	N/A	2	1
☐	sas-comments	1	N/A	2	1
☐	sas-compute	1	N/A	2	1
☐	sas-configuration	1	N/A	2	1
☐	sas-connect	1	N/A	2	1
☐	sas-connect-spawner	1	N/A	1	1
☐	sas-consul-server	2	N/A	3	2

Pod Disruption Budgets are another Kubernetes construct that is usually implemented together with Horizontal Pod Autoscalers.
A Pod Disruption Budget limits the number of pods that can be taken down voluntarily. They are not used in case of failures, but only with controlled restarts. Examples are when an administrator issues a rollout restart command, or when Kubernetes wants to scale down a node.
Kubernetes is a dynamic environment. Kubernetes, at any moment, can take decisions for any component, such as to kill it on one node and to start a new instance of another node.
Without a Pod Disruption Budget, even with clustering, Kubernetes might randomly decide to stop and restart all instances of a service at once. That would obviously cause disruptions to users.
With a Pod Disruption Budget set to X, Kubernetes will try its best to always keep X instances up and running and not kill them all together.

For quorum-based clustered applications, such as Consul and RabbitMQ, the Pod Disruption Budget corresponds to the minimum required number of instances to maintain the quorum. With default clusters of 3 replicas, the Pod Disruption Budget is two. This way, Kubernetes will never voluntarily kill so many pods to lose the quorum.

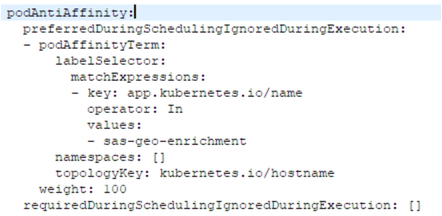
For all other applications, we just want to have one replica always up and running: that's enough to keep the service active. For these services, the Pod Disruption Budget is set to one.

Kubernetes Impacts on High Availability

Soft Anti-Affinity

Soft anti-affinity for stateless and stateful services

- Inform Kubernetes that identical pods have a preference to be scheduled on separate nodes
- but not required to be



```
podAntiAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - podAffinityTerm:
      labelSelector:
        matchExpressions:
        - key: app.kubernetes.io/name
          operator: In
          values:
          - sas-geo-enrichment
      namespaces: []
      topologyKey: kubernetes.io/hostname
    weight: 100
  requiredDuringSchedulingIgnoredDuringExecution: []
```

If all pods of the same service are on the same node:

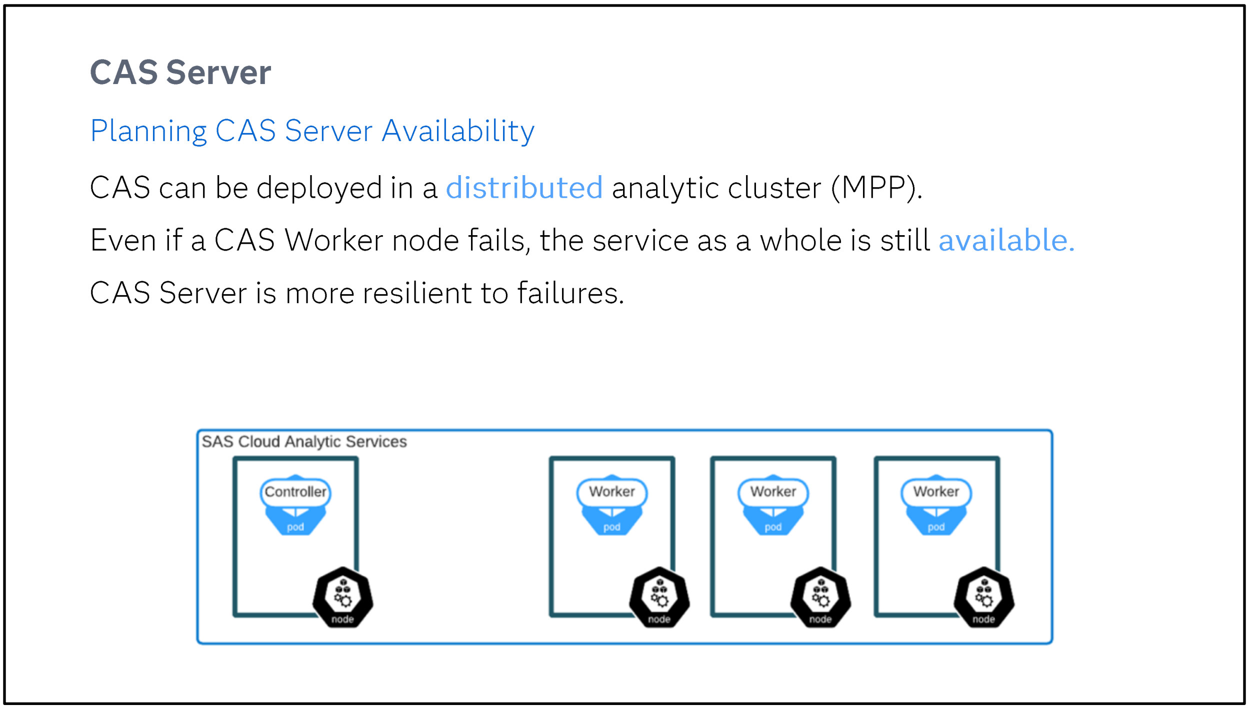
- failure of the node completely disrupts the availability of that service
- draining the node cannot be completed since there would be no surviving instances of that service

The last Kubernetes concept that we'll present are affinities, more specifically, soft anti-affinities. Affinities and anti-affinities are a way to give hints to Kubernetes about the desired pod distribution across multiple nodes.
With anti-affinities we inform Kubernetes that identical pods should go on separate nodes. Soft anti-affinities are a preference, not a requirement. If Kubernetes cannot satisfy the request, for example, because all of the other available nodes are full, it will still start multiple instances of a service all on the same node.

Having all instances of a service on the same node has some consequences.
The most obvious is that overall system availability is compromised because, if that node fails, that specific service is totally lost.
Another unintended consequence is that Kubernetes will not be able to satisfy a node drain command. The reason here is that the Pod Disruption Budget instructs Kubernetes to always keep one instance up and running, while the node drain command asks to stop all the pods on that node, which would cause all instances of that service to stop.
In this case, Kubernetes would report an error and ask the administrator to do something manually to solve the impasse. For example, the administrator could decide to accept a temporary service unavailability and set the Pod Disruption Budget to zero, or they could free some space on another node to manually move at least one instance there.

CAS Availability - Architecture

Let’s introduce how CAS Architecture design impact its availability, especially with regards to SMP versus MPP, and deployments with or without a backup controller.



CAS Server

Planning CAS Server Availability

CAS can be deployed in a distributed analytic cluster (MPP).

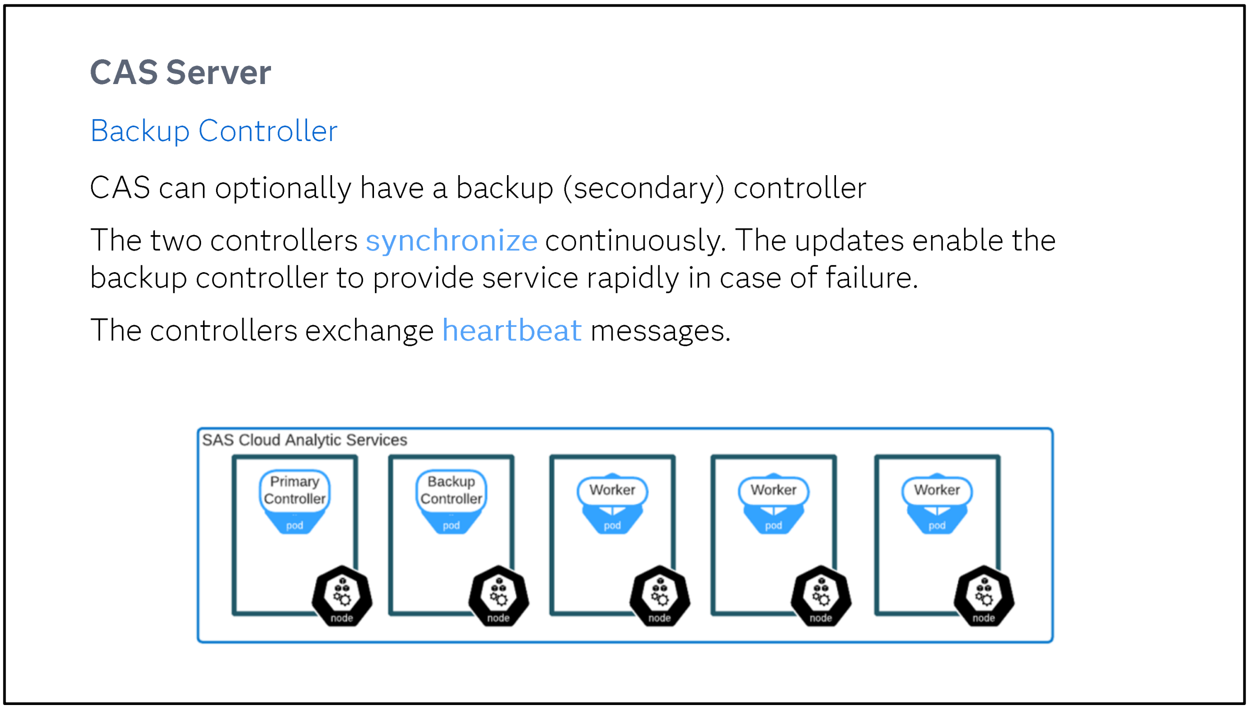
Even if a CAS Worker node fails, the service as a whole is still available.

CAS Server is more resilient to failures.

CAS can be deployed as a distributed server in an analytic cluster. We call this MPP.
In this configuration, there are at least one controller pod and two or more worker pods, possibly distributed to different infrastructure nodes.

In this case, even if a CAS worker fails, the service as a whole is still available to service user requests.

This capability makes CAS more resilient to failures, when compared to equivalent services available with previous versions.
If you are familiar with SAS 9, some offerings include the LASR server, the precursor to CAS. With LASR, if a node fails, the whole cluster is lost. This is not the case with CAS.



CAS Server

Backup Controller

CAS can optionally have a backup (secondary) controller

The two controllers synchronize continuously. The updates enable the backup controller to provide service rapidly in case of failure.

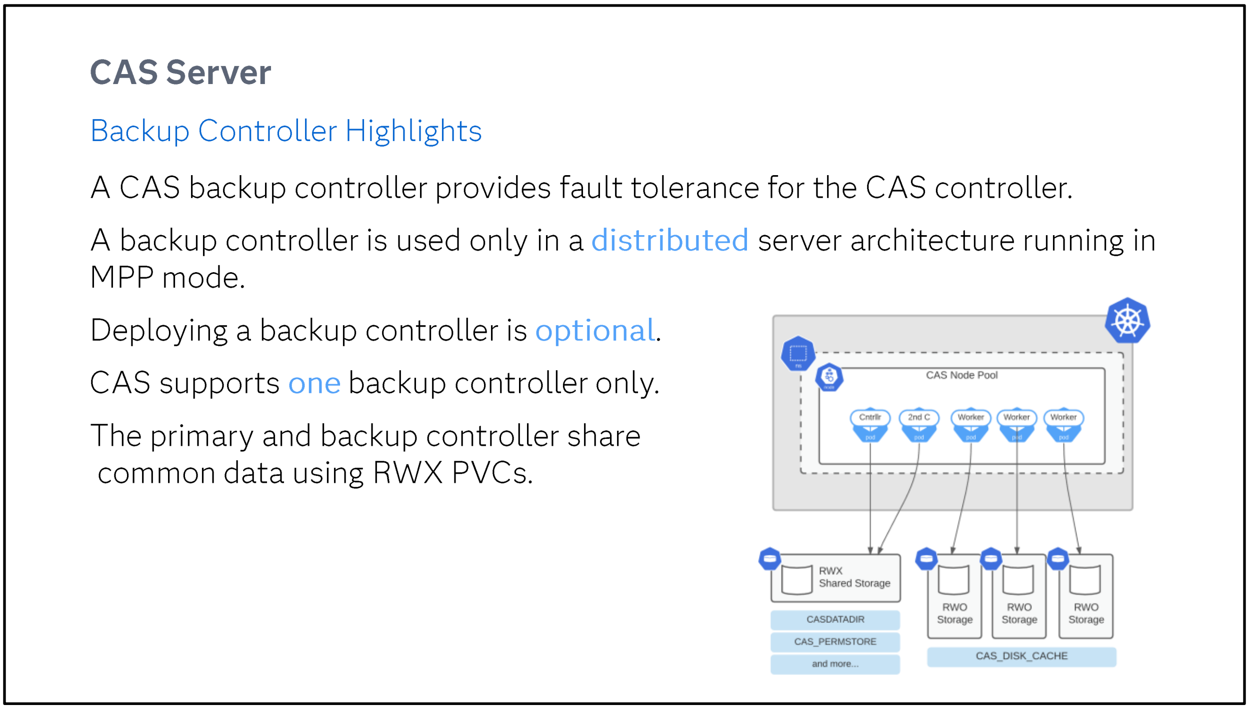
The controllers exchange heartbeat messages.

You might have noticed that, in the previous slide, we only talked about resiliency to CAS worker failures. What about the controller? We do not want that to be a single point of failure.

A CAS server can optionally include a backup or secondary controller.

The primary and the backup controller keep themselves in sync, and this synchronization enables the backup controller to rapidly step in and provide service in case of failure of the primary one.
How can the secondary controller know when to step in?

The two controllers continually exchange heartbeat messages between themselves. The primary knows if the secondary is up and vice versa.
When this heartbeat check is disrupted, then the backup controller understands that the primary is down and steps in.



CAS Server

Backup Controller Highlights

A CAS backup controller provides fault tolerance for the CAS controller.

A backup controller is used only in a distributed server architecture running in MPP mode.

Deploying a backup controller is optional.

CAS supports one backup controller only.

The primary and backup controller share common data using RWX PVCs.

Let’s see some more details about CAS backup controllers.
We have seen that a backup controller provides fault tolerance to the primary controller.

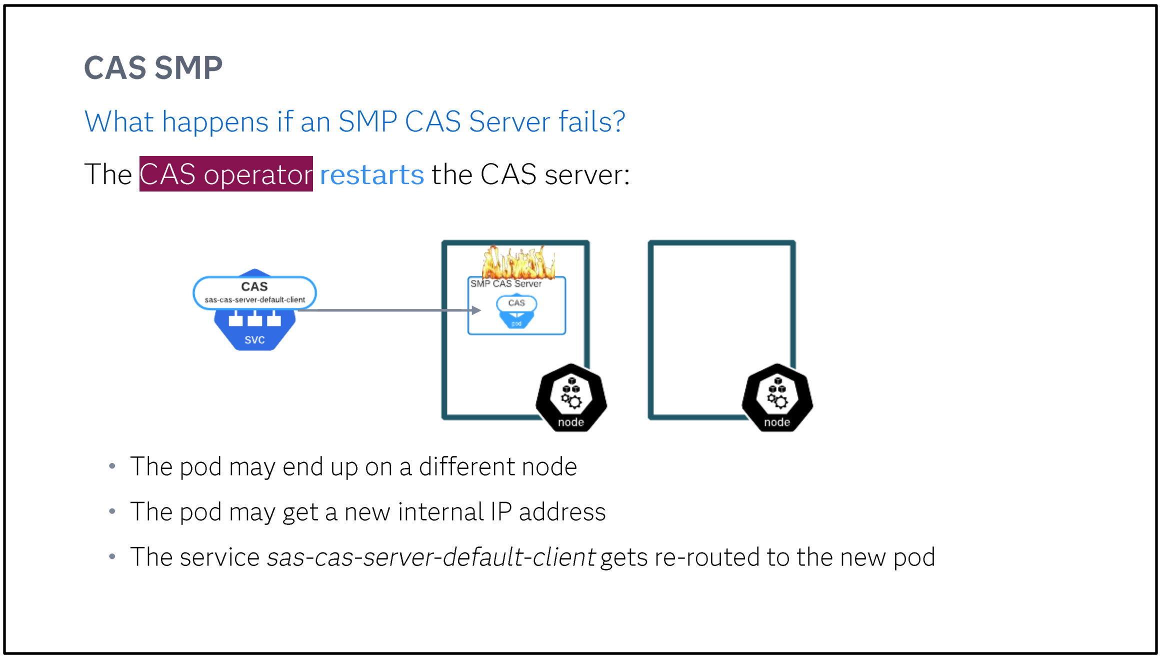
This capability is only available with a distributed architecture: it's not available if you have an SMP CAS server. If you have a single-node SMP CAS server, you cannot simply say “okay I will start a second node to become its backup”. It will not work.

Deploying a backup controller is optional. If you choose to deploy it, you can only have one instance, which means that the server can survive a controller failure, but no more than one.

The primary and the backup controllers share common data using a physical volume available to both pods with RWX mounts.
This is a change from previous versions.
With SAS Viya 3, each controller has a local copy of the common data: in case of failures, an administrator has to manually synchronize it, before recovering the failed primary. With SAS Viya 4 this storage is shared, and no further synchronization is required.

CAS Availability - Use Cases

Let’s now present different use cases in which a CAS component may fail.
We will see how CAS can minimize the impact of failures thanks to its availability capability.
We will also review what administrators or end users might have to do to recover in case of failures.



CAS SMP

What happens if an SMP CAS Server fails?

The CAS operator restarts the CAS server:

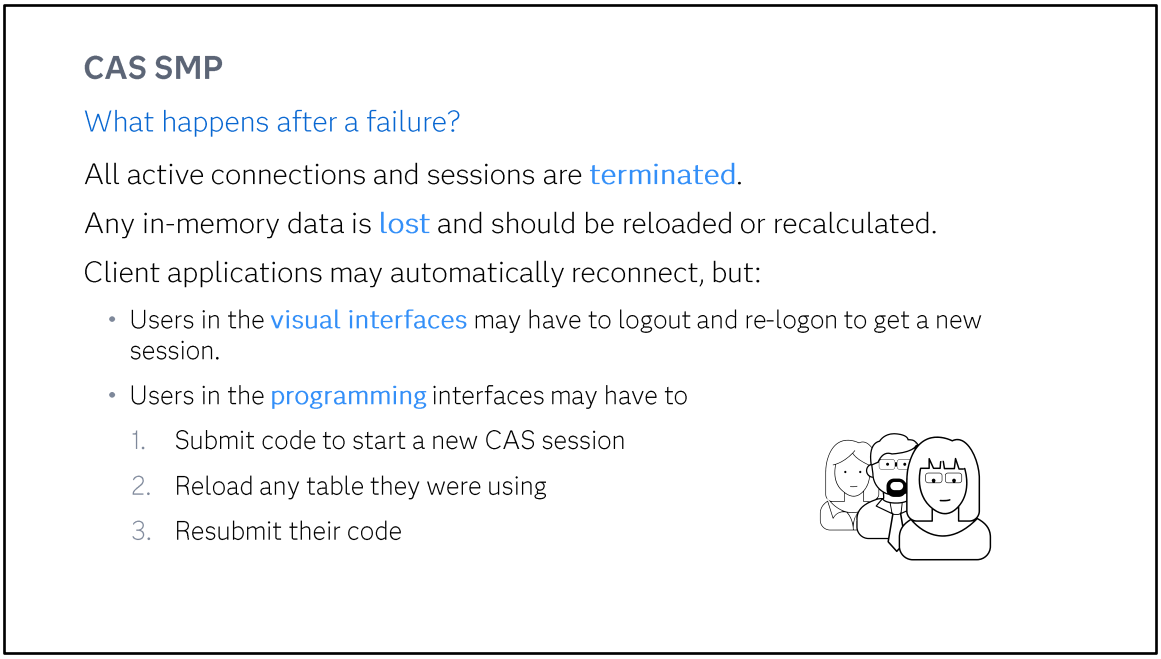
- The pod may end up on a different node
- The pod may get a new internal IP address
- The service *sas-cas-server-default-client* gets re-routed to the new pod

CAS servers are managed by the CAS operator. The operator manages every server's lifecycle, including reacting to failures.
You may remember that, when you have a SMP deployment, it means your instance runs in a single pod.

If it dies, it will be restarted by the CAS operator, but it may end up on a different node.

By ending up on a different node, it may get a new internal IP address, but that's not important. Kubernetes provides a service called SAS CAS server default client.

The service is automatically rerouted to the new pod. Clients are unaware if they are connected to the original pod, or to the newly restarted one. This is true for internal clients, such as microservices, but also for external ones, such as a Python program running on a laptop. Clients are redirected to the new pod because they are connected through the service.



CAS SMP

What happens after a failure?

All active connections and sessions are terminated.

Any in-memory data is lost and should be reloaded or recalculated.

Client applications may automatically reconnect, but:

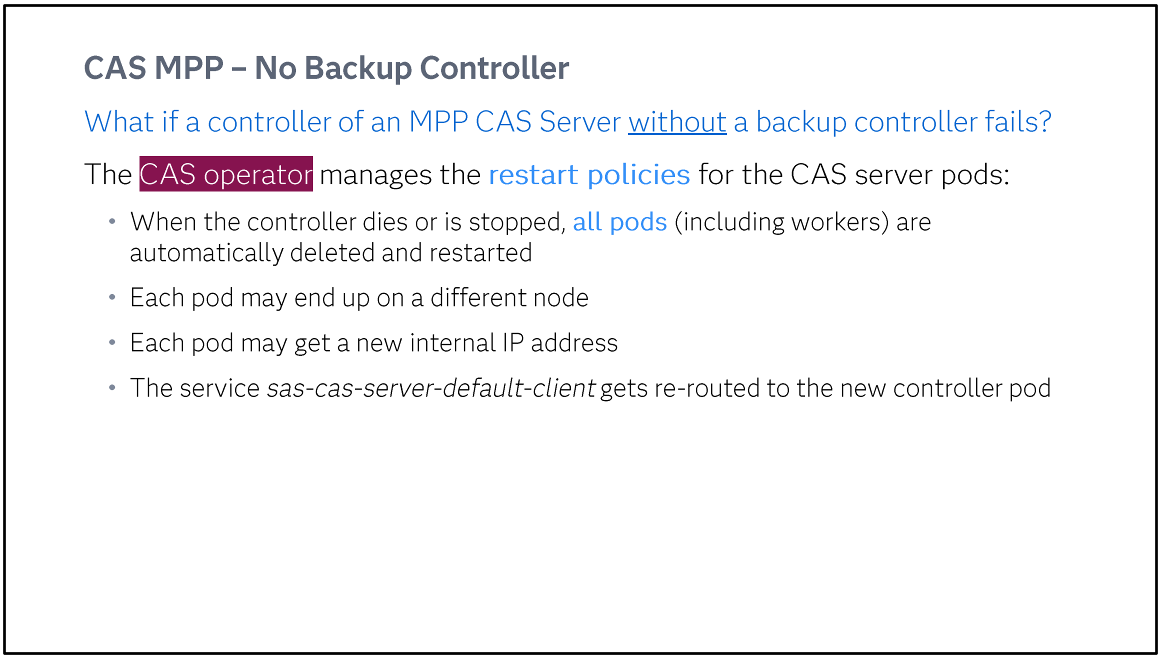
- Users in the visual interfaces may have to logout and re-logon to get a new session.
- Users in the programming interfaces may have to
  1. Submit code to start a new CAS session
  2. Reload any table they were using
  3. Resubmit their code

This automatic redirection is not transparent to users.

After the CAS pod goes down and gets restarted – possibly on another node – all active connections are severed and terminated.

If there was any data loaded in the memory of the failed pod, that's lost. When the CAS server is restarted, data can be reloaded or recalculated on the new instance.

Usually, client applications survive this kind of failures. Clients can automatically reconnect. As an example, if you're browsing a report in SAS Visual Analytics, the report can still connect to the backend server. But you might have to recover it by logging out and then back in.
Programming interfaces, such a python program, might not automatically reconnect.
You might have to resubmit code to start a new user session in the restarted CAS server. The previous session is lost when the pod restarts. Once a new session is established, you have to reload the tables that you were using, before you can submit again the code that was running previously.
In summary, it's quite a disruption to the end users.



CAS MPP – No Backup Controller

What if a controller of an MPP CAS Server without a backup controller fails?

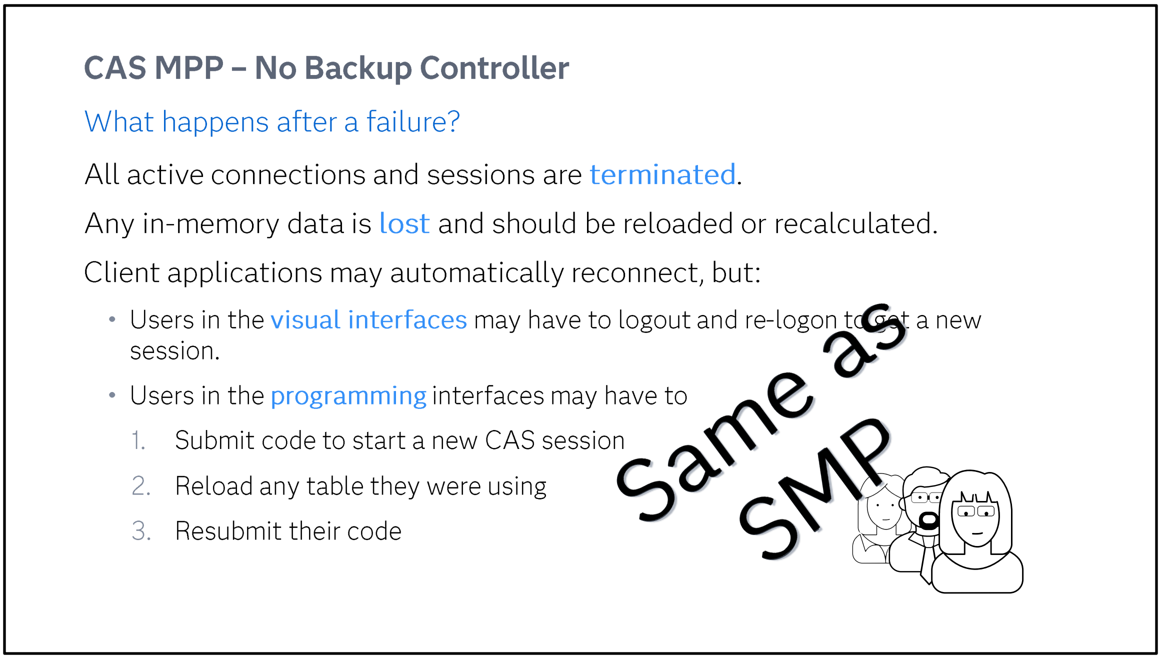
The CAS operator manages the restart policies for the CAS server pods:

- When the controller dies or is stopped, all pods (including workers) are automatically deleted and restarted
- Each pod may end up on a different node
- Each pod may get a new internal IP address
- The service *sas-cas-server-default-client* gets re-routed to the new controller pod

Let’s see if things can be better with a distributed environment. What happens with a MPP CAS server, configured without a backup controller, if the only controller fails?

Just as in the previous case, the CAS operator manages the restart policies for the CAS server pods. In this case, the operator instructs all the pods, including the workers, to stop and restart. Even if these pods had no issues. With this restart, all pods may end up on different nodes and may get new internal IP addresses.

Again, just as the previous case, the frontend service – SAS CAS server default client – is automatically rerouted to the new controller pod. And so from the outside, clients can still connect using the same service.



CAS MPP – No Backup Controller

What happens after a failure?

All active connections and sessions are terminated.

Any in-memory data is lost and should be reloaded or recalculated.

Client applications may automatically reconnect, but:

- Users in the visual interfaces may have to logout and re-logon to get a new session.
- Users in the programming interfaces may have to
  1. Submit code to start a new CAS session
  2. Reload any table they were using
  3. Resubmit their code

Same as SMP

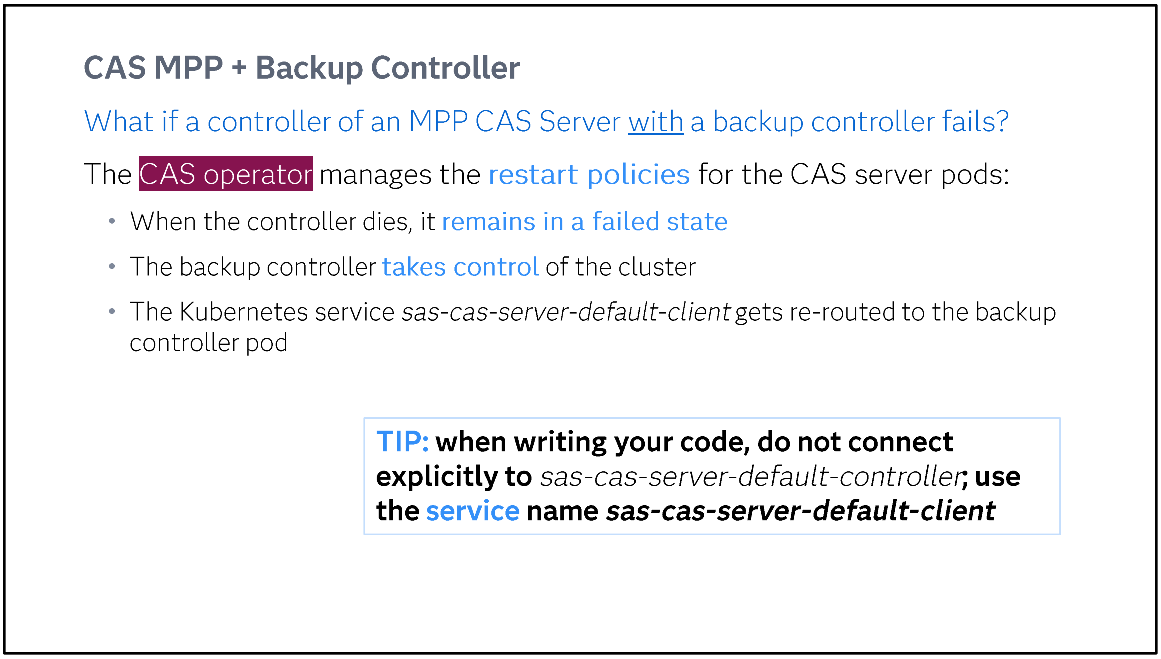
Again, this automatic redirection is not transparent to users.

Just as before, all user sessions are terminated. This is why the CAS operator kills all the worker pods as soon as it detects a failure of the controller. In fact, the state of these sessions is only stored in the controller. The workers don't know anything about the state of these end user sessions, which are lost as soon as the controller fails.

And once the end user sessions are lost, there is no point in keeping data available on the worker pods. In short, all pods get reset and restarted.

At this point, we are in the same situation as we are seen with a single-node SMP server.

From a client point of view, there is no difference between losing a single-node SMP server or losing a controller of a distributed environment that does not include a backup controller.



CAS MPP + Backup Controller

What if a controller of an MPP CAS Server with a backup controller fails?

The CAS operator manages the restart policies for the CAS server pods:

- When the controller dies, it remains in a failed state
- The backup controller takes control of the cluster
- The Kubernetes service *sas-cas-server-default-client* gets re-routed to the backup controller pod

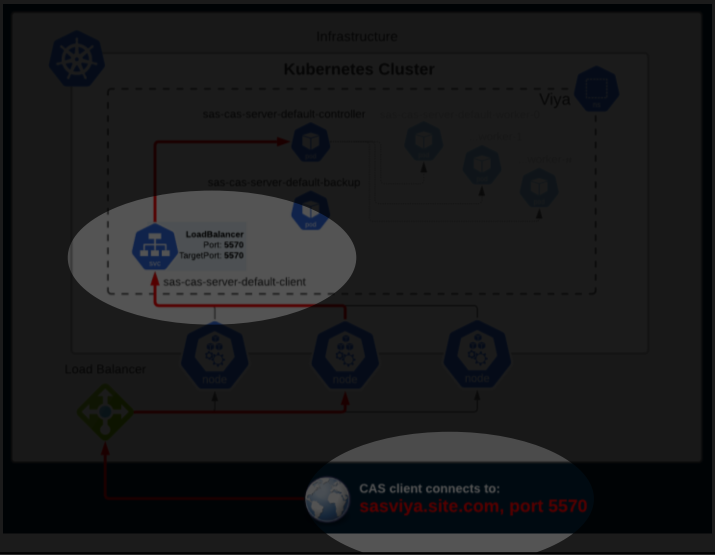
TIP: when writing your code, do not connect explicitly to *sas-cas-server-default-controller***; use the service name *sas-cas-server-default-client***

You can see a different behavior when you have a distributed environment that includes a backup controller.
Just as always, as the primary controller fails, the CAS operator detects the failure and reacts.
But, in this case, its behavior is different. The operator maintains the failed primary controller in a stopped state, without restarting it. Then it will mark the backup controller as the new primary.
The backup controller, through the missed heartbeat, has already noticed that the primary is down and is already setting itself ready to service clients and to take control of the workers.
Just as always, the Kubernetes service, SAS CAS server default client, gets rerouted. This time not to a restarted controller, but rather to the backup controller.
This way, all clients, just by connecting to the same service, can use the same CAS server without noticing that there was a switch in the controller.
This is different than what happened with SAS Viya 3. In that case, there was no front-end service and clients directly connected to the hostname of the primary controller. To support failover, SAS Viya 3 clients have to be aware of both primary and backup controller hostnames, and manage themselves a connection switchover.

With SAS Viya 4, we don’t connect to a hostname. We connect to a Kubernetes service. There is a service referencing the primary controller, and one pointing to the backup controller, but these should not be directly used. We want to connect to a third service, the one called SAS CAS server default client, because this is the one that is managed automatically, and it is always connected to the live controller.

On a side note. If the primary controller is stopped “cleanly” – that is, it does not die abruptly – the CAS Operator interprets that as an administrative command to shutdown the cluster and stops all CAS pods.

CAS MPP: Connecting to the Primary Controller



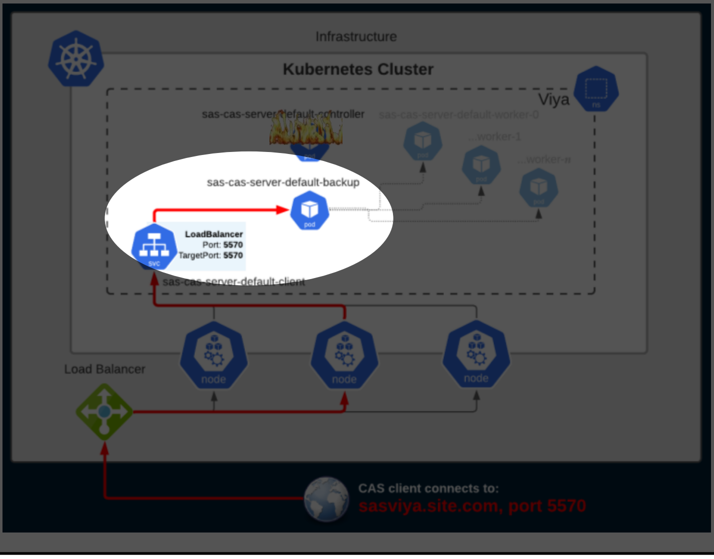
Let’s see a visual representation of this frontend client service switching between primary and backup controllers in case of failures.

Here you can see, at the bottom, a CAS client calling into the SAS Viya environment.

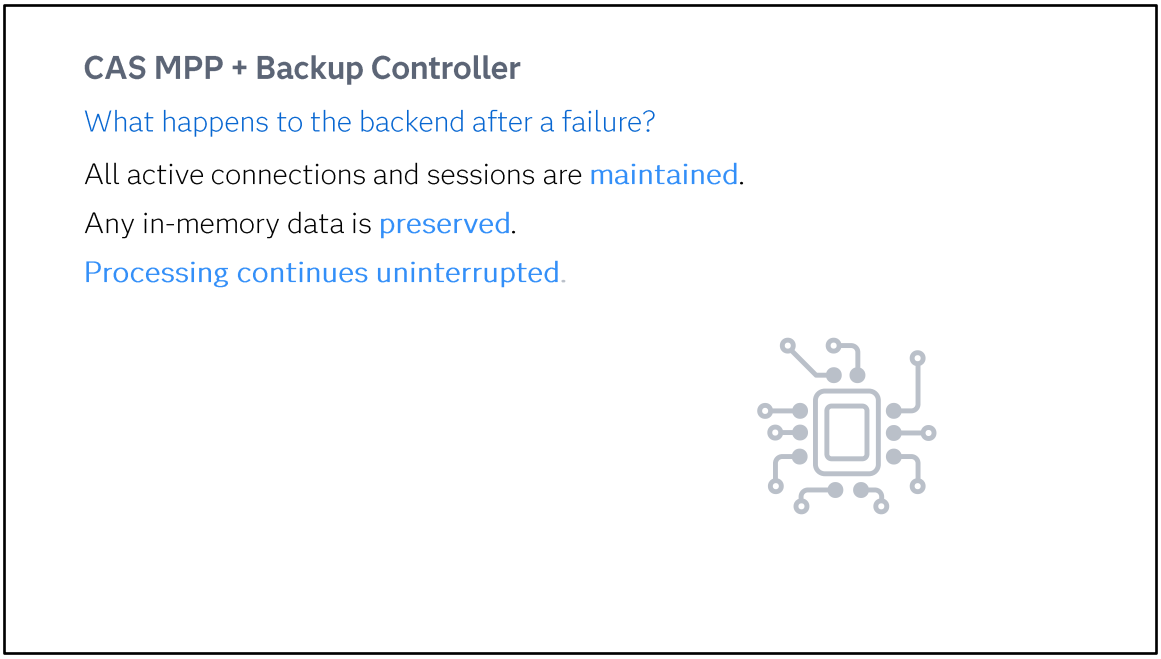
The route goes through multiple services, until it reaches the CAS primary controller.

Here in the center is the SAS CAS server default client, the one to which all internal and external clients connect to.

CAS MPP: Connecting After a Failure



When the primary controller fails, the backup controller steps in, and service is rerouted to it automatically.



CAS MPP + Backup Controller

What happens to the backend after a failure?

All active connections and sessions are maintained.

Any in-memory data is preserved.

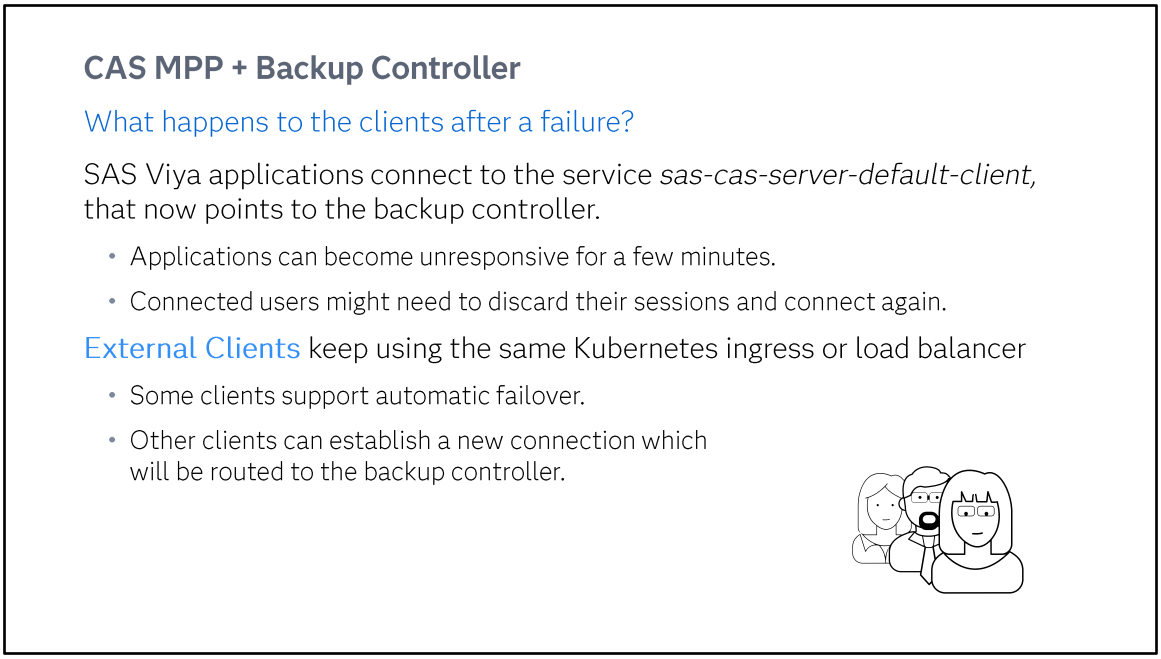
Processing continues uninterrupted.

What happens to user sessions and tables loaded in memory in this case?

When a backup controller is available, all the active connections and sessions are maintained. Why? That’s because the backup controller is always in sync with the primary controller. It has in memory the full state of the environment.

Since user sessions survive, the CAS operator does not need to kill and restart the CAS workers. This implies that all in-memory data and tables are preserved.

In summary, processing can continue uninterrupted.



CAS MPP + Backup Controller

What happens to the clients after a failure?

SAS Viya applications connect to the service *sas-cas-server-default-client,* that now points to the backup controller.

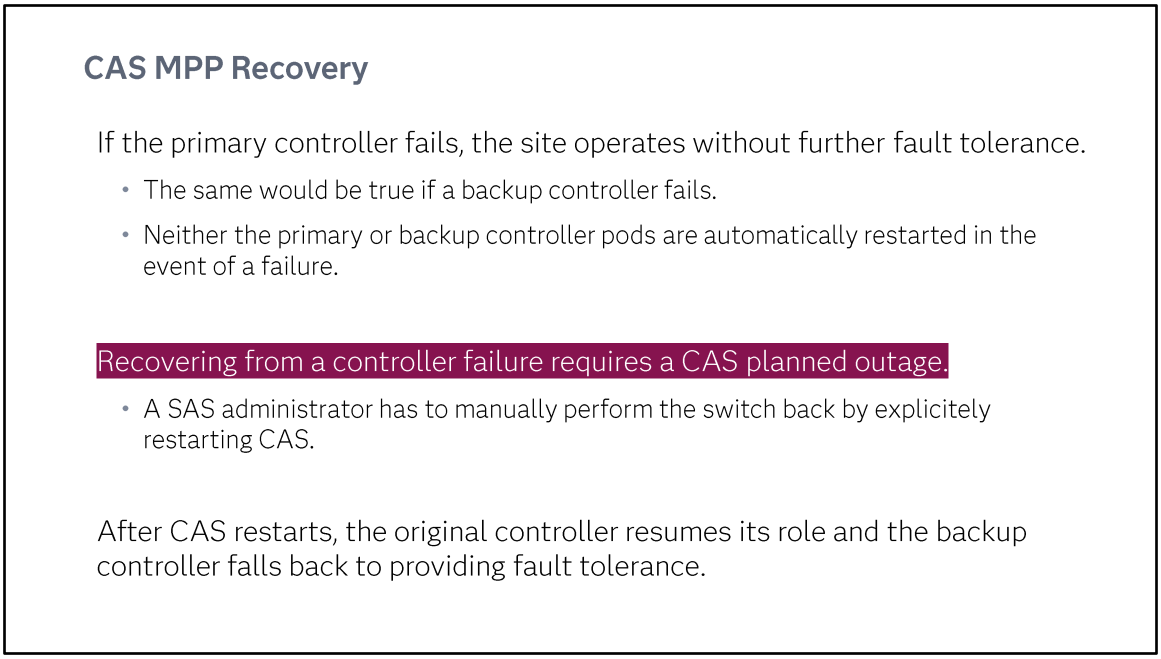
- Applications can become unresponsive for a few minutes.
- Connected users might need to discard their sessions and connect again.

External Clients keep using the same Kubernetes ingress or load balancer

- Some clients support automatic failover.
- Other clients can establish a new connection which will be routed to the backup controller.

And, finally, what is the user experience in this case?

We have seen that client applications connect to the “SAS CAS server default client” service, either directly (for internal clients) or through a frontend load balancer or ingress (for external clients). And this service automatically points to the backup controller, when a failure of the primary occurs.
The applications may still become unresponsive for a few minutes. So it might be noticeable for users. But usually, after one or two minutes, the environment should be back as it was before.
Those one or two minutes are not only due to CAS being unresponsive during the switch. The issue is that it might take the controller and the service a few seconds to notice the failure and react. These few seconds of CAS being unresponsive might cascade to other services, which become themselves unresponsive for another few seconds. In the end, all these little few seconds add up to become one or two minutes.
The good news is that users don’t have to do anything. They are still in their application, and everything resumes working automatically.
If a user wants to speed up recovery, they might discard their session by logging out and reconnecting. In this case, they will be reconnected to the backup controller, and everything restarts immediately.
External clients usually are connected through an ingress or a load balancer. Since the ingress or load balancer do not change, the client might, at most, notice that the backend is unresponsive for a few seconds and then move on.
For example, Python through SWAT can failover automatically. You don't have to do anything.



CAS MPP Recovery

If the primary controller fails, the site operates without further fault tolerance.

- The same would be true if a backup controller fails.
- Neither the primary or backup controller pods are automatically restarted in the event of a failure.

Recovering from a controller failure requires a CAS planned outage.

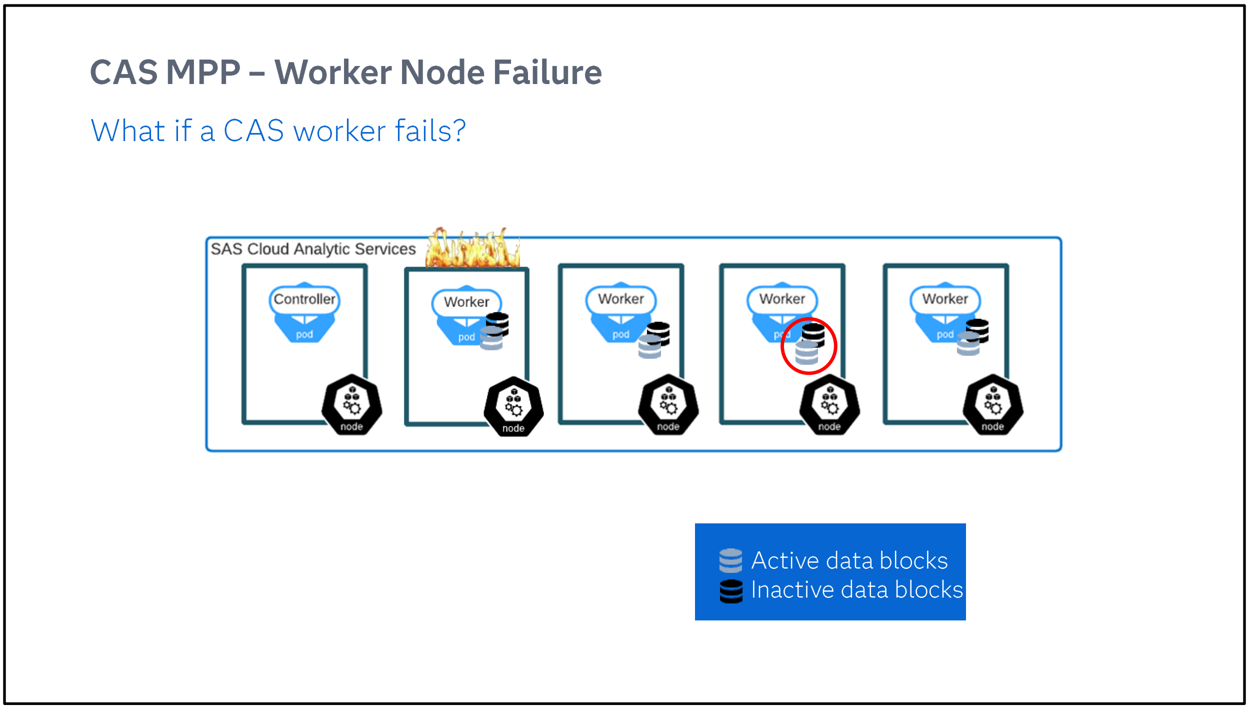
- A SAS administrator has to manually perform the switch back by explicitely restarting CAS.

After CAS restarts, the original controller resumes its role and the backup controller falls back to providing fault tolerance.

We still in the distributed environment case, with a backup controller.
We want to address what happens after the primary controller fails, and the backup controller steps in.
We have seen that the environment is still up and running, but we noted that the CAS operator keeps the failed pod down, it will not restart it.
And the same would happen if, instead, the backup controller would fail first: the primary controller would keep running, but the CAS operator would not restart the backup controller pod.
In summary, if one controller fails, the environment is still operational, but without any further fault tolerance.

Recovering from a controller failure, whether it is the primary of the backup, always requires a full CAS restart. This can obviously be planned to mitigate user disruption. A SAS administrator can do it whenever they want. Usually, it's better to recover as soon as possible, because now we are running with only one controller, with no additional failover capability.

Once CAS restarts, the original controller resumes its role as primary, and the backup controller falls back to be providing fault tolerance.



CAS MPP – Worker Node Failure

What if a CAS worker fails?

- Active data blocks
- Inactive data blocks

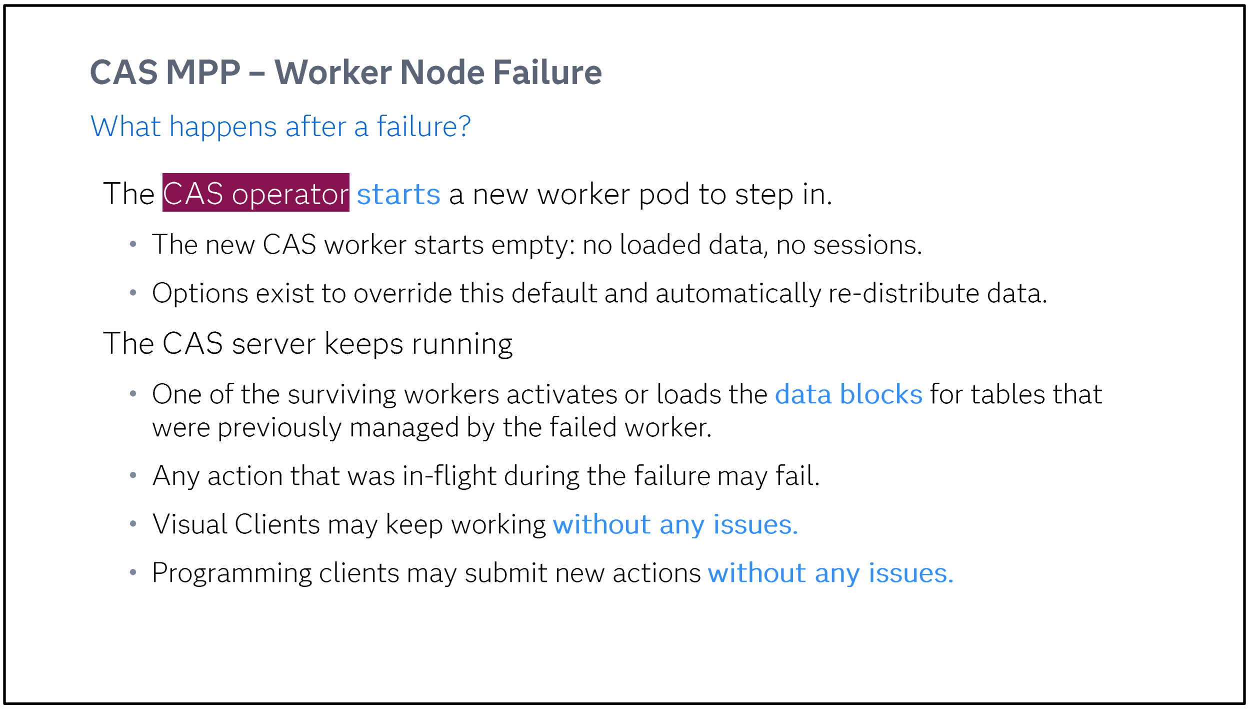
That’s all with regards to controller failures and recovery.
Let’s now move to the next topic, let's talk about CAS workers. What happens if a worker fails?
It's important to understand that, by default, data is loaded into CAS workers with multiple copies to improve resilience.
In this picture, gray icons represent data blocks that are loaded in memory and active in the CAS sessions. Each one has a replica, represented by a black icon. Replicas are inactive data blocks and are saved on disk, in the CAS disk cache of a different worker then the one holding the original copy. The CAS controller has a list of each data block, primary or replica, including where it is located, and whether it is active in memory or in stand-by on disk.

In this configuration, what happens when a worker dies?

The controller knows which workers manage the copies of the data blocks previously hosted by the failed one.

The controller can instruct these workers to load that inactive data into their memory and make active again. This way, the data is still available to service user requests.
All this process has been available since SAS Viya 3. SAS Viya 4 introduces a key difference from here on, thanks to Kubernetes.
The failed CAS worker, form a Kubernetes point of view, is just a dead pod managed by the CAS operator.

And just as with any failed Kubernetes pod, the CAS operator will restart it. Based on the chosen topology, it might restart it on the same node or on a different node, but the worker gets restarted.
Notice in this picture how there are no data blocks on the newly restarted CAS worker: by default, it gets restarted empty.



CAS MPP – Worker Node Failure

What happens after a failure?

The CAS operator starts a new worker pod to step in.

- The new CAS worker starts empty: no loaded data, no sessions.
- Options exist to override this default and automatically re-distribute data.

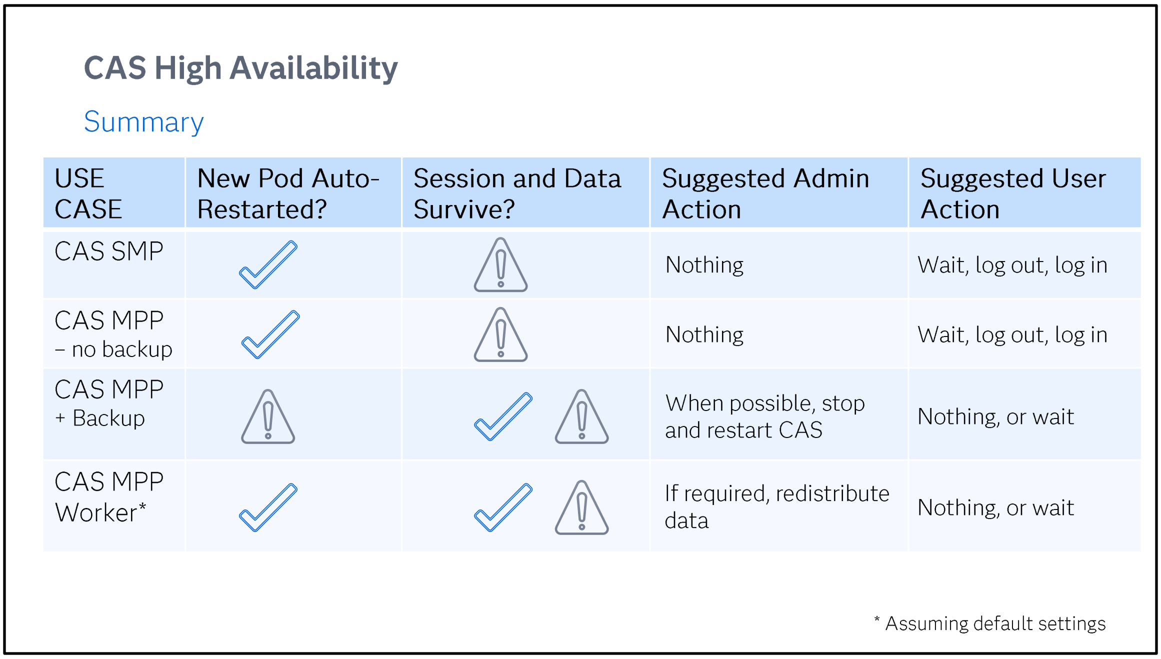
The CAS server keeps running

- One of the surviving workers activates or loads the data blocks for tables that were previously managed by the failed worker.
- Any action that was in-flight during the failure may fail.
- Visual Clients may keep working without any issues.
- Programming clients may submit new actions without any issues.

So, what’s the situation after a worker failure and recovery?

We have seen that the CAS operator restarts a new worker pod to step in. If we started with four worker nodes, as in the previous example, we’ll end up with four workers again.
By default, data distribution across nodes becomes skewed. After a worker failure, data is only available on the surviving original nodes. You might remember, from the CAS architecture lesson, that CAS is only as fast as its slowest node. After activating the data copies of the failed worker, probably now some worker will host more data than the others. This worker will be slower because it has to analyze a bit more data, and so the whole CAS server will be as slow as this node.
If you load any new data after the failed worker node is restarted, then the new data will be also placed on the restarted worker.
But the data existing from before the failure is not automatically re-distributed.
There are options that can be set per table or per library to modify this behavior, so that tables can be automatically re-distributed back into the restarted worker.
Otherwise, a data administrator can submit code to reshuffle the data to re-distribute it across all workers.

The CAS server, as a whole, keeps running unaffected by a worker failure.
If the server was processing an action during the failure, that single action, that was in flight, may fail. Therefore, the user might get an error pop up in the frontend client.
Usually, visual clients can recover automatically, without any warning to the user.
Programmers might be impacted. Maybe you were submitting CAS code through SAS Studio or through Python: you might get back a failure for that specific submission. In this case, simply resubmit it again, and it should run successfully.



CAS High Availability

Summary

USE CASE	New Pod Auto-Restarted?	Session and Data Survive?	Suggested Admin Action	Suggested User Action
CAS SMP	✓	⚠	Nothing	Wait, log out, log in
CAS MPP – no backup	✓	⚠	Nothing	Wait, log out, log in
CAS MPP + Backup	⚠	✓ ⚠	When possible, stop and restart CAS	Nothing, or wait
CAS MPP Worker*	✓	✓ ⚠	If required, redistribute data	Nothing, or wait

* Assuming default settings

This table summarizes all the cases that we have seen in this lesson about CAS server availability. Each line presents one use case.
Each column addresses whether the failed pod is automatically re-started, whether sessions and data survive the failure, what actions are suggested to SAS administrators and to SAS users.

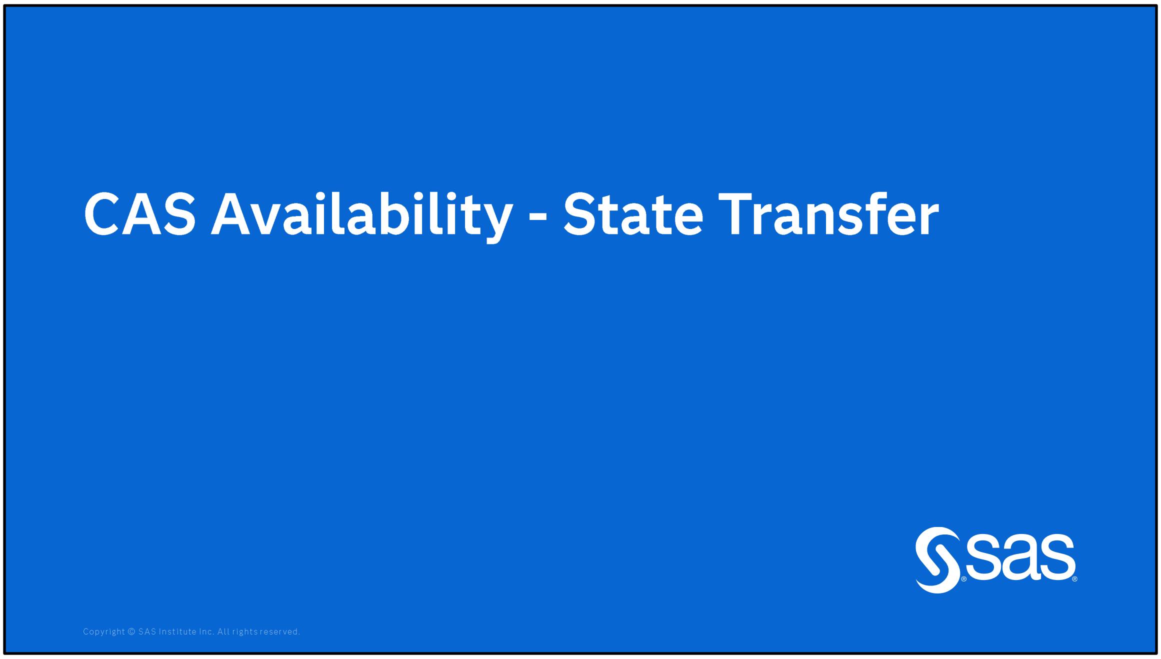
Starting with the SMP CAS case, we have seen that the CAS operator starts a new pod in case of failure. Since there is only one pod, sessions and data do not survive. SAS administrator don't have to do anything to recover, but users might have either to wait or, in the worst case, log out and log back in.

We have seen that the exact same considerations apply to the MPP case, when there is no backup controller.

Instead, if you configure a backup controller in a distributed environment, then the failed controller is not restarted automatically, but the sessions and loaded data survive. There is still a question mark because there is no further resilience to additional failures. For this reason, a SAS administrator should stop and restart CAS when possible, to go back to the original state. But, in, the immediate, SAS administrators don't have to do anything. In the best case, users don't have to do anything, as well: they will not even notice a failure. In worst cases, they might get error messages and have to wait a few minutes, then everything should automatically go back to normal.

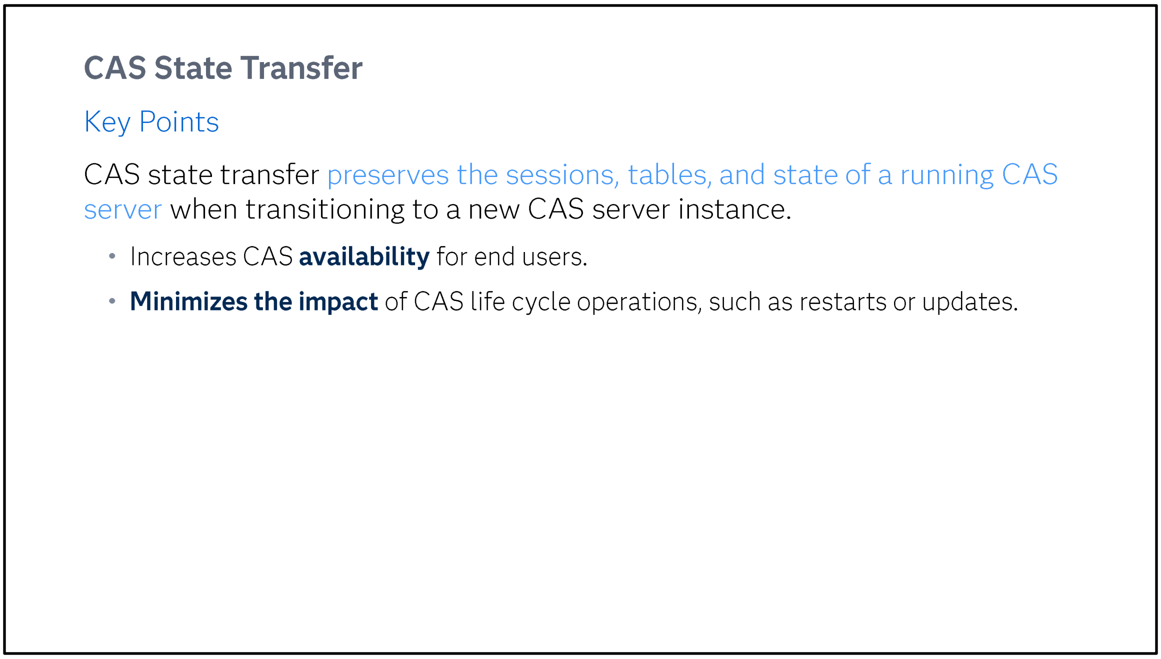
Finally, when a failure affects a worker node rather than a controller, notice here the footnote, we assume the default settings in which data is loaded with multiple copies, and no automatic re-distribution is configured.

In this case, the worker pod is auto-started, and loaded data survives, but again, we have an attention mark. By default, with no automatic data re-distribution, data will end up skewed across workers. In that case, a SAS administrator might want to re-distribute the tables; otherwise, no further action is required. Again, from an end-user perspective, in the best case nothing will happen. At worst, they might have to wait a bit, then the system will resume automatically its normal operations.



CAS Availability - State Transfer

The objective of the current module is to present the CAS State Transfer capability and discuss how it impacts the SAS Viya platform capability to provide Zero Downtime Rolling Updates.
The use case is to support planned updates, not to address unpredictable failures.



CAS State Transfer

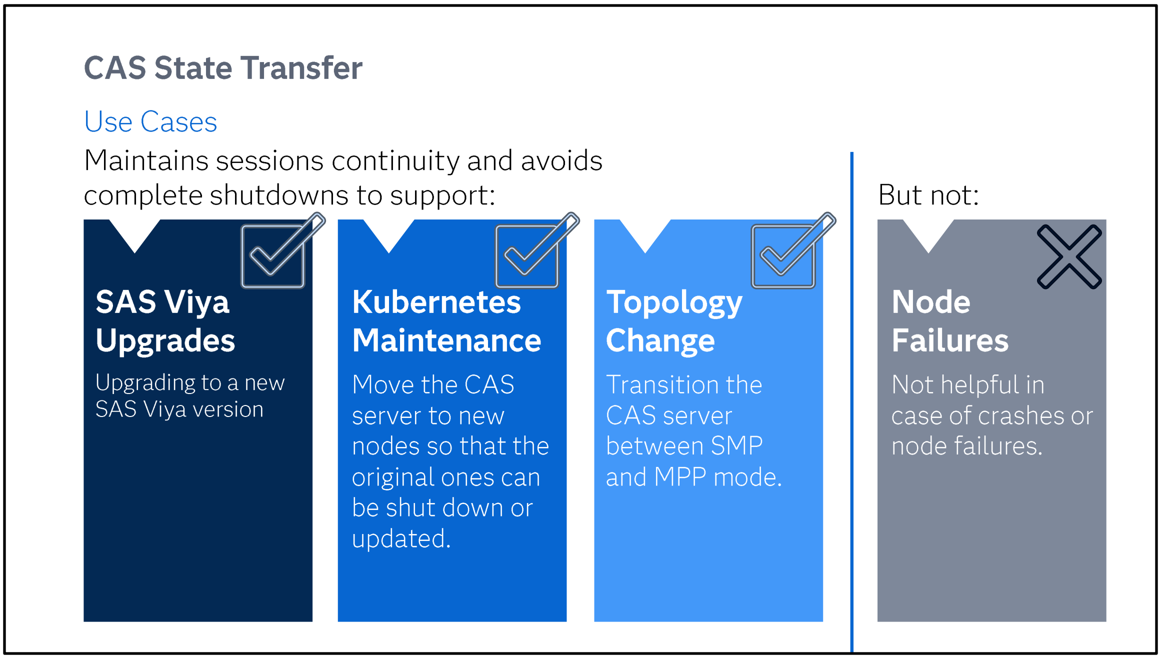
Key Points

CAS state transfer preserves the sessions, tables, and state of a running CAS server when transitioning to a new CAS server instance.

- Increases CAS **availability** for end users.
- **Minimizes the impact** of CAS life cycle operations, such as restarts or updates.

As of the LTS 2023.10 release of the SAS Viya platform, administrators can enable a feature called “CAS State Transfer” for CAS servers. This feature allows CAS servers to be restarted while preserving both CAS tables loaded in memory and active CAS sessions.
This capability increases the availability of CAS server and data for end users, by minimizing the impact of CAS life cycle operations, such as restarts or updates.

With the introduction of this CAS server feature, the SAS Viya platform administrator has a method to restart the CAS server without losing the in-memory CAS tables and active CAS sessions.



CAS State Transfer

Use Cases

Maintains sessions continuity and avoids complete shutdowns to support:

☑ **SAS Viya Upgrades**
Upgrading to a new SAS Viya version

☑ **Kubernetes Maintenance**
Move the CAS server to new nodes so that the original ones can be shut down or updated.

☑ **Topology Change**
Transition the CAS server between SMP and MPP mode.

But not:

☒ **Node Failures**
Not helpful in case of crashes or node failures.

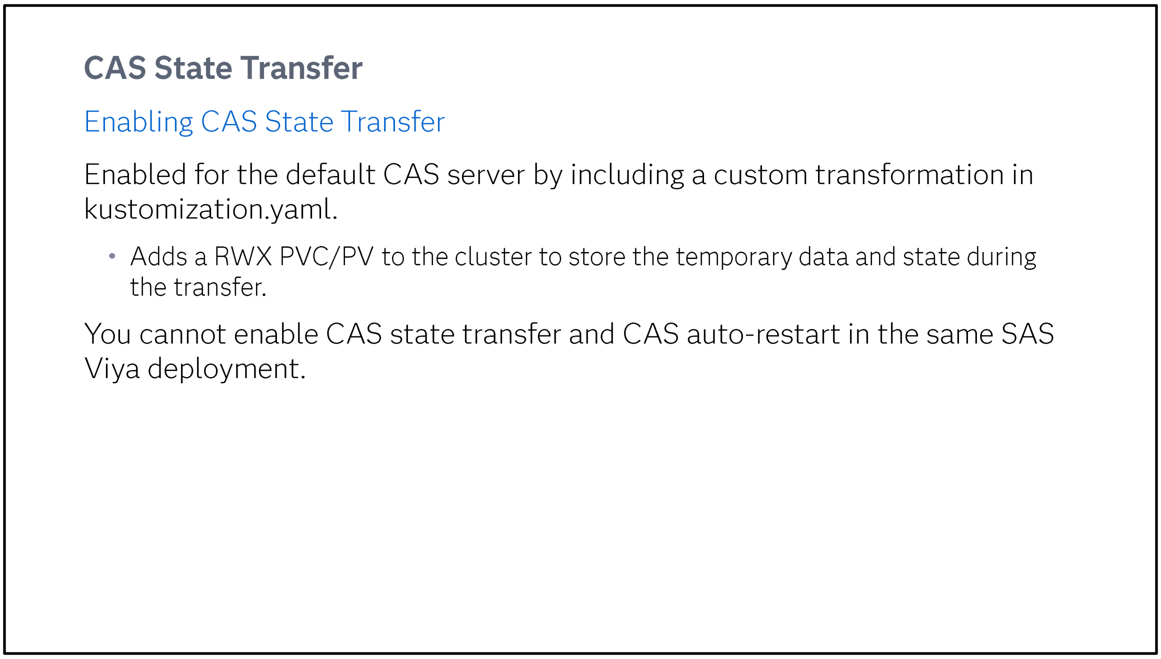
The CAS state transfer process can be used to maintain sessions continuity and avoid complete shutdowns to support multiple planned maintenance use cases.

You can use CAS state transfer when upgrading to a new SAS Viya version, or simply applying a patch.

Another use case is planned maintenance of the underlying nodes. A Kubernetes administrator can cordon a node, which means that new pods will not be started there. Then the SAS administrator starts the CAS state transfer, and the new CAS pods will be started on a new node. At the end of the transfer process, the Kubernetes administrator can drain the node, that is, have Kubernetes terminate and restart somewhere else any other pod that was eventually still running on the original node.

Finally, CAS state transfer can support topology changes, such as transitioning the CAS server between SMP and MPP mode.

While the CAS state transfer capability covers graceful restarts, it cannot help in case of software crashes or sudden node failures.



CAS State Transfer

Enabling CAS State Transfer

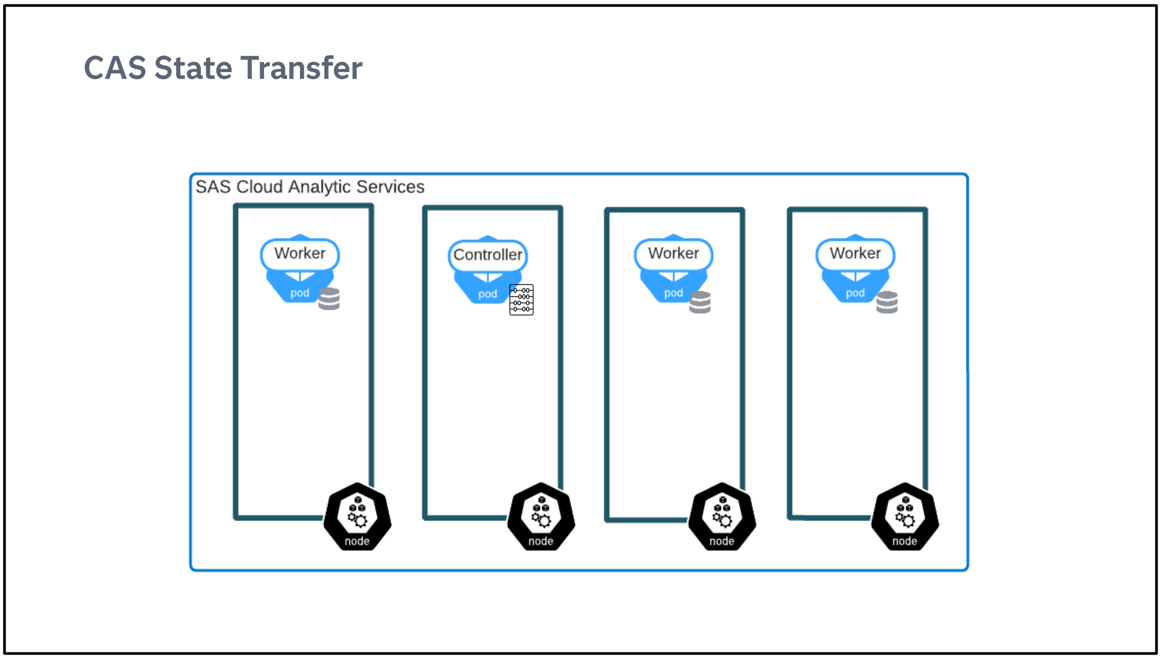
Enabled for the default CAS server by including a custom transformation in kustomization.yaml.

- Adds a RWX PVC/PV to the cluster to store the temporary data and state during the transfer.

You cannot enable CAS state transfer and CAS auto-restart in the same SAS Viya deployment.

The CAS state transfer process is disabled by default; it can be enabled by including a custom transformation in kustomization.yaml and then re-applying the SAS Viya deployment manifest. This creates a new RWX persistent volume claim and volume, to store the CAS data and state during the transfer.
If you plan to create additional CAS servers, the script that is used to create them accepts a flag to automatically enable the capability.

You cannot enable state transfer and CAS auto-restart in the same SAS Viya deployment. If you want to enable state transfer for a deployment that already has CAS auto-restart enabled, you must first disable CAS auto-restart before enabling state transfer.



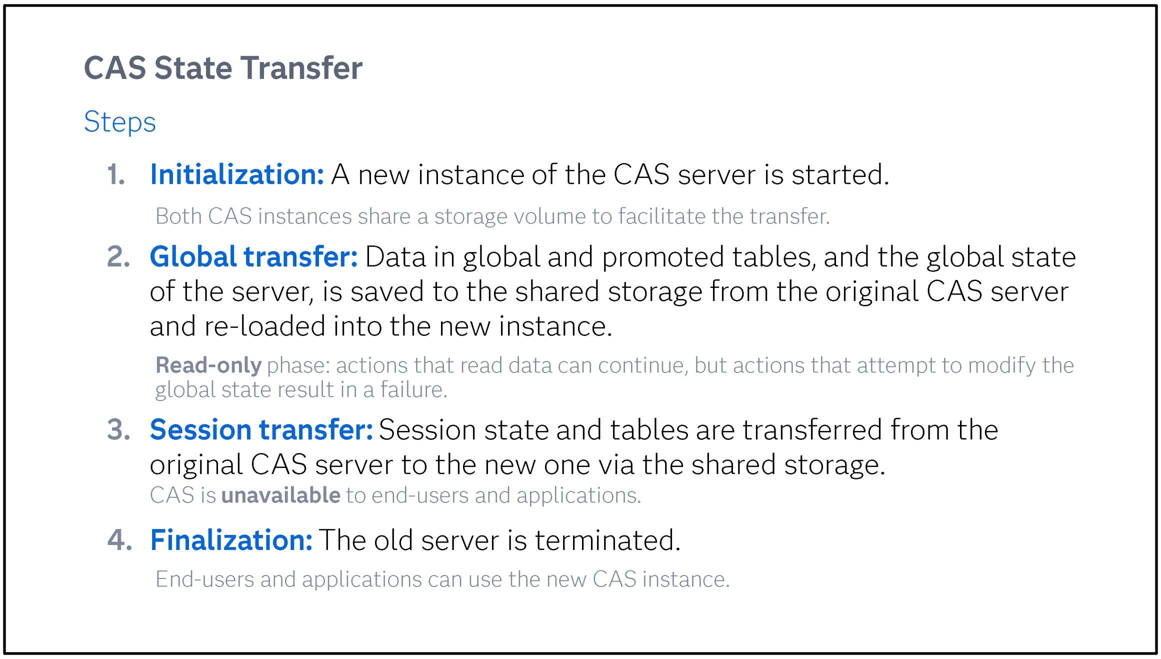
CAS State Transfer

This animation shows a sample CAS server deployed with a controller and 3 worker nodes. The SAS administrator wants to restart it, for example as part of a planned upgrade, minimizing user downtime. Initially, the CAS blue pods are running, with some data tables loaded in memory and the current state managed by the blue controller pod.

As the administrator initiates the process, a new CAS server comes online and is started in the green pods.
Note that, while the transfer is happening, the number of CAS pods running on each node doubles, and so will resource consumption.

Data is moved to and loaded in the new pods; the status is copied to the new green controller.

Finally, the original blue pods are terminated.



CAS State Transfer

Steps

1. **Initialization:** A new instance of the CAS server is started.

   Both CAS instances share a storage volume to facilitate the transfer.

2. **Global transfer:** Data in global and promoted tables, and the global state of the server, is saved to the shared storage from the original CAS server and re-loaded into the new instance.

   Read-only phase: actions that read data can continue, but actions that attempt to modify the global state result in a failure.

3. **Session transfer:** Session state and tables are transferred from the original CAS server to the new one via the shared storage.
   CAS is **unavailable** to end-users and applications.

4. **Finalization:** The old server is terminated.

   End-users and applications can use the new CAS instance.

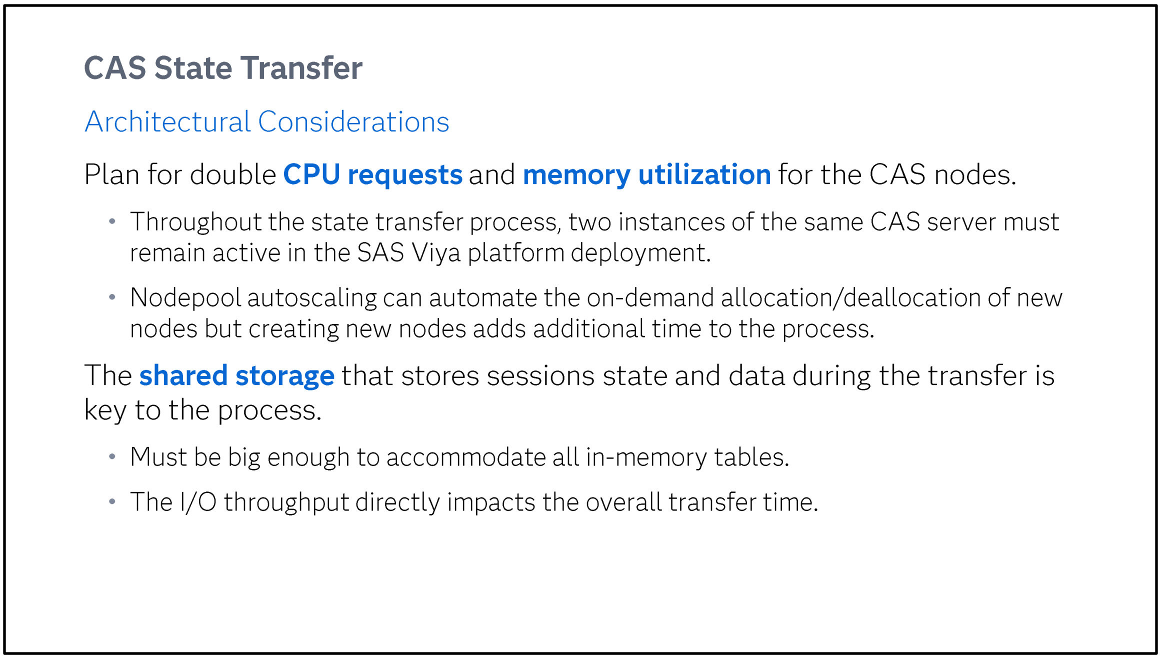
Let’s describe with a bit more detail what we’ve seen in the previous animation.

To initiate a state transfer operation, the SAS administrator patches the CAS server CASDeployment custom resource to set the startStateTransfer variable to true.
A new instance of the CAS server is started. Throughout the state transfer process, two instances of the same CAS server remain active in the SAS Viya platform environment. They share a storage volume to facilitate the transfer.
While the new instance is being initialized, the existing CAS server is fully functional.

As soon as the new CAS instance is ready, data in global and promoted tables, and the global state of the server, is saved to the shared storage from the original CAS server, then it is re-loaded into the new CAS instance.
In this phase, the global objects are marked as read-only. Actions that read data can continue, but actions that attempt to modify the global state result in a failure.

After global objects are transferred, it’s time for session objects. Session state and session tables are transferred from the old CAS server to the new one. In this phase, the CAS server becomes unavailable to the user community. It cannot process any action nor accept any new connection.

Once the CAS tables and sessions are transferred, the new CAS server instance becomes available for the user community. Subsequently, the initial instance of the CAS server is stopped.



CAS State Transfer

Architectural Considerations

Plan for double **CPU requests** and **memory utilization** for the CAS nodes.

- Throughout the state transfer process, two instances of the same CAS server must remain active in the SAS Viya platform deployment.
- Nodepool autoscaling can automate the on-demand allocation/deallocation of new nodes but creating new nodes adds additional time to the process.

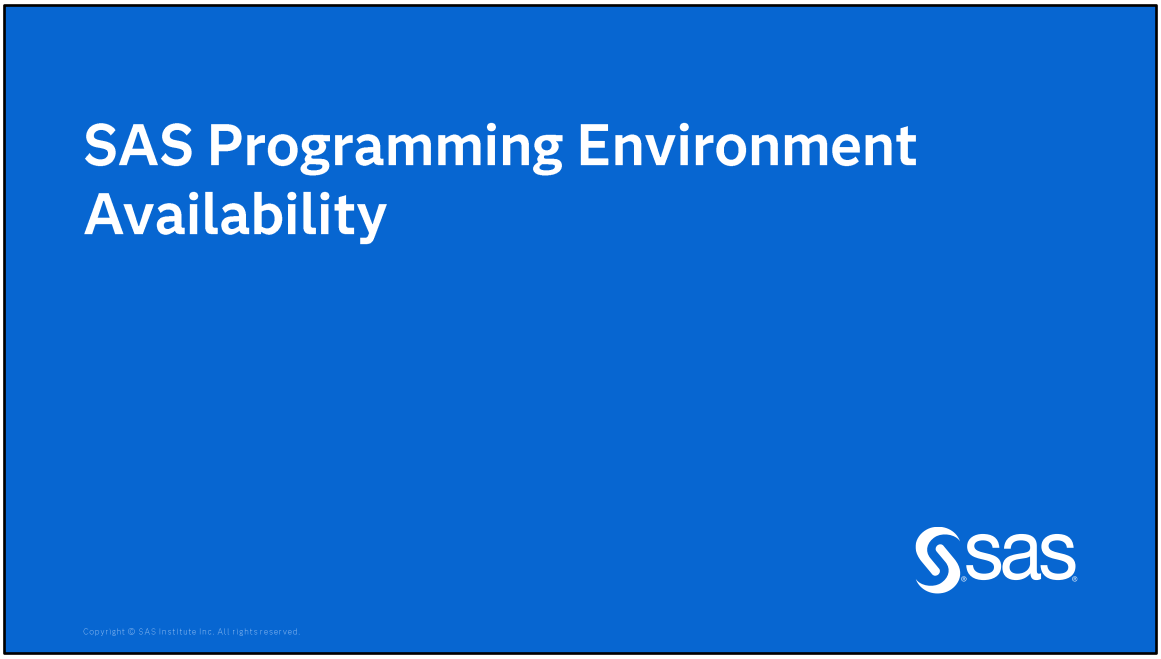
The **shared storage** that stores sessions state and data during the transfer is key to the process.

- Must be big enough to accommodate all in-memory tables.
- The I/O throughput directly impacts the overall transfer time.

As we have seen, utilizing the state transfer feature demands additional resources during the transfer process. This is because two instances of the CAS server must run simultaneously, and an extra volume is needed for the transfer operation. This has an impact on the SAS Viya platform operations costs. If the CAS nodepool is configured for autoscaling, then allocation and deallocation of new nodes is automated and can help controlling the cost but creating new nodes adds additional time to the process.

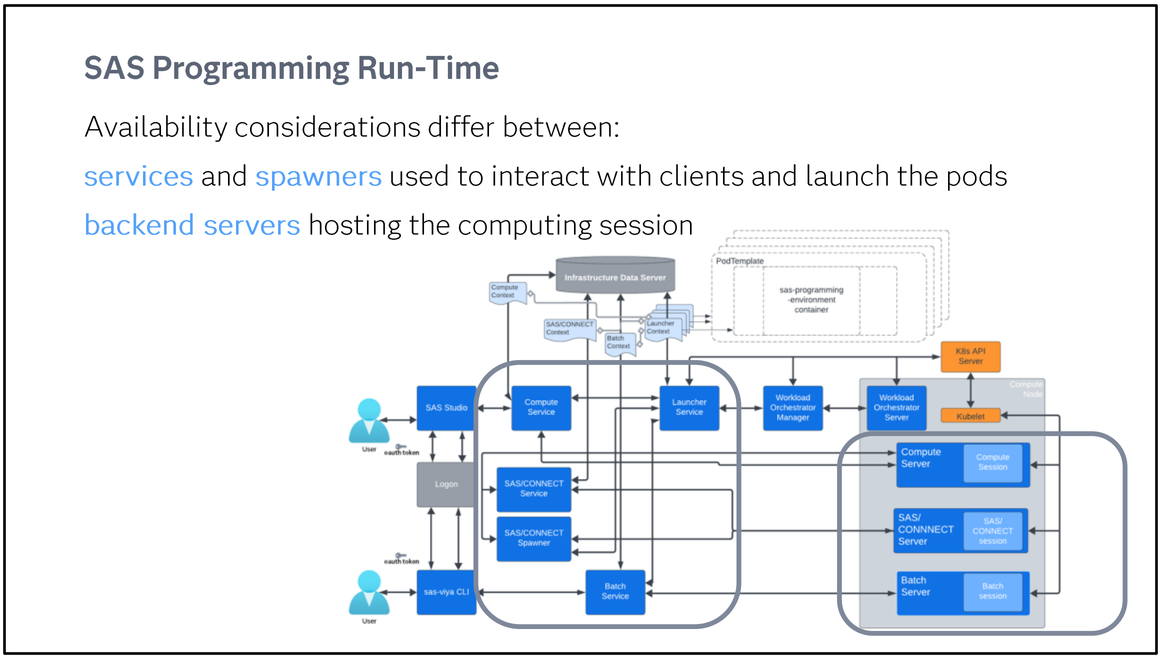
Similarly, proper design of the shared storage that stores CAS session and data during the transfer is key to the process. In-memory tables are written to the storage, which must be big enough to accommodate them. Plus, its I/O throughput directly impacts the overall transfer time. Since CAS is unavailable during the session transfer, faster storage can lower the total downtime, but it comes at an increased cost.
This requirement can be mitigated by using the MAXSESSIONTRANSFERSIZE option, which limits the combined table sizes for a single session that can be transferred to a new server. (And consequently the required time).

If the Kubernetes cluster lacks sufficient resources to support both instances of the CAS server, the process cannot be completed and will fail. If the new instance of the CAS server fails to start, the state transfer process aborts and the existing CAS server continues to be used.



SAS Programming Environment Availability

Availability of the SAS Programming Run-Time Environment.
Let’s review some availability considerations for SAS Compute, SAS Connect and SAS Batch components.



SAS Programming Run-Time

Availability considerations differ between:

services and spawners used to interact with clients and launch the pods

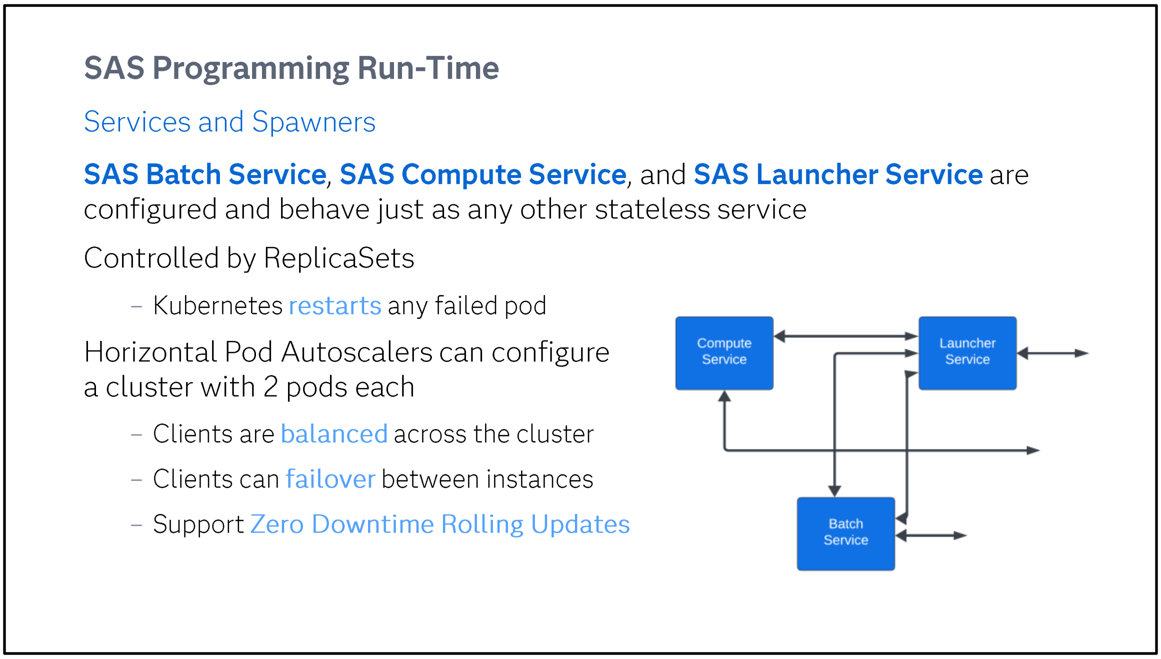
backend servers hosting the computing session

We have used this diagram that presents most components of the SAS programming runtime in other lessons.
It’s important to present it here to remember that the SAS programming runtime is made by so many different components, and it's important to cover all when thinking how something could be a point of failure.

When we discuss failures, or better, resilience to failures, we can recognize that the considerations are different when we talk about different platform levels.

Services and spawners, that is, the mid-tier components used to interact with the clients or launch the backend, have their own considerations.

Or are we talking about the failure of an actual compute session, a server session in the backend?



SAS Programming Run-Time

Services and Spawners

SAS Batch Service, **SAS Compute Service**, and **SAS Launcher Service** are configured and behave just as any other stateless service

Controlled by ReplicaSets

- Kubernetes restarts any failed pod

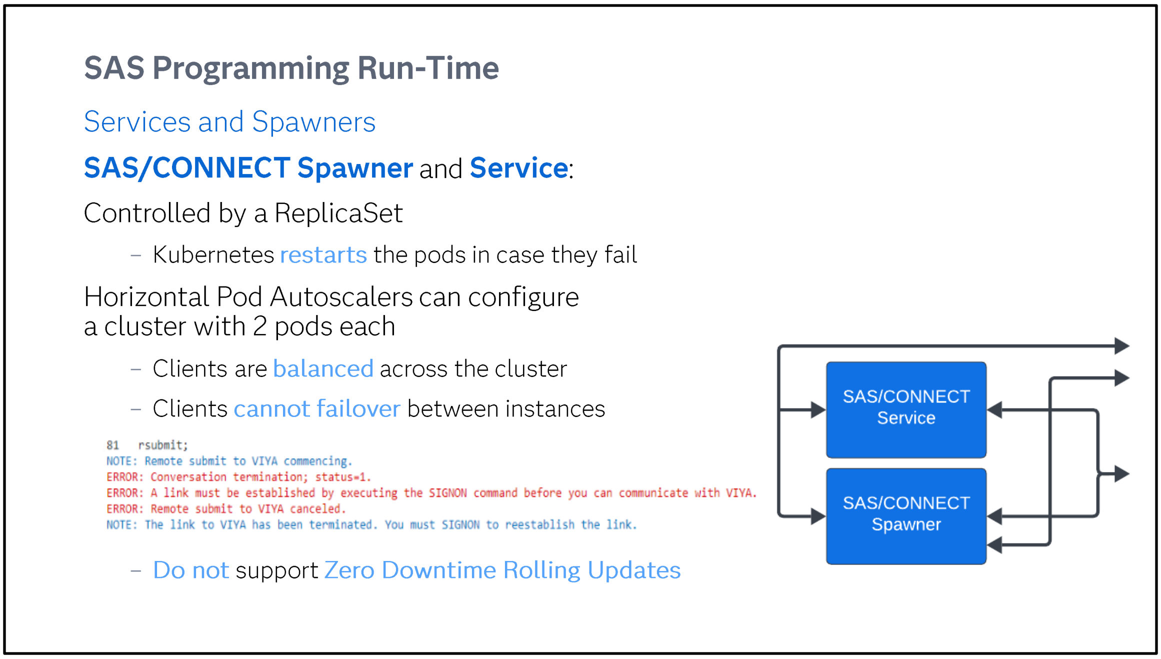
Horizontal Pod Autoscalers can configure a cluster with 2 pods each

- Clients are balanced across the cluster
- Clients can failover between instances
- Support Zero Downtime Rolling Updates

Let's start from the middle-tier services and spawners.
The batch service, the Compute service, and the Launcher service are all configured and behave just like any other stateless service.

This means that they are controlled by a Kubernetes replica set. Kubernetes takes care of restarting any failed pod.

In case you configure HA, then the horizontal pod autoscalers configure a cluster of two pods each.
For each service, clients are balanced across both pods.
In case of failures, Kubernetes automatically manages client failover between instances.
These services support zero downtime rolling updates. You can update the environment live.
With a cluster of at least 2 instances and a Pod Disruption Budget of 1, operations such as draining a Kubernetes node or performing a rolling update of the service do not interrupt service to the clients.



SAS Programming Run-Time

Services and Spawners

SAS/CONNECT Spawner and **Service**:

Controlled by a ReplicaSet

- Kubernetes restarts the pods in case they fail

Horizontal Pod Autoscalers can configure a cluster with 2 pods each

- Clients are balanced across the cluster
- Clients cannot failover between instances

```
81   rsubmit;
NOTE: Remote submit to VIYA commencing.
ERROR: Conversation termination; status=1.
ERROR: A link must be established by executing the SIGNON command before you can communicate with VIYA.
ERROR: Remote submit to VIYA canceled.
NOTE: The link to VIYA has been terminated. You must SIGNON to reestablish the link.
```

- Do not support Zero Downtime Rolling Updates

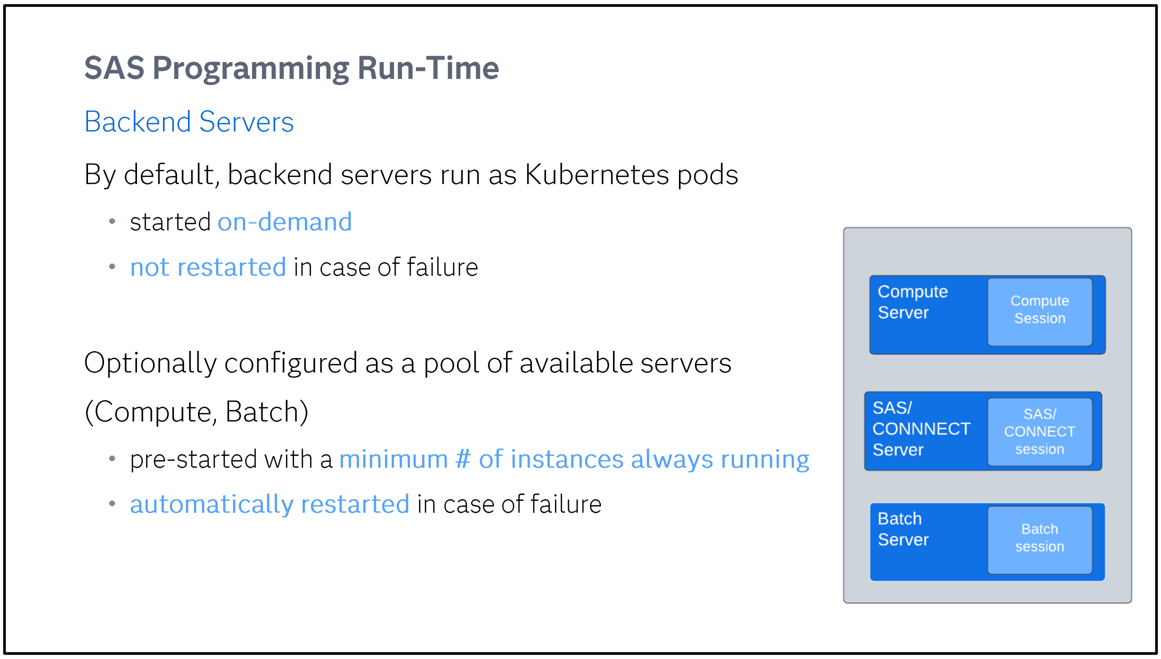
SAS Connect has two mid-tier components, the SAS Connect service, and the SAS Connect Spawner.

Just like the services we just discussed, they are controlled by a Kubernetes replica set, which takes care of restarting the pods in case of failure.

They can also be configured for HA with default Horizontal Pod Autoscalers, but their behavior here is a bit different.
Although client connections can be balanced between multiple mid-tier instances, they cannot survive failures.

With SAS Connect, if a mid-tier service fails, all backend servers managed by that instance are lost, and the connected clients will get an error similar to the one highlighted here.
In this case, the client will need to start a new connection which will instantiate a new backend.

For this reason, SAS Connect does not support Zero Downtime Rolling Updates.



SAS Programming Run-Time

Backend Servers

By default, backend servers run as Kubernetes pods

- started on-demand
- not restarted in case of failure

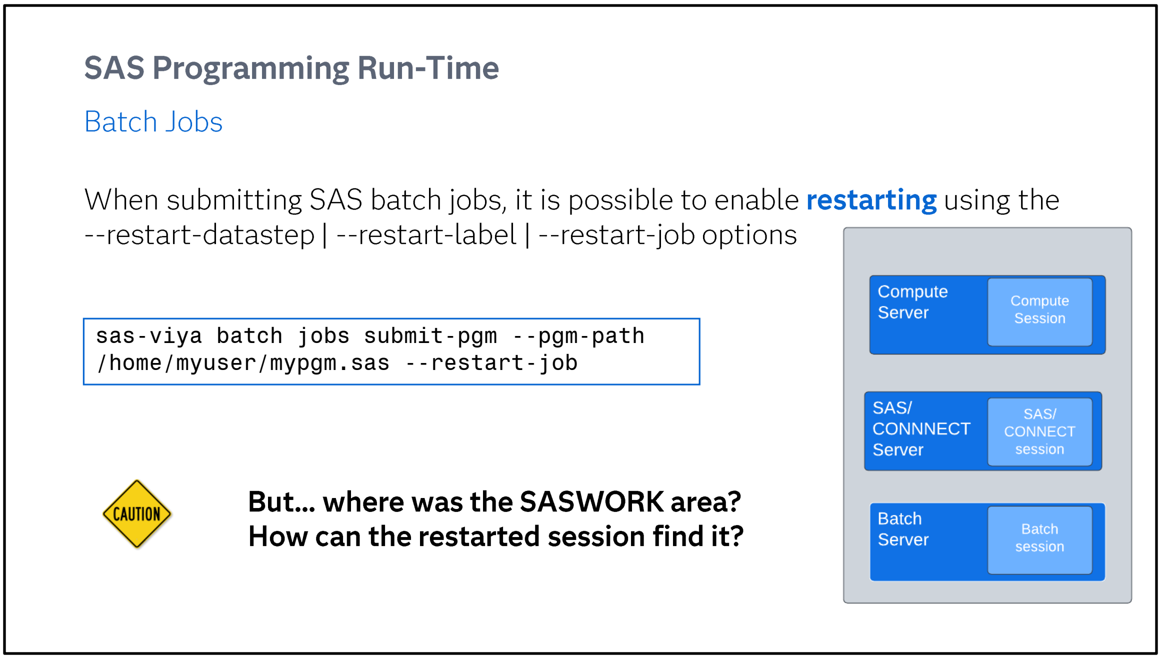
Optionally configured as a pool of available servers (Compute, Batch)

- pre-started with a minimum # of instances always running
- automatically restarted in case of failure

Compute Server
Compute Session
SAS/ CONNNECT Server
SAS/ CONNECT session
Batch Server
Batch session

Backend servers contain the engine that runs the SAS code.
They are usually started on demand, in which case, they are not restarted in case of failure.
It's up to the client application, or to the end user, to restart a failed backend server and to reload the data.
A partial exception to this rule happens during SAS Viya platform updates. If a failure happens during the programming run-time server initialization, SAS Workload Orchestrator and SAS Launcher can catch it and relaunch the affected jobs automatically.

Compute servers and Batch servers can be configured with a pool of available servers.
With this setting, the mid-tier services pre-start a given number of backend instances, monitor them, and take care of always keeping at least that minimum number up and running.
In this case, even when a failed backend server is automatically restarted, the running session is lost and, again, it’s up to the client to re-load data and re-submit the code.



SAS Programming Run-Time

Batch Jobs

When submitting SAS batch jobs, it is possible to enable **restarting** using the --restart-datastep | --restart-label | --restart-job options

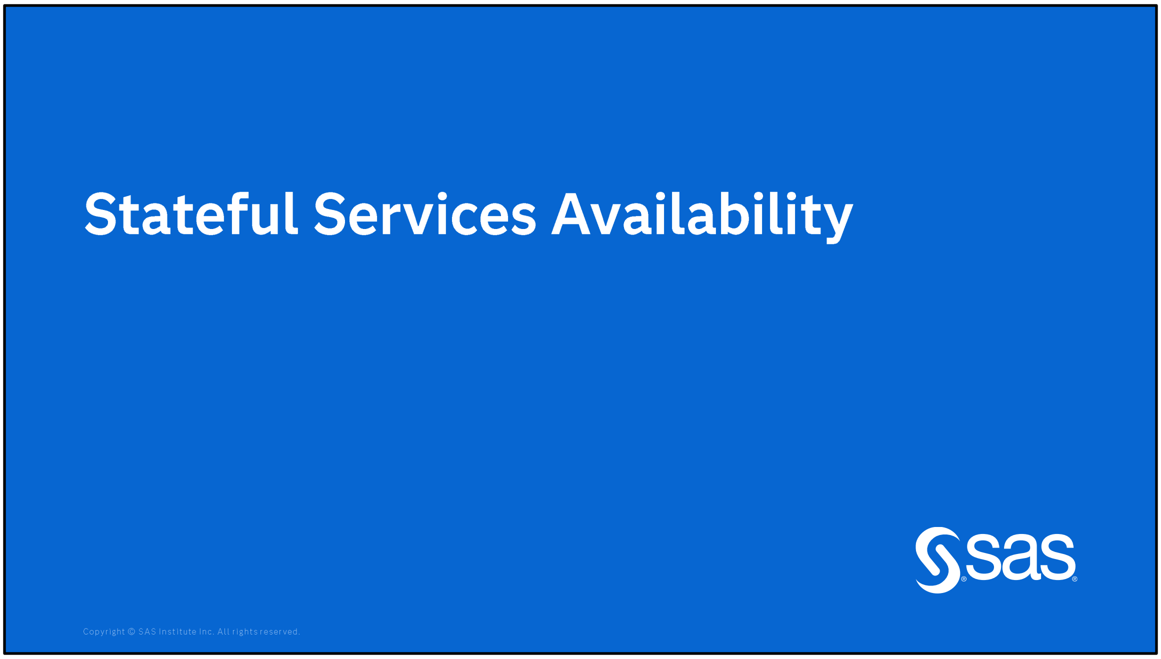
```
sas-viya batch jobs submit-pgm --pgm-path
/home/myuser/mypgm.sas --restart-job
```

But… where was the SASWORK area?
How can the restarted session find it?

CAUTION
Compute Server
Compute Session
SAS/ CONNNECT Server
SAS/ CONNECT session
Batch Server
Batch session

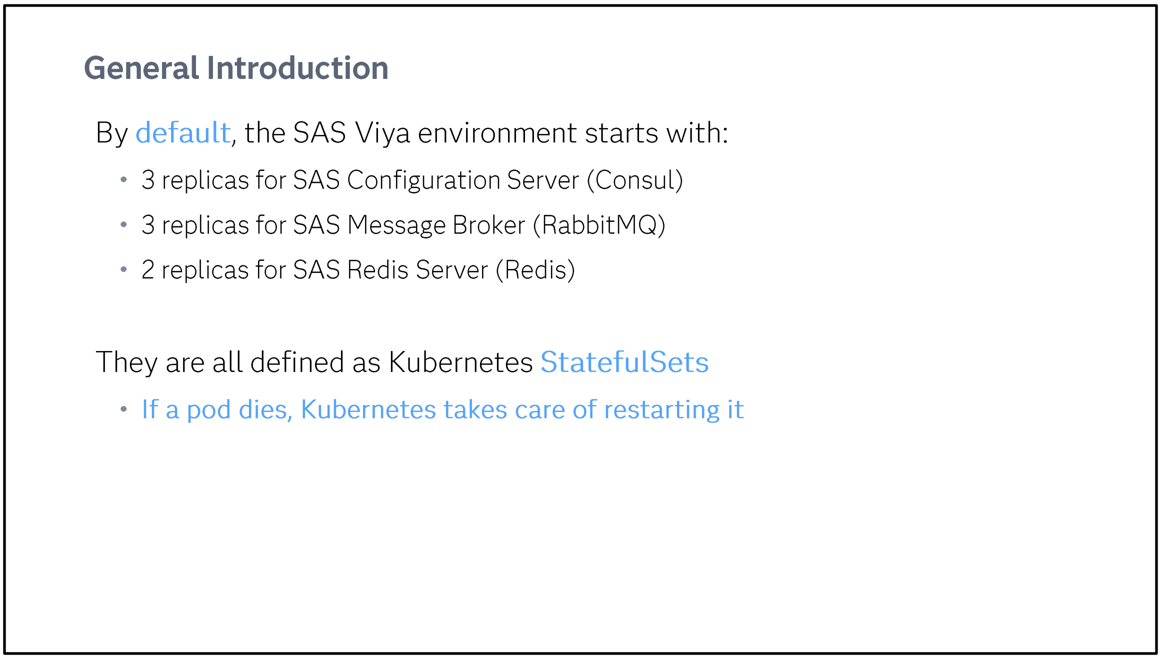
Batch servers support automatic restart of long-running jobs thanks to three “restart” command-line options.
In case of failures, the batch service will start a new backend server session and re-submit the previous job. Depending on the command-line option, the job will either be restarted from the beginning (restart-job), or from a data step checkpoint (restart-datastep) or from a label included in the SAS program (restart-label).

It is important to understand that, in all these cases, the new batch server pod might be restarted on a different node. When re-starting from the beginning, the job will re-load or re-calculate its data. But when re-starting from a checkpoint or a label, the job expects to run with an up-to-date status as it was before the failure. If it was working on temporary data, that was probably stored in the SASWORK area of the session that failed. After the job restarts, possibly on a different machine, can it find this data? This is only possible if the environment has been properly architected, for example, by locating the SASWORK area on shared storage.



Stateful Services Availability

Let’s review the main availability considerations for some of the SAS Viya stateful services: the SAS Configuration Server, Consul, the SAS Message Broker, RabbitMQ, and the SAS Redis Server.



General Introduction

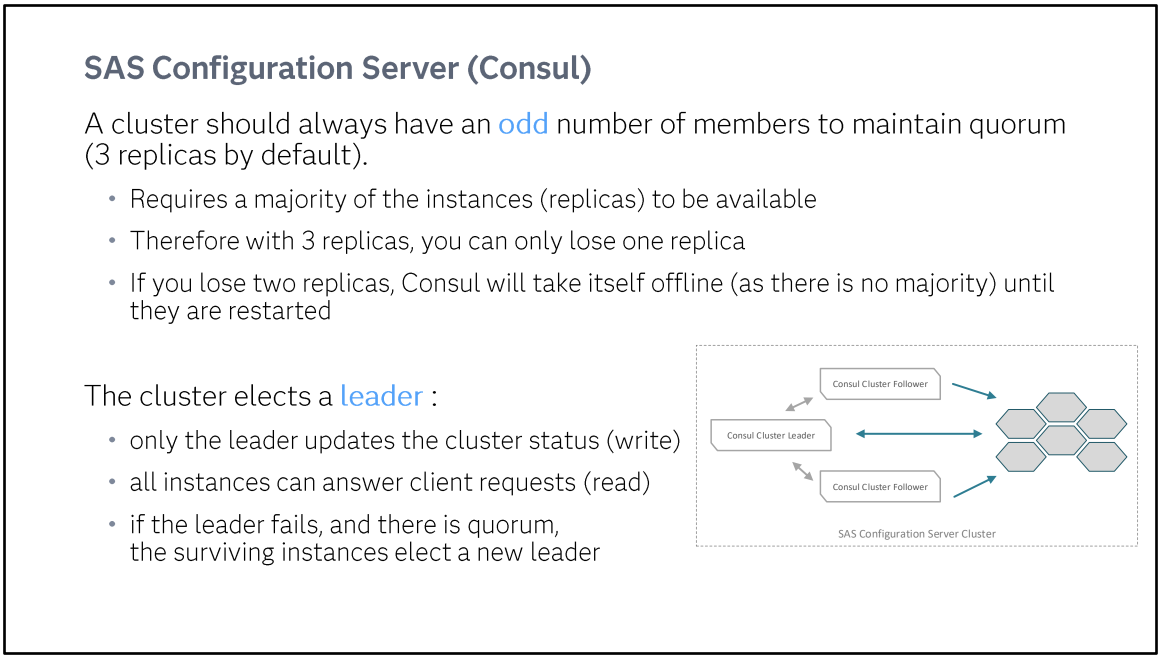
By default, the SAS Viya environment starts with:

- 3 replicas for SAS Configuration Server (Consul)
- 3 replicas for SAS Message Broker (RabbitMQ)
- 2 replicas for SAS Redis Server (Redis)

They are all defined as Kubernetes StatefulSets

- If a pod dies, Kubernetes takes care of restarting it

By default, SAS Viya environments are installed with these services already configured for high availability.
In detail, there are three replicas for the SAS Configuration Server, three replicas for the SAS Message Broker and two replicas for the SAS Redis Server.
They are all defined as Kubernetes stateful sets, so if a pod dies Kubernetes takes care of restarting each individual pod.



SAS Configuration Server (Consul)

A cluster should always have an odd number of members to maintain quorum (3 replicas by default).

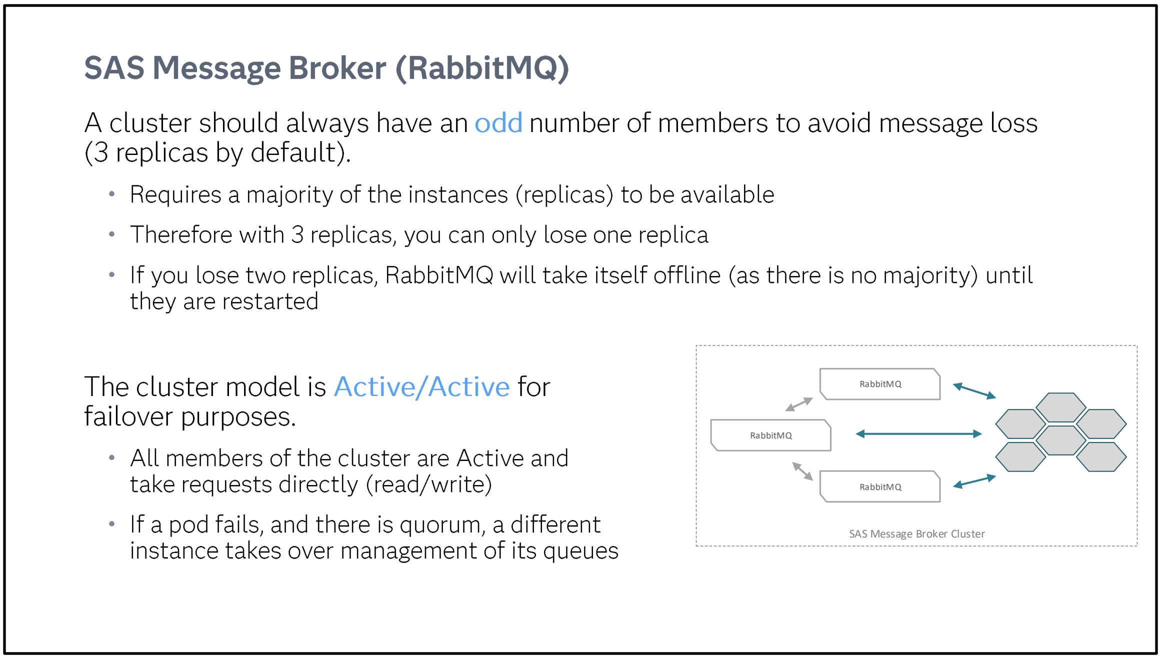
- Requires a majority of the instances (replicas) to be available
- Therefore with 3 replicas, you can only lose one replica
- If you lose two replicas, Consul will take itself offline (as there is no majority) until they are restarted

The cluster elects a leader :

- only the leader updates the cluster status (write)
- all instances can answer client requests (read)
- if the leader fails, and there is quorum, the surviving instances elect a new leader

Let’s see some details about the SAS Configuration Server, based on HashiCorp Consul.
It is deployed by default as a cluster with 3 server instances. It’s important that the cluster always starts with an odd number of replicas, so that they can form and maintain a quorum.
Having 3 servers enables SAS Configuration Server to maintain a quorum if one server fails. To achieve more redundancy, moving to 5 replicas enables SAS Configuration Server to operate with 2 failed servers. Having 3 or 5 servers is recommended to balance redundancy and performance.
With three replicas, if you lose one, you still have a functioning cluster. If two instances go down, then you lost the quorum and the whole cluster takes itself offline. As soon as Kubernetes restarts the pod, the pods rejoin the cluster, and they re-establish a quorum.

Having a quorum is important for the members of the cluster to elect a leader. Only the leader can update the cluster status (that is, perform write operations). All instances can answer client queries, that is, answer read requests.
If the leader fails while there is still a quorum, the surviving instances elect a new peer to become the new leader.



SAS Message Broker (RabbitMQ)

A cluster should always have an odd number of members to avoid message loss (3 replicas by default).

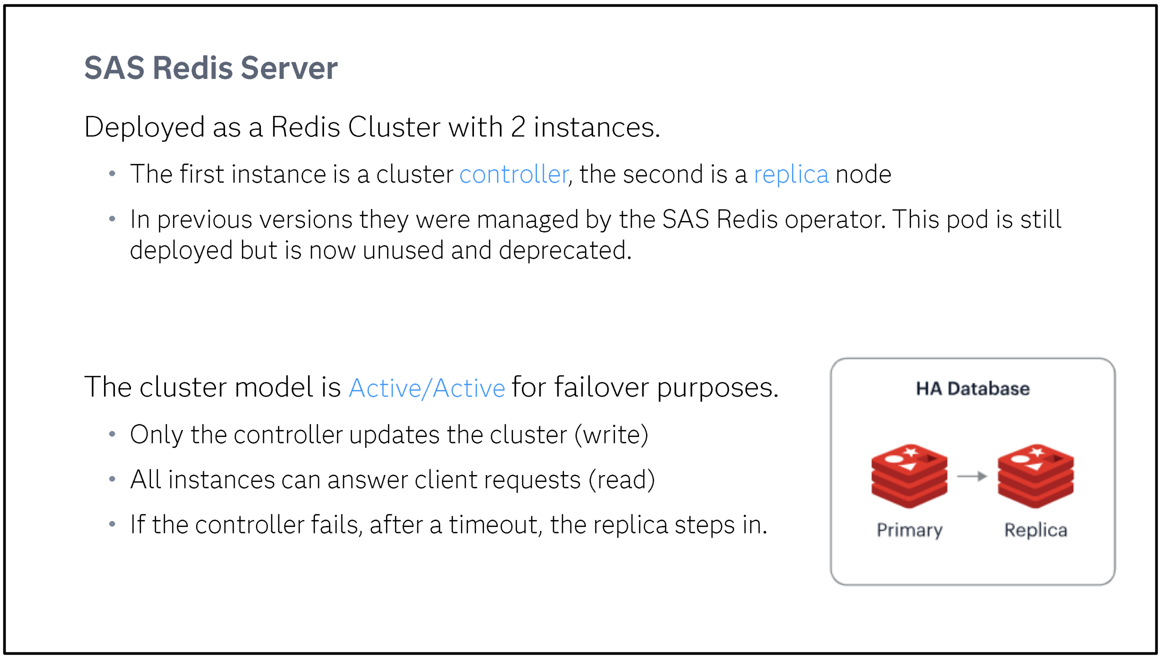
- Requires a majority of the instances (replicas) to be available
- Therefore with 3 replicas, you can only lose one replica
- If you lose two replicas, RabbitMQ will take itself offline (as there is no majority) until they are restarted

The cluster model is Active/Active for failover purposes.

- All members of the cluster are Active and take requests directly (read/write)
- If a pod fails, and there is quorum, a different instance takes over management of its queues

Also the SAS Message Broker, based on RabbitMQ, is deployed by default as a cluster with 3 server instances.
It is recommended to always run an odd number of RabbitMQ nodes in order to properly achieve a quorum. Several features, such as, quorum queues, and client tracking, require a consensus between cluster members, which can be achieved with an odd number of nodes.
Again, maintaining a quorum requires the majority of the instances to be available and therefore, with three replicas, you can only lose one instance. If you lose two nodes, RabbitMQ goes offline.

The cluster works in a different way than Consul.
For failover purposes, the cluster model is active/active. All members are active, and they all answer to both read and write requests. The details are a bit more nuanced, because each member is in charge for some of the queues and not for all of them.
Other cluster members forward requests for a specific queue to its manager instance, and that manager maintains and mirrors state information and content back to the other cluster members.
If one member of the cluster fails, then the leadership role, for the queues that it was managing, gets transferred to a surviving member.



SAS Redis Server

Deployed as a Redis Cluster with 2 instances.

- The first instance is a cluster controller, the second is a replica node
- In previous versions they were managed by the SAS Redis operator. This pod is still deployed but is now unused and deprecated.

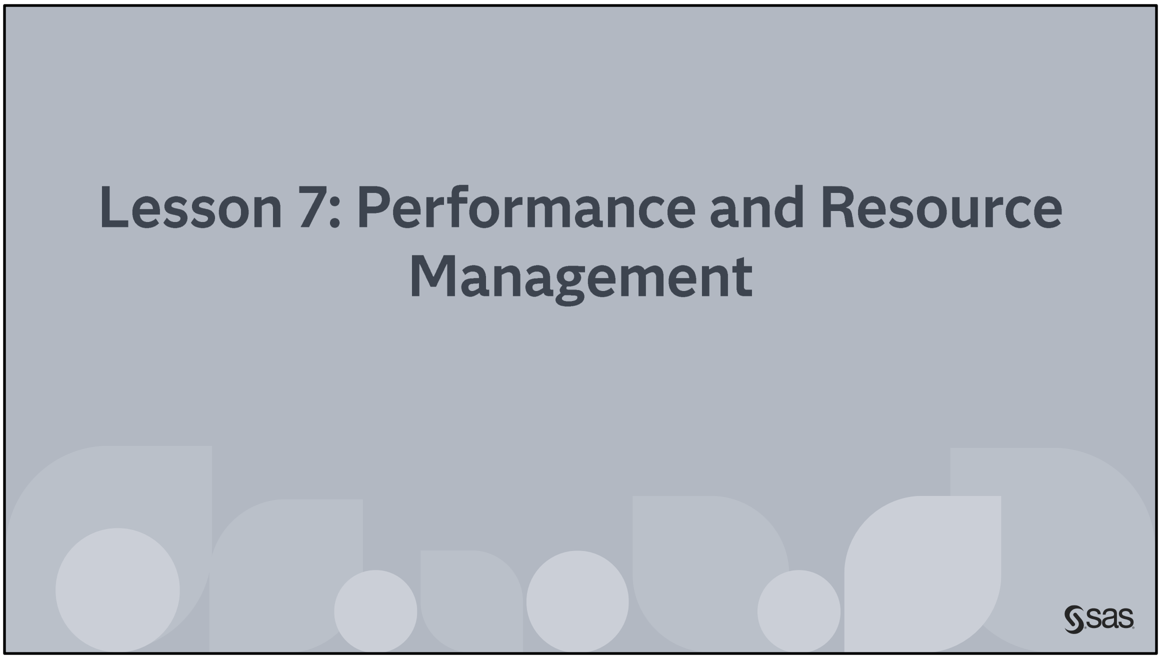
The cluster model is Active/Active for failover purposes.

- Only the controller updates the cluster (write)
- All instances can answer client requests (read)
- If the controller fails, after a timeout, the replica steps in.

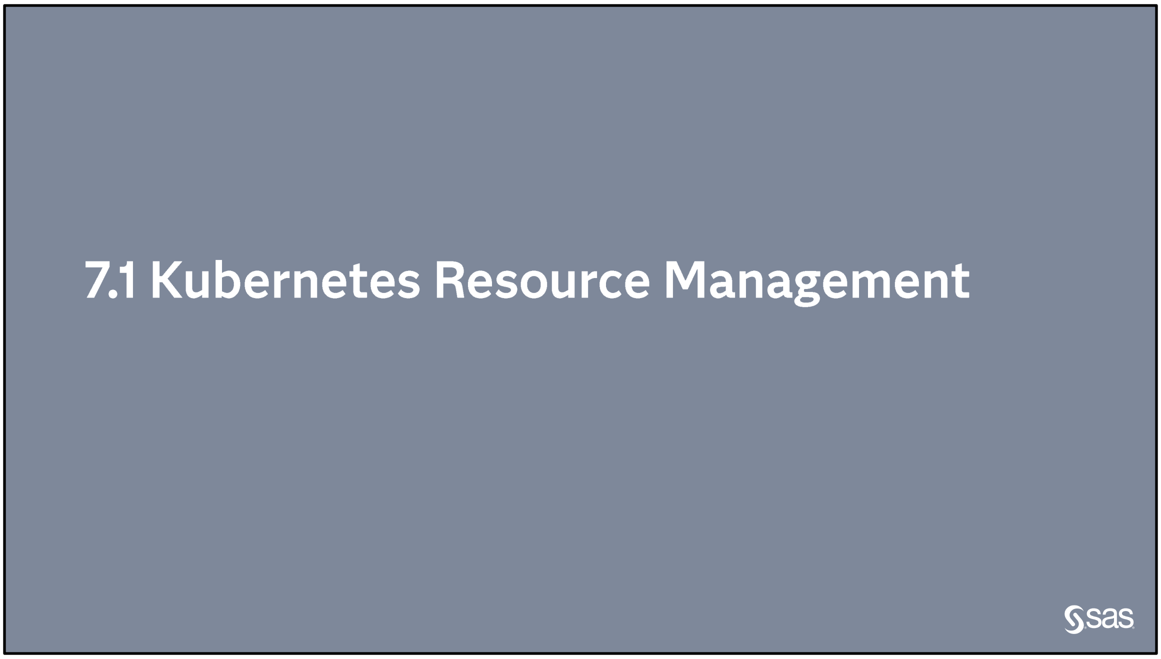
The SAS Redis server is deployed as a simple Redis cluster, composed by 2 instances deployed as a Kubernetes stateful set.
The first instance starts as a cluster controller, the second is a replica node.

In previous releases of the SAS Viya platform, they were managed by the SAS Redis operator. This pod is still deployed but is now unused and deprecated.

For failover purposes, both instances are always up and running and can answer client read requests, so the cluster model is Active/Active.
The controller is the only node accepting write requests and updates the replica asynchronously.
If the controller fails, then the replica waits for a few seconds, before stepping in, and becoming active in its place.



Lesson 7: Performance and Resource Management



7.1 Kubernetes Resource Management

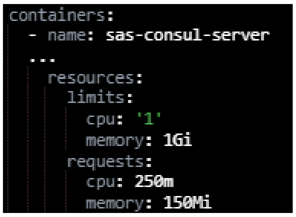
Container Resources

Container Resources

Kubernetes relies on container’s resource specifications to manage the cluster’s resources.

Resource requests and limits can be set for each container of the pod.

- Most common resource types are :
  - CPU
  - Memory
- *Other resource types (“Ephemeral storage” and “Huge pages”) exist but are not used very often (for example CAS).*



```
containers:
  - name: sas-consul-server
  ...
    resources:
      limits:
        cpu: '1'
        memory: 1Gi
      requests:
        cpu: 250m
        memory: 150Mi
```

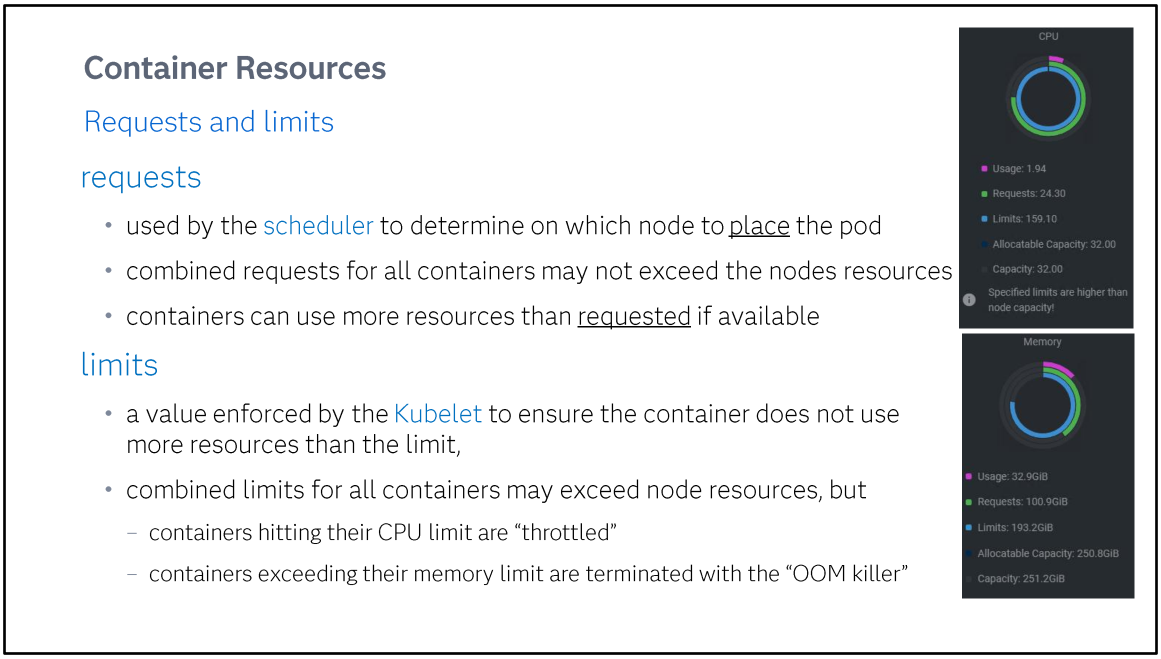
- The pod’s resources request is the sum of all the container requests.
- The pod’s resources request is evaluated by Kubernetes to decide if the pod can be scheduled and on which node

The Kubernetes scheduler evaluates various conditions before making the decision to schedule the pods in the cluster.

Kubernetes relies on containers resource specifications to manage the Kubernetes resources.
Depending on the sum of the defined resource requests of all the containers of the pod, Kubernetes decides if the pod can be scheduled and on which node.
It looks if there is enough resources available on a Kubernetes nodes before placing and starting the pod there.
That's basically how it works.
Of course, there are a lot of other factors that come into the final decision to schedule a pod somewhere. (such as tolerations and taints and node affinities and so on).
But the containers resource requests are also a key part of this decision.

Resource request and limits can be set for each container, inside the pod definition and we have an example on the right-hand side.
The most common resource types are CPU and memory.
There are other resource types like ephemeral storage or huge pages, but they are rarely used.
For example, in SAS Viya, ephemeral-storage request are used only for the CAS containers.

A pod typically contains one container, but it can have more. To determine the resource request for the pod, you must sum the requests for all containers. This total represents the pod's resource request, which Kubernetes uses to decide whether the pod can be scheduled and on which node, provided there is at least one node with enough available resources.



Container Resources

Requests and limits

requests

- used by the scheduler to determine on which node to place the pod
- combined requests for all containers may not exceed the nodes resources
- containers can use more resources than requested if available

limits

- a value enforced by the Kubelet to ensure the container does not use more resources than the limit,
- combined limits for all containers may exceed node resources, but
  - containers hitting their CPU limit are "throttled"
  - containers exceeding their memory limit are terminated with the "OOM killer"

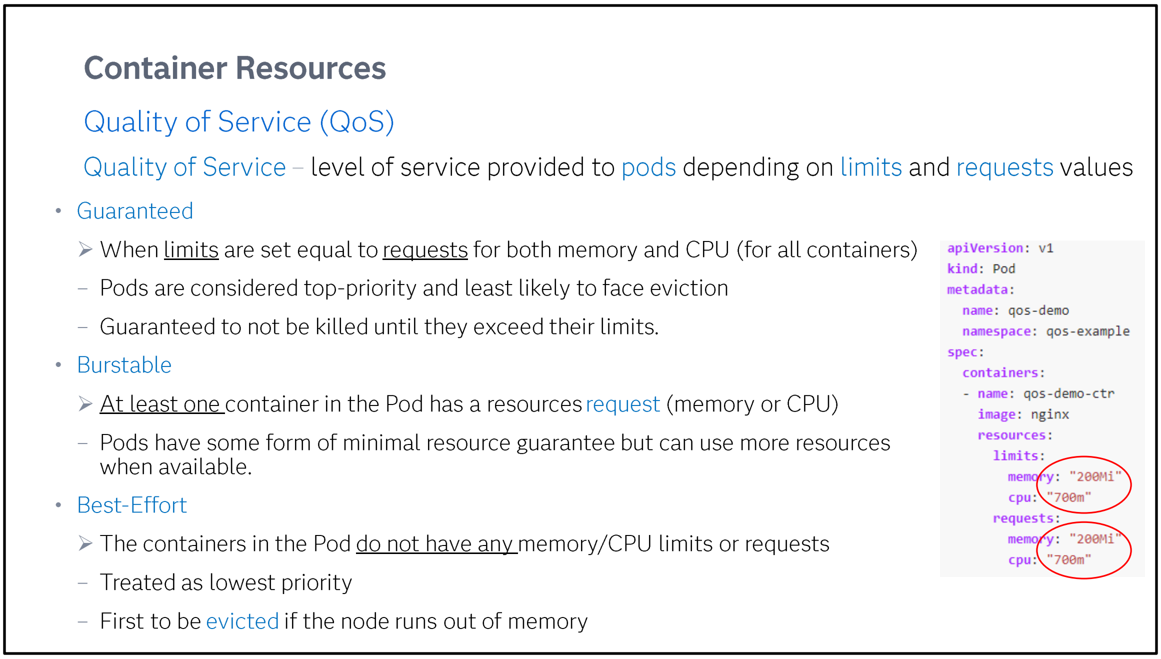
So what are these "request" and "limits" settings ?

As explained before , the request value is used to determine on which node to place the pod
Combined requests for all the containers may not exceed nodes resources.
It means that, if you make the sum of all the container requests of all your pods into the cluster, it could not be higher than the available resource inside the Kubernetes cluster.
For example, on the right-hand side here, there is a view from OpenLens and it tells you that the sum of all the containers CPU requests is 24.3 and the cluster CPU capacity is 32 cores.
So that's fine. BUT if we had a total of CPU requests in all the pod's definitions higher than 32 cores then, we would see some pods remain in the pending state.
Once scheduled on a node, the containers can consume more resources than requested if available.
Each container can have a given amount for the request but then the container can use more than this if needed.
The request is just what it needs to start, but after that, it could potentially use more resources than what has been initially requested.

Now, regarding the limits,

their value is used by the Kubelet running on each node to ensure the container is not using too much resources.
As a reminder, the kubelet is a little agent that is running on each Kubernetes node and the kubelet will make sure that the container does not consume more resource than the defined limit.
If you make the sum of all the limits of all the containers defined in all the pods, it can exceed the node's resource (as you can see on the Lens screenshot for the CPU limits).
But what the kubelet will do is that : if an individual container is exceeding the defined CPU limit, then the

containers’s process will be throttled.
And for the memory, if the individual container exceed its memory limit, then it could be terminated by the Linux Out Of Memory killer.
As you see, requests and limits are two things that are used by Kubernetes to manage the workload, but in a different way.



Container Resources

Quality of Service (QoS)

Quality of Service – level of service provided to pods depending on limits and requests values

- Guaranteed
  - ➢ When limits are set equal to requests for both memory and CPU (for all containers)
  - – Pods are considered top-priority and least likely to face eviction
  - – Guaranteed to not be killed until they exceed their limits.
- Burstable
  - ➢ At least one container in the Pod has a resources request (memory or CPU)
  - – Pods have some form of minimal resource guarantee but can use more resources when available.
- Best-Effort
  - ➢ The containers in the Pod do not have any memory/CPU limits or requests
  - – Treated as lowest priority
  - – First to be evicted if the node runs out of memory

```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "700m"
      requests:
        memory: "200Mi"
        cpu: "700m"
```

The quality of service is a level of service provided to pods.
We will see that this Quality of service is based directly on the resource definitions for the containers inside the pod.
Depending on the quality of service, there will be different behavior or different way for kubernetes to manage the pods.
There are 3 different “Quality of Service” levels, depending on resource limits and resource requests values

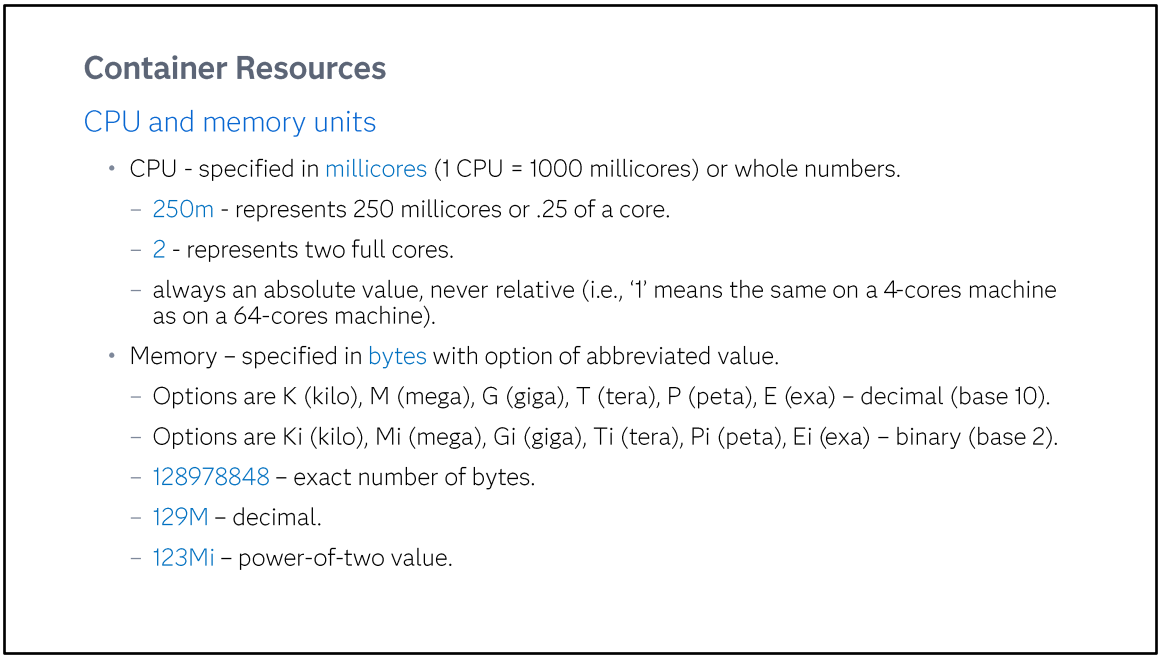
Let’s start with the Guaranteed Q O S or Guaranteed Quality Of Service, the pods are in this class only when the limit and requests are set to the same value for all the containers inside the pod.

You can see here an example on the right side where, for the memory, the limit and the request have the same value, 200 megabytes.
It must be the same thing for the CPU. CPU limit is 700 millicores and CPU request is also 700 millicores. So, it's the same.
For both resources types (memory and CPU), we have the same value.
So, if all the containers inside the pod follows the same rule, then your pod will be automatically categorized in the guaranteed QS class.
Now, when the pod is in guaranteed QS, it means that they are considered as top priority and will be the last ones to be evicted in case there is a shortage of resources on the node.
Using this guaranteed Q O S is a way to add additional guarantees that those pods would only be evicted at last resort.

The second QS class is “burstable”. It’s when at least one container in the pod has a resource request that is defined (either memory or CPU).

In this case the pods have some form of minimal resource guarantee, but they can use more resource than when available.
So “burstable” is the right name for them : there is like a request, but there's no limit defined for them.

The last category is “best efforts”. It's when the containers in the pod do not have any memory or CPU limits or requests specified.
The pods in this category will be treated as lowest priority and they will be the first to be evicted if the node runs out of memory.



Container Resources

CPU and memory units

- CPU - specified in millicores (1 CPU = 1000 millicores) or whole numbers.
  - 250m - represents 250 millicores or .25 of a core.
  - 2 - represents two full cores.
  - always an absolute value, never relative (i.e., ‘1’ means the same on a 4-cores machine as on a 64-cores machine).
- Memory – specified in bytes with option of abbreviated value.
  - Options are K (kilo), M (mega), G (giga), T (tera), P (peta), E (exa) – decimal (base 10).
  - Options are Ki (kilo), Mi (mega), Gi (giga), Ti (tera), Pi (peta), Ei (exa) – binary (base 2).
  - 128978848 – exact number of bytes.
  - 129M – decimal.
  - 123Mi – power-of-two value.

How do we express the CPU and memory resource requests and limits?

CPU power is expressed either in number of cores or in millicores, one core is equal to 1000 Millicores.
You represent millicores with a little M, like 250 M, which present 250 millicores or a quarter of a core.
You can also use an integer value like two, which means two full cores.
And it will always be an absolute value.
It's not something that is related to the node capacity: one or two millicores will always represent the same value no
matter what the node capacity is or no matter how many cores you have on the machine.

On the other hand, Memory is specified in bytes.
We have the habit of working with this kind of unit. So, kilobytes, megabytes, gigabytes, terabytes, et cetera.
You can give either the exact number of bytes or you can use one of the unit abbreviated value.
Finally, the “base two” notation is also available but it's roughly the same as the decimal notation.

Container Resources

SAS Viya container requests and limits example

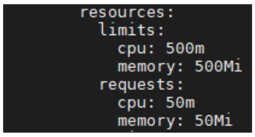
Example pulled from the SAS Viya manifest

- requests
  - 50m – 50 m lli-cores ~ 5 50) of a ore
  - 50Mi – 50 me
- limits
  - 500m - 0.
  - 500

QUIZ :what is the QoS class for the pod holding this container ?

Representative of SAS Viya microservices specs

sas-audit

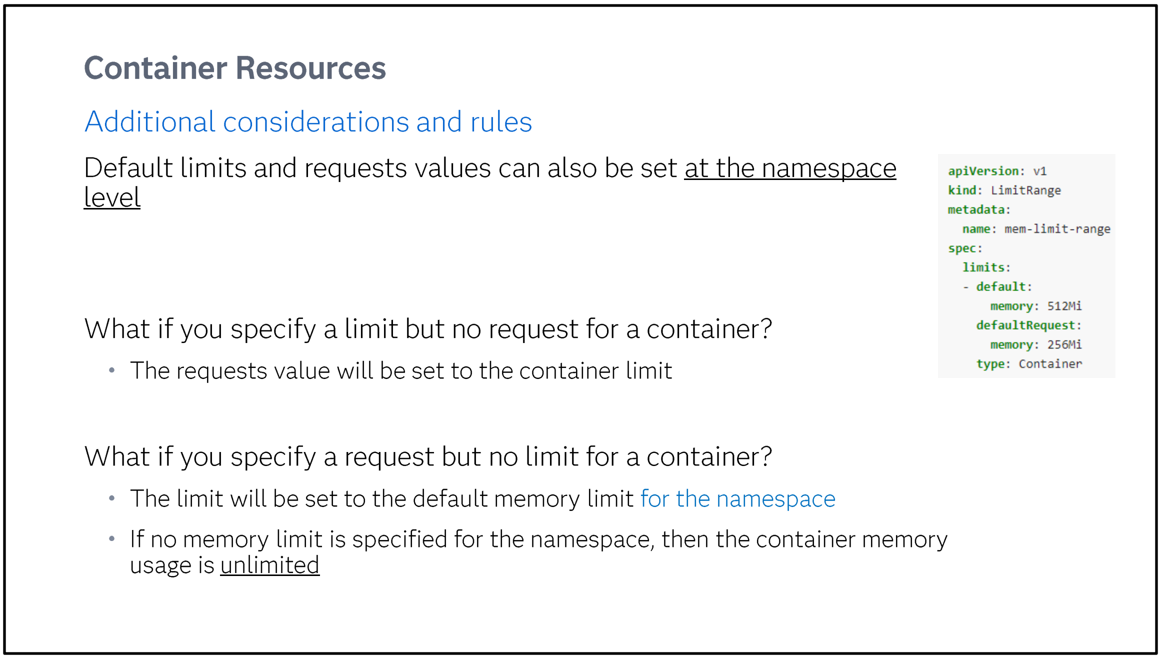


```
resources:
  limits:
    cpu: 500m
    memory: 500Mi
  requests:
    cpu: 50m
    memory: 50Mi
```

So here's a little example with a SAS Viya manifest.
These kind of resource limits requests correspond to the default resources settings for many lightweight microservice containers like sas-arke, sas-activities, sas-audit, sas-batch, etc… .
You can see that the request is 50 millicores, which is equivalent of 5% of a core and 50 megabytes.
It's pretty low, so normally Kubernetes should be able find a place to start this pods quite easily.
Then you have the limit for it.
The limit is half of a core and 500 megabytes of memory.
It means that IF this specific container, inside the pod, starts to use more than half a core then its process will
be throttled. Now if it uses more than 500 megabytes of memory, the container could be killed (by the Out Of Memory killer).
Again, as noted here, this kind of specifications that you see on the screenshot is pretty representative of the SAS Viya microservices. So, most of the microservices containers inside the stateless pods, will have those kind of resource limits and requests.

Now here is a little quizz for you : what would be the QoS level for the pod holding this container ?
OK let’s wait for a few seconds, to let you think about it…

The Answer is “Burstable” - because the request and limits are set but they have different values.



Container Resources

Additional considerations and rules

Default limits and requests values can also be set at the namespace level

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
```

What if you specify a limit but no request for a container?

- The requests value will be set to the container limit

What if you specify a request but no limit for a container?

- The limit will be set to the default memory limit for the namespace
- If no memory limit is specified for the namespace, then the container memory usage is unlimited

Some additional considerations regarding the resource management:

It is possible to set default requests and limits at the namespace level. You can see an example of that, although I'm not sure it would be interesting for a SAS Viya deployment) – Note that it is different from namespace ResourceQuotas (where you limit the total sum of compute resources that can be requested in a given namespace).

If you only specify the limit but not the request, then the requested value will be the limit value.

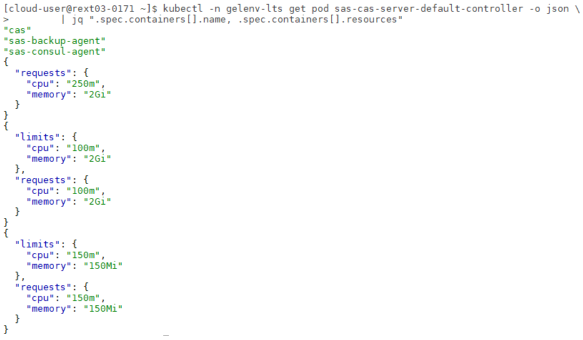
Other way around : if you specify a request but no limit, then the limit will be set to the default memory limit for the namespace, otherwise it will be unlimited…only limited by the Node's available memory (which could lead to the OOM killer being activated and starting to kill processes at the node level)

Container Resources

Show resource definitions

Use kubectl command to show resources and parse with jq utility

```
kubectl -n gelenv-lts get pod sas-cas-server-default-controller -o
json | jq ".spec.containers[].name, .spec.containers[].resources"
```

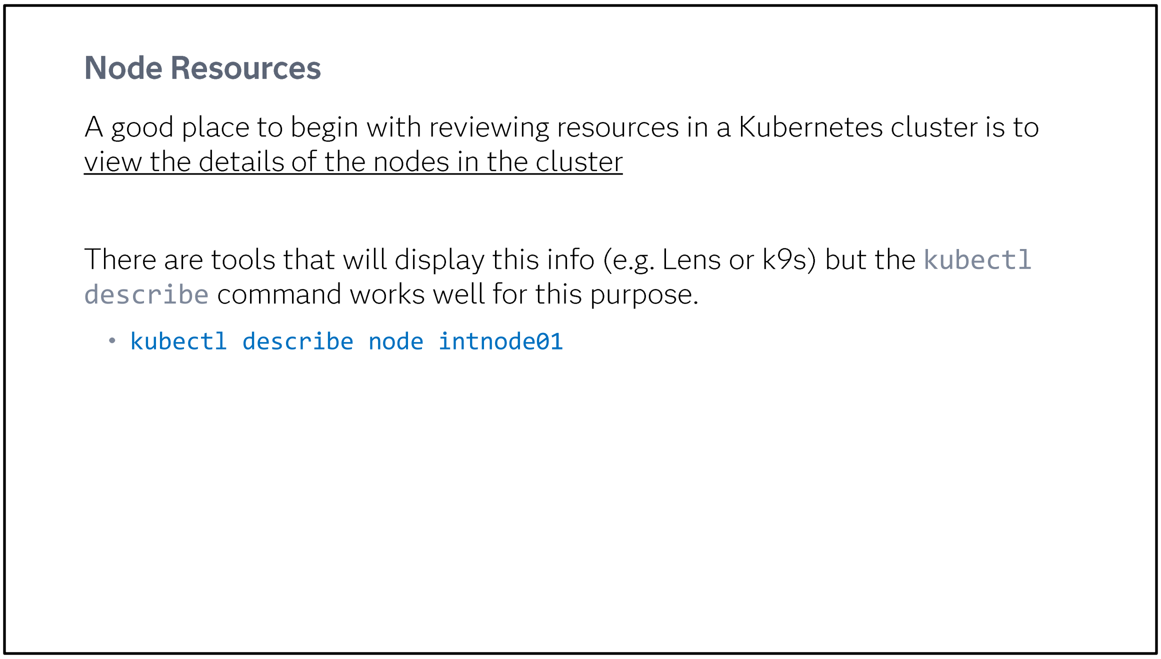


```
[cloud-user@rext03-0171 ~]$ kubectl -n gelenv-lts get pod sas-cas-server-default-controller -o json \
>          | jq ".spec.containers[].name, .spec.containers[].resources"
"cas"
"sas-backup-agent"
"sas-consul-agent"
{
  "requests": {
    "cpu": "250m",
    "memory": "2Gi"
  }
}
{
  "limits": {
    "cpu": "100m",
    "memory": "2Gi"
  },
  "requests": {
    "cpu": "100m",
    "memory": "2Gi"
  }
}
{
  "limits": {
    "cpu": "150m",
    "memory": "150Mi"
  },
  "requests": {
    "cpu": "150m",
    "memory": "150Mi"
  }
}
```

Finally, if we want to look at the resource requests and limit set for a particular pod, we can do it with the kubectl command and jq to parse the output as in this example.

We see a lot of lines because there are 3 containers in the “sas-cas-server-default-controller” pod.
We respectively see the request and limits for the cas container first,
then for the sas-backup agent
then for the sas-consul-agent
In this case, we can see that the CAS pod is in the burstable class of QoS with no CPU or memory limits configured for the CAS container.
That’s the CAS pod’s default resources configuration when the CAS-AUTORESOURCE is disabled.

Node Resources



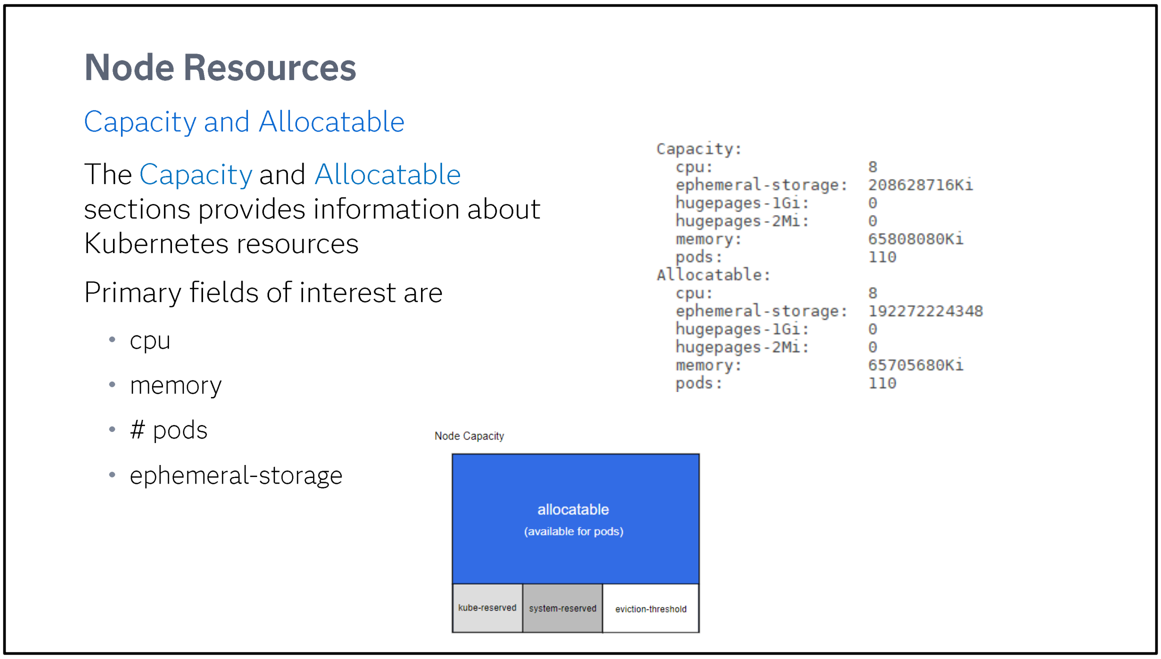
Node Resources

A good place to begin with reviewing resources in a Kubernetes cluster is to view the details of the nodes in the cluster

There are tools that will display this info (e.g. Lens or k9s) but the `kubectl describe` command works well for this purpose.

- `kubectl describe node intnode01`

A good place to begin when reviewing resources in a Kubernetes cluster is to view the details of the nodes in the cluster
The Kubernetes node resource information is easy to get.
You could use something like Lens or even simpler just run the kube CTL describe command for the nodes.
So now we will review the kind of information provided by the "kube CTL describe node" command.



Node Resources

Capacity and Allocatable

The Capacity and Allocatable sections provides information about Kubernetes resources

Primary fields of interest are

- cpu
- memory
- # pods
- ephemeral-storage

```
Capacity:
  cpu:                8
  ephemeral-storage:  208628716Ki
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             65808080Ki
  pods:               110
Allocatable:
  cpu:                8
  ephemeral-storage:  192272224348
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             65705680Ki
  pods:               110
```

The information on the node is also what will be used by the Kubernetes Scheduler to allocate pods based on their resource requests.

We can see that on this node we have :
8-cores,
64 GigaBytes of RAM
and around 200 GigaBytes of ephemeral storage (which correspond to the local disk where the slash var directory is located).
There’s NOT a lot of differences between “Capacity” and “Allocatable”.
'Allocatable' on a Kubernetes node is defined as the amount of compute resources that are available for pods.

Also, in the output’s screenshot on the right side, we can see that Hugepage are not configured on the node. It is something that sometimes may be activated on the Linux OS and could be used in the container resource requests.

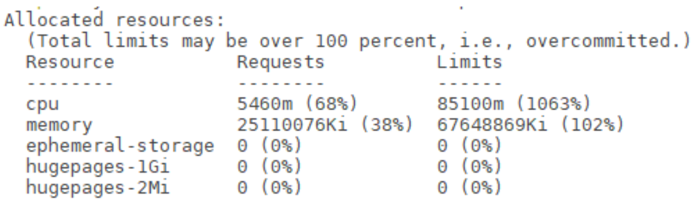
For a SAS Viya deployment, the key information are the number of cores, amount of memory and number of pods.

Node Resources

There is also an Allocated Resources section that reports the current allocation of those resources on the current node.

Primary fields of interest are:

- cpu (measured in millicores where 1000m = 1 CPU)
- memory
- ephemeral-storage



```
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests          Limits
  --------           --------          ------
  cpu                5460m (68%)       85100m (1063%)
  memory             25110076Ki (38%)  67648869Ki (102%)
  ephemeral-storage  0 (0%)            0 (0%)
  hugepages-1Gi      0 (0%)            0 (0%)
  hugepages-2Mi      0 (0%)            0 (0%)
```

There is also a section where you can see the real-time allocation of resources on the current node.
Be careful here : it is NOT the usage; it is the allocated resources (based on the sum of ALL the containers resources requests of the pods that are running in the cluster).
At the moment, we see that 68% of the CPU and 38% of the memory resources of the node have been allocated.

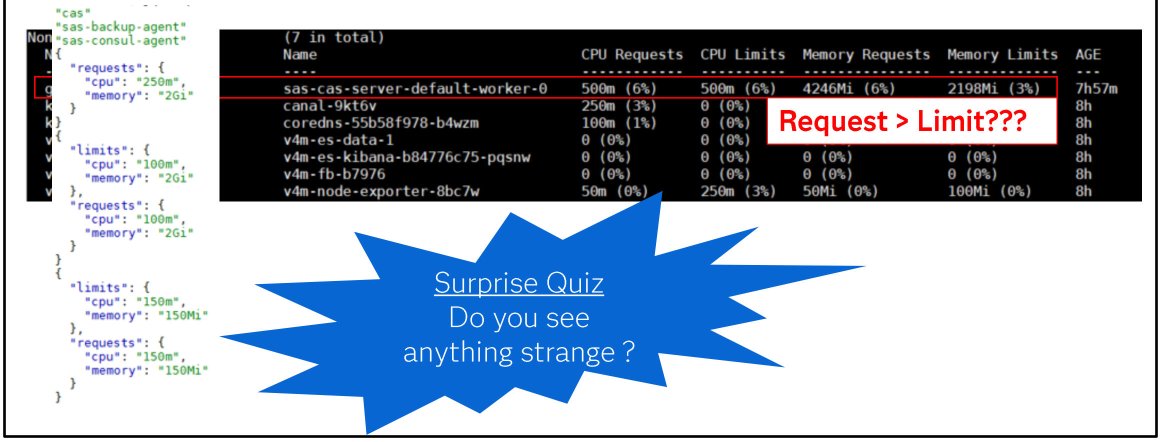
The limits column is not really meaningful in this table. One way to interpret it, though is to say that if all the containers were consuming CPU up to their limit, they would use 1000 percent of the cluster capacity.

On this node we can see that no ephemeral-storage has been requested, but if we were looking at a CAS node, we could see an amount corresponding to the CAS container’s request for ephemeral storage.

Node Resources

The Non-terminated Pods section lists the pods currently running on the node

This section also shows the CPU and Memory Requests/Limits by pods



```
"cas"
"sas-backup-agent"
"sas-consul-agent"
{
  "requests": {
    "cpu": "250m",
    "memory": "2Gi"
  }
}
{
  "limits": {
    "cpu": "100m",
    "memory": "2Gi"
  },
  "requests": {
    "cpu": "100m",
    "memory": "2Gi"
  }
}
{
  "limits": {
    "cpu": "150m",
    "memory": "150Mi"
  },
  "requests": {
    "cpu": "150m",
    "memory": "150Mi"
  }
}
```

```
  (7 in total)
  Name                               CPU Requests  CPU Limits  Memory Requests  Memory Limits  AGE
  ----                               ------------  ----------  ---------------  -------------  ---
  sas-cas-server-default-worker-0    500m (6%)     500m (6%)   4246Mi (6%)      2198Mi (3%)    7h57m
  canal-9kt6v                        250m (3%)     0 (0%)                                      8h
  coredns-55b58f978-b4wzm            100m (1%)     0 (0%)                                      8h
  v4m-es-data-1                      0 (0%)        0 (0%)                                      8h
  v4m-es-kibana-b84776c75-pqsnw      0 (0%)        0 (0%)      0 (0%)           0 (0%)         8h
  v4m-fb-b7976                       0 (0%)        0 (0%)      0 (0%)           0 (0%)         8h
  v4m-node-exporter-8bc7w            50m (0%)      250m (3%)   50Mi (0%)        100Mi (0%)     8h
```

Then there is an interesting section that lists all the pods running on this particular node.

Here is a little Quiz for you : Do you see anything strange here ?
OK let’s wait for a few seconds, to let you think about it…

Here is a hint : look at the cas worker line…
OK let’s wait for a few more seconds…

So yes the Memory Limit is below the Memory request
How can it be possible ?

It is because limits were NOT specified for the CAS container, remember we only had memory requests specified (so automatically the sum of memory request is higher than the sum of memory limits).

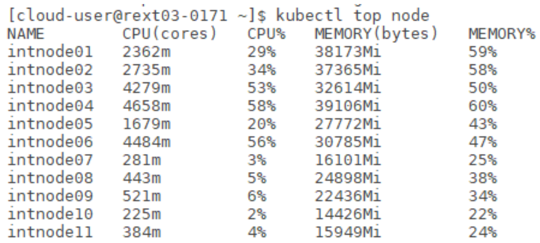
Note the percent, between brackets, after the requests and limits is the percentage of the node capacity that has been allocated to the pod.

Node Resources

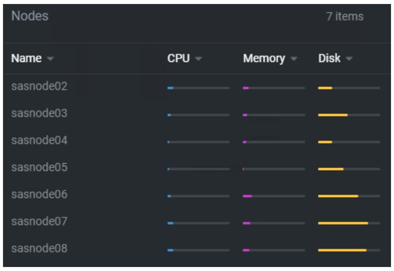
Tools (like Lens/k9s) and kubectl command can also be used to check current resource usage on all nodes of the cluster.

Shows CPU and memory currently in use

```
kubectl top node
```



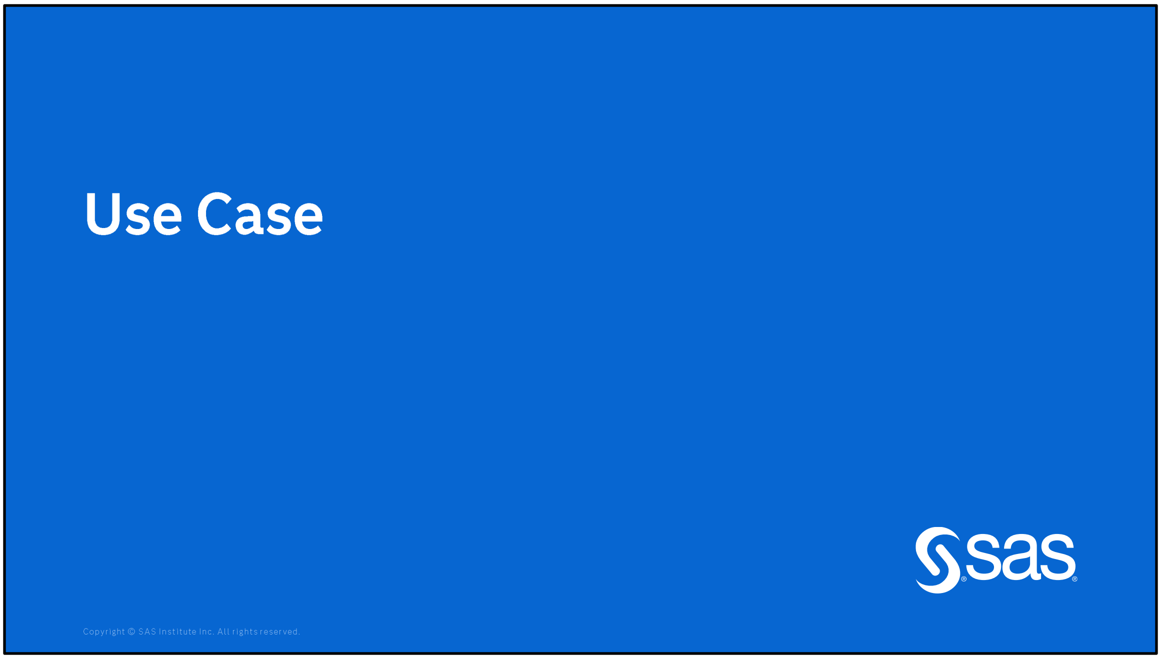
```
[cloud-user@rext03-0171 ~]$ kubectl top node
NAME         CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
intnode01    2362m        29%    38173Mi         59%
intnode02    2735m        34%    37365Mi         58%
intnode03    4279m        53%    32614Mi         50%
intnode04    4658m        58%    39106Mi         60%
intnode05    1679m        20%    27772Mi         43%
intnode06    4484m        56%    30785Mi         47%
intnode07    281m         3%     16101Mi         25%
intnode08    443m         5%     24898Mi         38%
intnode09    521m         6%     22436Mi         34%
intnode10    225m         2%     14426Mi         22%
intnode11    384m         4%     15949Mi         24%
```



Nodes 7 items

Name	CPU	Memory	Disk
sasnode02			
sasnode03			
sasnode04			
sasnode05			
sasnode06			
sasnode07			
sasnode08			

And finally, you also have the handy “kube CTL top node” command to get an overview of the current workload breakdown by node.
This time it corresponds to the usage, not the allocation.
Note that the “kube CTL top” command will only work if you have a metrics service deployed in the Kubernetes cluster.



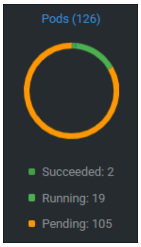
Use Case

Context

The customer is testing the deployment of a second Viya environment (in its own distinct namespace) of the same cluster

After 15 hours the second Viya environment is still not ready

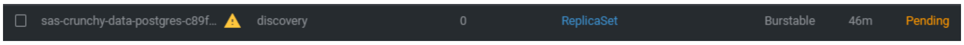
Most of the pods are not running or not ready



The customer is testing the deployment of a second Viya environment (in its own distinct namespace) of the same Kubernetes cluster.
But after 15 hours the second Viya environment is still not ready
Most of the pods are not running or not ready.

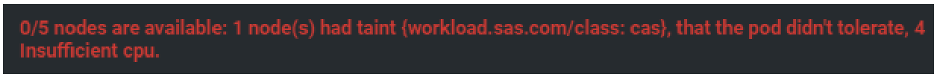
Kubernetes Node Resources - example

After some investigations, it turns out that the sas-crunchy-postgres pod remains PENDING (even after a restart), preventing other pods to start



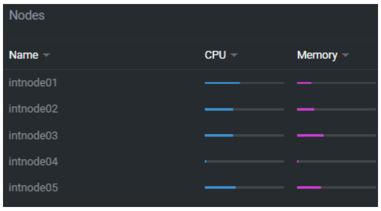
```
□ sas-crunchy-data-postgres-c89f... ⚠ discovery   0   ReplicaSet   Burstable   46m   Pending
```

In the crunchy pod event log, we have a message about insufficient CPU



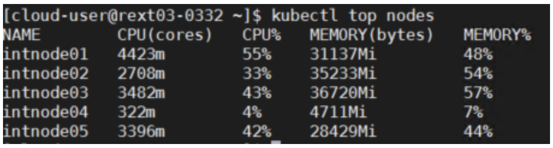
```
0/5 nodes are available: 1 node(s) had taint {workload.sas.com/class: cas}, that the pod didn't tolerate, 4
Insufficient cpu.
```

However, none of the Nodes seems particularly busy on their CPU…



Nodes

Name	CPU	Memory
intnode01		
intnode02		
intnode03		
intnode04		
intnode05		

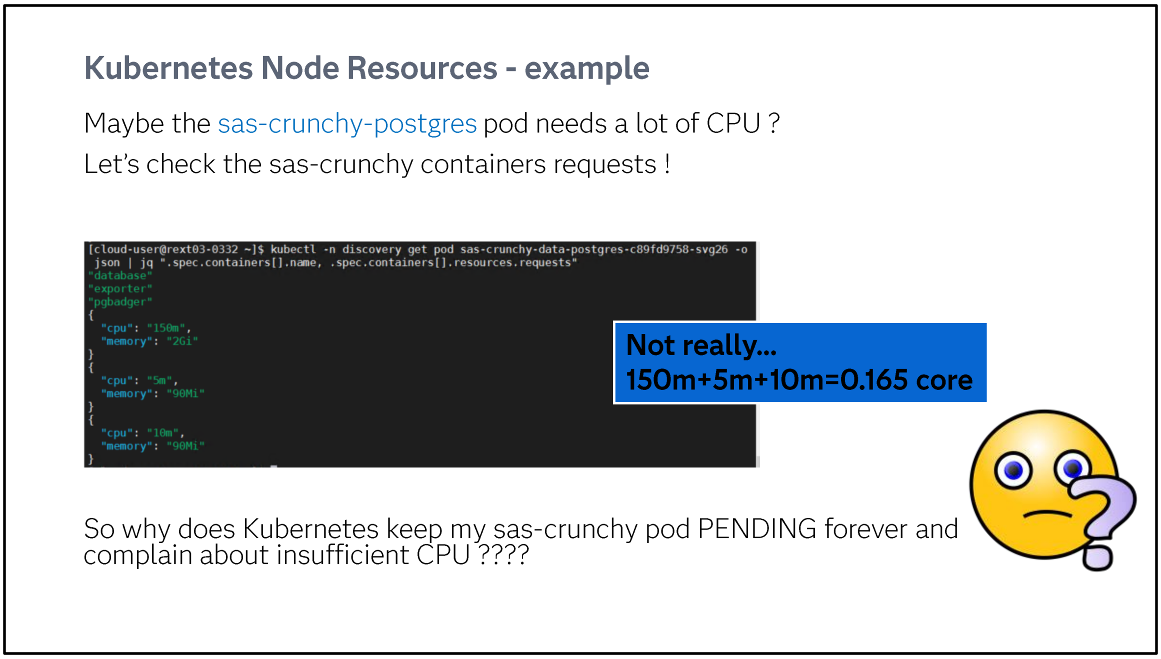


```
[cloud-user@rext03-0332 ~]$ kubectl top nodes
NAME         CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
intnode01    4423m        55%    31137Mi         48%
intnode02    2708m        33%    35233Mi         54%
intnode03    3482m        43%    36720Mi         57%
intnode04    322m         4%     4711Mi          7%
intnode05    3396m        42%    28429Mi         44%
```

After some investigations, it turns out that the sas-crunchy-postgres pod remains PENDING (even after a restart), preventing other pods to start

The pod’s event log tells us that the sas-crunchy-postgres can not start because 4 nodes have “insufficient CPU”. We can see this log with a “kube CTL describe” command.

However, when we look at the node’s current utilization, none of them seems particularly busy on their CPU…
You can see that we use the “kube CTL top” command here.
It is an equivalent of the linux “top” command that gives the real time CPU and memory utilization for each Kubernetes node.
Whether we look in Lens or in the “kube CTL top” command result, none of the node seems to be under high CPU pressure…
Each node has 8 cores and only “intnode01” seems to be using more than 50% of it.



Kubernetes Node Resources - example

Maybe the sas-crunchy-postgres pod needs a lot of CPU ?

Let’s check the sas-crunchy containers requests !

```
[cloud-user@rext03-0332 ~]$ kubectl -n discovery get pod sas-crunchy-data-postgres-c89fd9758-svg26 -o
 json | jq ".spec.containers[].name, .spec.containers[].resources.requests"
"database"
"exporter"
"pgbadger"
{
  "cpu": "150m",
  "memory": "2Gi"
}
{
  "cpu": "5m",
  "memory": "90Mi"
}
{
  "cpu": "10m",
  "memory": "90Mi"
}
```

Not really...
150m+5m+10m=0.165 core

So why does Kubernetes keep my sas-crunchy pod PENDING forever and complain about insufficient CPU ????

So maybe the explanation is that my SAS Crunchy Postgres needs a lot of CPU, and that is the reason why Kubernetes can not schedule it on any node ?

Let's check the SAS Crunchy container request.

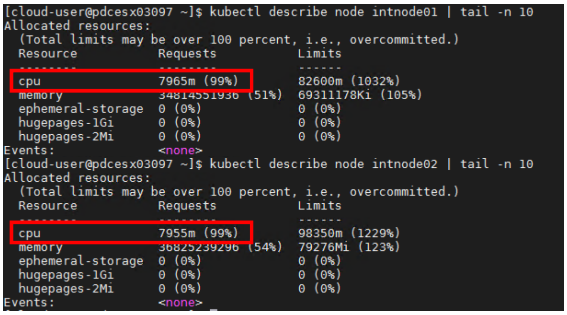
We have our very handy command, where we can see what has been defined in terms of CPU requests.

not we do the sum, it will be around 165 millicores.
It's really not a lot. It's like 16 percent of one core which is definitely not a very high request.

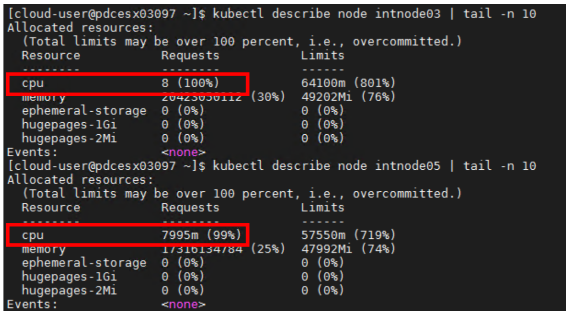
So why does Kubernetes keep my SAS Crunchy Postgres pending forever and complain about insufficient CPU?
I’ll let you think about what could be the explanation for a few seconds…

Kubernetes Node Resources – example

The answer is in the “Allocated Resources” table…



```
[cloud-user@pdcesx03097 ~]$ kubectl describe node intnode01 | tail -n 10
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests          Limits
  --------           --------          ------
  cpu                7965m (99%)       82600m (1032%)
  memory             34814551936 (51%)  69311178Ki (105%)
  ephemeral-storage  0 (0%)            0 (0%)
  hugepages-1Gi      0 (0%)            0 (0%)
  hugepages-2Mi      0 (0%)            0 (0%)
Events:              <none>
[cloud-user@pdcesx03097 ~]$ kubectl describe node intnode02 | tail -n 10
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests          Limits
  --------           --------          ------
  cpu                7955m (99%)       98350m (1229%)
  memory             36825239296 (54%)  79276Mi (123%)
  ephemeral-storage  0 (0%)            0 (0%)
  hugepages-1Gi      0 (0%)            0 (0%)
  hugepages-2Mi      0 (0%)            0 (0%)
Events:              <none>
```



```
[cloud-user@pdcesx03097 ~]$ kubectl describe node intnode03 | tail -n 10
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests          Limits
  --------           --------          ------
  cpu                8 (100%)          64100m (801%)
  memory             20423050112 (30%)  49202Mi (76%)
  ephemeral-storage  0 (0%)            0 (0%)
  hugepages-1Gi      0 (0%)            0 (0%)
  hugepages-2Mi      0 (0%)            0 (0%)
Events:              <none>
[cloud-user@pdcesx03097 ~]$ kubectl describe node intnode05 | tail -n 10
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests          Limits
  --------           --------          ------
  cpu                7995m (99%)       57550m (719%)
  memory             17316134784 (25%)  47992Mi (74%)
  ephemeral-storage  0 (0%)            0 (0%)
  hugepages-1Gi      0 (0%)            0 (0%)
  hugepages-2Mi      0 (0%)            0 (0%)
Events:              <none>
```

Look at the CPU Requests on each node

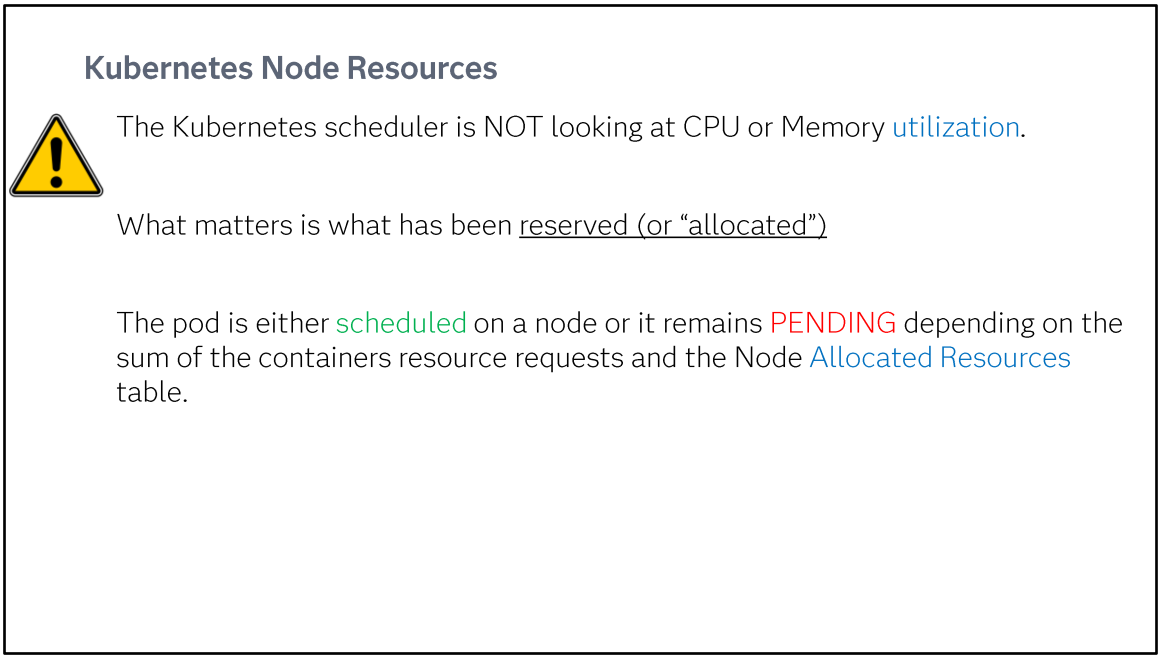
Kubernetes was right ! *“4, insufficient CPU”* (in terms of resource allocation)

The answer can be found in the node's descriptions and more precisely in the “Allocated Resources” table.

now you will see that each of the nodes that could hold my SAS Crunchy pods, the allocated resource for the CPU is around 100 percent.

So even if in terms of usage, the utilization of the CPU of my nodes is pretty low (maybe limited to 50 percent), the fact that we have a lot of containers doing requests will reserve ALL the CPU capacity declared on the nodes.
The CPU request on each node is too high.

What it tells us is that the Kubernetes scheduler is not looking at the CPU or memory utilization.
It is only looking at the sum of the specified requests, resource requests of all the containers.
What matters is really what has been already reserved or allocated.
The pods will be scheduled on a node if there is remaining, allocatable capacity, otherwise it will remain pending.
That's very important to understand.



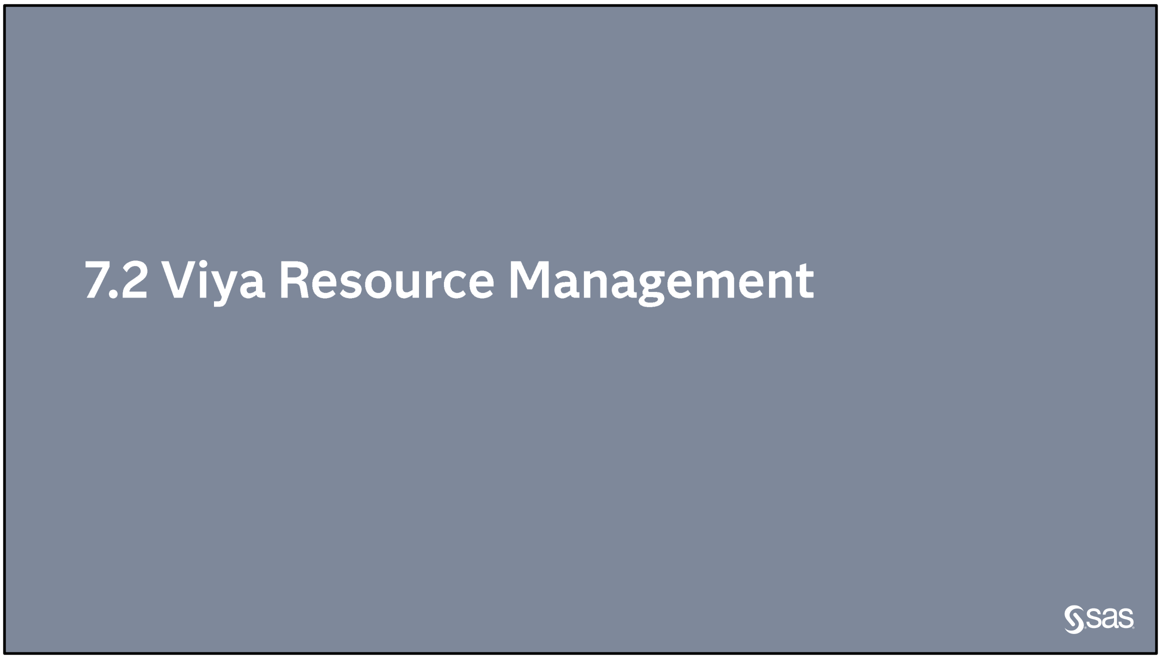
Kubernetes Node Resources

The Kubernetes scheduler is NOT looking at CPU or Memory utilization.

What matters is what has been reserved (or "allocated")

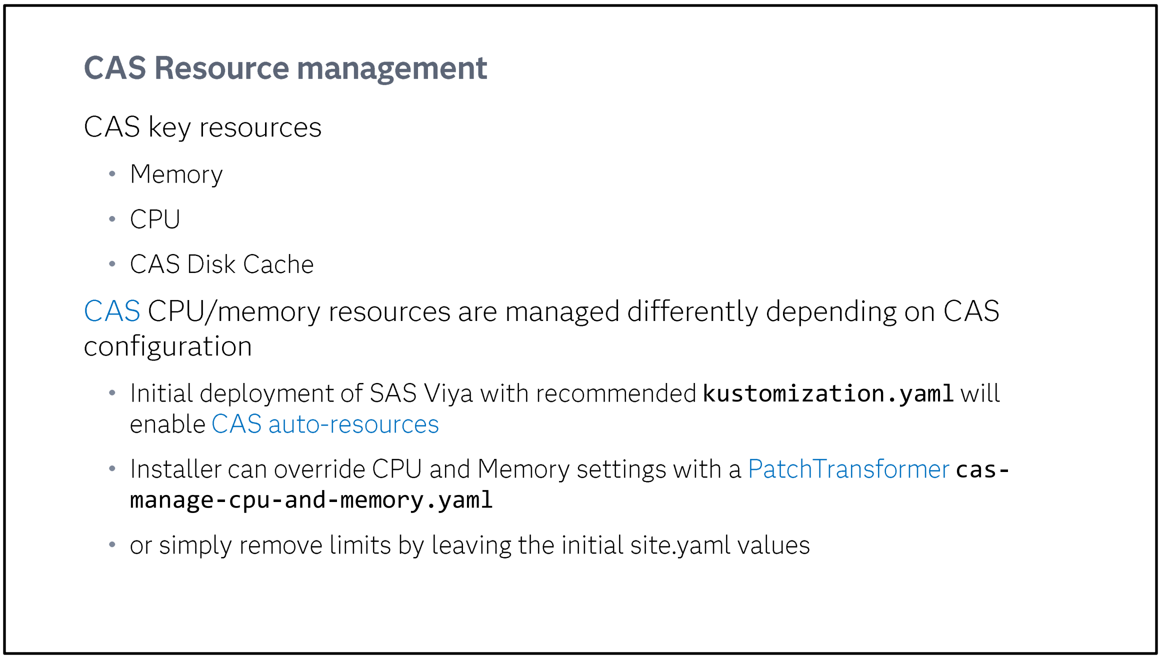
The pod is either scheduled on a node or it remains PENDING depending on the sum of the containers resource requests and the Node Allocated Resources table.

The key take-away of this use-case is that the Kubernetes scheduler is NOT looking at CPU or Memory utilization to make its decision to start a pod or not.
Instead, it compares the node's allocated resources and the new pod resource requests.
Of course, there can also be additional factors such as pod affinities and tolerations that can prevent a pod to be scheduled on a node…
But if all the nodes show something near 100% for their allocated resources, then it is unlikely that the cluster will accept any new workload.



7.2 Viya Resource Management

CAS Pods Resource Management Options



CAS Resource management

CAS key resources

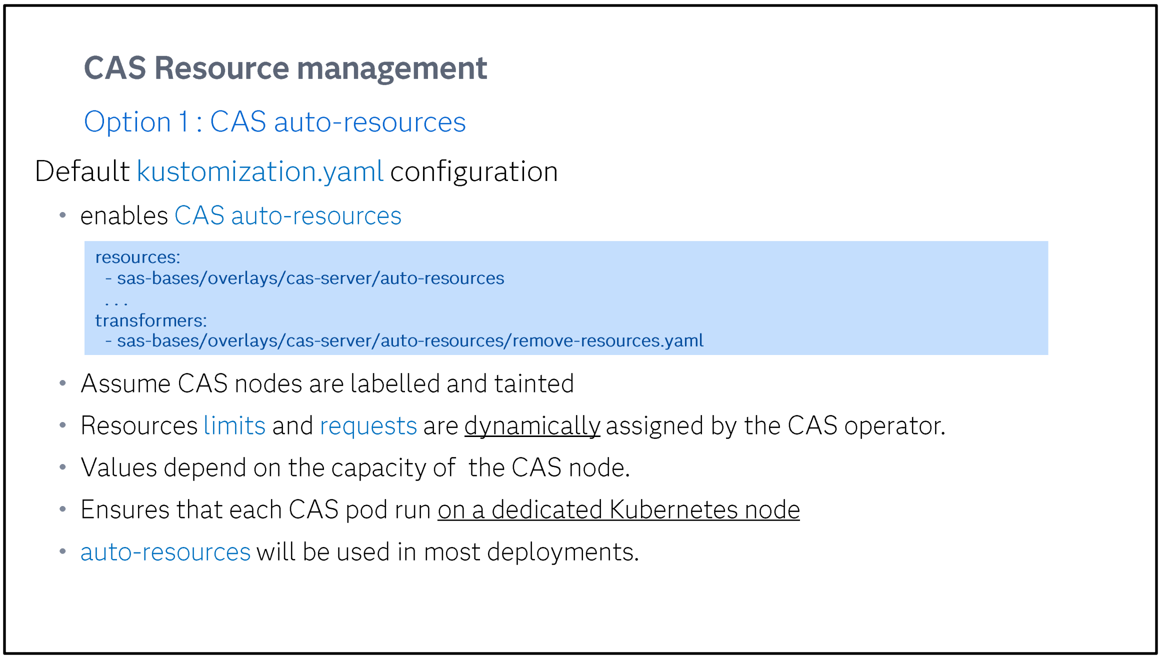
- Memory
- CPU
- CAS Disk Cache

CAS CPU/memory resources are managed differently depending on CAS configuration

- Initial deployment of SAS Viya with recommended `kustomization.yaml` will enable CAS auto-resources
- Installer can override CPU and Memory settings with a PatchTransformer `cas-manage-cpu-and-memory.yaml`
- or simply remove limits by leaving the initial site.yaml values

For any version of Viya, the 3 keys resources for CAS are the memory, the CPU and the CAS Disk cache.

There are three possibilities to configure the CPU and memory resources allocation for CAS:
First, the CAS Auto-Resources configuration (where the CPU and memory values are automatically set for you)
Then we have the “cas-manage-cpu-and-memory” YAML file where we can set our own values for the CPU and memory requests and limits.
And the last option you can simply just keep the default values: not choosing the CAS Auto Resource, and keep the default values for the CAS containers.
So there are basically three options available when it comes to the CPU and memory resource configuration for CAS and we will now detail each of them.



CAS Resource management

Option 1 : CAS auto-resources

Default kustomization.yaml configuration

- enables CAS auto-resources

```
resources:
  - sas-bases/overlays/cas-server/auto-resources
  . . .
transformers:
  - sas-bases/overlays/cas-server/auto-resources/remove-resources.yaml
```

- Assume CAS nodes are labelled and tainted
- Resources limits and requests are dynamically assigned by the CAS operator.
- Values depend on the capacity of the CAS node.
- Ensures that each CAS pod run on a dedicated Kubernetes node
- auto-resources will be used in most deployments.

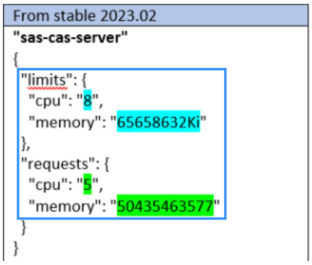
The CAS auto-resources is the recommended option and is enabled in the default kustomization DOT YAML file that is provided in the official documentation.
The 2 lines correspond to the activation of what we call the “CAS auto-resources” mode.
When this mode is enabled, it is expected that all the node(s) that have been dedicated to CAS have been: NOT only labelled but also tainted to prevent other pods to run there.
When the CAS operator detects that CAS Auto-resources is configured, it will search for nodes with the “CAS” label.
Then it will look at the Node’s capacity and dynamically assign the request and limit values for the CAS pods.

The objective here is to make sure that each CAS pod can use all the available resources on a node.
For example, in a CAS MPP deployment, the result is that each CAS instance (controllers and workers) will have a dedicated Kubernetes worker node.
CAS Auto Resources should be used in most deployment.
It is our official recommendation, because then CAS can use all the available resources in terms of CPU, memory on the node and maybe also you can plan some very fast storage system on those nodes.
We know that CAS is very CPU intensive and memory intensive, so that’s usually the best way to configure CAS.

CAS Resource management

Option 1 : CAS auto-resources

Example of auto-resources on CAS node pool with **8 CPU / 64GB** CAS machines



From stable 2023.02

```
"sas-cas-server"
{
 "limits": {
   "cpu": "8",
   "memory": "65658632Ki"
 },
 "requests": {
   "cpu": "5",
   "memory": "50435463577"
  }
}
```

- CPU and memory limits set to approximatively 100% of the node's capacity
- CPU and memory requests set to approximatively 75% of the node's capacity

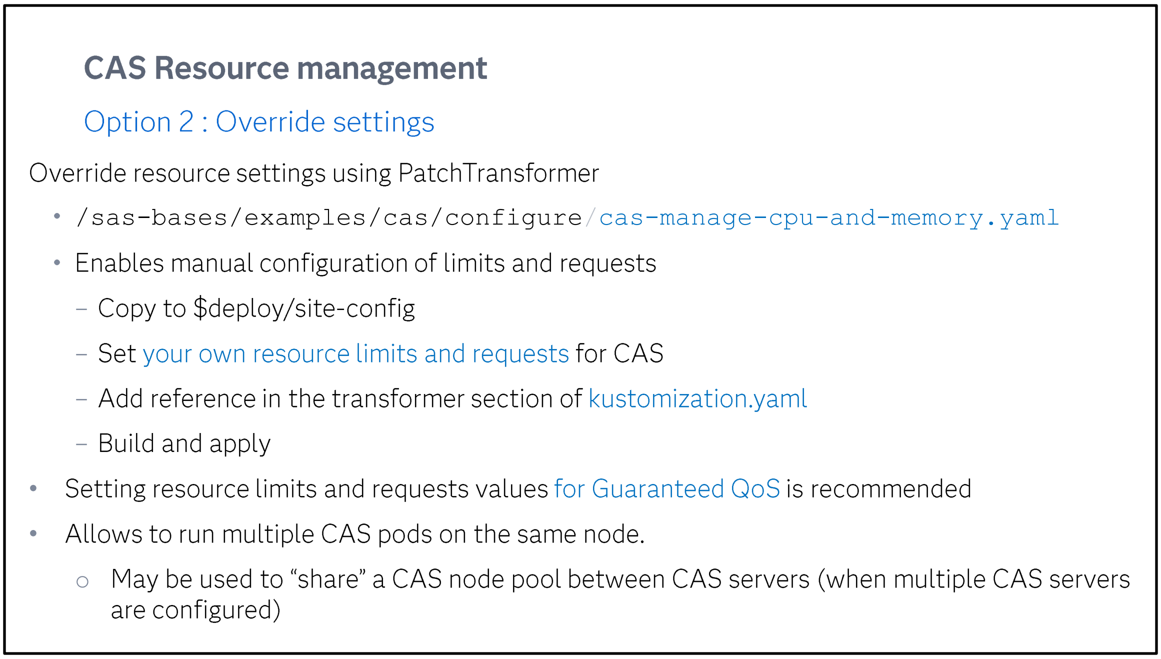
- Since stable 2023.02, the QoS class changed from Guaranteed to Burstable.
- "Guaranteed" QoS provides higher protection (against evictions) of the pod when it is in concurrence with other pods running on the same node. BUT it is not really **needed when the node is tainted…**
- "Burstable" QoS enables other containers in the pod to increase their resource usage when a burst of activity is required (cas-backup agent can request more CPU which reduce the backup execution times).

On this slide, we have an example of the values that will be set by the CAS operator when you enable the CAS Auto Resources.
In this example, we have CAS nodes with 8 CPU and 64 gigabytes.

* As you can see, the CPU and memory limits are set to approximately 100% of the nodes' capacity. Okay, so the cpu limit value is "8" here, because we have 8 cores on this machine.
* Then for the requests in terms of CPU and memory, it will be set to approximately 75% of the nodes' capacity.

In the past, we had the same values for the limits and requests.
By doing that we had this Guaranteed Quality of Service, and we were sure that there would be no risk of eviction of our CAS pods.
But it has been changed, because, if we assume that the CAS nodes have been tainted, then it means that ONLY our CAS pods running there.
So, there was no need to protect it against the eviction, right?

The other advantage to have a request that are lower than the limits, is that it allows other containers inside the CAS pod to increase their resource usage when needed.
For example, the CAS backup agent used to have a very limited access to the CPU power because of a fixed and low CPU limit, but now, since we are more flexible on the limits and requests, the CAS backup agent can increase its CPU utilization, which ultimately helps to reduce the backup execution times !
With this change there is a better utilization of the CAS node's resources.



CAS Resource management

Option 2 : Override settings

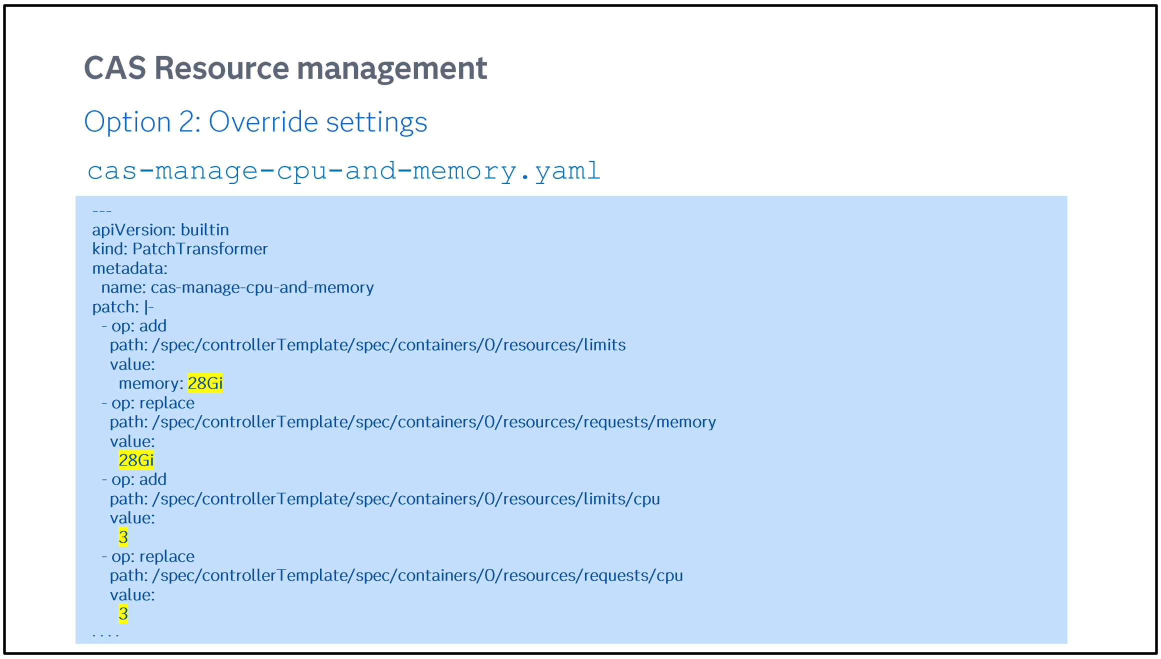
Override resource settings using PatchTransformer

- /sas-bases/examples/cas/configure/cas-manage-cpu-and-memory.yaml
- Enables manual configuration of limits and requests
  - Copy to $deploy/site-config
  - Set your own resource limits and requests for CAS
  - Add reference in the transformer section of kustomization.yaml
  - Build and apply
- Setting resource limits and requests values for Guaranteed QoS is recommended
- Allows to run multiple CAS pods on the same node.
  - May be used to “share” a CAS node pool between CAS servers (when multiple CAS servers are configured)

With the second option, we can specify our own values (instead of using the cas auto-resource mode). There is a specific Kustomize PatchTransformer where we set the CPU and memory requests and limits.

As usually to implement the configuration change, we need to copy the PatchTransformer template into our “site-config” directory, replace the limits and request placeholder with our own values, then reference the PatchTransformer file in our main kustomization DOT YAML and rebuild and re-apply the Kubernetes manifests.

Since in this case the CAS pod will co-exist with others pods on the nodes, it is strongly recommended to set the limit in a way the CAS pod goes in the Guaranteed Q O S, so it is protected against Kubernetes eviction.
It might be interesting to do that if we need to run multiple instances of CAS on the same set of nodes (if we have configured multiple CAS servers or multiple Viya environment in the same cluster).
We might also do that if we want to run CAS MPPs with a high number of workers, but we don’t have enough physical machines to run a worker by machine.



CAS Resource management

Option 2: Override settings

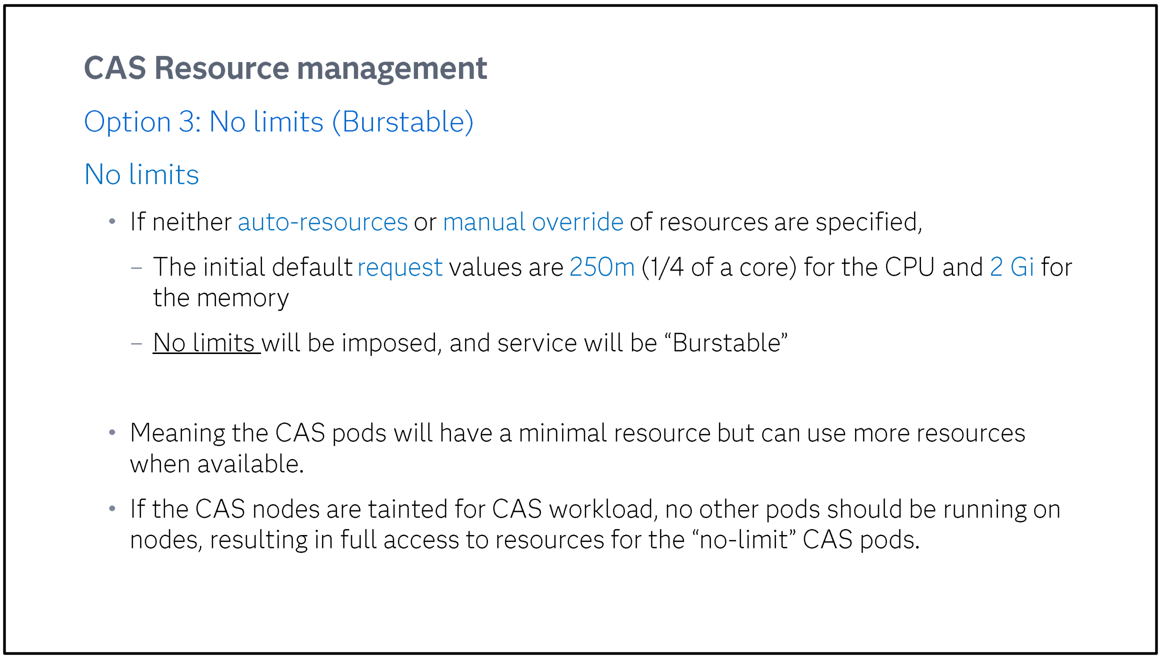
`cas-manage-cpu-and-memory.yaml`

```
---
apiVersion: builtin
kind: PatchTransformer
metadata:
  name: cas-manage-cpu-and-memory
patch: |-
  - op: add
    path: /spec/controllerTemplate/spec/containers/0/resources/limits
    value:
      memory: 28Gi
  - op: replace
    path: /spec/controllerTemplate/spec/containers/0/resources/requests/memory
    value:
      28Gi
  - op: add
    path: /spec/controllerTemplate/spec/containers/0/resources/limits/cpu
    value:
      3
  - op: replace
    path: /spec/controllerTemplate/spec/containers/0/resources/requests/cpu
    value:
      3
. . . .
```

Here is an example of the cas-manage-cpu-and-memory dot YAML file, where we implement the Guaranteed Quality of Service.

Note that the memory limit and request have the same value, while the CPU limit and request also have the same value.

Doing that for all the containers inside the pod will ensure that the pod is categorized in the Guaranteed QoS by the Kubernetes scheduler.



CAS Resource management

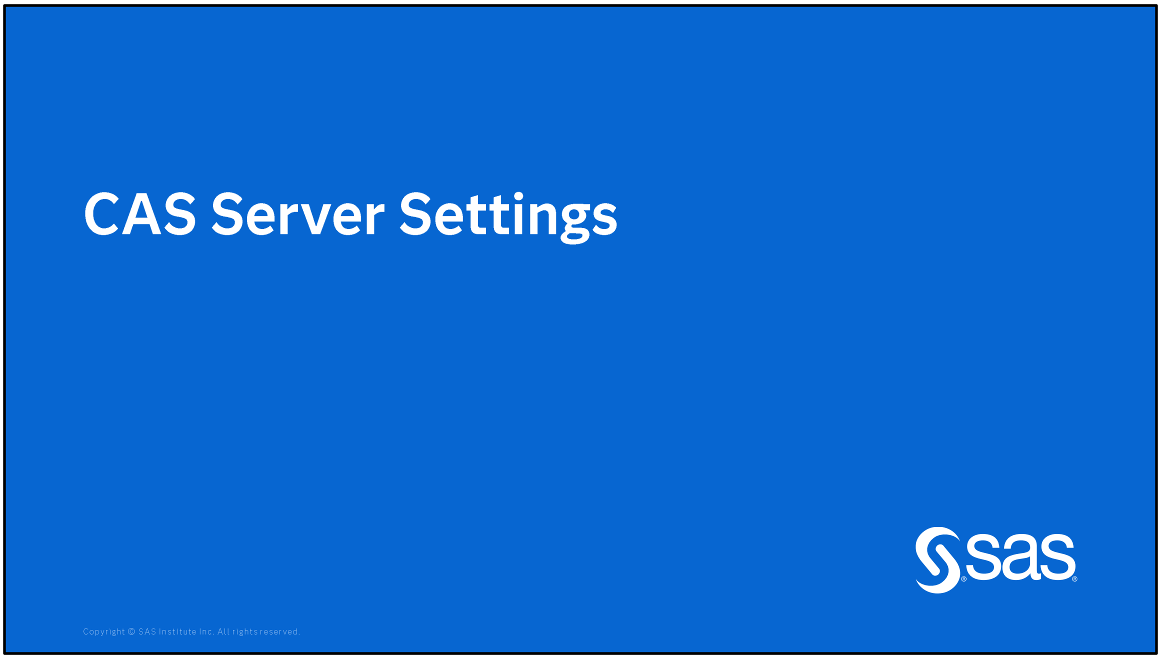
Option 3: No limits (Burstable)

No limits

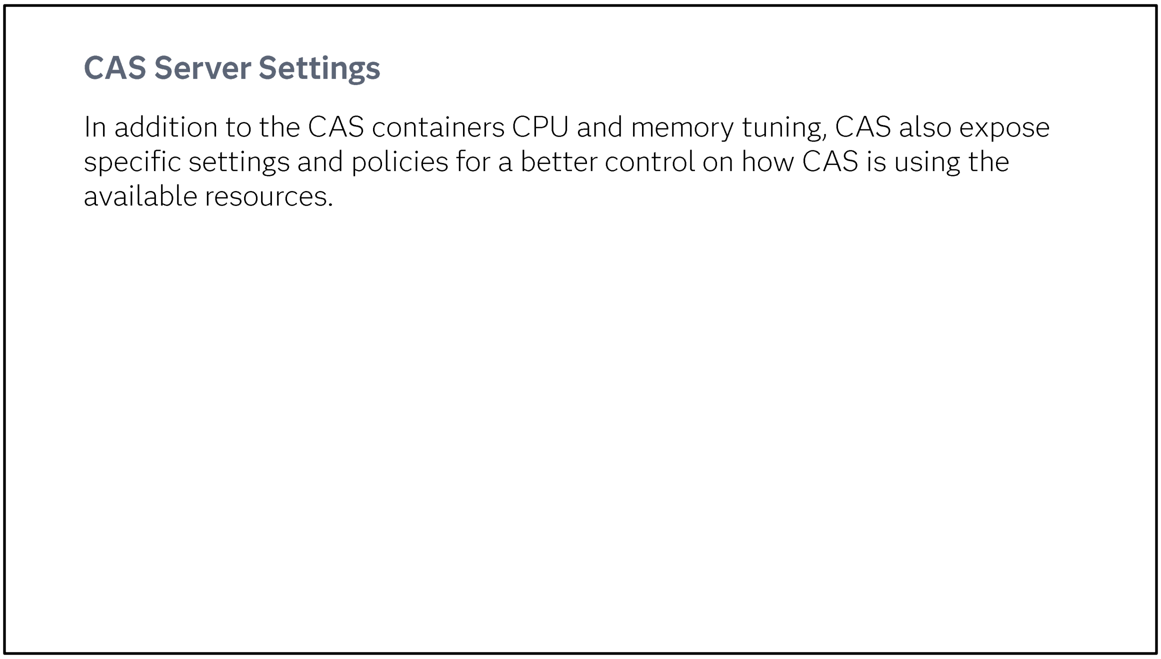
- If neither auto-resources or manual override of resources are specified,
  - The initial default request values are 250m (1/4 of a core) for the CPU and 2 Gi for the memory
  - No limits will be imposed, and service will be "Burstable"
- Meaning the CAS pods will have a minimal resource but can use more resources when available.
- If the CAS nodes are tainted for CAS workload, no other pods should be running on nodes, resulting in full access to resources for the "no-limit" CAS pods.

Finally, the last option for the CAS resources is the "no-limits" option.
It is when you are not using the cas auto-resource in kustomization.yaml and you are not using the PatchTransformer to specify your own CPU and memory values.
In such case, the CAS resources requests and limits will get the default values.

Remember that if the CAS nodes are tainted for CAS workload, no other pods should be on the CAS node - which results in full access to the Node's resources for the CAS pods.



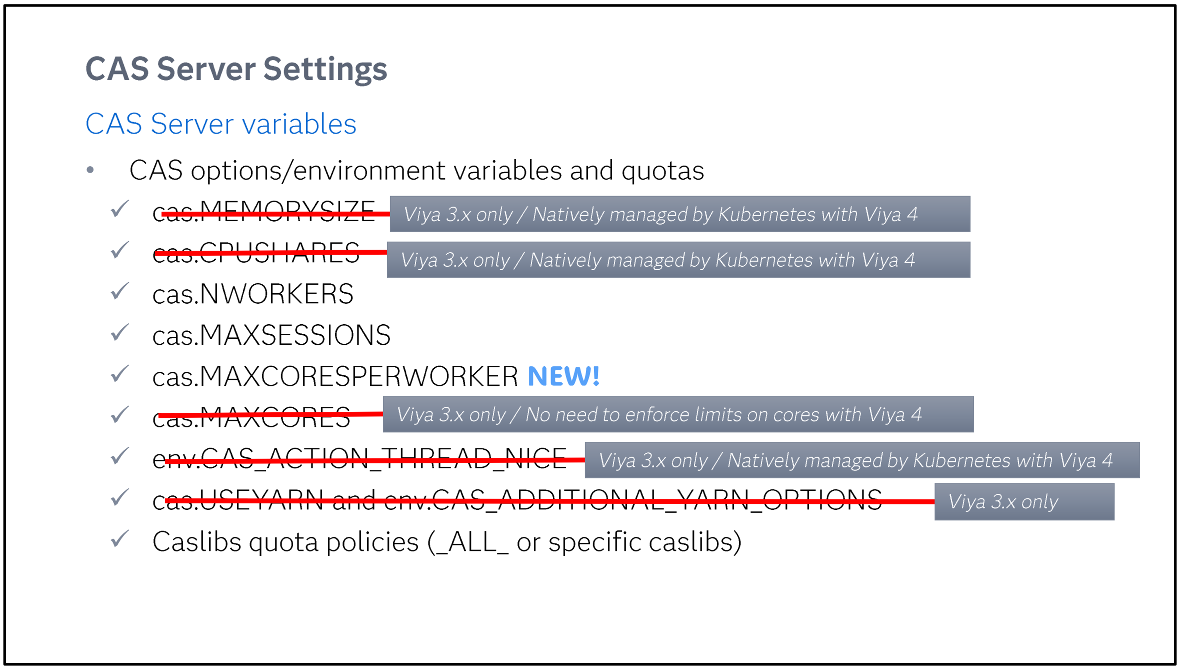
CAS Server Settings



CAS Server Settings

In addition to the CAS containers CPU and memory tuning, CAS also expose specific settings and policies for a better control on how CAS is using the available resources.

In addition to the CAS container CPU and memory tuning that we have in other modules, CAS can also expose some specific settings and policies
for better control on how CAS is using the available resource in the Kubernetes cluster.



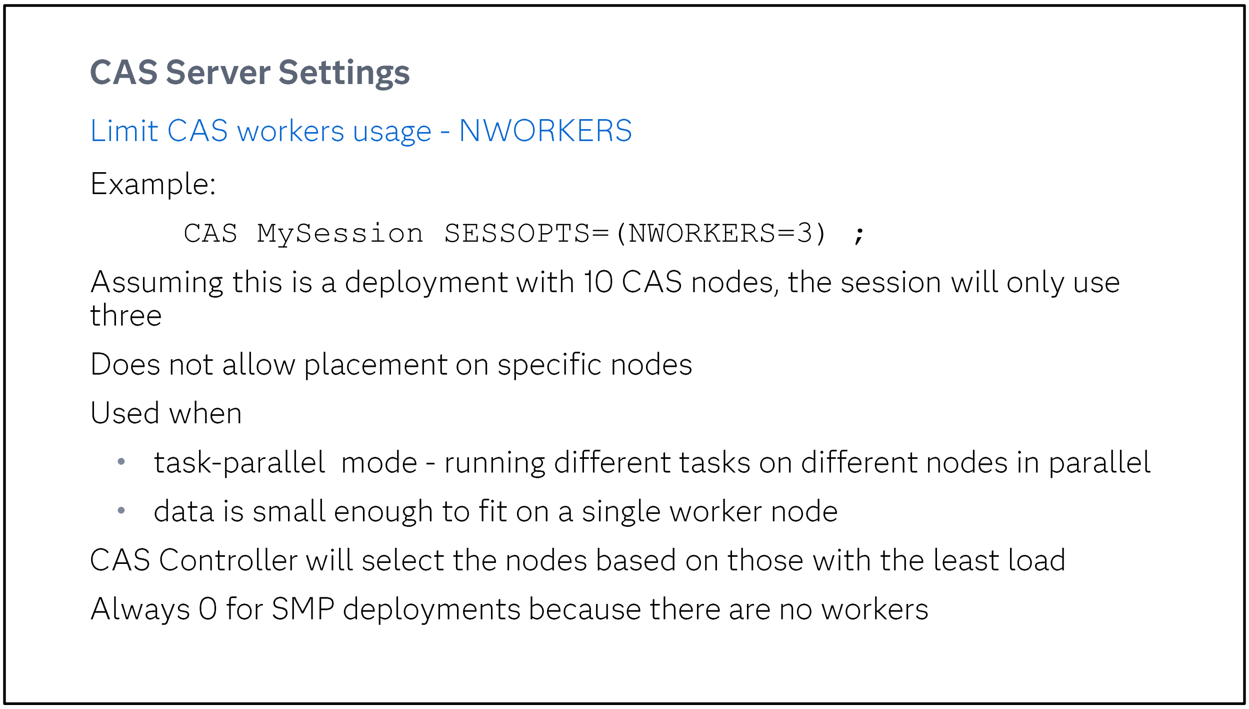
CAS Server Settings

CAS Server variables

- CAS options/environment variables and quotas
  - ✓ ~~cas.MEMORYSIZE~~ *Viya 3.x only / Natively managed by Kubernetes with Viya 4*
  - ✓ ~~cas.CPUSHARES~~ *Viya 3.x only / Natively managed by Kubernetes with Viya 4*
  - ✓ cas.NWORKERS
  - ✓ cas.MAXSESSIONS
  - ✓ cas.MAXCORESPERWORKER **NEW!**
  - ✓ ~~cas.MAXCORES~~ *Viya 3.x only / No need to enforce limits on cores with Viya 4*
  - ✓ ~~env.CAS_ACTION_THREAD_NICE~~ *Viya 3.x only / Natively managed by Kubernetes with Viya 4*
  - ✓ ~~cas.USEYARN and env.CAS_ADDITIONAL_YARN_OPTIONS~~ *Viya 3.x only*
  - ✓ Caslibs quota policies (_ALL_ or specific caslibs)

With Viya 3, we had some CAS server options like cas.MEMORYSIZE or cas.CPUSHARES to limit the CAS servers Memory and CPU utilization.
Now it is something that is managed by Kubernetes natively based on the containers resource requests and limits.

However, we still have settings like the NWORKERS where you can force a CAS action to run only on a subset of the nodes, or the MAXSESSION where you can globally limit the number of CAS session that an be running at the same time on the CAS server. In March 2025 a new option MAXCORESPERWORKER was introduced.
Finally we also still have the CASlib quota policies.



CAS Server Settings

Limit CAS workers usage - NWORKERS

Example:

```
      CAS MySession SESSOPTS=(NWORKERS=3) ;
```

Assuming this is a deployment with 10 CAS nodes, the session will only use three

Does not allow placement on specific nodes

Used when

- task-parallel mode - running different tasks on different nodes in parallel
- data is small enough to fit on a single worker node

CAS Controller will select the nodes based on those with the least load

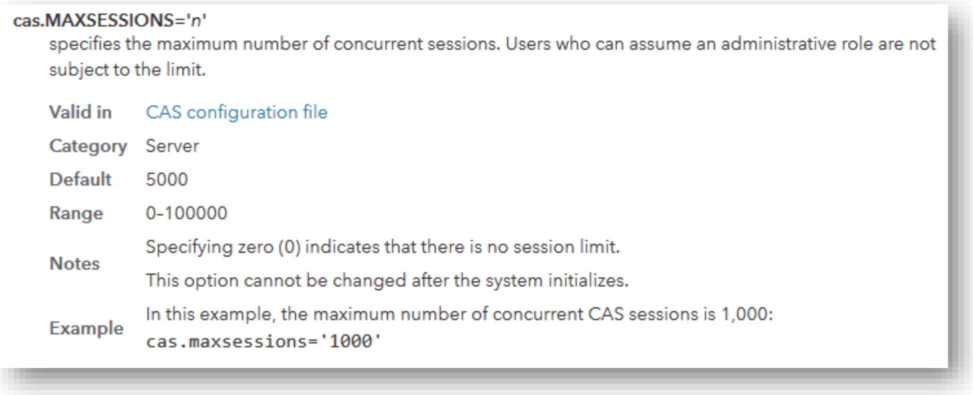
Always 0 for SMP deployments because there are no workers

The NWORKERS option allows the user to set the number of worker nodes to be used for the session in a distributed deployment.
This will typically only be used for small datasets so that data will be placed on one node.
Or it may be used for task-parallel mode, where three different nodes may run three different actions on three different datasets in parallel.
The user can NOT select on which nodes to execute.
The CAS controller places the session on nodes with the lowest load.
This features requires some testing to understand the potential implications on other workloads. Refer to the official documentation for additional details.

CAS Server Settings

MAXSESSIONS

MAXSESSIONS - specifies the maximum number of concurrent sessions



cas.MAXSESSIONS='*n*'

specifies the maximum number of concurrent sessions. Users who can assume an administrative role are not subject to the limit.

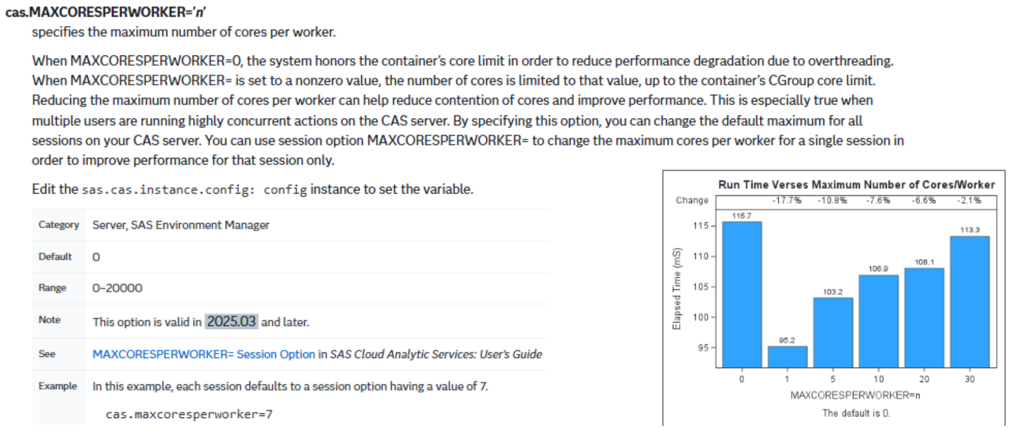
Valid in	CAS configuration file
Category	Server
Default	5000
Range	0-100000
Notes	Specifying zero (0) indicates that there is no session limit. This option cannot be changed after the system initializes.
Example	In this example, the maximum number of concurrent CAS sessions is 1,000: `cas.maxsessions='1000'`

If the administrator wants to limit the number of CAS sessions, then the MAXSESSIONS option can be used to do so.
This may be necessary to prevent overcommitting resources for a CAS server.
BUT If used, then it may be time to reconsider resizing the deployment.

CAS Server Settings

MAXCORESPERWORKER

MAXCORESPERWORKER - specifies the maximum number of cores per worker.



cas.MAXCORESPERWORKER='*n*'

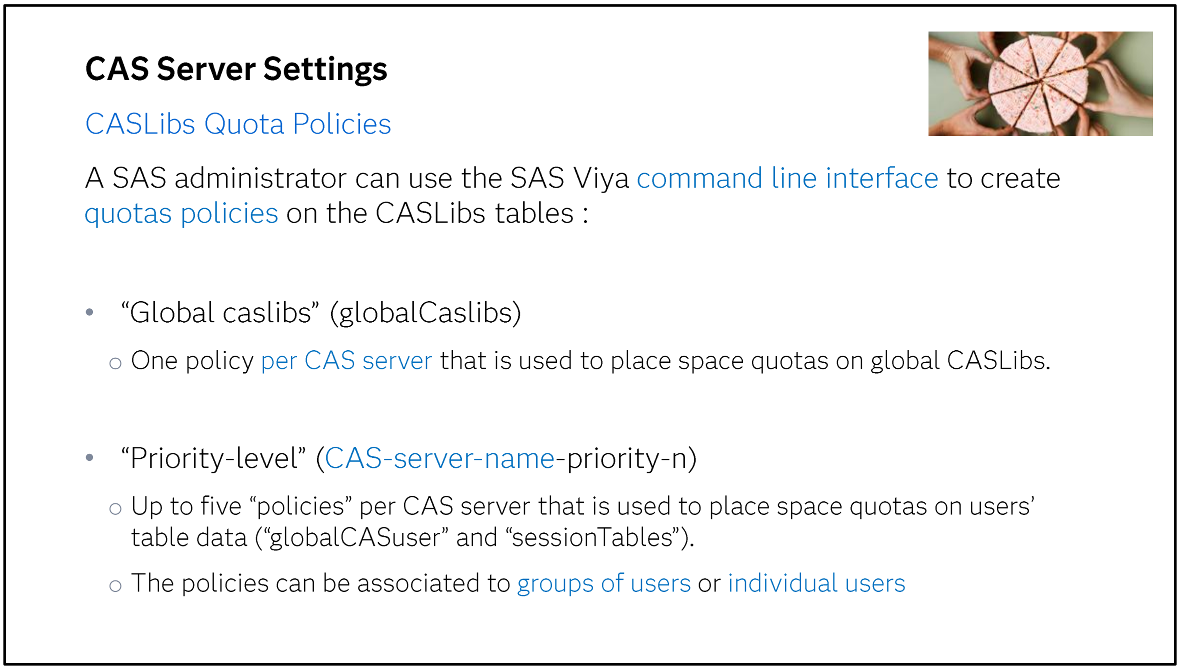
specifies the maximum number of cores per worker.

When MAXCORESPERWORKER=0, the system honors the container's core limit in order to reduce performance degradation due to overthreading. When MAXCORESPERWORKER= is set to a nonzero value, the number of cores is limited to that value, up to the container's CGroup core limit. Reducing the maximum number of cores per worker can help reduce contention of cores and improve performance. This is especially true when multiple users are running highly concurrent actions on the CAS server. By specifying this option, you can change the default maximum for all sessions on your CAS server. You can use session option MAXCORESPERWORKER= to change the maximum cores per worker for a single session in order to improve performance for that session only.

Edit the `sas.cas.instance.config: config` instance to set the variable.

Category	Server, SAS Environment Manager
Default	0
Range	0–20000
Note	This option is valid in 2025.03 and later.
See	MAXCORESPERWORKER= Session Option in *SAS Cloud Analytic Services: User's Guide*
Example	In this example, each session defaults to a session option having a value of 7. `cas.maxcoresperworker=7`

A new server option cas.MAXCORESPERWORKER was introduced in the stable version of March 2025, 2025.03.
It specifies the maximum number of cores per worker.
Reducing the maximum number of cores per worker can help reduce contention between cores and improve performance.
This is especially true when multiple users are running highly concurrent actions on the CAS server.
By specifying this option, you can change the default maximum for all sessions on your CAS server.
Note that in the documentation you can find a script example, to measure how the MAXCORESPERWORKER value can affect the computing elapsed time for you specific program. You can see a screenshot, here, of the results for a simple SUMMARY action.
Of course the results can vary significantly, depending on the number of cores that are available and the number of concurrent jobs that are running on your CAS server when the program is executed, but it can be very useful, for example to optimize CAS batch processing.



CAS Server Settings

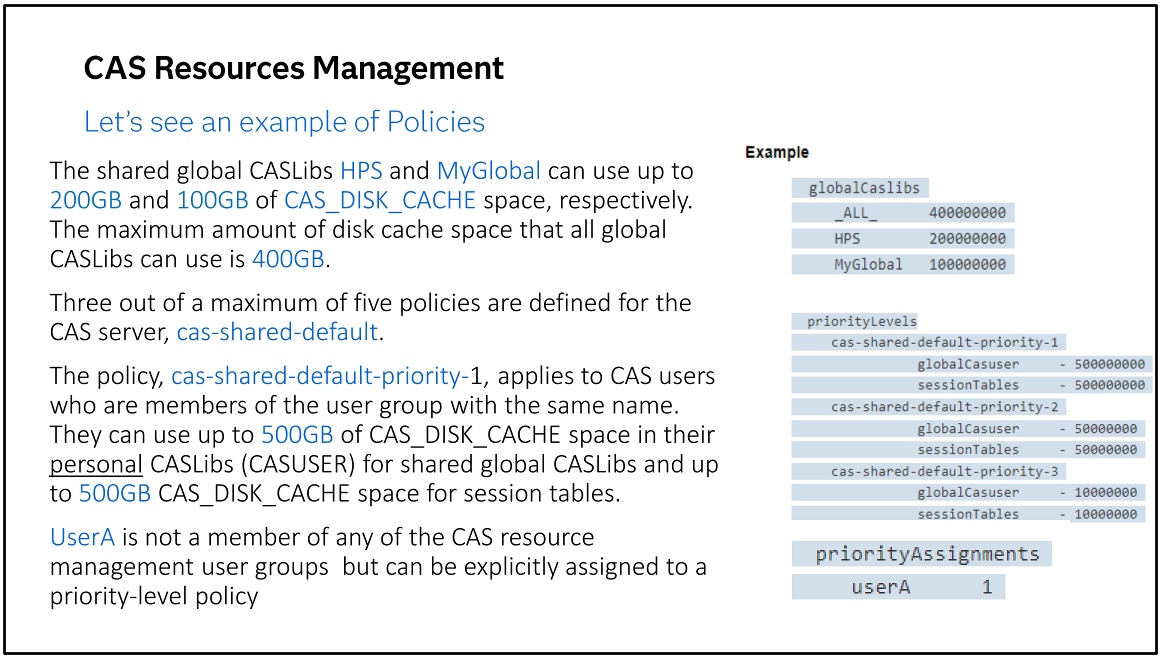
CASLibs Quota Policies

A SAS administrator can use the SAS Viya command line interface to create quotas policies on the CASLibs tables :

- “Global caslibs” (globalCaslibs)
  - One policy per CAS server that is used to place space quotas on global CASLibs.
- “Priority-level” (CAS-server-name-priority-n)
  - Up to five “policies” per CAS server that is used to place space quotas on users’ table data (“globalCASuser” and “sessionTables”).
  - The policies can be associated to groups of users or individual users

So, what are the CASLib quotas policies ?
It was already there with Viya 3, but it's basically a way to set some quotas on the utilization of the CAS tables.
So, there are two types of policies.
You have the global CASLib policy, and you can define only one for each CAS server. It allows you to place some space quotas on specific global CASLibs.
We will see an example of that.
Then the second type of policy is a priority level.
You can define up to five policies for each CAS server, the name of the policy must be associated to the CAS Server name.
That will be used to define space quotas for group of users, and it will limit the usage of the users’ personal tables data,
This type of quota policies can be associated to a group of users or to individual users.

A SAS administrator can use the SAS Viya command line (or CLI) interface to create quotas policies on the CAS tables and the policies are then stored in the SAS Configuration Server (aka Consul)



CAS Resources Management

Let’s see an example of Policies

The shared global CASLibs HPS and MyGlobal can use up to 200GB and 100GB of CAS_DISK_CACHE space, respectively. The maximum amount of disk cache space that all global CASLibs can use is 400GB.

Three out of a maximum of five policies are defined for the CAS server, cas-shared-default.

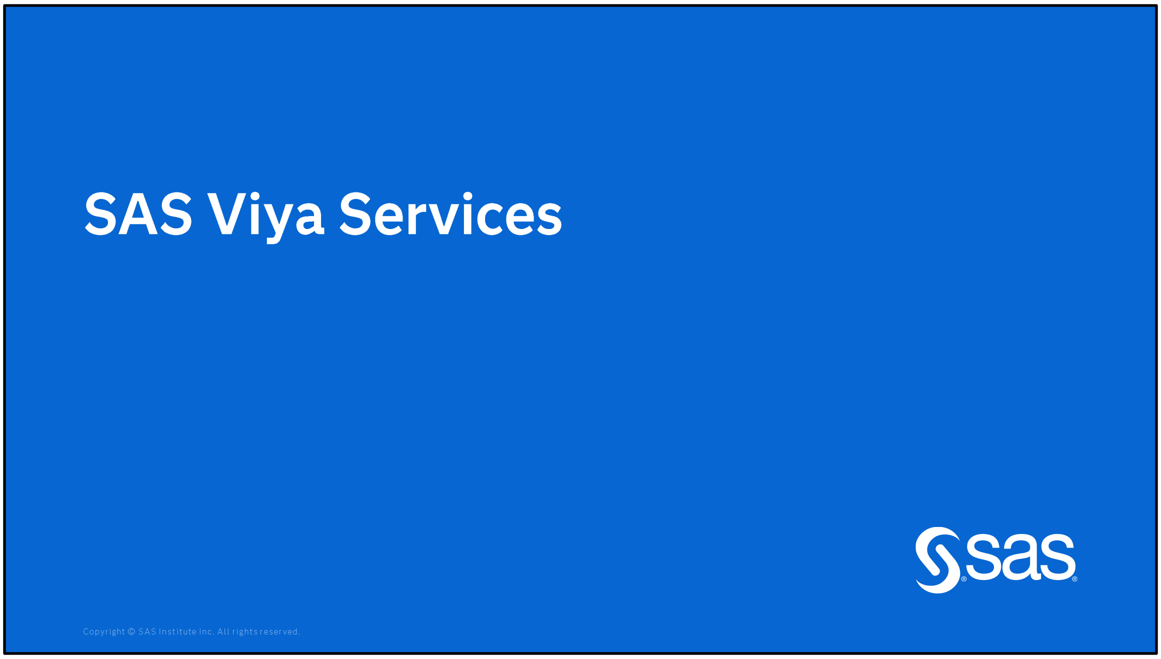
The policy, cas-shared-default-priority-1, applies to CAS users who are members of the user group with the same name. They can use up to 500GB of CAS_DISK_CACHE space in their personal CASLibs (CASUSER) for shared global CASLibs and up to 500GB CAS_DISK_CACHE space for session tables.

UserA is not a member of any of the CAS resource management user groups but can be explicitly assigned to a priority-level policy

Example

```
globalCaslibs
    _ALL_       400000000
    HPS         200000000
    MyGlobal    100000000

priorityLevels
    cas-shared-default-priority-1
                  globalCasuser     - 500000000
                  sessionTables     - 500000000
    cas-shared-default-priority-2
                  globalCasuser     - 50000000
                  sessionTables     - 50000000
    cas-shared-default-priority-3
                  globalCasuser     - 10000000
                  sessionTables     - 10000000

priorityAssignments
    userA        1
```

Here is an example where we have two shared global CASlibs “HPS” and “MyGlobal”, which can use up to 200 gigabytes and 100 gigabytes, respectively,
of CAS disk cache space.
And the maximum overall amount of disk cache space for all the global CASlibs is 400 gigabytes.
We can set limits for specific global CASlibs, and also set a global limit that applies to all global CASLibs.
Then if we look at the other kind of policy, priority levels, we can see that we have three out of a maximum of five policies that apply here for the “cas-shared-default” CAS server.
Remember that you could have multiple CAS servers in your Viya deployment , while “cas-shared-default” is the default name for the CAS server you could have other CAS Servers with different names.
The first policy, “priority one” applies to the CAS users who are members of a user group with the same name.
So, you need to create a user group with the same name, “cas dash shared dash default dash priority dash one”
If you are a member of this group, then you are limited in your usage of the CAS disk cache space, for your own personal CASUSER Caslib (CAS USER) and also for the temporary session tables.
In this example, we see that there are three different policies.
So, the “priority dash one” is for the users who have the highest amount of space available for their tables.
And then priority three, it's limited. Also, you can define a specific user and assign it directly to one of those priority policy points, okay?
So those kind of policies, as of today, can only be set with the Viya CLI.
And there were some blogs that were written at the time of SAS Viya 3 about them, you can find them in the Reference section of the notes.



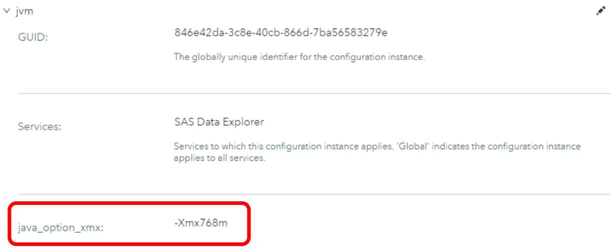
SAS Viya Services

SAS Viya Resource Management

The SAS Viya services (stateless, stateful, compute) have similar configuration items as in Viya 3.x (exposed in SAS Environment Manager.)

Some Java services have NOT been converted to Go and as a result it is possible to set the well-known Java runtime settings (-Xmx, -Xms, …)

These values should only be adjusted if required



jvm

GUID: 846e42da-3c8e-40cb-866d-7ba56583279e

The globally unique identifier for the configuration instance.

Services: SAS Data Explorer

Services to which this configuration instance applies. 'Global' indicates the configuration instance applies to all services.

java_option_xmx: -Xmx768m

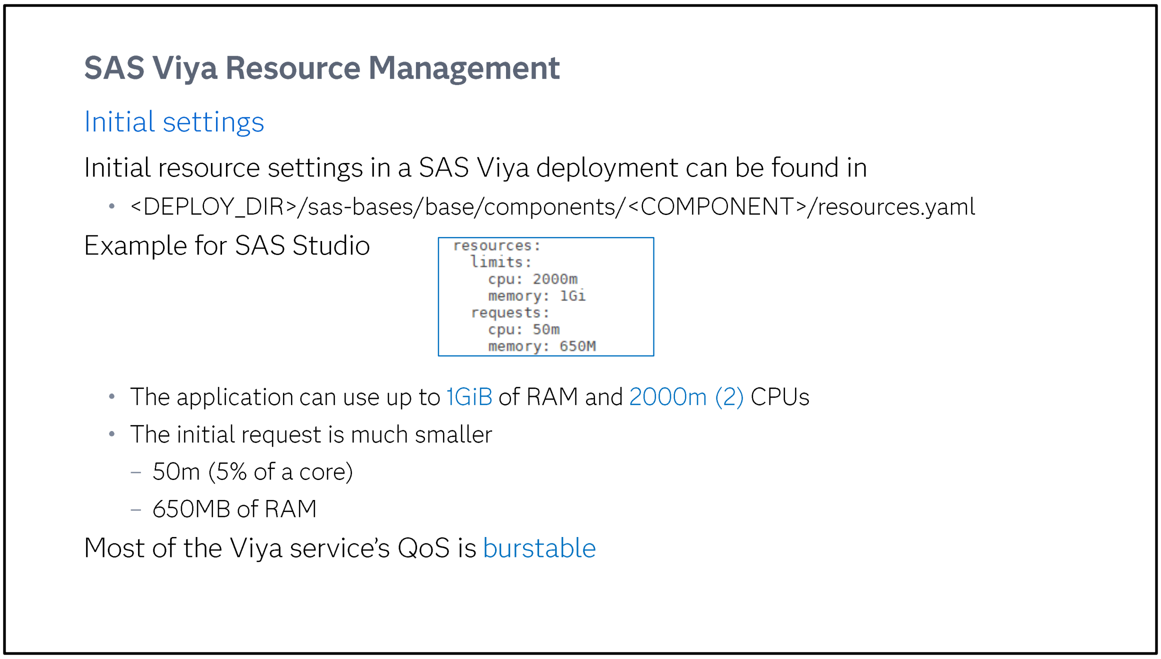
SAS Viya services—whether stateless, stateful, or compute-based—include configuration options that are quite similar to what was available in SAS Viya 3.x. These settings are all accessible through SAS Environment Manager.

When it comes to Java-based applications—like SAS Logon or SAS Authorization—you can configure familiar Java runtime settings, such as XMX and XMS.

That said, it's important to proceed with caution. The values you assign to XMX and XMS must align with the container's resource limits. Here's why.

Let's say you set the initial Java heap size, the XMS value, to 4 gigabytes. But your container is only allowed 1 gigabyte of memory. In that case, the pod is likely to be terminated shortly after startup. Why? Because the Linux Out-of-Memory Killer will be triggered by the kubelet, and it will shut the pod down.

Just as a reminder, XMX defines the maximum memory that the Java Virtual Machine can allocate, while XMS defines the initial amount of memory it starts with.
So always be sure your Java memory settings are in sync with your container's resource configuration.



SAS Viya Resource Management

Initial settings

Initial resource settings in a SAS Viya deployment can be found in

- <DEPLOY_DIR>/sas-bases/base/components/<COMPONENT>/resources.yaml

Example for SAS Studio

```
resources:
  limits:
    cpu: 2000m
    memory: 1Gi
  requests:
    cpu: 50m
    memory: 650M
```

- The application can use up to 1GiB of RAM and 2000m (2) CPUs
- The initial request is much smaller
  - 50m (5% of a core)
  - 650MB of RAM

Most of the Viya service's QoS is burstable

Let's talk about how initial resource settings are handled in a Viya deployment.

The starting point for service-level resource requests and limits—like how much CPU or memory a service can use—is defined in the deployment asset files. If you navigate to the sas-bases/base/components directory, you'll find a list of subdirectories. Each one corresponds to a different Viya service.

Inside each of those, you'll see a file called resources.yaml. That file contains the default configuration settings for CPU and memory—both the requests and the limits.

Take SAS Studio as an example. Its default limit is set to 2000 millicores—that's two CPU cores—and one gigabyte of RAM.

Now here's something important: most Viya services fall under what's called the burstable Quality of Service category. This means that at least one container in the pod has a resource request defined—either for memory, CPU, or both. Because of that, the pod is guaranteed a minimum level of resources, but it can also use more if the cluster has them available.

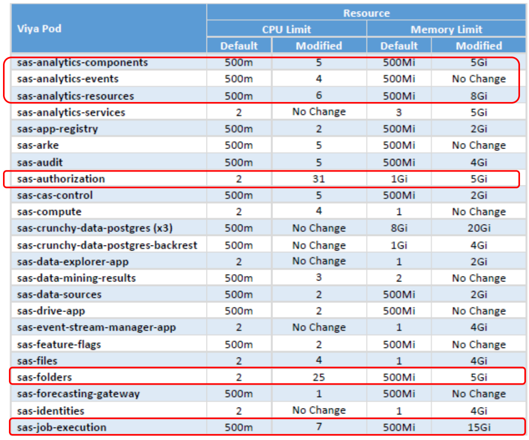
This model helps balance resource efficiency with performance, especially in shared environments where workload demands can vary.

SAS Viya Resource Management

Tuning resource limits

The default resource requests and limits may need to be tuned for the expected workload and number of users.

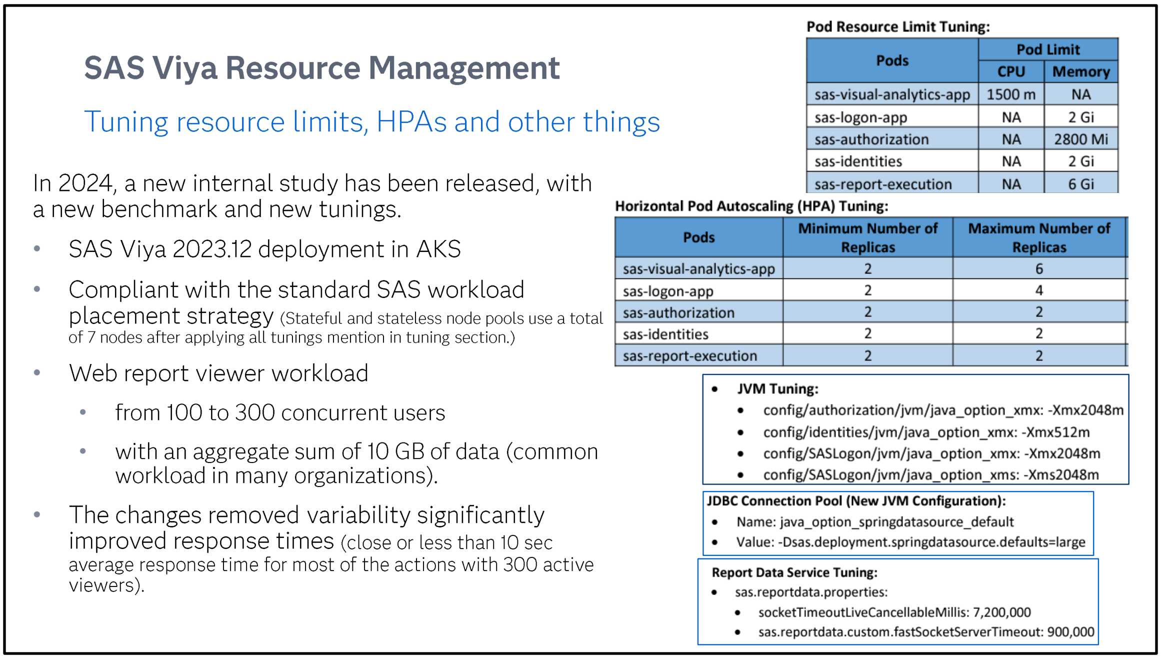
- Extract from a Viya 4 Case study paper published in September 2021 by the *SAS EPD Division*.
- SAS Viya 4 platform (2021.1.2, SAS Visual Data Science bundle, Large T-Shirt sizing)
- SAS Visual Analytics use case
  - incremental user concurrency up to 200 browser-based users (CAS light actions)
  - and 1000 REST API users (HTTP calls : login, browse, logout).



Viya Pod	Resource			
	CPU Limit		Memory Limit	
	Default	Modified	Default	Modified
sas-analytics-components	500m	5	500Mi	5Gi
sas-analytics-events	500m	4	500Mi	No Change
sas-analytics-resources	500m	6	500Mi	8Gi
sas-analytics-services	2	No Change	3	5Gi
sas-app-registry	500m	2	500Mi	2Gi
sas-arke	500m	5	500Mi	No Change
sas-audit	500m	5	500Mi	4Gi
sas-authorization	2	31	1Gi	5Gi
sas-cas-control	500m	5	500Mi	2Gi
sas-compute	2	4	1	No Change
sas-crunchy-data-postgres (x3)	500m	No Change	8Gi	20Gi
sas-crunchy-data-postgres-backrest	500m	No Change	1Gi	4Gi
sas-data-explorer-app	2	No Change	1	2Gi
sas-data-mining-results	500m	3	2	No Change
sas-data-sources	500m	2	500Mi	2Gi
sas-drive-app	500m	2	500Mi	No Change
sas-event-stream-manager-app	2	No Change	1	4Gi
sas-feature-flags	500m	2	500Mi	No Change
sas-files	2	4	1	4Gi
sas-folders	2	25	500Mi	5Gi
sas-forecasting-gateway	500m	1	500Mi	No Change
sas-identities	2	No Change	1	4Gi
sas-job-execution	500m	7	500Mi	15Gi

SAS Viya comes with default, out-of-the-box settings for pod resource requests and limits. But in many cases, these default values need to be tuned to better match the expected workload and number of users.
Back in September 2021, the SAS performance and benchmarking team published a great internal study that focused on this exact topic. The scenario they explored involved a SAS Visual Analytics workload, with up to 200 users interacting through a web browser, and another 1,000 batch requests coming in through the API.

Now, the study included examples of changes made to CPU and memory limits for specific containers. But that's just part of the story. What makes the study so helpful is that it also dives into other important adjustments.
For example, they had to increase the number of pod replicas for certain services. They also had to tune Java settings—like the Java heap size using JAVA_OPTION_XMX—and update connection pool limits for Tomcat data sources.
So, while the defaults are a great starting point, real-world usage often demands a bit more fine-tuning. If you're scaling for larger teams or heavier loads, that paper is definitely worth reviewing.



SAS Viya Resource Management

Tuning resource limits, HPAs and other things

In 2024, a new internal study has been released, with a new benchmark and new tunings.

- SAS Viya 2023.12 deployment in AKS
- Compliant with the standard SAS workload placement strategy (Stateful and stateless node pools use a total of 7 nodes after applying all tunings mention in tuning section.)
- Web report viewer workload
  - from 100 to 300 concurrent users
  - with an aggregate sum of 10 GB of data (common workload in many organizations).
- The changes removed variability significantly improved response times (close or less than 10 sec average response time for most of the actions with 300 active viewers).

Pod Resource Limit Tuning:

Pods	Pod Limit	
	CPU	Memory
sas-visual-analytics-app	1500 m	NA
sas-logon-app	NA	2 Gi
sas-authorization	NA	2800 Mi
sas-identities	NA	2 Gi
sas-report-execution	NA	6 Gi

Horizontal Pod Autoscaling (HPA) Tuning:

Pods	Minimum Number of Replicas	Maximum Number of Replicas
sas-visual-analytics-app	2	6
sas-logon-app	2	4
sas-authorization	2	2
sas-identities	2	2
sas-report-execution	2	2

- **JVM Tuning:**
  - config/authorization/jvm/java_option_xmx: -Xmx2048m
  - config/identities/jvm/java_option_xmx: -Xmx512m
  - config/SASLogon/jvm/java_option_xmx: -Xmx2048m
  - config/SASLogon/jvm/java_option_xms: -Xms2048m

JDBC Connection Pool (New JVM Configuration):

- Name: java_option_springdatasource_default
- Value: -Dsas.deployment.springdatasource.defaults=large

Report Data Service Tuning:

- sas.reportdata.properties:
  - socketTimeoutLiveCancellableMillis: 7,200,000
  - sas.reportdata.custom.fastSocketServerTimeout: 900,000

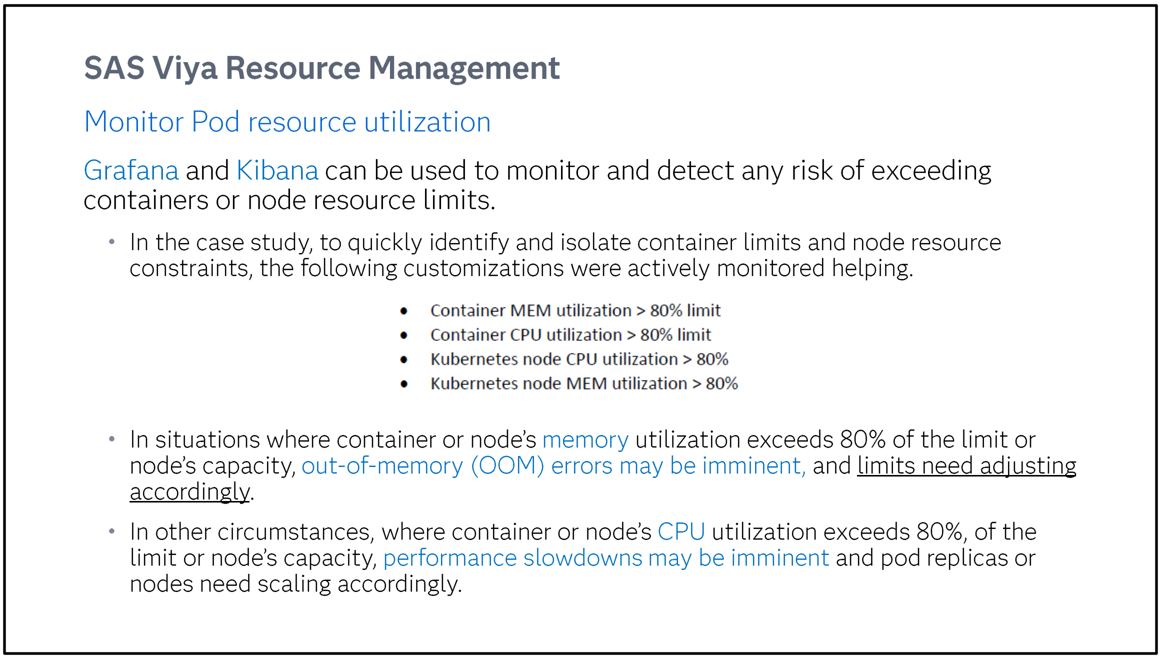
In a more recent internal performance study, several tests and benchmarks were conducted to identify resource bottlenecks in a different scenario—this time involving between 100 and 300 concurrent users accessing reports through the Web Report Viewer. These reports were based on a total of 10 gigabytes of datasets.
During the tests, the performance team noticed CPU and memory issues affecting multiple microservices. These included the SAS Visual Analytics app, the SAS Logon app, SAS Authorization, SAS Identities, and SAS Report Execution.

The first step was to adjust the pod resource limits for those services.
This helped improve response times, but the team still observed some variability. In particular, the maximum response times were a bit too high.

To address this, they turned their attention to the Horizontal Pod Autoscaling settings, tuning the HPA for each of the microservices that had been flagged.
This combination—tuning pod limits along with HPA—did a lot to smooth out the response times. But the improvements didn't stop there.

The team also had to make sure that the Java Virtual Machine settings were consistent with the new memory configurations. Specifically, they aligned the XMS and XMX values to match the adjusted pod memory limits.

And finally, with additional tuning for the JDBC connection pool and the Report Data Service, the overall response times improved significantly.
It's a great example of how layered tuning—resources, autoscaling, JVM, and service-specific configurations—can make a real impact on performance.



SAS Viya Resource Management

Monitor Pod resource utilization

Grafana and Kibana can be used to monitor and detect any risk of exceeding containers or node resource limits.

- In the case study, to quickly identify and isolate container limits and node resource constraints, the following customizations were actively monitored helping.
  - Container MEM utilization > 80% limit
  - Container CPU utilization > 80% limit
  - Kubernetes node CPU utilization > 80%
  - Kubernetes node MEM utilization > 80%
- In situations where container or node's memory utilization exceeds 80% of the limit or node's capacity, out-of-memory (OOM) errors may be imminent, and limits need adjusting accordingly.
- In other circumstances, where container or node's CPU utilization exceeds 80%, of the limit or node's capacity, performance slowdowns may be imminent and pod replicas or nodes need scaling accordingly.

Monitoring and logging tools like Grafana and Kibana can also play a key role in identifying potential resource issues before they impact performance.

During the benchmark studies, the team set up specific dashboards and alerts to help detect when containers or nodes were approaching their resource limits. These visualizations made it easier to spot where more CPU or memory might be needed.

For example, by using custom dashboards and targeted alerts, teams can proactively adjust memory limits, increase the number of replicas, or even scale up nodes. This gives you a much better chance of making adjustments before any performance degradation or service availability issues occur.

SAS Viya Resource Management

Avoid OOM issues on CAS pods

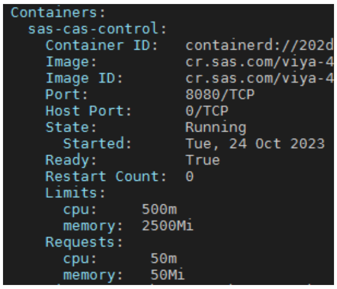
The sas-cas-control pod is a stateless service whose purpose is to assist CAS.

Supporting many concurrent users can cause the sas-cas-control service to throw an out-of-memory (OOM) error.

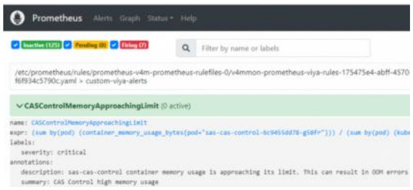
The memory limit default value for the sas-cas-control pod was increased from 500 to 2500 MiB

It is recommended to include the sas-cas-control pod in the monitoring configuration.

- Setting an alert that fires when the memory consumption for that pod exceeds 80% of its memory limit should help detect and prevent OOM problems with user concurrency.



```
Containers:
  sas-cas-control:
    Container ID:   containerd://202d
    Image:          cr.sas.com/viya-4
    Image ID:       cr.sas.com/viya-4
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 24 Oct 2023
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     500m
      memory:  2500Mi
    Requests:
      cpu:      50m
      memory:   50Mi
```



/etc/prometheus/rules/prometheus-v4m-prometheus-rulefiles-0/v4mmon-prometheus-viya-rules-175475e4-abff-4570-f6f934c5790c.yaml > custom-viya-alerts

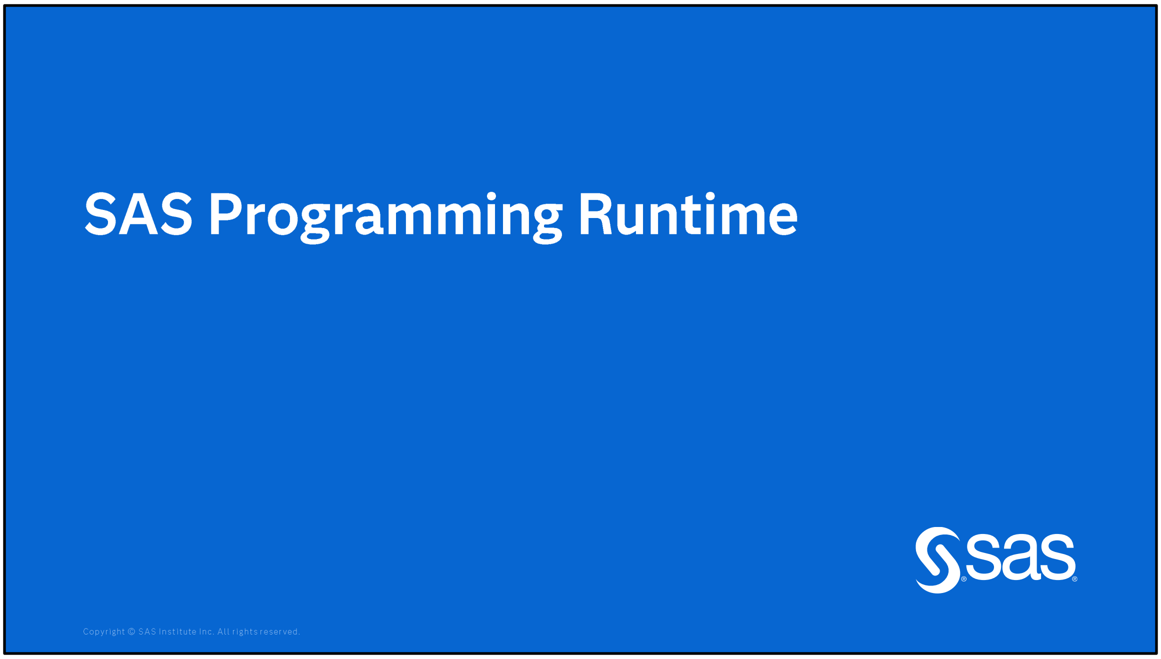
CASControlMemoryApproachingLimit (0 active)

```
name: CASControlMemoryApproachingLimit
expr: (sum by(pod) (container_memory_usage_bytes{pod="sas-cas-control-6c9455dd78-g50fr"})) / (sum by(pod) (kub
labels:
  severity: critical
annotations:
  description: sas-cas-control container memory usage is approaching its limit. This can result in OOM errors
  summary: CAS Control high memory usage
```

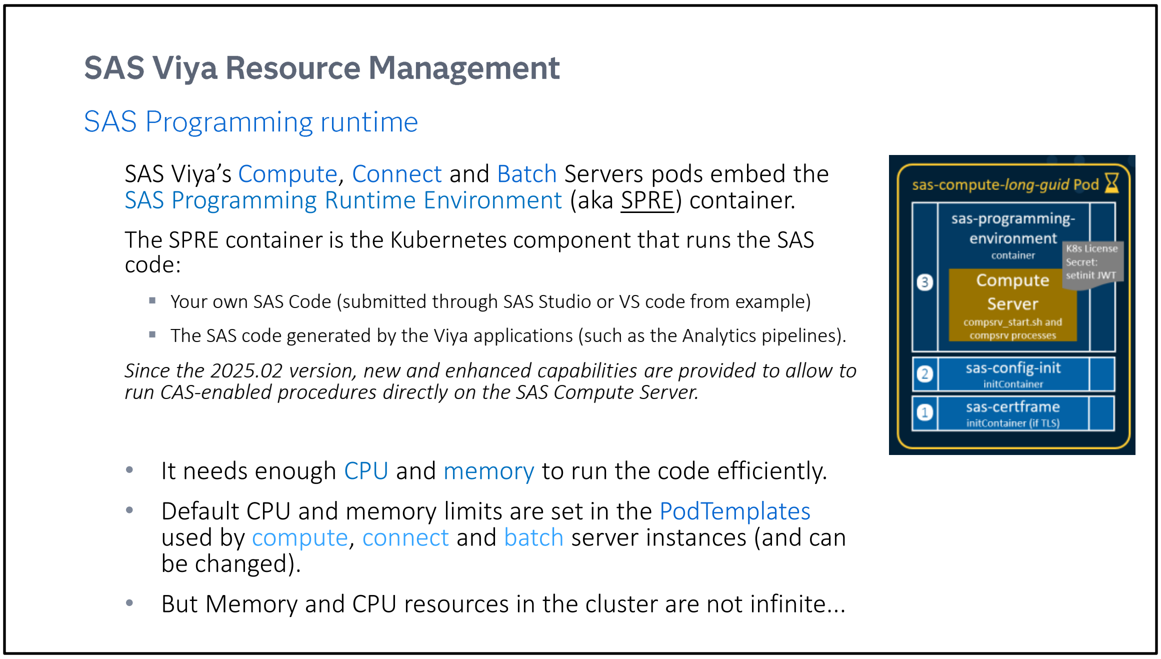
And finally, let's talk about the sas-cas-control pod.
This is a stateless service designed to assist CAS, and it plays a very important role in the Viya deployment. It serves as the connection point that allows applications like Visual Analytics, Model Studio, and others to reach the CAS servers. It also manages the CAS servers themselves, including session handling, library assignments, and permission controls for all CAS-related resources.

Why does this matter?

Because it's been observed that the sas-cas-control pod can become a bottleneck when the CAS server is under heavy load, particularly when many CAS sessions are being launched at once. In environments where this kind of load is expected, it's a good idea to include this pod in your monitoring setup. If necessary, increase its resource limits to avoid slowdowns.
That said, recent versions of Viya already ship with raised default limits. The pod is now configured to use up to 500 millicores—that's half a CPU core—and 2.5 gigabytes of memory. This improvement helps, but in high-demand situations, further adjustments might still be needed.



SAS Programming Runtime



SAS Viya Resource Management

SAS Programming runtime

SAS Viya's Compute, Connect and Batch Servers pods embed the SAS Programming Runtime Environment (aka SPRE) container.

The SPRE container is the Kubernetes component that runs the SAS code:

- Your own SAS Code (submitted through SAS Studio or VS code from example)
- The SAS code generated by the Viya applications (such as the Analytics pipelines).

Since the 2025.02 version, new and enhanced capabilities are provided to allow to run CAS-enabled procedures directly on the SAS Compute Server.

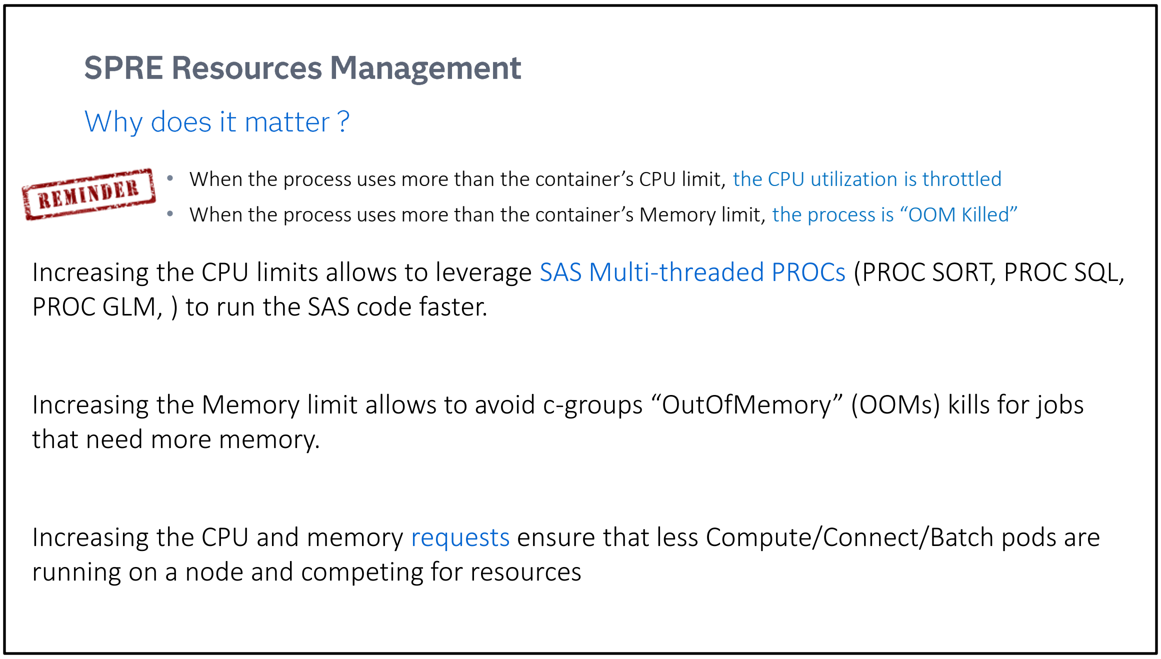
sas-compute-*long-guid* Pod
sas-programming-environment
container
3
Compute Server
compsrv_start.sh and compsrv processes
K8s License Secret: setinit JWT
2
sas-config-init
initContainer
1
sas-certframe
initContainer (if TLS)

- It needs enough CPU and memory to run the code efficiently.
- Default CPU and memory limits are set in the PodTemplates used by compute, connect and batch server instances (and can be changed).
- But Memory and CPU resources in the cluster are not infinite...

So SAS Viya's compute, connect, and batch servers pods embed the “SAS Programming Runtime Environment” container, also sometimes called SPRE.
So this SAS Programming Runtime Environment container is the one where the SAS code is executed.
It can run the SAS code that you submit through SAS Studio or VS Code, for example, or it can be the SAS code that is generated by other SAS Viya applications.
For example, when you run the Analytics Pipeline in Model studio, behind the scene, it will start up one or several compute server sessions.

Starting in SAS Viya 2025.02, major enhancements to the SAS Compute Server simplify SAS programming by enabling advanced analytics procedures to run directly on the Compute Server—eliminating the need for a CAS server in many cases.
These updates introduce in-process execution, multithreading support, and expanded native Python integration, providing greater flexibility and efficiency.

So we can expect intensive processing to happen in this sas-programming-environment container.
And this specific container will need to have access to enough CPU and memory to run the SAS code efficiently.
Default CPU and memory limits are set for the container in the pod templates used by the compute, connect, and batch server instances.
And they can be changed.



SPRE Resources Management

Why does it matter ?

REMINDER

- When the process uses more than the container’s CPU limit, the CPU utilization is throttled
- When the process uses more than the container’s Memory limit, the process is “OOM Killed”

Increasing the CPU limits allows to leverage SAS Multi-threaded PROCs (PROC SORT, PROC SQL, PROC GLM,) to run the SAS code faster.

Increasing the Memory limit allows to avoid c-groups “OutOfMemory” (OOMs) kills for jobs that need more memory.

Increasing the CPU and memory requests ensure that less Compute/Connect/Batch pods are running on a node and competing for resources

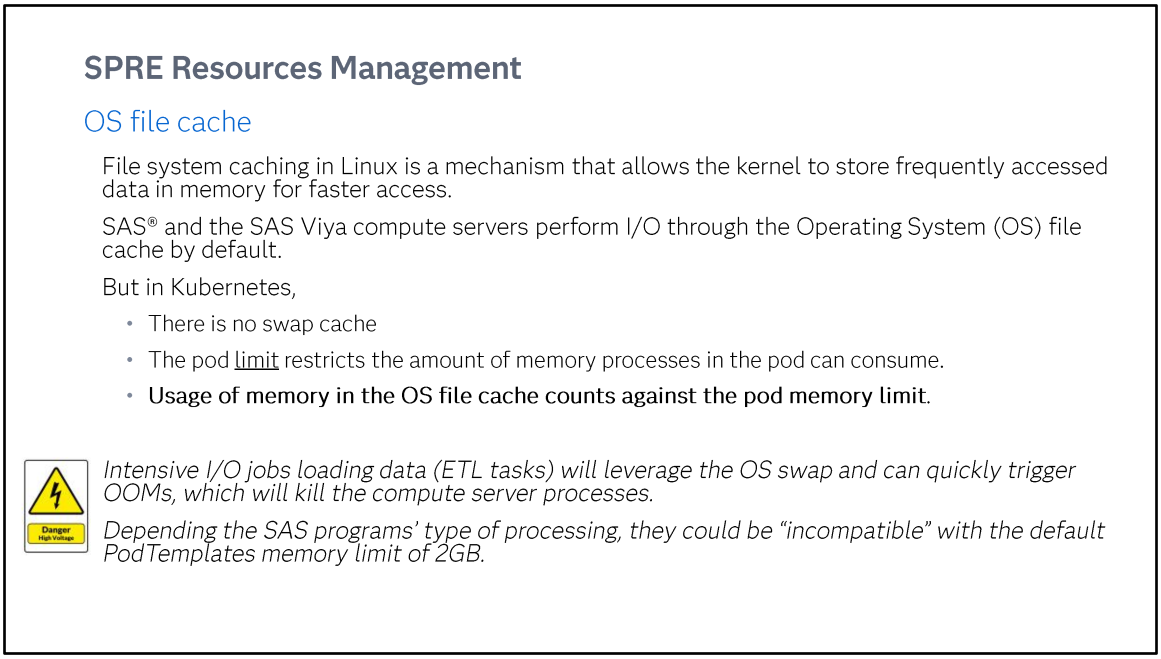
Why is it important to tune the SAS Programming runtime container CPU and memory request and limits?
You remember that in Kubernetes, when the process uses more than the container CPU limit, then the CPU utilization is throttled.
It's done through C-groups.
Also, when the process uses more than the container's memory limit, the system kernel will terminate the process that attempted the allocation
with an out-of-memory error. The kubelet and the container runtimes, which configure the kernel C-groups, are enforcing these limits in Kubernetes.

Increasing the CPU limit allows to leverage SAS multi-threaded procedures like PROC SORT, PROC SQL, PROC GLM, and others.
There is a link in the notes that sends you to the official documentation that lists the supported threaded procedure.
So, it's good to increase the CPU limits if you are running those kind of SAS procedures because it will allow the SAS code to run faster.

Also increasing the memory limit reduces the risk to have those out-of-memory killer situations when the SAS program of the job that is running needs more memory than the defined limit.

Finally, Increasing the CPU and memory requests ensure that less Compute/Connect/Batch pods are running on a node and competing for resources, which means that there are less risks to see a node accepting too many pods and running out of resources in case all compute sessions are asking for more resources at the same time.
It's a way to kind of limit the number of SAS programming runtime environment instances on the same

machine.
It's a way to prevent all the compute servers to exhaust the node.



SPRE Resources Management

OS file cache

File system caching in Linux is a mechanism that allows the kernel to store frequently accessed data in memory for faster access.

SAS® and the SAS Viya compute servers perform I/O through the Operating System (OS) file cache by default.

But in Kubernetes,

- There is no swap cache
- The pod limit restricts the amount of memory processes in the pod can consume.
- Usage of memory in the OS file cache counts against the pod memory limit.

Intensive I/O jobs loading data (ETL tasks) will leverage the OS swap and can quickly trigger OOMs, which will kill the compute server processes.

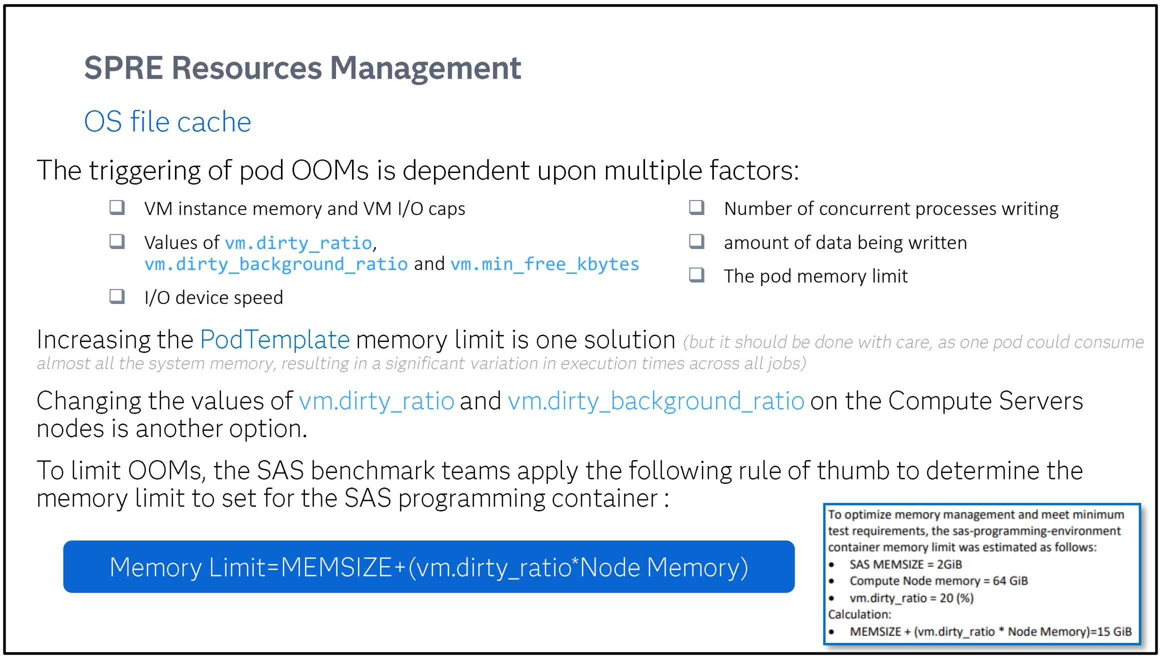
Depending the SAS programs' type of processing, they could be "incompatible" with the default PodTemplates memory limit of 2GB.

Now there is something to understand about this "SAS Programming runtime" container.
It contains a SAS legacy engine which natively exploits caching of files on the host for better performance.

On Linux systems, the file system caching is a mechanism that allows the kernel to store frequently accessed data in memory for faster access.
And the SAS programming runtime leverages that.
Basically, it means that during I/O operations, instead of writing the data blocks on disk immediately, the system defer those write operations and keep the data blocks in memory as it will allow to speed up the next operations that maybe use those data blocks.

But unfortunately, it is not something that Kubernetes will really take into account…
Inside the container, the data blocks that are kept in memory will count against the container's memory limit (which is usually pretty limited).

Therefore, intensive I/O jobs loading data, for example, ETL tasks (extract transform loading), will leverage the OS swap and can quickly trigger those out-of-memory terminations which will kill the compute server processed.
So, depending on the SAS program's type of processing, the default Pod template's memory of 2 GigaBytes might not be enough…



SPRE Resources Management

OS file cache

The triggering of pod OOMs is dependent upon multiple factors:

- ❑ VM instance memory and VM I/O caps
- ❑ Values of `vm.dirty_ratio`, `vm.dirty_background_ratio` and `vm.min_free_kbytes`
- ❑ I/O device speed
- ❑ Number of concurrent processes writing
- ❑ amount of data being written
- ❑ The pod memory limit

Increasing the PodTemplate memory limit is one solution *(but it should be done with care, as one pod could consume almost all the system memory, resulting in a significant variation in execution times across all jobs)*

Changing the values of vm.dirty_ratio and vm.dirty_background_ratio on the Compute Servers nodes is another option.

To limit OOMs, the SAS benchmark teams apply the following rule of thumb to determine the memory limit to set for the SAS programming container :

Memory Limit=MEMSIZE+(vm.dirty_ratio*Node Memory)

To optimize memory management and meet minimum test requirements, the sas-programming-environment container memory limit was estimated as follows:

- SAS MEMSIZE = 2GiB
- Compute Node memory = 64 GiB
- vm.dirty_ratio = 20 (%)

Calculation:

- MEMSIZE + (vm.dirty_ratio * Node Memory)=15 GiB

It's hard to say at which point the Compute servers will be terminated by the out-of-memory killer because it depends on many factors.
Such as the VM instance memory and VM I/O caps, the values of VM dirty ratio, the speed of the IO device, the number of concurrent processes writing, the amount of data being written, and the pod memory limit.
Those VM dirty ratio and VM dirty ratio background settings actually influences the behavior of the file cache memory usage for “write behind I/O” and also the aggressiveness of flushing write behind.
To limit the risk of terminated compute servers, there are basically two solutions.

One is to increase the podTemplate memory limit. However, it should be done carefully because you could end up with one or few pods consuming almost all the system memory on the node. It could result in a significant variation in execution times across all jobs, and it could also exhaust the underlying node.

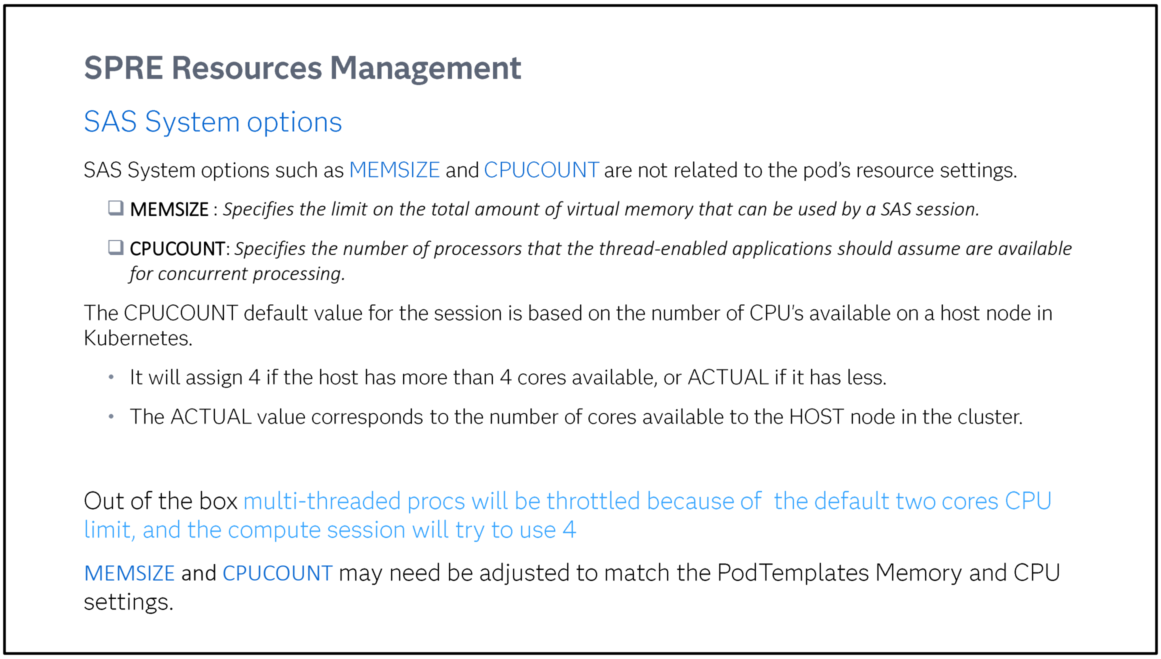
The other solution is to change the value of the VM dirty ratio and VM dirty background ratio on the underlying compute server nodes. That's another possibility.
Internally at SAS, the benchmark team has been using a formula to determine the right memory limit values.

This formula is based on the SAS MEMSIZE (which is the legacy SAS system option configured for the SAS programming runtime)

The addition of the MEMSIZE and the VM Dirty ratio multiplied by the node memory indicates the recommended memory limit.
If we take an example here where the SAS MEMSIZE has a value of two Gigabytes, and where we are using a compute node machine

that has 64 gigs of memory with a standard VM Dirty ratio of 20%, then calculation of this formula gives us 15 Gigabytes.

So, you would, in this case, change the default memory limit for the SAS programming container from two Gigabytes to 15 Gigabytes.



SPRE Resources Management

SAS System options

SAS System options such as MEMSIZE and CPUCOUNT are not related to the pod's resource settings.

- ❑ MEMSIZE : *Specifies the limit on the total amount of virtual memory that can be used by a SAS session.*
- ❑ CPUCOUNT: *Specifies the number of processors that the thread-enabled applications should assume are available for concurrent processing.*

The CPUCOUNT default value for the session is based on the number of CPU's available on a host node in Kubernetes.

- It will assign 4 if the host has more than 4 cores available, or ACTUAL if it has less.
- The ACTUAL value corresponds to the number of cores available to the HOST node in the cluster.

Out of the box multi-threaded procs will be throttled because of the default two cores CPU limit, and the compute session will try to use 4

MEMSIZE and CPUCOUNT may need be adjusted to match the PodTemplates Memory and CPU settings.

Another item of interest that is specific to the "SAS Programming runtime" container is the SAS system options such as MEMSIZE and CPUCOUNT,
which can also influence the amount of memory and CPU used by the SAS code running inside the container.
The SAS MEMSIZE specifies the limit on the total amount of virtual memory that can be used by a SAS session.
CPUCOUNT specifies the number of processors that the thread-enabled applications, like PROC SORT, PROC SQL, PROC SUMMARY or PROC GLM, should assume are available for parallel processing.
Basically, it's the number of cores that the SAS system considers for multithreading the activities, when those specific threaded PROCs are used.
The CPUCOUNT default value will be:
either 4
or the number of CPU on the host if it is less than 4.
But…you can set your own values for the CPUCOUNT. You can set it to something like 6 or 8 if you want, or you can let it take the value that corresponds to the number of CPU available on the underlying hosts if you use the "ACTUAL" key term as the value.
If you say CPUCOUNT equals "ACTUAL", and if your host machine has 64 cores, then the CPUCOUNT will be 64 cores and then each running pod or each running SAS programming container will consider that it can use up to 64 cores to run those multi-threading procedures for example.
However, out of the box, multithreading procedures will be throttled because of the default two-core CPU limit (which is set at the container level).

So, when tuning the compute servers, the SAS MEMSIZE and CPU count would need to be adjusted to match the pod templates memory and CPU settings.

SPRE Resources Management

How to tune it ?

The SAS Launcher Service provides CPU and memory settings for the launched service (e.g. Compute Server from SAS Studio)

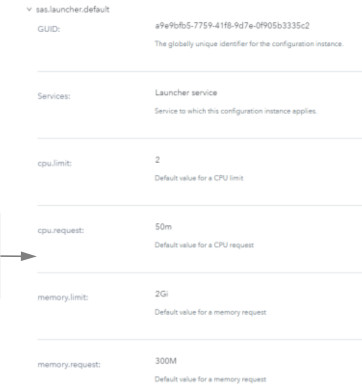
Defaults

- `cpu.limit  - 2 CPUs`
- `cpu.request – 50m`
- `memory.limit – 2Gi`
- `memory.request – 300M`

Can be changed in SAS Environment Manager

…Or by applying PodTemplate annotations

Applies to ALL launched pods



sas.launcher.default

GUID: a9e9bfb5-7759-41f8-9d7e-0f905b3335c2
The globally unique identifier for the configuration instance.

Services: Launcher service
Service to which this configuration instance applies.

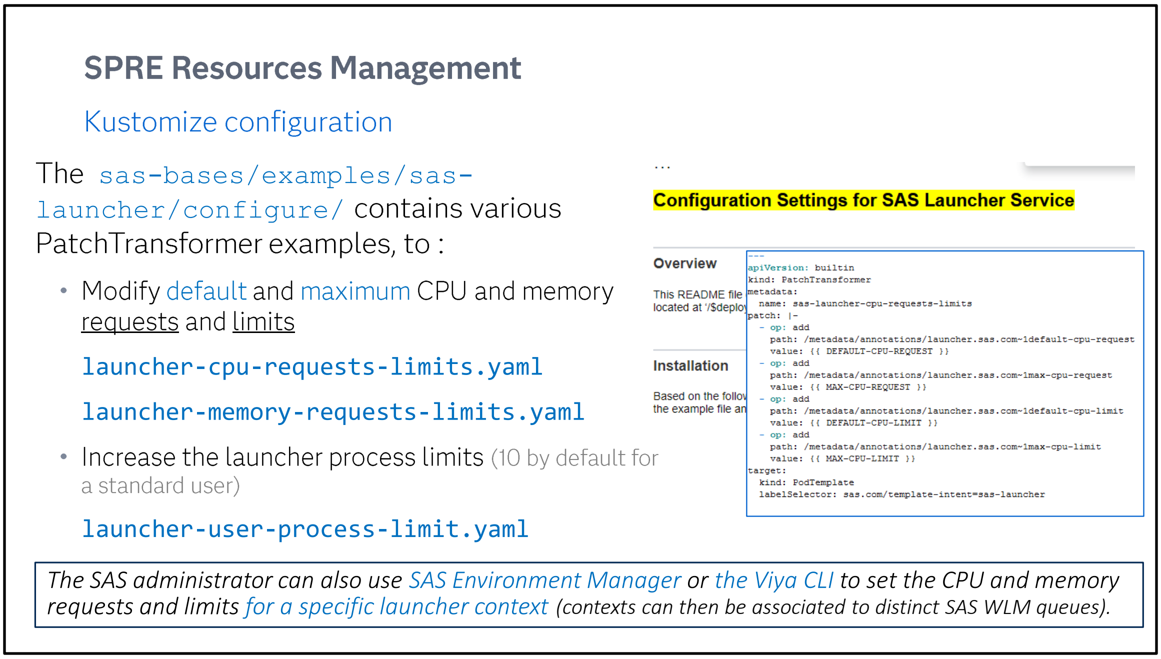
cpu.limit: 2
Default value for a CPU limit

cpu.request: 50m
Default value for a CPU request

memory.limit: 2Gi
Default value for a memory request

memory.request: 300M
Default value for a memory request

We have talked about why we should tune the SAS programming container CPU and memory resource requests and limits.
Now, how do we do it?
It is the “SAS Launcher” service that controls the CPU and memory settings for the launched services like Compute Server.
The default value for the CPU request is 50 millicores, and for the memory request, it is 300 megabytes.
And then the limit, are two CPU or two cores and two Gigabytes for the memory limit, okay?
It might be enough, but it really depends on the kind of SAS programming workload that is expected in the environment.
You can change those values either in SAS Environment Manager or you can use “podTemplates” annotations (with a kustomize “PatchTransformer”).
When you change the values in Environment Manager, it will apply to all the launched pods.
So, whether it's a Connect Server pod or a Batch Server pod or Compute Server pod, it will be the same values that you have changed in Environment Manager.



SPRE Resources Management

Kustomize configuration

The `sas-bases/examples/sas-launcher/configure/` contains various PatchTransformer examples, to :

- Modify default and maximum CPU and memory requests and limits

  `launcher-cpu-requests-limits.yaml`

  `launcher-memory-requests-limits.yaml`

- Increase the launcher process limits (10 by default for a standard user)

  `launcher-user-process-limit.yaml`

...

Configuration Settings for SAS Launcher Service

Overview

This README file
located at '/$deploy

Installation

Based on the follow
the example file an

```
---
apiVersion: builtin
kind: PatchTransformer
metadata:
  name: sas-launcher-cpu-requests-limits
patch: |-
  - op: add
    path: /metadata/annotations/launcher.sas.com~1default-cpu-request
    value: {{ DEFAULT-CPU-REQUEST }}
  - op: add
    path: /metadata/annotations/launcher.sas.com~1max-cpu-request
    value: {{ MAX-CPU-REQUEST }}
  - op: add
    path: /metadata/annotations/launcher.sas.com~1default-cpu-limit
    value: {{ DEFAULT-CPU-LIMIT }}
  - op: add
    path: /metadata/annotations/launcher.sas.com~1max-cpu-limit
    value: {{ MAX-CPU-LIMIT }}
target:
  kind: PodTemplate
  labelSelector: sas.com/template-intent=sas-launcher
```

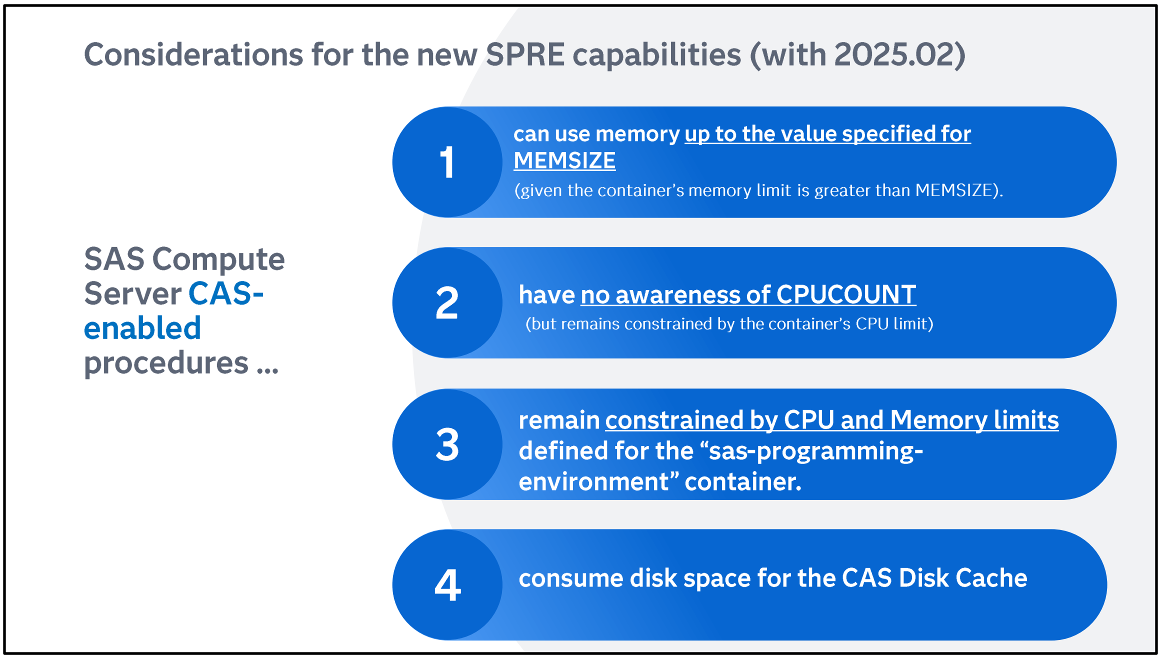
The SAS administrator can also use SAS Environment Manager or the Viya CLI to set the CPU and memory requests and limits for a specific launcher context (contexts can then be associated to distinct SAS WLM queues).

Now, if you use the pod templates patch transformers (and the location is provided here), you can target a specific type of launched pod.

You can target, for example just the Batch Server or Connect Server or just Compute Server sessions. You can do that. It's not in the example, but it's possible to do it by changing the “labelSelector” in the Target specification.
There is also a transformer to increase the launcher process limits, which is 10 by default for a standard user.
And finally, something that is very interesting is that you can also assign different CPU and memory limits for different contexts, for different “launcher contexts”.

So that's something that you can now do either with the Viya CLI or in Environment Manager.
What is also interesting is that you could then attach different contexts to distinct SAS Workload Management queues.

So, I have just presented two slides here in this module about configuring those things…
But you can find more details and explanation on how to configure the SAS launcher service in the GEL administration training material.
There is a programming runtime section, you can go there, and you will find video slides and exercise to do that.



Considerations for the new SPRE capabilities (with 2025.02)

SAS Compute Server CAS-enabled procedures …

1. can use memory up to the value specified for MEMSIZE (given the container's memory limit is greater than MEMSIZE).
2. have no awareness of CPUCOUNT (but remains constrained by the container's CPU limit)
3. remain constrained by CPU and Memory limits defined for the "sas-programming-environment" container.
4. consume disk space for the CAS Disk Cache

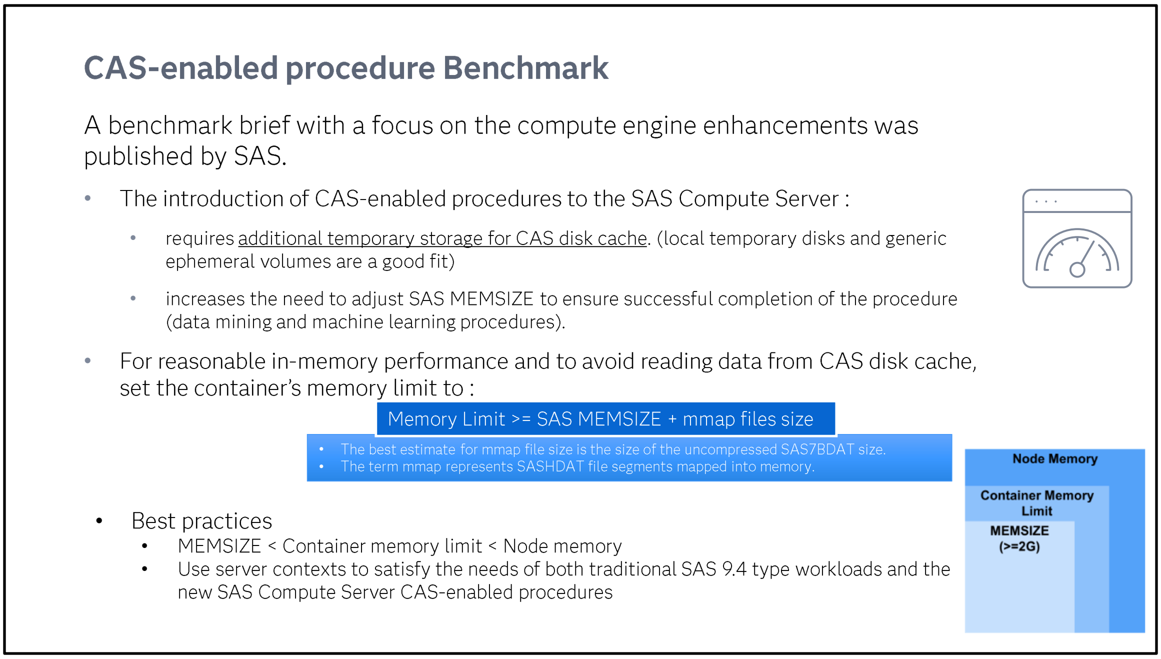
As noted at the beginning of this module, starting with the 2025.02 version, new capabilities have been added in the SAS programming runtime and allows to run CAS-enabled procedures directly on the SAS Compute Server.
Let's review some associated considerations.

While CAS is unaware of MEMSIZE, SAS Compute Server "CAS-enabled" procedures can only use memory up to the value specified for MEMSIZE.

The SAS option CPUCOUNT defaults to 4 CPU.
While SAS 9 procedures running in the SAS Compute Server are constrained by the value of CPUCOUNT, CAS and the SAS Compute Server "CAS-enabled" procedures have no awareness of CPUCOUNT.

What also appears in the tests is that the SAS Compute Server CAS-enabled procedures remain constrained by CPU and Memory limits defined for the "sas-programming-environment" container.

Finally each SAS Compute Server "CAS-enabled" procedure creates a separate table loaded into CAS disk cache (/tmp) as SASHDAT and the table is deleted upon procedure termination.



CAS-enabled procedure Benchmark

A benchmark brief with a focus on the compute engine enhancements was published by SAS.

- The introduction of CAS-enabled procedures to the SAS Compute Server :
  - requires additional temporary storage for CAS disk cache. (local temporary disks and generic ephemeral volumes are a good fit)
  - increases the need to adjust SAS MEMSIZE to ensure successful completion of the procedure (data mining and machine learning procedures).
- For reasonable in-memory performance and to avoid reading data from CAS disk cache, set the container's memory limit to :

Memory Limit >= SAS MEMSIZE + mmap files size

- The best estimate for mmap file size is the size of the uncompressed SAS7BDAT size.
- The term mmap represents SASHDAT file segments mapped into memory.

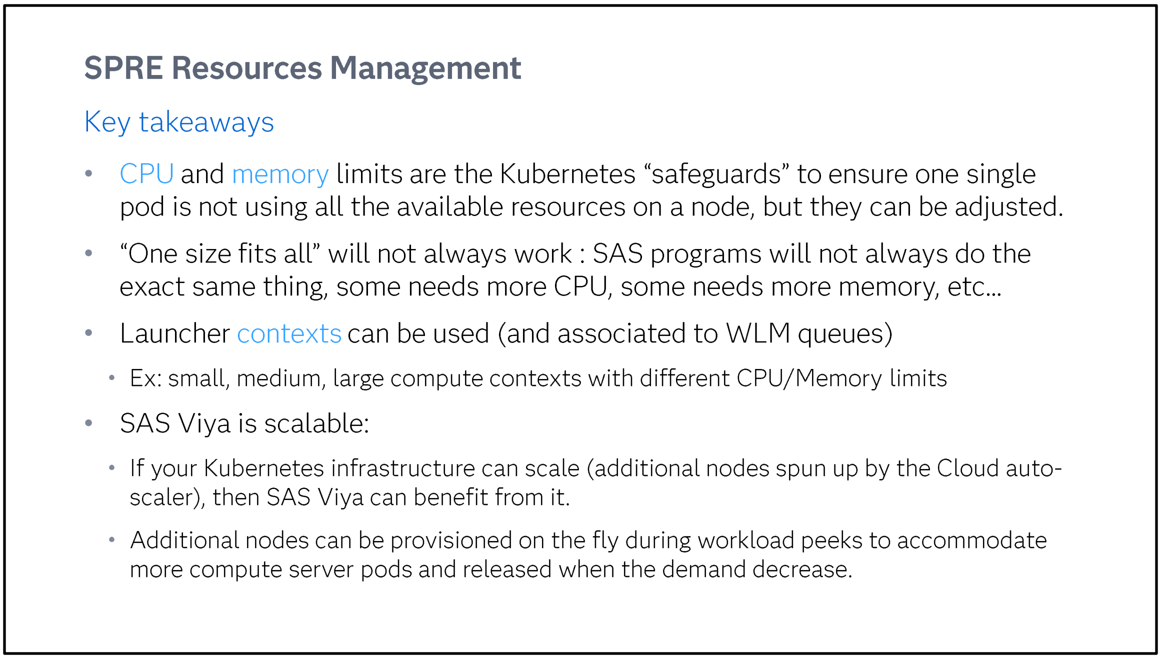
- Best practices
  - MEMSIZE < Container memory limit < Node memory
  - Use server contexts to satisfy the needs of both traditional SAS 9.4 type workloads and the new SAS Compute Server CAS-enabled procedures

A benchmark was published with a focus on the utilization of the Enhanced Compute Engine that is now part of the SAS programming runtime. The benchmark paper is available to SAS employees and SAS partners.
Here are some of the findings :

The introduction of CAS-enabled procedures to the SAS Compute Server requires additional temporary storage for CAS disk cache.
It also increases the need to adjust SAS MEMSIZE to ensure successful completion of the procedures. The data mining and machine learning procedures executed during testing exhibited varying memory requirements based on several factors. The memory required by the procedures is a factor of the number of features and effects, cardinality, hyper-parameter settings, number of rows, and number of threads used.

For reasonable in-memory performance and to avoid reading data from CAS disk cache it is also recommended to increase the container's memory limit to take into account the memory mapped files that corresponds to the CAS Disk Cache and can be estimated based on the loaded dataset's size.

Finally, the benchmark brief provides some general best practices :
First, SAS MEMSIZE should be less than the container memory limit to avoid "Out Of Memory" conditions. The container memory limit should be less than the total node memory in order to provide adequate resources for the operating system and kubelet to prevent pod evictions. The container memory limit should not exceed the allocatable memory on the node.
Then with the variety of resources needed between data management (ETL) and CAS-enabled procedure it is a good idea to separate these workloads using batch and compute contexts tailored to the needs of each type of work.



SPRE Resources Management

Key takeaways

- CPU and memory limits are the Kubernetes “safeguards” to ensure one single pod is not using all the available resources on a node, but they can be adjusted.
- “One size fits all” will not always work : SAS programs will not always do the exact same thing, some needs more CPU, some needs more memory, etc…
- Launcher contexts can be used (and associated to WLM queues)
  - Ex: small, medium, large compute contexts with different CPU/Memory limits
- SAS Viya is scalable:
  - If your Kubernetes infrastructure can scale (additional nodes spun up by the Cloud auto-scaler), then SAS Viya can benefit from it.
  - Additional nodes can be provisioned on the fly during workload peeks to accommodate more compute server pods and released when the demand decrease.

To conclude this module, let’s review the key takeaways.

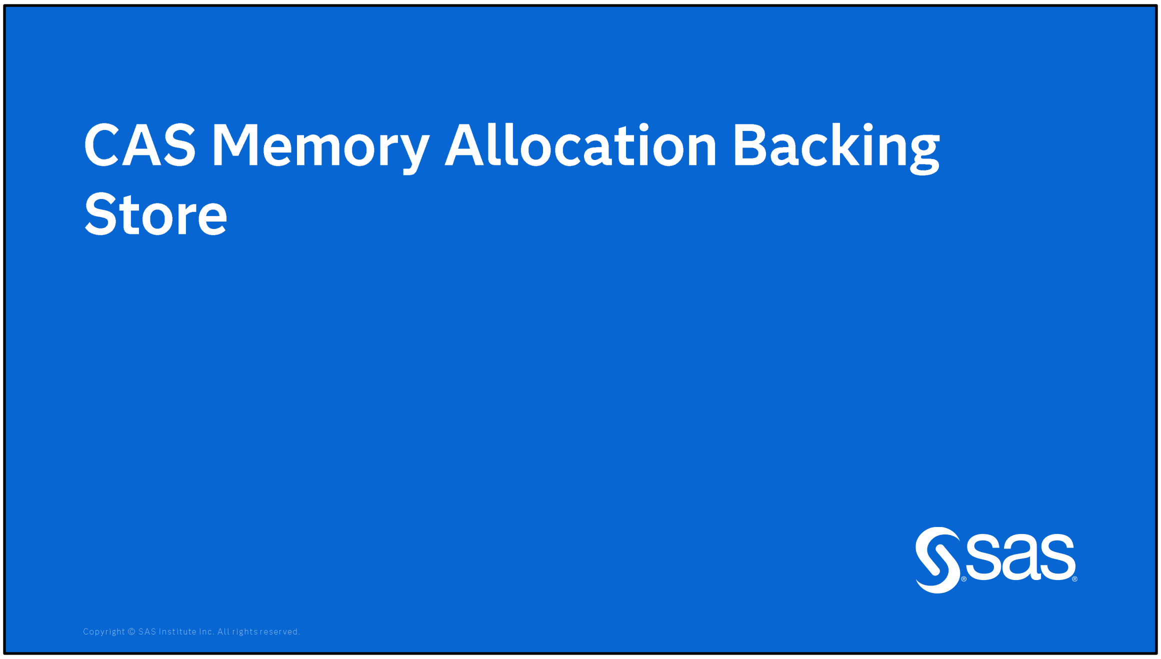
CPU and memory limits are the Kubernetes safeguards to ensure one single pod is not using all the available resources on the node, but they can be adjusted.
It is recommended to review and adjust the SAS programming environment container CPU and memory limits to meet the SAS programs workload requirements.

One size fits all will usually not work for this because the SAS programs will not always do exactly the same thing.
Some needs more CPU, some needs more memory, et cetera.

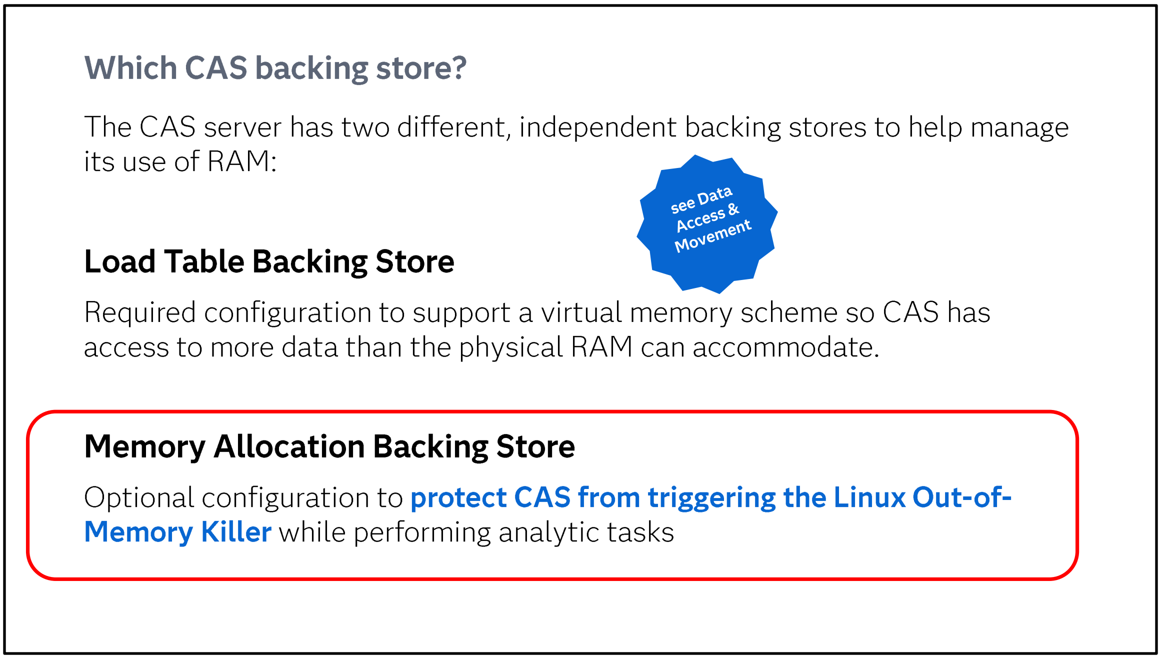
The launcher context can be used and associated to workload management queues.
For example, you could have a small, medium, large compute context with different settings for the CPU and memory limits.
And you could then attach them to different queues in workload management.

Finally, SAS Viya is scalable. Also keep that in mind when you are doing this kind of computing, tuning, planning.
If your Kubernetes infrastructure can scale, then SAS Viya can benefit from it.
Additional nodes can be provisioned on the fly during workload peaks to accommodate more compute server pods, it will allow you to run more workload.
Also, the nice thing is that Viya can scale up, but it can also scale down.
When there are less demand, less workload requests on the system, then the provisioned nodes can be also released.
So yeah, that's something important to factor in, if you are deploying SAS Viya in a scalable

environment, typically a cloud environment.



CAS Memory Allocation Backing Store



Which CAS backing store?

The CAS server has two different, independent backing stores to help manage its use of RAM:

see Data Access & Movement

Load Table Backing Store

Required configuration to support a virtual memory scheme so CAS has access to more data than the physical RAM can accommodate.

Memory Allocation Backing Store

Optional configuration to **protect CAS from triggering the Linux Out-of-Memory Killer** while performing analytic tasks

In this module we want to discuss the “CAS Memory Allocation Backing Store”.

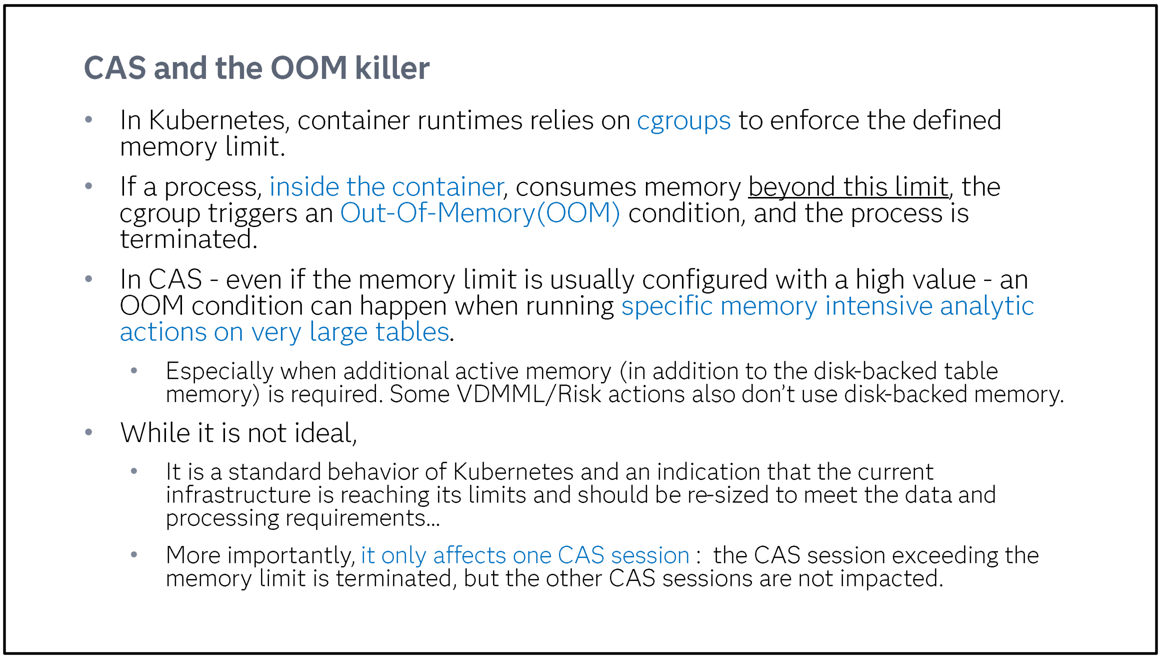
As of the 2024.09 release of SAS Viya, we now need to distinguish between two different kind of backing stores.

The first is the load table backing store. This is a required configuration you specify that lets CAS use local disk as part of a virtual memory scheme so that it has the access it needs to volumes of data that might be larger than the physical RAM offered on the host machines.

We cover more about this in the Data Access and Movement content.

Next is the memory allocation backing store. This is a recent feature added to CAS. It provides an option for you to configure CAS to protect its use of RAM during analytic processing operations.

This memory allocation backing store is the topic that we cover in this module.
We will first explain the problematic with CAS and the Out-Of-Memory condition, then briefly explain what the Cgroups are and how they work and finally describe the new backing store feature and how it improves the stability and reliability of the CAS Server.



CAS and the OOM killer

- In Kubernetes, container runtimes relies on cgroups to enforce the defined memory limit.
- If a process, inside the container, consumes memory beyond this limit, the cgroup triggers an Out-Of-Memory(OOM) condition, and the process is terminated.
- In CAS - even if the memory limit is usually configured with a high value - an OOM condition can happen when running specific memory intensive analytic actions on very large tables.
  - Especially when additional active memory (in addition to the disk-backed table memory) is required. Some VDMML/Risk actions also don't use disk-backed memory.
- While it is not ideal,
  - It is a standard behavior of Kubernetes and an indication that the current infrastructure is reaching its limits and should be re-sized to meet the data and processing requirements…
  - More importantly, it only affects one CAS session : the CAS session exceeding the memory limit is terminated, but the other CAS sessions are not impacted.

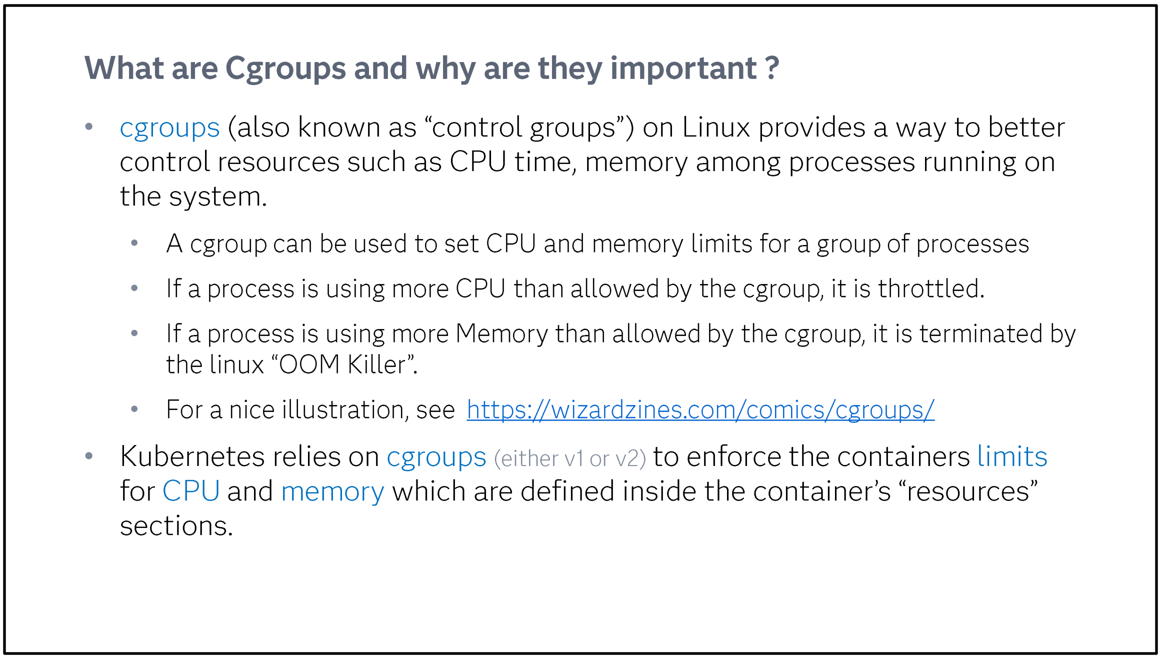
To understand why the "Backing Store for CAS Memory Allocations" was introduced, it is first important to understand how Kubernetes handles the situation where CAS is asking for too much memory.

On Linux, the container runtime typically configures kernel cgroups that apply and enforce the limits that are defined in resources limit specifications.
If the container tries to allocate more memory than this limit, the Linux kernel out-of-memory subsystem activates and, typically, intervenes by stopping the process in the container that tried to allocate memory.

In some situations, it is possible that a CAS action tries to consume more memory than the configured Memory limit that has been defined for the CAS container.
For example, it could happen with Risk computation with a very high number of dimensions or machine learning algorithms working on very large datasets.

In such case the process that corresponds to the CAS session that has run the CAS action will be killed by the Linux Out-Of-Memory Killer.
It is not ideal, but it is understandable as the system protect itself against a single process asking for too much resources, potentially putting other sessions at risk if not interrupted.
Also if it happens too often, it probably means that more infrastructure resources are needed to perform such kind of processing on such amount of data. It makes sense that more complex calculation on more data requires more capacities…

But, more importantly, it only affects one CAS session : while the CAS session exceeding the memory limit is terminated, the other CAS sessions are NOT impacted.



What are Cgroups and why are they important ?

- cgroups (also known as “control groups”) on Linux provides a way to better control resources such as CPU time, memory among processes running on the system.
  - A cgroup can be used to set CPU and memory limits for a group of processes
  - If a process is using more CPU than allowed by the cgroup, it is throttled.
  - If a process is using more Memory than allowed by the cgroup, it is terminated by the linux “OOM Killer”.
  - For a nice illustration, see https://wizardzines.com/comics/cgroups/
- Kubernetes relies on cgroups (either v1 or v2) to enforce the containers limits for CPU and memory which are defined inside the container’s “resources” sections.

Before we go any further, let’s a step back and briefly recap on how cgroups are working…

Cgroups (also known as “control groups”) provides, on Linux, a way to control resources such as CPU time, memory among processes running on the system.

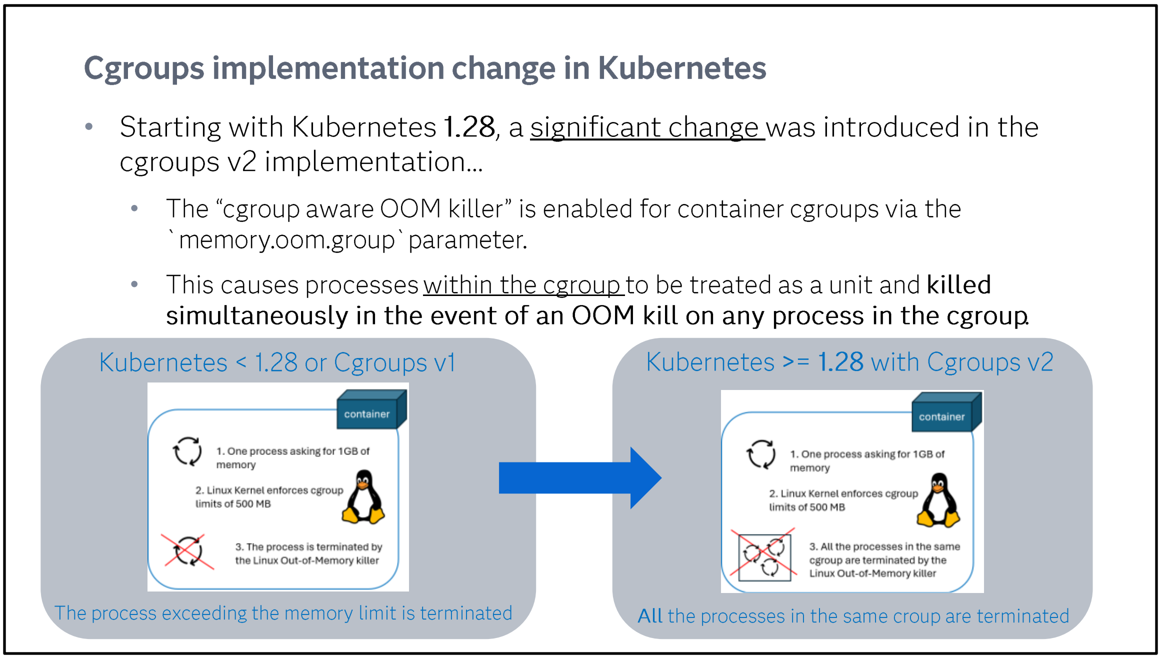
A cgroup can be used to set CPU and memory limits for a group of processes

If a process is using more CPU than allowed by the cgroup, it is throttled.

If a process is using more Memory than allowed by the cgroup, it is terminated by the linux “OOM Killer”.

If you open the provided link here, you will find a little cartoon that is helpful to understand how cgroups work…
It explains what they are used in a very fun and simple way.

Finally, cgroups are very important because Kubernetes relies on them to enforce the containers limits for CPU and memory which are defined inside the container’s “resources” sections.



Cgroups implementation change in Kubernetes

- Starting with Kubernetes **1.28**, a significant change was introduced in the cgroups v2 implementation…
  - The "cgroup aware OOM killer" is enabled for container cgroups via the `memory.oom.group` parameter.
  - This causes processes within the cgroup to be treated as a unit and **killed simultaneously in the event of an OOM kill on any process in the cgroup**.

Kubernetes < 1.28 or Cgroups v1

1. One process asking for 1GB of memory

2. Linux Kernel enforces cgroup limits of 500 MB

3. The process is terminated by the Linux Out-of-Memory killer

The process exceeding the memory limit is terminated

Kubernetes >= **1.28** with Cgroups v2

1. One process asking for 1GB of memory

2. Linux Kernel enforces cgroup limits of 500 MB

3. All the processes in the same cgroup are terminated by the Linux Out-of-Memory killer

All the processes in the same croup are terminated

Now the reason why we've spent a bit of time on the cgroups mechanism explanation is that there was a recent change in the way they are implemented in Kubernetes that has significantly impacted the CAS server.

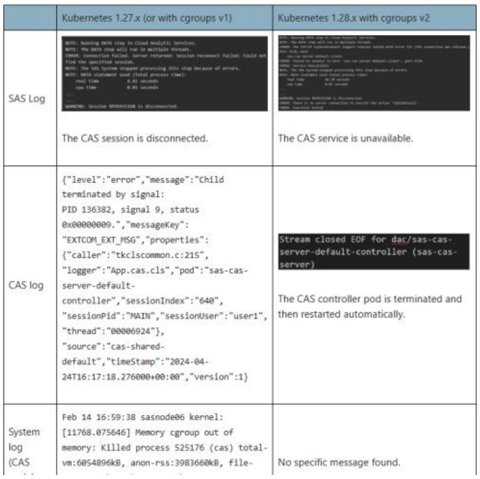
From Kubernetes 1.28, with cgroups v2, The "cgroup aware O-O-M killer" is enabled for container cgroups via the `memory.oom.group` parameter.
This causes the processes within the cgroup to be treated as a unit and killed simultaneously in the event of an Out-Of-Memory kill on any process in the cgroup.

We can illustrate it with the two diagrams below.

So until this change : in case of an Out-Of-Memory condition, only the incriminated process, asking for too much memory was terminated – that's what you can see on the left hand side.
But now, with a recent version of Kubernetes (and if you are using cgroups v2), all the processes which are in the same cgroup will be killed and that's what is illustrated on the right hand side…

Impact of the cgroup change for CAS…

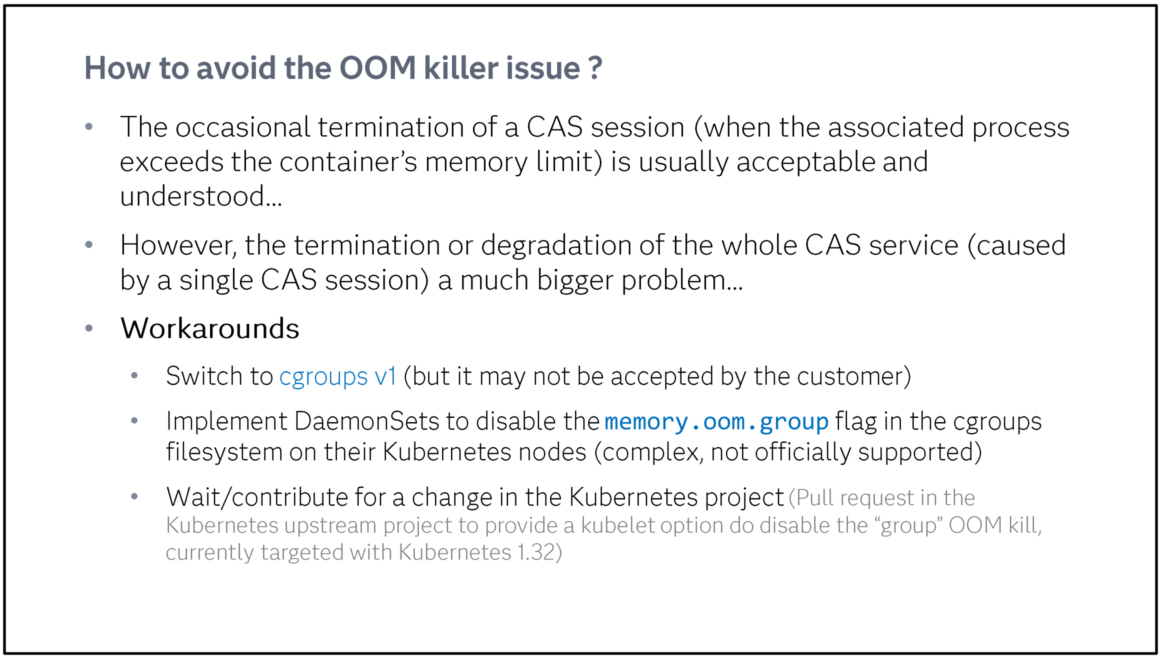
- When a single CAS session process is terminated by the OOM killer, **all the other processes in the same cgroup are also terminated**, which leads the whole CAS server (SMP) or individual CAS worker(s) (MPP) to go down, impacting the whole CAS service.
  - *Although CAS pods will be automatically restarted, it is likely that all the tables loaded in memory will be dropped (and will need to be reloaded again).*



	Kubernetes 1.27.x (or with cgroups v1)	Kubernetes 1.28.x with cgroups v2
SAS Log	NOTE: Running DATA step in Cloud Analytic Services. NOTE: The DATA step will run in multiple threads. ERROR: Connection failed. Server returned: Session reconnect failed: Could not find the specified session. NOTE: The SAS System stopped processing this step because of errors. NOTE: DATA statement used (Total process time): real time [illegible] seconds cpu time [illegible] seconds ... ... WARNING: Session MYSESSION is disconnected. The CAS session is disconnected.	NOTE: Running DATA step in Cloud Analytic Services. NOTE: The DATA step will run in multiple threads. ERROR: [illegible] [illegible] sas-cas-server-default-client ERROR: Failed to connect to host 'sas-cas-server-default-client', port [illegible]. ERROR: Service Unavailable NOTE: The SAS System stopped processing this step because of errors. NOTE: DATA statement used (Total process time): real time [illegible] seconds cpu time [illegible] seconds ... ... WARNING: Session [illegible] is disconnected. ERROR: There is no server connection to execute the action [illegible]. ERROR: Execution failed. The CAS service is unavailable.
CAS log	{"level":"error","message":"Child terminated by signal: PID 136382, signal 9, status 0x00000009.","messageKey": "EXTCOM_EXT_MSG","properties": {"caller":"tkclscommon.c:215", "logger":"App.cas.cls","pod":"sas-cas-server-default-controller","sessionIndex":"640", "sessionPid":"MAIN","sessionUser":"user1", "thread":"00006924"}, "source":"cas-shared-default","timeStamp":"2024-04-24T16:17:18.276000+00:00","version":1}	Stream closed EOF for dac/sas-cas-server-default-controller (sas-cas-server) The CAS controller pod is terminated and then restarted automatically.
System log (CAS	Feb 14 16:59:38 sasnode06 kernel: [11768.075646] Memory cgroup out of memory: Killed process 525176 (cas) total-vm:6054896kB, anon-rss:3983660kB, file-	No specific message found.

So what does it mean for CAS ?
Well it is not a great news because, with this change :
When a single CAS session process is terminated by the O-O-M killer, all the other processes in the same cgroup are also terminated.
Which means all the other CAS process and attached sessions (including the main CAS server process).
So it leads the whole CAS server (SMP) or individual CAS workers (MPP) to go down, impacting the whole CAS service.
The table that you see on the left hand-side is an extract from an internal test summary with CAS deployed in SMP mode (Remember SMP stands for Symmetric MultiProcessing, it means a single CAS instance playing the role of Controller and worker).

The table shows the impact - in a CAS SMP deployment - of a CAS action using too much memory…
It compares the logs and behaviors between two environments : one with Kubernetes 1.27 and another with 1.28 and the new cgroupv2 implementation.



How to avoid the OOM killer issue ?

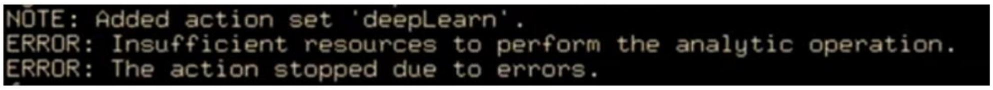
- The occasional termination of a CAS session (when the associated process exceeds the container’s memory limit) is usually acceptable and understood…
- However, the termination or degradation of the whole CAS service (caused by a single CAS session) a much bigger problem…
- Workarounds
  - Switch to cgroups v1 (but it may not be accepted by the customer)
  - Implement DaemonSets to disable the `memory.oom.group` flag in the cgroups filesystem on their Kubernetes nodes (complex, not officially supported)
  - Wait/contribute for a change in the Kubernetes project (Pull request in the Kubernetes upstream project to provide a kubelet option do disable the “group” OOM kill, currently targeted with Kubernetes 1.32)

This change makes the consequences of a single CAS action O-O-M event much more serious…
While the occasional termination of a CAS session consuming too much memory is usually acceptable, the termination or degradation of the whole CAS service (caused by a single CAS session) a much bigger problem…
So how could we avoid this issue ?

Several options have been explored :
switch from cgroups v2 to cgroups v1, but it may not be accepted by the customers or supported in the future or by the Cloud provider…
implement a Daemonset to disable the memory.oom.group flag in the cgroups filesystem is another potential workaround but it requires specific development and configuration and is currently not a tested or supported SAS configuration…

The CAS Memory Allocation Backing Store

- A new feature (2024.09) to prevent the CAS OOM condition.
- Interrupts a CAS action about to use more memory than allowed, **before** the Out-Of-Memory Kill is triggered.



```
NOTE: Added action set 'deepLearn'.
ERROR: Insufficient resources to perform the analytic operation.
ERROR: The action stopped due to errors.
```

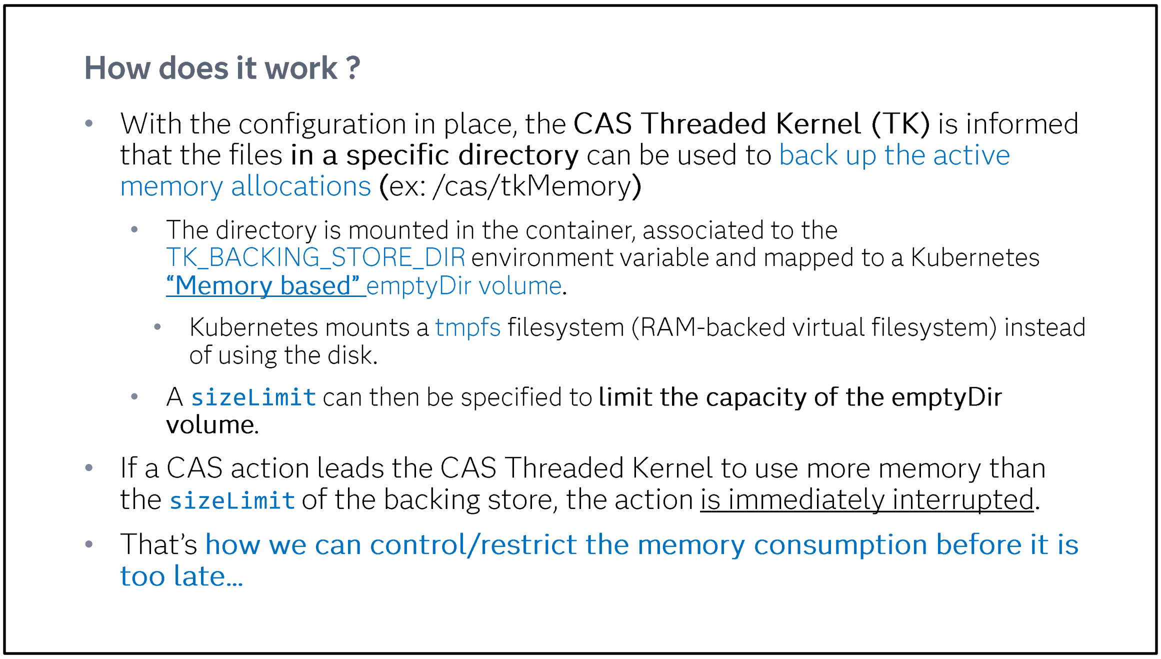
- The error message informs the end-user (for example in SAS Studio or any other CAS client) that the CAS action has failed because it ran out of memory.
- Improves the stability and reliability of the CAS Server.

In order to address this issue, SAS has released a new feature called “CAS Memory allocation backing store”.
The idea is to associate a limit to a memory backing store and then to use this limit as a threshold for the CAS action.

So if a CAS action is about to use more memory than this limit, the CAS action is interrupted before the Out-Of-Memory Kill is triggered.

The end-user will see a nice and clear message to let him know there is not enough resources to perform the CAS action.

But more importantly, the cgroup limit is NOT exceeded and the Linux Out-of-Memory condition is avoided, which improves the stability and reliability of the CAS Server.



How does it work ?

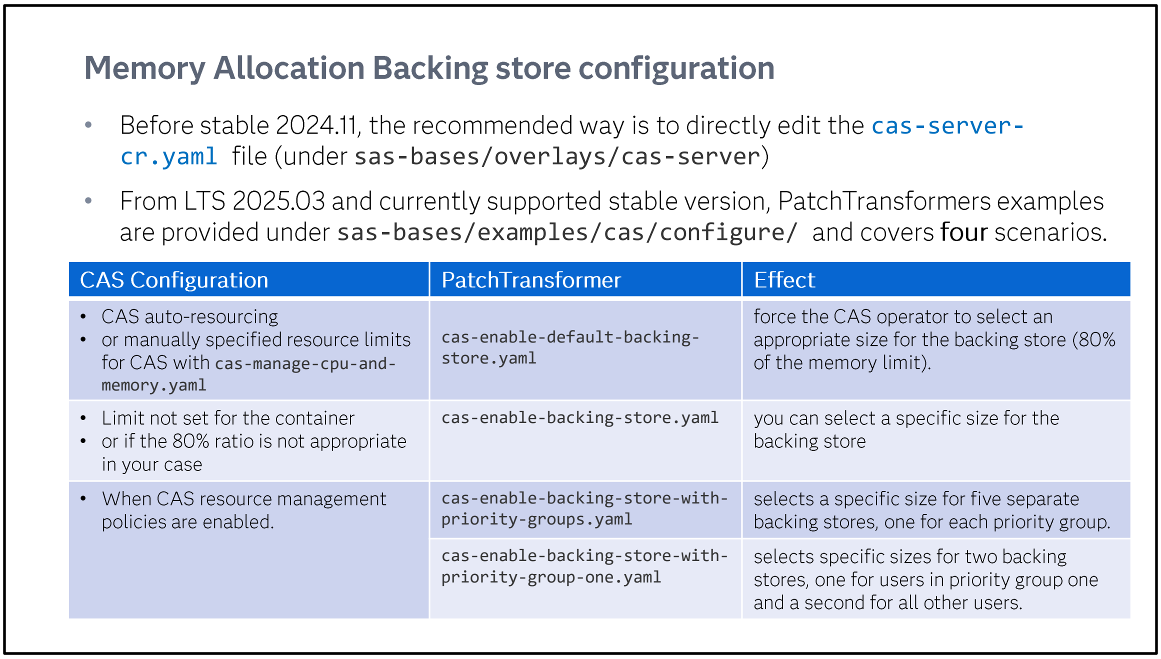
- With the configuration in place, the **CAS Threaded Kernel (TK)** is informed that the files **in a specific directory** can be used to back up the active memory allocations **(**ex: /cas/tkMemory**)**
  - The directory is mounted in the container, associated to the TK_BACKING_STORE_DIR environment variable and mapped to a Kubernetes “Memory based” emptyDir volume.
    - Kubernetes mounts a tmpfs filesystem (RAM-backed virtual filesystem) instead of using the disk.
  - A `sizeLimit` can then be specified to **limit the capacity of the emptyDir** volume.
- If a CAS action leads the CAS Threaded Kernel to use more memory than the `sizeLimit` of the backing store, the action is immediately interrupted.
- That’s how we can control/restrict the memory consumption before it is too late…

So how does it work ?

When the “Backing Store for CAS Memory Allocations” is enabled then the CAS Threaded Kernel (or TK) is informed that the files in a specific directory can be used to back up the active memory allocations (for example: “/cas/tkMemory”).

This directory is mounted in the container, associated to the TK_BACKING_STORE_DIR environment variable and mapped to a Kubernetes emptyDir volume.
By default, the content of emptyDir volumes is stored on the node’s root disk (typically under /var/lib/kubelet).
However, it is possible to set the emptyDir.medium field to “Memory”, the documentation explains that Kubernetes mounts a tmpfs filesystem (RAM-backed virtual filesystem) instead of using the disk.
This kind of “memory based” emptyDir is what we use here.

A size limit can then be specified to limits the capacity of the emptyDir volume. That’s how we can control/restrict the memory consumption before it is too late…
If a CAS action leads the CAS Threaded Kernel to exceed the sizeLimit of the backing store, the action is immediately interrupted.



Memory Allocation Backing store configuration

- Before stable 2024.11, the recommended way is to directly edit the `cas-server-cr.yaml` file (under `sas-bases/overlays/cas-server`)
- From LTS 2025.03 and currently supported stable version, PatchTransformers examples are provided under `sas-bases/examples/cas/configure/` and covers **four** scenarios.

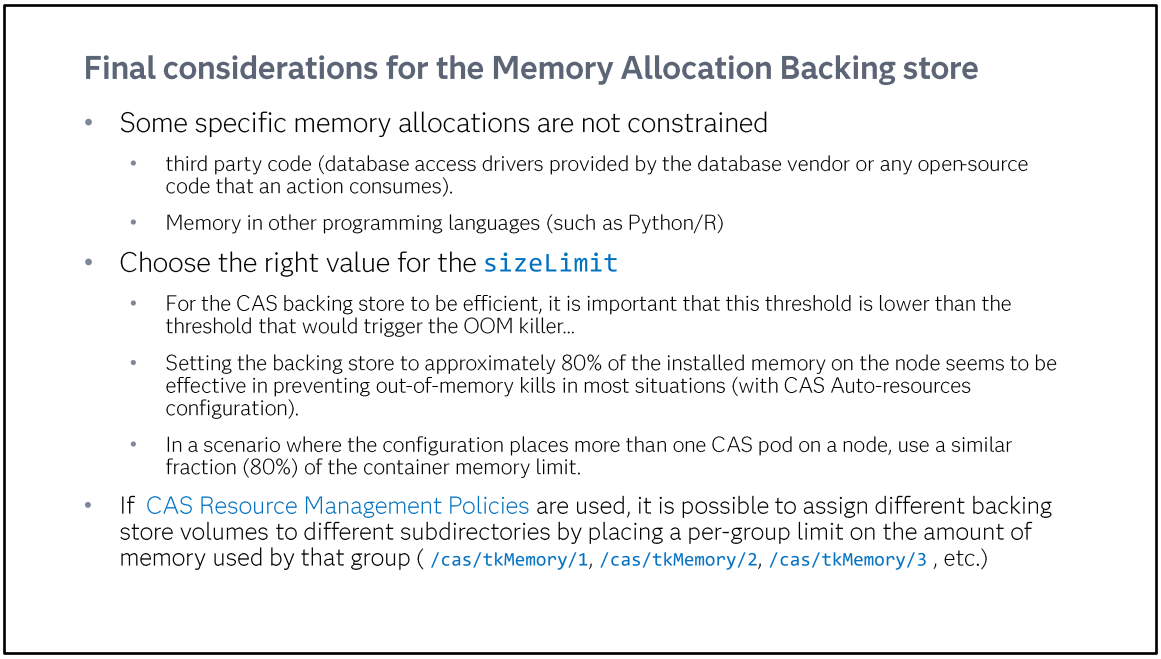
CAS Configuration	PatchTransformer	Effect
• CAS auto-resourcing • or manually specified resource limits for CAS with `cas-manage-cpu-and-memory.yaml`	`cas-enable-default-backing-store.yaml`	force the CAS operator to select an appropriate size for the backing store (80% of the memory limit).
• Limit not set for the container • or if the 80% ratio is not appropriate in your case	`cas-enable-backing-store.yaml`	you can select a specific size for the backing store
• When CAS resource management policies are enabled.	`cas-enable-backing-store-with-priority-groups.yaml`	selects a specific size for five separate backing stores, one for each priority group.
	`cas-enable-backing-store-with-priority-group-one.yaml`	selects specific sizes for two backing stores, one for users in priority group one and a second for all other users.

How is the “Backing Store for CAS Memory Allocations” configured?

For a Viya version that is older than stable 2024.11 (for example LTS 2024.09), you need to edit the CAS Server Custom Resource manifest file before the deployment.
3 changes are required :
A new environment variable,
A volumeMount
and a Volume definition must be added in the CAS Custom Resource manifest file.

From LTS 2025.03 and currently supported stable versions, four PatchTransformers are provided under “sas-bases/examples/cas/configure/” and covers four scenarios.

The most common case is when the default CAS auto-resourcing configuration is in place, in this case you can simply reference the PatchTransformer to force the CAS operator to select an appropriate size for the backing store (80% of the memory limit).
If you don’t set a limit for the CAS container or if you want to set your own SizeLimit value, you can use a different PatchTransformer.
Finally in the case where you have specifically configured CAS resource management policies, then you can define separated size Limit values that correspond to the different priority groups.



Final considerations for the Memory Allocation Backing store

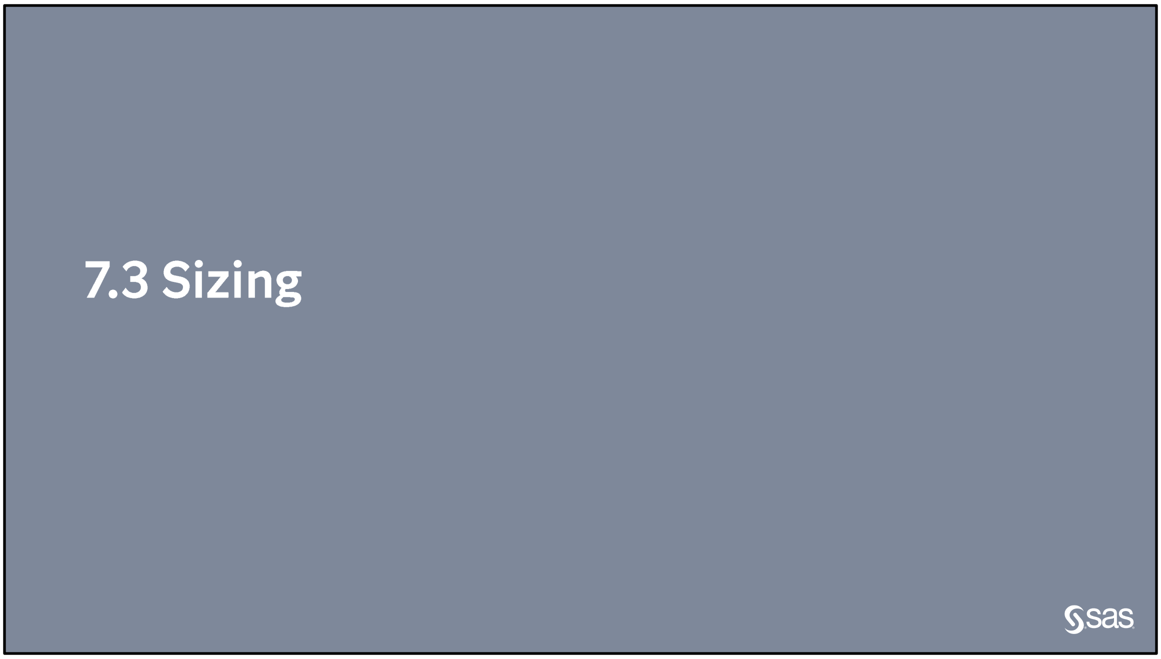
- Some specific memory allocations are not constrained
  - third party code (database access drivers provided by the database vendor or any open-source code that an action consumes).
  - Memory in other programming languages (such as Python/R)
- Choose the right value for the `sizeLimit`
  - For the CAS backing store to be efficient, it is important that this threshold is lower than the threshold that would trigger the OOM killer…
  - Setting the backing store to approximately 80% of the installed memory on the node seems to be effective in preventing out-of-memory kills in most situations (with CAS Auto-resources configuration).
  - In a scenario where the configuration places more than one CAS pod on a node, use a similar fraction (80%) of the container memory limit.
- If CAS Resource Management Policies are used, it is possible to assign different backing store volumes to different subdirectories by placing a per-group limit on the amount of memory used by that group (`/cas/tkMemory/1`, `/cas/tkMemory/2`, `/cas/tkMemory/3` , etc.)

Finally, there are some extra considerations to take into account

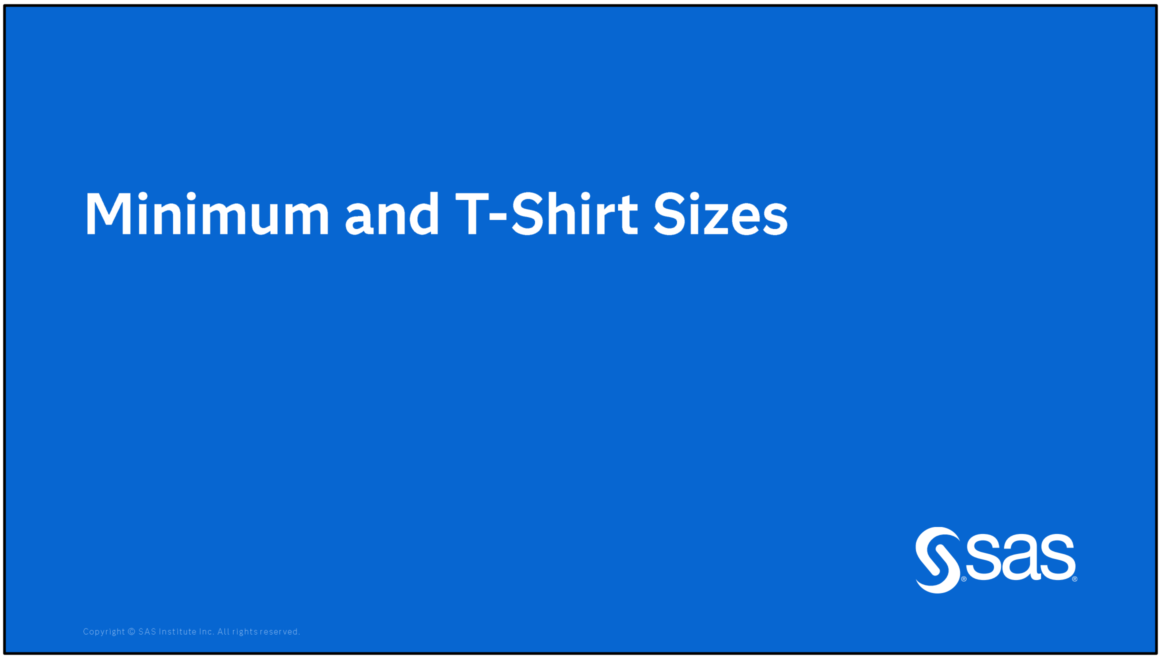
First of all, while most of the use cases benefits from the “Backing Store for CAS Memory Allocations” feature to prevent O-O-M kills, there are some exceptions.
Not all memory consumption is constrained by the size of the backing store.
These include:
Memory allocated by third party code (database access drivers provided by the database vendor or any open-source code that an action consumes).
Memory in other programming languages – when an action spawns processes or threads in other programming languages (such as Python or Java) to implement the action’s logic

The second consideration is that the sizeLimit value should be chosen with care. Because it depends on the way CAS is configured for the resource utilization. For example, with CAS MPP auto-resource, it makes sense to have a value that corresponds to 80% of the installed memory on the worker nodes. But if CAS has been configured to co-exist with other CAS pods on the same node, then it is better to use a value that corresponds to 80% of the CAS container’s memory limit.

And lastly, when CAS Resource Management Policies are used you can define several “priority groups” (up to 5). In such case, it is possible to assign different backing store volumes to different subdirectories by placing a “per-group” limit on the amount of memory used by that group…



7.3 Sizing



Minimum and T-Shirt Sizes

Minimum Sizing Recommendations

SAS® Viya® Platform Operations | 2023.10 ▾ PDF | EPUB | Feedback

Welcome
- What's New in Operations
- Getting Started
- System Requirements
  - Virtual Infrastructure Requirements
  - Platform-Specific Requirements
  - Hardware and Resource Requirements
  - **Sizing Recommendations**
  - Data Source Requirements

System Requirements

Sizing Recommendations

- Sizing Recommendations for Azure
- Sizing Recommendations for AWS
- Sizing Recommendations for GCP and Anthos Clusters on VMware
- Sizing Recommendations for OpenShift
- Sizing Recommendations for Open Source Kubernetes

While an official sizing (questionnaire, HW esimates) is recommended for all proposals, **the official SAS Viya operation guide also provides minimum sizing recommendations**

- The estimates are derived from SAS Performance Teams tests and simulations
- For each supported Cloud platform (Azure, AWS, GCP/GKE on VMware, Openshift, Open-source Kubernetes)
- For only one offering: SAS® Viya® *(include VA, VS, VDMML, Data Prep, etc...)*
- “Small” Deployments
  - No more than 8 concurrent user sessions distributed across SAS Viya user interfaces.
  - Batch jobs were run during "off-peak hours," and simulated concurrent user jobs were run during "business hours".
  - Data sets: 2-5 GB and batch jobs used file sizes of 5 GB

Provides a good starting point and general idea of the infrastructure footprint and costs !

It's always better to have an official sizing for your customers, especially for production and critical environments.
But you can also find very useful information about sizing in the documentation in the SAS Viya operation guide.

And those sizing estimates are coming from the “SAS Performance tests and simulations” team.

They are provided for each cloud platform, Azure, AWS, GCP, GKE on VMware, OpenShift, and even open-source Kubernetes.

BUT they only exist for one offering, which is a SAS Viya offering, which is probably the most common offering because it includes VA, VS, VDMML, but also the Data prep product stack.

The assumption for those kind of Sizings is that it's a small deployment with no more than 8 concurrent user sessions distributed across SAS Viya user interfaces.
And in terms of data amounts, we are between 2 and 5 Gigabytes of data and 5 Gigabytes of files for the batch jobs.

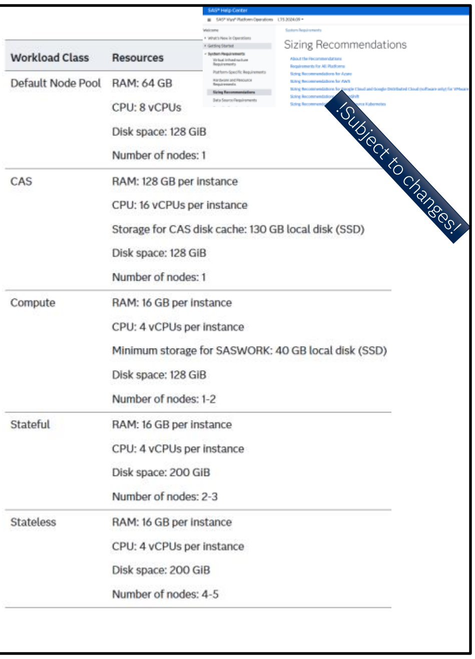
It is a good starting point, and it can give you a good first idea of the infrastructure footprint and potentially the costs that could be associated if you are running SAS Viya in the Cloud.

Sizing Recommendations - Azure

Infrastructure Assumptions:

- 3 Node pools minimum (default, CAS, other)
- VM instance types with Intel chips
- VM instance types that support premium storage (for nodes that requires PV)
- High Memory-to-CPU ratio
- No mix of instance type generations (such as v4/v5)

Customer scenario inputs roughly correspond to the "small" WW sizing T-shirt size.

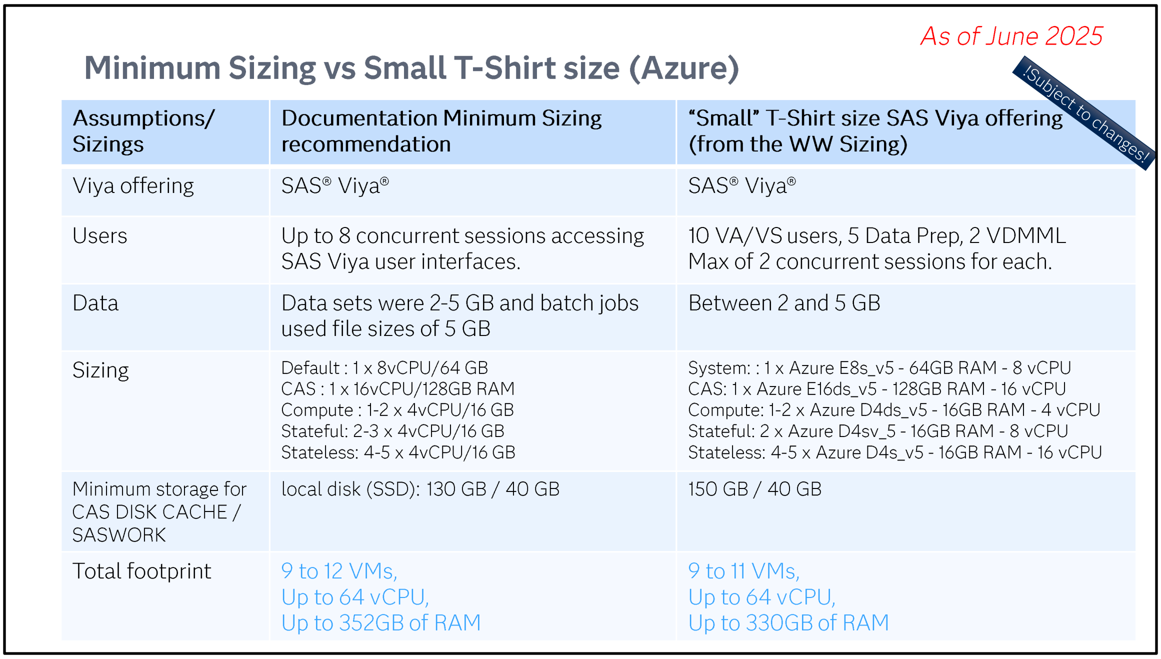

SAS® Help Center
SAS® Viya® Platform Operations LTS 2024.09
Welcome
What's New in Operations
Getting Started
System Requirements
Virtual Infrastructure Requirements
Platform-Specific Requirements
Hardware and Resource Requirements
Sizing Recommendations
Data Source Requirements
System Requirements
Sizing Recommendations
About the Recommendations
Requirements for All Platforms
Sizing Recommendations for Azure
Sizing Recommendations for AWS
Sizing Recommendations for Google Cloud and Google Distributed Cloud (software only) for VMware
!Subject to changes!

Workload Class	Resources
Default Node Pool	RAM: 64 GB CPU: 8 vCPUs Disk space: 128 GiB Number of nodes: 1
CAS	RAM: 128 GB per instance CPU: 16 vCPUs per instance Storage for CAS disk cache: 130 GB local disk (SSD) Disk space: 128 GiB Number of nodes: 1
Compute	RAM: 16 GB per instance CPU: 4 vCPUs per instance Minimum storage for SASWORK: 40 GB local disk (SSD) Disk space: 128 GiB Number of nodes: 1-2
Stateful	RAM: 16 GB per instance CPU: 4 vCPUs per instance Disk space: 200 GiB Number of nodes: 2-3
Stateless	RAM: 16 GB per instance CPU: 4 vCPUs per instance Disk space: 200 GiB Number of nodes: 4-5

On the right side of the screen, we have an extract from the official SAS Viya documentation with the minimum sizing recommendation for a deployment in Azure. We have the number of vCPUs and the amount of RAM, and even the required disk space and an indication on the number of machines (which could be used as the maximum number of nodes in the node pool).
In terms of infrastructure, the assumptions are that :
We have at least three node pools,
We are using VMs with Intel chipset,
We are using instance type that supports premium storage.
Regarding the computing resources, there is a quite high "memory to CPU" ratio, meaning that we are usually on a 1 to 4 ratio between the number of cores and the number of Gigabytes of RAM.
And also, there's no mix of instance type generation because in the cloud environment you could have different generations of instance types.
For example, now, in Azure, the latest VM generation is v5, but before that, we had the v4 instance types.

For each Cloud provider, the documentation provides some guidelines to pick up the right instance types, then we have the number of vCPUs and the amount of RAM for this minimum sizing.

So, what is the difference between this minimum sizing recommendation and the small WW sizing T-Shirt size ?
That's what we tried to determine in the next slide.



As of June 2025

!Subject to changes!

Minimum Sizing vs Small T-Shirt size (Azure)

Assumptions/ Sizings	Documentation Minimum Sizing recommendation	"Small" T-Shirt size SAS Viya offering (from the WW Sizing)
Viya offering	SAS® Viya®	SAS® Viya®
Users	Up to 8 concurrent sessions accessing SAS Viya user interfaces.	10 VA/VS users, 5 Data Prep, 2 VDMML Max of 2 concurrent sessions for each.
Data	Data sets were 2-5 GB and batch jobs used file sizes of 5 GB	Between 2 and 5 GB
Sizing	Default : 1 x 8vCPU/64 GB CAS : 1 x 16vCPU/128GB RAM Compute : 1-2 x 4vCPU/16 GB Stateful: 2-3 x 4vCPU/16 GB Stateless: 4-5 x 4vCPU/16 GB	System: : 1 x Azure E8s_v5 - 64GB RAM - 8 vCPU CAS: 1 x Azure E16ds_v5 - 128GB RAM - 16 vCPU Compute: 1-2 x Azure D4ds_v5 - 16GB RAM - 4 vCPU Stateful: 2 x Azure D4sv_5 - 16GB RAM - 8 vCPU Stateless: 4-5 x Azure D4s_v5 - 16GB RAM - 16 vCPU
Minimum storage for CAS DISK CACHE / SASWORK	local disk (SSD): 130 GB / 40 GB	150 GB / 40 GB
Total footprint	9 to 12 VMs, Up to 64 vCPU, Up to 352GB of RAM	9 to 11 VMs, Up to 64 vCPU, Up to 330GB of RAM

So if we compare these "Minimum Sizings" from the official documentation with the T-shirt sizing, we can see that, in terms of customer scenario and users' input information, it corresponds to the small t-shirt sizing.
In the table, we compare what we have in the documentation, for the minimum sizing, to the small t-shirt size, that is provided by the Worldwide sizing team.
We do that for the same offering to remain consistent.
This concept of concurrency can mean different things, depending on how you express it, how you write it, how you explain it to the customer.
In the sizing team methodology, with the 8 concurrent sessions it means that you have potentially at one time 8 people running Analytics against the CAS server from various Viya clients.
It does not mean that the usage with such sizing is limited to 8 users.
For example, you could have 20 users with their screen showing the Visual Analytics UI, but it doesn't necessarily mean that they will all click at the same time on a button or on a link to perform a processing action that will require back-end processing.
In terms of data, the assumption are also very similar.
What we can see is that, in the recent versions of the documentation the minimum sizing is aligned with the "small" T-Shirt size.

The goal here is to also provide some basic idea of what a small sizing is.
As you can see it represent a fairly high number of machines.
However, most of the instance type are pretty small (4 vcpu and 16 Gigabytes of RAM), so we could have less machines if we used 8-cores VMs for the compute, stateful and stateless workload.

But what is also interesting is to look at the associated costs in the cloud.

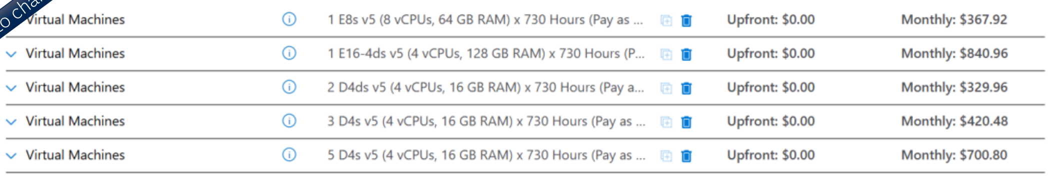
Azure Sizing costs

Pay as you go (estimates)

12 VMs ("small" T-shirt sizing) ~ US$ 2,660

As of March 2025

!Subject to changes!



Virtual Machines	1 E8s v5 (8 vCPUs, 64 GB RAM) x 730 Hours (Pay as ...	Upfront: $0.00	Monthly: $367.92
Virtual Machines	1 E16-4ds v5 (4 vCPUs, 128 GB RAM) x 730 Hours (P...	Upfront: $0.00	Monthly: $840.96
Virtual Machines	2 D4ds v5 (4 vCPUs, 16 GB RAM) x 730 Hours (Pay a...	Upfront: $0.00	Monthly: $329.96
Virtual Machines	3 D4s v5 (4 vCPUs, 16 GB RAM) x 730 Hours (Pay as ...	Upfront: $0.00	Monthly: $420.48
Virtual Machines	5 D4s v5 (4 vCPUs, 16 GB RAM) x 730 Hours (Pay as ...	Upfront: $0.00	Monthly: $700.80

It provides a rough idea of the costs, but...

:

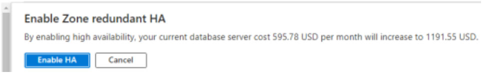
If we look at the costs for this small T-Shirt sizing.
Doing a very basic estimation with the "pay as you go" model, the cost of the Virtual machines in Azure is close to 2 thousands and 6 hundred US dollars a month.
What you see on the screen is really a rough estimates with the recommended instance types and the maximum number of nodes.

Sizing Estimate Caveats

The estimate does not account for other services that you may add to the environment

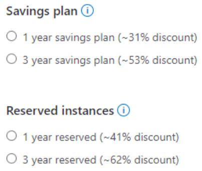
- Persistent Storage
- External Postgres
- Monitoring (Grafana-Prometheus)
- Logging (Kibana-ElasticSearch)
- Others

Example
Azure NetApp Files Ultra
4TB ~ $1.6K /monthly



Enable Zone redundant HA

By enabling high availability, your current database server cost 595.78 USD per month will increase to 1191.55 USD.



Savings plan

○ 1 year savings plan (~31% discount)
○ 3 year savings plan (~53% discount)

Reserved instances

○ 1 year reserved (~41% discount)
○ 3 year reserved (~62% discount)

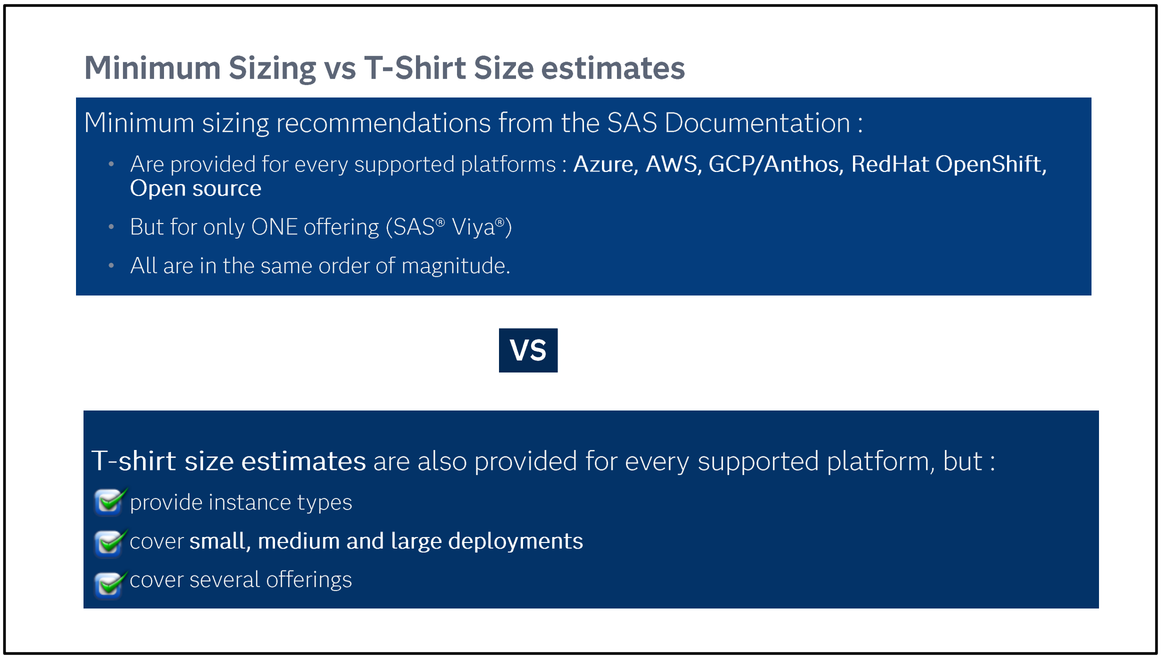
- Stopping the cluster when not used can reduce the bill
- Other Cloud pricing models reduce the monthly cost

:

But our rough estimates doesn't consider other things

like persistent storage.
If you go, for example, with the Azure NetApp files, it could be more costly.
If you are running an external Postgres, you will also have to pay for the managed Postgres SQL services in Azure.
And then if you also deploy things like monitoring and logging, they need some significant infrastructure addition.
And so you will have to pay for that too.
Also, for example, if you enable HA for your external Postgres, it's another cost on top of it.

On the good side though, you have options to reduce the bill.
For example, with the “pay as you go” model, when you stop your AKS cluster when it's not used, you can reduce the bill.
Also in the cloud, instead of going with the “pay as you go” model, you can commit to use the resources for a given time, like one year or three years to pay less.



Minimum Sizing vs T-Shirt Size estimates

Minimum sizing recommendations from the SAS Documentation :

- Are provided for every supported platforms : **Azure, AWS, GCP/Anthos, RedHat OpenShift, Open source**
- But for only ONE offering (SAS® Viya®)
- All are in the same order of magnitude.

VS

T-shirt size estimates are also provided for every supported platform, but :

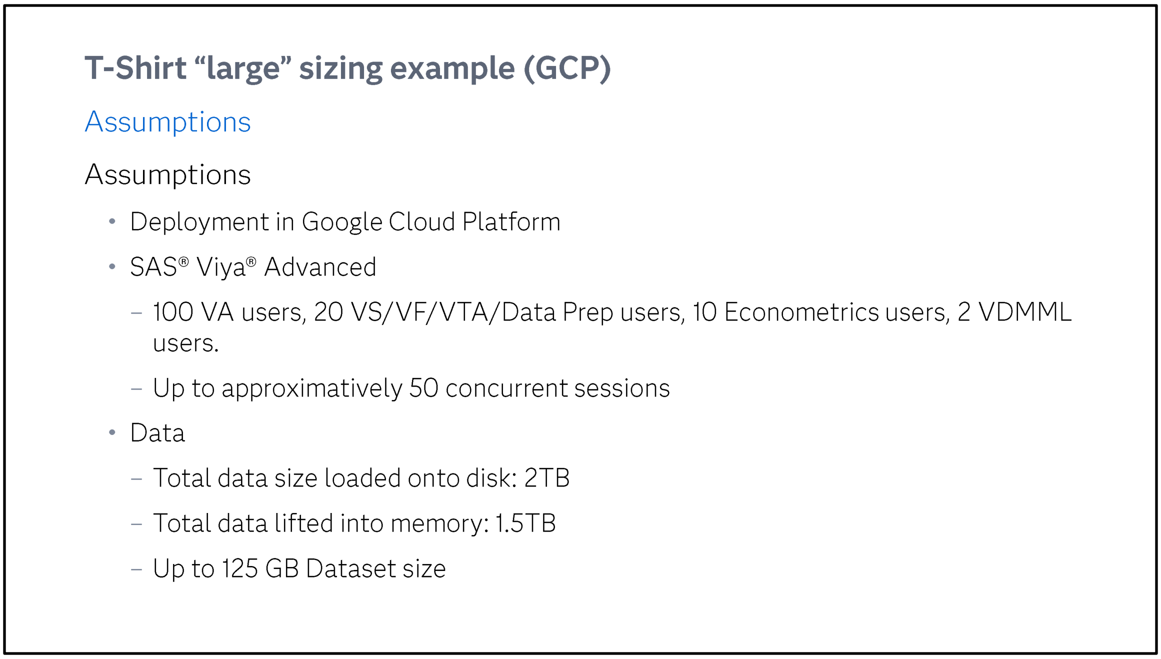
✓ provide instance types

✓ cover **small, medium and large deployments**

✓ cover several offerings

To summarize, the minimum sizing recommendations from the SAS Documentation :
Are provided for every supported platforms : Azure, AWS, GCP/Anthos, RedHat OpenShift, Open source
But for only ONE offering (SAS® Viya®)
All are in the same order of magnitude.

On the other side the T-shirt estimates are also provided for every supported platform, but :
provide instance types
cover small, medium and large deployments
cover several offerings such as SAS Visual Analytics, SAS Viya advanced, or SAS Viya enterprise.



T-Shirt “large” sizing example (GCP)

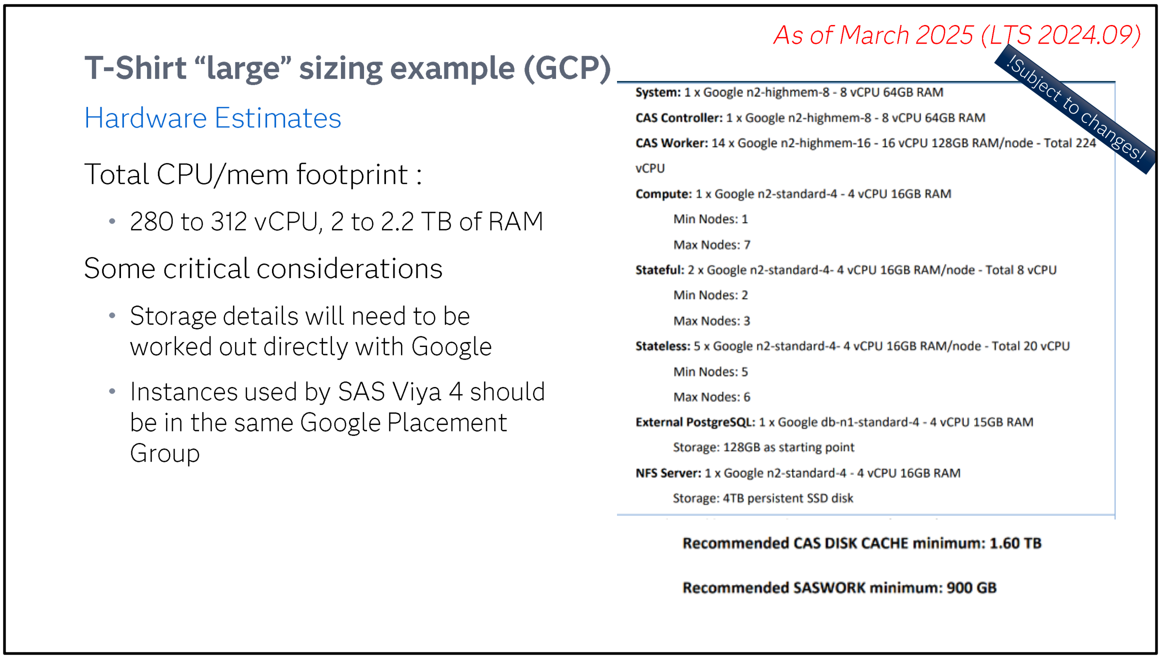
Assumptions

Assumptions

- Deployment in Google Cloud Platform
- SAS® Viya® Advanced
  - 100 VA users, 20 VS/VF/VTA/Data Prep users, 10 Econometrics users, 2 VDMML users.
  - Up to approximatively 50 concurrent sessions
- Data
  - Total data size loaded onto disk: 2TB
  - Total data lifted into memory: 1.5TB
  - Up to 125 GB Dataset size

Now, let’s take another example : since we already saw a small T-shirt size, let’s look at the details of a large T-Shirt sizing for “SAS Viya Advanced” in GCP.
Compared to the standard “SAS Viya Offering”, the “SAS Viya Advanced” offering includes Econometrics optimization, Visual Forecasting, Visual text Analytics and IML.
In this large T-shit scenario, we have
100 users for Visual Analytics , 20 users for Visual Statistics, Visual Forecasting, Visual Text Analytics and Data Prep, 10 users for Econometrics and 2 VDMML users. With potentially up to 50 concurrent sessions for both CAS and SAS Compute analytics.
In terms of Data : the total data size loaded onto disk is around 2 Tera Bytes and the data lifted into memory amounts to 1.5 Terabytes.
Finally, we have large datasets (up to 125 Giga bytes on disk).

As you see, we need to make those kind of assumptions in order to derive the sizing or a hardware estimate.



As of March 2025 (LTS 2024.09)

!Subject to changes!

T-Shirt “large” sizing example (GCP)

Hardware Estimates

Total CPU/mem footprint :

- 280 to 312 vCPU, 2 to 2.2 TB of RAM

Some critical considerations

- Storage details will need to be worked out directly with Google
- Instances used by SAS Viya 4 should be in the same Google Placement Group

System: 1 x Google n2-highmem-8 - 8 vCPU 64GB RAM

CAS Controller: 1 x Google n2-highmem-8 - 8 vCPU 64GB RAM

CAS Worker: 14 x Google n2-highmem-16 - 16 vCPU 128GB RAM/node - Total 224 vCPU

Compute: 1 x Google n2-standard-4 - 4 vCPU 16GB RAM

Min Nodes: 1

Max Nodes: 7

Stateful: 2 x Google n2-standard-4- 4 vCPU 16GB RAM/node - Total 8 vCPU

Min Nodes: 2

Max Nodes: 3

Stateless: 5 x Google n2-standard-4- 4 vCPU 16GB RAM/node - Total 20 vCPU

Min Nodes: 5

Max Nodes: 6

External PostgreSQL: 1 x Google db-n1-standard-4 - 4 vCPU 15GB RAM

Storage: 128GB as starting point

NFS Server: 1 x Google n2-standard-4 - 4 vCPU 16GB RAM

Storage: 4TB persistent SSD disk

Recommended CAS DISK CACHE minimum: 1.60 TB

Recommended SASWORK minimum: 900 GB

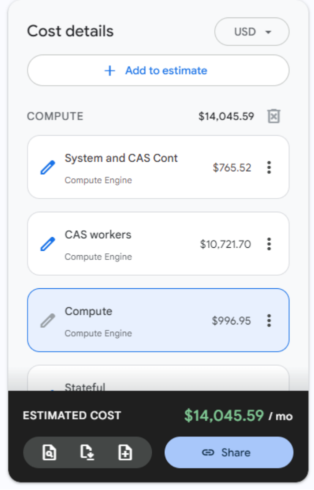
So with this assumption, you end up with a pretty massive infrastructure…
14 CAS worker nodes, each with 16 cores and 128 Gigabytes of memory.
One to seven small compute nodes.
Then many small stateful and stateless nodes with 4vCPU each.
If we also add a few cores for the external postgres and the NFS server, we end up with 280 to 312 vCPUs and around 2 Tera bytes of memory.
In the sizing, you can also find some critical considerations regarding the storage and the recommendation to place the nodes in the same Google placement groups.
It means that your nodes will be even closer inside their zone, which reduces the latency in the communications between the nodes.
This recommendation is particularly useful for the CAS cluster: if you have 14 CAS workers they need to communicate quickly together.

As of March 2025

T-Shirt “large” sizing example (GCP)

“Pay as you go” estimates

- Rough estimates
- Only count the VMs costs
  - Not the storage, not the network, not the PostgreSQL instance
- 7d/24h up
- “Pay as you go”



Cost details USD

+ Add to estimate

COMPUTE $14,045.59

System and CAS Cont $765.52
Compute Engine

CAS workers $10,721.70
Compute Engine

Compute $996.95
Compute Engine

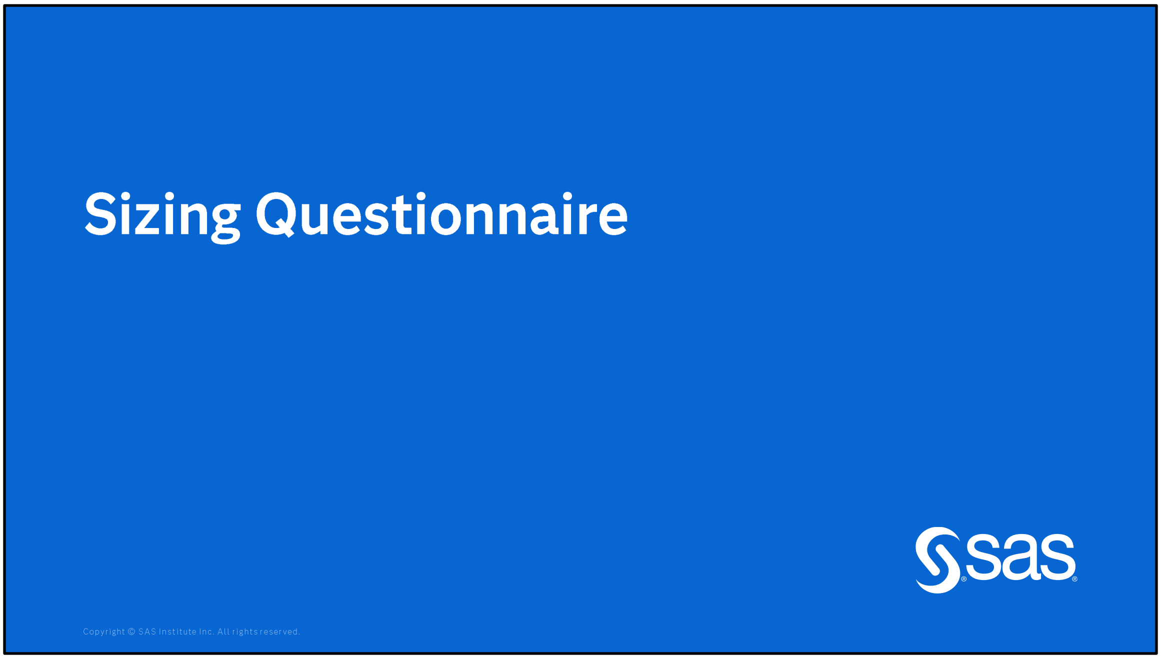
Stateful

ESTIMATED COST $14,045.59 / mo

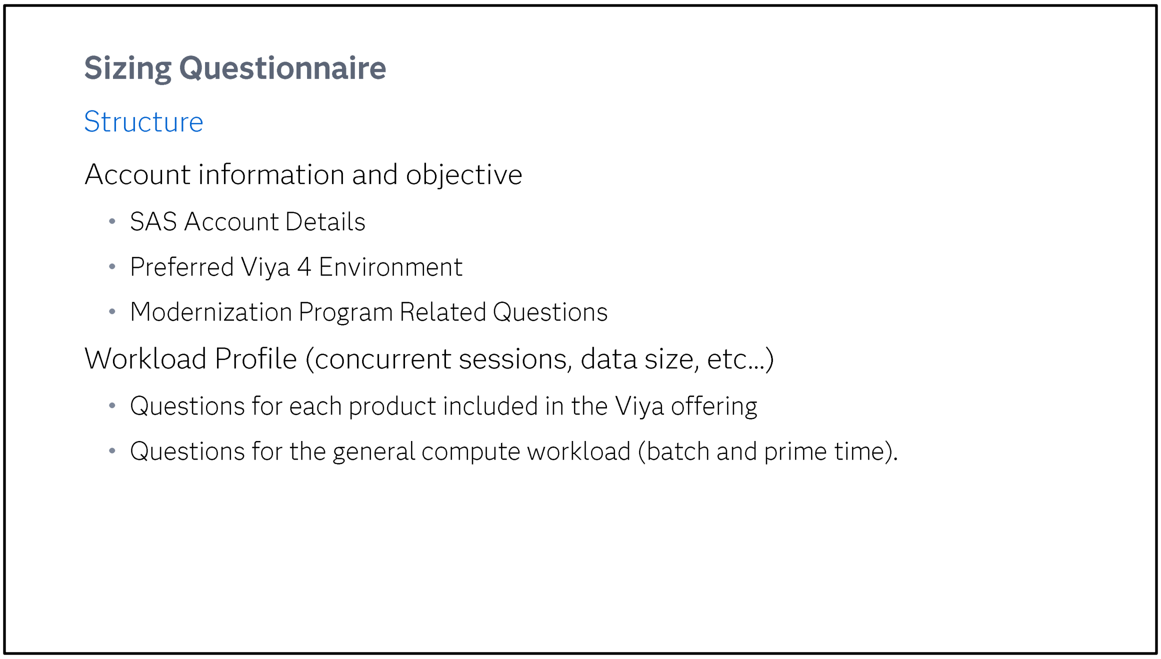
Share

And finally, if we look at the associated costs, again, it's a rough estimate.
And we are just counting the VM costs, but it will cost around 15,000 US dollar a month.
Also, with the assumption that it is 7 days, 24 hours up and it is a “pay as you go” model.
The bill can be reduced by working on those two items.

So again, although it is not 100% precise and may not match exactly with your customer case, it gives us another reference point in terms of hardware footprint and overall cost.



Sizing Questionnaire



Sizing Questionnaire

Structure

Account information and objective

- SAS Account Details
- Preferred Viya 4 Environment
- Modernization Program Related Questions

Workload Profile (concurrent sessions, data size, etc...)

- Questions for each product included in the Viya offering
- Questions for the general compute workload (batch and prime time).

SAS provides a Worldwide sizing process for the SAS Viya platform.
It corresponds to what is now known as the “Bespoke Sizing services” (that’s how it is now advertised on the “WorldWide sizing” homepage)
We want to specifically focus on the “Questionnaire” in this module.

The questionnaire has two parts:
The account information objective is the first part of the questionnaire,
and then you have the workload profile, where you will enter all the information that you can have from the customer on the workload:
so concurrent sessions, data size, et cetera, and you will have to provide information for each product that is in the Viya offering
that correspond to the sizing.
Note that you will also have questions for what is called “the general compute workload” which corresponds to the SAS programming run-time based processing. It is an important information to collect.

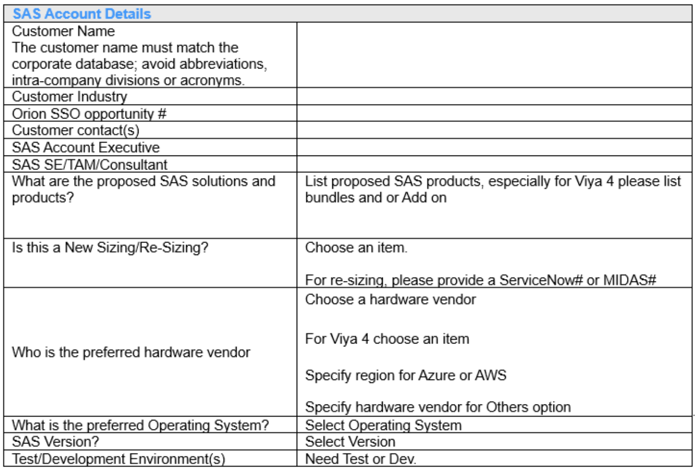
Sizing Input

Sizing Questionnaire

- ➢ Example for the **SAS® Viya** offering

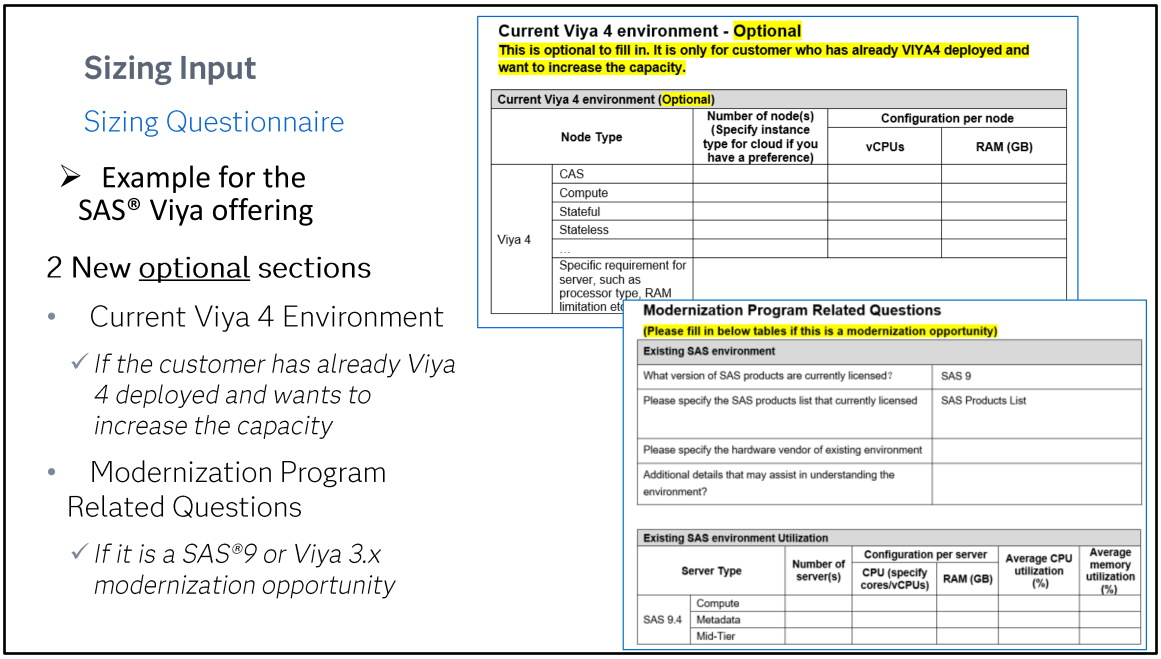
(previously known as "Visual Machine Learning")

SAS Account Details
- Customer details
- List of SAS Products (Add-on, etc,…)
- Hardware vendor / Kubernetes Platform / Cloud Region(for Azure and AWS)



SAS Account Details	
Customer Name The customer name must match the corporate database; avoid abbreviations, intra-company divisions or acronyms.	
Customer Industry	
Orion SSO opportunity #	
Customer contact(s)	
SAS Account Executive	
SAS SE/TAM/Consultant	
What are the proposed SAS solutions and products?	List proposed SAS products, especially for Viya 4 please list bundles and or Add on
Is this a New Sizing/Re-Sizing?	Choose an item. For re-sizing, please provide a ServiceNow# or MIDAS#
Who is the preferred hardware vendor	Choose a hardware vendor For Viya 4 choose an item Specify region for Azure or AWS Specify hardware vendor for Others option
What is the preferred Operating System?	Select Operating System
SAS Version?	Select Version
Test/Development Environment(s)	Need Test or Dev.

We will look at an example , as usually, with the "SAS Viya" offering, which was known as "SAS Visual Machine Learning", but since has been renamed simply as "SAS Viya".
For the SAS account details, we must provide the customer details, the list of the SAS products, the hardware, vendor, Kubernetes platform, and cloud region.



Sizing Input

Sizing Questionnaire

- ➢ Example for the SAS® Viya offering

2 New optional sections

- Current Viya 4 Environment
  - ✓ *If the customer has already Viya 4 deployed and wants to increase the capacity*
- Modernization Program Related Questions
  - ✓ *If it is a SAS®9 or Viya 3.x modernization opportunity*

Current Viya 4 environment - Optional

This is optional to fill in. It is only for customer who has already VIYA4 deployed and want to increase the capacity.

Current Viya 4 environment (Optional)				
Node Type		Number of node(s) (Specify instance type for cloud if you have a preference)	Configuration per node	
			vCPUs	RAM (GB)
Viya 4	CAS			
	Compute			
	Stateful			
	Stateless			
	…			
	Specific requirement for server, such as processor type, RAM limitation etc			

Modernization Program Related Questions

(Please fill in below tables if this is a modernization opportunity)

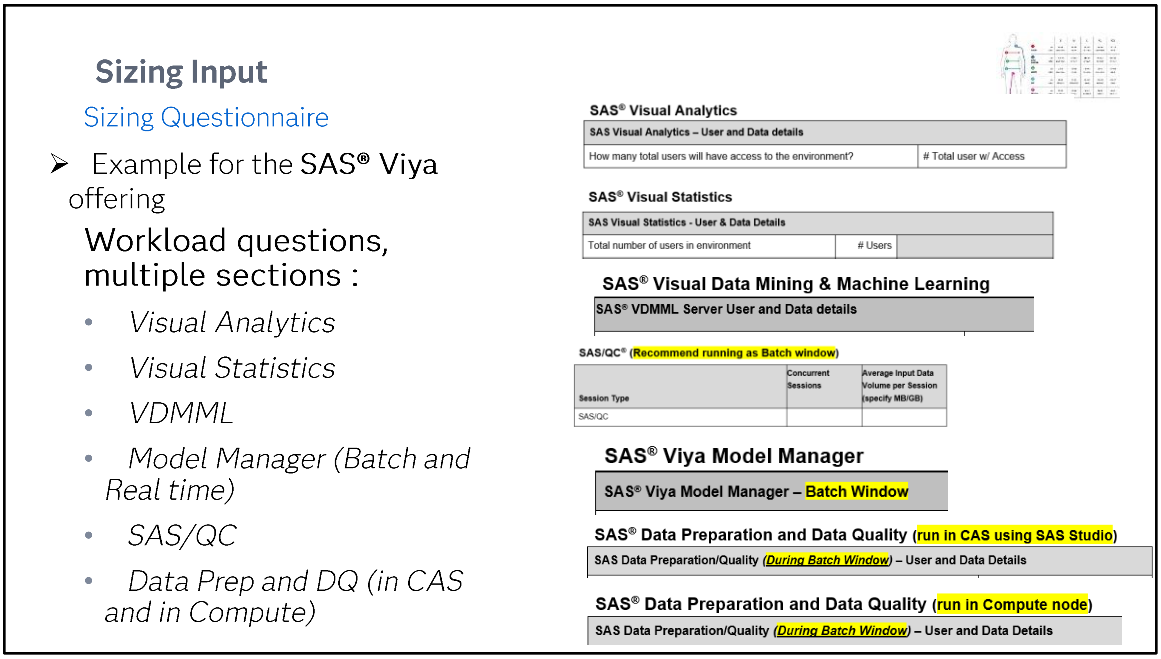
Existing SAS environment	
What version of SAS products are currently licensed?	SAS 9
Please specify the SAS products list that currently licensed	SAS Products List
Please specify the hardware vendor of existing environment	
Additional details that may assist in understanding the environment?	

Existing SAS environment Utilization						
Server Type		Number of server(s)	Configuration per server		Average CPU utilization (%)	Average memory utilization (%)
			CPU (specify cores/vCPUs)	RAM (GB)		
SAS 9.4	Compute					
	Metadata					
	Mid-Tier					

Then you have two new optional sections that have been recently added.

The first one is called Current “Viya 4 environment”, and it is for the situations where the customer has already deployed SAS Viya and wants to increase the capacity.
It's something that can be useful if the customer had already decided on the infrastructure but realize that there is not enough capacity for the expected or observed workload.

The second new section that has been added in the questionnaire is called “Modernization program-related questions”.
In case of a SAS 9 or Viya 3 modernization opportunity, this table can be used to collect useful information for a new sizing.
If the sizing team also has information on the current usage in SAS 9 or Viya 3, they can provide a better Hardware estimate for the Viya platform.



Sizing Input

Sizing Questionnaire

➢ Example for the **SAS® Viya** offering

Workload questions, multiple sections :

- *Visual Analytics*
- *Visual Statistics*
- *VDMML*
- *Model Manager (Batch and Real time)*
- *SAS/QC*
- *Data Prep and DQ (in CAS and in Compute)*

SAS® Visual Analytics

SAS Visual Analytics – User and Data details	
How many total users will have access to the environment?	# Total user w/ Access

SAS® Visual Statistics

SAS Visual Statistics - User & Data Details		
Total number of users in environment	# Users	

SAS® Visual Data Mining & Machine Learning

SAS® VDMML Server User and Data details

SAS/QC® (Recommend running as Batch window)

Session Type	**Concurrent Sessions**	**Average Input Data Volume per Session (specify MB/GB)**
SAS/QC		

SAS® Viya Model Manager

SAS® Viya Model Manager – Batch Window

SAS® Data Preparation and Data Quality (run in CAS using SAS Studio)

SAS Data Preparation/Quality (*During Batch Window*) – User and Data Details

SAS® Data Preparation and Data Quality (run in Compute node)

SAS Data Preparation/Quality (*During Batch Window*) – User and Data Details

Then if we look at the bulk of the questionnaire which is the workload profile, we have all the workload questions with the corresponding tables.
The workload questions that we see in the questionnaire depend on the offering that is associated to the sizing questionnaire.
The biggest the offering is, the more tables you will have to fill, of course.
So here in our example with the SAS Viya offering, we will have questions about "Visual Analytics" usage, "Visual Statistics" usage, VDMML usage, and so on.
Each time you will have to tell how many concurrent users, what is the biggest data, what are the average data size, et cetera, et cetera.
As a reminder, when we say 100 concurrent users we don't mean "100 users logged in", we mean "100 users opening or browsing a report at the same time"

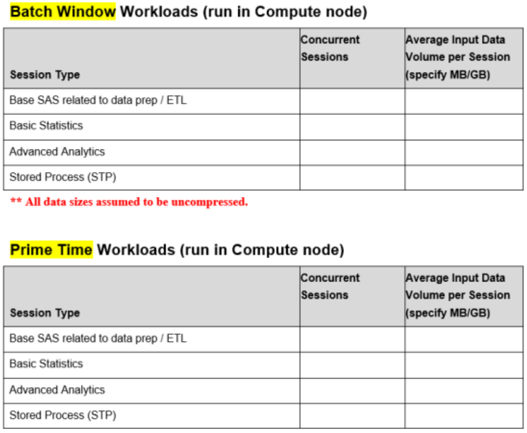
Sizing Input

Sizing Questionnaire

➢ Example for the SAS® Viya offering

Workload questions

- *Compute (Batch and Prime Time)*



Batch Window Workloads (run in Compute node)

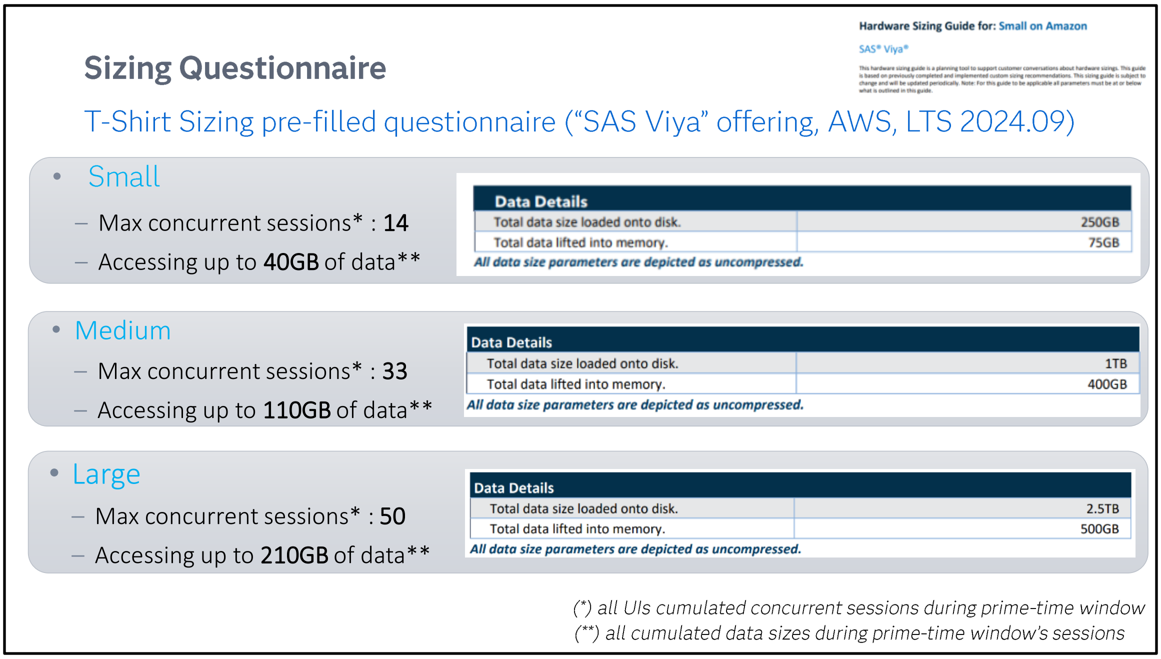
Session Type	Concurrent Sessions	Average Input Data Volume per Session (specify MB/GB)
Base SAS related to data prep / ETL		
Basic Statistics		
Advanced Analytics		
Stored Process (STP)		

** All data sizes assumed to be uncompressed.

Prime Time Workloads (run in Compute node)

Session Type	Concurrent Sessions	Average Input Data Volume per Session (specify MB/GB)
Base SAS related to data prep / ETL		
Basic Statistics		
Advanced Analytics		
Stored Process (STP)		

At the end of the questionnaire, we have the “Batch windows” and “prime time windows” workloads expected for the Compute node.
That’s where we run the SAS code.



Hardware Sizing Guide for: Small on Amazon

SAS® Viya®

This hardware sizing guide is a planning tool to support customer conversations about hardware sizings. This guide is based on previously completed and implemented custom sizing recommendations. This sizing guide is subject to change and will be updated periodically. Note: For this guide to be applicable all parameters must be at or below what is outlined in this guide.

Sizing Questionnaire

T-Shirt Sizing pre-filled questionnaire ("SAS Viya" offering, AWS, LTS 2024.09)

- Small
  - Max concurrent sessions* : **14**
  - Accessing up to **40GB** of data**

Data Details	
Total data size loaded onto disk.	250GB
Total data lifted into memory.	75GB

All data size parameters are depicted as uncompressed.

- Medium
  - Max concurrent sessions* : **33**
  - Accessing up to **110GB** of data**

Data Details	
Total data size loaded onto disk.	1TB
Total data lifted into memory.	400GB

All data size parameters are depicted as uncompressed.

- Large
  - Max concurrent sessions* : **50**
  - Accessing up to **210GB** of data**

Data Details	
Total data size loaded onto disk.	2.5TB
Total data lifted into memory.	500GB

All data size parameters are depicted as uncompressed.

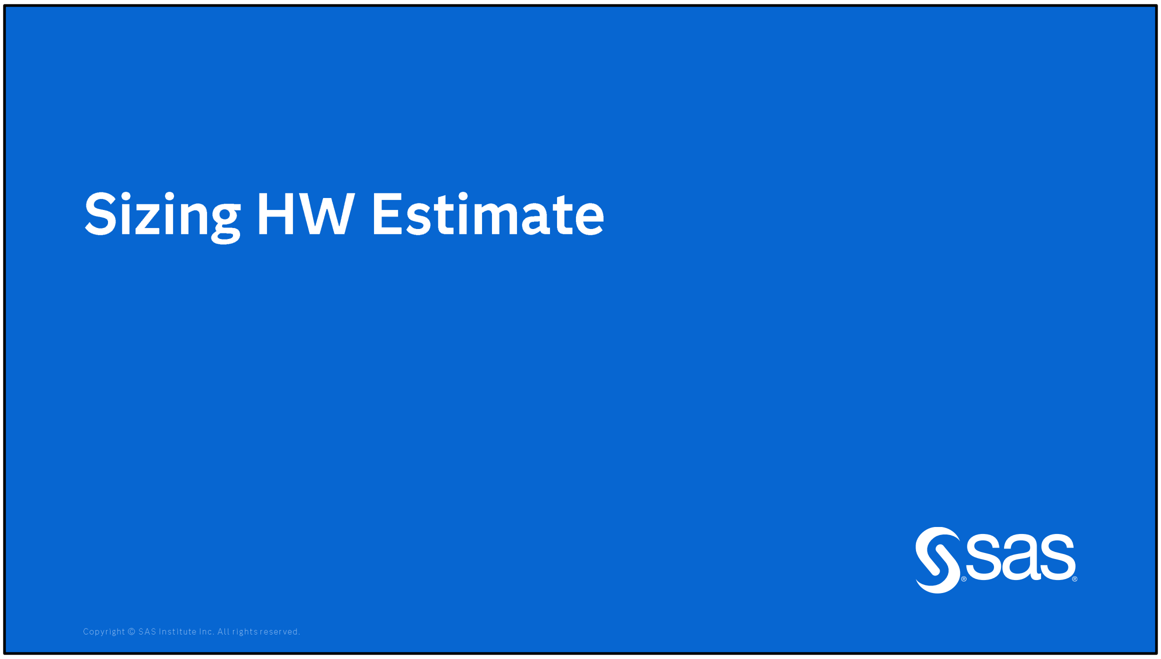
(*) all UIs cumulated concurrent sessions during prime-time window
(**) all cumulated data sizes during prime-time window's sessions

Finally, what we want to show here is a summary of the questionnaires associated to the small, medium and large T-shirt Sizings.
Because in the provide T-shirt sizing (or hardware guides), we have not only the results in term of required CPU/memory but also the associated questionnaire !

For example, with the small T-shirt sizing we have 250 Gigabytes of data on disk, and we load 75 Gigabytes of data in memory, in CAS.
If we sum all the potential concurrent sessions across all the user interfaces as noted in the questionnaire (for Visual analytics, Data prep and data quality, Visual statistics, Visual Data Mining & Machine Learning and any SAS code user interface) during the day we end up with around 14 sessions.
If we sum the associated data sizes, we obtain up to 35 Gigabytes of data being manipulated by the sessions.

And here are the numbers of the medium and large T-shirts scenarios.

These numbers are here to give an idea of the range of the customer cases in terms of users and data volumes across the three levels (small, medium and large.)
Depending on the offering, the data and concurrent sessions assumptions will slightly differ as there will be more or less products with associated users and data assumptions.

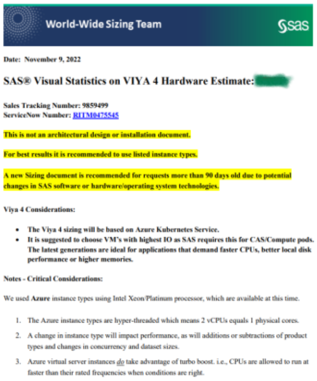


Sizing HW Estimate

Hardware Estimate

WW Sizing HW estimates - Structure

- Warnings (*“This is <u>not</u> an architectural design or installation document.”*)
- SAS Viya 4 general considerations
- Critical considerations
  - Specific to the Solution (when applicable)
  - Specific to the platform (AWS, Azure, GCP, RedHat OpenShift)
- General disclaimer *(“Any use or reliance on this estimate is at customer’s risk. SAS disclaims any liability with regards to…”)*
- Workload Assumptions (taken from the Questionnaire)
- Hardware Estimate
- General and storage assumptions
- Final disclaimers



World-Wide Sizing Team

Date: November 9, 2022

SAS® Visual Statistics on VIYA 4 Hardware Estimate: [redacted]

Sales Tracking Number: 9859499
ServiceNow Number: RITM0475545

This is not an architectural design or installation document.

For best results it is recommended to use listed instance types.

A new Sizing document is recommended for requests more than 90 days old due to potential changes in SAS software or hardware/operating system technologies.

Viya 4 Considerations:

- **The Viya 4 sizing will be based on Azure Kubernetes Service.**
- **It is suggested to choose VM’s with highest IO as SAS requires this for CAS/Compute pods. The latest generations are ideal for applications that demand faster CPUs, better local disk performance or higher memories.**

Notes - Critical Considerations:

We used **Azure** instance types using Intel Xeon/Platinum processor, which are available at this time.

1. The Azure instance types are hyper-threaded which means 2 vCPUs equals 1 physical cores.
2. A change in instance type will impact performance, as will additions or subtractions of product types and changes in concurrency and dataset sizes.
3. Azure virtual server instances <u>do</u> take advantage of turbo boost. i.e., CPUs are allowed to run at faster than their rated frequencies when conditions are right.

This module has a focus on the Hardware estimate which is the output of the Worldwide sizing process.

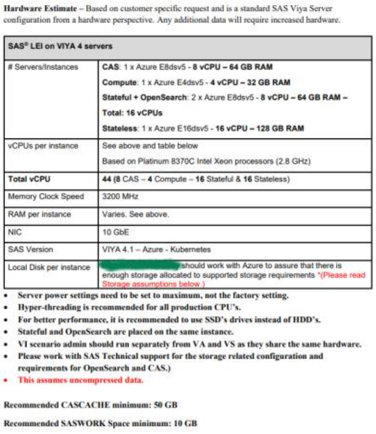
Here is the structure of the Hardware Estimates document.
First there are some warnings.
For example, this is not an architectural design or installation document, that's a very important point.
The sizing is just one of the inputs of the architecture design.
We can also find some general considerations about Viya 4 and the selected platform (such as I/O, CNI recommendations).
Then there are some critical considerations. Some of them are specific to the solution (Depending on the selected offering, you might see them or not).
There are also critical considerations related to the specific Kubernetes platform.
For example, for a sizing in Azure, there are some considerations related to the Azure Cloud infrastructure.
Then there is a standard disclaimer (that is included in every hardware estimates document provided worldwide sizing team)
Finally, there will be two things that are specific to the customer's case, and they appear in blue on the slide :
Workload assumptions and hardware estimates.
And at end of the document, some general storage assumptions and final disclaimers are written.
That's basically the content of your hardware estimates document that will come back from the worldwide sizing team.

Hardware Estimate

WW Sizing HW estimates

Hardware Estimate

- CPU and RAM estimates
- #CAS nodes (SMP or MPP)
- Instance types
- Recommended minimum CAS DISK CACHE
- Recommended minimum SASWORK
- Can be in 2 parts (for Viya & SAS 9.4 Solutions)



Hardware Estimate – Based on customer specific request and is a standard SAS Viya Server configuration from a hardware perspective. Any additional data will require increased hardware.

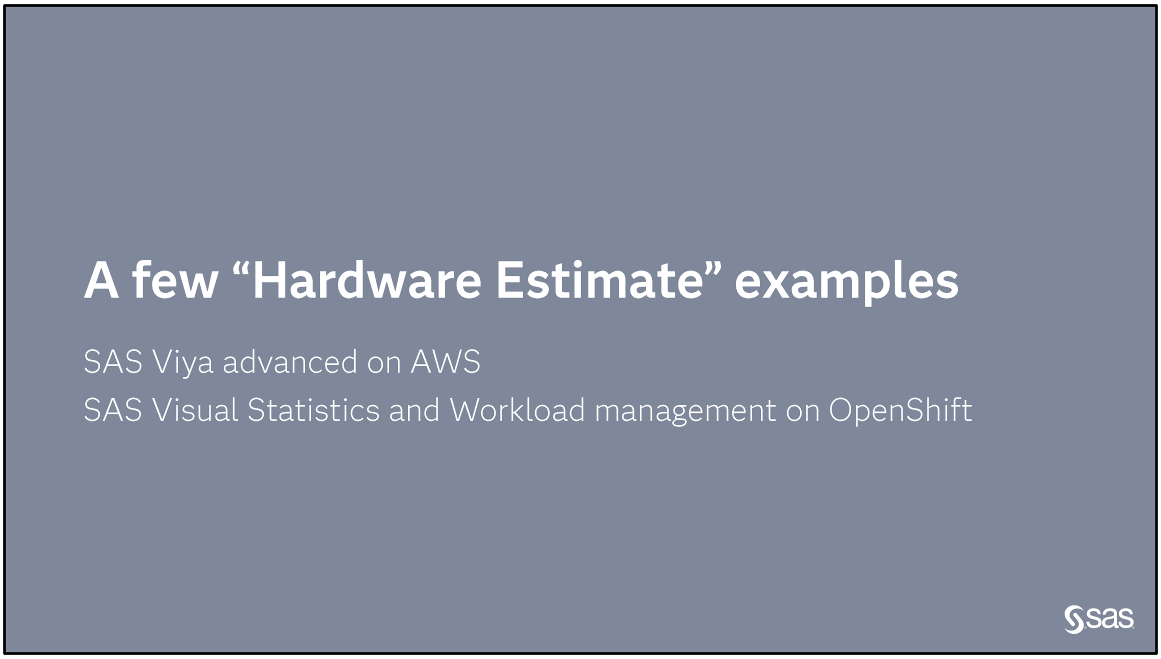
SAS® LEI on VIYA 4 servers	
# Servers/Instances	**CAS**: 1 x Azure E8dsv5 - **8 vCPU – 64 GB RAM** **Compute**: 1 x Azure E4dsv5 - **4 vCPU – 32 GB RAM** **Stateful + OpenSearch**: 2 x Azure E8dsv5 - **8 vCPU – 64 GB RAM –** **Total: 16 vCPUs** **Stateless**: 1 x Azure E16dsv5 - **16 vCPU – 128 GB RAM**
vCPUs per instance	See above and table below Based on Platinum 8370C Intel Xeon processors (2.8 GHz)
Total vCPU	**44** (**8** CAS – **4** Compute – **16** Stateful & **16** Stateless)
Memory Clock Speed	3200 MHz
RAM per instance	Varies. See above.
NIC	10 GbE
SAS Version	VIYA 4.1 – Azure - Kubernetes
Local Disk per instance	[redacted] should work with Azure to assure that there is enough storage allocated to supported storage requirements *(Please read Storage assumptions below.)

- **Server power settings need to be set to maximum, not the factory setting.**
- **Hyper-threading is recommended for all production CPU's.**
- **For better performance, it is recommended to use SSD's drives instead of HDD's.**
- **Stateful and OpenSearch are placed on the same instance.**
- **VI scenario admin should run separately from VA and VS as they share the same hardware.**
- **Please work with SAS Technical support for the storage related configuration and requirements for OpenSearch and CAS.)**
- **This assumes uncompressed data.**

Recommended CASCACHE minimum: 50 GB

Recommended SASWORK Space minimum: 10 GB

Now if we zoom specifically on the Hardware estimates which is the key deliverable here, we can see from the example's screenshot (on the right-hand side) that we have the CPU on RAM estimates.
We have the number of CAS nodes, whether it is SMP, symmetric massively processing or multiple parallel processing.
We see the Instance types, recommended minimum CAS disk cache and recommended minimum SAS work.
There are some solutions that need both platforms, SAS 9 and Viya.
In such case, you will have two parts in the sizing document.
Note that, in the HW estimates screenshot, SAS L-E-I stands for SAS Viya Law Enforcement Intelligence (a Viya 4 solution that includes SAS Visual Investigator)

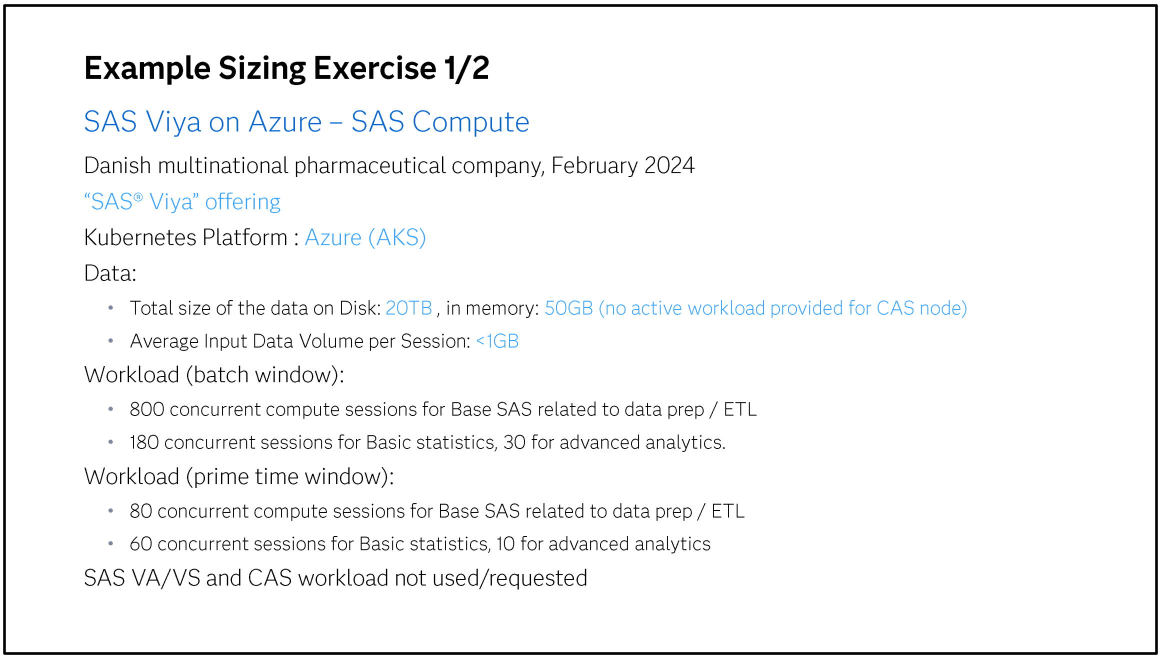


A few “Hardware Estimate” examples

SAS Viya advanced on AWS

SAS Visual Statistics and Workload management on OpenShift

So we will finish this module with a review of few examples to illustrate what a “bespoke” sizing could look like.
We will provide the context and main assumptions, then the corresponding HW estimates.
These are real Sizings example (that have been anonymized).



Example Sizing Exercise 1/2

SAS Viya on Azure – SAS Compute

Danish multinational pharmaceutical company, February 2024

“SAS® Viya” offering

Kubernetes Platform : Azure (AKS)

Data:

- Total size of the data on Disk: 20TB , in memory: 50GB (no active workload provided for CAS node)
- Average Input Data Volume per Session: <1GB

Workload (batch window):

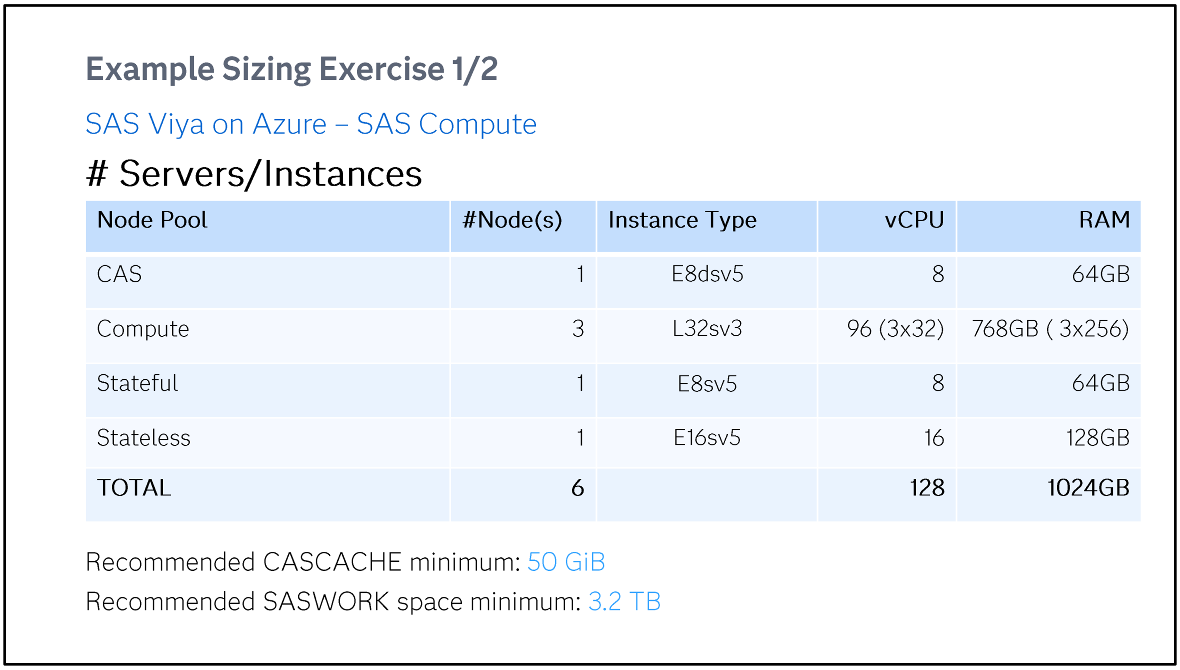
- 800 concurrent compute sessions for Base SAS related to data prep / ETL
- 180 concurrent sessions for Basic statistics, 30 for advanced analytics.

Workload (prime time window):

- 80 concurrent compute sessions for Base SAS related to data prep / ETL
- 60 concurrent sessions for Basic statistics, 10 for advanced analytics

SAS VA/VS and CAS workload not used/requested

Let's start with this sizing, which was done for a customers in Danmark, in the healthcare sector in February 2024.
The offering is SAS Viya
The Kubernetes platform was in Azure, so running in AKS, Azure Kubernetes Services.
In terms of data, we can see that there is about 20 Terabytes of data on disk to process with SAS.
In memory, we only assume a workload of 50 gigabytes, as no active workload provided for the CAS node.
In average the data set size is below one Gigabyte.
The requested workload is only for the SAS Base computing runtime.
Up to 800 concurrent sessions could be run during the batch window for the data preparation steps in SAS Base.
No specific workload requirement were provided for the “SAS Visual Analytics”, “SAS Visual Statistics” or the “SAS® Data Preparation and Data Quality” applications.



Example Sizing Exercise 1/2

SAS Viya on Azure – SAS Compute

Servers/Instances

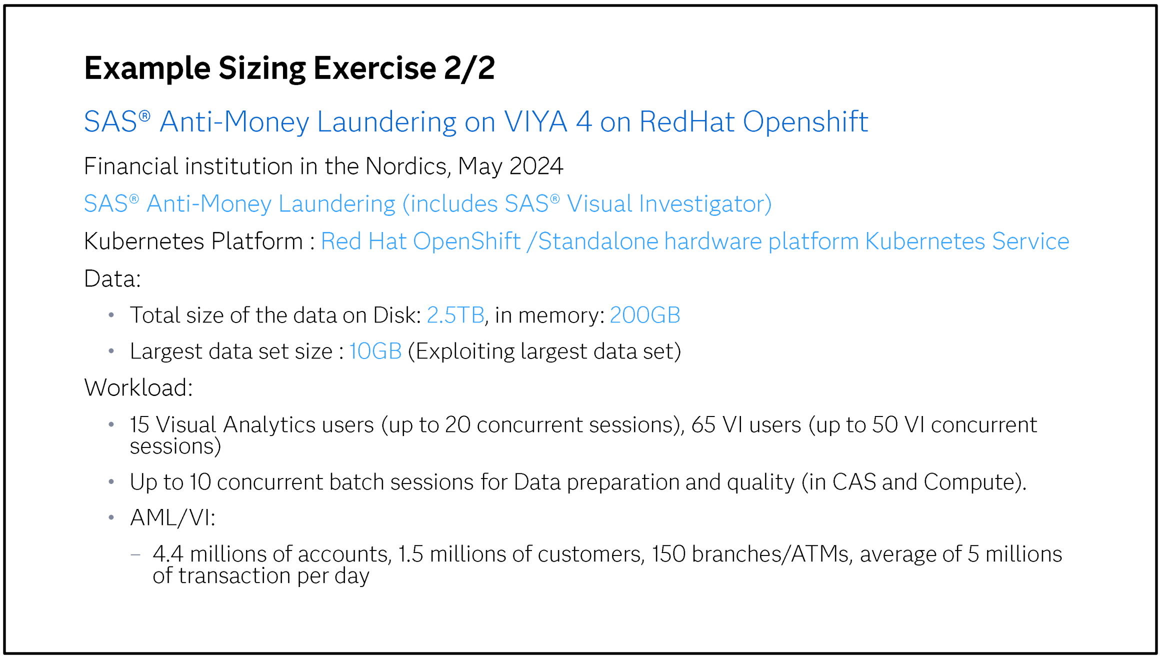
Node Pool	#Node(s)	Instance Type	vCPU	RAM
CAS	1	E8dsv5	8	64GB
Compute	3	L32sv3	96 (3x32)	768GB (3x256)
Stateful	1	E8sv5	8	64GB
Stateless	1	E16sv5	16	128GB
TOTAL	6		128	1024GB

Recommended CASCACHE minimum: 50 GiB

Recommended SASWORK space minimum: 3.2 TB

What we get for these assumptions, basically, is this kind of sizing with one CAS node, three compute nodes, and 1 node for the stateful and stateless pods.
It is in the Azure Cloud, you can see the Azure instance types here, and you can see that for CAS, we need 48 cores, and 384 gigabytes of memory.
So now if we do the sum of all the resources, we end up with 128 vCPU and around 1 Terabyte of memory.
For this specific case, most of the CPU and memory is allocated to the compute nodes and the required space for the SASWORK is significant.
These estimates make sense as most of the workload is expected in SAS Base programs with a lot of concurrent sessions.

This is typically the kind of information that you will get from the sizing, and that is helpful with infrastructure setup.



Example Sizing Exercise 2/2

SAS® Anti-Money Laundering on VIYA 4 on RedHat Openshift

Financial institution in the Nordics, May 2024

SAS® Anti-Money Laundering (includes SAS® Visual Investigator)

Kubernetes Platform : Red Hat OpenShift /Standalone hardware platform Kubernetes Service

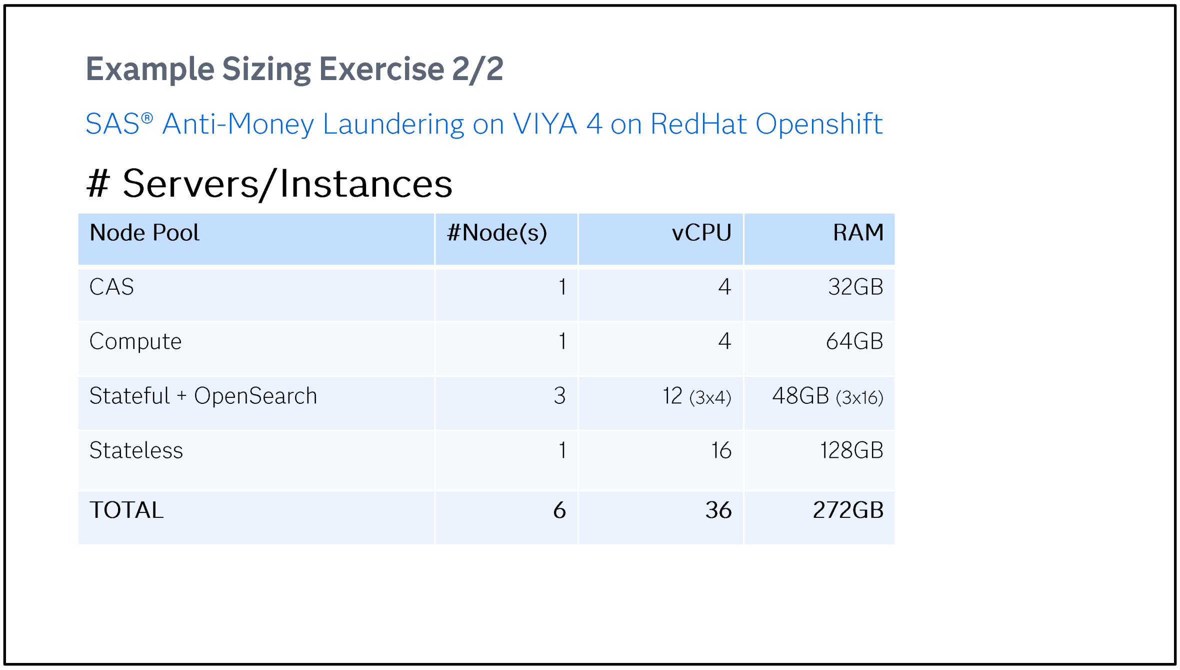
Data:

- Total size of the data on Disk: 2.5TB, in memory: 200GB
- Largest data set size : 10GB (Exploiting largest data set)

Workload:

- 15 Visual Analytics users (up to 20 concurrent sessions), 65 VI users (up to 50 VI concurrent sessions)
- Up to 10 concurrent batch sessions for Data preparation and quality (in CAS and Compute).
- AML/VI:
  - 4.4 millions of accounts, 1.5 millions of customers, 150 branches/ATMs, average of 5 millions of transaction per day

Let's start with this sizing, which was done for a Financial institution in the Nordics in May 2024.
The offering is Anti-Money Laundering on VIYA 4 (which includes SAS Visual Investigator), so we can see that we can also have sizing requests with SAS Viya solutions too.
The Kubernetes platform was RedHat Openshift running on-premises.
In terms of data, we can see that there's about 2.5 Terabytes on disk and 200 Gigabytes to be loaded into memory.
In terms of workload, there are 15 Visual Analytics users, up to 20 concurrent sessions, 65 Visual Investigator users (with up to 50 VI concurrent sessions).
Regarding the A-M-L specific metrics there are 4.4 millions of accounts, 1.5 millions of customers, 150 branches (including the ATMs), and an average of 5 millions of transactions per day.
So overall, it is a pretty busy environment…



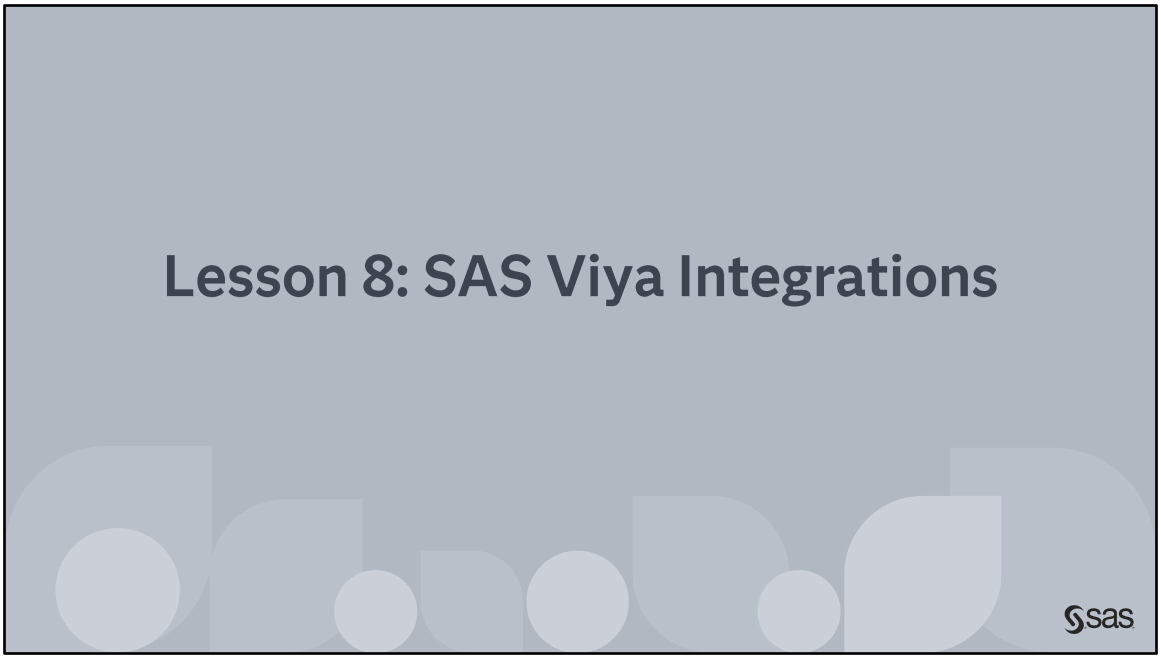
Example Sizing Exercise 2/2

SAS® Anti-Money Laundering on VIYA 4 on RedHat Openshift

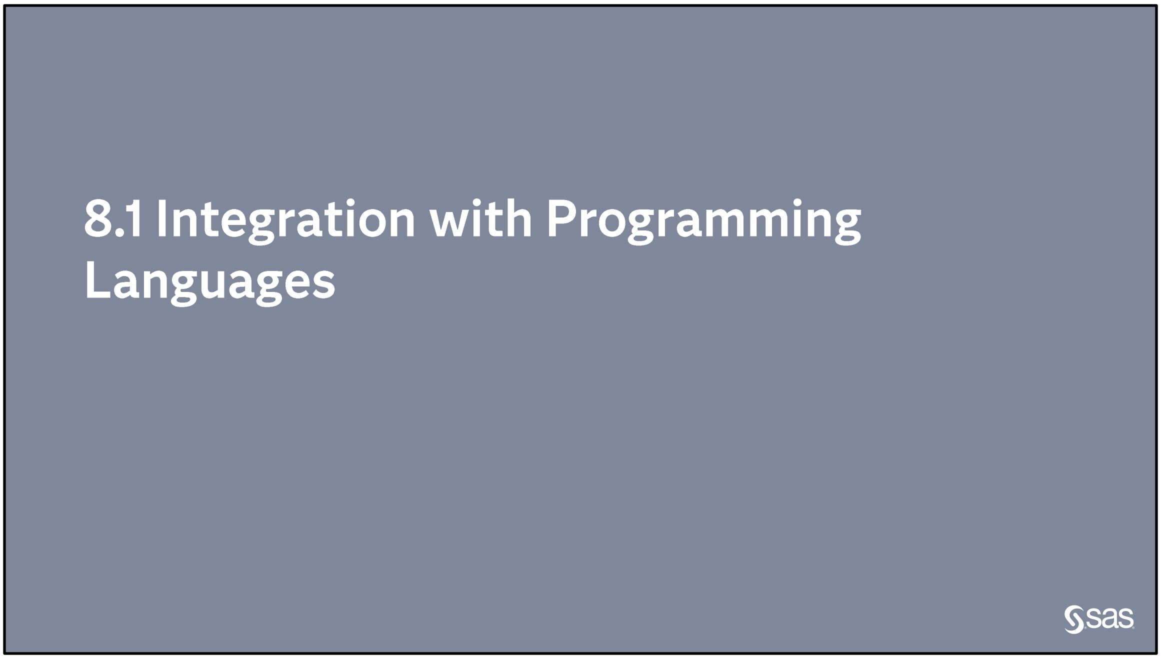
Servers/Instances

Node Pool	#Node(s)	vCPU	RAM
CAS	1	4	32GB
Compute	1	4	64GB
Stateful + OpenSearch	3	12 (3x4)	48GB (3x16)
Stateless	1	16	128GB
TOTAL	6	36	272GB

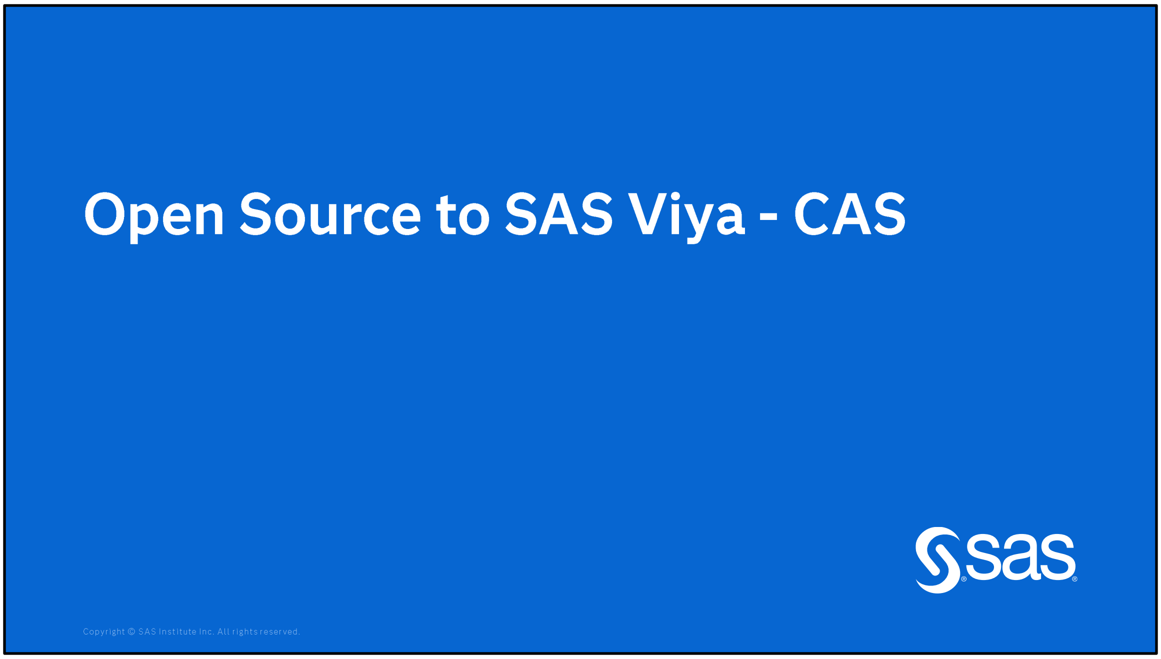
What we get for these assumptions, basically, is a sizing with one CAS node, one compute node, three nodes for the stateful and open search pods, and one node for the stateless pods.
Since SAS Viya runs on an OpenShift /Standalone hardware platform (and not in cloud) there are no instance type. The estimates only contains a number of vCPU and memory size requirements for the nodes.
If we do the sum of all the resources, we end up with something around 36vCPU and around 300 Gigabytes of memory.
For this estimate, there is no specific minimum space requirements for the CAS disk cache and SASWORK. However, the Estimates contains the recommendation that the customer should work with the provider of the “Red Hat OpenShift Standalone hardware platform” to ensure that there is enough storage allocated to support the SAS Viya storage requirements.



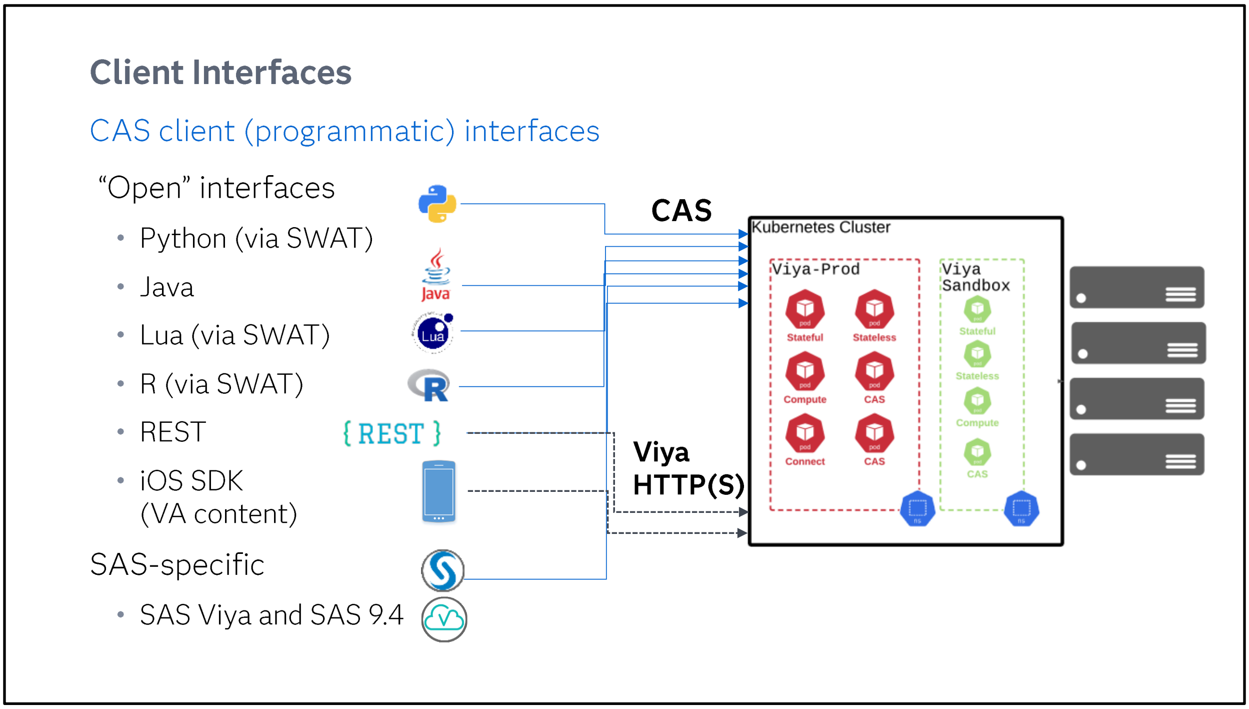
Lesson 8: SAS Viya Integrations



8.1 Integration with Programming Languages



Open Source to SAS Viya - CAS



Client Interfaces

CAS client (programmatic) interfaces

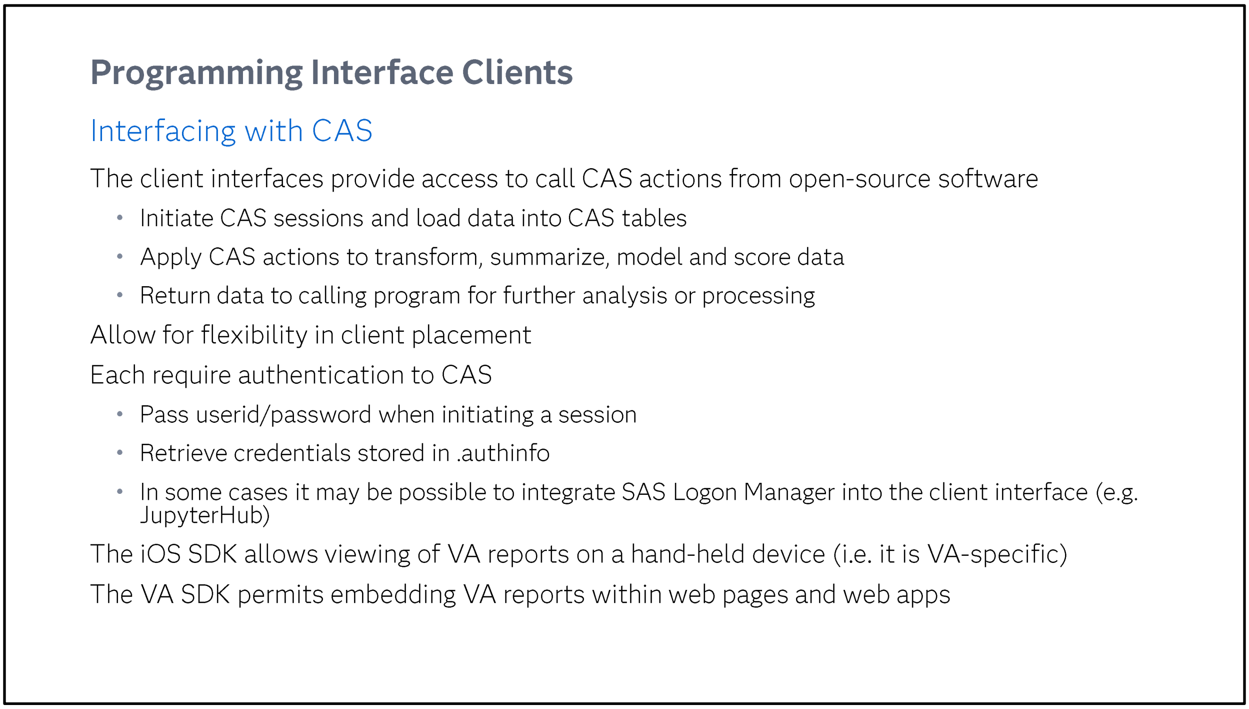
“Open” interfaces

- Python (via SWAT)
- Java
- Lua (via SWAT)
- R (via SWAT)
- REST
- iOS SDK
  (VA content)

SAS-specific

- SAS Viya and SAS 9.4

When we think about CAS programming client interfaces, we categorize them as "open" and SAS-specific interfaces.
Under the open interfaces, we have:
Python, Lua, and R, all of which utilize the SWAT interface.
Java, which also connects programmatically.
The ability to program natively using the REST interface, enabling calls to SAS Viya functions.
And the iOS SDK, tailored for VA content.
On the SAS-specific side, interfaces can come from another SAS Viya environment or from a SAS 9.4 environment.



Programming Interface Clients

Interfacing with CAS

The client interfaces provide access to call CAS actions from open-source software

- Initiate CAS sessions and load data into CAS tables
- Apply CAS actions to transform, summarize, model and score data
- Return data to calling program for further analysis or processing

Allow for flexibility in client placement

Each require authentication to CAS

- Pass userid/password when initiating a session
- Retrieve credentials stored in .authinfo
- In some cases it may be possible to integrate SAS Logon Manager into the client interface (e.g. JupyterHub)

The iOS SDK allows viewing of VA reports on a hand-held device (i.e. it is VA-specific)

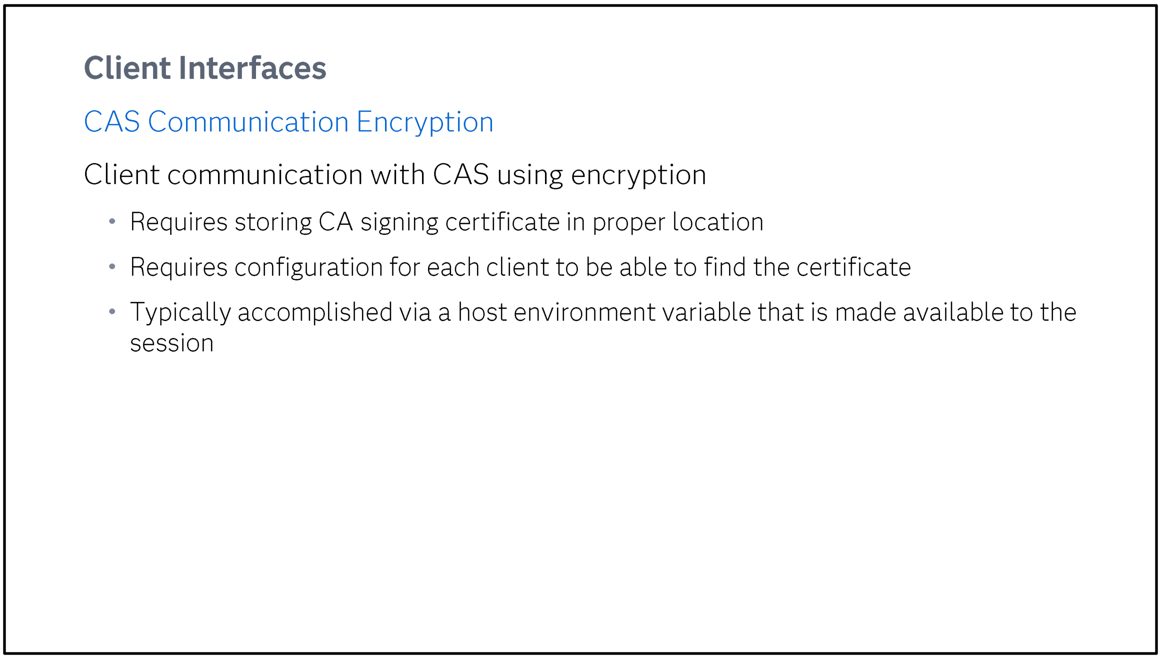
The VA SDK permits embedding VA reports within web pages and web apps

Interfacing with the CAS server.
The client interfaces provide access to core CAS actions from open-source software, such as Python, R, Lua, and others.
You can initiate CAS sessions and load data into CAS tables.
You can also apply CAS actions to transform, summarize, model, and score data.
And once that’s complete, you can return the data to the calling program, perhaps for further analysis, processing, or presentation.

This setup allows for flexibility in client placement.

Each method requires authentication to CAS.
This can be done by passing a user ID and password when initiating a session or by retrieving credentials stored in a .authinfo file.
In some cases, it may be possible to integrate SAS Logon Manager into the client interface.
For example, this can be achieved when using JupyterHub.

The iOS SDK allows viewing of VA reports on handheld devices and is specific to VA.
The VA SDK permits embedding VA reports within web pages and web applications.



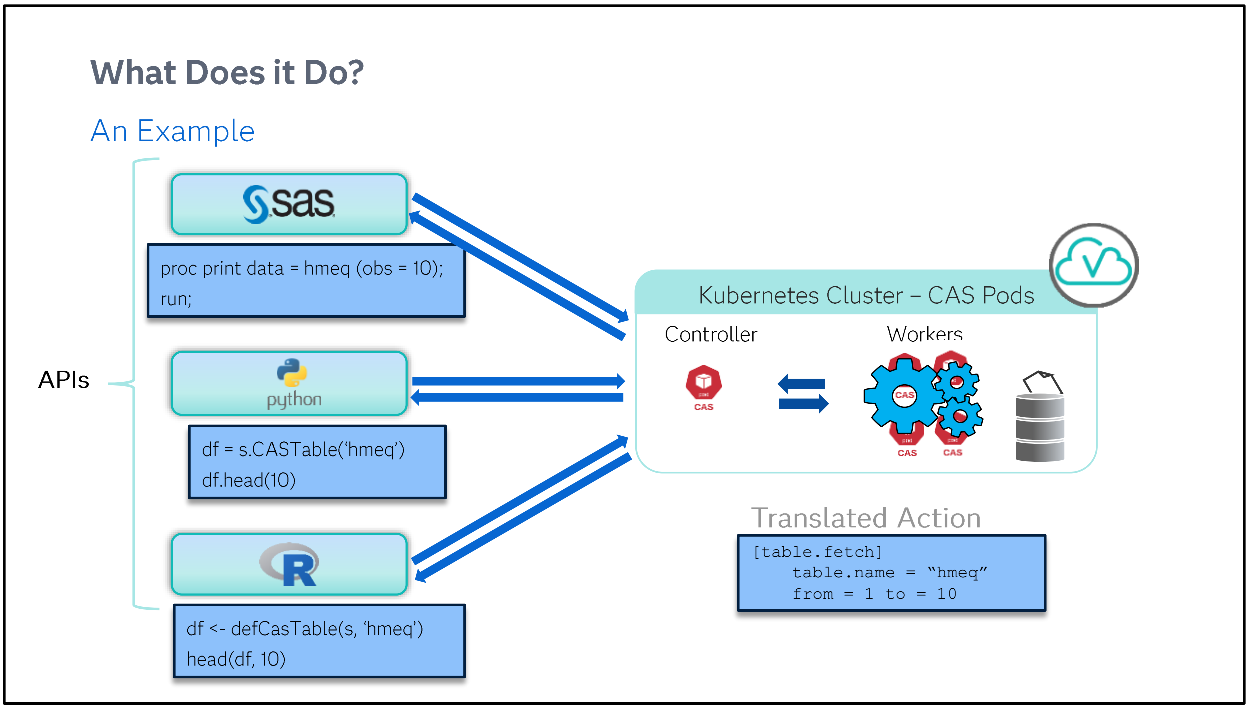
Client Interfaces

CAS Communication Encryption

Client communication with CAS using encryption

- Requires storing CA signing certificate in proper location
- Requires configuration for each client to be able to find the certificate
- Typically accomplished via a host environment variable that is made available to the session

As I mentioned earlier, communication with CAS is encrypted.
So, client communication with CAS requires encryption, which involves storing the CA signing certificate in the proper location.
It also requires configuration for each client to be able to locate that certificate.
Typically, this is accomplished using a host environment variable that is made available to the session.



What Does it Do?

An Example

```
proc print data = hmeq (obs = 10);
run;
```

```
df = s.CASTable('hmeq')
df.head(10)
```

```
df <- defCasTable(s, 'hmeq')
head(df, 10)
```

Translated Action

```
[table.fetch]
    table.name = "hmeq"
    from = 1 to = 10
```

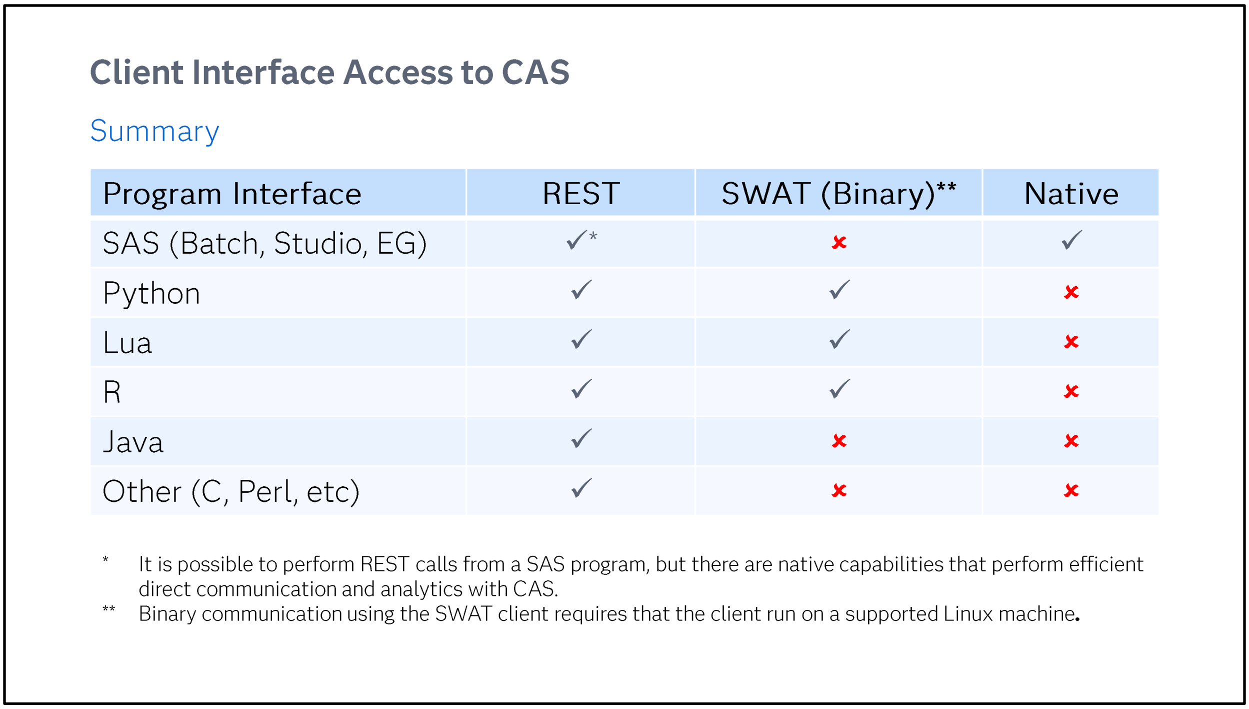
So, how does this work?

When we consider various programming languages, we can see that a native connection to CAS is utilized through SAS.

If we were using Python, the connection and the code might look slightly different, but it achieves the same functionality.

Similarly, for R, the syntax would differ, but the actions performed remain consistent.

The key takeaway here is that regardless of the language, at the back end, the same CAS actions are executed to ensure consistent results.



Client Interface Access to CAS

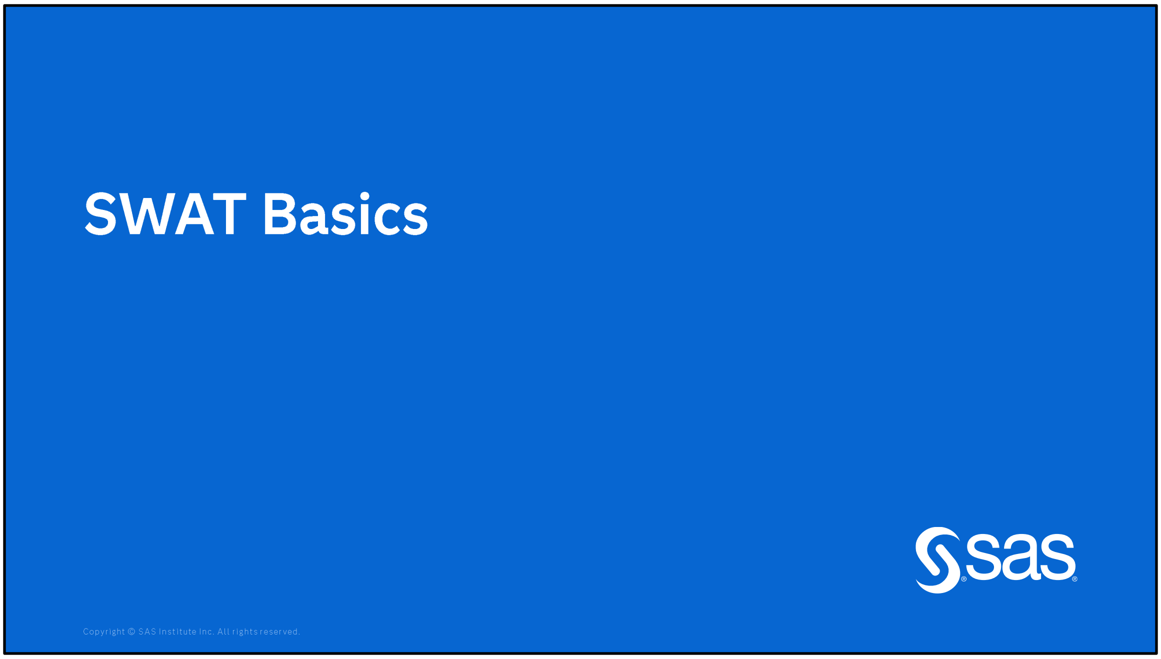
Summary

Program Interface	REST	SWAT (Binary)**	Native
SAS (Batch, Studio, EG)	✓*	✗	✓
Python	✓	✓	✗
Lua	✓	✓	✗
R	✓	✓	✗
Java	✓	✗	✗
Other (C, Perl, etc)	✓	✗	✗

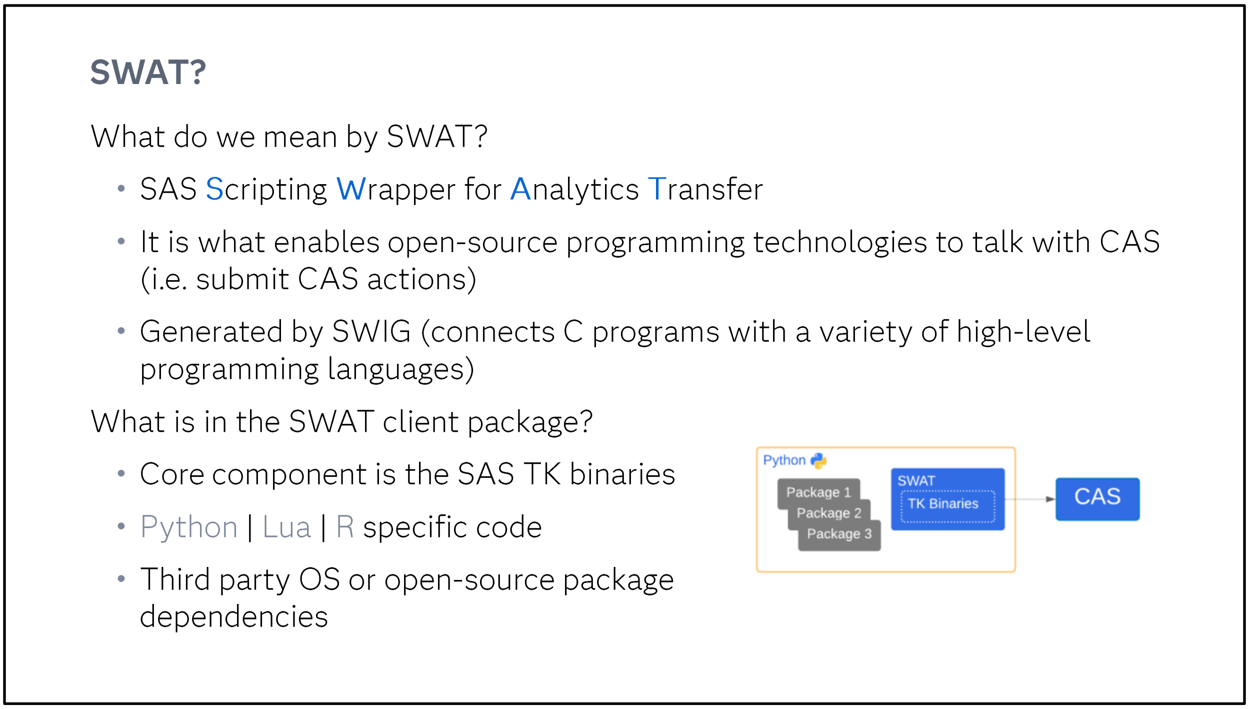
* It is possible to perform REST calls from a SAS program, but there are native capabilities that perform efficient direct communication and analytics with CAS.

** Binary communication using the SWAT client requires that the client run on a supported Linux machine.

So, in summary, let’s take a look at the client interface options for accessing CAS, as shown in this table.
It outlines various programming interfaces and their methods of connection.
For SAS, whether you’re working in Batch, SAS Studio, or Enterprise Guide, there’s native support for accessing CAS.
While REST calls are also possible, native capabilities provide efficient direct communication and analytics with CAS.
Moving to Python, Lua, and R, you have the flexibility to use the REST interface or the SWAT client.
However, it’s worth noting that binary communication via the SWAT client requires the client to run on a supported Linux machine.
For Java, the REST interface is the primary method of access.
And for other languages like C or Perl, REST calls remain an available option for interfacing with CAS.



SWAT Basics



SWAT?

What do we mean by SWAT?

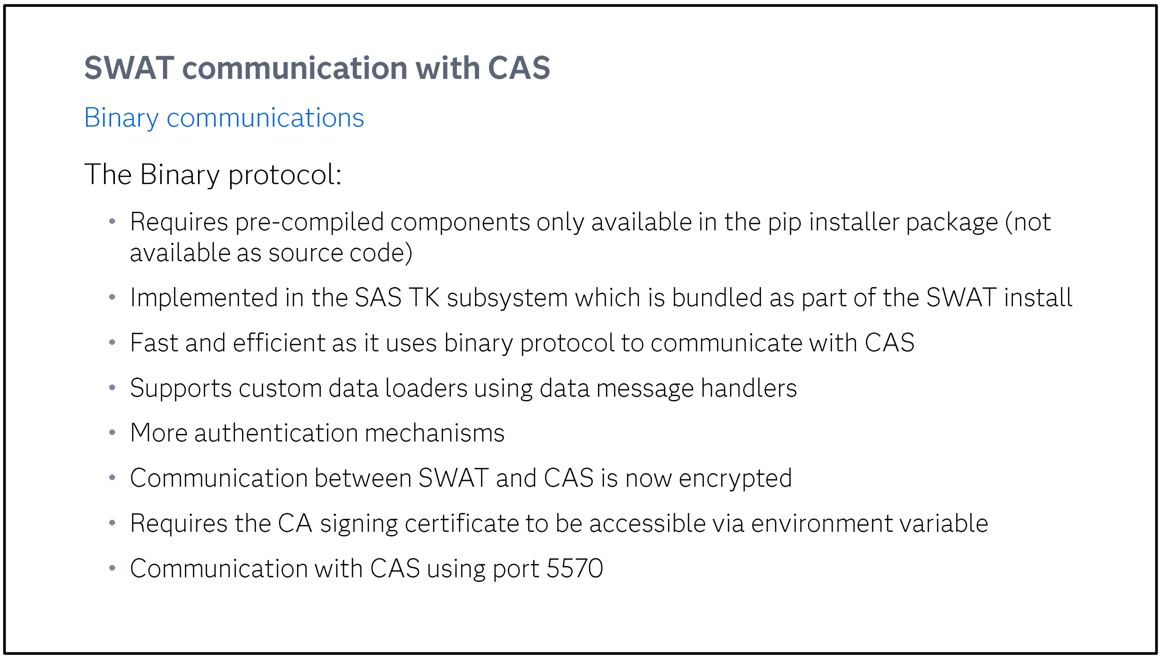
- SAS Scripting Wrapper for Analytics Transfer
- It is what enables open-source programming technologies to talk with CAS (i.e. submit CAS actions)
- Generated by SWIG (connects C programs with a variety of high-level programming languages)

What is in the SWAT client package?

- Core component is the SAS TK binaries
- Python | Lua | R specific code
- Third party OS or open-source package dependencies

So, what do we mean by SWAT?
SWAT stands for SAS Scripting Wrapper for Analytics Transfer.
This is what enables open-source programming technologies, or programming clients, to communicate with CAS and submit CAS actions.
It’s generated using SWIG, which connects C programs with a variety of high-level programming languages.

Now, what’s included in the SWAT client package?
The core component is the SAS TK binaries.
Additionally, there’s program-specific code tailored for Python, Lua, or R.
And, of course, there may also be third-party OS or open-source package dependencies included in the package.



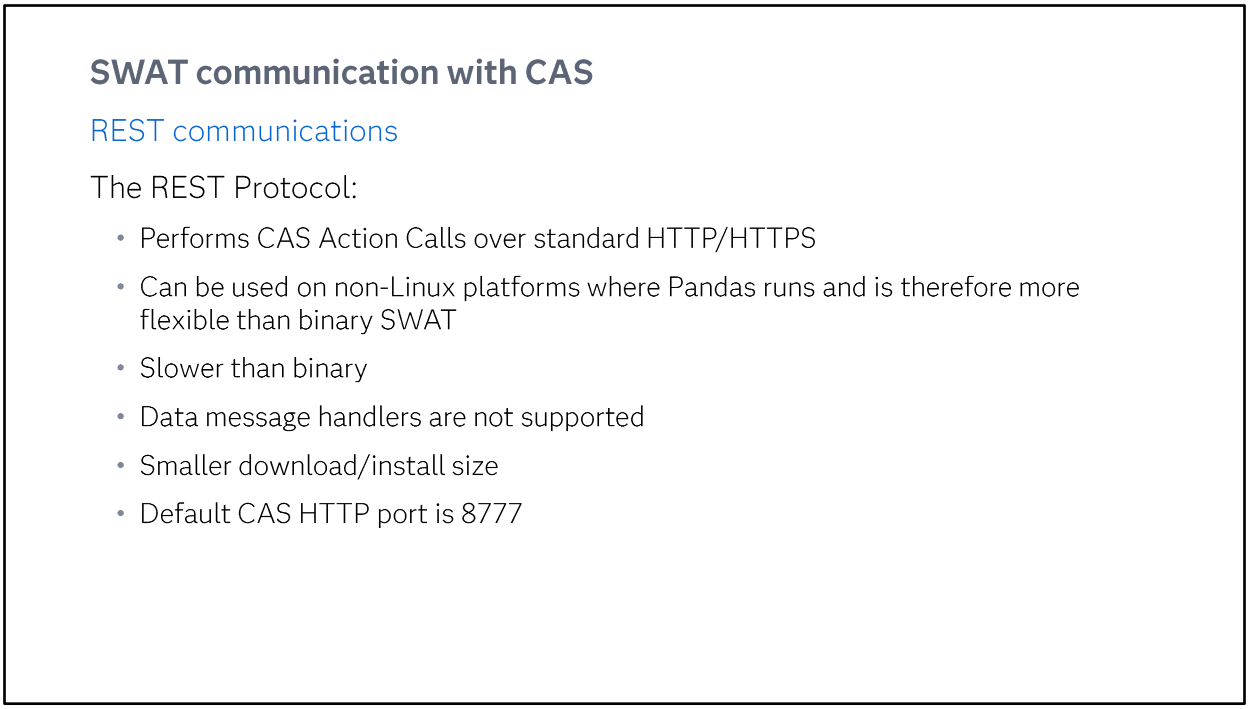
SWAT communication with CAS

Binary communications

The Binary protocol:

- Requires pre-compiled components only available in the pip installer package (not available as source code)
- Implemented in the SAS TK subsystem which is bundled as part of the SWAT install
- Fast and efficient as it uses binary protocol to communicate with CAS
- Supports custom data loaders using data message handlers
- More authentication mechanisms
- Communication between SWAT and CAS is now encrypted
- Requires the CA signing certificate to be accessible via environment variable
- Communication with CAS using port 5570

When we talk about SWAT communication with CAS, let's focus on the binary protocol.
The binary protocol requires precompiled components, which are only available via the pip installer package and are not provided as source code.
This protocol is implemented within the SAS TK subsystem, which comes bundled as part of the SWAT installation.
It’s designed to be fast and efficient, leveraging the binary protocol to handle communication with CAS.
Additionally, it supports custom data loaders through data message handlers, giving you flexibility in how data is managed.
The binary protocol also offers more authentication mechanisms compared to the REST protocol.
Communication between SWAT and CAS is encrypted, which is critical for security concerns.
However, this requires that the CA signing certificate is accessible, typically via an environment variable.
Finally, communication occurs with CAS using port 5570.



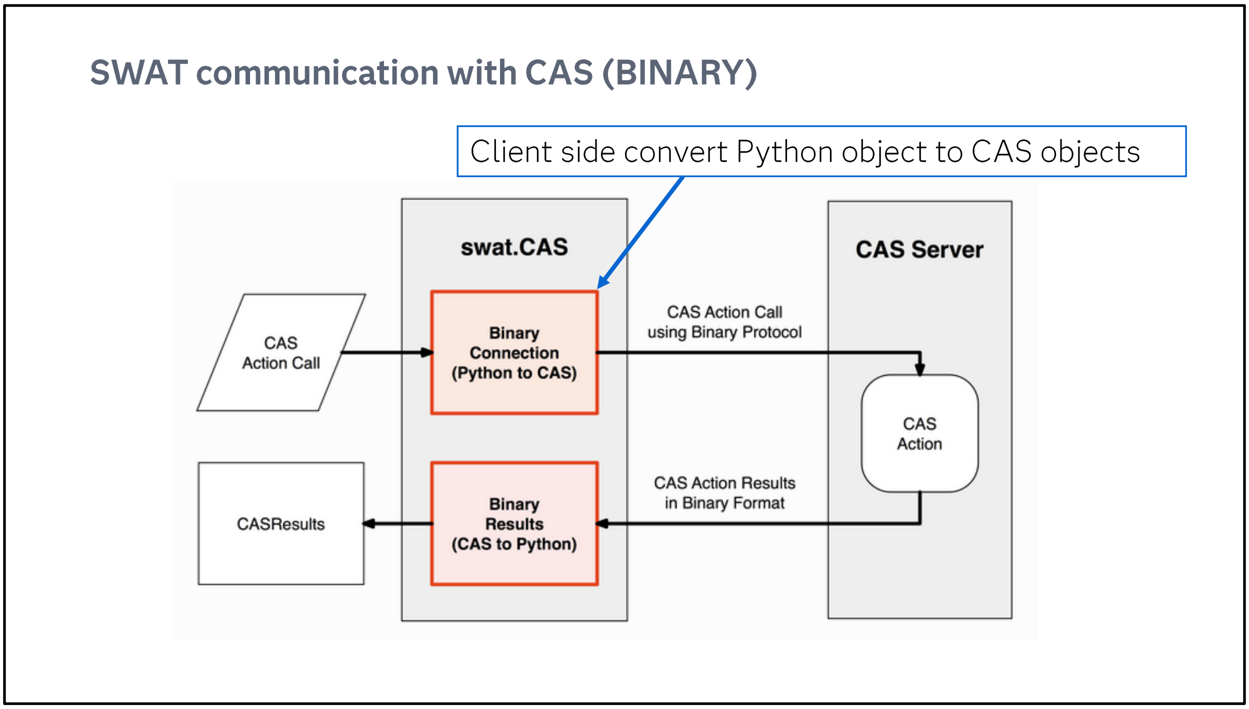
SWAT communication with CAS

REST communications

The REST Protocol:

- Performs CAS Action Calls over standard HTTP/HTTPS
- Can be used on non-Linux platforms where Pandas runs and is therefore more flexible than binary SWAT
- Slower than binary
- Data message handlers are not supported
- Smaller download/install size
- Default CAS HTTP port is 8777

Now let's discuss the REST protocol for SWAT communication with CAS.
The REST protocol allows you to perform CAS action calls over standard HTTP or HTTPS.
This makes it compatible with non-Linux platforms where Pandas runs, offering greater flexibility compared to the binary SWAT package.
However, the REST protocol is slower than the binary protocol and does not support data message handlers.
On the plus side, it has a smaller download and installation size, making it easier to deploy in certain environments.
The default CAS HTTP port for REST communication is 8777.



SWAT communication with CAS (BINARY)

Client side convert Python object to CAS objects

swat.CAS

CAS Server

CAS
Action Call

Binary
Connection
(Python to CAS)

CAS Action Call
using Binary Protocol

CAS
Action

CAS Action Results
in Binary Format

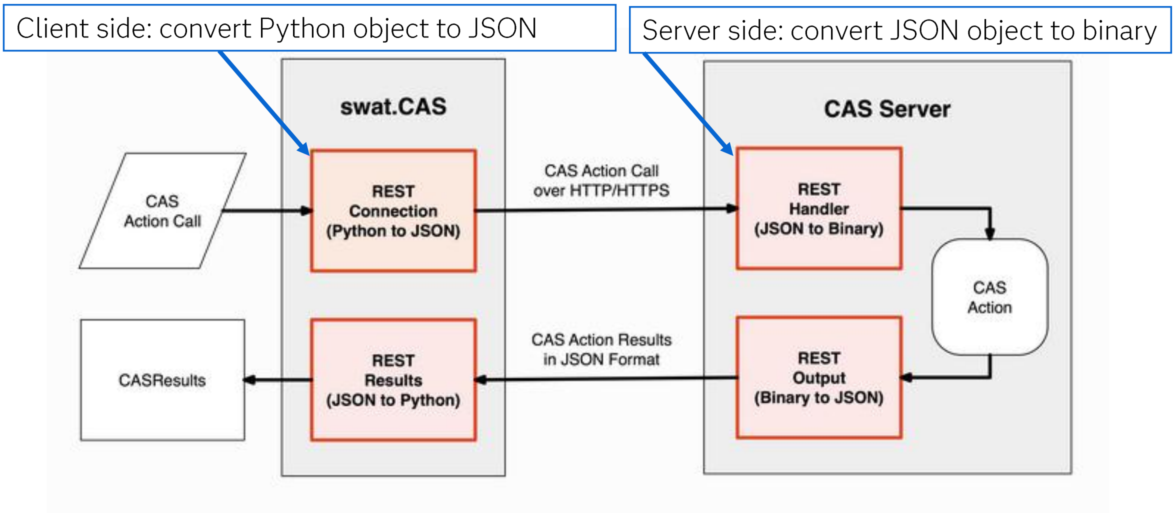
CASResults

Binary
Results
(CAS to Python)

Let's take a closer look at the binary protocol communication.
In this diagram, we can see the interaction between the Python client and the CAS server.
I’ve installed the SWAT package in my Python environment, and when I call a CAS action, it initiates the binary connection using the binary protocol.

The conversion between Python objects to CAS objects occurs on the client side.
Once the CAS action is performed, the results are returned using the binary protocol.
Finally, we translate the results and pass them back to the calling program.

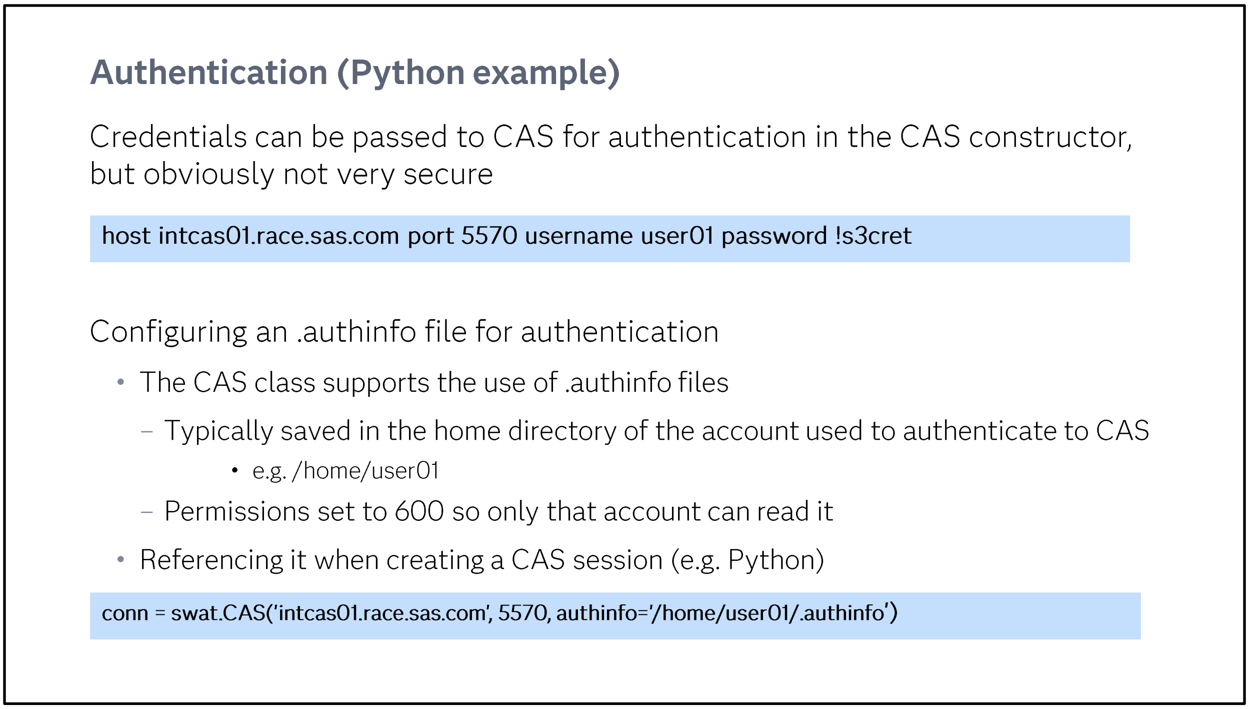
SWAT communication with CAS (REST)


Client side: convert Python object to JSON
Server side: convert JSON object to binary
swat.CAS
CAS Server
CAS Action Call
REST Connection (Python to JSON)
CAS Action Call over HTTP/HTTPS
REST Handler (JSON to Binary)
CAS Action
CASResults
REST Results (JSON to Python)
CAS Action Results in JSON Format
REST Output (Binary to JSON)

Now, let’s look at the REST communication.
In this example, using Python again, we have our programming client.

This time, when we call CAS, the SWAT client converts the Python object to JSON. The JSON object is then sent over an HTTPS session to the CAS server, making the CAS action call.

On the server side, we convert that JSON object back to the binary protocol to perform the CAS action. The results are then sent back, and this time, we convert the binary data to JSON format. The JSON is passed back over the HTTP/HTTPS session to the calling program, where we convert it back into a Python object.



Authentication (Python example)

Credentials can be passed to CAS for authentication in the CAS constructor, but obviously not very secure

```
host intcas01.race.sas.com port 5570 username user01 password !s3cret
```

Configuring an .authinfo file for authentication

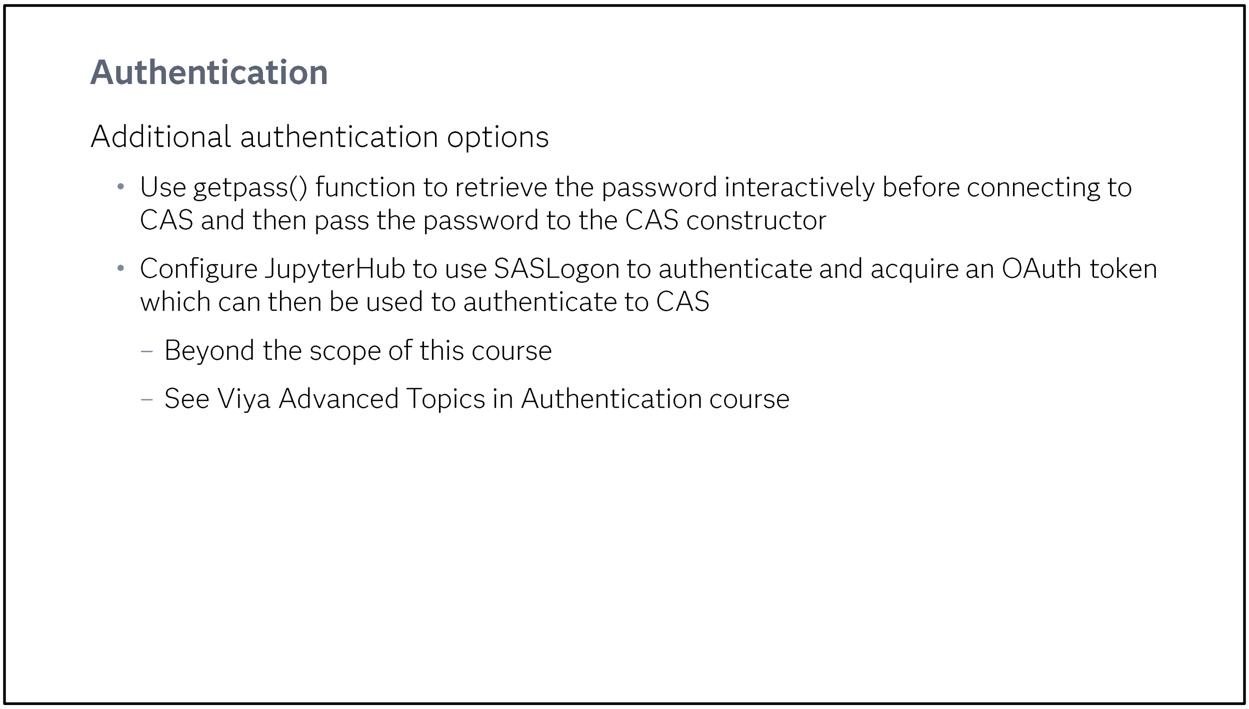
- The CAS class supports the use of .authinfo files
  - Typically saved in the home directory of the account used to authenticate to CAS
    - e.g. /home/user01
  - Permissions set to 600 so only that account can read it
- Referencing it when creating a CAS session (e.g. Python)

```
conn = swat.CAS(‘intcas01.race.sas.com’, 5570, authinfo=’/home/user01/.authinfo’)
```

In this Python example, when we authenticate to a host, we can pass the credentials directly in the CAS constructor.
However, as you can see, this is not very secure because the credentials are passed in clear text.
In this case, we are connecting to the host on port 5570 and authenticating as user01 with the password specified.

A better way to handle this is by configuring an .authinfo file for authentication.
The CAS class supports the use of .authinfo files for this purpose.
These files are typically saved in the home directory of the account used to authenticate to CAS.
In our example, the file is stored under the user01 account.
It's important to set the permissions on this file to 600 so that only the account used for authentication can read it.

When creating the CAS session, we reference this .authinfo file, which is located under the user01 directory.



Authentication

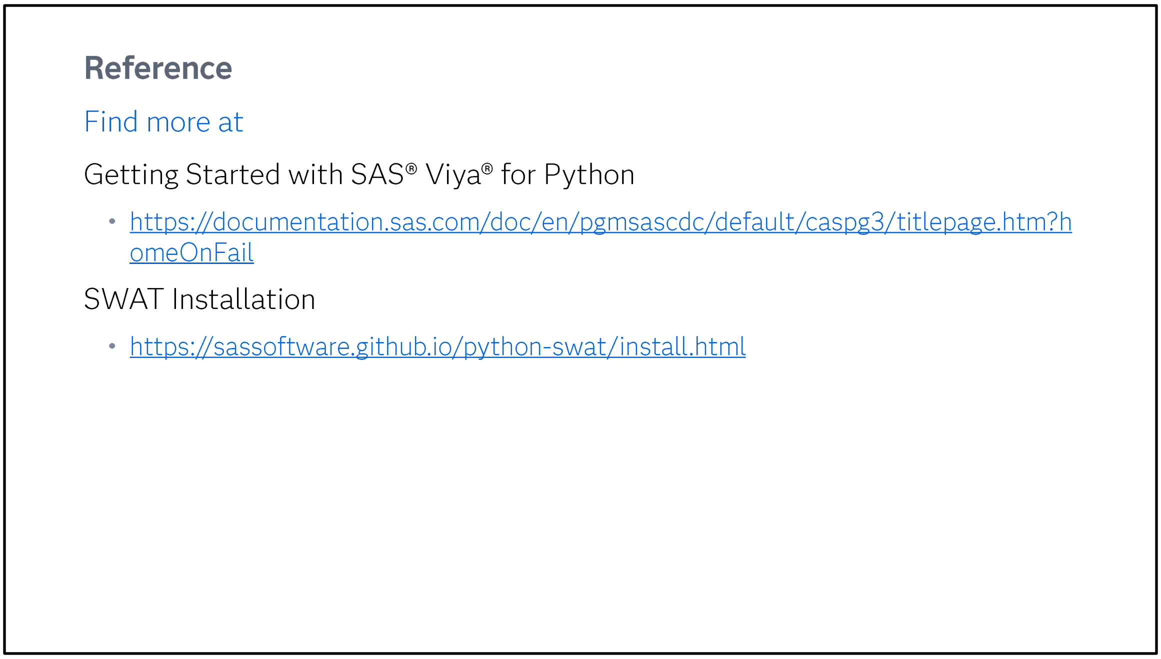
Additional authentication options

- Use getpass() function to retrieve the password interactively before connecting to CAS and then pass the password to the CAS constructor
- Configure JupyterHub to use SASLogon to authenticate and acquire an OAuth token which can then be used to authenticate to CAS
  - Beyond the scope of this course
  - See Viya Advanced Topics in Authentication course

There are additional authentication options available.

For example, you could use the getpass() function to retrieve the password interactively before connecting to CAS, and then pass the password to the CAS constructor.

Another option, if you're using JupyterHub, is to configure it to use SAS Logon for authentication.
This will allow you to acquire an OAuth token, which can then be used to authenticate to CAS.
Now, these topics go beyond the scope of this course, but they are covered in the Viya Advanced Topics in Authentication course available on learn.sas.com



Reference

Find more at

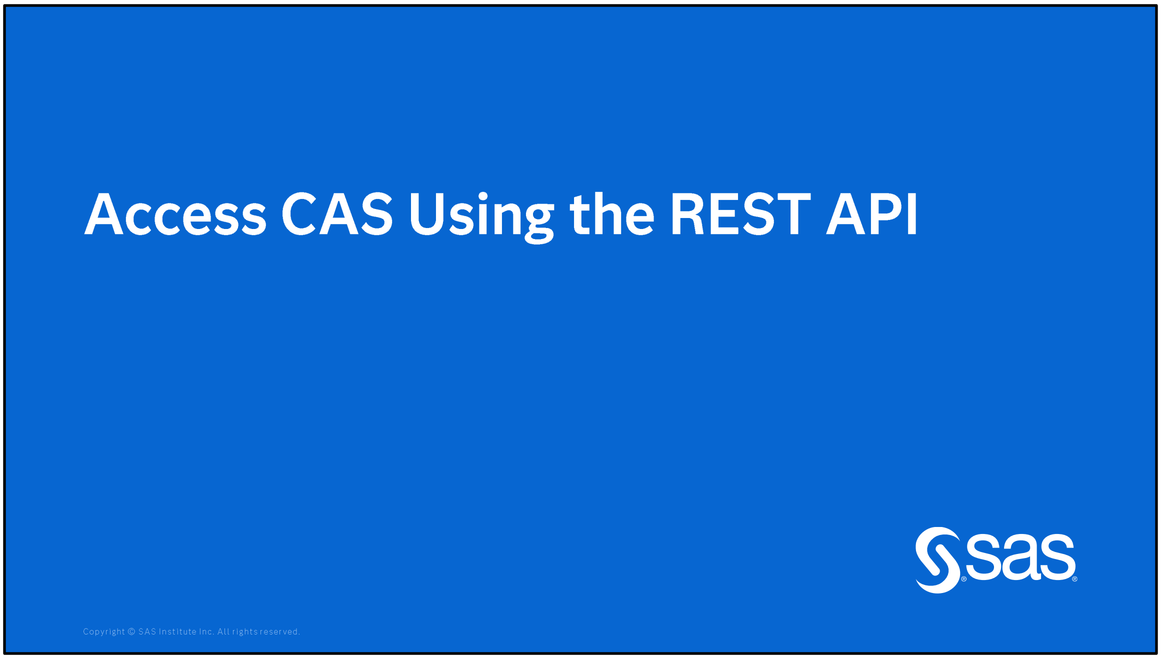
Getting Started with SAS® Viya® for Python

- https://documentation.sas.com/doc/en/pgmsascdc/default/caspg3/titlepage.htm?homeOnFail

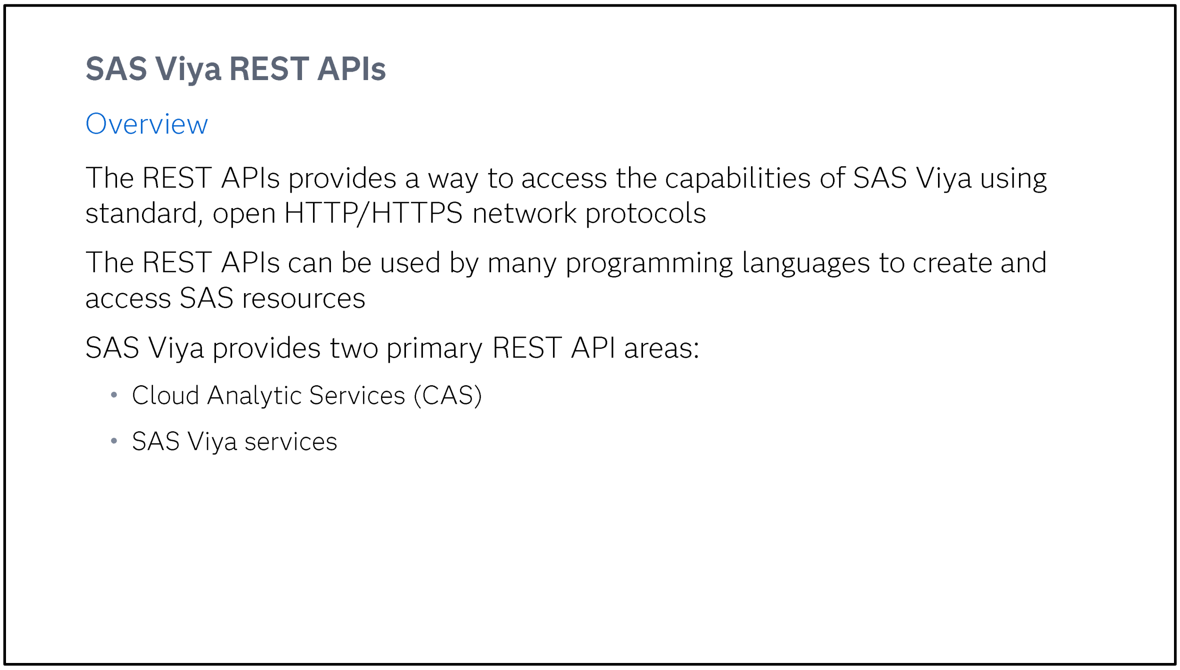
SWAT Installation

- https://sassoftware.github.io/python-swat/install.html

And finally, here are a couple of references for more information.
For getting started with SAS® Viya® for Python, you can visit the link provided: Getting Started with SAS Viya for Python.
Additionally, for SWAT installation, you can find the details at: SWAT Installation Guide.



Access CAS Using the REST API



SAS Viya REST APIs

Overview

The REST APIs provides a way to access the capabilities of SAS Viya using standard, open HTTP/HTTPS network protocols

The REST APIs can be used by many programming languages to create and access SAS resources

SAS Viya provides two primary REST API areas:

- Cloud Analytic Services (CAS)
- SAS Viya services

The REST APIs provide a way to access the capabilities of SAS Viya using standard, open HTTP or HTTPS protocols.

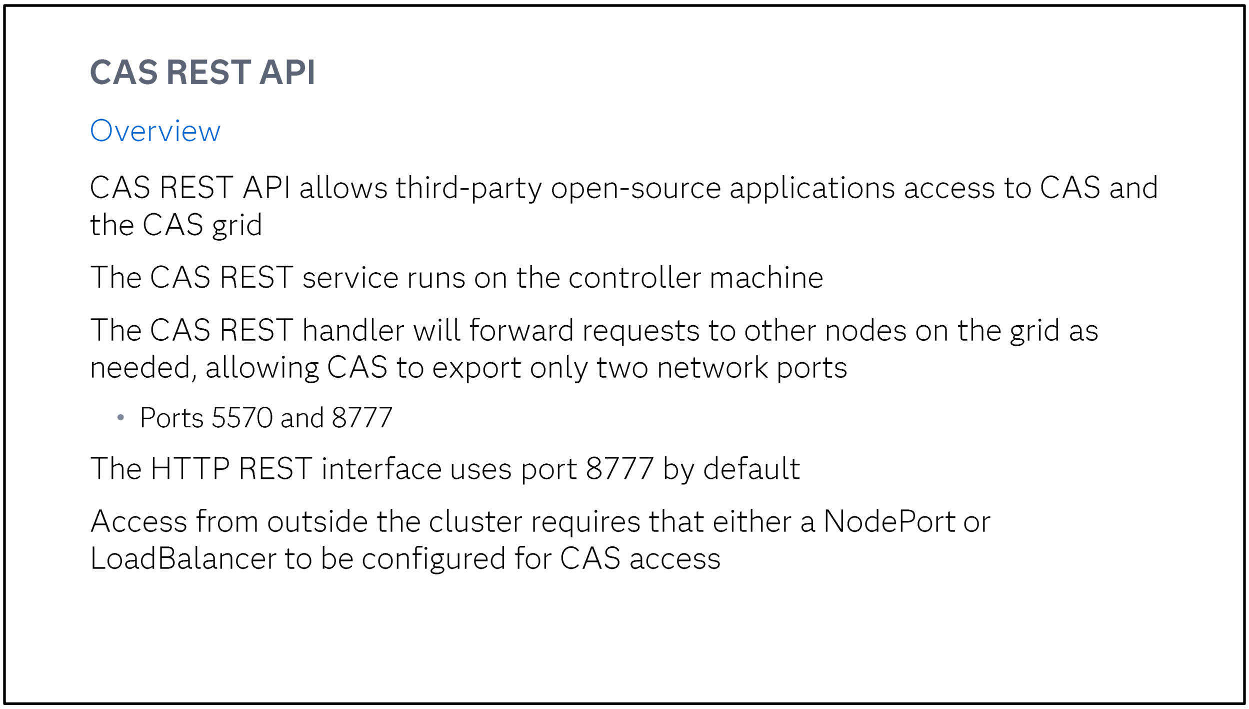
The REST APIs can be used by many programming languages to create and access SAS resources. This flexibility is one of the key benefits of using the REST APIs.

SAS Viya offers two primary areas of REST APIs:
The first is for Cloud Analytic Services, or CAS,
and the second is for accessing SAS Viya microservices.

TAG_Audio
The REST APIs are available for application developers, data scientists, and administrators.
On the screenshot on the right here, you can see the information from developer.sas.com.
But in this module, we will focus on using the CAS REST APIs.

TAG_Audio
There are API operations for executing CAS actions, managing CAS sessions, monitoring the system, and inspecting the CAS grid.
And again, these are documented on developer.sas.com, which is the main resource for finding the documentation and examples for this.

TAG_Print
See: REST APIs for SAS Cloud Analytics Services (CAS) | SAS
https://developer.sas.com/guides/restapis/cas-rest.html



CAS REST API

Overview

CAS REST API allows third-party open-source applications access to CAS and the CAS grid

The CAS REST service runs on the controller machine

The CAS REST handler will forward requests to other nodes on the grid as needed, allowing CAS to export only two network ports

- Ports 5570 and 8777

The HTTP REST interface uses port 8777 by default

Access from outside the cluster requires that either a NodePort or LoadBalancer to be configured for CAS access

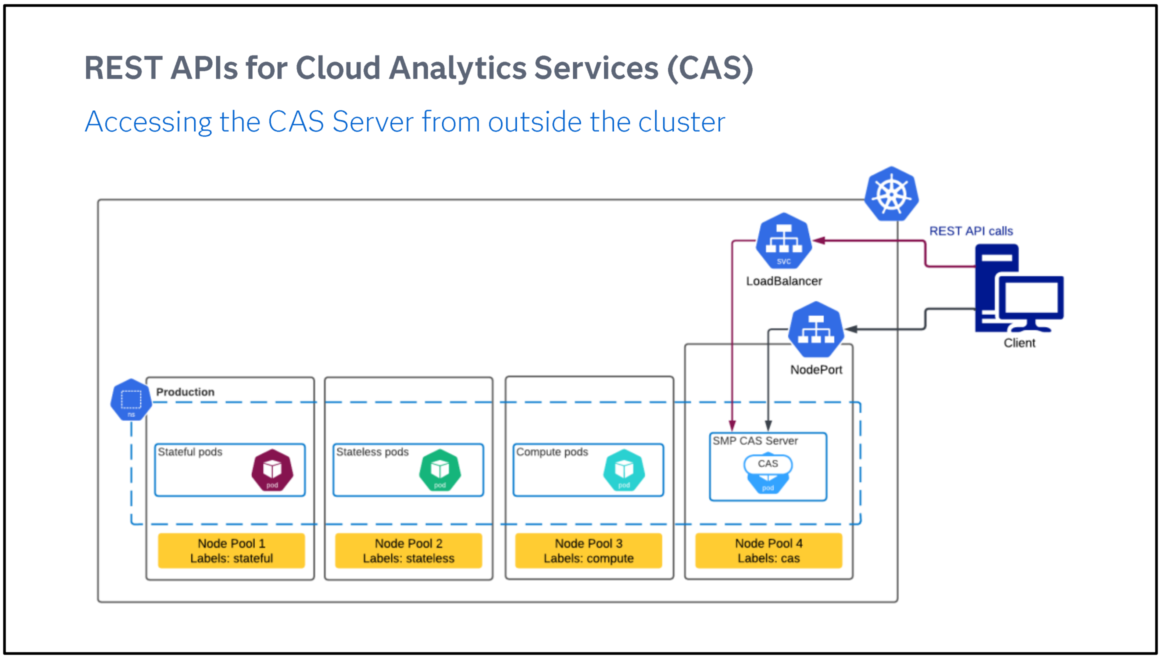
The CAS REST API allows third-party open-source applications to access CAS and the CAS grid.

The CAS REST service runs on the controller machine.

The CAS REST handler forwards requests to other nodes on the grid as needed.
This design allows CAS to expose only two network ports: 5570 and 8777.

The HTTP REST interface uses port 8777 by default.

Accessing CAS from outside the cluster requires configuring either a NodePort or a LoadBalancer for CAS access.



REST APIs for Cloud Analytics Services (CAS)

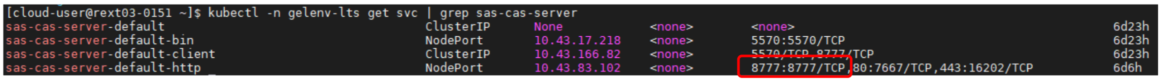
Accessing the CAS Server from outside the cluster

So here, we can see our programming client running outside the Kubernetes cluster, making REST calls. As we discussed earlier, there are two options for accessing the CAS server from outside the cluster: you can either go through a NodePort configuration or use a LoadBalancer service.

CAS REST API

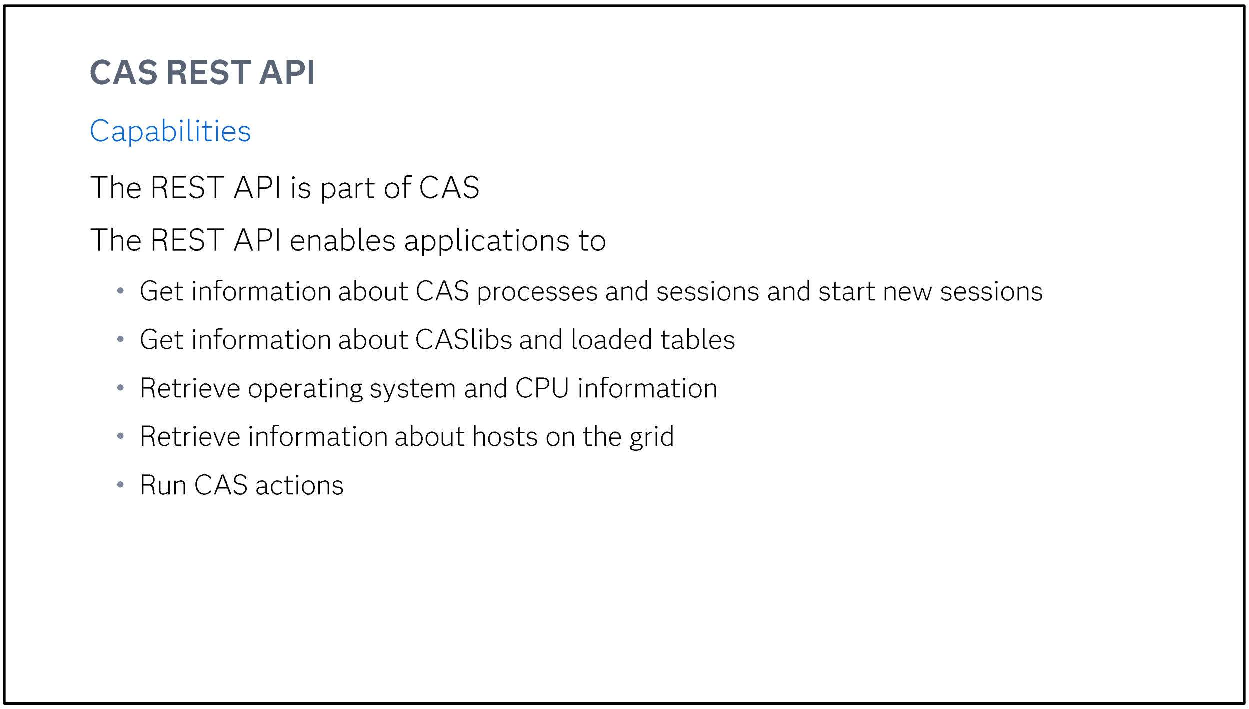
SAS Viya deployment example

This example shows that NodePort 8777 has been assigned to port 8777 post deployment



```
[cloud-user@rext03-0151 ~]$ kubectl -n gelenv-lts get svc | grep sas-cas-server
sas-cas-server-default                         ClusterIP      None           <none>        <none>                                             6d23h
sas-cas-server-default-bin                     NodePort       10.43.17.218   <none>        5570:5570/TCP                                      6d23h
sas-cas-server-default-client                  ClusterIP      10.43.166.82   <none>        5570/TCP,8777/TCP                                  6d23h
sas-cas-server-default-http                    NodePort       10.43.83.102   <none>        8777:8777/TCP,80:7667/TCP,443:16202/TCP            6d6h
```

Looking at a deployment example, this shows that NodePort 8777 has been assigned to port 8777 as a post deployment step.
We can see here that we're performing a GET on the services and filtering on the SAS CAS server.
If we examine the default HTTP service for the SAS CAS server, we can confirm that it is running on a NodePort.
You can see the address for the NodePort, and in this example, port 8777 has been mapped directly to port 8777.



CAS REST API

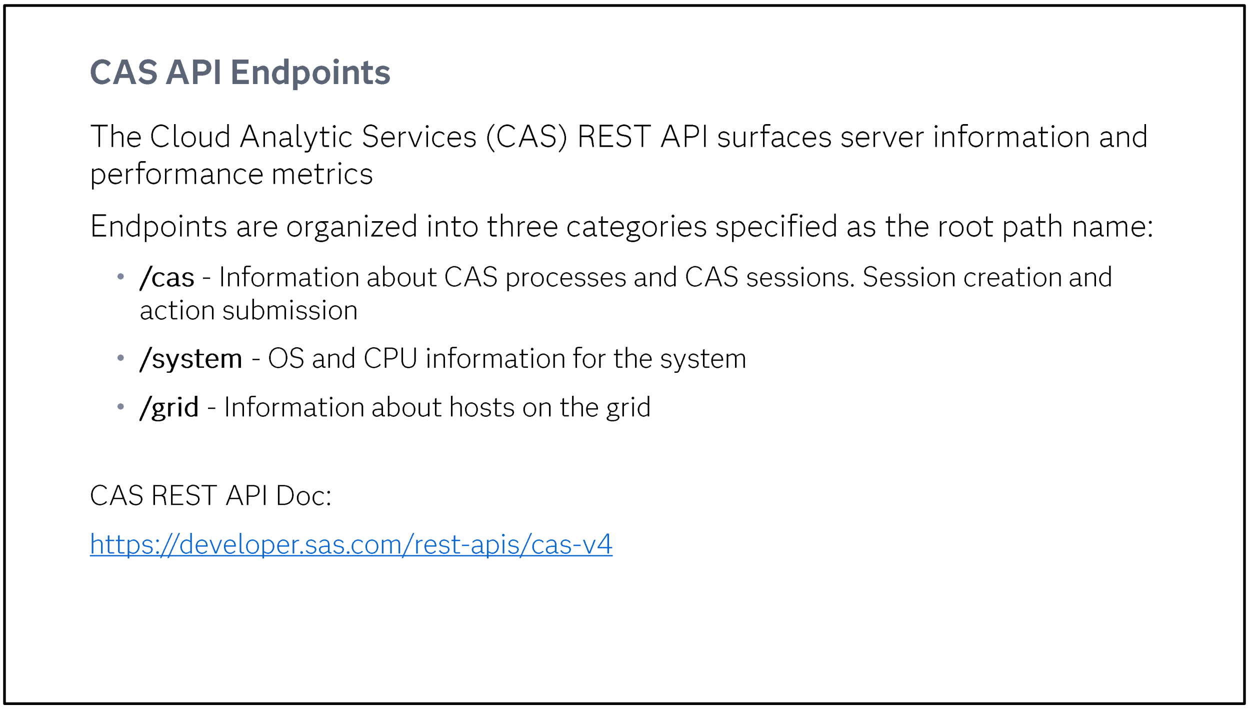
Capabilities

The REST API is part of CAS

The REST API enables applications to

- Get information about CAS processes and sessions and start new sessions
- Get information about CASlibs and loaded tables
- Retrieve operating system and CPU information
- Retrieve information about hosts on the grid
- Run CAS actions

When you think about the capabilities of the CAS REST API, it is an integral part of CAS.
The REST API enables applications to perform various tasks, such as getting information about CAS processes and sessions and starting new sessions.
It also allows you to retrieve information about CASlibs and loaded tables, obtain details about the operating system and CPU, and gather information about hosts on the grid.
And of course, one of its core functionalities is the ability to run CAS actions.



CAS API Endpoints

The Cloud Analytic Services (CAS) REST API surfaces server information and performance metrics

Endpoints are organized into three categories specified as the root path name:

- /cas - Information about CAS processes and CAS sessions. Session creation and action submission
- /system - OS and CPU information for the system
- /grid - Information about hosts on the grid

CAS REST API Doc:

https://developer.sas.com/rest-apis/cas-v4

The Cloud Analytic Services, or CAS, REST API surfaces server information and performance metrics.

Endpoints in the CAS REST API are organized into three main categories based on the root path name.
First, we have /cas, which provides information about CAS processes and sessions, including session creation and action submission.
Next is /system, which offers details about the operating system and CPU.
And finally, /grid, which gives information about the hosts on the CAS grid.

For more details, you can refer to the CAS REST API documentation available at the link shown here.

Requirements for the REST API

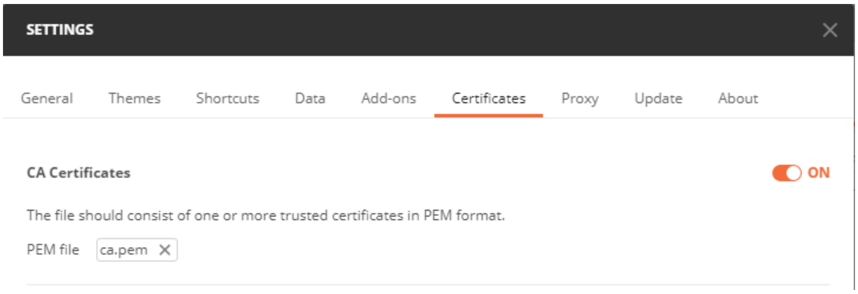
One fast method to test REST API calls is using a tool called Postman

To communicate with encrypted CAS you will need the certificate

The certificate can be retrieved using a command similar to the following

-
  ```
  kubectl -n ${NS} get secret sas-viya-ca-certificate-secret -
  o=jsonpath="{.data.ca\.crt}"|base64 -d > ./ca.pem
  ```

Add the ca.pem certificate to Postman certificates



CA Certificates ON

The file should consist of one or more trusted certificates in PEM format.

PEM file ca.pem

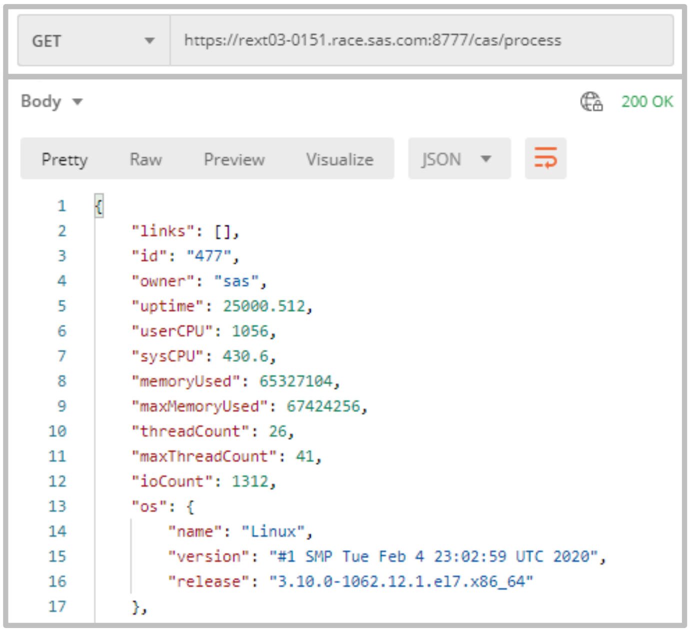
Let’s take a closer look at the requirements for using the REST API.
One fast and effective method to test REST API calls is by using a tool like Postman.
To communicate with an encrypted CAS server, you will need the server's certificate.
The certificate can be retrieved using a command similar to the one shown here.
Once retrieved, the ca.pem certificate needs to be added to Postman’s certificate settings.

This screenshot shows part of the Postman configuration process, where the certificate has been downloaded and added.
With the certificate in place, you’re now ready to connect securely to the CAS server and begin making REST API calls.

REST API EXAMPLE

This simple example returns information about the CAS servers in the environment in JSON format

- Method=GET
- Endpoint=/cas/process
- Requires authentication and encryption
- Use Basic Auth and supply credentials



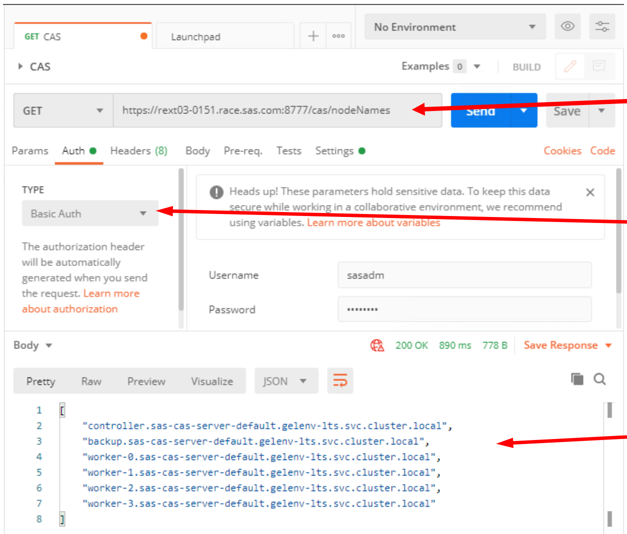
```
{
    "links": [],
    "id": "477",
    "owner": "sas",
    "uptime": 25000.512,
    "userCPU": 1056,
    "sysCPU": 430.6,
    "memoryUsed": 65327104,
    "maxMemoryUsed": 67424256,
    "threadCount": 26,
    "maxThreadCount": 41,
    "ioCount": 1312,
    "os": {
        "name": "Linux",
        "version": "#1 SMP Tue Feb 4 23:02:59 UTC 2020",
        "release": "3.10.0-1062.12.1.el7.x86_64"
    },
```

In this simple example, we’re retrieving information about the CAS servers in the environment in JSON format.
We’re using the GET method and calling the /cas/process endpoint.

Authentication is required, and in this case, we’re using Basic Auth to supply the necessary credentials.

On the right, you can see a screenshot of the call being made and the corresponding body response that we get back from it.

Example: Display Node Names



GET https://rext03-0151.race.sas.com:8777/cas/nodeNames

TYPE

Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Username: sasadm

Password: ••••••••

200 OK 890 ms 778 B

```
[
    "controller.sas-cas-server-default.gelenv-lts.svc.cluster.local",
    "backup.sas-cas-server-default.gelenv-lts.svc.cluster.local",
    "worker-0.sas-cas-server-default.gelenv-lts.svc.cluster.local",
    "worker-1.sas-cas-server-default.gelenv-lts.svc.cluster.local",
    "worker-2.sas-cas-server-default.gelenv-lts.svc.cluster.local",
    "worker-3.sas-cas-server-default.gelenv-lts.svc.cluster.local"
]
```

Method=GET
Endpoint=/cas/nodeNames

Using basic authentication. The userid and passwords is passed in the header of the request.

Prints a list of nodenames as seen by Kubernetes

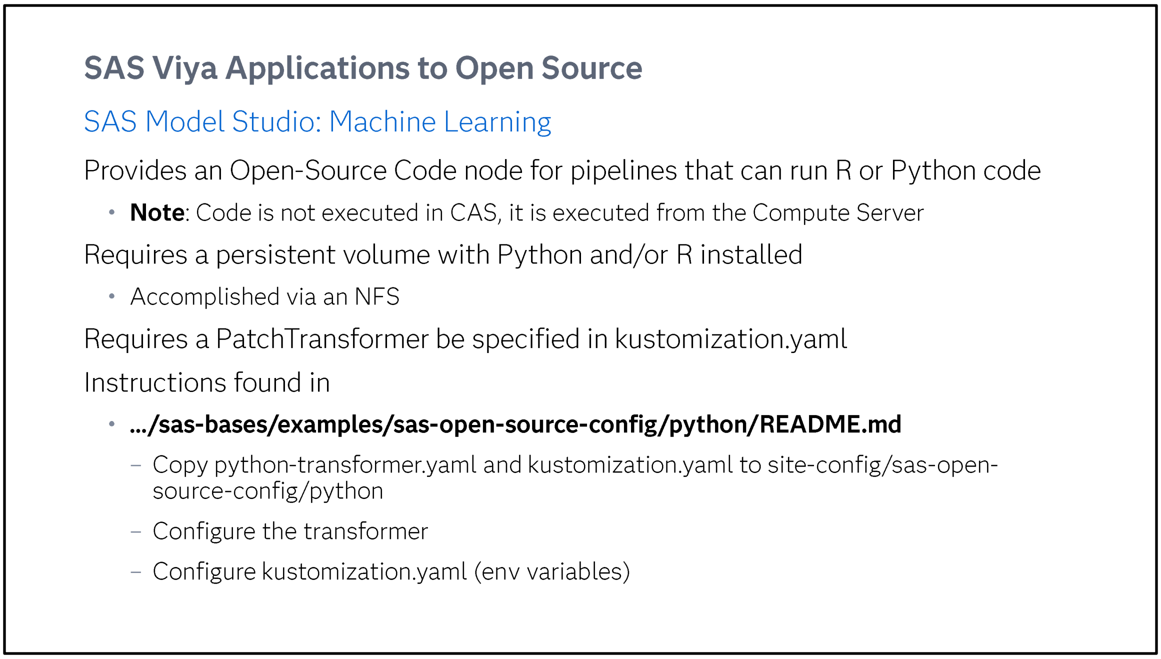
In this example, we're displaying the node names.

As you can see, we're using the GET method, but this time, we're calling the CAS node names endpoint.

We're again using Basic Authentication, where the user ID and password are specified in the header request.
In this case, the user is sasadm with a corresponding password.

The results show the node names from the CAS grid, including the controller, backup controller, and the four worker names, as seen in the output.
This example uses Postman to call the REST APIs.

SAS Viya Applications to Open Source - Details



SAS Viya Applications to Open Source

SAS Model Studio: Machine Learning

Provides an Open-Source Code node for pipelines that can run R or Python code

- **Note**: Code is not executed in CAS, it is executed from the Compute Server

Requires a persistent volume with Python and/or R installed

- Accomplished via an NFS

Requires a PatchTransformer be specified in kustomization.yaml

Instructions found in

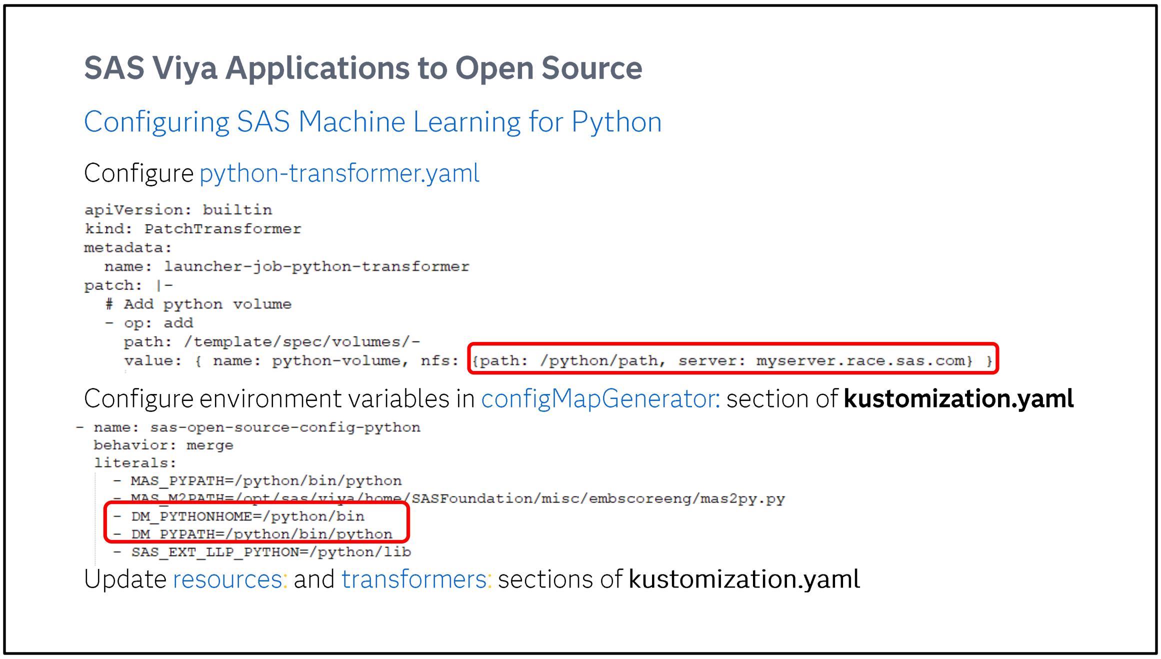
- **…/sas-bases/examples/sas-open-source-config/python/README.md**
  - Copy python-transformer.yaml and kustomization.yaml to site-config/sas-open-source-config/python
  - Configure the transformer
  - Configure kustomization.yaml (env variables)

We'll start by looking at SAS Model Studio for machine learning.
SAS Model Studio provides an open-source code node for pipelines that can run R or Python code.
Note that the code is not executed on the CAS server. It is executed from the compute server.

It requires a persistent volume with the Python and/or R binaries installed.
This could be accomplished using an NFS share.

Additionally, it requires a patch transformer to be specified in the kustomization.yaml.

You can find instructions for this under the SAS Bases examples, specifically in the SAS open-source configuration for Python readme.
You will need to copy the python-transformer.yaml and kustomization.yaml files to the site-config/sas-open-source-config/python directory.
After that, configure the transformer and the kustomization.yaml with the appropriate environment variables.



SAS Viya Applications to Open Source

Configuring SAS Machine Learning for Python

Configure python-transformer.yaml

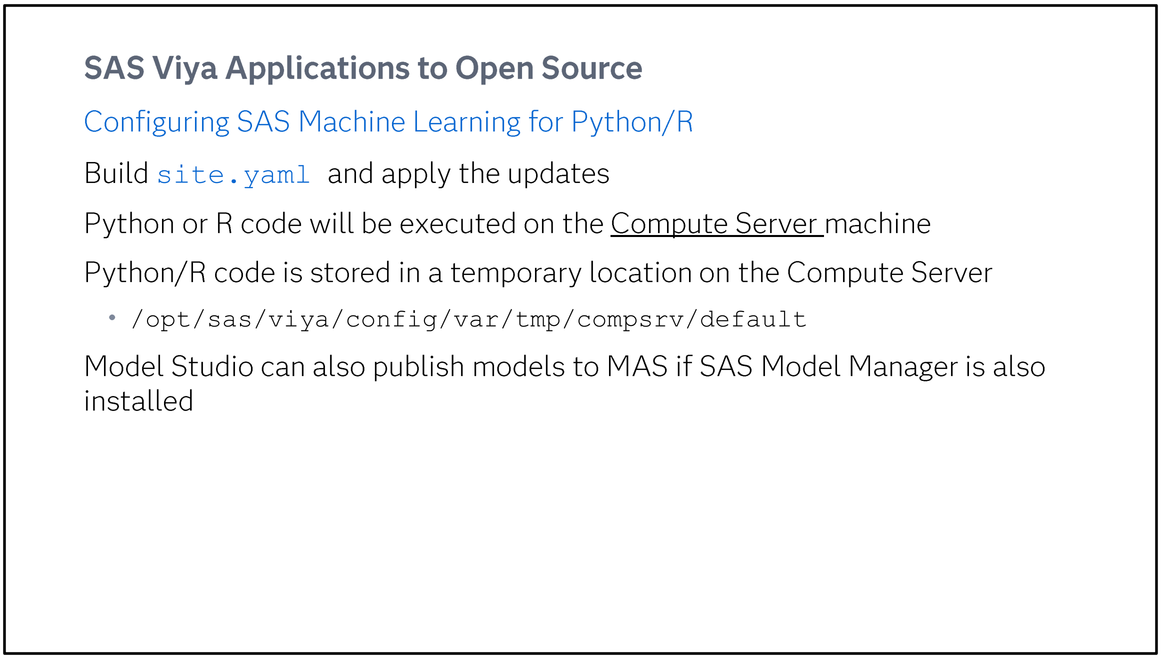
```
apiVersion: builtin
kind: PatchTransformer
metadata:
  name: launcher-job-python-transformer
patch: |-
  # Add python volume
  - op: add
    path: /template/spec/volumes/-
    value: { name: python-volume, nfs: {path: /python/path, server: myserver.race.sas.com} }
```

Configure environment variables in configMapGenerator: section of **kustomization.yaml**

```
- name: sas-open-source-config-python
  behavior: merge
  literals:
    - MAS_PYPATH=/python/bin/python
    - MAS_M2PATH=/opt/sas/viya/home/SASFoundation/misc/embscoreeng/mas2py.py
    - DM_PYTHONHOME=/python/bin
    - DM_PYPATH=/python/bin/python
    - SAS_EXT_LLP_PYTHON=/python/lib
```

Update resources: and transformers: sections of kustomization.yaml

So if we look at this, the python-transformer.yaml is where we specify the information for the NFS share. Here, we're specifying the Python volume, indicating that it’s an NFS share, the path of the share, and the server where it is mounted.
Next, we need to configure the environment variables.
This is done using a configMapGenerator in the kustomization.yaml.
In this example, you can see that we are specifying things like the Python home and the path to the binaries.
Finally, we need to update the resources and transformers sections of the kustomization.yaml.



SAS Viya Applications to Open Source

Configuring SAS Machine Learning for Python/R

Build `site.yaml` and apply the updates

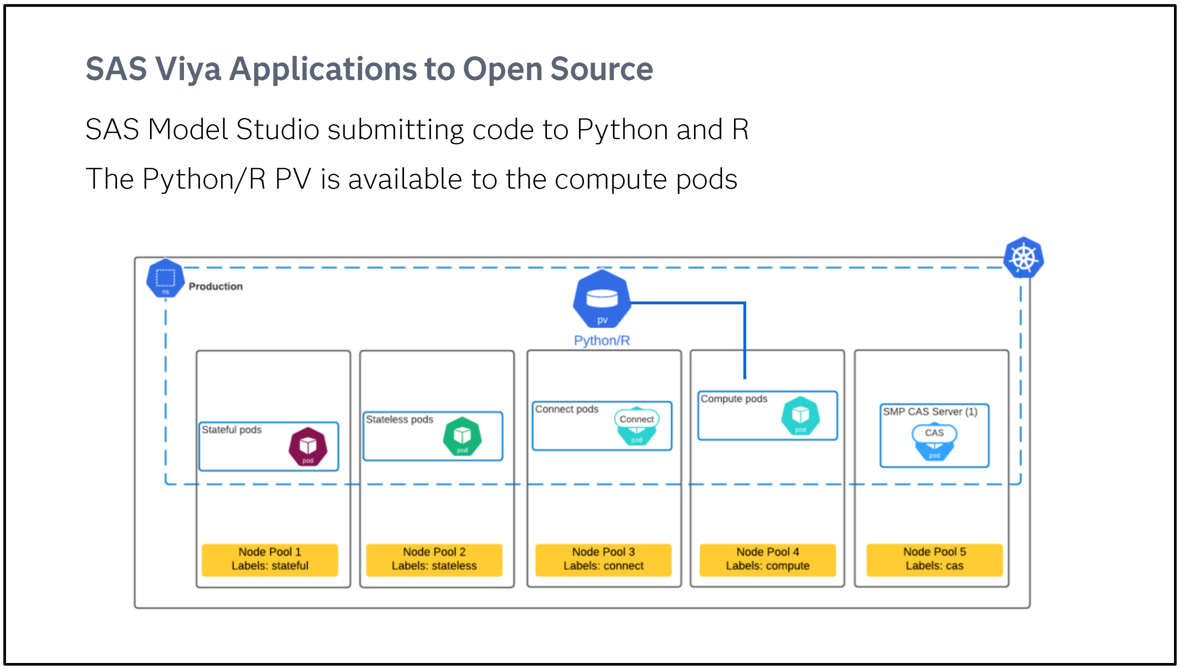
Python or R code will be executed on the Compute Server machine

Python/R code is stored in a temporary location on the Compute Server

- `/opt/sas/viya/config/var/tmp/compsrv/default`

Model Studio can also publish models to MAS if SAS Model Manager is also installed

So, after configuring SAS Machine Learning for Python and R, you need to re-build and re-apply the updated site.yaml.
The Python or R code will be executed on the Compute Server machine.
This code is stored in a temporary location on the Compute Server under the path shown here.
Additionally, Model Studio can also publish models to MAS if SAS Model Manager is installed as part of the order.



SAS Viya Applications to Open Source

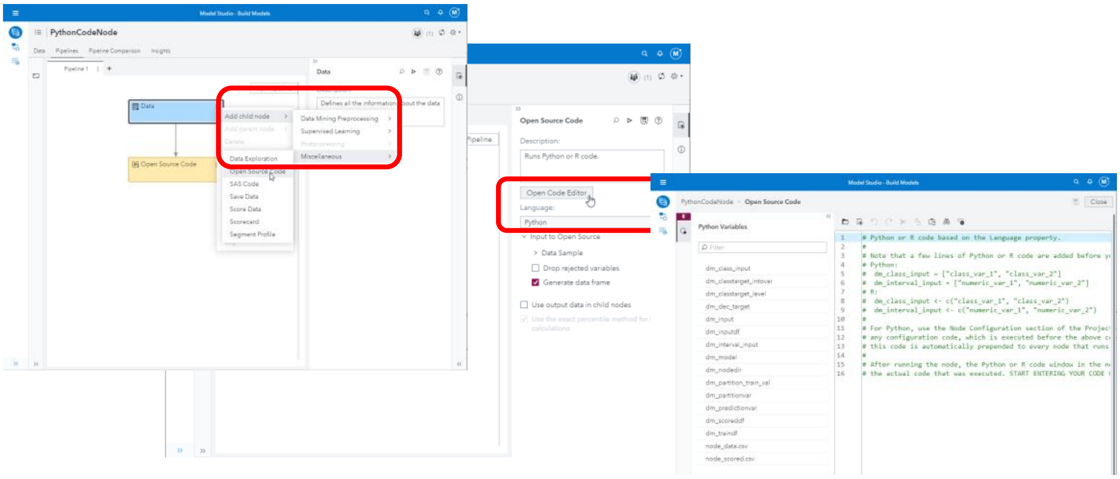
SAS Model Studio submitting code to Python and R

The Python/R PV is available to the compute pods

If we look at this visually, you can see SAS Model Studio submitting Python and R code.
We can observe that the Model Studio pods need access to the Python and R binaries in this setup.
Just for completeness, it should be noted that while I'm showing the connect node pool here, that's an optional configuration.
The key takeaway is that the persistent volume is available to our applications running on the stateless pods.

SAS Viya Applications to Open Source

SAS Model Studio (steps to add an Open-Source Code node)



Data

Description: Defines all the information about the data

Open Source Code

Description: Runs Python or R code.

Language: Python

Input to Open Source

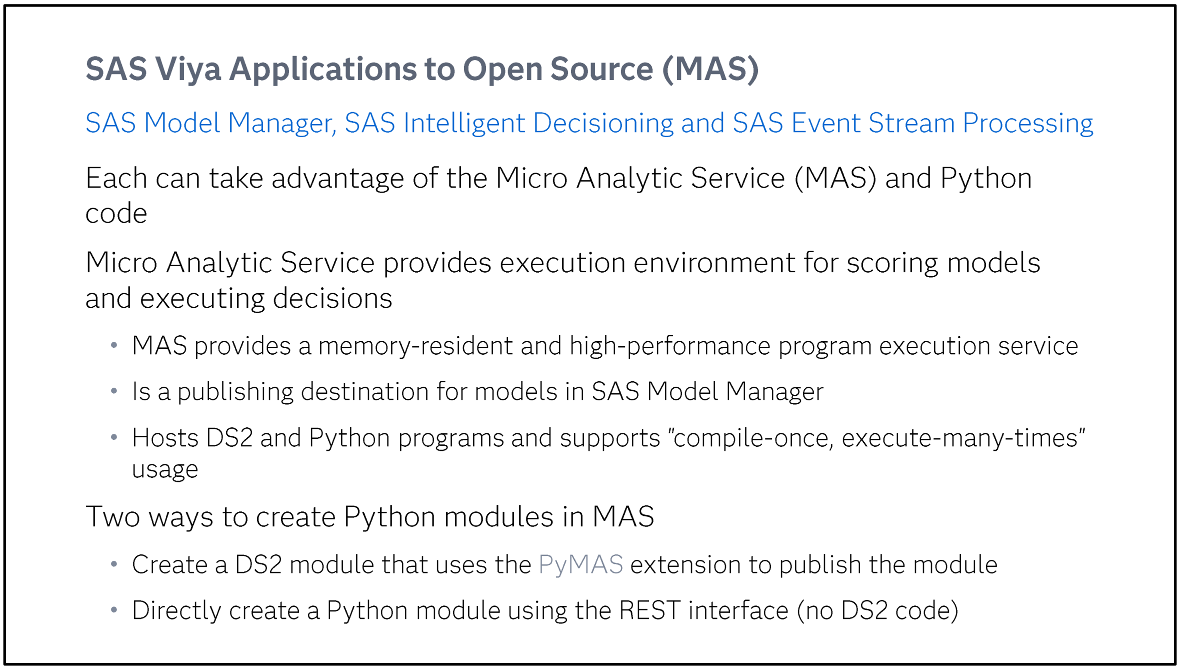
- Data Sample
- ☐ Drop rejected variables
- ☑ Generate data frame

☐ Use output data in child nodes

☑ Use the exact percentile method for calculations

```
# Python or R code based on the Language property.
#
# Note that a few lines of Python or R code are added before y
# Python:
#  dm_class_input = ["class_var_1", "class_var_2"]
#  dm_interval_input = ["numeric_var_1", "numeric_var_2"]
# R:
#  dm_class_input <- c("class_var_1", "class_var_2")
#  dm_interval_input <- c("numeric_var_1", "numeric_var_2")
#
# For Python, use the Node Configuration section of the Projec
# any configuration code, which is executed before the above c
# this code is automatically prepended to every node that runs
#
# After running the node, the Python or R code window in the n
# the actual code that was executed. START ENTERING YOUR CODE
```

When you're in SAS Model Studio, the steps to add the Open-Source Code node are as follows.
First, you go into the interface, and you can see where to specify adding the open-source code node.
Then, you select the code editor.
Finally, on the right side here, you can see where you specify the Python or R code that you're going to run.



SAS Viya Applications to Open Source (MAS)

SAS Model Manager, SAS Intelligent Decisioning and SAS Event Stream Processing

Each can take advantage of the Micro Analytic Service (MAS) and Python code

Micro Analytic Service provides execution environment for scoring models and executing decisions

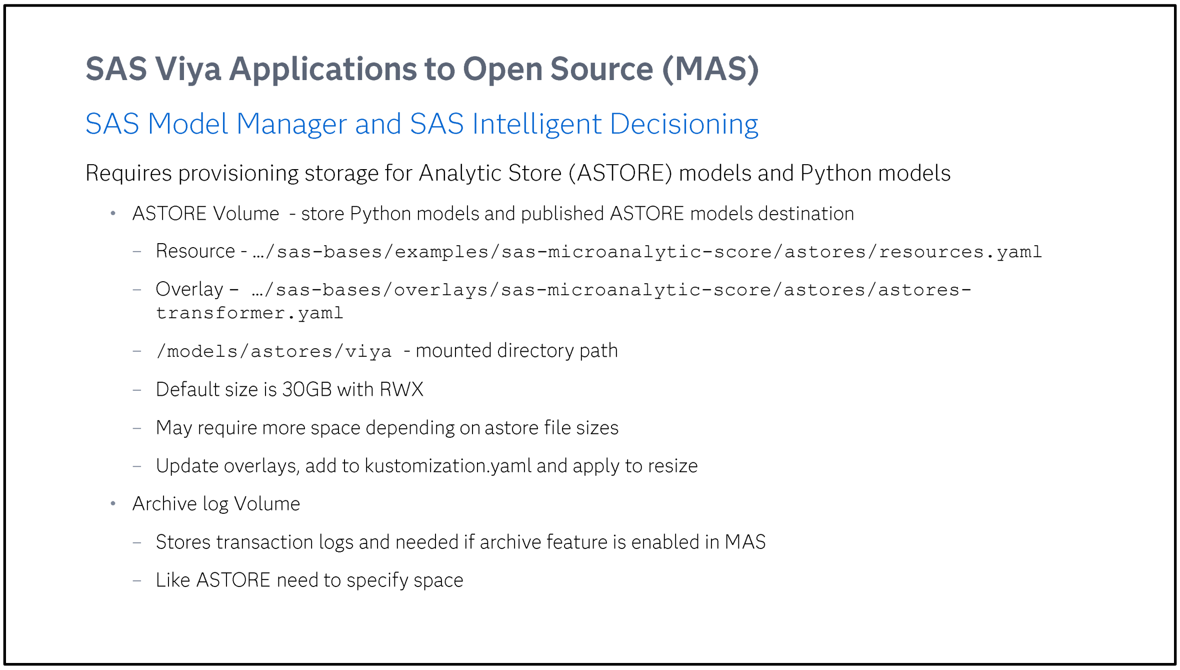
- MAS provides a memory-resident and high-performance program execution service
- Is a publishing destination for models in SAS Model Manager
- Hosts DS2 and Python programs and supports "compile-once, execute-many-times" usage

Two ways to create Python modules in MAS

- Create a DS2 module that uses the PyMAS extension to publish the module
- Directly create a Python module using the REST interface (no DS2 code)

Now, we will look at SAS Model Manager, SAS Intelligent Decisioning, and SAS Event Stream Processing.
Each of these can take advantage of the Micro Analytic Service (MAS) and Python code.
The Micro Analytic Service (MAS) provides an execution environment for scoring models and executing decisions.
MAS itself offers an in-memory resident, high-performance program execution service.
It is also a publishing destination for models in SAS Model Manager.
MAS hosts DS2 and Python programs, supporting the "compile-once, execute-many-times" usage.

There are two ways to create Python modules in MAS.
You can create a DS2 module that specifies the PyMAS extension to publish the module.
Alternatively, you can directly create a Python module using the REST interface, in which case, you don't need any DS2 code.



SAS Viya Applications to Open Source (MAS)

SAS Model Manager and SAS Intelligent Decisioning

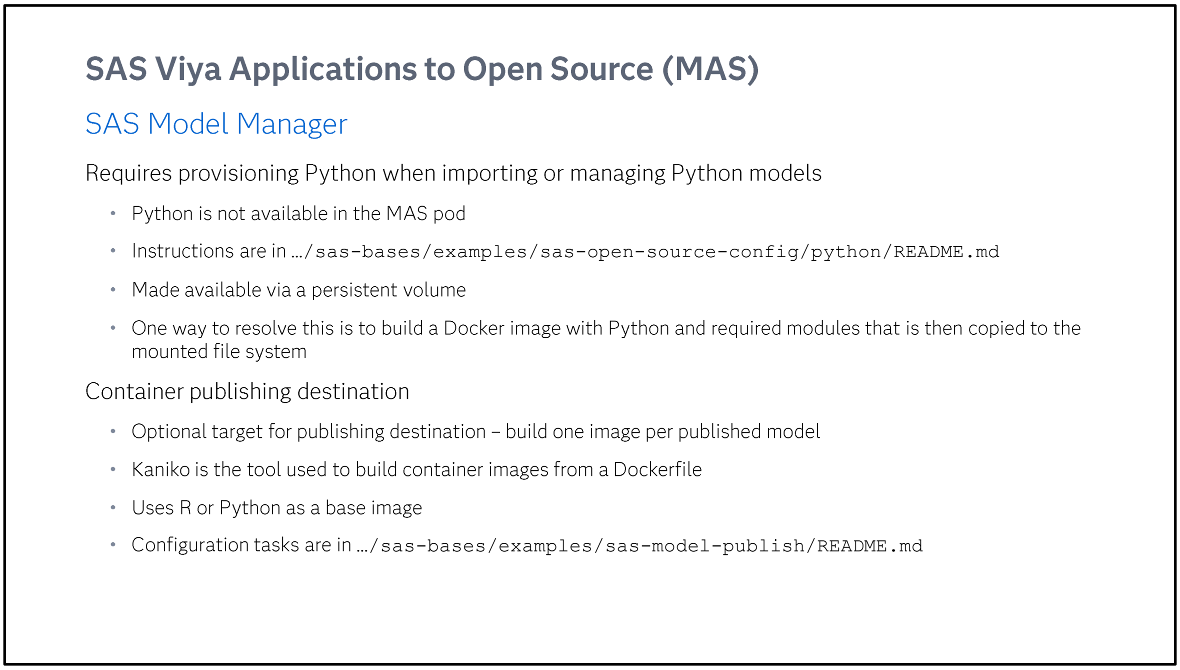
Requires provisioning storage for Analytic Store (ASTORE) models and Python models

- ASTORE Volume - store Python models and published ASTORE models destination
  - Resource - …/`sas-bases/examples/sas-microanalytic-score/astores/resources.yaml`
  - Overlay - …/`sas-bases/overlays/sas-microanalytic-score/astores/astores-transformer.yaml`
  - `/models/astores/viya` - mounted directory path
  - Default size is 30GB with RWX
  - May require more space depending on astore file sizes
  - Update overlays, add to kustomization.yaml and apply to resize
- Archive log Volume
  - Stores transaction logs and needed if archive feature is enabled in MAS
  - Like ASTORE need to specify space

You need to provision some storage for the Analytic Store (ASTORE) files and models, as well as the Python models.

Configuring the ASTORE storage is necessary to store those Python models and to publish the ASTORE model destinations.
Resources for this are available under the SAS Bases examples, specifically in the SAS Micro-analytic Score directory, in the astores/resources.yaml file.
You also need to include the overlay found in the SAS Bases overlays directory under sas-microanalytic-score/astores/astores-transformer.yaml.
As part of that configuration, you'll specify the mount point for the ASTORE storage.
Additionally, determine the amount of storage you want to use.
The default storage needs to be RWX storage, allowing access to multiple pods.
You may require more space depending on the number of ASTORE models you are publishing.
You also need to add the overlays to the kustomization.yaml, then apply and configure them for your environment.

Additionally, there is storage required for the archive log volume.
This volume stores transaction logs and is needed if the archive feature is enabled in MAS.
Just like the ASTORE storage, you need to specify the path to the storage and the amount of space required.



SAS Viya Applications to Open Source (MAS)

SAS Model Manager

Requires provisioning Python when importing or managing Python models

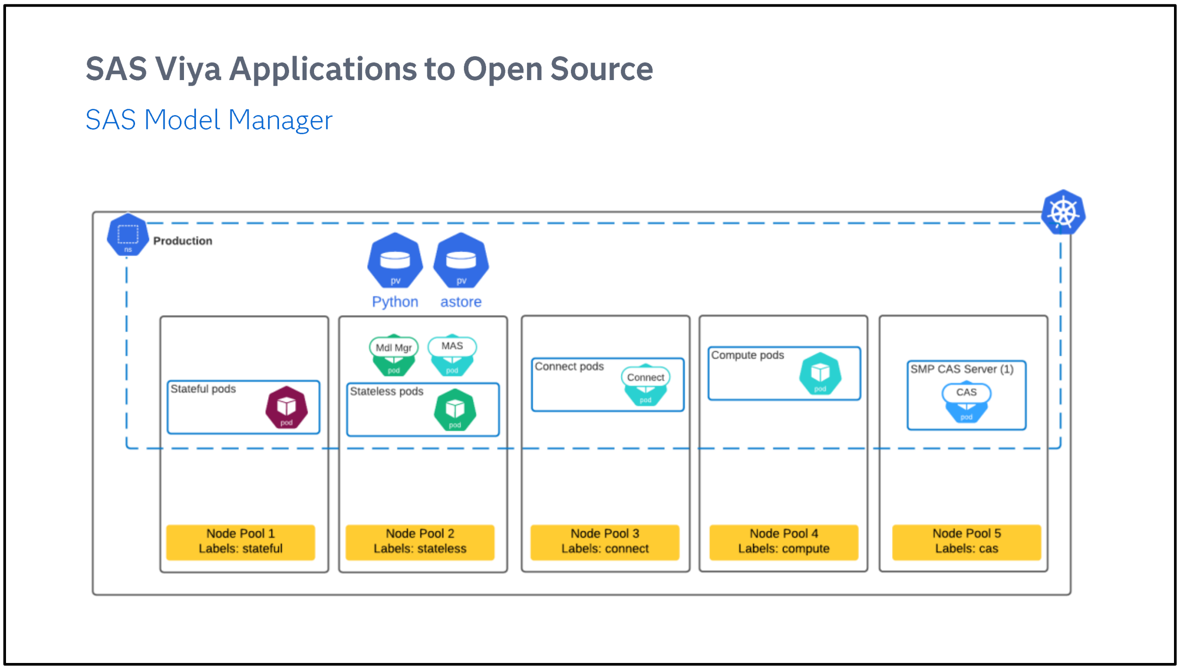
- Python is not available in the MAS pod
- Instructions are in …/`sas-bases/examples/sas-open-source-config/python/README.md`
- Made available via a persistent volume
- One way to resolve this is to build a Docker image with Python and required modules that is then copied to the mounted file system

Container publishing destination

- Optional target for publishing destination – build one image per published model
- Kaniko is the tool used to build container images from a Dockerfile
- Uses R or Python as a base image
- Configuration tasks are in …/`sas-bases/examples/sas-model-publish/README.md`

SAS Model Manager requires provisioning Python when importing or managing Python models.
Python is not available in the MAS pod, so you need to set it up.
Instructions for doing this can be found under the SAS Bases examples, specifically in the sas-open-source-config/python/README.md.
To make Python available, it needs to be provided via a persistent volume.
One way to achieve this is by building a Docker image with Python and the required modules and then copying that image to the mounted file system.

As for container publishing destinations, this is an optional target for publishing, which allows you to build one image per model.
To set this up, you'll use Kaniko, a tool designed to build container images from a Dockerfile.
Kaniko uses R or Python as the base image for building these containers.
Configuration tasks related to this process can be found under the SAS Bases examples, in the sas-model-publish/README.md.



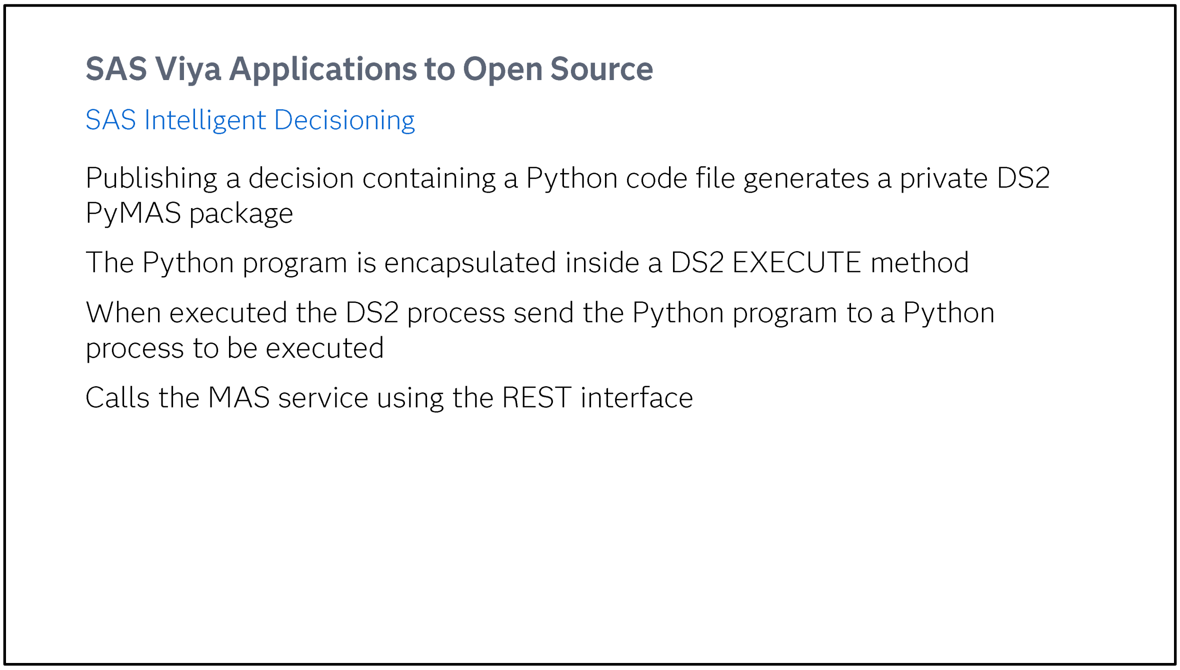
SAS Viya Applications to Open Source

SAS Model Manager

Again, if we look at this visually, we can see, once again, our Kubernetes cluster, and specifically our SAS Viya production namespace.

We need to ensure that the persistent volumes are created for both our Python storage and ASTORE storage.

These volumes need to be available to both the SAS Model Manager and MAS pods.



SAS Viya Applications to Open Source

SAS Intelligent Decisioning

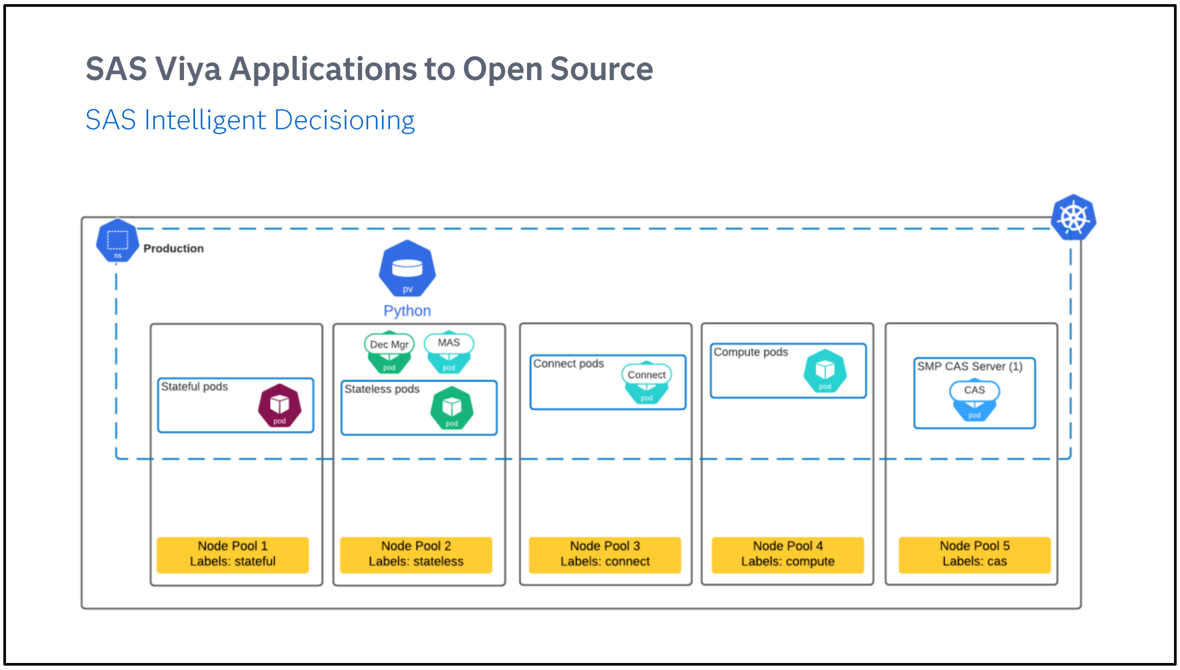
Publishing a decision containing a Python code file generates a private DS2 PyMAS package

The Python program is encapsulated inside a DS2 EXECUTE method

When executed the DS2 process send the Python program to a Python process to be executed

Calls the MAS service using the REST interface

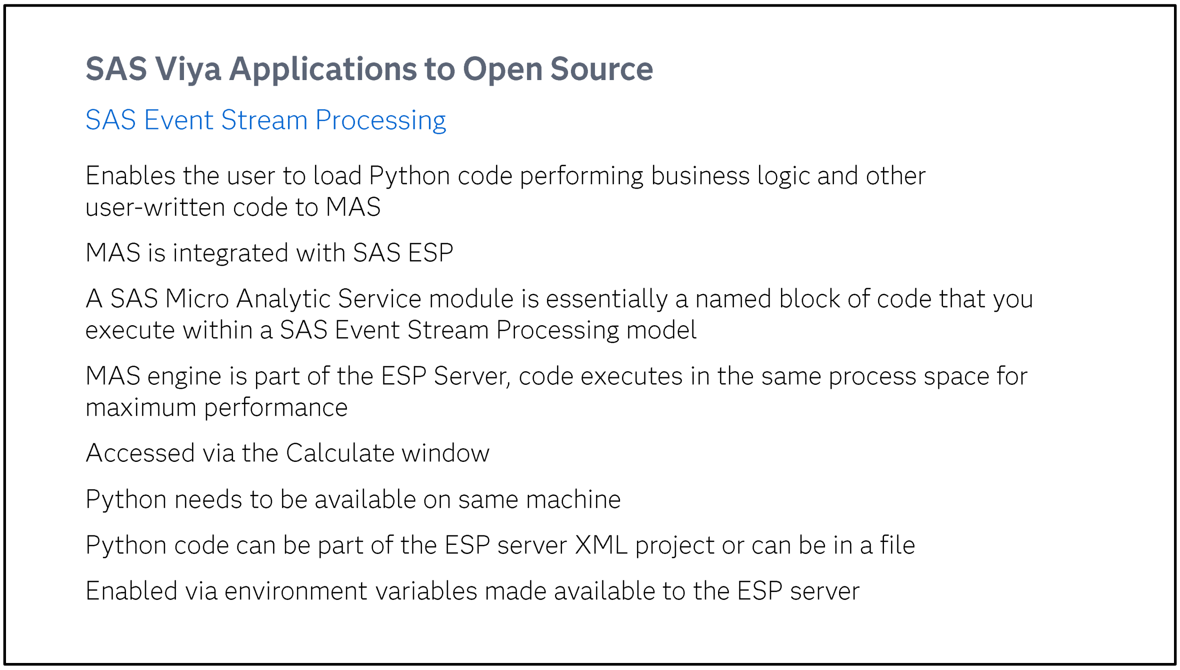
SAS Intelligent Decisioning.
Publishing a decision that contains a Python code file generates a private DS2 PyMAS package.
The Python program is encapsulated inside a DS2 EXECUTE method.
When executed, the DS2 process sends the Python program to a Python process to be executed.
It calls the MAS service using the REST interface.



SAS Viya Applications to Open Source

SAS Intelligent Decisioning

So again, if we're looking at this visually, we can see that we've got our decision manager and our Python pods.
These need to have the Python code available to them via that persistent volume claim definition.



SAS Viya Applications to Open Source

SAS Event Stream Processing

Enables the user to load Python code performing business logic and other user-written code to MAS

MAS is integrated with SAS ESP

A SAS Micro Analytic Service module is essentially a named block of code that you execute within a SAS Event Stream Processing model

MAS engine is part of the ESP Server, code executes in the same process space for maximum performance

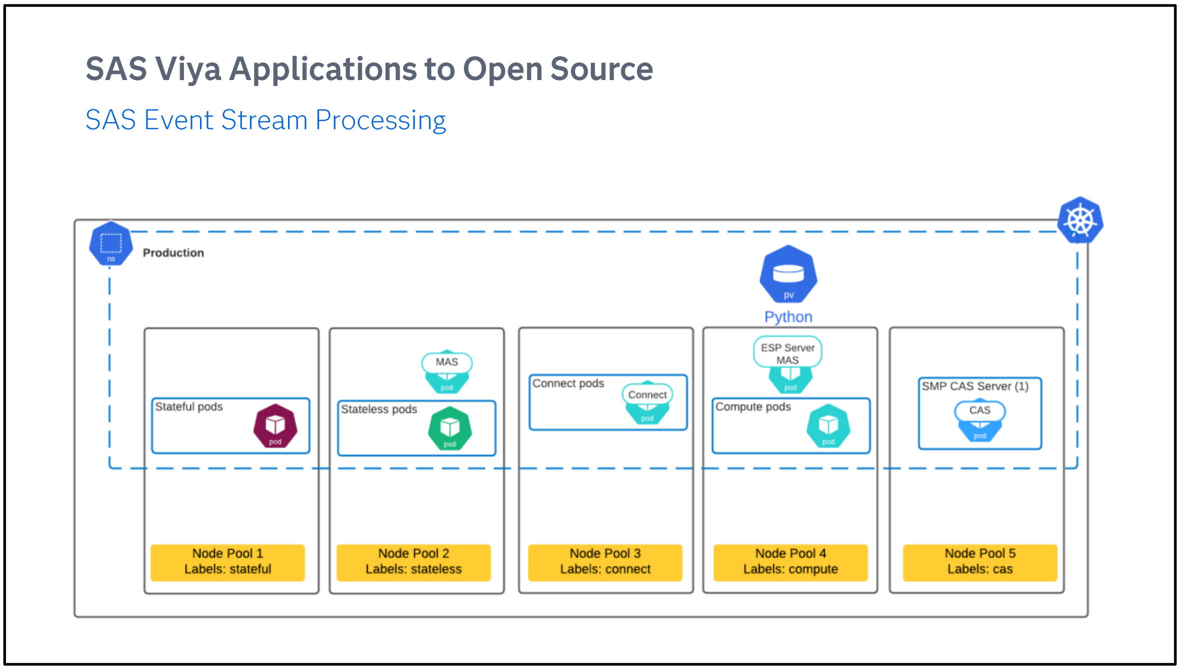
Accessed via the Calculate window

Python needs to be available on same machine

Python code can be part of the ESP server XML project or can be in a file

Enabled via environment variables made available to the ESP server

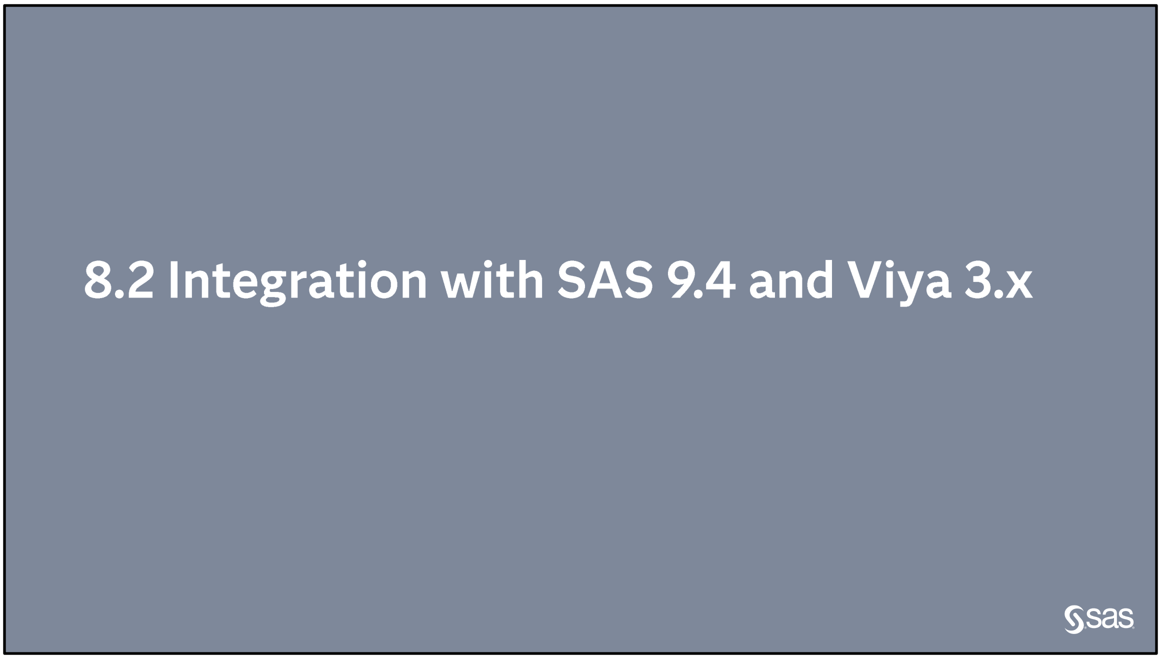
In SAS Event Stream Processing, we enable the user to load Python code performing business logic and other user-written code to MAS.
It is integrated with SAS ESP.
The SAS Micro Analytic Service module is essentially a named block of code that you execute within a SAS Event Stream Processing model.
The MAS engine is part of the ESP Server, and the code executes in the same process space for maximum performance.
It can be accessed via the Calculate window in ESP.
Python needs to be available on the same machine where the pods are running.
The Python code can be part of the ESP server XML project or stored in a file.
Access is enabled via environment variables made available to the ESP server.



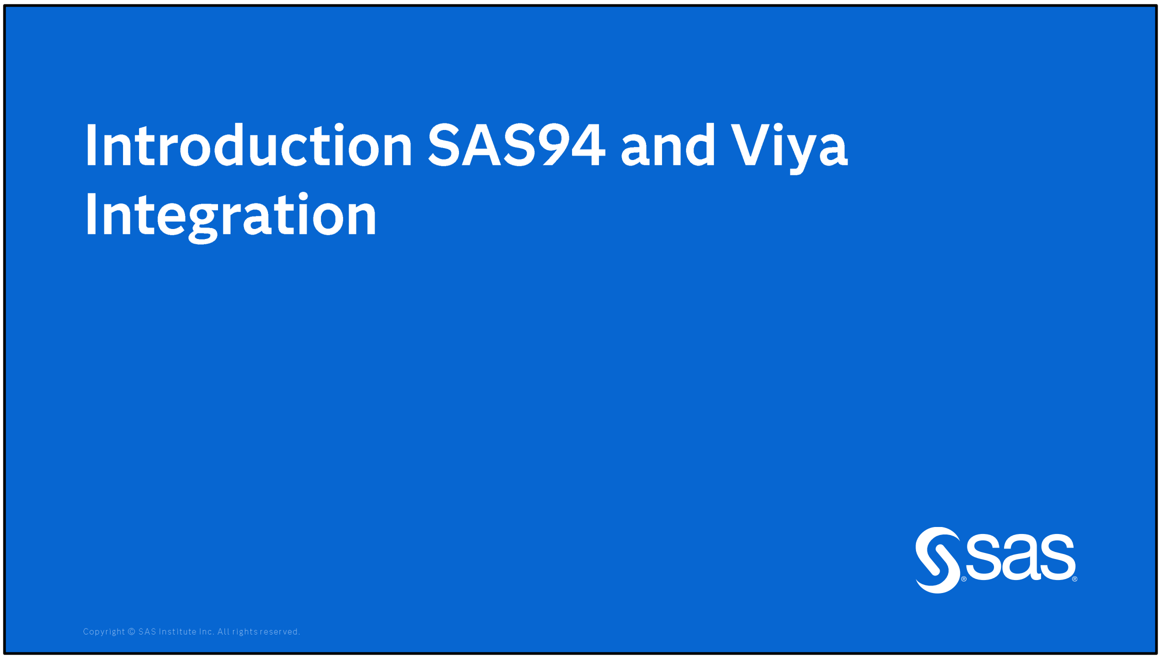
SAS Viya Applications to Open Source

SAS Event Stream Processing

So again, looking at this visually, we've got our ESP server and MAS pods.
And we've got our Python PVC there, making the Python code available to that.



8.2 Integration with SAS 9.4 and Viya 3.x



Introduction SAS94 and Viya Integration

The integration pieces

To be able to communicate, two SAS environments require:

- An open communication channel
- A common language
- A shared security framework

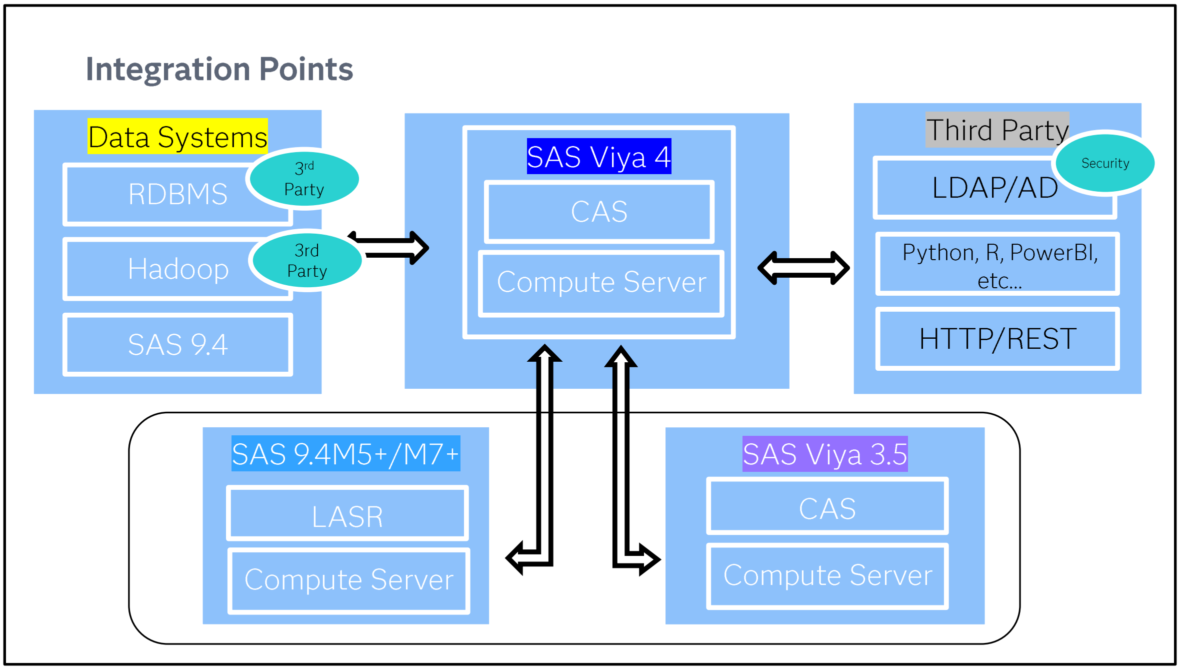
你好
HALLO
안녕
HOLA
नमस्ते
CIAO
HELLO
γεια
こんにちは
привет
BONJOUR
مرحبا
OLÁ
I am nobody
Who are you?

Welcome to this introductory module on the integration of SAS 9.4 and Viya 3.5 with, Viya 4.
In this module we will provide some insights into what is currently possible.
It’s worth starting this module by briefly talking about what constitutes integration in this context.

It’s about having open communication channels that the SAS components from different technologies use a common language on.

And because the communication is in effect taking place between two systems,

security needs to be a factor to allow a users to leverage the two systems seamlessly.



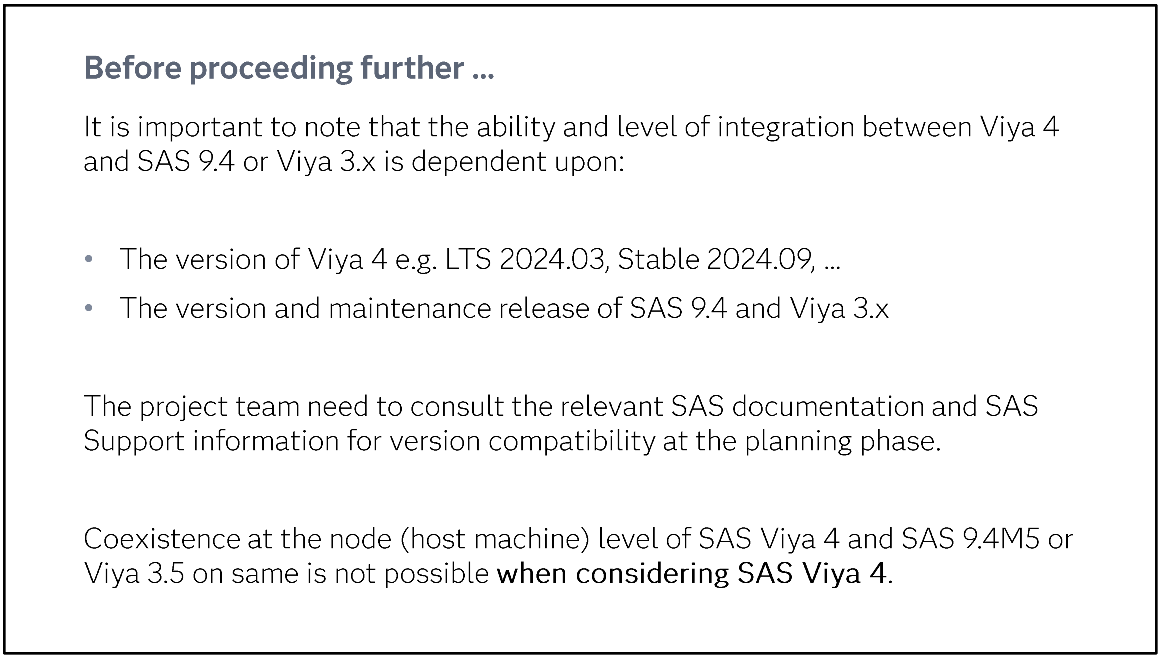
Let’s put Viya 4 in the center of what we do.

For this purpose, we assume that the SAS Viya environment has at least one CAS Server and Compute Server available to users.

As SAS Viya 4 is an enterprise system, we will be leveraging identities and authentication providers to authenticate our users.
We may also be using a third-party technology like Python to be integrated with Viya.

Accessing data will be the next consideration.
There will be drivers required to enable Viya to access the various data systems.

For this module, we are concerned with SAS 9.4 and Viya 3.5 integration with Viya 4. Note that in SAS 9.4 there is the SAS LASR server, which was the predecessor to the CAS Server.



Before proceeding further …

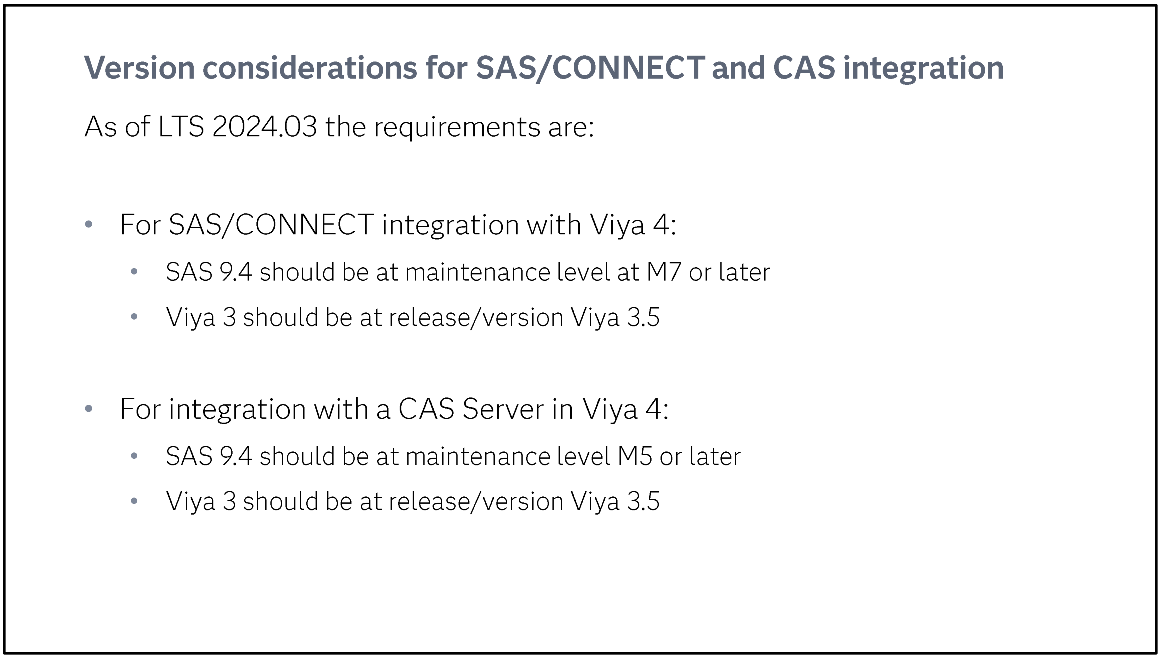
It is important to note that the ability and level of integration between Viya 4 and SAS 9.4 or Viya 3.x is dependent upon:

- The version of Viya 4 e.g. LTS 2024.03, Stable 2024.09, …
- The version and maintenance release of SAS 9.4 and Viya 3.x

The project team need to consult the relevant SAS documentation and SAS Support information for version compatibility at the planning phase.

Coexistence at the node (host machine) level of SAS Viya 4 and SAS 9.4M5 or Viya 3.5 on same is not possible **when considering SAS Viya 4**.

Architecture design is a key part of planning when considering the integration between systems. That applies to considering the integration of SAS 9.4 with SAS Viya. The project team needs to consult the relevant SAS documentation and the latest SAS Support notes for version compatibility when creating the architecture design.
An additional point that is worth mentioning, especially to those unfamiliar with Kubernetes. For host machines running Kubernetes clusters, it is recommended or even unfeasible in some cases to have software running on the same host machine outside of the Kubernetes cluster. So, installing SAS 9.4 on the same host machine as SAS Viya 4 is not an option.



Version considerations for SAS/CONNECT and CAS integration

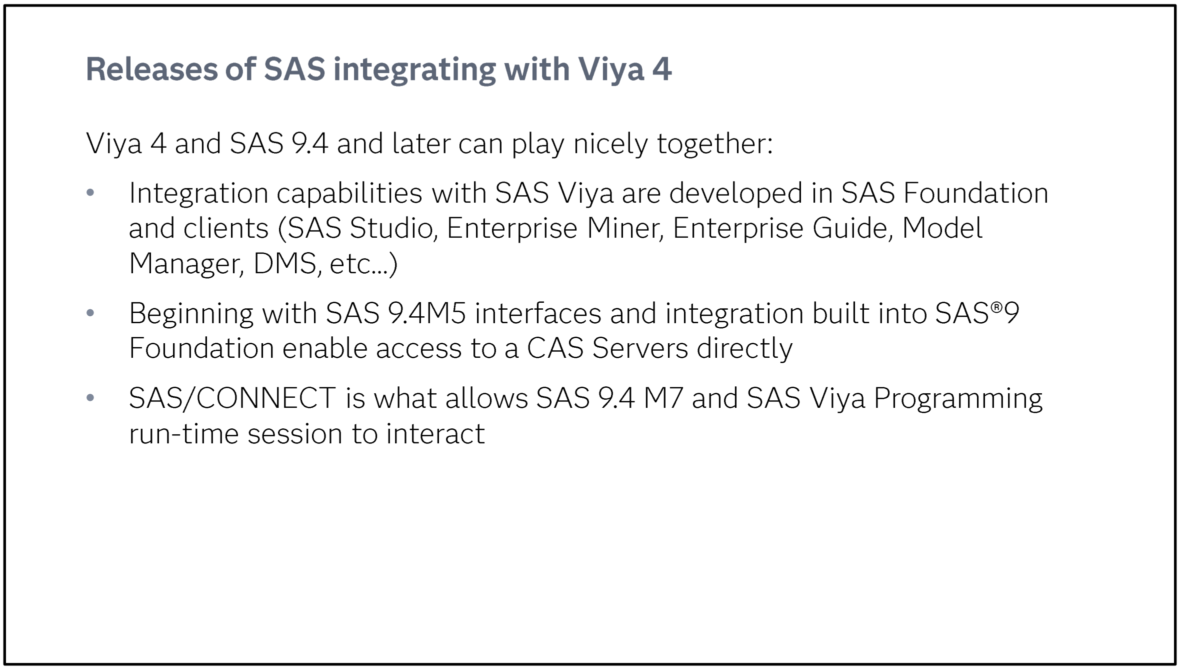
As of LTS 2024.03 the requirements are:

- For SAS/CONNECT integration with Viya 4:
  - SAS 9.4 should be at maintenance level at M7 or later
  - Viya 3 should be at release/version Viya 3.5
- For integration with a CAS Server in Viya 4:
  - SAS 9.4 should be at maintenance level M5 or later
  - Viya 3 should be at release/version Viya 3.5

When consuming the remaining content within this topic, it is important to note the information provided here. References to SAS 9.4 or Viya 3.x integration with Viya 4 will be making implicit references to these requirements.

For using SAS/Connect to allow SAS 9.4 users to connect to Viya 4, the version of SAS 9.4 will be at maintenance level M7 or later. For Viya 3, the release will be Viya 3.5.

Where the integration for SAS 9.4 users is only focused on a CAS server within Viya, then the maintenance level is M5 or later. For Viya 3, the release will be Viya 3.5.
Should a 9.4 user community require both SAS/Connect and CAS integration with Viya 4, then the version 9.4 at maintenance level M7 or later will be required.

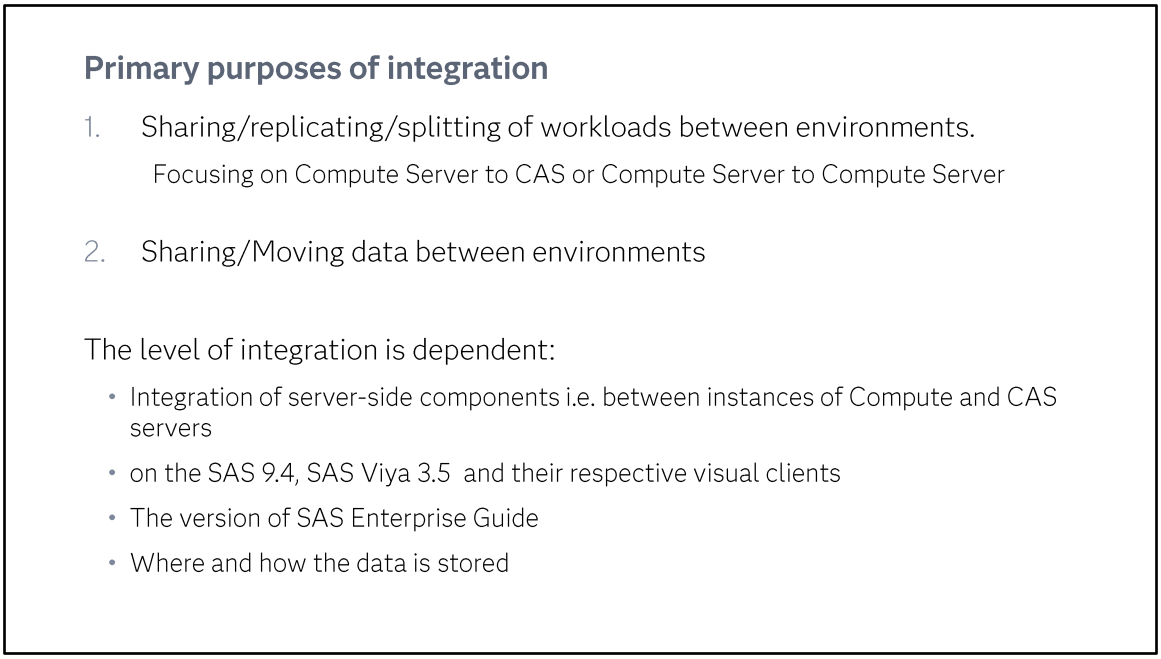


Releases of SAS integrating with Viya 4

Viya 4 and SAS 9.4 and later can play nicely together:

- Integration capabilities with SAS Viya are developed in SAS Foundation and clients (SAS Studio, Enterprise Miner, Enterprise Guide, Model Manager, DMS, etc…)
- Beginning with SAS 9.4M5 interfaces and integration built into SAS®9 Foundation enable access to a CAS Servers directly
- SAS/CONNECT is what allows SAS 9.4 M7 and SAS Viya Programming run-time session to interact

To enhance the user experience when integrating SAS 9.4 and Viya 3 with Viya 4, it's essential to consider the client applications involved. The clients mentioned here are commonly found in organizations that have licensed SAS 9.4. While this is not an exhaustive list, it highlights the most frequently used client applications. Organizations using SAS Solutions may have additional client applications with similar integration capabilities. As previously noted, the maintenance levels of SAS 9.4 determine its compatibility for integration with CAS Server and SAS/Connect.



Primary purposes of integration

1. Sharing/replicating/splitting of workloads between environments.

   Focusing on Compute Server to CAS or Compute Server to Compute Server

2. Sharing/Moving data between environments

The level of integration is dependent:

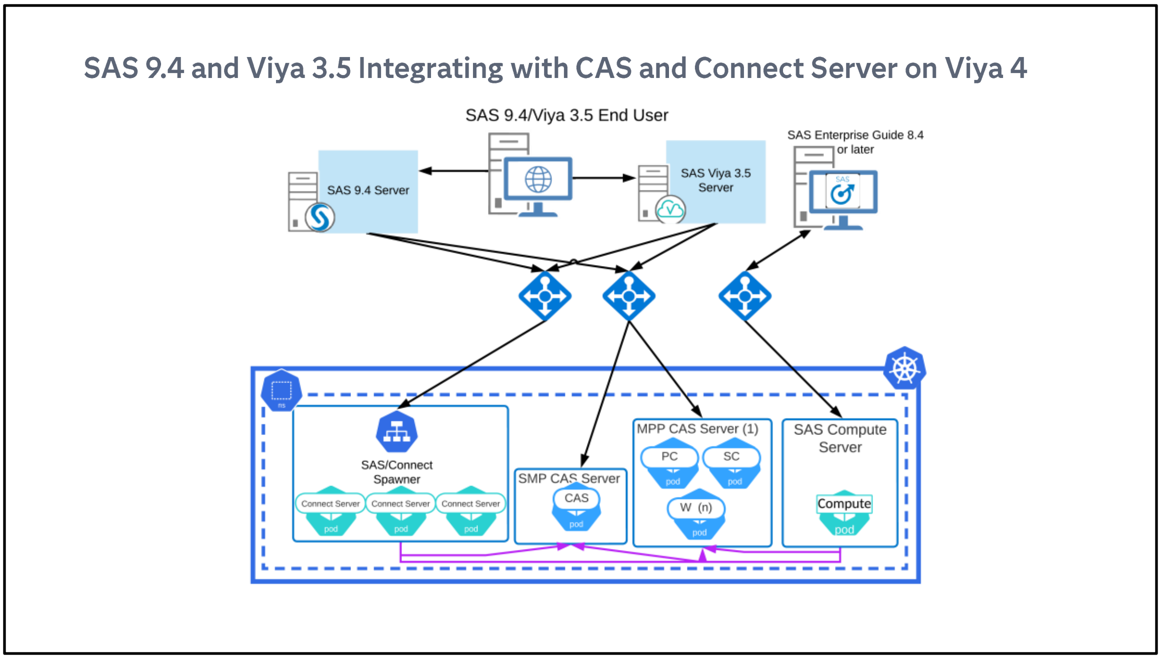
- Integration of server-side components i.e. between instances of Compute and CAS servers
- on the SAS 9.4, SAS Viya 3.5 and their respective visual clients
- The version of SAS Enterprise Guide
- Where and how the data is stored

Let us think about the 2 most common reasons for integrating Viya 4 with either Viya 3.5 or SAS 9.4.

The first reason is the sharing, replicating, or splitting of workloads between Viya 4 and the previous versions. Now you may be asking in what scenarios this may happen? Well, a couple of scenarios spring to mind. In one scenario, an organization with SAS 9.4 may be in the early days of trialling SAS Viya and may wish to compare the performance aspects of SAS compute job executions between SAS 9.4 and SAS Viya. In another scenario an organization may wish to begin the process of modifying existing 9.4 programs to leverage the potential performance improvements of using a CAS Server in Viya. And for some organizations a third scenario may exist where the modernization from SAS 9.4 to SAS Viya will take place in an iterative manner or multiple phases. So, in the first phase, maybe 30% of workloads are moved over to SAS Viya, with the remaining 70% of workloads moved in successive phases. Across most of the use cases within the scenarios, the users would typically log on to their SAS 9.4 or Viya 3.5 environment and initiate a workflow within a client that would include a connection to Viya 4 to generated workloads on Viya 4.
There are 3 options for sharing workloads with the current version SAS Viya such as LTS 2024.03. The first option is for the compute servers on SAS 9.4 or Viya 3.5 to request CAS processing in Viya 4. The second option is for compute servers on SAS 9.4 or Viya 3.5 to request Compute Server processing in Viya 4. The third option is really a combination of the first 2 options, where workloads are sent to Viya to leverage a combination of Compute Server as well as CAS processing.

The second reason is integration around data, and how data can be shared between multiple versions of SAS. While sharing of data between SAS Viya and other releases of SAS technology is possible, organizations considering this should consider factors like change management for the data and authorization and permissions to access the data.

The level of integration whether it’s for sharing workloads or sharing data is dependent on multiple factors. For example, do all the versions of SAS have access to the source data? What specific versions of SAS 9.4 are being used and what client applications are being used.



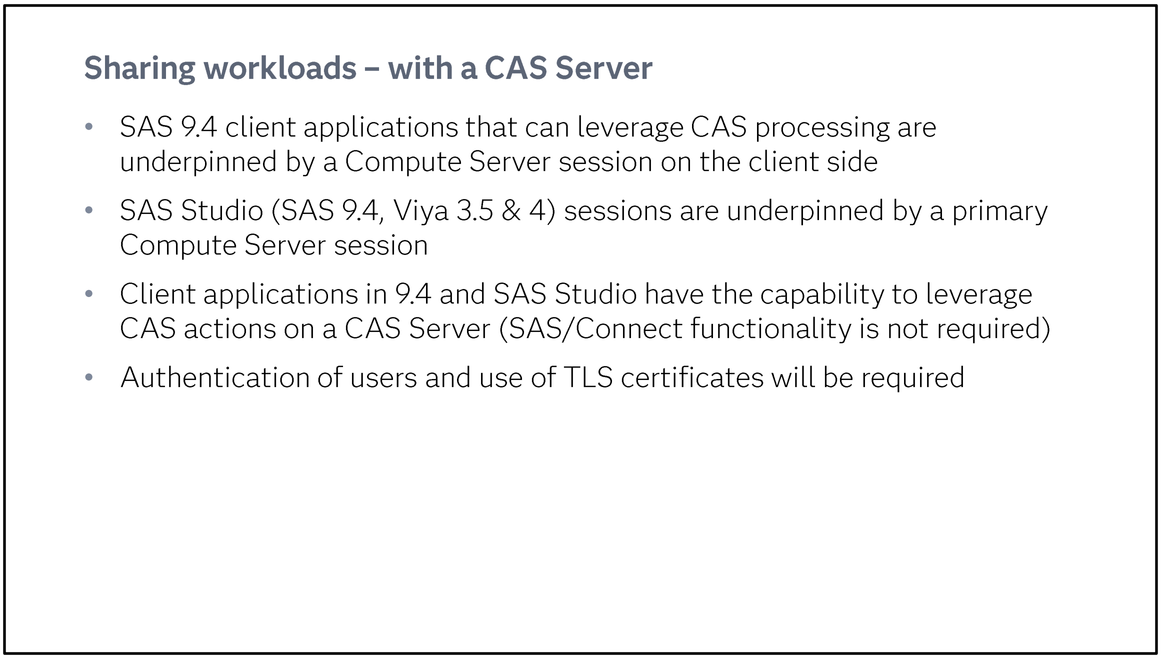
SAS 9.4 and Viya 3.5 Integrating with CAS and Connect Server on Viya 4

In this simplified diagram, the connections to SAS Viya 4 from other SAS platform technologies are identified. The key takeaways are:

Both SAS 9.4 and Viya 3.5 can connect to Viya 4 using SAS/Connect technology for what is in essence a SAS Compute Server type session. That is to say, the use cases will see a user from a SAS 9.4 or Viya 3.5 client use such a connection to submit SAS Foundation programs (for example, SAS/Base, SAS/Stat, etc.) to Viya. In some instances, such a connection may only use processing capabilities of a SAS Viya Compute Server, which is for all intents and purposes is what gets started using SAS/Connect. However, such a connection could also allow a user to write their SAS Foundation programs to make a subsequent connection to a SAS Viya 4 CAS Server session. Using such a session provides the processing power of both a SAS Compute Server and a SAS CAS Server session and can limit the volume of data moving between SAS Viya 4 and other platform technologies.

Depending on the type of the visual application in SAS 9.4 or Viya 3.5, users can also create a direct connection to a CAS server in Viya 4. This bypasses the need to use SAS/Connect in this scenario. Using a direct connection to a CAS server allows users to directly harness the power of CAS without an intermediatory step. This approach is useful when SAS Foundation processing in a Compute Server within Viya 4 is not required. Users of this approach need to be mindful of how much data is being moved to and from Viya 4.

In the first half of 2024, an additional approach to allow users who currently have 9.4 or Viya 3.5, to interact directly with Viya 4. Users of version 8.4 of Enterprise Guide onwards, will be able to make a direct connection to a SAS Compute Server in Viya 4. Prior to Enterprise Guide 8.4, users would have had to have made a connection to a SAS 9.4 or Viya 3.5 Workspace server and then made a subsequent connection to Viya 4. With Enterprise Guide 8.4, users within organizations who are adopting to Viya 4 will have more flexibility. This may also help SAS administrators manage the adoption of Enterprise Guide users more coherently through the various stages of adoption to Viya 4.



Sharing workloads – with a CAS Server

- SAS 9.4 client applications that can leverage CAS processing are underpinned by a Compute Server session on the client side
- SAS Studio (SAS 9.4, Viya 3.5 & 4) sessions are underpinned by a primary Compute Server session
- Client applications in 9.4 and SAS Studio have the capability to leverage CAS actions on a CAS Server (SAS/Connect functionality is not required)
- Authentication of users and use of TLS certificates will be required

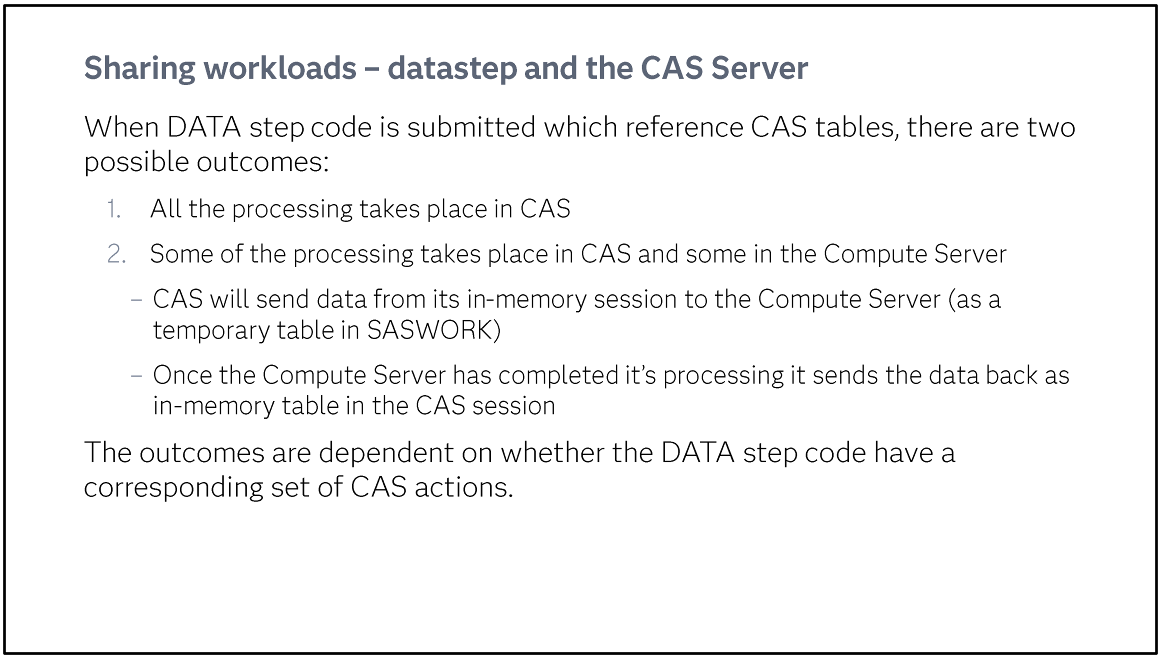
We will take a few moments to focus on the SAS 9.4 or SAS Viya 3.5 client applications accessing a SAS Viya 4 CAS Server.

Since 9.4M5, some of the end-user or client applications like SAS Studio, Enterprise Guide, Enterprise Miner and Add-in to Microsoft Office have been able to access CAS processing in both Viya 3 and Viya 4.

What is important to note is the end-user applications have underpinning them a Compute Server on the 9.4 server-side, which allow the CAS actions to be directed to a Viya 4 CAS Server. To be clear, client applications like SAS Studio, SAS Enterprise Guide or SAS Enterprise Miner must have 9.4 workspace server sessions started. The end-user applications cannot connect directly to a CAS session running in Viya 4 in the initial release. All traffic must be routed through the SAS 9.4 server first. Data can also be seamlessly moved between Viya 4 CAS Servers and previous versions of SAS.

It is worth noting that with the release of Enterprise Guide 8.4, users who have access to both 9.4 environments and Viya 4 environments have 2 options from which to leverage CAS server processing. If the user wants to leverage artifacts and processing within 9.4 and then leverage a CAS Server in Viya 4, the user will be able to do so. As mentioned earlier, that will entail an Enterprise Guide 8.4 user starting a 9.4 workspace server session first, before one or more parts of the workflow request a connection to a CAS server. The second option, available to Enterprise Guide 8.4 users, is to bypass SAS 9.4 processing completely and initiate a Compute Server session on Viya 4. From this Compute Server session, the user can then start a CAS Server session.

To allow this integration, users must be authenticated both within the SAS 9.4 or Viya 3.5 environment and the Viya 4 environment. In addition, CA certificates must be placed in the relevant locations for end user applications. Please consult the administration documentation for SAS 9.4 and Enterprise Guide 8.4 for more details. Similar documentation also exists for Viya 3.5.



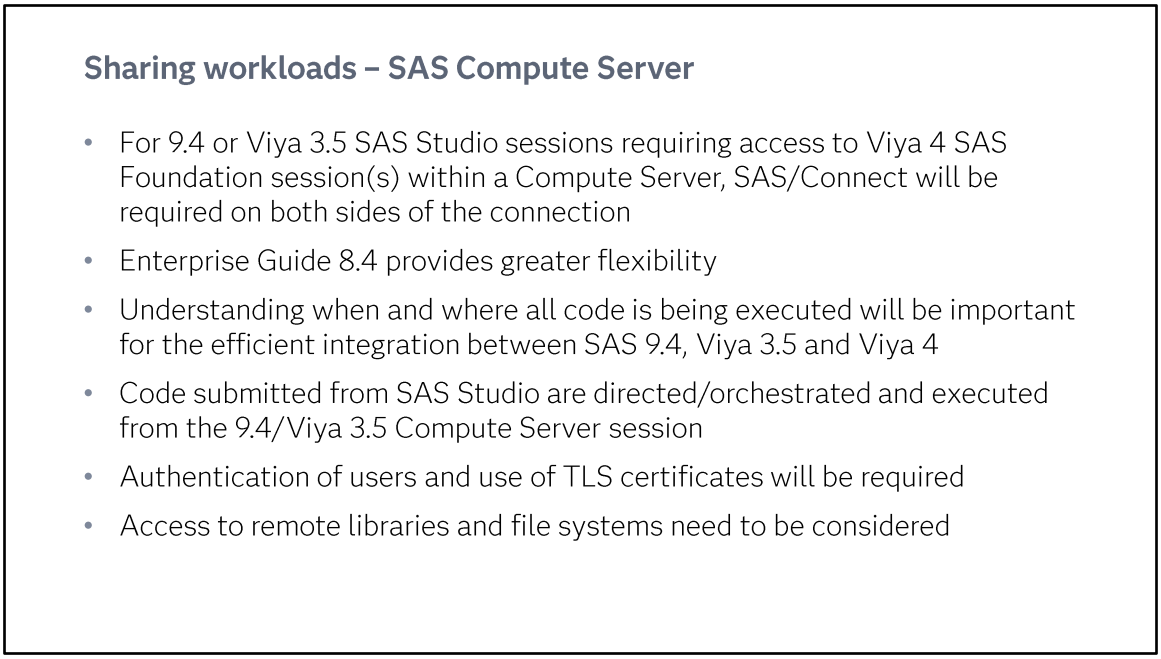
Sharing workloads – datastep and the CAS Server

When DATA step code is submitted which reference CAS tables, there are two possible outcomes:

1. All the processing takes place in CAS
2. Some of the processing takes place in CAS and some in the Compute Server
   - CAS will send data from its in-memory session to the Compute Server (as a temporary table in SASWORK)
   - Once the Compute Server has completed it’s processing it sends the data back as in-memory table in the CAS session

The outcomes are dependent on whether the DATA step code have a corresponding set of CAS actions.

Let’s take a few moments to consider what happens when datastep workloads are shared between Compute or Workspace Servers with CAS Servers.
The flexibility of datastep code allows SAS programmers to submit datastep code and have much of it run in CAS in a seamless fashion. This is a great feature, but care needs to be taken to ensure that any processing of the code ends up running where the programmer intended.
For example, if the program contains datastep which does not have an equivalent set of CAS actions, then the data table is moved across the network to the SAS 9.4 or Viya 3.5 compute server so that the datastep can be executed successfully. If the table is small then the user may not even notice such data movement, but for large data tables found in say an MPP CAS Server, the end-user experience may be sub-optimal due to the time taken for the data to move between the network and written to disk.
So, the recommendation is for SAS programmers that wish to use CAS Server processing, is to write their code in such a way to minimize data movement between Compute or Workspace Server to a CAS Server in Viya 4.



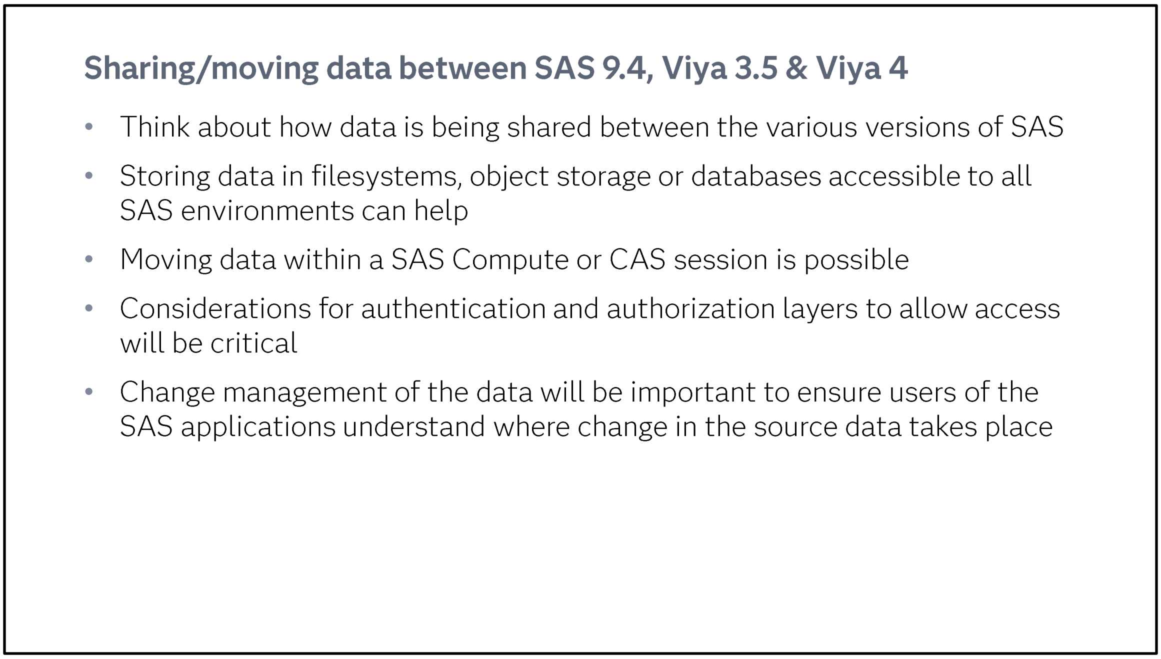
Sharing workloads – SAS Compute Server

- For 9.4 or Viya 3.5 SAS Studio sessions requiring access to Viya 4 SAS Foundation session(s) within a Compute Server, SAS/Connect will be required on both sides of the connection
- Enterprise Guide 8.4 provides greater flexibility
- Understanding when and where all code is being executed will be important for the efficient integration between SAS 9.4, Viya 3.5 and Viya 4
- Code submitted from SAS Studio are directed/orchestrated and executed from the 9.4/Viya 3.5 Compute Server session
- Authentication of users and use of TLS certificates will be required
- Access to remote libraries and file systems need to be considered

On this slide we now turn our attention to SAS 9.4 or Viya 3.5 based SAS Studio sessions leveraging a remote SAS session in Viya 4. Here the term “SAS session in Viya 4” refers to the ability to run SAS Foundation code within Viya 4 using SAS/Connect sessions. This is done using SAS/Connect software. At the programming-level, users will use SAS/Connect statements like SIGNON and RSUBMIT to direct code to execute either in their 9.4 or Viya 3.5 Compute Server or the Viya Compute session or within SAS/Connect sessions. In addition to executing code remotely, SAS/Connect also allows data transfer between the Viya 4 and previous releases of SAS.

Now let us consider Enterprise Guide 8.4. For direct connections to a Compute Server in Viya 4, the SAS administrator will provide connection details including whether a dedicated client ID and secret are in use, or whether Enterprise Guide can use a built-in “seguide” client ID. Take a moment to read more details in the SAS Enterprise Guide 8.4 user guide. It should be noted that Enterprise Guide 8.4 users can make connections to SAS 9.4 workspace servers which in turn using SAS/Connect, can make a connection to SAS Viya 4. The use of SAS/Connect in this approach would require SAS 9.4M7 or later. Executing code within a Viya 4 SAS/Connect session will require users to be provided with a SASWORK area. Indeed, it is important that the amount of compute resources like RAM, CPU cores and performant storage available to SAS Viya 4 Compute Server pods, should at least match or be improved when comparing against the existing SAS 9.4 environment.

Again, users will need to be authenticated in both 9.4 and Viya environments, and the SAS/Connect configuration in both Viya and SAS 9.4 will need a TLS certificate. Please refer to the SAS documentation for further details.

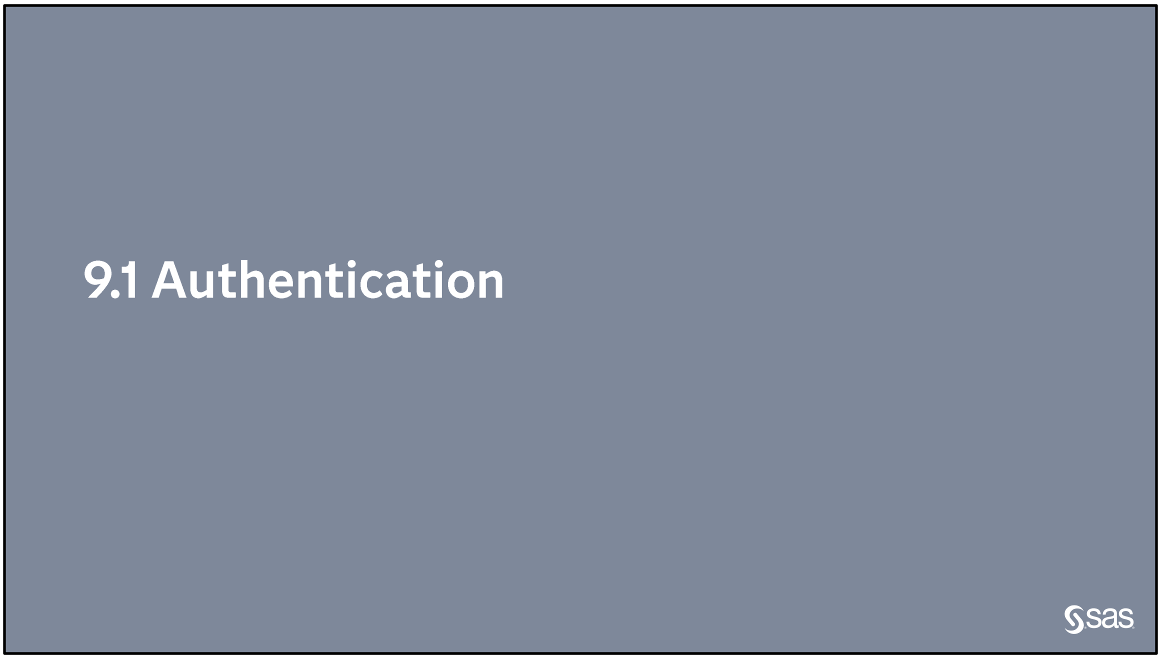


Sharing/moving data between SAS 9.4, Viya 3.5 & Viya 4

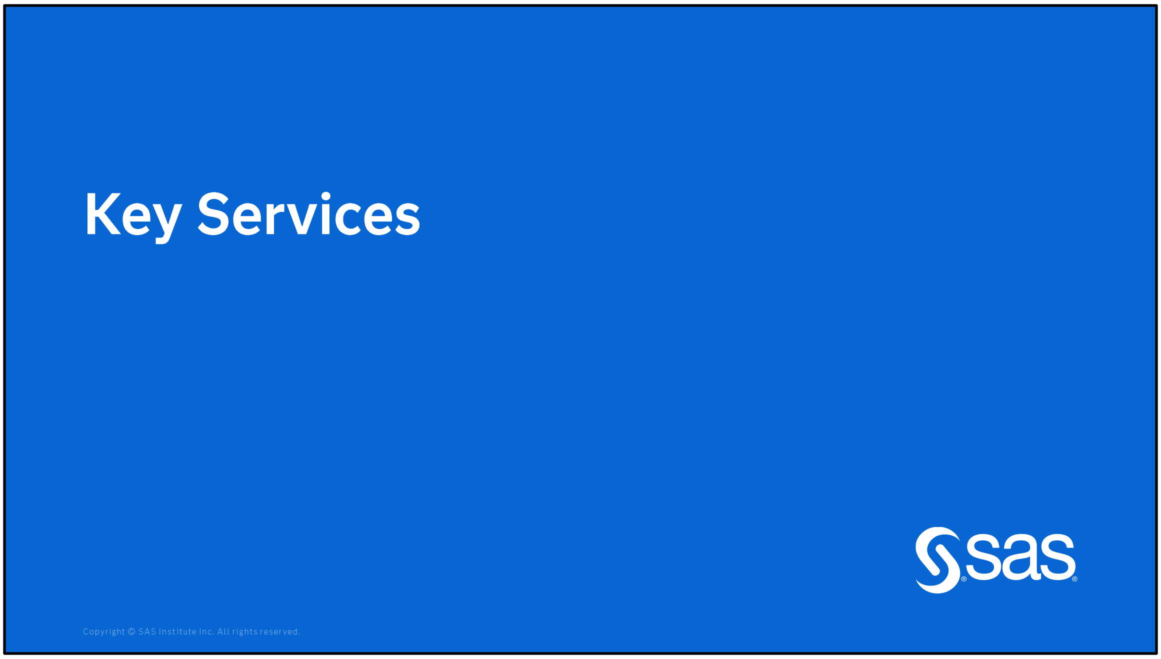
- Think about how data is being shared between the various versions of SAS
- Storing data in filesystems, object storage or databases accessible to all SAS environments can help
- Moving data within a SAS Compute or CAS session is possible
- Considerations for authentication and authorization layers to allow access will be critical
- Change management of the data will be important to ensure users of the SAS applications understand where change in the source data takes place

Moving data or at least accessing data between Viya 4 and previous releases or versions of SAS is supported. However, the approach to moving data needs to be considered by the team responsible for data management. How will change management of data being accessed and modified by 2 or 3 versions of SAS be enforced? If the plan is to enable access through a shared storage area, what security needs to be put in place for all versions of SAS to access it? What storage technologies are available to use from all operational versions of SAS the customer may have?
One final note on storage capacity. Using datastep can enable data movement between versions of SAS, but care must be taken to ensure sufficient capacity is available on both the SAS Viya 4 and the other SAS platforms.

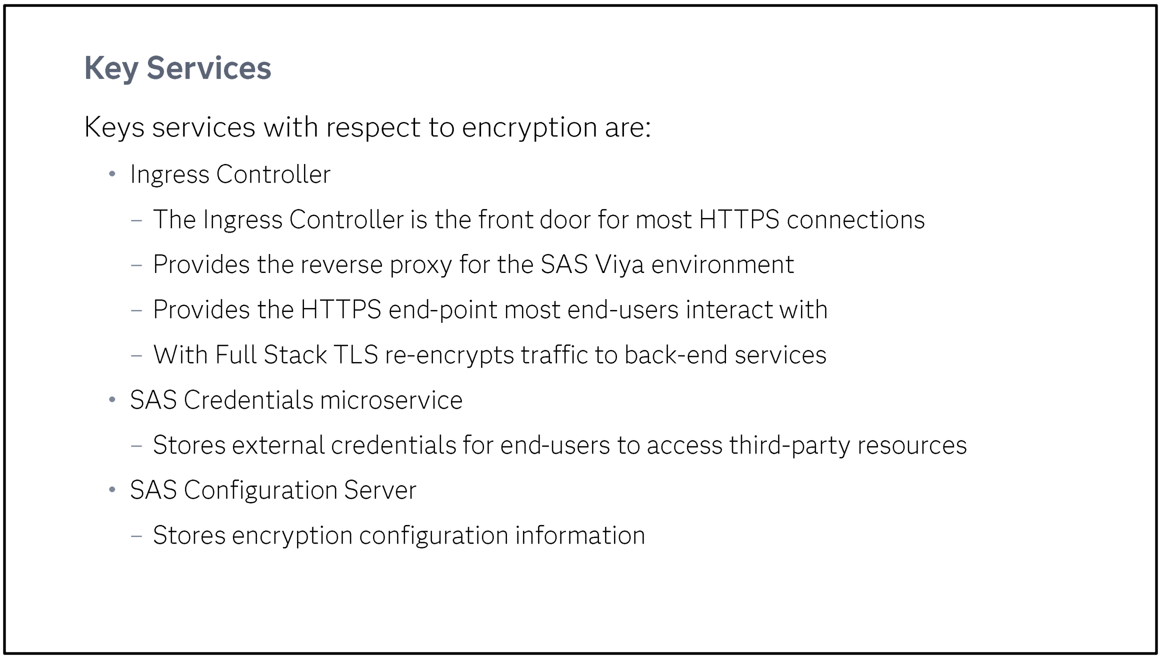
Lesson 9: Security



9.1 Authentication



Key Services



Key Services

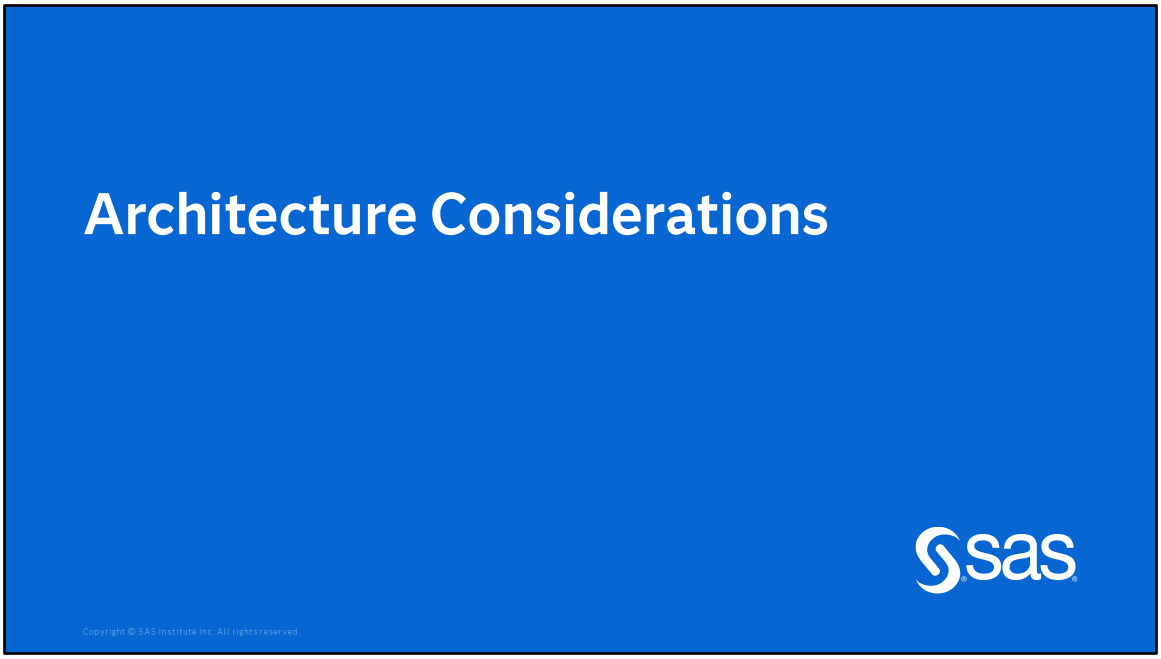
Keys services with respect to encryption are:

- Ingress Controller
  - The Ingress Controller is the front door for most HTTPS connections
  - Provides the reverse proxy for the SAS Viya environment
  - Provides the HTTPS end-point most end-users interact with
  - With Full Stack TLS re-encrypts traffic to back-end services
- SAS Credentials microservice
  - Stores external credentials for end-users to access third-party resources
- SAS Configuration Server
  - Stores encryption configuration information

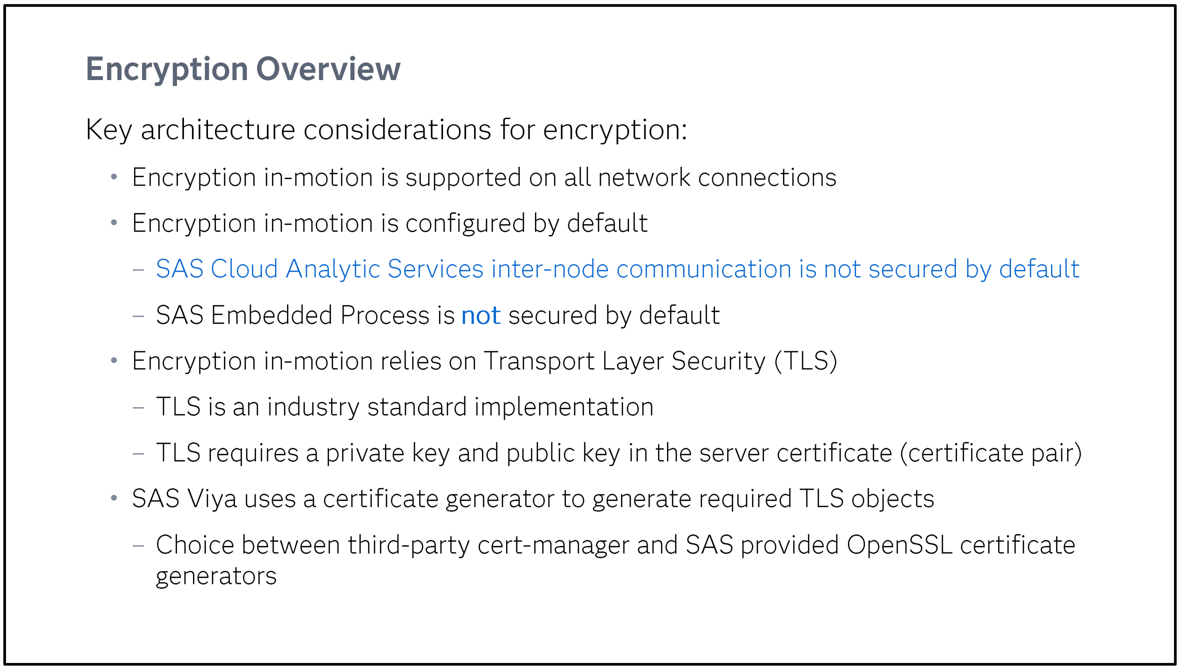
Our key services with respect to encryption are the Ingress Controller, NGINX.
The Ingress Controller is the front door for most of our HTTPS connections.
It provides a reverse proxy for the SAS Viya environment.
It provides the HTTPS endpoint that most users interact with.
With full stack TLS, it's going to re-encrypt the traffic to the backend services.

The SAS Credentials microservice stores external credentials for end users to access third-party resources.

The SAS Configuration Server stores encryption configuration information.



Architecture Considerations



Encryption Overview

Key architecture considerations for encryption:

- Encryption in-motion is supported on all network connections
- Encryption in-motion is configured by default
  - SAS Cloud Analytic Services inter-node communication is not secured by default
  - SAS Embedded Process is not secured by default
- Encryption in-motion relies on Transport Layer Security (TLS)
  - TLS is an industry standard implementation
  - TLS requires a private key and public key in the server certificate (certificate pair)
- SAS Viya uses a certificate generator to generate required TLS objects
  - Choice between third-party cert-manager and SAS provided OpenSSL certificate generators

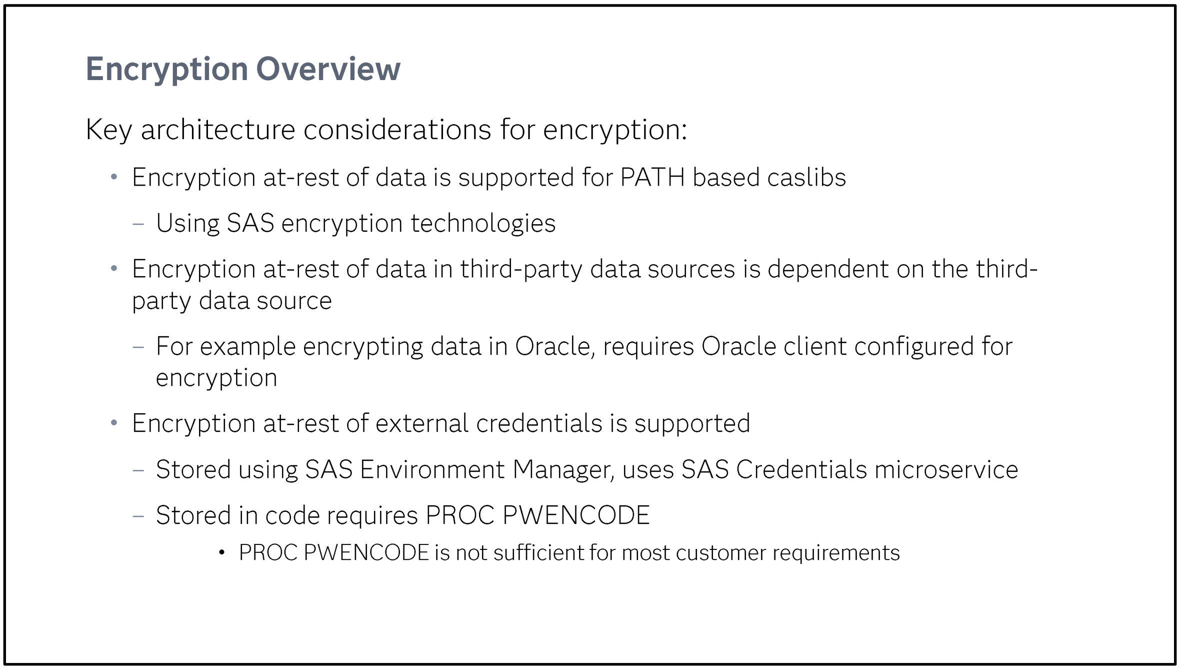
So our key architecture considerations for encryption,

Encryption in Motion is supported on all network connections.

Encryption in Motion is configured by default.
But the SAS Cloud Analytics services inter-node communication is not secured by default.
Equally, our SAS-embedded process is not secured by default.

Encryption in Motion relies on transport layer security.
TLS is an industry standard implementation.
TLS requires a private key and public key.
That public key is inside the server certificate, and we'll call that our certificate pair.

SAS Viya uses a certificate generator to generate the required TLS objects.
We have a choice between the third-party cert manager certificate generator
and the SAS-provided OpenSSL certificate generator.



Encryption Overview

Key architecture considerations for encryption:

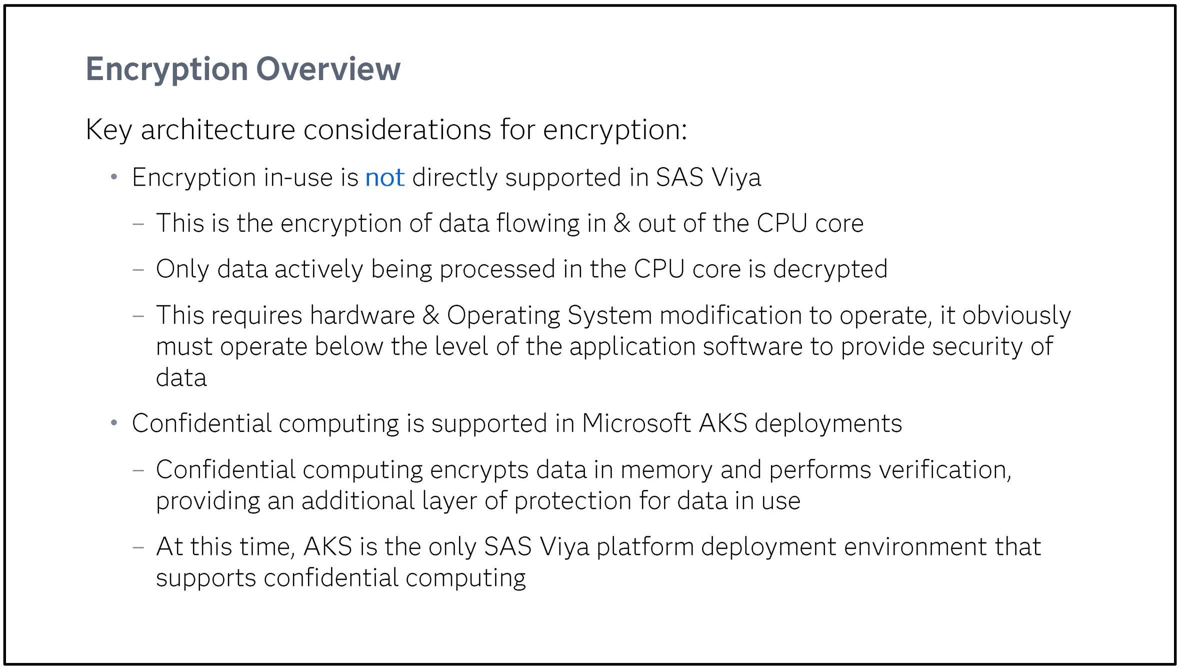
- Encryption at-rest of data is supported for PATH based caslibs
  - Using SAS encryption technologies
- Encryption at-rest of data in third-party data sources is dependent on the third-party data source
  - For example encrypting data in Oracle, requires Oracle client configured for encryption
- Encryption at-rest of external credentials is supported
  - Stored using SAS Environment Manager, uses SAS Credentials microservice
  - Stored in code requires PROC PWENCODE
    - PROC PWENCODE is not sufficient for most customer requirements

Further architecture considerations for encryption,

Encryption at Rest of data is supported for path-based CASlibs.
This is using SAS encryption technology.

Encryption at Rest of data in third-party data sources is dependent on that third-party data source.
For example, encrypting data in Oracle requires Oracle to be configured for encryption.

Encryption at Rest of external credentials is supported.
External credentials can be stored using the environment manager, and this uses the SAS credentials microservice.
Or we could use ENCODE, PROC PW ENCODE, but we should be aware that PROC PW ENCODE is not sufficient for most customer requirements around security.



Encryption Overview

Key architecture considerations for encryption:

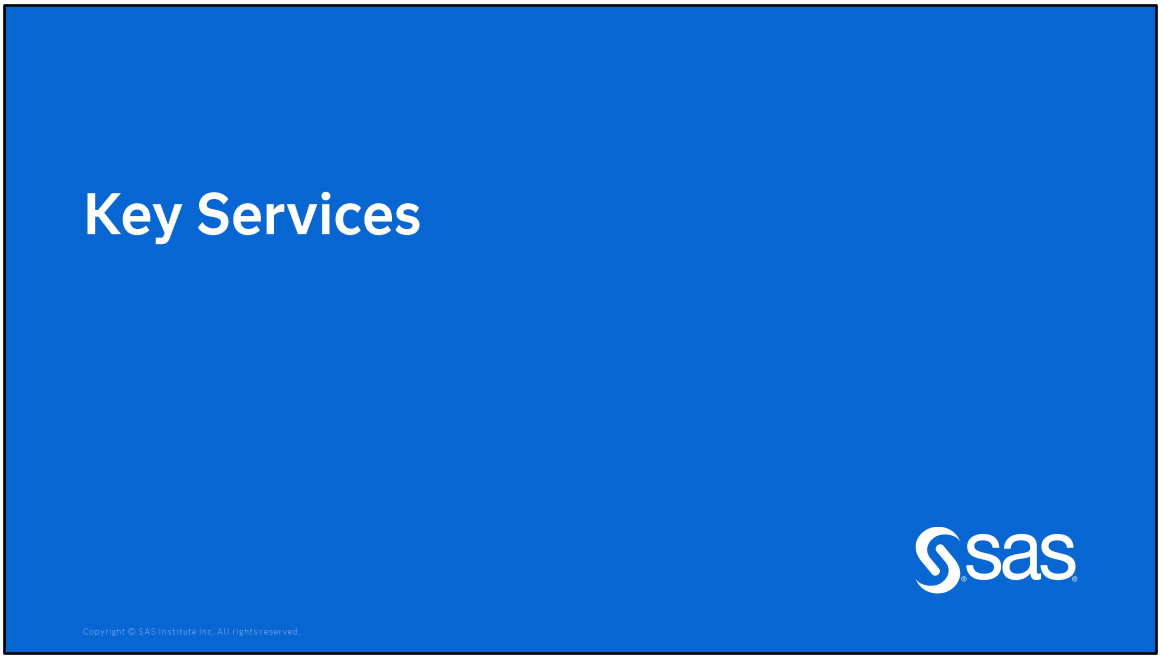
- Encryption in-use is not directly supported in SAS Viya
  - This is the encryption of data flowing in & out of the CPU core
  - Only data actively being processed in the CPU core is decrypted
  - This requires hardware & Operating System modification to operate, it obviously must operate below the level of the application software to provide security of data
- Confidential computing is supported in Microsoft AKS deployments
  - Confidential computing encrypts data in memory and performs verification, providing an additional layer of protection for data in use
  - At this time, AKS is the only SAS Viya platform deployment environment that supports confidential computing

Some further architecture considerations for encryption,

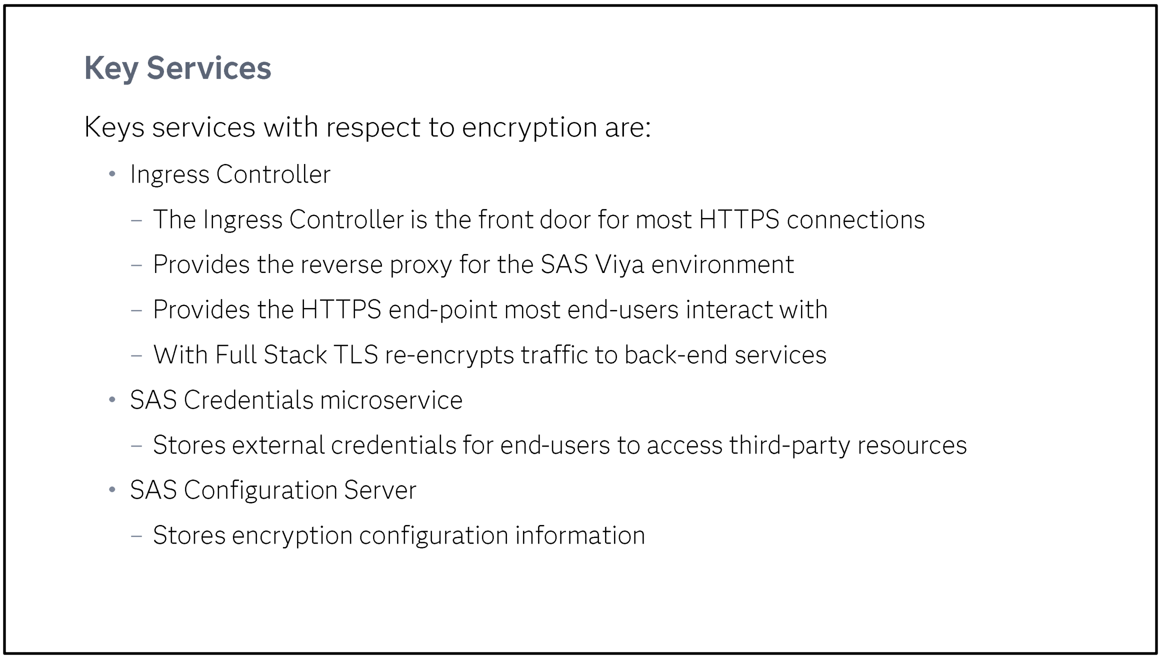
Encryption in use is not directly supported in SAS Viya.
This is the encryption of data flowing in and out of the CPU cores.
Only data that's actively being processed in the core is decrypted.
This requires hardware and operating system modifications to operate.
Obviously, this must operate at a layer below that where SAS runs as an application software.
Otherwise, there'd be no security of the data.

This concept of encryption in use is sometimes referred to as confidential computing.
Confidential computing is supported in Microsoft AKS deployments.
So confidential computing encrypts the data in memory and performs verification, providing an additional layer of protection for data in use.
At this time, AKS is the only SAS Viya platform deployment environment that supports confidential computing.

9.2 Encryption



Key Services



Key Services

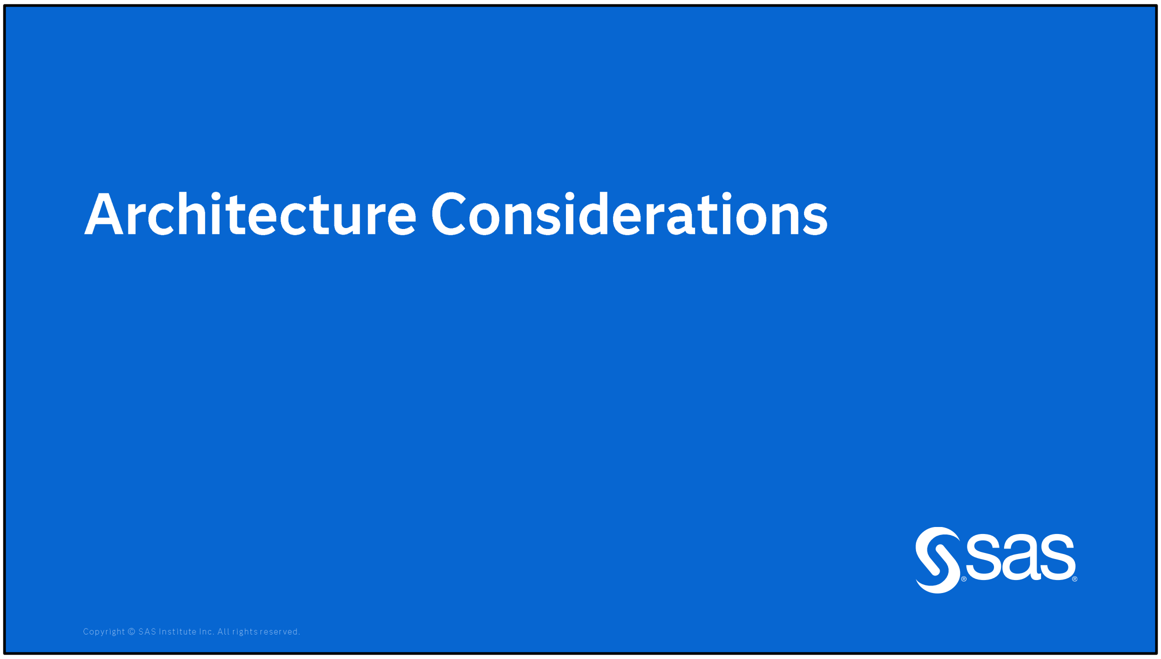
Keys services with respect to encryption are:

- Ingress Controller
  - The Ingress Controller is the front door for most HTTPS connections
  - Provides the reverse proxy for the SAS Viya environment
  - Provides the HTTPS end-point most end-users interact with
  - With Full Stack TLS re-encrypts traffic to back-end services
- SAS Credentials microservice
  - Stores external credentials for end-users to access third-party resources
- SAS Configuration Server
  - Stores encryption configuration information

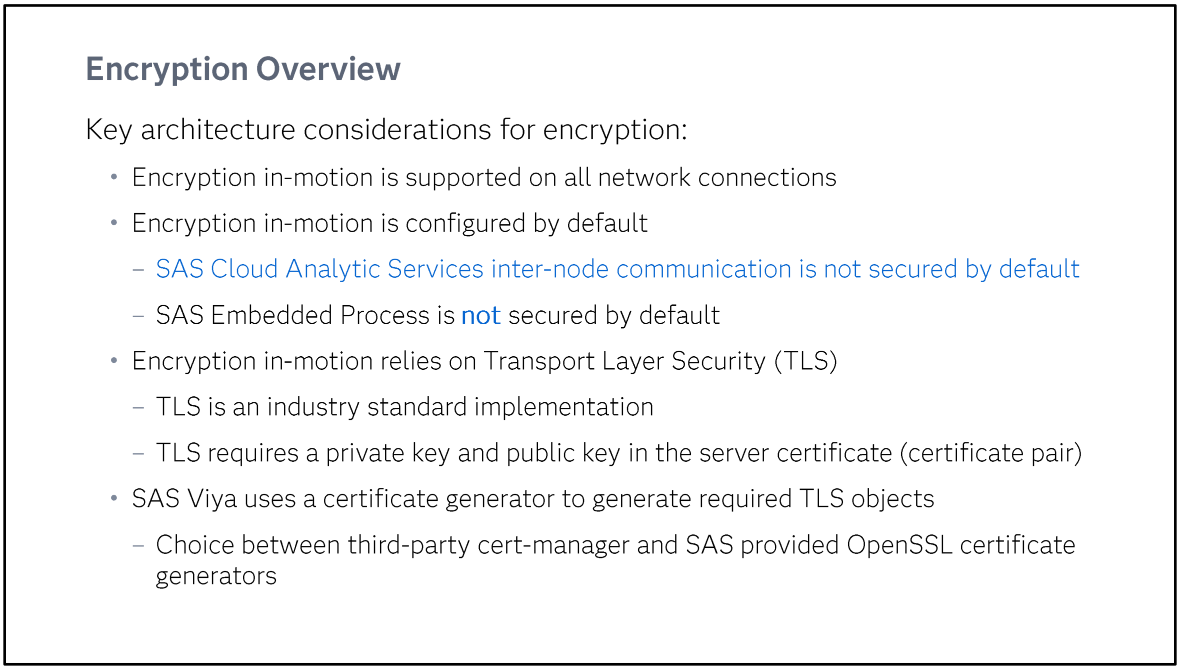
Our key services with respect to encryption are the Ingress Controller, NGINX.
The Ingress Controller is the front door for most of our HTTPS connections.
It provides a reverse proxy for the SAS Viya environment.
It provides the HTTPS endpoint that most users interact with.
With full stack TLS, it's going to re-encrypt the traffic to the backend services.

The SAS Credentials microservice stores external credentials for end users to access third-party resources.

The SAS Configuration Server stores encryption configuration information.



Architecture Considerations



Encryption Overview

Key architecture considerations for encryption:

- Encryption in-motion is supported on all network connections
- Encryption in-motion is configured by default
  - SAS Cloud Analytic Services inter-node communication is not secured by default
  - SAS Embedded Process is **not** secured by default
- Encryption in-motion relies on Transport Layer Security (TLS)
  - TLS is an industry standard implementation
  - TLS requires a private key and public key in the server certificate (certificate pair)
- SAS Viya uses a certificate generator to generate required TLS objects
  - Choice between third-party cert-manager and SAS provided OpenSSL certificate generators

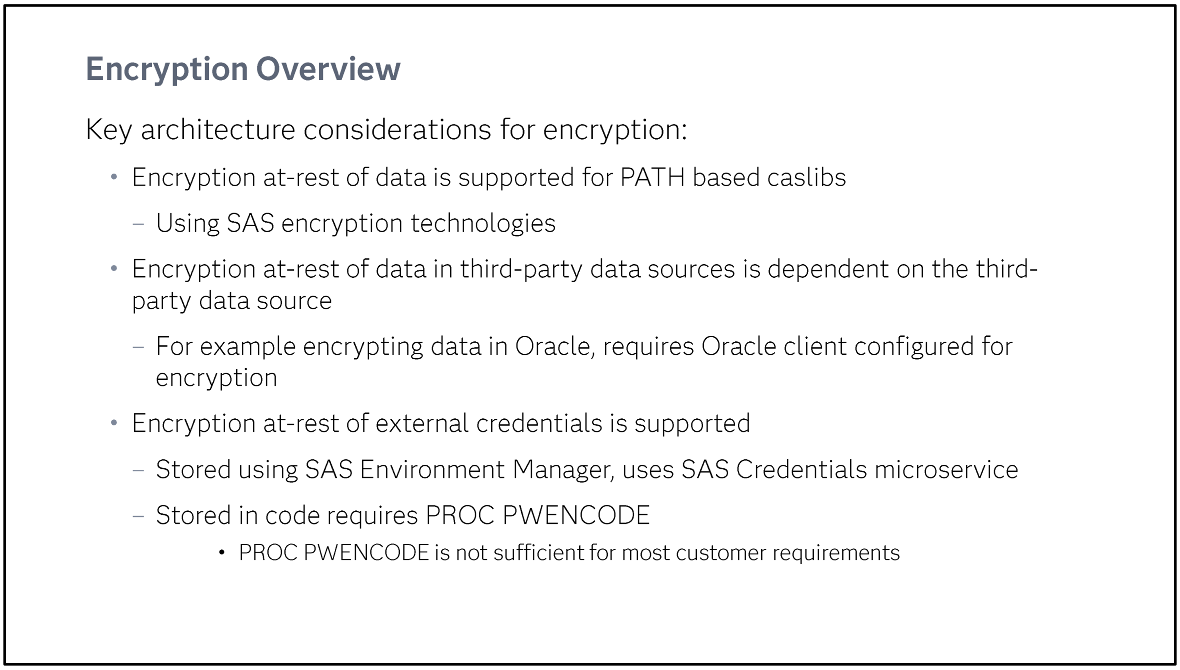
So our key architecture considerations for encryption,

Encryption in Motion is supported on all network connections.

Encryption in Motion is configured by default.
But the SAS Cloud Analytics services inter-node communication is not secured by default.
Equally, our SAS-embedded process is not secured by default.

Encryption in Motion relies on transport layer security.
TLS is an industry standard implementation.
TLS requires a private key and public key.
That public key is inside the server certificate, and we'll call that our certificate pair.

SAS Viya uses a certificate generator to generate the required TLS objects.
We have a choice between the third-party cert manager certificate generator
and the SAS-provided OpenSSL certificate generator.



Encryption Overview

Key architecture considerations for encryption:

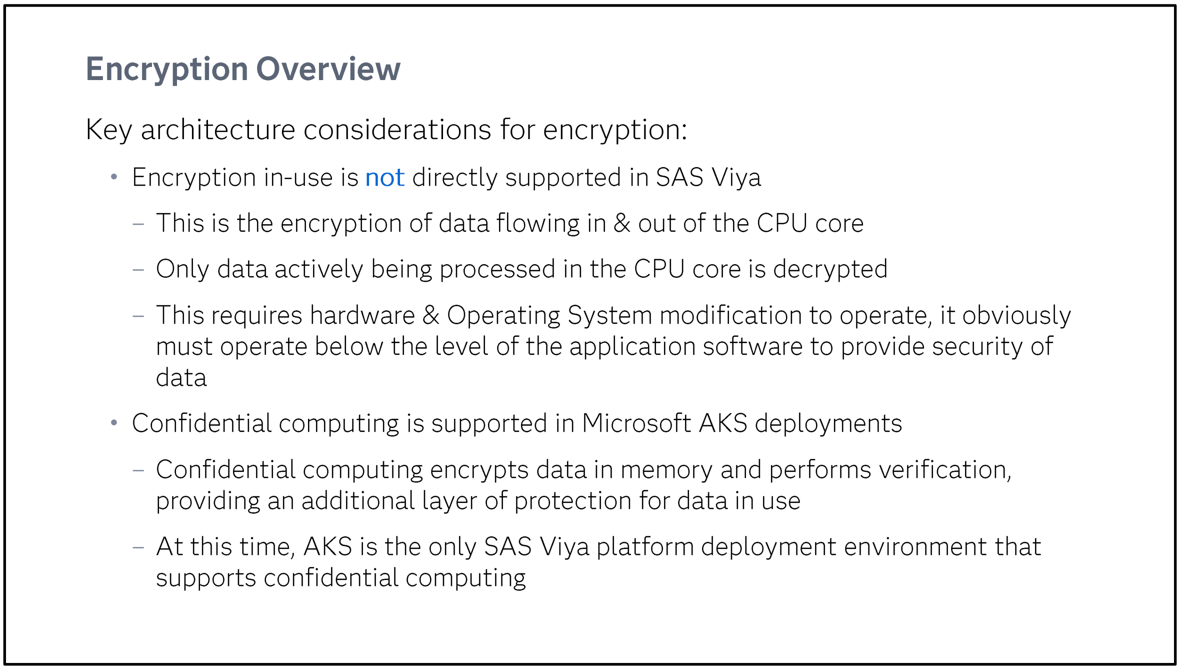
- Encryption at-rest of data is supported for PATH based caslibs
  - Using SAS encryption technologies
- Encryption at-rest of data in third-party data sources is dependent on the third-party data source
  - For example encrypting data in Oracle, requires Oracle client configured for encryption
- Encryption at-rest of external credentials is supported
  - Stored using SAS Environment Manager, uses SAS Credentials microservice
  - Stored in code requires PROC PWENCODE
    - PROC PWENCODE is not sufficient for most customer requirements

Further architecture considerations for encryption,

Encryption at Rest of data is supported for path-based CASlibs.
This is using SAS encryption technology.

Encryption at Rest of data in third-party data sources is dependent on that third-party data source.
For example, encrypting data in Oracle requires Oracle to be configured for encryption.

Encryption at Rest of external credentials is supported.
External credentials can be stored using the environment manager, and this uses the SAS credentials microservice.
Or we could use ENCODE, PROC PW ENCODE, but we should be aware that PROC PW ENCODE is not sufficient for most customer requirements around security.



Encryption Overview

Key architecture considerations for encryption:

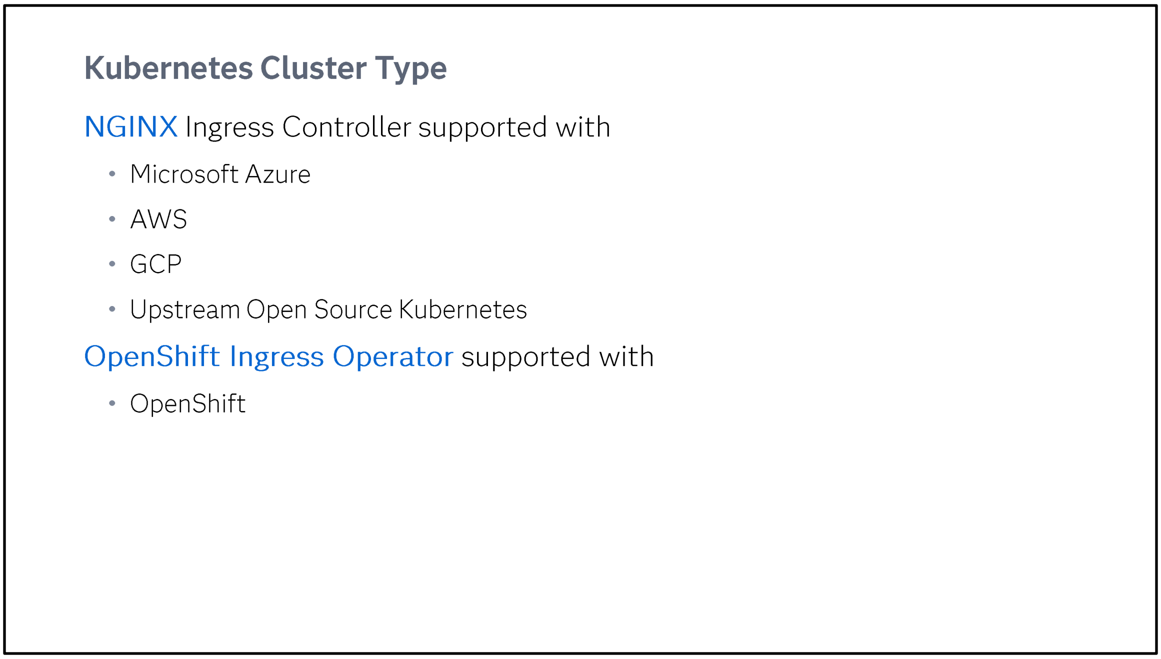
- Encryption in-use is not directly supported in SAS Viya
  - This is the encryption of data flowing in & out of the CPU core
  - Only data actively being processed in the CPU core is decrypted
  - This requires hardware & Operating System modification to operate, it obviously must operate below the level of the application software to provide security of data
- Confidential computing is supported in Microsoft AKS deployments
  - Confidential computing encrypts data in memory and performs verification, providing an additional layer of protection for data in use
  - At this time, AKS is the only SAS Viya platform deployment environment that supports confidential computing

Some further architecture considerations for encryption,

Encryption in use is not directly supported in SAS Viya.
This is the encryption of data flowing in and out of the CPU cores.
Only data that's actively being processed in the core is decrypted.
This requires hardware and operating system modifications to operate.
Obviously, this must operate at a layer below that where SAS runs as an application software.
Otherwise, there'd be no security of the data.

This concept of encryption in use is sometimes referred to as confidential computing.
Confidential computing is supported in Microsoft AKS deployments.
So confidential computing encrypts the data in memory and performs verification, providing an additional layer of protection for data in use.
At this time, AKS is the only SAS Viya platform deployment environment that supports confidential computing.

Design Decisions



Kubernetes Cluster Type

NGINX Ingress Controller supported with

- Microsoft Azure
- AWS
- GCP
- Upstream Open Source Kubernetes

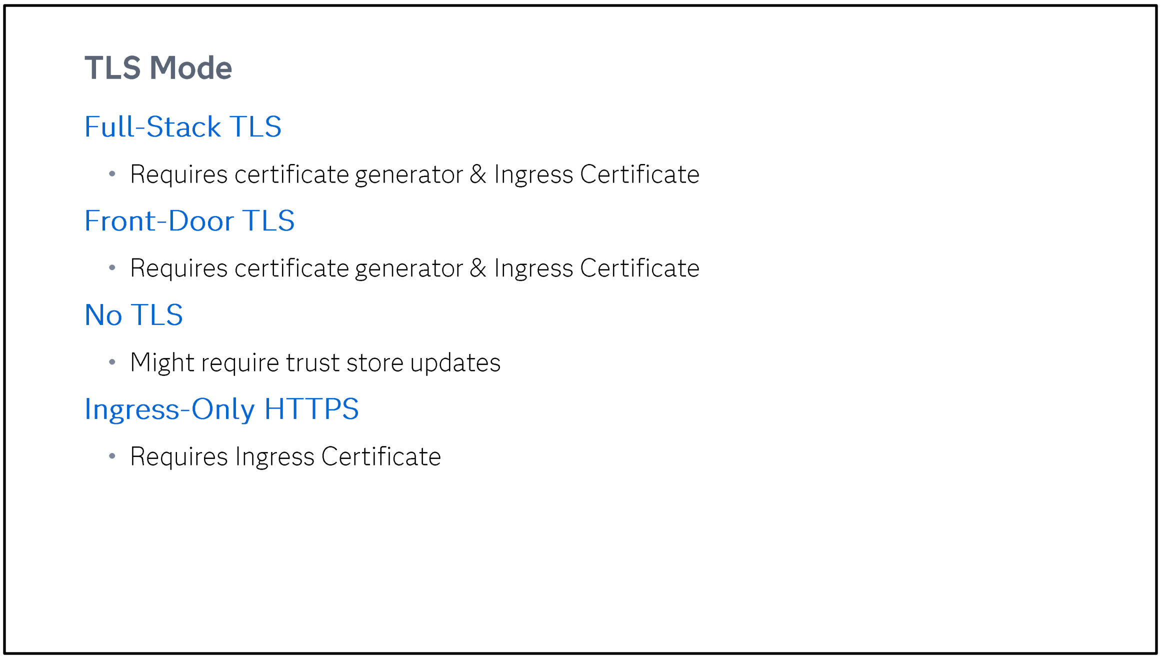
OpenShift Ingress Operator supported with

- OpenShift

Now we'll look at some high-level design decisions that are going to be important to the configuration of encryption for SAS Viya.
First, let's consider the Kubernetes cluster type.

We need to remember that our NGINX ingress controller is supported with Microsoft Azure, AWS, Google, and the upstream open-source Kubernetes,

whereas the OpenShift ingress operator is only supported when we're using OpenShift.



TLS Mode

Full-Stack TLS

- Requires certificate generator & Ingress Certificate

Front-Door TLS

- Requires certificate generator & Ingress Certificate

No TLS

- Might require trust store updates

Ingress-Only HTTPS

- Requires Ingress Certificate

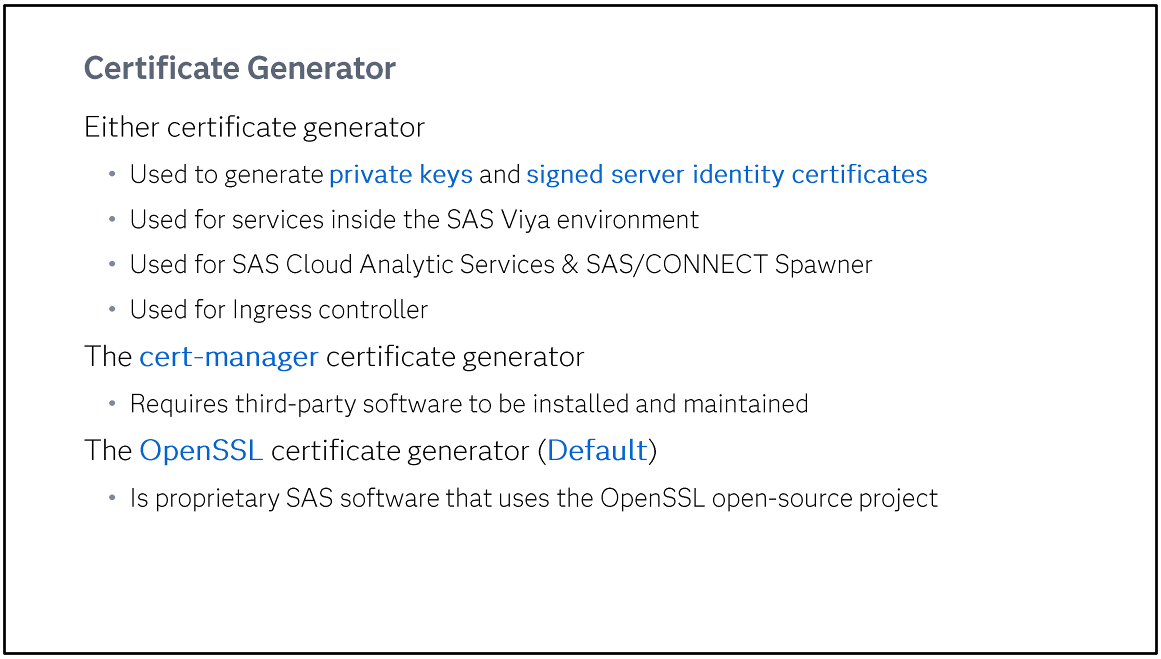
Next, we need to think about the TLS mode.

We could configure full-stack TLS.
That's going to require a certificate generator and an ingress certificate.

We could configure front-door TLS.
Again, we require a certificate generator and an ingress certificate.

We could configure no TLS.
With this, we might still need to update the trust store.
For example, if we're going to connect to a secure LDAP server, we may need to add the chain for that LDAP server into our trust store.

And finally, we have that ingress-only HTTPS mode, where all we're going to require is an ingress certificate.



Certificate Generator

Either certificate generator

- Used to generate private keys and signed server identity certificates
- Used for services inside the SAS Viya environment
- Used for SAS Cloud Analytic Services & SAS/CONNECT Spawner
- Used for Ingress controller

The cert-manager certificate generator

- Requires third-party software to be installed and maintained

The OpenSSL certificate generator (Default)

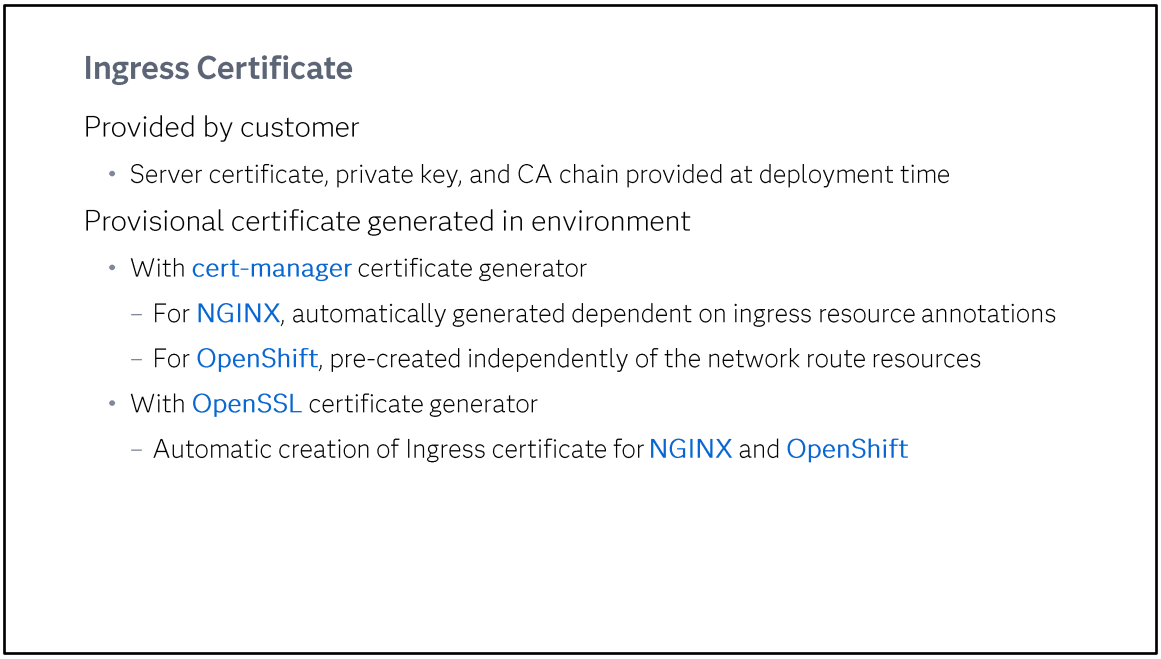
- Is proprietary SAS software that uses the OpenSSL open-source project

And then we have the certificate generator.

Either certificate generator is used to generate the private keys and signed server identity certificates.
Used for services inside the SAS Viya environment.
It's also used for SAS Cloud Analytics services, and our SAS Connect Spawner.
It can be used for our ingress controller.

The cert manager certificate generator requires third-party software to be installed and maintained.
That software needs to be updated on a regular basis.

The OpenSSL certificate generator, which is the default, is proprietary SAS software that uses the OpenSSL open source project.



Ingress Certificate

Provided by customer

- Server certificate, private key, and CA chain provided at deployment time

Provisional certificate generated in environment

- With cert-manager certificate generator
  - For NGINX, automatically generated dependent on ingress resource annotations
  - For OpenShift, pre-created independently of the network route resources
- With OpenSSL certificate generator
  - Automatic creation of Ingress certificate for NGINX and OpenShift

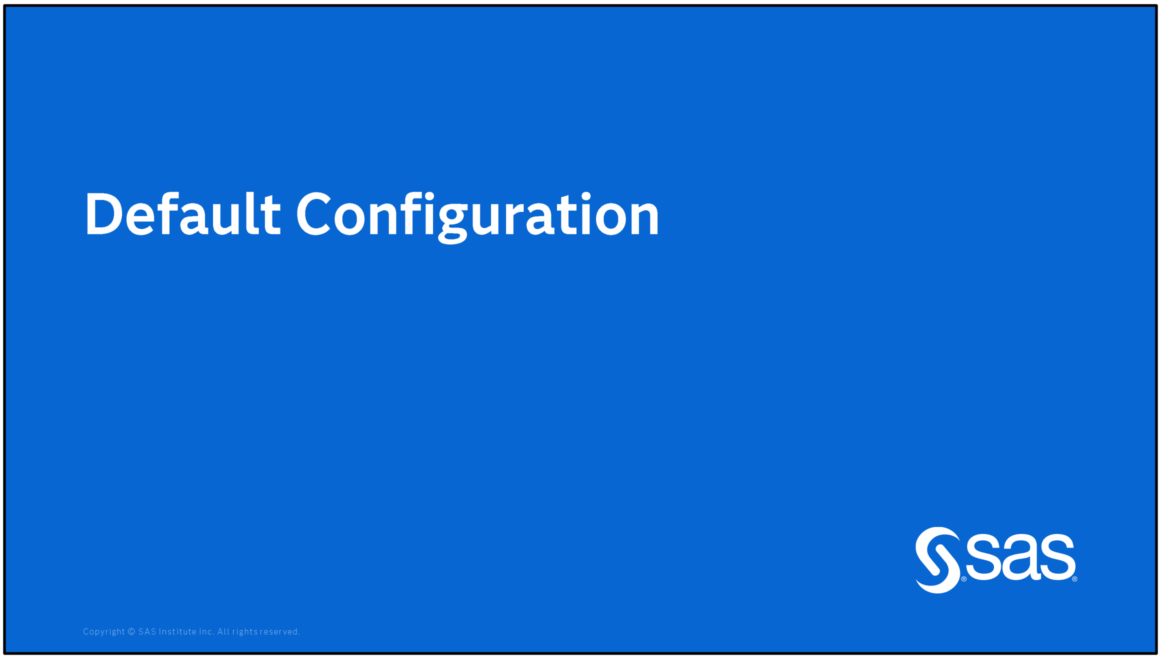
Finally, we need to think about our ingress certificate.

That ingress certificate could be provided by the customer.
That means we need the server certificate, the private key, and the chain, all provided to us at deployment time.

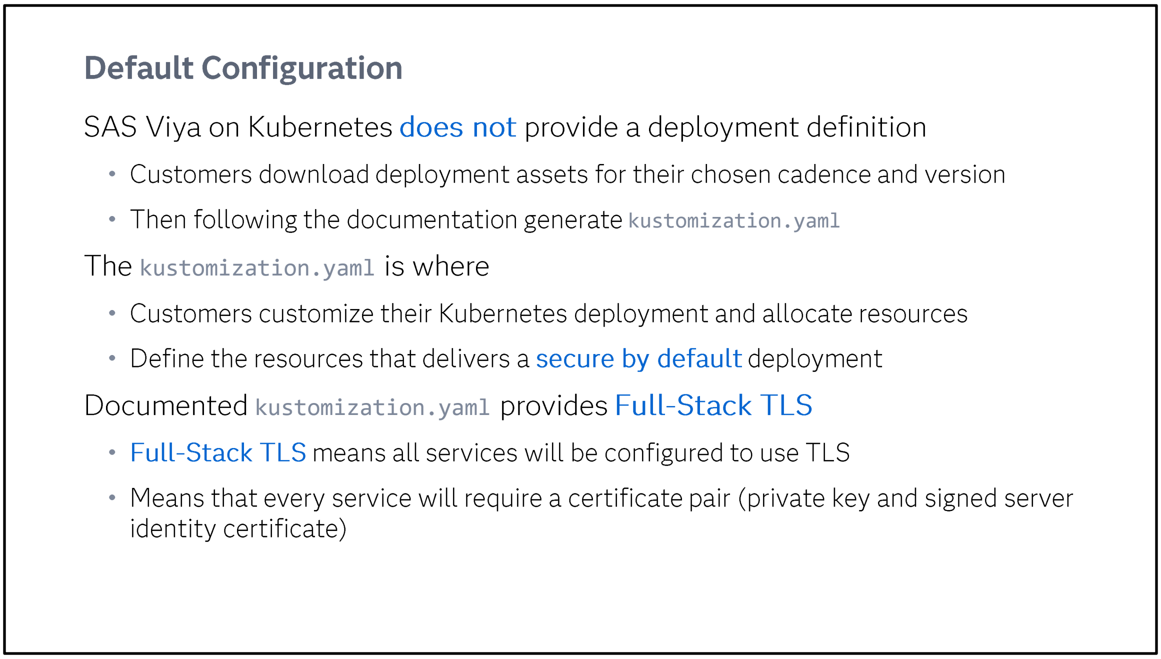
Alternatively, we could use a provisional certificate that's generated in the environment.

With cert manager certificate generator, for NGINX, we can automatically generate that provisional certificate dependent on how we configure our ingress resources.
For OpenShift, we can pre-create that ingress certificate independently of the network route resources.

And with the OpenSSL certificate generator, we can leverage that certificate generator for automatic creation of the ingress certificate for either NGINX or OpenShift.



Default Configuration



Default Configuration

SAS Viya on Kubernetes does not provide a deployment definition

- Customers download deployment assets for their chosen cadence and version
- Then following the documentation generate `kustomization.yaml`

The `kustomization.yaml` is where

- Customers customize their Kubernetes deployment and allocate resources
- Define the resources that delivers a secure by default deployment

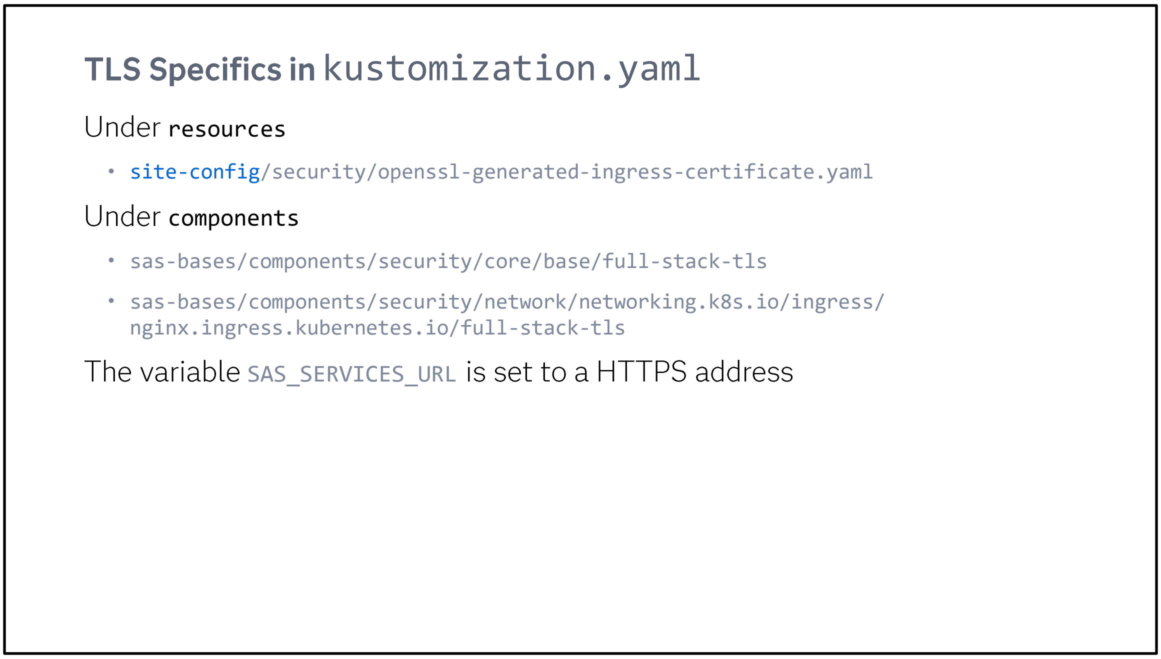
Documented `kustomization.yaml` provides Full-Stack TLS

- Full-Stack TLS means all services will be configured to use TLS
- Means that every service will require a certificate pair (private key and signed server identity certificate)

Now we'll look at the default configuration of TLS.
SAS Viya on Kubernetes does not provide a deployment definition.
Customers download their deployment assets for their chosen cadence and version.
They then follow the documentation to generate their kustomization.yaml.

The kustomization.yaml is where customers customize their Kubernetes deployment and allocate resources.
The kustomization.yaml is where we define the resources that delivers a secure by default deployment.

The documented kustomization.yaml provides full stack TLS.
Full stack TLS means that all services will be configured to use TLS.
It means that every service will require a certificate pair.
That's a private key and signed server identity certificate.



TLS Specifics in `kustomization.yaml`

Under `resources`

- `site-config/security/openssl-generated-ingress-certificate.yaml`

Under `components`

- `sas-bases/components/security/core/base/full-stack-tls`
- `sas-bases/components/security/network/networking.k8s.io/ingress/nginx.ingress.kubernetes.io/full-stack-tls`

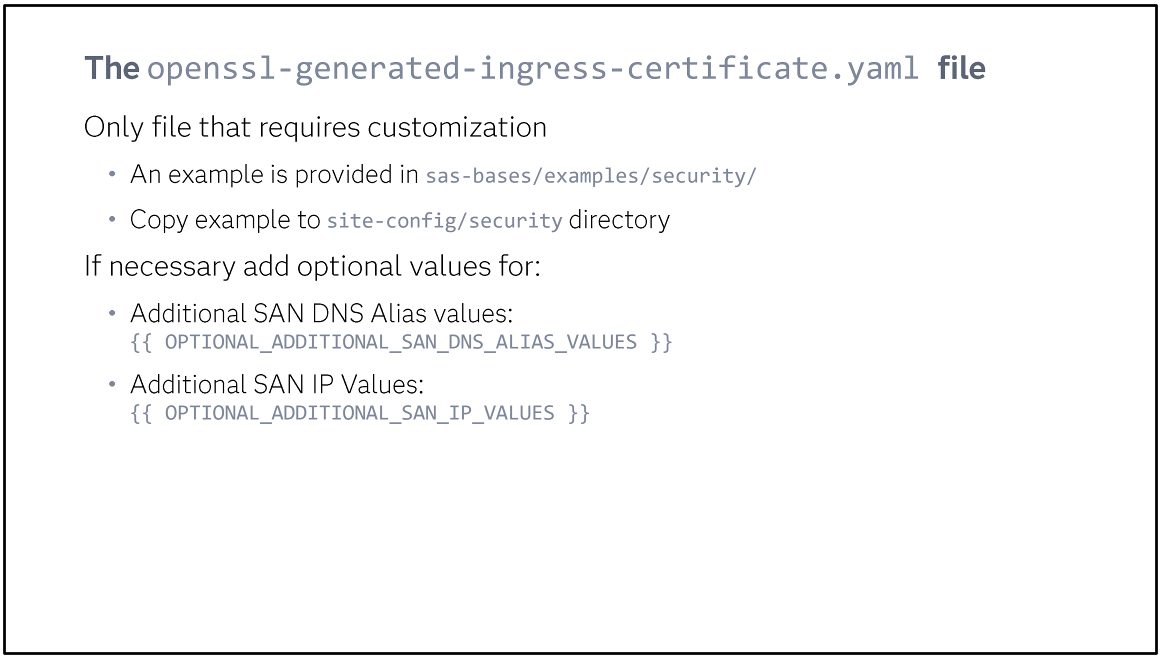
The variable `SAS_SERVICES_URL` is set to a HTTPS address

Let's look at the TLS specifics in our customization.yaml.

Under the resources section, we have under our site config directory an open SSL generated ingress certificate.yaml.

And under components, we have two entries that describe the type of TLS mode that we've deployed, and that's our full stack TLS.
And notice that the second entry is specific to the ingress technology that we're going to use.

And then our variable, the SAS services URL, will also be set to an HTTPS address.



The `openssl-generated-ingress-certificate.yaml` file

Only file that requires customization

- An example is provided in `sas-bases/examples/security/`
- Copy example to `site-config/security` directory

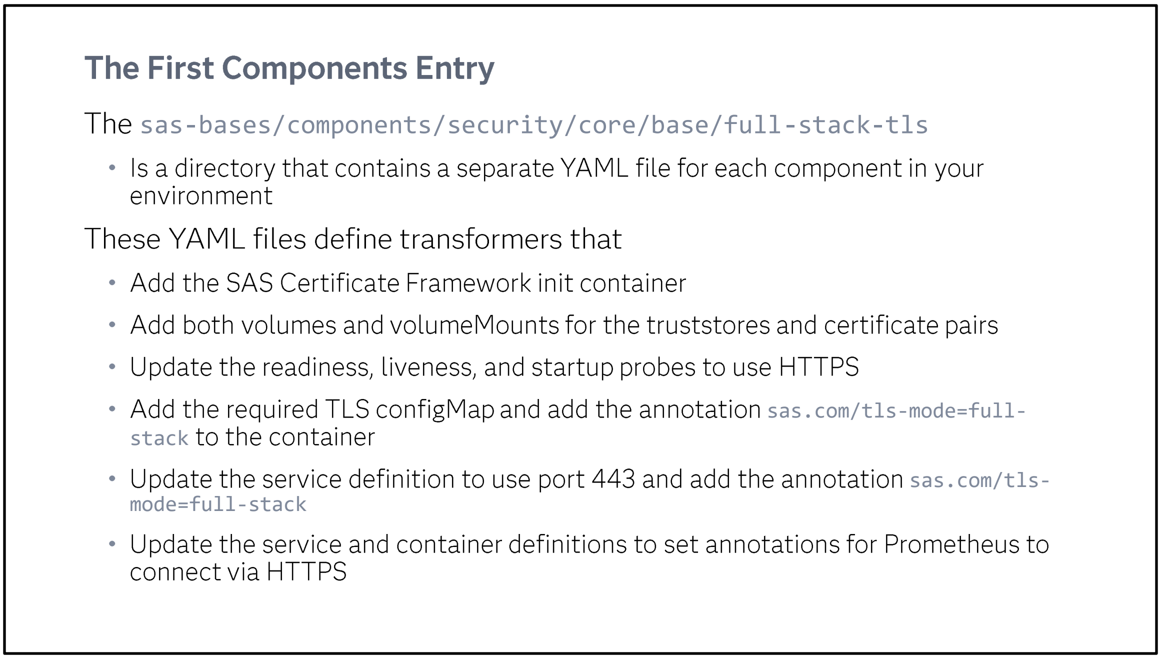
If necessary add optional values for:

- Additional SAN DNS Alias values:
  `{{ OPTIONAL_ADDITIONAL_SAN_DNS_ALIAS_VALUES }}`
- Additional SAN IP Values:
  `{{ OPTIONAL_ADDITIONAL_SAN_IP_VALUES }}`

So let's look at that first file, the open SSL generated ingress certificate.yaml file.

It's the only file that may require customization.
An example is provided under SAS Bases, examples, security.
We'd copy that example to our site config security directory.

And then if necessary, we can add optional values that allow us to add additional SAN DNS or SAN IP values into the certificate that will be generated for our ingress controller.



The First Components Entry

The `sas-bases/components/security/core/base/full-stack-tls`

- Is a directory that contains a separate YAML file for each component in your environment

These YAML files define transformers that

- Add the SAS Certificate Framework init container
- Add both volumes and volumeMounts for the truststores and certificate pairs
- Update the readiness, liveness, and startup probes to use HTTPS
- Add the required TLS configMap and add the annotation `sas.com/tls-mode=full-stack` to the container
- Update the service definition to use port 443 and add the annotation `sas.com/tls-mode=full-stack`
- Update the service and container definitions to set annotations for Prometheus to connect via HTTPS

Next, the first of the entries under components.

So this is security, core, base, full stack TLS.
This is a directory that contains a separate YAML file for each component in your environment.

These YAML files define a series of transformers.
These transformers add the SAS certificate framework init container,

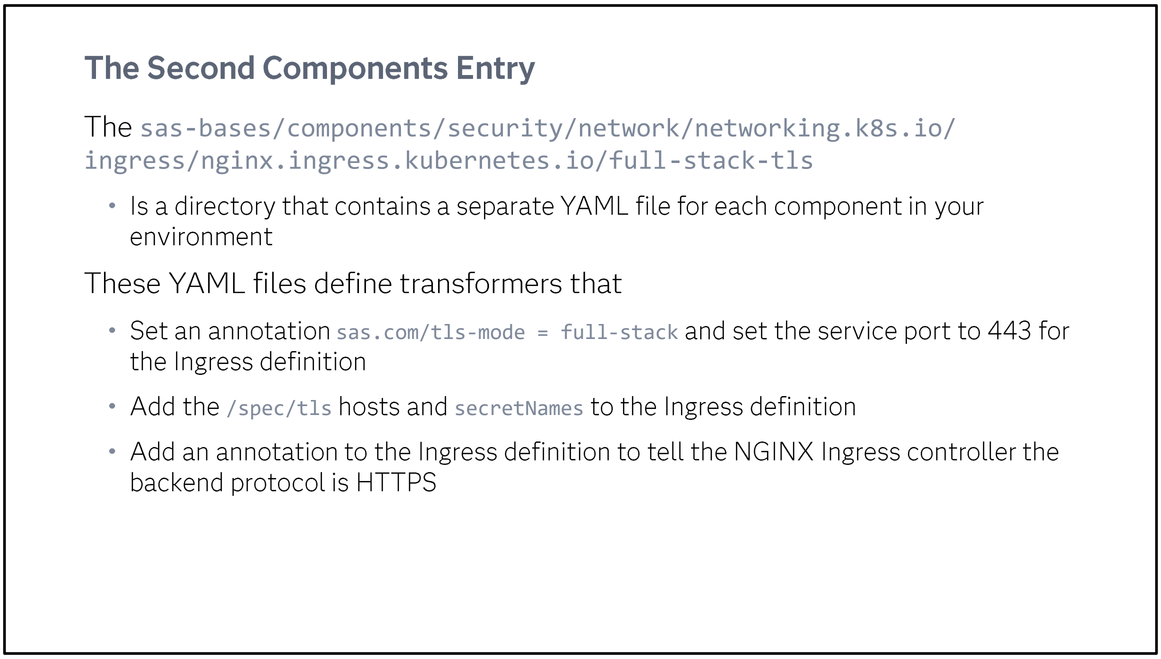
add both volumes and volume mounts for the trust stores and certificate pairs,

update the readiness, liveness, and startup probes to use HTTPS,

add the required TLS config map, and then add an annotation that defines the TLS mode and that's added to the container.
That annotation is not used by the software but is there to make it easy for a user to examine the pod to see what TLS mode has been used.

Also, the transformers update the service definition to use port 443 and add that same annotation to the service definition.

We update the service and container definitions to set annotations for Prometheus to connect via HTTPS.
So all of those are done through the transformers that are in that core base full stack TLS directory.



The Second Components Entry

The `sas-bases/components/security/network/networking.k8s.io/ingress/nginx.ingress.kubernetes.io/full-stack-tls`

- Is a directory that contains a separate YAML file for each component in your environment

These YAML files define transformers that

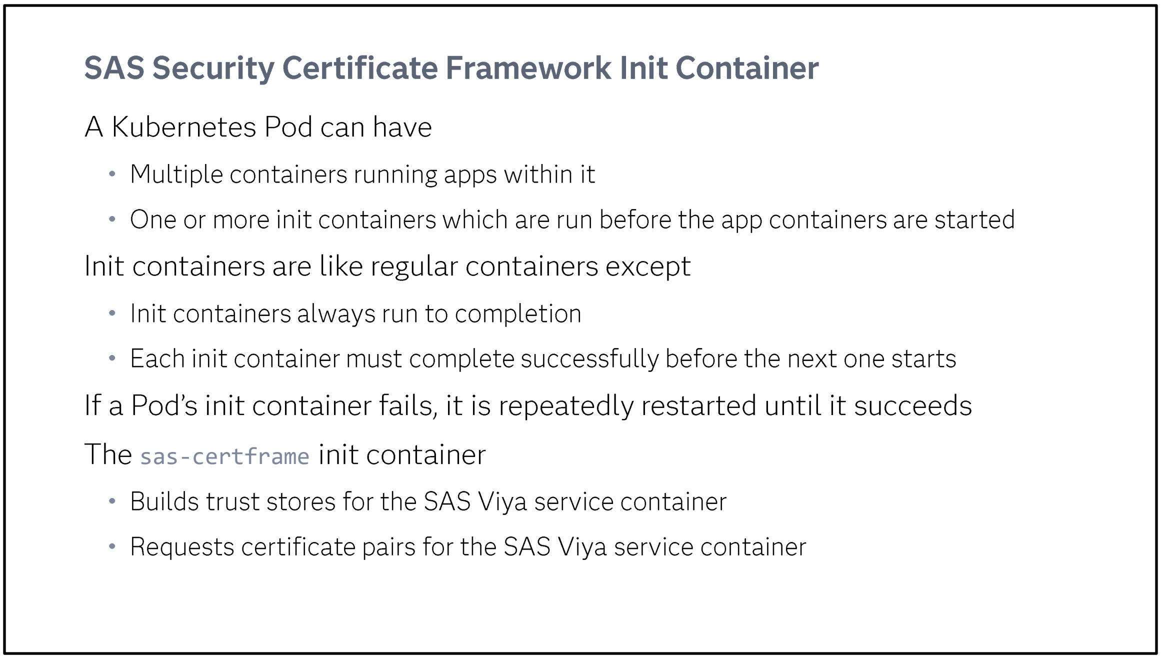
- Set an annotation `sas.com/tls-mode = full-stack` and set the service port to 443 for the Ingress definition
- Add the `/spec/tls` hosts and `secretNames` to the Ingress definition
- Add an annotation to the Ingress definition to tell the NGINX Ingress controller the backend protocol is HTTPS

The second entry under components is the network, networking Kubernetes IO, Ingress, NGINX ingress, full stack TLS.
Again, this is a directory that contains a separate YAML file for each component in your environment.

And these YAML files define a series of transformers, and these transformers set that annotation and the service port to 443 for the ingress definition,

add the spec TLS hosts and secret names to the ingress definition,

and an annotation to the ingress definition to tell the NGINX ingress controller that the backend protocol is now HTTPS.
So we can see the stuff in that directory is handling the ingress definitions.



SAS Security Certificate Framework Init Container

A Kubernetes Pod can have

- Multiple containers running apps within it
- One or more init containers which are run before the app containers are started

Init containers are like regular containers except

- Init containers always run to completion
- Each init container must complete successfully before the next one starts

If a Pod's init container fails, it is repeatedly restarted until it succeeds

The `sas-certframe` init container

- Builds trust stores for the SAS Viya service container
- Requests certificate pairs for the SAS Viya service container

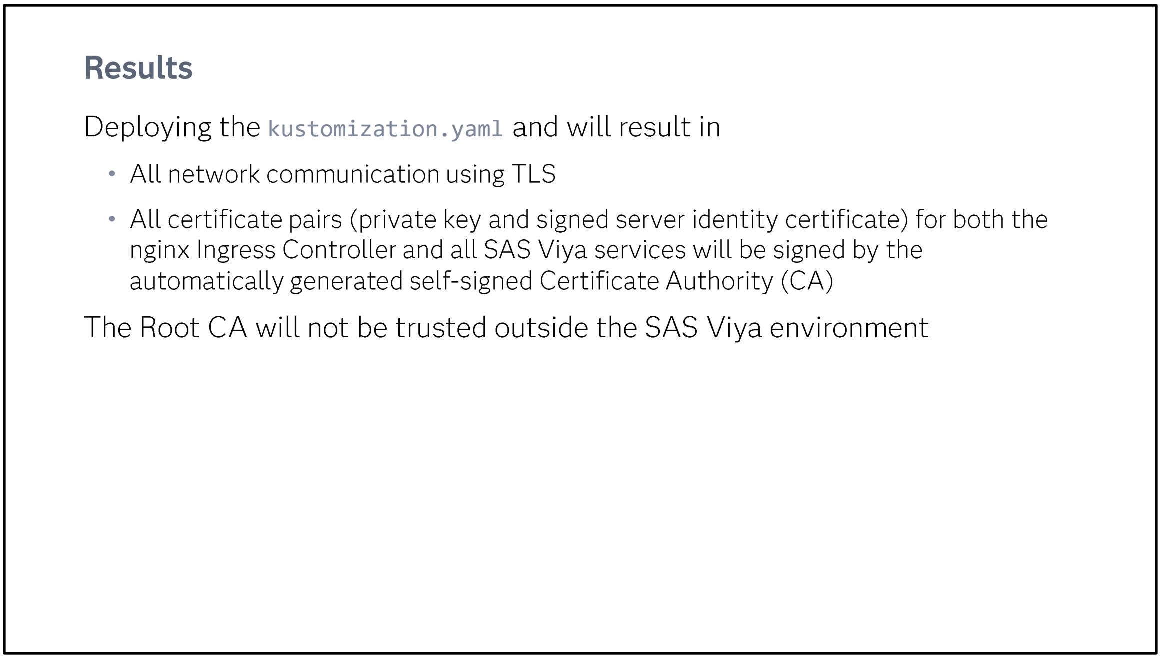
So what about that init container that gets included into our pods?

A Kubernetes pod can have multiple containers running apps within it.
It can have one or more init containers, which are run before the application containers are started.

Init containers are like regular containers, except init containers always run to completion.
Each init container must complete successfully before the next one starts.

If a pods init container fails, it is repeatedly restarted until it succeeds.

The SAS cert frame init container builds trust stores for the SAS Viya service container and requests certificate pairs for the SAS Viya service container.



Results

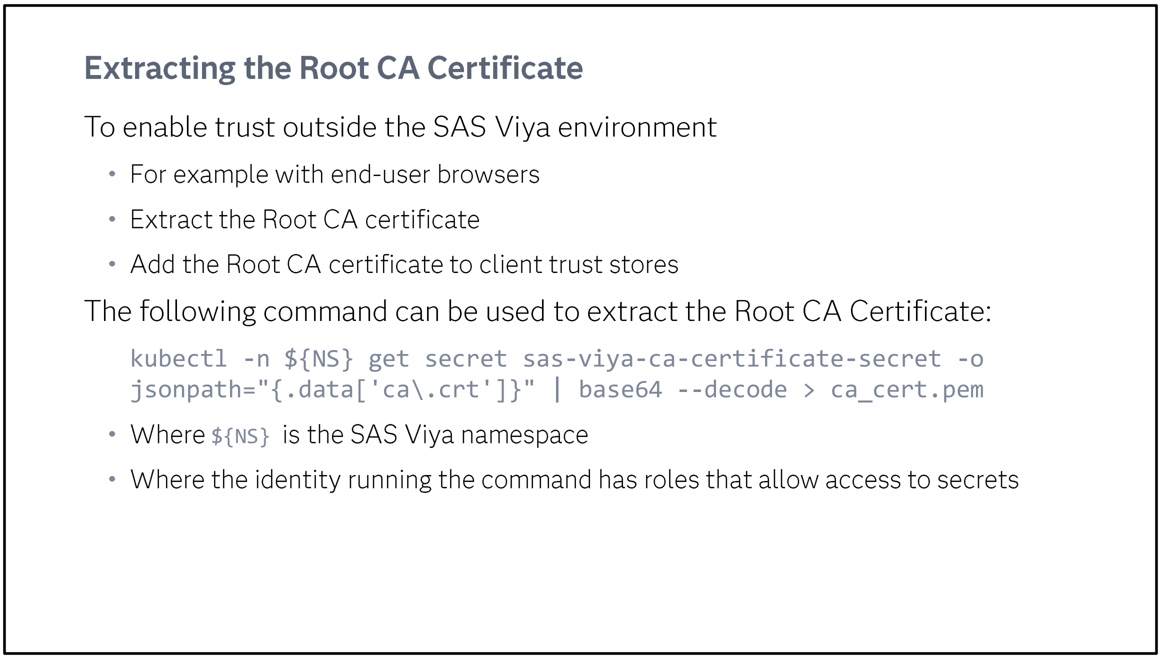
Deploying the `kustomization.yaml` and will result in

- All network communication using TLS
- All certificate pairs (private key and signed server identity certificate) for both the nginx Ingress Controller and all SAS Viya services will be signed by the automatically generated self-signed Certificate Authority (CA)

The Root CA will not be trusted outside the SAS Viya environment

Deploying the kustomization.yaml and setting the ingress controller cert manager issuer to the SAS Viya issuer will result in all network communications using transport layer security and all certificate pairs for both the ingress controller and all SAS Viya services signed by the automatically generated self-signed certificate authority.

The root CA will not be trusted outside of the SAS Viya environment.



Extracting the Root CA Certificate

To enable trust outside the SAS Viya environment

- For example with end-user browsers
- Extract the Root CA certificate
- Add the Root CA certificate to client trust stores

The following command can be used to extract the Root CA Certificate:

```
kubectl -n ${NS} get secret sas-viya-ca-certificate-secret -o
jsonpath="{.data['ca\.crt']}" | base64 --decode > ca_cert.pem
```

- Where `${NS}` is the SAS Viya namespace
- Where the identity running the command has roles that allow access to secrets

So what about being able to extract that root CA certificate?

To be able to enable trust outside of our Viya environment, for example, for end-user browsers, we need to extract the root CA certificate and then we can add that CA certificate to the client trust stores.

We can use the following command to extract that root CA certificate.
We're running kubectl against our namespace, we're doing a get of a secret, and we're outputting just the CA.crt.
All of those entries are Base64 encoded, so we need to run it through, we can see the pipe there, we need to run it through Base64 decode, and then we can redirect that final output to a file.
The identity that runs this command must have the correct roles that allows it to access secrets within the namespace.