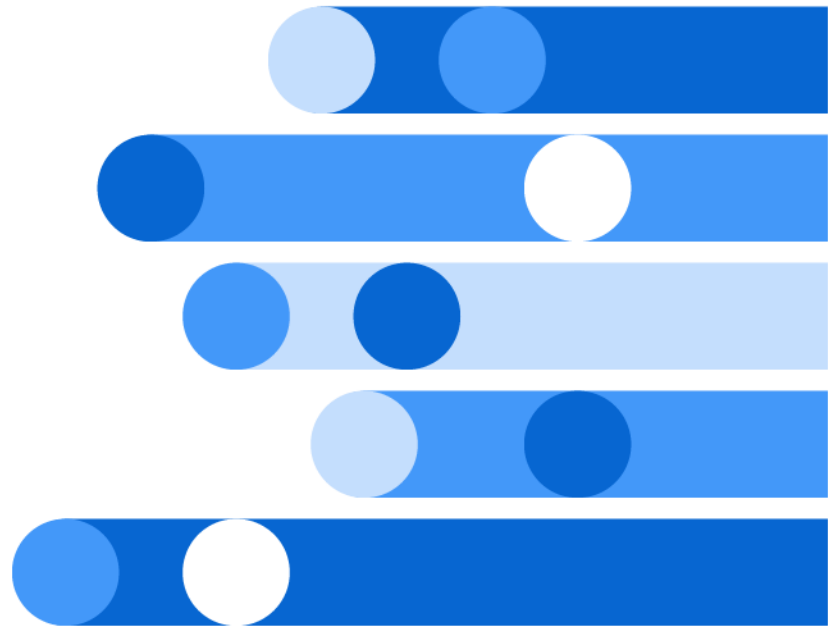




SAS[®] SpeedyStore: Administration and Configuration Guide

2026.05 - 2026.09*



* This document might apply to additional versions of the software. Open this document in [SAS Help Center](#) and click on the version in the banner to see all available versions.



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2026. *SAS® SpeedyStore: Administration and Configuration Guide*. Cary, NC: SAS Institute Inc.

SAS® SpeedyStore: Administration and Configuration Guide

Copyright © 2026, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

September 2026

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

v_002-P1:calsinglestore

Contents

Chapter 1 / Introduction	1
About SAS SpeedyStore	1
SAS SpeedyStore and SingleStoreDB Versions	2
Chapter 2 / Deployment	3
System Requirements and Deployment	3
Modifying IaC for Azure CNI	4
Using IaC with AWS, Google Cloud Platforms, or Open Source Kubernetes	5
Adding SAS SpeedyStore to an Existing SAS Viya Platform Deployment	6
Updating SAS SpeedyStore	7
Chapter 3 / Configuration and Administration	9
Deployment Architecture	10
Managing the SingleStore Cluster	13
Tuning and Optimization	16
Security	20
Logging and Monitoring	34
Backup and Restore	36
Creating the Metadata Database for the SAS Embedded Process	37
Determining the SingleStore Version	38
Character Set Considerations	38
Troubleshooting	40

Introduction

<i>About SAS SpeedyStore</i>	1
<i>SAS SpeedyStore and SingleStoreDB Versions</i>	2

About SAS SpeedyStore

SAS SpeedyStore (formerly SAS with SingleStore) offers the following benefits:

- The SAS SpeedyStore data connector provides parallel data transfer, or *streaming*, between CAS and SingleStore. Parallel data transfer can improve performance for very large tables. For information about the data connector, see [“SAS SpeedyStore Data Connector” in SAS Cloud Analytic Services: User’s Guide](#).
- By using the SAS Embedded Process, the SAS SpeedyStore data connector automatically pushes WHERE expressions and computed columns to be processed in SingleStore. Certain formatting can be performed inside the database. Performing these operations in the database reduces the amount of data sent through the network. For more information, see [“Improving Performance Using In-Database Processing” in SAS Cloud Analytic Services: User’s Guide](#).
- In-database scoring is available by using the SingleStore LIBNAME engine with the ACCELERATOR procedure. For an example, see [“Publish, Run, and Delete a Model in SingleStore” in SAS In-Database Products: User’s Guide](#). Alternatively, you can use the data connector with either the SCOREACCEL procedure or the publishing and scoring actions. For an example, see [“Publish, Run, and Delete a Model in SingleStore” in SAS In-Database Products: User’s Guide](#).
- In-database DATA step processing is available by using the SingleStore LIBNAME engine. For more information about the DATA step, see [“DATA Step Acceleration” in SAS In-Database Products: User’s Guide](#). For more information about the LIBNAME engine, see [“SAS/ACCESS Interface to SingleStore” in SAS/ACCESS for Relational Databases: Reference](#).

Note: A SingleStore standard data connector is also available. Do not use the standard data connector for a database that is integrated in SAS SpeedyStore. For

more information, see [“SingleStore Standard Data Connector” in SAS Cloud Analytic Services: User’s Guide](#).

.....

SAS SpeedyStore and SingleStoreDB Versions

Here is a list of the recent versions of SingleStoreDB that ship with SAS SpeedyStore. If you want to know the version of the database in your deployed instance of SAS SpeedyStore, see [“Determining the SingleStore Version” on page 38](#).

Table 1.1 SAS SpeedyStore and the Shipped SingleStoreDB Versions

SAS SpeedyStore	SingleStoreDB
2026.05–current	9.0
2025.03–2026.04	8.9
2024.10–2025.02	8.7
2024.02–2024.09	8.5
2023.06–2024.01	8.1

Deployment

<i>System Requirements and Deployment</i>	3
<i>Modifying IaC for Azure CNI</i>	4
<i>Using IaC with AWS, Google Cloud Platforms, or Open Source Kubernetes</i>	5
<i>Adding SAS SpeedyStore to an Existing SAS Viya Platform Deployment</i>	6
Overview	6
Adding SingleStore with a Full-System Migration on Microsoft Azure	6
<i>Updating SAS SpeedyStore</i>	7

System Requirements and Deployment

To deploy SAS SpeedyStore, use the [SAS Viya Operations](#) documentation.

You must ensure that SAS Viya platform system requirements are met. See [System Requirements for the SAS Viya Platform](#). Among the product-specific requirements, see the [requirements for SAS SpeedyStore](#).

Before deploying the software, you perform some configuration by modifying YAML files. For customizations that are specific to SingleStore, see the README for the SAS SingleStore Cluster Operator. The file is available at `$deploy/sas-bases/examples/sas-singlestore/README.md` for Markdown format or `$deploy/sas-bases/docs/sas_singlestore_cluster_operator.htm` for HTML format.

One of the tasks described in the README is how to apply the SingleStore license. When you complete an order for SAS SpeedyStore, SAS provides you with the SingleStore license key.

Modifying IaC for Azure CNI

SAS recommends using Infrastructure as Code (IaC) to provision the resources for deployment. For more information about IaC, see [“Help with Cluster Setup” in System Requirements for the SAS Viya Platform](#). The documentation includes a link to a GitHub project for Azure.

Note: IaC is also supported for SAS SpeedyStore on AWS, Google Cloud platforms, and open source Kubernetes. The instructions in this topic are not necessary for those platforms.

Here are the essential concepts:

- To specify Azure CNI, you must set the `aks_network_plugin` variable to "azure" in your Terraform variables (.tfvars) file.
- To specify Azure CNI Overlay, you must set the `aks_network_plugin` variable to "azure" and the `aks_network_plugin_mode` to "overlay". For details, see the Microsoft Azure document [Configure Azure CNI Overlay networking in Azure Kubernetes Service \(AKS\)](#).
- Currently, using Azure NetApp Files (ANF) for persistent storage volumes is not recommended. Using NFS is recommended. To specify NFS, set `storage_type="standard"` in your .tfvars file.

If you specify ANF by setting `storage_type="ha"`, then intermittent failures can occur. The main symptom is that SAS Viya platform pods are unable to mount a volume from the ANF storage (either by PVC or NFS mount). The failures occur because using ANF with the default `network_features="basic"` is not supported by Azure when the Azure VNet contains more than 1000 IPs. It is very common for the VNet to exceed 1000 IPs when you are using Azure CNI.

In order to use ANF with Azure CNI, you would need to set `storage_type="ha"` and set `network_features="standard"` instead of the default "basic." The "standard" setting can handle more than 1000 IPs. However, currently Azure has the "standard" setting in public preview, and therefore it is not supported by SAS. For more information about Network Features, see the documentation for Azure NetApp Files at <https://docs.microsoft.com/en-us/azure/azure-netapp-files/configure-network-features>. This page includes the steps to register if you want to use the "standard" setting.

- When you deploy the SAS Viya platform with Azure CNI:
 - You must specify IP ranges in the `subnets` variable in your .tfvars file in order to increase the range for the Azure Kubernetes Service (AKS) subnet (for example, /19 CIDR for 8192 IPs). The other subnets, `misc` and `netapp` (if using ANF), can stay small (for example, /24 CIDR 256 IPs). See an example below.

- SAS Viya platform pods use routable IPs from the Azure VNet subnet. You are responsible for managing the IPs that are available in the Azure VNet subnet. An understanding of networking design is strongly recommended.

In the following example, the customer overrides the default ranges to increase the number of IPs for aks. The customer also overrides the ranges for misc and netapp but does not increase the number of their IPs. As a result (not shown in the example), the IPs for aks increase from 510 to 2,046. See the subnets variable section in the examples/sample-input-singlestore.tfvars file.

Table 2.1 Example IP Ranges

Default IP Range	Specified IP Range
<pre>aks = { "prefixes": ["192.168.0.0/23"], ... misc = { "prefixes": ["192.168.2.0/24"], ... netapp = { "prefixes": ["192.168.3.0/24"], ...</pre>	<pre>aks = { "prefixes": ["192.168.0.0/21"], ... misc = { "prefixes": ["192.168.8.0/24"], ... netapp = { "prefixes": ["192.168.9.0/24"], ...</pre>

Using IaC with AWS, Google Cloud Platforms, or Open Source Kubernetes

SAS recommends using Infrastructure as Code (IaC) to provision the resources for deployment. For more information about IaC, see [“Help with Cluster Setup” in System Requirements for the SAS Viya Platform](#). The documentation includes links to the GitHub projects.

Starting with [2025.12](#), SAS SpeedyStore supports IaC on the Google Cloud platforms that are supported for SAS Viya deployment.

Starting with [2023.03](#), SAS SpeedyStore supports IaC on AWS.

Starting with [2023.01](#), SAS SpeedyStore supports IaC on open source Kubernetes.

Adding SAS SpeedyStore to an Existing SAS Viya Platform Deployment

Overview

You can deploy SAS SpeedyStore when you have an existing SAS Viya platform deployment. The instructions depend on the CNI of the existing SAS Viya platform deployment.

If you plan to deploy SAS SpeedyStore on Azure, the Azure Kubernetes Service (AKS) cluster must be set up with Azure container networking interface (CNI). SingleStore does not support using kubenet CNI on Azure. Azure CNI is required.

Here are the scenarios:

- If the existing cluster is set up with Azure CNI (Azure only), or Calico (open source Kubernetes only), then you can add SAS SpeedyStore to the existing SAS Viya platform deployment. Follow the instructions in [“Options to Deploy Software with a Modified Software Order” in SAS Viya Platform Operations: Updating Software Tasks](#).
- If the existing cluster is set up with kubenet CNI, then you must redeploy. An existing SAS Viya platform deployment does not support converting the AKS cluster from one CNI to another. You must create a new AKS cluster with Azure CNI and deploy the SAS Viya platform to the new AKS cluster.
 - If you do not want to perform a migration from your previous deployment, then follow the instructions in [SAS Viya Platform: Deployment Guide](#) to deploy SAS SpeedyStore to a new AKS cluster created with Azure CNI. See important details in [“Modifying IaC for Azure CNI” on page 4](#).
 - If you do want to perform a migration, then follow the instructions in [“Adding SingleStore with a Full-System Migration on Microsoft Azure” on page 6](#).

Adding SingleStore with a Full-System Migration on Microsoft Azure

Use these instructions if you must create a new AKS cluster with Azure CNI, and you want to perform a migration from the previous deployment. For details about the deployment scenarios, see [“Overview” on page 6](#).

- 1 Back up your source SAS Viya platform environment. Follow the instructions in [“Step 2: Obtain BackUp \(Cluster Administrator\)” in SAS Viya Platform: System Migration](#).
- 2 Redeploy the SAS Viya platform. SAS recommends that you redeploy from the software order of the environment that you backed up. In other words, deploy the same version and the same cadence as the source environment. However, if you are deploying on Azure, you must use the Azure CNI. See important details in [“Modifying IaC for Azure CNI” on page 4](#). Follow the instructions in [“Restore to a New Environment” in SAS Viya Platform: System Migration](#).
- 3 Restore the backup in the SAS Viya platform. The instructions are included in [“Restore to a New Environment” in SAS Viya Platform: System Migration](#).
- 4 Use the new software order to update to SAS SpeedyStore. Follow the instructions in [“Option 3 – Modify Software for an Existing Deployment and Update to a New Version” in SAS Viya Platform Operations: Updating Software Tasks](#).
- 5 Validate the new deployment. Follow the instructions in [“Step 4: Validate” in SAS Viya Platform: System Migration](#).

Updating SAS SpeedyStore

SAS recommends updating to a new version of SAS SpeedyStore and applying patch updates as they become available. The latest software can include new features, enhancements to features, bug fixes, and security patches. For more information, see [SAS Viya Platform Operations: Updating Software Tasks](#).

You must pause the SingleStore cluster before you update SAS SpeedyStore or before you apply a patch update. As of [2025.01](#), when you update by using either the SAS Viya Platform Deployment Operator or the `sas-orchestration` command, the SingleStore cluster is paused automatically. If you update to a release prior to [2025.01](#), or if you update by using Kubernetes commands, then you must manually pause the SingleStore cluster before you update. For the details and instructions, see [“Pausing and Resuming SingleStore” on page 14](#).

If an update fails before it completes, SingleStore is left in a paused state. For information about resuming the SingleStore cluster, see [“Pausing and Resuming SingleStore” on page 14](#). For information about resolving the update failure, see the following troubleshooting topics:

- [“Troubleshooting – Updating Software” in SAS Viya Platform Operations: Updating Software Tasks](#)
- [“Troubleshooting” in SAS Viya Platform: Deployment Guide](#)

Each release of the SAS Viya platform uses a [particular version of SingleStoreDB](#). When you update to a more recent release of the SAS Viya platform, the update

might include a new version of SingleStore. Do not manually change the version of SingleStore.

IMPORTANT In the following releases, the SingleStoreDB instance that is included with SAS SpeedyStore is automatically updated. The update is not backward compatible. Following the update, you cannot roll back your SingleStoreDB databases to an earlier version. If you need to use databases from an earlier release, you must uninstall the SAS Viya platform and redeploy the earlier version. SAS recommends creating a backup of the database before starting the SAS Viya platform update.

- In 2026.05, the included instance of SingleStoreDB is updated from version 8.9 to 9.0.
- In 2025.03, the included instance of SingleStoreDB is updated from version 8.7 to 8.9.
- In 2024.10, the included instance of SingleStoreDB is updated from version 8.5 to 8.7.
- In 2024.02, the included instance of SingleStoreDB is updated from version 8.1 to 8.5.
- In 2023.06, the included instance of SingleStoreDB is updated from version 7.8 to 8.1.

IMPORTANT

Beginning with the 2023.06 release, the SingleStore `data_conversion_compatibility_level` has changed to 8.0. This change affects the behavior when multibyte character data exceeds the defined length and is truncated. Partial characters are now rejected. To avoid this error, ensure that the column lengths are large enough to prevent truncation.

Configuration and Administration

Deployment Architecture	10
Architecture Diagram	10
Viewing Components in Your Deployment	11
Managing the SingleStore Cluster	13
Overview	13
Pausing and Resuming SingleStore	14
Creating Database Snapshots	15
Tuning and Optimization	16
Resource Considerations	16
Setting max_connection_threads	17
Optimizing Performance for Large Data	18
Optimizing Performance When Loading SAS Data to SingleStore	19
Optimizing the Query Plancache	20
Conserving Memory	20
Security	20
Overview	20
Enabling Single Sign-On for Microsoft Azure	21
Logging and Monitoring	34
Overview	34
Monitoring with SingleStore Grafana Dashboards	34
SAS Viya Monitoring for Kubernetes	35
Accessing SAS Embedded Process Logs in SingleStore	36
Backup and Restore	36
Creating the Metadata Database for the SAS Embedded Process	37
Determining the SingleStore Version	38
Character Set Considerations	38
Support for 4-Byte Characters	38
Setting or Modifying the Collation	39
Troubleshooting	40
Leaf Node Container Repeatedly Restarts Before It Can Finish	
Replaying Transaction Logs	40
SingleStore Pod "FailedScheduling" Event Due to Insufficient CPU or Memory	41
Memory Has Reached the maximum_memory Setting	45
Connection Time-Out Errors	46

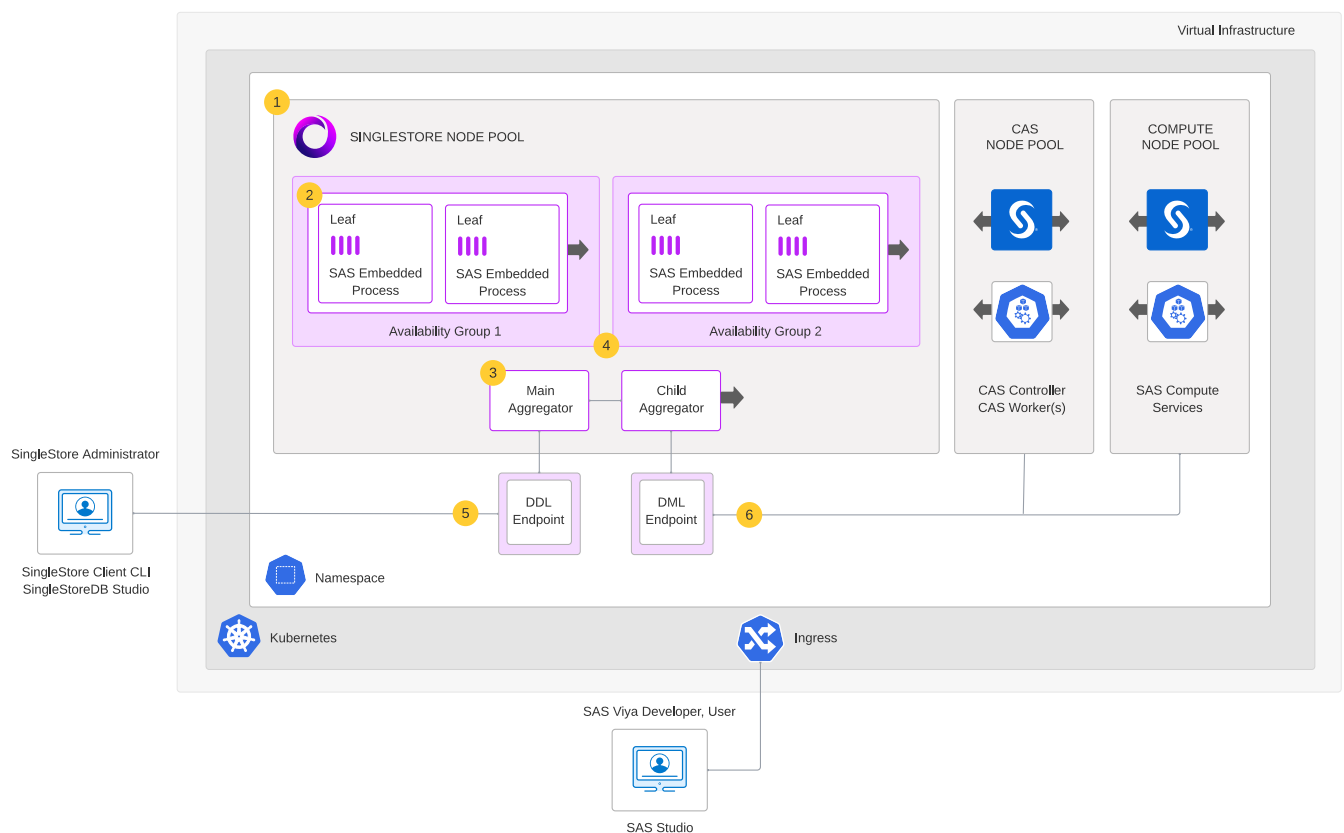
sas-singlestore-sync: Time-Out Error While Closing Your mysqlconnector Connection	46
Operation ALTER Time-Out Error	47

Deployment Architecture

Architecture Diagram

The following diagram illustrates the main components that are included in a SAS SpeedyStore deployment. Not all components and options are depicted. An actual deployment has more node pools than are depicted in this diagram. Node pools are dedicated to a workload class, which is identified by the work that is performed by the associated pods. See [“Plan the Workload Placement” in SAS Viya Platform: Deployment Guide](#) for information about labeling and tainting your nodes.

For a reference diagram of a SAS Viya platform deployment, see [“Reference Diagram” in SAS Viya Platform: Overview](#). SAS SpeedyStore can be deployed on Azure, AWS, Google Cloud platforms, and upstream open source Kubernetes. (For the details about supported platforms, see [System Requirements for the SAS Viya Platform](#).)



- 1 A SAS SpeedyStore deployment includes a SingleStore cluster within the SAS Viya namespace. The SingleStore cluster consists of leaf pods and aggregator pods.
- 2 Leaves store slices of data called *partitions*, and these partitions are distributed across the leaf pods. Leaves also function as compute nodes, and the SAS Embedded Process is deployed to each leaf pod. SAS Embedded Process is automatically invoked to support computed columns and WHERE clause filtering as needed. It has the potential to greatly reduce data movement when filtering on WHERE clause components that are specific to the SAS Viya platform. For more information, see [In-Database Processing](#).
- 3 Aggregators include one main aggregator (MA) pod and one or more child aggregators (CA). The MA pod is responsible for data definition language execution, cluster monitoring, and failover. CA pods run data manipulation commands on leaf pods, as well as read metadata from the MA pod.
- 4 High Availability (HA) is enabled through SingleStore availability groups. The HA option doubles the number of predefined leaf pods and creates a second Availability Group (AG).

VMs that host the leaf pods must be allocated in pairs. Each of the pods is distributed to one of the availability groups (AG1 or AG2). High availability is achieved by having two copies of a partition in a primary-secondary, active-passive pairing across multiple VMs. The primary partitions are spread across all leaf pods in both availability groups, AG1 and AG2, in the SingleStore cluster. In HA mode, leaf pods and VM hosts must be managed in pairs.
- 5 SingleStore administrators can manage the SingleStore cluster through the SingleStore command-line interface (CLI) or SingleStoreDB Studio. For more information, see [“Managing the SingleStore Cluster” on page 13](#).

The clients connect to the MA pod through the DDL endpoint, a load balancer that is deployed to the cluster as a Kubernetes service. DDL operations include create, drop, or alter table commands, for example.
- 6 SAS Viya clients connect through the DML endpoint from CAS worker sessions or the SAS Compute Server’s SingleStore LIBNAME engine. Child aggregators are reached through this DML endpoint and route queries to the MA for DDL or leaves for DML, then aggregate intermediate results and send the results back to CAS.

Viewing Components in Your Deployment

Run the following commands to view the architecture diagram components:

- 1 The commands use an environment variable for the SAS Viya namespace option. Run the following command to create this variable:

`viya=name_of_your_viya_namespace`
- 2 To view SingleStore pods in your environment, run the following command:

```
kubectl -n $viya get pods | egrep 'singlestore|NAME'
```

To get more information, run the following command:

```
kubectl -n $viya get pods -o wide | egrep 'singlestore|NAME'
```

The following example output shows:

- the SingleStore operator. The operator runs on a SERVICES node, as it is classed as a stateless service.
- the main aggregator pod (node-sas-singlestore-cluster-master-0).
- the child aggregator pod (node-sas-singlestore-cluster-aggregator-0).
- four leaf pods (node-sas-singlestore-cluster-leaf-ag1-0, -ag1 -1, -ag2 -0, -ag2 -1).
- six daemonsets. Daemonsets run on each node in your SingleStore cluster.

NAME	READY	STATUS	RESTARTS	AGE
node-sas-singlestore-cluster-aggregator-0	2/2	Running	0	4h16m
node-sas-singlestore-cluster-leaf-ag1-0	2/2	Running	0	4h11m
node-sas-singlestore-cluster-leaf-ag1-1	2/2	Running	0	4h16m
node-sas-singlestore-cluster-leaf-ag2-0	2/2	Running	0	4h10m
node-sas-singlestore-cluster-leaf-ag2-1	2/2	Running	0	4h16m
node-sas-singlestore-cluster-master-0	2/2	Running	0	4h16m
sas-singlestore-operator-d6cc4bbbd-57gjb	1/1	Running	0	4h16m
sas-singlestore-osconfig-8xh9l	1/1	Running	0	4h16m
sas-singlestore-osconfig-b5vzt	1/1	Running	0	4h14m
sas-singlestore-osconfig-brxdv	1/1	Running	0	4h16m
sas-singlestore-osconfig-qc9nl	1/1	Running	0	4h14m
sas-singlestore-osconfig-qtmfx	1/1	Running	0	4h15m
sas-singlestore-osconfig-rnwcb	1/1	Running	0	4h14m

- 3 To view daemonsets in your environment, run the following command:

```
kubectl -n $viya get daemonsets.apps
```

- 4 To view more information about a daemonset pod, run the following command. (Use a pod name from the Step 2 output.)

```
kubectl -n viya describe pod sas-singlestore-osconfig-xxxxx
```

The following command finds the pod name and adds the name to an environment variable. Then the environment variable is used in the subsequent command:

```
# Get the pod name
pod_name=$(kubectl -n $viya get pods | grep sas-singlestore-osconfig |
awk '{ print $1 }' | sed 1q)
# Run the describe command
kubectl -n $viya describe pod ${pod_name}
```

- 5 Clients connect through the following two load balancers that are deployed to the cluster as Kubernetes services:

- Data Definition Language (DDL) endpoint (svc-sas-singlestore-cluster-ddl)
- Data Manipulation Language (DML) endpoint (svc-sas-singlestore-cluster-dml)

To get information about these services in your cluster, run the following command:

```
kubectl -n $viya get svc | egrep 'NAME|singlestore'
```

In the following example output, the SingleStore DDL and DML services are available on port 3306 (unsecure) and port 443 (secure).

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
sas-singlestore-operator	ClusterIP	10.0.203.15	<none>	9090/TCP
svc-sas-singlestore-cluster	ClusterIP	None	<none>	3306/TCP
svc-sas-singlestore-cluster-ddl	LoadBalancer	10.0.219.243	20.241.200.211	3306:31169/TCP,443:30405/TCP
svc-sas-singlestore-cluster-dml	LoadBalancer	10.0.138.45	20.241.203.17	3306:30307/TCP,443:31274/TCP

- To view the memsqlcluster custom resource definition (CRD), run the following command:

```
kubectl -n $viya get memsqlclusters.memsql.com
```

The following output displays the Redundancy Level as 2, which signifies that this environment has HA.

NAME	AGGREGATORS	LEAVES	REDUNDANCY LEVEL	AGE
sas-singlestore-cluster	2	2	2	23h

For more details about your SingleStore cluster, run the following command:

```
kubectl -n $viya describe memsqlclusters.memsql.com
```

Managing the SingleStore Cluster

Overview

You can use the following methods to manage your SingleStore cluster:

SingleStore command-line interface (CLI)

The SingleStore CLI enables you to run SQL queries against your SingleStore cluster and is available inside the master aggregator pod by default.

You can also download the SingleStore CLI outside the cluster. This CLI is known as the `singlestore-client` package. For information, see [SingleStore Client](#).

MySQL-compatible Tools

Connect to your SingleStore deployment from MySQL-compatible tools. For information, see [Connect to SingleStore](#).

Note: SingleStore has announced that they are deprecating SingleStore DB Studio. SingleStore recommends exploring these supported third-party tools, which have been tested by SingleStore and might be familiar to developers.

SAS Data and AI Studio (known as SAS Studio prior to 2026.06)

You can use SAS Data and AI Studio as your SQL editor. Here is an example of the connection information that is needed:

```
/* Connection details for SingleStore */
%let MYUID='admin_user';
%let MYPWD='password';
%let MYDB='db_name';
%let MYPORT='3306';
%let MYHOST='svc-sas-singlestore-cluster-dml';
```

You can get the DDL and DML service endpoints for SingleStore with the following command:

```
kubectl -n viya_ns get svc | egrep 'singlestore|NAME'
```

Pausing and Resuming SingleStore

If you use Kubernetes commands to update SAS SpeedyStore to a newer version, or to apply a patch update, you must pause the SingleStore cluster prior to update. If you use either the SAS Viya Platform Deployment Operator or the `sas-orchestration` command for updating, then as of [2025.01](#), those methods pause the SingleStore cluster automatically.

Similarly, if you use CronJobs to start and stop a SAS Viya Platform deployment, as of [2025.01](#) the SingleStore pods are also managed by the CronJobs. When you execute the `sas-stop-all` CronJob, it pauses SingleStore. When you execute the `sas-start-all` CronJob, it restarts SingleStore. These CronJob tasks are described in [“Managing a SAS Viya Platform Deployment” in SAS Viya Platform: General Management of Servers and Services](#). Prior to [2025.01](#), you are required to pause SingleStore manually before you execute the `sas-stop-all` CronJob.

Manually pausing SingleStore is recommended before you uninstall SAS SpeedyStore.

If you need to manually pause SingleStore, the best method is to set the `clusterState` property to `pause` in the `MemsqlCluster` resource. Here is a sample portion of the YAML file where the property is set:

```
apiVersion: memsql.com/v1alpha1
kind: MemsqlCluster
metadata:
  name: memsql-cluster
spec:
  clusterState: pause
```

For example, here is a patch command to pause the cluster:

```
kubectl -n name-of-namespace patch MemsqlCluster \
  sas-singlestore-cluster --type=json -p \
  '[{"op": "add", "path": "/spec/clusterState", "value": "pause"}]'
```

When you pause the cluster, either manually or automatically, snapshots are taken of all the databases. The snapshot process can lengthen the shutdown time,

especially if a large amount of new data requires a snapshot. To check the status of a cluster pod, submit the following command:

```
kubectl get pods node-sas-singlestore-cluster-master-0 \
-n name-of-namespace
```

If the command returns the status of the pod, then the pod is not terminated. If the command returns an error message that the pod is **not found**, the error indicates the pod is terminated.

If you are updating by using Kubernetes commands, you can proceed with the SAS Viya platform update after all of the pods terminate.

If you are updating by using Kubernetes commands, after the update you can resume the SingleStore cluster by deleting `spec:clusterState:pause` from the YAML file. For example, here is a patch command to delete the property:

```
kubectl -n name-of-namespace patch MysqlCluster \
sas-singlestore-cluster --type=json -p \
'[{ "op": "remove", "path": "/spec/clusterState" } ]'
```

Note: Manually pausing SingleStore is the recommended method for shutting down and starting up SingleStore nodes. SingleStore nodes are removed after the snapshots are taken and they are started up again when you take the SingleStore cluster out of the pause state.

If you need to stop SingleStore nodes to perform system maintenance, follow these instructions: [Taking Leaves Offline with Cluster Downtime](#). To delete the SingleStore cluster or to delete the SingleStore Operator, follow these instructions: [Delete the Cluster or Operator](#).

Creating Database Snapshots

Creating database snapshots can improve restart time and stability of SingleStore databases. For example, you might be able to avoid the error that is described in [“Leaf Node Container Repeatedly Restarts Before It Can Finish Replaying Transaction Logs” on page 40](#). Here are ways to create database snapshots:

- You can create your own snapshot by using the `SNAPSHOT DATABASE` command followed by the `_SYNC_SNAPSHOT` command. Follow these best practices:
 - Create a snapshot after you load data into a database.
 - Create a snapshot of each database prior to restarting SingleStore.
- SingleStore creates a snapshot automatically when the log size reaches the limit of the `snapshot_trigger_size` parameter. If the log size is not appropriate, edit the `sas-singlestore-cluster-config.yaml` file and adjust the `snapshot_trigger_size` parameter.
- The SingleStore operator creates a database snapshot before certain operations such as deleting a node.

For more information about database snapshots, refer to the [SingleStore documentation website](#).

Here is an example command that uses the SingleStoreDB Toolbox utility to run the snapshot queries:

```

NAMESPACE=viya
S2SVCEXTIPDDL=$(kubectl -n $NAMESPACE get service svc-sas-singlestore-
cluster-ddl|grep
-v NAME|awk '{print $4}')
MYS2DB=viya
MYS2USER=root
export MYSQL_PWD=$(kubectl -n $NAMESPACE get secret sas-singlestore-
cluster
-o yaml|grep "^ ROOT_PASSWORD"|awk '{print $2}'|base64 -d --wrap=0)
singlestore -h $S2SVCEXTIPDDL -P 3306 -u $MYS2USER -e "SNAPSHOT
DATABASE viya;
_SYNC_SNAPSHOT $MYS2DB;"

```

Tuning and Optimization

Resource Considerations

SingleStore partitions act as independent databases. As a best practice, provide uniform resources for each partition. The number of vCPUs in the SingleStore cluster leaf pods should be evenly divisible by the number of partitions, and the number of partitions should be evenly divisible by the number of leaf pods.

When selecting virtual machine types, consider that SingleStore recommends uniform resourcing and at least 30 MB of RAM per physical CPU core or vCPU. In Microsoft Azure, for example, the VMs within the Es-series can have different types of processor, or processors that span multiple generations, and thus they do not conform to the uniform-resourcing requirement.

When provisioning CPU and memory resources, take the following factors into consideration:

- CPU resources are required to support connections from CAS to SingleStore CAs. SAS recommends:
 - having at least 1 CA vCPU to every 2 CAS worker vCPUs.
 - limiting CAS worker vCPUs to no more than 3 per SingleStore leaf pod vCPU.
- RAM resources are required to enable SingleStore leaf nodes to support table metadata in memory. SAS recommends:
 - having at 8 least GiB of RAM per vCPU.

Note: Memory is also needed for uncompressing of columnstore data from storage to satisfy queries and to support the SAS Embedded Process.

- ❑ allocating 30 MB/s of I/O bandwidth and 300 IOPS per vCPU in each leaf pod.

SAS SpeedyStore resource requirements also depend on the following factors in your environment:

- the expected number of simultaneous CAS user sessions.
- the type of CAS workload. Solutions can start multiple CAS sessions per user.
- the number of CAS workers (with MPP CAS).
- the number of SingleStore child aggregators (CAs).
- the number of partitions per leaf node.

SAS recommends having multiple partitions per leaf node so that you can more easily add resources if necessary.

- the number of vCPUs per partition.

Either 4 vCPUs or 8 vCPUs per partition is recommended.

- the SingleStore HA policy that you have applied.
- VNet configuration.

SAS Viya platform and SingleStore VMs should be running within a single availability zone or proximity group in order to minimize latency. Be aware that stopping and restarting the deployment might cause the movement of some nodes into different proximity groups or availability zones.

- the compression ratio of columnstore tables.
- database or database table options.

The SingleStore Support Help Center includes guidance about provisioning your environment for the expected number of database partitions. For more information, see [Number of database partitions required](#).

For more SingleStore-specific guidance, see the "System Requirements and Recommendations" section of the SingleStore [documentation website](#).

Setting max_connection_threads

Increasing the max_connection_threads can help prevent issues like the following log message from the aggregator node container:

```
WARN: The ready queue has not decreased (currently 502 elements, 13110 pops)
for 18 seconds. This workload needs more threads.
```

The maximum value is 8192 (per aggregator). SAS recommends a value of at least 4096 (4 KB), but 8192 is preferred.

The following command changes the max_connection_threads variable to 4096 for each aggregator:

```
kubectl -n name-of-namespace patch memsqlclusters.memsql.com sas-
singlestore-cluster --type=json -p '[{"op": "replace", "path": "/spec/
aggregatorSpec/globalVariables/max_connection_threads", "value":
"4096"}]'
```

Note that when you increase `max_connection_threads`, you also increase utilization of the aggregator container CPU. You might experience a counter-productive scenario where too many connection threads are active. A high number of threads can cause CPU context switching, also known as CPU thrashing, which reduces performance.

Adding more aggregators (which results in more total connection threads) can be helpful if your workload suffers from excessive CPU context switching. For example, the following command sets the count at 5 aggregators:

```
kubectl -n name-of-namespace patch memsqlclusters.memsql.com sas-
singlestore-cluster --type=json -p '[{"op": "replace", "path": "/spec/
aggregatorSpec/count", "value": 5}]'
```

Optimizing Performance for Large Data

The leaf pod height and the number of partitions per leaf pod determine the number of columns that the database can support. These guidelines assume that tables have sufficient rows such that all partitions have a significant row count per table.

SAS testing configured the SingleStore node pool with the following resources. Testing confirmed that, with these specifications, the database can support approximately 1,000,000 columns for statistical analysis.

- Leaf pod height of 1
- Overpack factor of 0.15
- vCPU per partition ratio of 4:1, which implies two partitions per leaf pod (8 vCPU)
- One leaf pod per VM

If you need to support more columns, choose one of the following methods. Make either of these changes in order to double the number of columns that the database can support:

- Increase the number leaf pods to provide one partition per leaf pod (keeping the VM size as is).
- Increase the height to 2, with two partitions per leaf (which requires a larger VM). For example, SAS testing was performed on an Azure E-series VM with 8 vCPUs. Increasing the height to 2 would require an upgrade to 16 vCPUs.

Optimizing Performance When Loading SAS Data to SingleStore

The following guidelines can improve performance during the initial load of large tables (greater than 2 terabytes) to SingleStore. This list assumes that you are loading SAS data sets (SAS7DBAT files) or CAS tables (SASHDAT files) from a storage source to a SingleStore database. This list includes both usage and configuration recommendations.

- Save one table at a time.
- Use a SingleStore caslib and the table.save action. For SAS7DBAT files, specify the action parameter singlePass=true. For SASHDAT files, specify singlePass=false. Here is an example:

```
proc cas;
  table.save /
    table={
      caslib="source",
      name="mydata.sashdat" ,
      singlePass=true
    }
    caslib="s2caslib" ,
    name="mydata",
    exportOptions={fileType="AUTO"},
    replace=TRUE;
quit;
```

- Aim for a 1:4 ratio of database partitions to total leaf vCPU. In SAS testing, a 1:4 ratio shows better performance than a 1:1 ratio, which is the default if you do not specify the number of partitions to create.

Note: *Total leaf vCPU* is relevant for SingleStore with High Availability (HA), which is enabled by default. In this case, there are two availability groups. Each availability group has *leafCount* number of leaves. In the ratio, use the sum of vCPU across both availability groups.

- You would probably not benefit from increasing the size of your CAS VM. For example, an Azure E4ds_v4 VM is sufficient, and the more expensive E64ds_v4 VM does not improve performance.
- Performance improves as you increase CAS worker count up 8 workers, or a number of workers equal to the SingleStore leafCount. However, performance can decline if CAS worker count increases above that number.

Optimizing the Query Plancache

The default SingleStore plancache settings assume a significant reuse of queries. However, queries in the SAS Viya platform often use WHERE clause filters that are automatically pushed to the SAS Embedded Process. These queries are not reusable.

The plancache consumes memory and requires CPU cycles to manage. Most customers can benefit from reducing the following SingleStore global variables. Do not set the values to zero.

- `plan_expiration_minutes`
- `disk_plan_expiration_minutes`

For more information about the variables, see the [SingleStore documentation website](#).

Conserving Memory

Many customers limit the use of cloud-based storage in order to reduce spending. Here are some tips:

- Use `VARCHAR(n)` instead of `CHAR(n)`. A `CHAR(n)` column uses $4 * n$ bytes of storage because its charset is `utfmb4`.
- For data that is dominated by floating-point DOUBLES, compression is not as effective as with other data types. If floating-point data is accurate to *X* decimal places, consider using a scale factor of 10^X and saving the integer value in the DOUBLE.

Only fractions that are a sum of powers of 2^{-N} are exactly representable and possibly compressible. For example, for doubles that have 4 decimal places of accuracy, you can scale to integer values in the following way: `(int(value*104))`.

Security

Overview

Authentication to SingleStore requires external credentials. The SAS SpeedyStore data connector supports using the `authenticationDomain=` option to provide the

credentials. Alternatively, you can specify `userName=` and `password=` options. For information about authentication domains, see [SAS Viya Platform: External Credentials](#). For information about the data connector, see “SAS SpeedyStore Data Connector” in *SAS Cloud Analytic Services: User's Guide*.

Data in motion between CAS and SingleStore is encrypted as specified in the TLS mode that you set for the SAS Viya platform. The instructions are included in [SAS Viya Platform: Deployment Guide](#). SAS recommends full-stack TLS. You can specify the path to a different certificate authority file by setting the `ssl_ca=` option in the SAS SpeedyStore data connector. For more information about encryption, see [SAS Viya Platform Encryption: Data in Motion](#).

In addition, in the SingleStore data source, it is strongly recommended to grant `REQUIRE SSL` for all users. Follow the instructions in [Server Configuration to Require Secure Client Connections](#) on the SingleStore documentation website.

IMPORTANT In the base `kustomization.yaml` (`$deploy/kustomization.yaml`) file, ensure that the `sas-singlestore` component is added before any TLS components.

Using SingleStore with CAS involves multiple layers of data authorization:

- Access to data in SingleStore requires at least the `SHOW METADATA` permission. For more information about role-based access control in SingleStore, refer to the [SingleStore documentation website](#).
- Access to data in SingleStore also requires permissions to the metadata database that is used by the SAS Embedded Process. For more information, see “Creating the Metadata Database for the SAS Embedded Process” on page 37.
- Access to in-memory data in CAS is protected by CAS access controls. For more information, see [SAS Viya Platform: CAS Authorization](#).

Enabling Single Sign-On for Microsoft Azure

Overview of Steps to Enable Single Sign-On

SAS SpeedyStore uses Microsoft Entra ID (formerly Azure Active Directory) to provide single sign-on authentication support. To authenticate with SingleStore, CAS obtains a Microsoft Entra ID JWT (JSON web token) and passes it to SingleStore. SingleStore is already configured to use JWTs for authentication.

With SSO configured, the user does not need to specify `userName=`, `password=`, or `authenticationDomain=` options to define or access a SingleStore caslib.

Perform the following tasks to enable single sign-on for SAS SpeedyStore:

- 1 “Configure Single Sign-on for the SAS Viya Platform” on page 22

- 2 [“Set Up JWT Authentication for SingleStore Users” on page 22](#)
- 3 Choose either LDAP or SCIM for identity management. (For more information about the two choices, see [“Identity Management: Overview” in SAS Viya Platform: Identity Management](#).) SCIM is supported starting in 2025.02.
 - [“Use the LDAP Tool to Add Users and Groups to SingleStore” on page 24](#)
 - [“Enable and Configure SCIM” on page 28](#)

Configure Single Sign-on for the SAS Viya Platform

Configure Microsoft Entra ID as an OpenID Connect (OIDC) provider. For instructions, see [“Scenario: OIDC with Microsoft Entra ID” in SAS Viya Platform: Authentication](#).

Note: As you follow the instructions at the link above, these two tasks are specific to SAS SpeedyStore:

- When you perform [Step 10](#) of the instructions to configure single sign-on, copy the URL that appears in the **OpenID Connect** metadata document field and open it in a new browser tab. Retrieve the value for `jwtks_uri`. In the next topic, [“Set Up JWT Authentication for SingleStore Users” on page 22](#), you specify that value in the `jwtks_endpoint=` variable. The value is in a form similar to the following URL:


```
https://login.microsoftonline.com/entra_tenant_id/discovery/v2.0/keys
```
 - When you [configure OIDC provider properties for Microsoft Entra ID](#) for **sas.logon.oauth.providers**, you must include `profile` in the list of values for the **scopes** configuration field. This value ensures the use of user principal name (UPN) for the user name claim. In the next topic, [“Set Up JWT Authentication for SingleStore Users” on page 22](#), you specify `upn` in the `jwtks_username_field=` variable.
-

Set Up JWT Authentication for SingleStore Users

A JWT user is a user who can authenticate with a JWT when connecting to a SingleStore cluster through CAS and the computer server. Clusters are configured to accept JWTs using JSON Web Key Sets (JWKS). With JWKS, JWTs are matched to JSON Web Keys (JWKs) to validate the JWT. For more information, see [Authenticate via JWT](#).

Configure two [engine variables](#) in the SingleStore cluster.

- The `jwtks_username_field` variable specifies the name of the JWT field that determines the database user name. The User Principal Name (UPN) attribute is used for this purpose. Run the following command to set the variable to UPN:

```
kubectl -n namespace patch memsqlclusters.memsql.com sas-singlestore-
cluster
--type=json -p "[{"op": "add", "path": "/spec/globalVariables/
jwks_username_field", "value": "upn"}]"
```

- The `jwks_endpoint` variable contains the endpoint that is used to retrieve JWTs from Microsoft Entra ID.

- Retrieve the `jwks_uri` from your Azure portal. Click **Overview**. On the App registrations Overview page, click **Endpoints**. Open the OpenID Connect metadata document. Retrieve the `jwks_uri` and replace `"https://login.microsoftonline.com/entra_tenant_id/discovery/v2.0/keys"` with your value in the `kubectl` command below.

- Run the following command to add the `jwks_uri` to the `jwks_endpoint` variable:

```
kubectl -n namespace patch memsqlclusters.memsql.com sas-
singlestore-cluster
--type=json -p "[{"op": "add", "path": "/spec/
globalVariables/jwks_endpoint", "value":
"https://login.microsoftonline.com/entra_tenant_id/discovery/v2.0/
keys"}]"
```

Note: Run the `kubectl` commands only once. If you need to update the variables, use `replace` instead of `add` in the above commands.

To review SingleStore settings for JWKS endpoint and JWKS user name, use the SingleStore CLI within the SingleStore cluster:

- show global variables like `'jwks_username%'`;

Output:

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| jwks_username_field | upn |
+-----+-----+
```

- show global variables like `'jwks_endpoint'`;

Output:

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| jwks_endpoint | https://login.microsoftonline.com/entra_tenant_id/discovery/v2.0/keys |
+-----+-----+
```

Note: An alternative to running the `kubectl` command is to create and apply an overlay file that contains the engine variables' values. Here is an example:

- 1 Create a file called `transformer-jwt-azure-ad.yaml` that contains the following code:

```
---
apiVersion: builtin
kind: PatchTransformer
metadata:
  name: patch-transformer-jwt-s2-azure
```

```

patch: |-
  - op: add
    path: /spec/globalVariables/jwks_endpoint
    value: "https://login.microsoftonline.com/entra_tenant_id/discovery/v2.0/keys"
  - op: add
    path: /spec/globalVariables/jwks_username_field
    value: "upn"
target:
  kind: MemsqCluster

```

- 2 Save the file in your `$deploy/site-config` directory.
- 3 Add a reference to the file in your base `kustomization.yaml` file.
- 4 Redeploy using the method of your choice. See [“Deploy the Software” in SAS Viya Platform: Deployment Guide](#).

Use the LDAP Tool to Add Users and Groups to SingleStore

Note: You can use either LDAP or SCIM for identity management. You cannot use both.

Use the LDAP tool (`sdb-admin sync-ldap`) to import users and groups from Lightweight Directory Access Protocol (LDAP) implementations into SingleStore. The LDAP tool provides a one-way user and group synchronization from any LDAP directory to SingleStore. The SingleStore user authenticates with a JSON Web Token (JWT), which is the authentication protocol that is configured in SingleStore.

Perform the following steps to import users and groups to SingleStore. These steps use the SingleStore documentation that is located here: [Sync LDAP JWT Users](#). Substitute your values for any values that are in italics.

- 1 Verify that the necessary tools are installed and running with the latest versions. For example: [sdb-toolbox](#), [SingleStore Client](#), and [Connect with MySQL](#).
 - You might need to update the version of the SingleStore toolbox in order to recognize the `--uris` flag of the LDAP tool.


```

sudo yum-config-manager --add-repo https://release.memsql.com/production/rpm/x86_64/repodata/memsql.repo
sudo yum repolist
sudo yum update -y singlestoredb-toolbox
sdb-admin version
          
```
- 2 The SingleStore user who runs the LDAP tool must be authorized to obtain user and group information from the LDAP directory server. Verify that these authorization prerequisites are met:

- A user can connect to the cluster using the user and password parameters.
- If a user or password is not specified, Toolbox assumes the database `root` user and uses a secure, hashed password. If a non-root user is specified and a password is not set, Toolbox prompts for a password; an error is thrown if using the LDAP tool in non-interactive mode.
- To keep your SingleStoreDB user password secure, set the `MEMSQL_PASSWORD` environment variable in the Linux shell.

3 Check connectivity to SingleStore from outside of the cluster:

- a Determine the external SingleStore host name and port. Here is an example that obtains the host and port with `kubectl` commands and assigns them to environment variables. The variables are used in the next step.

```
export KUBECONFIG=location_of_kubeconfig_file
export SSHOST_EXTERNAL=`kubectl -n namespace get pods -o wide |
grep -i master | awk '{print $7}'`
export SSPORT_EXTERNAL=`kubectl -n namespace get svc | grep
singlestore-cluster-ddl | cut -d':' -f2 | cut -d '/' -f1`
```

Note: The default port is 3306.

- b Use `memsql` on a command line outside of the cluster:

```
memsql -h${SSHOST_EXTERNAL} -uadmin user -P${SSPORT_EXTERNAL} -
ppassword -e 'SHOW USERS'
```

4 Run the `sdb-admin sync-ldap` command with the required configuration parameters. Use the following command that includes the required flags as an example.

Note: SingleStore and SAS recommend that you use a configuration file because it provides an easier way to provide the necessary parameters. See [Step 5 on page 27](#) for the YAML configuration file that includes the equivalent command.

```
sdb-admin sync-ldap \
--uris URI(s)_LDAP_directory_server:port \
--groups group1,group2,group3 \
--search-base dc=memsql,dc=ldap,dc=com \
--query-filter "(|(objectClass=group) (&(objectClass=user)
(userPrincipalName=*)))" \
--auth-method JWT \
--show-detail \
--bind-user "CN=Some User,CN=Users,DC=memsql,DC=ldap,DC=testing" \
--bind-credentials credentials_LDAP_bind_user \
--user-object-class user \
--group-object-class group \
--user-attribute userPrincipalName \
--group-attribute sAMAccountName \
--user-member-of-attributememberOf \
--group-members-attribute member \
--require-ssl \
--ssl \
```

```
--ssl-ca path_to_CA_file \
--user admin \
--password password \
--host hostname_for_SingleStore_node\
--port port_for_SingleStore_node
```

Where:

- `uris` is the URI(s) of the LDAP directory server. Multiple values must be separated by commas.
- `groups` is the group names to sync. Group names must be separated by commas. Here is an example: `Medical,Engineering`. Only the groups that are listed are granted or revoked. Note that this option must contain non-empty value(s) for the command to successfully run.
- `search-base` is the LDAP Base DN that specifies the base of the subtree in which the search is to be constrained for a search object. Here is an example: `dc=example,dc=org`.
- `query-filter` is the filter that limits the search where group, user and `userPrincipalName` fields are not empty.
- `auth-method` is the authentication method to authenticate SingleStore users that are created from the LDAP sync. Use `JWT` for this value.
- `show-detail` shows synchronizing details about users.
- `bind-user` is the user name DN to log in to LDAP. Here is an example: `cn=admin,dc=example,dc=org`.
- `bind-credentials` is the credentials of the LDAP bind user to authenticate. To keep your bind credentials secure, use the `LDAP_BIND_CREDENTIALS` environment variable in the Linux shell. Alternatively, a configuration file can be used.
- `user-object-class` is the user objectClass name to match against an LDAP group.
- `group-object-class` is the group objectClass name to match against an LDAP group.
- `user-attribute` is the user name that is used for the SingleStore database. This field must have the value of `userPrincipalName`.
- `group-attribute` is the field name for an LDAP group.
- `user-member-of-attribute` is the field name for an LDAP user's groups.
- `group-members-attribute` is the field name for an LDAP group's users.
- `require-ssl` enables the REQUIRE SSL mode for the synchronized users.
- `ssl` allows the SQL user to connect to SingleStoreDB via SSL.
- `ssl-ca` is the path to the CA file to use when the SQL user connects to SingleStoreDB via SSL. If this option is not specified, the default CA file is used.
- `user` is the SQL user for connecting to SingleStore. The default is `root`.
- `password` is the SQL password for connecting to SingleStore.
- `host` is the cluster-addressable host name for the SingleStore node.

- port is the cluster-addressable port for the SingleStore node.

The command supports several optional flags. For more information, see [Sync LDAP Usage Information](#). For more `sdb-admin sync-ldap` examples, see [Using the LDAP Tool](#).

- 5 (OPTIONAL) As an alternative to specifying options on the command line, use a YAML configuration file and pass it to the `sdb-admin sync-ldap` command with the `--config-file` option. SingleStore and SAS recommend that you use a configuration file because it provides an easier way to provide the necessary parameters. The following YAML configuration file includes the equivalent command from Step 4:

```
sdb-admin sync-ldap --config-file directory_location/
ldap_configuration.yaml

drop_unmanaged_memsql_users: false
groups:
- group1
- group2
- group3
show_detail: true
ldap_users_to_lower: false
ldap_client:
  uris:
  - URI(s)_LDAP_directory_server:port
  start_tls: false
  ca_paths: []
  dial_with_tls_config: false
  insecure_skip_verify: false
  bind:
    user: "CN=Some User,CN=Users,DC=memsql,DC=ldap,DC=testing"
    credentials: password
  search:
    base: dc=memsql,dc=ldap,dc=com
    filter: (|(objectClass=group)&(objectClass=user)
(userPrincipalName=*))
    detail:
      user_object_class: user
      user_attribute: userPrincipalName
      user_member_of_attribute: memberOf
      group_object_class: group
      group_attribute: sAMAccountName
      group_members_attribute: member
sql_user:
  resource_pool: ""
  failed_login_attempts_limit: 0
  password_lock_time: 0
  auth_method: jwt
  pam_auth_service: ""
  kerberos_realm: ""
require_ssl: true
sql_client:
  user: admin
  password: password
  host: singlestore-cluster-ddl
  port: 3306
```

```
ssl: true
  ssl_ca_file: location_of_certs
```

- 6 Verify that JWT users and groups are added to SingleStore. Run the following commands from the `singlestore` command line:

- `show groups;`
- `show users;`
- `show users for group 'ldap_users_internal_group';`

A group called `ldap_users_internal_group` is created that **MUST** not be modified. This group contains the JWT users that were created with the `sync-ldap` tool.

Note: By default, groups in SingleStore have no granted privileges and users in the group have only `USAGE` privilege. The SingleStore database administrator must grant or revoke privileges for users.

Enable and Configure SCIM

Overview of Steps to Enable and Configure SCIM

Note: You can use either LDAP or SCIM for identity management. You cannot use both.

The System for Cross-domain Identity Management (SCIM) standard is supported starting in [2025.02](#). If you want to use SCIM, follow these steps:

- 1 Disable LDAP. Follow the instructions in [“Disable LDAP \(Required\)” in SAS Viya Platform: Identity Management](#).
- 2 [“Create a SingleStore Account and Password for Synchronization” on page 29](#)
- 3 [“Generate a SAS Viya Platform OAuth Token and a Kubernetes Secret” on page 29](#)
- 4 Schedule and start the CronJob, permanently or temporarily:
 - [“Overview of the sas-singlestore-sync CronJob” on page 31](#)
 - [“Permanently Schedule the CronJob” on page 31](#)
 - [“Temporarily Schedule and Immediately Run the CronJob” on page 32](#)
- 5 Configure verbose logging for debugging (optional), permanently or temporarily:
 - [“Overview of Verbose Logging for Debugging \(Optional\)” on page 32](#)
 - [“Permanently Configure Verbose Logging for Debugging \(Optional\)” on page 32](#)

- “Temporarily Configure Verbose Logging for Debugging (Optional)” on page 33

Create a SingleStore Account and Password for Synchronization

Create or choose one SingleStore service account and password that the synchronization tool can use. In the next topic, these values are referred to as *singlestore-username* and *singlestore-password*.

It is recommended for security reasons that the synchronization account has only minimal privileges to create, delete, and update users and groups. That is, it is recommended that you create a new account, and not reuse the database administrator account or the Kubernetes administrator account. Continue to use the database administrator account for the usual tasks that are not covered in this documentation, such as granting roles to groups and users for permissions.

See the SingleStore documentation <https://docs.singlestore.com/db/v9.0/reference/sql-reference/security-management-commands/grant/> for more information about grants.

Generate a SAS Viya Platform OAuth Token and a Kubernetes Secret

In this topic, you register an OAuth client for the identity provider and you use the client credentials to generate a SAS Viya Platform OAuth API access token. Then you create a Kubernetes secret.

More information is available in the following documents:

- You can register an OAuth client by using manual client registration or CLI client registration. See “Register OAuth Client for SCIM Client” in *SAS Viya Platform: Identity Management* and “Register a Custom Application (OAuth Client)” in *SAS Viya Platform: Authentication*.
- After you register an OAuth client, you obtain the access token by using the client credentials. See “Obtain an Access Token” in *SAS Viya Platform: Identity Management* and “Obtain an Access Token” in *SAS Viya Platform: Authentication*.

The shell script below includes an example of manual client registration. You might have a different method that you prefer for client registration. The following tasks are required, no matter which method you choose:

- Create a secret named `sas-singlestore-sync` that specifies key-value pairs for `SINGLESTORE_USER`, `SINGLESTORE_PASS`, and `OAUTH_API_ACCESS_TOKEN`.
- The three values must be encoded with a Base64 format.

Before you run the shell script below, replace the first seven variable values. For the SAS Viya platform user name, specify an account that is a member of the `SASAdministrators` group. For best practices in choosing a client ID and secret, see <https://www.oauth.com/oauth2-servers/client-registration/client-id-secret/>.

IMPORTANT In the example below, the value "access_token_validity": 63070000 specifies an extremely long-lived (approximately two-year) token. For security purposes, you can choose to reduce the duration of that token. Note that the token must be replaced before it expires.

```
#!/bin/bash
set -e

VIYA_INGRESS_URL=https://hostname.example.com

# Specify SAS Viya platform username and password.
VIYA_USERNAME=my-viya-user-name
VIYA_PASSWORD=my-viya-password

# Create an OAuth client ID and OAuth client secret to register the
OAuth client
OAUTH_CLIENT_ID=user-created-oauth-client-id
OAUTH_CLIENT_SECRET=user-created-oauth-client-secret

# Specify SingleStore account for synchronization.
SINGLESTORE_USERNAME=singlstore-username
SINGLESTORE_PASSWORD=singlstore-password

echo
echo "Executing steps to generate an OAuth API token."
echo

# Generate a bearer token.
export BEARER_TOKEN=$(curl -sk -X POST "${VIYA_INGRESS_URL}/SASLogon/
oauth/token" -u "sas.cli:" \
-H "Content-Type: application/x-www-form-urlencoded" \
-d "grant_type=password&username=${VIYA_USERNAME}&password=${
VIYA_PASSWORD}" | jq -r '.access_token')

json=$(cat <<EOF
{
  "client_id": "$OAUTH_CLIENT_ID",
  "client_secret": "$OAUTH_CLIENT_SECRET",
  "scope": ["uaa.none"],
  "authorities": ["identities"],
  "access_token_validity": 63070000,
  "authorized_grant_types": ["urn:ietf:params:oauth:grant-type:jwt-
bearer", "client_credentials"]
}
EOF
)

# Register an OAuth client.
echo "Registering the client. Registration response follows."
echo
curl -kX POST "${VIYA_INGRESS_URL}/SASLogon/oauth/clients" \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $BEARER_TOKEN" \
-d "$json"
```

```

OAUTH_API_ACCESS_TOKEN=$(curl -ks -X POST "${VIYA_INGRESS_URL}/
SASLogon/oauth/token" \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d "grant_type=client_credentials" \
    -u "${OAUTH_CLIENT_ID}:${OAUTH_CLIENT_SECRET}" | jq -r
'.access_token')

# Create the secret using kubectl. Note that the secret must be named
sas-singlestore-sync.
echo
echo "Creating secret via 'kubectl create secret...'"
echo
kubectl create secret generic sas-singlestore-sync \
    --from-literal=SINGLESTORE_USER=$SINGLESTORE_USERNAME \
    --from-literal=SINGLESTORE_PASS=$SINGLESTORE_PASSWORD \
    --from-literal=OAUTH_API_ACCESS_TOKEN=$OAUTH_API_ACCESS_TOKEN

```

Overview of the sas-singlestore-sync CronJob

To synchronize the SAS Viya platform SCIM users and groups with SingleStore, you must schedule the sas-singlestore-sync CronJob. Your deployment includes this CronJob beginning in [2025.02](#). Submit the following command to check whether the CronJob exists:

```
kubectl get cronjobs | grep sas-singlestore-sync
```

The command returns information that is similar to the following if the CronJob exists:

sas-singlestore-sync	0 0 1 1 *	True	0	<none>	133m
----------------------	-----------	------	---	--------	------

The following topics provide instructions to permanently or temporarily schedule the sas-singlestore-sync CronJob.

Permanently Schedule the CronJob

To permanently set the sas-singlestore-sync CronJob schedule, follow these steps:

- 1 Copy `$deploy/sas-bases/examples/sas-singlestore-sync/schedule-transformer.yaml` to a directory under `$deploy/site-config`. If you need to create a directory first, a recommended name is `$deploy/site-config/sas-singlestore-sync`.
- 2 Review the Kubernetes documentation for [CronJob schedule syntax](#). Update the CronJob to the required schedule by editing the `$deploy/site-config/sas-singlestore-sync/schedule-transformer.yaml` file, replacing `{{ CRON-SCHEDULE }}` with the desired schedule. Note that this transformer also sets the suspend value of the CronJob to "False", enabling the CronJob to run on the specified schedule. If you want the CronJob to remain suspended, set the suspend value to "True".
- 3 In the base kustomization.yaml file (`$deploy/kustomization.yaml`), add a reference to the schedule-transformer.yaml file path to the transformers block.

For example, if you stored the `schedule-transformer.yaml` file in a directory with the path `$deploy/site-config/sas-singlestore-sync`, you would add:

```
...
transformers:
...
- site-config/sas-singlestore-sync/schedule-transformer.yaml
```

- 4 In order for this change to take effect, you must redeploy your software according to the method that you originally used.

Temporarily Schedule and Immediately Run the CronJob

Note: This configuration is temporary. In order to persist the CronJob schedule through re-deployments, see [“Permanently Schedule the CronJob” on page 31](#).

Submit a command such as the following in order to set the `sas-singlestore-sync` CronJob schedule. This command sets the CronJob to run every day at 1 a.m.:

```
kubectl patch cronjob sas-singlestore-sync --type=merge -p '{"spec":
{"schedule":"0 1 * * *", "suspend":false}}'
```

If you want to synchronize immediately, run a command such as the following. In order to prevent concurrent runs, you must ensure that the CronJob is in a suspended state when you run this command.

```
kubectl create job --from=cronjob/sas-singlestore-sync manual-job-name
-n name-of-namespace
```

Overview of Verbose Logging for Debugging (Optional)

The `sas-singlestore-sync` tool creates pods that carry out the synchronization work. The logs for each synchronization run are located in the pod of each job.

By default, verbose logging is turned off and provides minimal details about the execution of the synchronization tool. The following topics provide instructions to permanently or temporarily enable verbose logging.

The CronJob has a default time-to-live (TTL) value of 12 hours after it finishes, and a history limit of 15 successful jobs. Take note of this ephemeral configuration when considering your strategy to save and view logs.

Permanently Configure Verbose Logging for Debugging (Optional)

To permanently enable verbose logging, follow these steps:

- 1 Copy `$deploy/sas-bases/examples/sas-singlestore-sync/configuration-transformer.yaml` to a directory under `$deploy/site-config`. You can use the same directory as for the `schedule-transformer.yaml` file if you created a directory in the task [“Permanently Schedule the CronJob” on page 31](#). The recommended directory name in that task is `$deploy/site-config/sas-singlestore-sync`.

- 2 Edit the `$deploy/site-config/sas-pyconfig/configuration-transformer.yaml` file and set the value of `DEBUG` to "True" for verbose logging. Set the value to "False" for minimal logging.
- 3 In the base `kustomization.yaml` file (`$deploy/kustomization.yaml`), add a reference to the `configuration-transformer.yaml` file path to the `transformers` block. For example, if you stored the `configuration-transformer.yaml` file in a directory with the path `$deploy/site-config/sas-singlestore-sync`, you would add this code:

```
...
transformers:
...
- site-config/sas-singlestore-sync/configuration-transformer.yaml
```

- 4 In order for this change to take effect, you must redeploy your software according to the method that you originally used.

Temporarily Configure Verbose Logging for Debugging (Optional)

Note: This configuration is temporary. In order to persist verbose logging through re-deployments, see ["Permanently Configure Verbose Logging for Debugging \(Optional\)" on page 32](#).

To temporarily enable verbose logging, follow these steps:

- 1 Determine the name of the `sas-singlestore-sync-parameters` config map:


```
kubectl get configmaps -n name-of-namespace | grep sas-singlestore-sync
```
- 2 Edit the config map using the following command. Substitute the name that you determined in the first step.


```
kubectl edit configmap configmap-name -n name-of-namespace
```
- 3 Set the value of `DEBUG` to "True" for verbose logging, and save your change. Set the value to "False" for minimal logging. After you make this change, the synchronization tool runs with the specified level of logging the next time that the CronJob is scheduled to run.

Logging and Monitoring

Overview

You can view log messages and monitor metrics for SingleStore in the following ways:

- Use the `kubectl` command-line interface (CLI).
- Use SingleStore tools to manage, log, and monitor SingleStore. For example, SingleStore has an enhanced monitoring solution that provides custom Grafana Dashboards to monitor the SingleStore cluster. For more information about monitoring the SingleStore cluster, see [Cluster Health and Performance](#) in the SingleStore documentation.

Note: The SingleStore documentation supports SingleStore deployments in a variety of environments. Be sure to identify the steps specifically for running SingleStore on Kubernetes and the compatible version of SingleStore. To find the SingleStore version in your deployment, see [“Determining the SingleStore Version” on page 38](#).

- Deploy, configure, and use SAS Viya Monitoring for Kubernetes logging and monitoring tools.
- Use SAS Enterprise Session Monitor. When SAS Enterprise Session Monitor is deployed to a Kubernetes cluster where the SAS Viya platform is running, it automatically detects, categorizes, and monitors SAS Viya and SAS SpeedyStore processes. For more information see [SAS SpeedyStore Observability using SAS Enterprise Session Monitor](#).
- Use any log-monitoring and metric-monitoring solution such as your cloud provider’s solution.

Monitoring with SingleStore Grafana Dashboards

SingleStore has a monitoring solution that is designed to capture SingleStore cluster events over time. The solution uses a SingleStore pipeline to pull the data from the exporter process to collect metrics from the SingleStore cluster and stores the data in a *metrics* database. You can then analyze and monitor the metrics data with a visualization tool, such as Grafana Dashboards. SingleStore provides custom Grafana Dashboards that you can integrate into a Grafana instance. For more information about this architecture, see [High-Level Architecture](#).

You can configure and deploy the SingleStore monitoring solution through the following ways:

- Use the instructions in SingleStore's documentation that are found here: [Configure Monitoring](#)

Note: The SingleStore documentation supports SingleStore deployments in a variety of environments. To implement their monitoring solution, be sure to identify the steps specifically for running SingleStore on Kubernetes and the compatible version with your SingleStore cluster.

- Deploy the SAS Viya Monitoring for Kubernetes project and use the [SAS SpeedyStore Monitoring](#) sample file that demonstrates how to extend SAS Viya Monitoring for Kubernetes to monitor the SingleStore instance that is embedded within SAS SpeedyStore. This allows administrators to monitor the SAS Viya components and the SingleStore cluster within the same Grafana instance. The project deploys the customized Grafana Dashboards that are provided by SingleStore.

SingleStore Grafana Dashboards include the following information:

- High-level view and availability of the SingleStore cluster
- Resource utilization by node and host
- Statistics for parameterized query execution including run count, time spent, and resource utilization
- Granular breakdown of memory use for a host

For the list of charts and what it shows, see [View the Dashboards](#) in the SingleStore documentation.

SAS Viya Monitoring for Kubernetes

The SAS Viya Monitoring for Kubernetes solution enables you to collect and monitor log messages and metrics.

The SAS Viya platform produces log messages from applications, servers, and services. Log messages are collected from all pods in a Kubernetes cluster, not only the pods that are used for the SAS Viya platform. Log messages are collected for SingleStore and (starting with [2026.09](#)) for the SAS Embedded Process.

The metric-monitoring components are used to monitor not only the resources that are specific to a SAS Viya platform environment, but also the performance of the Kubernetes cluster. Generic operations and performance metrics for SingleStore are gathered.

The [SAS Viya Monitoring for Kubernetes](#) solution is free and available on GitHub. The solution is supported by SAS. For more information, see the [SAS Viya Monitoring for Kubernetes](#) documentation.

TIP You can configure and deploy SingleStore's monitoring solution with SAS Viya Monitoring for Kubernetes. This allows administrators to monitor their SingleStore cluster using the same Grafana instance that they use to monitor the rest of their SAS Viya deployment. See the [Monitoring SAS Viya SpeedyStore \(SingleStore\)](#) sample for details.

Accessing SAS Embedded Process Logs in SingleStore

Note: Starting with 2026.09, the SAS Embedded Process writes its log to the Kubernetes pod log, instead of writing its log to the local file system of each pod. The instructions in this topic are no longer needed. Instead, see [“SAS Viya Monitoring for Kubernetes” on page 35](#).

Prior to 2026.09, the SAS Embedded Process does not write its log to the pod. The SAS Embedded Process writes its log to a file on the local file system of each pod. If you need to check whether SAS Embedded Process is running, you can check the logs.

Prior to 2026.09, use the `kubectl exec` to run a shell command into the pod:

- 1 Get a list of node names:

```
kubectl -n name-of-namespace get pods | grep singlestore-cluster | awk '{print $1}'
```

- 2 Issue the following command. Replace `$1` with one of the node names that were returned by the command in step 1.

```
kubectl exec -it $1 -- /bin/bash
```

- 3 Navigate to `/opt/sas/ep/home/log/` and open the file `sasep_01.log`.

The `sasep_01.log` file is always the most recent log file. When `sasep_01.log` exceeds the maximum size, it is renamed `sasep_02.log`. If `sasep_02.log` exists, it is renamed `sasep_03.log`, and so on.

- 4 Return to step 2 if you want to specify another node name. Usually, the SAS Embedded Process runs on one or more leaf nodes.

Backup and Restore

Starting with 2026.07, you can choose to configure automatic SingleStore backups by patching the `MemsqCluster` resource with a `backupSpec`. Automated backup

configuration is supported for Amazon S3 (starting with 2026.07) or Microsoft Azure Blob Storage (starting with 2026.08).

For configuration details, see the SAS SingleStore Cluster Operator README in the SAS deployment assets:

- Markdown: `$deploy/sas-bases/examples/sas-singlestore/backup/README.md`
- HTML: `$deploy/sas-bases/docs/optional_sas_singlestore_backup_configuration.htm`

Note that in 2026.07, the README can be found in a different location:

- Markdown: `$deploy/sas-bases/examples/sas-singlestore/README.md`
- HTML: `$deploy/sas-bases/docs/sas_singlestore_cluster_operator.htm`

The backup component supports automatic backups. SAS SpeedyStore does not support automatic restores at this time.

For restore operations, use SingleStore backup and restore tools. For more information, refer to the [SingleStore documentation website](#).

Creating the Metadata Database for the SAS Embedded Process

When you order SAS SpeedyStore, the order includes a SingleStore deployment that has SAS Embedded Process already installed. In this deployment, some in-database processing is automatically pushed into SingleStore. For details, see [“Improving Performance Using In-Database Processing” in SAS Cloud Analytic Services: User’s Guide](#).

When you assign a SAS SpeedyStore caslib, you can use the `epDatabase=` option to specify a SingleStore database for SAS Embedded Process to store metadata. A dedicated database is recommended. The metadata is not encrypted. If you must keep users from seeing each others’ metadata, then specify a separate `epDatabase=` for each user. In this case, you must also protect the databases by using role-based access control in SingleStore.

The database contains tables and functions that are used during processing by SAS Embedded Process. The metadata is automatically deleted after processing completes. Examples of in-database processing are WHERE expressions or computed columns that are automatically processed in the SingleStore data source. For details about supported processing, see [“Restrictions for WHERE Processing and Computed Columns” in SAS Cloud Analytic Services: User’s Guide](#).

One benefit of using `epDatabase=` is to separate the permissions granted for metadata from the permissions granted for data. Regardless of whether you use `epDatabase=`, you must have permissions to the SingleStore database where SAS Embedded Process stores temporary metadata. If you do not specify `epDatabase=`,

then the database specified in the `database=` option is used. To grant the permissions, an administrator uses role-based access control in SingleStore.

For information about the option, see [“epDatabase= Data Connector Option” in SAS Cloud Analytic Services: User’s Guide](#).

Determining the SingleStore Version

You can submit a query from the command-line interface. Here is an example:

```
kubectl -n name-of-namespace exec -it node-sas-singlestore-cluster-master-0 -- /bin/singlestore -u admin -pmy_password -e "select @@memsql_version"
```

Each release of SAS SpeedyStore uses a [particular version of SingleStoreDB](#). When you update to a more recent release of the SAS Viya platform, the update might include a new version of SingleStore. Do not manually change the version of SingleStore. For more information, see [“Updating SAS SpeedyStore” on page 7](#).

Character Set Considerations

Support for 4-Byte Characters

Data that includes 4-byte characters is supported with `utf8mb4` encoding. This is the default encoding for SingleStore as of [2024.10](#). For more information, see the [SingleStore documentation website](#).

However, 4-byte characters are not supported in table names or column names. This is a limitation of SingleStore.

Beginning with [2025.04](#), 4-byte characters are supported in table labels and column labels.

Beginning with [2023.06](#), the SingleStore `data_conversion_compatibility_level` has changed to 8.0. This change affects the behavior when multibyte character data exceeds the defined length and is truncated. Partial characters are now rejected. To avoid this error, ensure that the column lengths are large enough to prevent truncation.

The binary character set is not supported by SAS SpeedyStore.

Setting or Modifying the Collation

Setting the Default Collation

IMPORTANT Beginning with 2026.05, the instance of SingleStoreDB that is included with SAS SpeedyStore has a default collation of `utf8mb4_bin`, which causes string comparisons to be case-sensitive. Previous versions have a default collation of `utf8mb4_general_ci`, which is not case-sensitive. New clusters are subject to the new default if you do not set the server collation in your deployment. Existing clusters continue to use the previous collation setting or the previous default if the collation was not set. In previous releases, SAS recommended setting the server collation to `utf8mb4_bin` in the deployment before any data is loaded into SingleStore. If you followed that recommendation, or if you are not using data that was previously loaded into SingleStore, then no changes are necessary. If you did not set the collation and used the default, the difference can lead to unexpected result sets if case distinctions are meaningful. To change the setting for existing tables, see [“Modifying the Collation for Existing Tables” on page 40](#).

By default, SingleStore uses the `utf8mb4` character set. Beginning with 2026.05, the default collation is `utf8mb4_bin`. This collation causes string comparisons to be case-sensitive. SAS recommends keeping this default. If you want to change the server collation setting, SAS recommends changing the server collation setting in the SingleStore deployment before any data is loaded into tables in SingleStore.

For example, if you want to set the cluster collation server to `utf8mb4_unicode_ci`, submit the following command using the SingleStore CLI within the SingleStore cluster:

```
SET CLUSTER collation_server = 'utf8mb4_unicode_ci';
```

Note: If you set the `character_set_server` variable using `SET CLUSTER` (or its equivalent `SET GLOBAL`) or `SET AGGREGATOR`, you must also run `FLUSH CONNECTION POOLS` on every aggregator node and restart the SQL session for the new collation value to take effect.

For more information, see <https://docs.singlestore.com/db/v9.0/reference/sql-reference/character-encoding/specifying-character-set-and-collation-for-clusters/>.

Modifying the Collation for Existing Tables

If you deploy SAS SpeedyStore without changing the default collation, you can make that change later. Here are the instructions to safely change the collation after tables have been populated with data:

- 1 Change the default collation as described in the previous topic, “[Setting the Default Collation](#)” on page 39.
- 2 Modify all character columns in existing tables to use the new collation. Follow the SingleStore instructions at <https://docs.singlestore.com/db/v9.0/reference/sql-reference/data-definition-language-ddl/alter-table/#alter-table-to-modify-column-data-types-sizes-and-collation> to alter the collation on columns in existing tables. Without doing so, you could compare two columns in different tables that use different collations (the old one versus the new one). Comparing strings in different collations results in very poor performance, because all sharding is disabled.

Troubleshooting

Leaf Node Container Repeatedly Restarts Before It Can Finish Replaying Transaction Logs

After restarting SingleStore, if a workload causes `max_used_connections` to reach the value of `max_connection_threads` on the aggregators, the system can become unresponsive until the queued requests are cleared. Restarting SingleStore before it clears the queues can also cause issues.

This issue can manifest as the leaf node containers reporting “Replaying logs for db” messages that get stopped by a container restart before they can reach 100% complete, and then indefinitely restarting the node container with no apparent progress. This is a “start-up probe failure” that can be alleviated by configuring the leaf node start-up probe to wait longer before declaring leaf start-up a failure and restarting the SingleStore node container.

The solution is to modify the `memsqlcluster` resource to increase `periodSeconds` for `startupProbeSpec` in the `leafSpec`. The default for `periodSeconds` is 5.

```
kubectl -n name-of-namespace edit memsqlclusters.memsql.com sas-  
singlestore-cluster  
spec:  
  leafSpec:  
    startupProbeSpec:
```

```

failureThreshold: 60
initialDelaySeconds: 0
periodSeconds: 60
successThreshold: 1
timeoutSeconds: 120

```

SingleStore Pod "FailedScheduling" Event Due to Insufficient CPU or Memory

Overview

Kubernetes schedules a pod onto a VM that has sufficient allocatable CPU and memory to honor the CPU and memory requests of that pod's containers. Additional constraints include tolerating taints, nodeAffinity, and podAffinity settings. For more information, see ["Plan the Workload Placement" in SAS Viya Platform: Deployment Guide](#). A "FailedScheduling" event occurs when the CPU or memory that is requested by the SingleStore pod is greater than the CPU or memory resources that are available on any of the Kubernetes nodes. This problem can occur when additional pods, such as kube-system pods, SAS Viya Monitoring for Kubernetes pods, or SAS Enterprise Session Monitor pods consume too much of the allocatable CPU and memory on the nodes.

The solution is to modify the value for the `--overpack-factor` option. SingleStore uses the overpack factor to ensure that SingleStore pods can be scheduled on Kubernetes nodes.

Modify the Overpack Factor

The `--overpack-factor` option is set in the SingleStore Operator pod as a command-line argument. The default value is `0.15`.

Perform the following steps to override the default configuration:

- 1 Determine the best value for the `--overpack-factor` option. See ["How to Determine the Overpack Factor for Your Environment"](#) on page 43.
- 2 Create an overlay file that contains the new command line argument under the `op:` add key-value pair. Here is an example that sets the value to `0.17`:

```

---
apiVersion: builtin
kind: PatchTransformer
metadata:
  name: s2_operator_overpack_factor
patch: |-
  - op: add
    path: /spec/template/spec/containers/0/args/-

```

```

        value: "--overpack-factor"
      - op: add
        path: /spec/template/spec/containers/0/args/-
        value: "0.17"
    target:
      kind: Deployment
      name: sas-singlestore-operator
  ---

```

- 3 Save the file in your `$deploy/site-config` directory.
- 4 Add a reference to the file in your base `kustomization.yaml` file.
- 5 Redeploy using the method of your choice. See [“Deploy the Software” in SAS Viya Platform: Deployment Guide](#).

How the Overpack Factor Affects Resource Requests

The overpack factor specifies a portion of the SingleStore leaf pods' *node* container limits to subtract from the SingleStore leaf pods' *node* container requests. That is, $request = (1 - overpack\ factor) * limit$.

For example, if the overpack factor has a value of .10, the $request = (1 - .10) * limit = .90 * limit$. The node container's CPU and memory requests are set to 90% of the node container's CPU and memory limits.

For each leaf, the node container limits are $leaf\ height * cores\ per\ unit$ and $leaf\ height * memory\ per\ unit$. You can set these properties in `$deploy/sas-bases/examples/sas-singlestore/sas-singlestore-cluster-config.yaml` by applying a patch transformer, as shown in the following table:

Table 3.1 Where to Set Properties That Affect Container Limits

Property	PatchTransformer	Example Values
/spec/leafspec/height	MemSQL custom resource	1 unit, 1.5 units, or 2 units
/spec/aggregatorSpec/height	MemSQL custom resource	1 unit, 1.5 units, or 2 units
--cores-per-unit	SingleStore Operator	Default value: 8 vCPU
--memory-per-unit	SingleStore Operator	Default value: 64 GB

For example, if leaf height is set to 1, then each leaf is limited to 8 vCPU and 64GB memory.

For each leaf, the node container request is reduced by the overpack factor: $leaf\ core\ request = (1 - overpack\ factor) * leaf\ height * cores\ per\ unit$.

In the example where each leaf is limited to 8 vCPU and the overpack factor is 0.10, then the core request per leaf is 0.9×8 or 7.2 vCPU. Similarly, the memory request per leaf is 0.9×64 or 57.6 GB.

To view the leaf's node container limits and the leaf's node container requests in your environment, run the following command on one of your SingleStore leaf pods:

```
kubectl -n name-of-namespace describe pods node-sas-singlestore-
cluster-leaf-ag1-0 | grep 'Containers:' -A17 | grep -e Limits: -A2 -e
Requests -A2
```

Example output:

```
Limits:
  cpu:      8
  memory:   64Gi
Requests:
  cpu:      7200m
  memory:   58982Mi
```

How to Determine the Overpack Factor for Your Environment

The overpack factor is a positive number between 0 and 1. Choose a number that is as close to 0 as possible, but do not use 0 as your overpack factor. The overpack factor of 0 results in situations where leaf pods cannot be scheduled on VMs where the leaf request is equal to VM capacity, which is always greater than what is allocatable. Secondly, with the setting of 0, VMs in the SingleStore node pool might be underused.

The allocatable vCPU on a VM is always a small amount less than the VM's vCPU capacity. For example, if the node container requests 8 vCPU and the VM has 8 vCPU, then the allocatable vCPU on the VM is about 7.836 vCPU. In addition, other containers on the VM take up some of the allocatable vCPU space with their own requests.

Use the following steps to determine the best value for the `--overpack-factor` option. Each command is followed by an example of output from an environment with 12 E8s_v4 VMs in a SingleStore node pool in an Azure Kubernetes Cluster (AKS).

- 1 Gather resource information from your existing SingleStore nodes. Run the following commands for each node in your SingleStore node pool.
 - Find the minimum allocatable CPU and minimum allocatable memory of the node:

```
kubectl describe node name-of-SingleStore-node | grep Allocatable -A6 |
grep -e cpu -e memory | sort | uniq
```

In this output, allocatable CPU is 7820m, or 7.82 vCPU and allocatable memory is 59350100KiB, or 60775 MB.

```
cpu:      7820m
memory:   59350100Ki
```

- Find the maximum CPU and maximum memory that is already allocated to containers on the SingleStore VMs.

```
kubectl describe node name-of-SingleStore-node | grep 'Allocated
resources:' -A10 | grep -e cpu -e memory | sort -k2 | uniq
```

The output shows the total resources that are allocated to or requested by the containers.

```
memory          698Mi (1%)  2862Mi (4%)
cpu             710m (9%)   1001m (12%)
```

- If you want to list the containers and their resource requests that are scheduled on your SingleStore VMs, you can run this command:

```
kubectl describe node name-of-SingleStore-node | grep 'Non-terminated
Pods:' -A11
```

Figure 3.1 Example Output

Non-terminated Pods: (9 in total)		CPU Requests		CPU Limits		Memory Requests		Memory Limits		Age	
Namespace	Name										
kube-system	azure-ip-masq-agent-np7xm	100m (1%)	500m (6%)	50Mi (0%)	250Mi (0%)	4d20h					
kube-system	azure-npm-snnqh	250m (3%)	251m (3%)	300Mi (0%)	1000Mi (1%)	4d20h					
kube-system	cloud-node-manager-vbq4v	50m (0%)	0 (0%)	50Mi (0%)	512Mi (0%)	4d20h					
kube-system	csi-azuredisk-node-lsj2h	30m (0%)	0 (0%)	60Mi (0%)	400Mi (0%)	4d20h					
kube-system	csi-azurefile-node-srzgk	30m (0%)	0 (0%)	60Mi (0%)	600Mi (1%)	4d20h					
kube-system	kube-proxy-c7z78	100m (1%)	0 (0%)	0 (0%)	0 (0%)	4d20h					
logging	v4m-fb-v49bx	100m (1%)	0 (0%)	128Mi (0%)	0 (0%)	16h					
monitoring	v4m-node-exporter-65kzg	50m (0%)	250m (3%)	50Mi (0%)	100Mi (0%)	16h					
viya	sas-singlestore-osconfig-rhxp	0 (0%)	0 (0%)	0 (0%)	0 (0%)	4d20h					

- Each SingleStore leaf pod has a second container that is named *exporter*. The requests of vCPU and memory of these *exporter* containers must also be included in the total CPU and memory that is requested on your SingleStore VMs.

The exporter container has the following requests, by default: 100m vCPU and 90MiB, which equals 94MB. You can run the following command to view the exporter container requests:

```
kubectl -n viya describe pods node-sas-singlestore-cluster-leaf-
ag1-0 | grep 'exporter:' -A17 | grep 'Requests' -A3
```

Example output:

```
Requests:
  cpu:      100m
  memory:   90Mi
```

- 2 Calculate how much of the allocatable CPU and memory are available for the SingleStore leaf's node container. Use the results from Step 1 for CPU and memory values.

- The available CPU for the node container is *allocatable CPU – exporter container CPU*

For example, $\text{allocatable}(7820\text{m}) - \text{allocated}(710\text{m}) - \text{exporter}(100\text{m}) = 7010\text{m} = 7.01 \text{ vCPU}$.

- The available memory for the node container is *allocatable memory – exporter container memory*

For example, allocatable(60775MB) – allocated(732MB) – exporter(94MB)
= 59949MB.

- 3 Determine the CPU and memory limits for your SingleStore leaf's node container. For an example, see [“How the Overpack Factor Affects Resource Requests” on page 42](#). In that example, each leaf's node container is limited to 8 vCPU and 64GB of memory.
- 4 Calculate the percentage of what remains in the allocatable space versus the node container limit.
 - The percentage of allocatable vCPU = $\text{allocatable CPU} / \text{SingleStore VM CPU} * 100$.
Using the example values from Step 3, allocatable vCPU = $7.01 / 8.00 = 0.87625 * 100 = 87.625\%$.
 - The percentage of allocatable memory = $\text{allocatable memory} / \text{SingleStore VM memory} * 100$.
Using the example values from Step 3, allocatable memory = $59.949\text{GB} / 64\text{GB} = 0.9367 * 100 = 93.67\%$.
- 5 You must use the smaller of the two calculated percentages from Step 4, because the overpack factor impacts both CPU and memory requests. The minimum possible `--overpack-factor` is 1 – *smaller of the two calculated percentages*.
In the example, allocatable vCPU at 87.625% is the smaller of the two percentages. Therefore, the minimum possible `--overpack-factor` is $1 - 0.87625 = 0.12375$.
- 6 (OPTIONAL) You might want to include a small amount of extra resources, such as 5%, on the SingleStore VMs. Extra resources can be helpful if another pod needs to be added, or if the CPU or memory request is slightly increased for a kube-system pod. In cases such as these, extra resources ensure that your SingleStore leaf pod is still able to schedule on the SingleStore VMs.
For example, for 5% extra resources, `--overpack-factor` = $1 - (\text{smaller of the two calculated percentages} - 0.05) = 0.17$.

Memory Has Reached the maximum_memory Setting

The following text can be seen in various Leaf Error messages:

```
Memory used by MemSQL (XXXX Mb) has reached the 'maximum_memory'
```

Changing the following variables might enable the workload to finish:

- `--overpack-factor`
- `columnstore_disk_insert_threshold`

■ columnstore_segment_rows

The most likely long-term fix is to increase the number of leaf nodes, if possible, by doubling them. If multiple partitions exist per leaf node, then you must rebalance the partitions when you increase the number of leaf nodes.

Workload management might be a solution. For more information, see [SAS Viya Platform: Workload Management](#).

Connection Time-Out Errors

The following types of errors are most likely connection time-outs. These types of time-outs occur when all or some of the CAs approach 100% CPU utilization:

```
The libs2client 'S2ClientInit' call returned the code=2002 (Can't connect to
server on 'svc-sas-singlestore-cluster-dml.viya.svc.cluster.local' (115)).

The libs2client 'GetNextChunk' call returned the code=5 (Some readers failed: -
Cannot connect to node ... Unable to get
address information
```

Increasing the global variable `connect_timeout` from its default value of 10 seconds might help. If the ratio of one CA vCPU to two CAS worker vCPUs is maintained, then CA vCPU utilization is unlikely to approach 100%. Otherwise, increase the number of SingleStore CAs.

sas-singlestore-sync: Time-Out Error While Closing Your mysqlconnector Connection

When the time-out value is set too low, you could experience the following error:

```
Looks like there was an error closing your mysqlconnector connection:
<class 'requests.exceptions.ReadTimeout'> HTTPSConnectionPool(host='sas-identities',
port=443):
Read timed out.
```

To address this error, beginning with [2026.03](#), the default SAS SpeedyStore time-out has been increased to 120 seconds. If you need a different time-out value for troubleshooting, you can change it temporarily or permanently.

Follow these instructions if you want to temporarily change the `API_TIMEOUT` value in the ConfigMap:

- 1 Determine the name of the `sas-singlestore-sync-parameters` config map:

```
kubectl get configmaps -n name-of-namespace | grep sas-singlestore-sync
```
- 2 Edit the config map using the following command. Substitute the name that you determined in the first step.

```
kubectl edit configmap configmap-name -n name-of-namespace
```

- 3 Set the value of API_TIMEOUT under "data" to the desired value (in seconds) and save. Note that the API_TIMEOUT value must be a string (for example, "60").

Follow these instructions if you want to permanently change the API_TIMEOUT value:

- 1 Copy \$deploy/sas-bases/examples/sas-singlestore-sync/configuration-transformer.yaml to a directory under \$deploy/site-config. You can use the same directory as for the schedule-transformer.yaml file if you created a directory in the task ["Permanently Schedule the CronJob" on page 31](#). The recommended directory name in that task is \$deploy/site-config/sas-singlestore-sync.
- 2 Edit the \$deploy/site-config/sas-pyconfig/configuration-transformer.yaml file and set the value of API_TIMEOUT to the desired time-out value. For example, the following code sets the value to 300 seconds. Use a value that is appropriate for your environment.

```
- op: replace
  path: /data/API_TIMEOUT
  value: "300"
```

- 3 In the base kustomization.yaml file (\$deploy/kustomization.yaml), add a reference to the configuration-transformer.yaml file path to the transformers block. For example:

```
...
transformers:
...
- site-config/sas-singlestore-sync/configuration-transformer.yaml
```

- 4 In order for this change to take effect, you must redeploy your software according to the method that you originally used.

Operation ALTER Time-Out Error

The following error can occur when SAS fails to create a table:

```
Operation ALTER timed out while waiting for concurrent operation to finish.
Use SHOW PROCESSLIST to investigate long running concurrent operation,
or consider increasing the value of alter statement's timeout value or the
default_distributed_ddl_timeout global variable.
```

Increasing the value of the SingleStore global variable default_distributed_ddl_timeout usually fixes this issue.

When you create a table, SAS first creates a temporary table and then renames it to your specified name by using the ALTER table command. To perform this operation, SingleStore requests an exclusive lock on the file. If SingleStore cannot secure the lock in 3 minutes, the operation fails. This error is most common when you create a large number of large tables concurrently.

