



Add a cluster

Astra Control Service

NetApp
March 19, 2024

Table of Contents

- Add a cluster to Astra Control Service 1
 - Install Astra Connector for private clusters. 1
 - Add a provider-managed cluster 4
 - Add a self-managed cluster 13

Add a cluster to Astra Control Service

After you set up your environment, you're ready to create a Kubernetes cluster and then add it to Astra Control Service. This enables you to use Astra Control Service to protect your applications on the cluster.

Depending on the type of cluster you need to add to Astra Control Service, you need to use different steps to add the cluster.

- [Add a public provider-managed cluster to Astra Control Service](#): Use these steps to add a cluster that has a public IP address and is managed by a cloud provider. You will need the Service Principal account, service account, or user account for the cloud provider.
- [Add a private provider-managed cluster to Astra Control Service](#): Use these steps to add a cluster that has a private IP address and is managed by a cloud provider. You will need the Service Principal account, service account, or user account for the cloud provider.
- [Add a public self-managed cluster to Astra Control Service](#): Use these steps to add a cluster that has a public IP address and is managed by your organization. You will need to create a kubeconfig file for the cluster you want to add.
- [Add a private self-managed cluster to Astra Control Service](#): Use these steps to add a cluster that has a private IP address and is managed by your organization. You will need to create a kubeconfig file for the cluster you want to add.

Install Astra Connector for private clusters

Astra Control Service uses Astra Connector to enable communication between Astra Control Service and private clusters. You need to install Astra Connector on private clusters that you want to manage.

Astra Connector supports the following types of private clusters:

- Amazon Elastic Kubernetes Service (EKS)
- Azure Kubernetes Service (AKS)
- Google Kubernetes Engine (GKE)
- Red Hat OpenShift Service on AWS (ROSA)
- ROSA with AWS PrivateLink
- Red Hat OpenShift Container Platform on-premise

Install Astra Connector

About this task

- When you perform these steps, execute these commands against the private cluster that you want to manage with Astra Control Service.
- If you are using a bastion host, issue these commands from the command line of the bastion host.

Before you begin

- You need access to the private cluster you want to manage with Astra Control Service.

- You need Kubernetes administrator permissions to install the Astra Connector operator on the cluster.

Steps

1. Install the Astra Connector operator on the private cluster you want to manage with Astra Control Service. When you run this command, the namespace `astra-connector-operator` is created and the configuration is applied to the namespace:

```
kubectl apply -f https://github.com/NetApp/astra-connector-operator/releases/download/23.07.0-202310251519/astraconnector_operator.yaml
```

2. Verify that the operator is installed and ready:

```
kubectl get all -n astra-connector-operator
```

3. Get an API token from Astra Control. Refer to the [Astra Automation documentation](#) for instructions.
4. Create the `astra-connector` namespace:

```
kubectl create ns astra-connector
```

5. Create the Astra Connector CR file and name it `astra-connector-cr.yaml`. Update the values in brackets `<>` to match your Astra Control environment and cluster configuration:

- **<ASTRA_CONTROL_SERVICE_URL>**: The web UI URL of Astra Control Service. For example:

```
https://astra.netapp.io
```

- **<ASTRA_CONTROL_SERVICE_API_TOKEN>**: The Astra Control API token you obtained in the preceding step.
- **<PRIVATE_AKS_CLUSTER_NAME>**: (AKS clusters only) - The cluster name of the private Azure Kubernetes Service cluster. Uncomment and populate this line only if you are adding a private AKS cluster.
- **<ASTRA_CONTROL_ACCOUNT_ID>**: Obtained from the Astra Control web UI. Select the figure icon at the top right of the page and select **API access**.

```

apiVersion: netapp.astraconnector.com/v1
kind: AstraConnector
metadata:
  name: astra-connector
  namespace: astra-connector
spec:
  natssync-client:
    cloud-bridge-url: <ASTRA_CONTROL_SERVICE_URL>
  imageRegistry:
    name: theotw
    secret: ""
  astra:
    token: <ASTRA_CONTROL_SERVICE_API_TOKEN>
    #clusterName: <PRIVATE_AKS_CLUSTER_NAME>
    accountId: <ASTRA_CONTROL_ACCOUNT_ID>
    acceptEULA: yes

```

6. After you populate the `astra-connector-cr.yaml` file with the correct values, apply the CR:

```
kubectl apply -f astra-connector-cr.yaml
```

7. Verify that the Astra Connector is fully deployed:

```
kubectl get all -n astra-connector
```

8. Verify that the cluster is registered with Astra Control:

```
kubectl get astraconnector -n astra-connector
```

You should see output similar to the following:

NAME	REGISTERED	ASTRACONNECTORID
STATUS		
astra-connector	true	be475ae5-1511-4eaa-9b9e-712f09b0d065
Registered with Astra		



Make note of the ASTRACONNECTORID; you will need it when you add the cluster to Astra Control.

What's next?

Now that you've installed Astra Connector, you're ready to add your private cluster to Astra Control Service.

- [Add a private provider-managed cluster to Astra Control Service](#): Use these steps to add a cluster that has a private IP address and is managed by a cloud provider. You will need the Service Principal account, service account, or user account for the cloud provider.
- [Add a private self-managed cluster to Astra Control Service](#): Use these steps to add a cluster that has a private IP address and is managed by your organization. You will need to create a kubeconfig file for the cluster you want to add.

For more information

- [Add a cluster](#)

Add a provider-managed cluster

Add a public provider-managed cluster to Astra Control Service

After you set up your cloud environment, you're ready to create a Kubernetes cluster and then add it to Astra Control Service.

- [Create a Kubernetes cluster](#)
- [Add the cluster to Astra Control Service](#)
- [Change the default storage class](#)

Create a Kubernetes cluster

If you don't have a cluster yet, you can create one that meets the requirements of one of the following providers:

- [Astra Control Service requirements for Azure Kubernetes Service \(AKS\) with Azure NetApp Files](#)
- [Astra Control Service requirements for Azure Kubernetes Service \(AKS\) with Azure managed disks](#)
- [Astra Control Service requirements for Google Kubernetes Engine \(GKE\)](#)
- [Astra Control Service requirements for Amazon Elastic Kubernetes Service \(EKS\)](#)



Astra Control Service supports AKS clusters that use Azure Active Directory (Azure AD) for authentication and identity management. When you create the cluster, follow the instructions in the [official documentation](#) to configure the cluster to use Azure AD. You'll need to make sure your clusters meet the requirements for AKS-managed Azure AD integration.

Add the cluster to Astra Control Service

After you log in to Astra Control Service, your first step is to start managing your clusters. Before you add a cluster to Astra Control Service, you'll need to perform specific tasks and make sure the cluster meets certain requirements.

When you manage Azure Kubernetes Service and Google Kubernetes Engine clusters, note that you have two options for Astra Control Provisioner installation and lifecycle management:

- You can use Astra Control Service to automatically manage the lifecycle of Astra Control Provisioner. To do this, make sure that Astra Trident is not installed and Astra Control Provisioner is not enabled on the cluster that you want to manage with Astra Control Service. In this case, Astra Control Service automatically enables Astra Control Provisioner when you begin managing the cluster, and Astra Control Provisioner upgrades are handled automatically.
- You can manage the lifecycle of Astra Control Provisioner yourself. To do this, enable Astra Control Provisioner on the cluster before managing the cluster with Astra Control Service. In this case, Astra Control Service detects that Astra Control Provisioner is already enabled and does not reinstall it or manage Astra Control Provisioner upgrades. Refer to [Enable Astra Control Provisioner](#) for steps enable Astra Control Provisioner.

When you manage Amazon Web Services clusters with Astra Control Service, if you need storage backends that can only be used with Astra Control Provisioner, you need to enable Astra Control Provisioner manually on the cluster before you manage it with Astra Control Service. Refer to [Enable Astra Control Provisioner](#) for steps to enable Astra Control Provisioner.

Before you begin

Amazon Web Services

- You should have the JSON file containing the credentials of the IAM user that created the cluster. [Learn how to create an IAM user.](#)
- Astra Control Provisioner is required for Amazon FSx for NetApp ONTAP. If you plan to use Amazon FSx for NetApp ONTAP as a storage backend for your EKS cluster, refer to the Astra Control Provisioner information in the [EKS cluster requirements](#).
- (Optional) If you need to provide `kubectl` command access for a cluster to other IAM users that are not the cluster's creator, refer to the instructions in [How do I provide access to other IAM users and roles after cluster creation in Amazon EKS?](#)
- If you plan to use NetApp Cloud Volumes ONTAP as a storage backend, you need to configure Cloud Volumes ONTAP to work with Amazon Web Services. Refer to the Cloud Volumes ONTAP [setup documentation](#).

Microsoft Azure

- You should have the JSON file that contains the output from the Azure CLI when you created the service principal. [Learn how to set up a service principal.](#)

You'll also need your Azure subscription ID, if you didn't add it to the JSON file.

- If you plan to use NetApp Cloud Volumes ONTAP as a storage backend, you need to configure Cloud Volumes ONTAP to work with Microsoft Azure. Refer to the Cloud Volumes ONTAP [setup documentation](#).

Google Cloud

- You should have the service account key file for a service account that has the required permissions. [Learn how to set up a service account.](#)
- If you plan to use NetApp Cloud Volumes ONTAP as a storage backend, you need to configure Cloud Volumes ONTAP to work with Google Cloud. Refer to the Cloud Volumes ONTAP [setup documentation](#).

Steps

1. (Optional) If you are adding an Amazon EKS cluster or want to manage the installation and upgrades of

Astra Control Provisioner yourself, enable Astra Control Provisioner on the cluster. Refer to [Enable Astra Control Provisioner](#) for enablement steps.

2. Open the Astra Control Service web UI in a browser.
3. On the Dashboard, select **Manage Kubernetes cluster**.

Follow the prompts to add the cluster.

4. **Provider:** Select your cloud provider and then either provide the required credentials to create a new cloud instance, or select an existing cloud instance to use.
 - a. **Amazon Web Services:** Provide details about your Amazon Web Services IAM user account by uploading a JSON file or by pasting the contents of that JSON file from your clipboard.

The JSON file should contain the credentials of the IAM user that created the cluster.

- b. **Microsoft Azure:** Provide details about your Azure service principal by uploading a JSON file or by pasting the contents of that JSON file from your clipboard.

The JSON file should contain the output from the Azure CLI when you created the service principal. It can also include your subscription ID so it's automatically added to Astra. Otherwise, you need to manually enter the ID after providing the JSON.

- c. **Google Cloud Platform:** Provide the service account key file either by uploading the file or by pasting the contents from your clipboard.

Astra Control Service uses the service account to discover clusters running in Google Kubernetes Engine.

- d. **Other:** This tab is for use with self-managed clusters only.

5. **Cloud instance name:** Provide a name for the new cloud instance that will be created when you add this cluster. Learn more about [cloud instances](#).
6. Select **Next**.

Astra Control Service displays a list of clusters that you can choose from.

7. **Cluster:** Select a cluster from the list to add to Astra Control Service.



When you are selecting from the list of clusters, pay careful attention to the **Eligibility** column. If a cluster is "Ineligible" or "Partially eligible", hover over the status to determine if there's an issue with the cluster. For example, it might identify that the cluster doesn't have a worker node.

8. Select **Next**.
9. (Optional) **Storage:** Optionally, select the storage class that you'd like Kubernetes applications deployed to this cluster to use by default.
 - a. To select a new default storage class for the cluster, enable the **Assign a new default storage class** check box.
 - b. Select a new default storage class from the list.



Each cloud provider storage service displays the following price, performance, and resilience information:

- Cloud Volumes Service for Google Cloud: Price, performance, and resilience information
- Google Persistent Disk: No price, performance, or resilience information available
- Azure NetApp Files: Performance and resilience information
- Azure Managed disks: No price, performance, or resilience information available
- Amazon Elastic Block Store: No price, performance, or resilience information available
- Amazon FSx for NetApp ONTAP: No price, performance, or resilience information available
- NetApp Cloud Volumes ONTAP: No price, performance, or resilience information available

Each storage class can utilize one of the following services:

- [Cloud Volumes Service for Google Cloud](#)
- [Google Persistent Disk](#)
- [Azure NetApp Files](#)
- [Azure managed disks](#)
- [Amazon Elastic Block Store](#)
- [Amazon FSx for NetApp ONTAP](#)
- [NetApp Cloud Volumes ONTAP](#)

Learn more about [storage classes for Amazon Web Services clusters](#), [storage classes for GKE clusters](#), and [storage classes for AKS clusters](#).

10. Select **Next**.

11. **Review & Approve:** Review the configuration details.

12. Select **Add** to add the cluster to Astra Control Service.

Result

If this is the first cluster that you have added for this cloud provider, Astra Control Service creates an object store for the cloud provider for backups of applications running on eligible clusters. (When you add subsequent clusters for this cloud provider, no further object stores are created.) If you specified a default storage class, Astra Control Service sets the default storage class that you specified. For clusters managed in Amazon Web Services or Google Cloud Platform, Astra Control Service also creates an admin account on the cluster. These actions can take several minutes.

Change the default storage class

You can change the default storage class for a cluster.

Change the default storage class using Astra Control

You can change the default storage class for a cluster from within Astra Control. If your cluster uses a

previously installed storage backend service, you might not be able to use this method to change the default storage class (the **Set as default** action is not selectable). In this case, you can [Change the default storage class using the command line](#).

Steps

1. In the Astra Control Service UI, select **Clusters**.
2. On the **Clusters** page, select the cluster that you want to change.
3. Select the **Storage** tab.
4. Select the **Storage classes** category.
5. Select the **Actions** menu for the storage class that you want to set as default.
6. Select **Set as default**.

Change the default storage class using the command line

You can change the default storage class for a cluster using Kubernetes commands. This method works regardless of your cluster's configuration.

Steps

1. Log in to your Kubernetes cluster.
2. List the storage classes in your cluster:

```
kubectl get storageclass
```

3. Remove the default designation from the default storage class. Replace <SC_NAME> with the name of the storage class:

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-  
class":"false"}}}'
```

4. Mark a different storage class as default. Replace <SC_NAME> with the name of the storage class:

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

5. Confirm the new default storage class:

```
kubectl get storageclass
```

Add a private provider-managed cluster to Astra Control Service

You can use Astra Control Service to manage the following types of private provider-managed clusters:

- Amazon Elastic Kubernetes Service (EKS)
- Azure Kubernetes Service (AKS)
- Google Kubernetes Engine (GKE)
- Red Hat OpenShift Service on AWS (ROSA)
- ROSA with AWS PrivateLink

These instructions assume that you have already created a private cluster and prepared a secure method to remotely access it; for more information about creating and accessing private clusters, refer to the following documentation:

- [Azure documentation for private AKS clusters](#)
- [Azure documentation for private OpenShift clusters](#)
- [Amazon EKS documentation](#)
- [Google Kubernetes Engine \(GKE\) documentation](#)
- [Red Hat OpenShift Service on AWS \(ROSA\) documentation](#)

You need to perform the following tasks to add your private cluster to Astra Control Service:

1. [Install Astra Connector](#)
2. [Set up persistent storage](#)
3. [Add the private provider-managed cluster to Astra Control Service](#)

Install Astra Connector

Before you add a private cluster, you need to install Astra Connector on the cluster so that Astra Control can communicate with it. Refer to [Install Astra Connector for private clusters](#) for instructions.

Set up persistent storage

Configure persistent storage for the cluster. Refer to the Get Started documentation for more information about configuring persistent storage:

- [Set up Microsoft Azure with Azure NetApp Files](#)
- [Set up Microsoft Azure with Azure managed disks](#)
- [Set up Amazon Web Services](#)
- [Set up Google Cloud](#)

Add the private provider-managed cluster to Astra Control Service

You can now add the private cluster to Astra Control Service.

When you manage Azure Kubernetes Service and Google Kubernetes Engine clusters, note that you have two options for Astra Control Provisioner installation and lifecycle management:

- You can use Astra Control Service to automatically manage the lifecycle of Astra Control Provisioner. To do this, make sure that Astra Trident is not installed and Astra Control Provisioner is not enabled on the cluster that you want to manage with Astra Control Service. In this case, Astra Control Service automatically enables Astra Control Provisioner when you begin managing the cluster, and Astra Control Provisioner upgrades are handled automatically.

- You can manage the lifecycle of Astra Control Provisioner yourself. To do this, enable Astra Control Provisioner on the cluster before managing the cluster with Astra Control Service. In this case, Astra Control Service detects that Astra Control Provisioner is already enabled and does not reinstall it or manage Astra Control Provisioner upgrades. Refer to [Enable Astra Control Provisioner](#) for steps enable Astra Control Provisioner.

When you manage Amazon Web Services clusters with Astra Control Service, if you need storage backends that can only be used with Astra Control Provisioner, you need to enable Astra Control Provisioner manually on the cluster before you manage it with Astra Control Service. Refer to [Enable Astra Control Provisioner](#) for steps to enable Astra Control Provisioner.

Before you begin

Amazon Web Services

- You should have the JSON file containing the credentials of the IAM user that created the cluster. [Learn how to create an IAM user.](#)
- Astra Control Provisioner is required for Amazon FSx for NetApp ONTAP. If you plan to use Amazon FSx for NetApp ONTAP as a storage backend for your EKS cluster, refer to the Astra Control Provisioner information in the [EKS cluster requirements](#).
- (Optional) If you need to provide `kubectl` command access for a cluster to other IAM users that are not the cluster's creator, refer to the instructions in [How do I provide access to other IAM users and roles after cluster creation in Amazon EKS?](#)
- If you plan to use NetApp Cloud Volumes ONTAP as a storage backend, you need to configure Cloud Volumes ONTAP to work with Amazon Web Services. Refer to the Cloud Volumes ONTAP [setup documentation](#).

Microsoft Azure

- You should have the JSON file that contains the output from the Azure CLI when you created the service principal. [Learn how to set up a service principal.](#)

You'll also need your Azure subscription ID, if you didn't add it to the JSON file.

- If you plan to use NetApp Cloud Volumes ONTAP as a storage backend, you need to configure Cloud Volumes ONTAP to work with Microsoft Azure. Refer to the Cloud Volumes ONTAP [setup documentation](#).

Google Cloud

- You should have the service account key file for a service account that has the required permissions. [Learn how to set up a service account.](#)
- If the cluster is private, the [authorized networks](#) must allow the Astra Control Service IP address:

52.188.218.166/32
- If you plan to use NetApp Cloud Volumes ONTAP as a storage backend, you need to configure Cloud Volumes ONTAP to work with Google Cloud. Refer to the Cloud Volumes ONTAP [setup documentation](#).

Steps

1. (Optional) If you are adding an Amazon EKS cluster or want to manage the installation and upgrades of Astra Control Provisioner yourself, enable Astra Control Provisioner on the cluster. Refer to [Enable Astra](#)

[Control Provisioner](#) for enablement steps.

2. Open the Astra Control Service web UI in a browser.
3. On the Dashboard, select **Manage Kubernetes cluster**.

Follow the prompts to add the cluster.

4. **Provider:** Select your cloud provider and then either provide the required credentials to create a new cloud instance, or select an existing cloud instance to use.
 - a. **Amazon Web Services:** Provide details about your Amazon Web Services IAM user account by uploading a JSON file or by pasting the contents of that JSON file from your clipboard.

The JSON file should contain the credentials of the IAM user that created the cluster.

- b. **Microsoft Azure:** Provide details about your Azure service principal by uploading a JSON file or by pasting the contents of that JSON file from your clipboard.

The JSON file should contain the output from the Azure CLI when you created the service principal. It can also include your subscription ID so it's automatically added to Astra. Otherwise, you need to manually enter the ID after providing the JSON.

- c. **Google Cloud Platform:** Provide the service account key file either by uploading the file or by pasting the contents from your clipboard.

Astra Control Service uses the service account to discover clusters running in Google Kubernetes Engine.

- d. **Other:** This tab is for use with self-managed clusters only.
5. **Cloud instance name:** Provide a name for the new cloud instance that will be created when you add this cluster. Learn more about [cloud instances](#).
 6. Select **Next**.

Astra Control Service displays a list of clusters that you can choose from.

7. **Cluster:** Select a cluster from the list to add to Astra Control Service.



When you are selecting from the list of clusters, pay careful attention to the **Eligibility** column. If a cluster is "Ineligible" or "Partially eligible", hover over the status to determine if there's an issue with the cluster. For example, it might identify that the cluster doesn't have a worker node.

1. Select **Next**.
2. (Optional) **Storage:** Optionally, select the storage class that you'd like Kubernetes applications deployed to this cluster to use by default.
 - a. To select a new default storage class for the cluster, enable the **Assign a new default storage class** check box.
 - b. Select a new default storage class from the list.



Each cloud provider storage service displays the following price, performance, and resilience information:

- Cloud Volumes Service for Google Cloud: Price, performance, and resilience information
- Google Persistent Disk: No price, performance, or resilience information available
- Azure NetApp Files: Performance and resilience information
- Azure Managed disks: No price, performance, or resilience information available
- Amazon Elastic Block Store: No price, performance, or resilience information available
- Amazon FSx for NetApp ONTAP: No price, performance, or resilience information available
- NetApp Cloud Volumes ONTAP: No price, performance, or resilience information available

Each storage class can utilize one of the following services:

- [Cloud Volumes Service for Google Cloud](#)
- [Google Persistent Disk](#)
- [Azure NetApp Files](#)
- [Azure managed disks](#)
- [Amazon Elastic Block Store](#)
- [Amazon FSx for NetApp ONTAP](#)
- [NetApp Cloud Volumes ONTAP](#)

Learn more about [storage classes for Amazon Web Services clusters](#), [storage classes for GKE clusters](#), and [storage classes for AKS clusters](#).

3. Select **Next**.
4. **Review & Approve**: Review the configuration details.
5. Select **Add** to add the cluster to Astra Control Service.

Result

If this is the first cluster that you have added for this cloud provider, Astra Control Service creates an object store for the cloud provider for backups of applications running on eligible clusters. (When you add subsequent clusters for this cloud provider, no further object stores are created.) If you specified a default storage class, Astra Control Service sets the default storage class that you specified. For clusters managed in Amazon Web Services or Google Cloud Platform, Astra Control Service also creates an admin account on the cluster. These actions can take several minutes.

Change the default storage class

You can change the default storage class for a cluster.

Change the default storage class using Astra Control

You can change the default storage class for a cluster from within Astra Control. If your cluster uses a

previously installed storage backend service, you might not be able to use this method to change the default storage class (the **Set as default** action is not selectable). In this case, you can [Change the default storage class using the command line](#).

Steps

1. In the Astra Control Service UI, select **Clusters**.
2. On the **Clusters** page, select the cluster that you want to change.
3. Select the **Storage** tab.
4. Select the **Storage classes** category.
5. Select the **Actions** menu for the storage class that you want to set as default.
6. Select **Set as default**.

Change the default storage class using the command line

You can change the default storage class for a cluster using Kubernetes commands. This method works regardless of your cluster's configuration.

Steps

1. Log in to your Kubernetes cluster.
2. List the storage classes in your cluster:

```
kubectl get storageclass
```

3. Remove the default designation from the default storage class. Replace <SC_NAME> with the name of the storage class:

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-  
class":"false"}}}'
```

4. Mark a different storage class as default. Replace <SC_NAME> with the name of the storage class:

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

5. Confirm the new default storage class:

```
kubectl get storageclass
```

Add a self-managed cluster

Add a public self-managed cluster to Astra Control Service

After you set up your environment, you're ready to create a Kubernetes cluster and then add it to Astra Control Service.

A self-managed cluster is a cluster that you directly provision and manage. Astra Control Service supports self-managed clusters that run in a public cloud environment. You can add a self-managed cluster to Astra Control Service by uploading a `kubeconfig.yaml` file. You'll need to ensure the cluster meets the requirements outlined here.

Supported Kubernetes distributions

You can use Astra Control Service to manage the following types of public, self-managed clusters:

Kubernetes distribution	Supported versions
Kubernetes (Upstream)	1.27 to 1.29
Rancher Kubernetes Engine (RKE)	RKE 1: Versions 1.24.17, 1.25.13, 1.26.8 with Rancher Manager 2.7.9 RKE 2: Versions 1.23.16 and 1.24.13 with Rancher Manager 2.6.13 RKE 2: Versions 1.24.17, 1.25.14, 1.26.9 with Rancher Manager 2.7.9
Red Hat OpenShift Container Platform	4.12 through 4.14

These instructions assume that you have already created a self-managed cluster.

- [Add the cluster to Astra Control Service](#)
- [Change the default storage class](#)

Add the cluster to Astra Control Service

After you log in to Astra Control Service, your first step is to start managing your clusters. Before you add a cluster to Astra Control Service, you'll need to perform specific tasks and make sure the cluster meets certain requirements.

Before you begin

A self-managed cluster is a cluster that you directly provision and manage. Astra Control Service supports self-managed clusters that run in a public cloud environment. Your self-managed clusters can use Astra Control Provisioner to interface with NetApp storage services, or they can use Container Storage Interface (CSI) drivers to interface with Amazon Elastic Block Store (EBS), Azure Managed Disks, and Google Persistent Disk.

Astra Control Service supports self-managed clusters that use the following Kubernetes distributions:

- Red Hat OpenShift Container Platform
- Rancher Kubernetes Engine
- Upstream Kubernetes

Your self-managed cluster needs to meet the following requirements:

- The cluster must be accessible via the internet.
- If you are using or plan to use storage enabled with CSI drivers, the appropriate CSI drivers must be installed on the cluster. For more information on using CSI drivers to integrate storage, refer to the documentation for your storage service.
- You have access to the cluster kubeconfig file that includes only one context element. Follow [these instructions](#) to generate a kubeconfig file.
- If you are adding the cluster using a kubeconfig file that references a private Certificate Authority (CA), add the following line to the `cluster` section of the kubeconfig file. This enables Astra Control to add the cluster:

```
insecure-skip-tls-verify: true
```

- **Rancher only:** When managing application clusters in a Rancher environment, modify the application cluster's default context in the kubeconfig file provided by Rancher to use a control plane context instead of the Rancher API server context. This reduces load on the Rancher API server and improves performance.
- **Astra Control Provisioner requirements:** You should have a properly configured Astra Control Provisioner, including its Astra Trident components, to manage clusters.
 - **Review Astra Trident environment requirements:** Prior to installing or upgrading Astra Control Provisioner, review the [supported frontends, backends, and host configurations](#).
 - **Enable Astra Control Provisioner functionality:** It's highly recommended that you install Astra Trident 23.10 or later and enable [Astra Control Provisioner advanced storage functionality](#). In coming releases, Astra Control will not support Astra Trident if the Astra Control Provisioner is not also enabled.
 - **Configure a storage backend:** At least one storage backend must be [configured in Astra Trident](#) on the cluster.
 - **Configure a storage class:** At least one storage class must be [configured in Astra Trident](#) on the cluster. If a default storage class is configured, ensure that it is the **only** storage class that has the default annotation.
 - **Configure a volume snapshot controller and install a volume snapshot class:** [Install a volume snapshot controller](#) so that snapshots can be created in Astra Control. [Create](#) at least one

Steps

1. On the Dashboard, select **Manage Kubernetes cluster**.

Follow the prompts to add the cluster.

2. **Provider:** Select the **Other** tab to add details about your self-managed cluster.

- a. **Other:** Provide details about your self-managed cluster by uploading a `kubeconfig.yaml` file or by pasting the contents of the `kubeconfig.yaml` file from your clipboard.



If you create your own `kubeconfig` file, you should define only **one** context element in it. Refer to [Kubernetes documentation](#) for information about creating `kubeconfig` files.

3. **Credential name:** Provide a name for the self-managed cluster credential you are uploading to Astra Control. By default, the credential name is auto-populated as the name of the cluster.
4. **Private route identifier:** This field is for use with private clusters only.
5. Select **Next**.
6. (Optional) **Storage:** Optionally, select the storage class that you'd like Kubernetes applications deployed to this cluster to use by default.
 - a. To select a new default storage class for the cluster, enable the **Assign a new default storage class** check box.
 - b. Select a new default storage class from the list.



Each cloud provider storage service displays the following price, performance, and resilience information:

- Cloud Volumes Service for Google Cloud: Price, performance, and resilience information
- Google Persistent Disk: No price, performance, or resilience information available
- Azure NetApp Files: Performance and resilience information
- Azure Managed disks: No price, performance, or resilience information available
- Amazon Elastic Block Store: No price, performance, or resilience information available
- Amazon FSx for NetApp ONTAP: No price, performance, or resilience information available
- NetApp Cloud Volumes ONTAP: No price, performance, or resilience information available

Each storage class can utilize one of the following services:

- [Cloud Volumes Service for Google Cloud](#)
- [Google Persistent Disk](#)
- [Azure NetApp Files](#)
- [Azure managed disks](#)

- [Amazon Elastic Block Store](#)
- [Amazon FSx for NetApp ONTAP](#)
- [NetApp Cloud Volumes ONTAP](#)

Learn more about [storage classes for Amazon Web Services clusters](#), [storage classes for GKE clusters](#), and [storage classes for AKS clusters](#).

7. Select **Next**.
8. **Review & Approve**: Review the configuration details.
9. Select **Add** to add the cluster to Astra Control Service.

Change the default storage class

You can change the default storage class for a cluster.

Change the default storage class using Astra Control

You can change the default storage class for a cluster from within Astra Control. If your cluster uses a previously installed storage backend service, you might not be able to use this method to change the default storage class (the **Set as default** action is not selectable). In this case, you can [Change the default storage class using the command line](#).

Steps

1. In the Astra Control Service UI, select **Clusters**.
2. On the **Clusters** page, select the cluster that you want to change.
3. Select the **Storage** tab.
4. Select the **Storage classes** category.
5. Select the **Actions** menu for the storage class that you want to set as default.
6. Select **Set as default**.

Change the default storage class using the command line

You can change the default storage class for a cluster using Kubernetes commands. This method works regardless of your cluster's configuration.

Steps

1. Log in to your Kubernetes cluster.
2. List the storage classes in your cluster:

```
kubectl get storageclass
```

3. Remove the default designation from the default storage class. Replace <SC_NAME> with the name of the storage class:

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-  
class":"false"}}}'
```

4. Mark a different storage class as default. Replace <SC_NAME> with the name of the storage class:

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

5. Confirm the new default storage class:

```
kubectl get storageclass
```

Add a private self-managed cluster to Astra Control Service

After you set up your environment, you're ready to create a Kubernetes cluster and then add it to Astra Control Service.

A self-managed cluster is a cluster that you directly provision and manage. Astra Control Service supports self-managed clusters that run in a public cloud environment. You can add a self-managed cluster to Astra Control Service by uploading a `kubeconfig.yaml` file. You'll need to ensure the cluster meets the requirements outlined here.

Supported Kubernetes distributions

You can use Astra Control Service to manage the following types of private, self-managed clusters:

Kubernetes distribution	Supported versions
Kubernetes (Upstream)	1.27 to 1.29
Rancher Kubernetes Engine (RKE)	RKE 1: Versions 1.24.17, 1.25.13, 1.26.8 with Rancher Manager 2.7.9 RKE 2: Versions 1.23.16 and 1.24.13 with Rancher Manager 2.6.13 RKE 2: Versions 1.24.17, 1.25.14, 1.26.9 with Rancher Manager 2.7.9
Red Hat OpenShift Container Platform	4.12 through 4.14

These instructions assume that you have already created a private cluster and prepared a secure method to remotely access it.

You need to perform the following tasks to add your private cluster to Astra Control Service:

1. [Install Astra Connector](#)
2. [Set up persistent storage](#)

3. [Add the private self-managed cluster to Astra Control Service](#)

Install Astra Connector

Before you add a private cluster, you need to install Astra Connector on the cluster so that Astra Control can communicate with it. Refer to [Install Astra Connector for private clusters](#) for instructions.

Set up persistent storage

Configure persistent storage for the cluster. Refer to the Get Started documentation for more information about configuring persistent storage:

- [Set up Microsoft Azure with Azure NetApp Files](#)
- [Set up Microsoft Azure with Azure managed disks](#)
- [Set up Amazon Web Services](#)
- [Set up Google Cloud](#)

Add the private self-managed cluster to Astra Control Service

You can now add the private cluster to Astra Control Service.

Before you begin

A self-managed cluster is a cluster that you directly provision and manage. Astra Control Service supports self-managed clusters that run in a public cloud environment. Your self-managed clusters can use Astra Control Provisioner to interface with NetApp storage services, or they can use Container Storage Interface (CSI) drivers to interface with Amazon Elastic Block Store (EBS), Azure Managed Disks, and Google Persistent Disk.

Astra Control Service supports self-managed clusters that use the following Kubernetes distributions:

- Red Hat OpenShift Container Platform
- Rancher Kubernetes Engine
- Upstream Kubernetes

Your self-managed cluster needs to meet the following requirements:

- The cluster must be accessible via the internet.
- If you are using or plan to use storage enabled with CSI drivers, the appropriate CSI drivers must be installed on the cluster. For more information on using CSI drivers to integrate storage, refer to the documentation for your storage service.
- You have access to the cluster kubeconfig file that includes only one context element. Follow [these instructions](#) to generate a kubeconfig file.
- If you are adding the cluster using a kubeconfig file that references a private Certificate Authority (CA), add the following line to the `cluster` section of the kubeconfig file. This enables Astra Control to add the cluster:

```
insecure-skip-tls-verify: true
```

- **Rancher only:** When managing application clusters in a Rancher environment, modify the application cluster's default context in the kubeconfig file provided by Rancher to use a control plane context instead of the Rancher API server context. This reduces load on the Rancher API server and improves performance.
- **Astra Control Provisioner requirements:** You should have a properly configured Astra Control Provisioner, including its Astra Trident components, to manage clusters.
 - **Review Astra Trident environment requirements:** Prior to installing or upgrading Astra Control Provisioner, review the [supported frontends, backends, and host configurations](#).
 - **Enable Astra Control Provisioner functionality:** It's highly recommended that you install Astra Trident 23.10 or later and enable [Astra Control Provisioner advanced storage functionality](#). In coming releases, Astra Control will not support Astra Trident if the Astra Control Provisioner is not also enabled.
 - **Configure a storage backend:** At least one storage backend must be [configured in Astra Trident](#) on the cluster.
 - **Configure a storage class:** At least one storage class must be [configured in Astra Trident](#) on the cluster. If a default storage class is configured, ensure that it is the **only** storage class that has the default annotation.
 - **Configure a volume snapshot controller and install a volume snapshot class:** [Install a volume snapshot controller](#) so that snapshots can be created in Astra Control. [Create](#) at least one

Steps

1. On the Dashboard, select **Manage Kubernetes cluster**.

Follow the prompts to add the cluster.

2. **Provider:** Select the **Other** tab to add details about your self-managed cluster.
3. **Other:** Provide details about your self-managed cluster by uploading a `kubeconfig.yaml` file or by pasting the contents of the `kubeconfig.yaml` file from your clipboard.



If you create your own `kubeconfig` file, you should define only **one** context element in it. Refer to [these instructions](#) for information about creating `kubeconfig` files.

4. **Credential name:** Provide a name for the self-managed cluster credential you are uploading to Astra Control. By default, the credential name is auto-populated as the name of the cluster.
5. **Private route identifier:** Enter the private route identifier, which you can obtain from the Astra Connector. If you query the Astra Connector via the `kubectl get astraconnector -n astra-connector` command, the private route identifier is referred to as the `ASTRACONNECTORID`.



The private route identifier is the name associated with the Astra Connector that enables a private Kubernetes cluster to be managed by Astra. In this context, a private cluster is a Kubernetes cluster that does not expose its API server to the internet.

6. Select **Next**.
7. (Optional) **Storage:** Optionally, select the storage class that you'd like Kubernetes applications deployed to this cluster to use by default.
 - a. To select a new default storage class for the cluster, enable the **Assign a new default storage class** check box.
 - b. Select a new default storage class from the list.



Each cloud provider storage service displays the following price, performance, and resilience information:

- Cloud Volumes Service for Google Cloud: Price, performance, and resilience information
- Google Persistent Disk: No price, performance, or resilience information available
- Azure NetApp Files: Performance and resilience information
- Azure Managed disks: No price, performance, or resilience information available
- Amazon Elastic Block Store: No price, performance, or resilience information available
- Amazon FSx for NetApp ONTAP: No price, performance, or resilience information available
- NetApp Cloud Volumes ONTAP: No price, performance, or resilience information available

Each storage class can utilize one of the following services:

- [Cloud Volumes Service for Google Cloud](#)
- [Google Persistent Disk](#)
- [Azure NetApp Files](#)
- [Azure managed disks](#)
- [Amazon Elastic Block Store](#)
- [Amazon FSx for NetApp ONTAP](#)
- [NetApp Cloud Volumes ONTAP](#)

Learn more about [storage classes for Amazon Web Services clusters](#), [storage classes for GKE clusters](#), and [storage classes for AKS clusters](#).

8. Select **Next**.
9. **Review & Approve**: Review the configuration details.
10. Select **Add** to add the cluster to Astra Control Service.

Change the default storage class

You can change the default storage class for a cluster.

Change the default storage class using Astra Control

You can change the default storage class for a cluster from within Astra Control. If your cluster uses a previously installed storage backend service, you might not be able to use this method to change the default storage class (the **Set as default** action is not selectable). In this case, you can [Change the default storage class using the command line](#).

Steps

1. In the Astra Control Service UI, select **Clusters**.
2. On the **Clusters** page, select the cluster that you want to change.
3. Select the **Storage** tab.
4. Select the **Storage classes** category.
5. Select the **Actions** menu for the storage class that you want to set as default.
6. Select **Set as default**.

Change the default storage class using the command line

You can change the default storage class for a cluster using Kubernetes commands. This method works regardless of your cluster's configuration.

Steps

1. Log in to your Kubernetes cluster.
2. List the storage classes in your cluster:

```
kubectl get storageclass
```

3. Remove the default designation from the default storage class. Replace <SC_NAME> with the name of the storage class:

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-  
class":"false"}}}'
```

4. Mark a different storage class as default. Replace <SC_NAME> with the name of the storage class:

```
kubectl patch storageclass <SC_NAME> -p '{"metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

5. Confirm the new default storage class:

```
kubectl get storageclass
```

Check the Astra Trident version

To add a self-managed cluster that uses Astra Control Provisioner or Astra Trident for storage services, ensure that the installed version of Astra Trident is 23.10 or latest.

Steps

1. Determine the Astra Trident version you are running:

```
kubectl get tridentversions -n trident
```

If Astra Trident is installed, you see output similar to the following:

NAME	VERSION
trident	24.02.0

If Astra Trident is not installed, you see output similar to the following:

```
error: the server doesn't have a resource type "tridentversions"
```

2. Do one of the following:
 - If you are running Astra Trident 23.01 or earlier, use these [instructions](#) to upgrade to a more recent version of Astra Trident before upgrading to the Astra Control Provisioner. You can [perform a direct upgrade](#) to Astra Control Provisioner 24.02 if your Astra Trident is within a four-release window of version 24.02. For example, you can directly upgrade from Astra Trident 23.04 to Astra Control Provisioner 24.02.

- If you are running Astra Trident 23.10 or later, verify that Astra Control Provisioner has been [enabled](#). Astra Control Provisioner will not work with releases of Astra Control Center earlier than 23.10. [Upgrade your Astra Control Provisioner](#) so that it has the same version as the Astra Control Center you are upgrading to access the latest functionality.

3. Ensure that the pods are running:

```
kubectl get pods -n trident
```

4. Check if the storage classes are using the supported Astra Trident drivers. The provisioner name should be `csi.trident.netapp.io`. Refer to the following example:

```
kubectl get sc
```

Sample response:

NAME	PROVISIONER	RECLAIMPOLICY
VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION	AGE
ontap-gold (default)	csi.trident.netapp.io	Delete
Immediate	true	5d23h

Create a kubeconfig file

You can add a cluster to Astra Control Service using a kubeconfig file. Depending on the type of cluster you want to add, you might need to manually create a kubeconfig file for your cluster using specific steps.

- [Create a kubeconfig file for Amazon EKS clusters](#)
- [Create a kubeconfig file for Red Hat OpenShift Service on AWS \(ROSA\) clusters](#)
- [Create a kubeconfig file for other types of clusters](#)

Create a kubeconfig file for Amazon EKS clusters

Follow these instructions to create a kubeconfig file and permanent token secret for Amazon EKS clusters. A permanent token secret is required for clusters hosted in EKS.

Steps

1. Follow the instructions in the Amazon documentation to generate a kubeconfig file:

[Creating or updating a kubeconfig file for an Amazon EKS cluster](#)

2. Create a service account as follows:

- Create a service account file called `astraccontrol-service-account.yaml`.

Adjust the service account name as needed. The namespace `kube-system` is required for these steps. If you change the service account name here, you should apply the same changes in the

following steps.

```
<strong>astracontrol-service-account.yaml</strong>
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: astra-admin-account
  namespace: kube-system
```

3. Apply the service account:

```
kubectl apply -f astracontrol-service-account.yaml
```

4. Create a ClusterRoleBinding file called astracontrol-clusterrolebinding.yaml.

```
<strong>astracontrol-clusterrolebinding.yaml</strong>
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: astra-admin-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: astra-admin-account
  namespace: kube-system
```

5. Apply the cluster role binding:

```
kubectl apply -f astracontrol-clusterrolebinding.yaml
```

6. Create a service account token secret file called astracontrol-secret.yaml.

```
<strong>astracontrol-secret.yaml</strong>
```

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: astra-admin-account
  name: astra-admin-account
  namespace: kube-system
type: kubernetes.io/service-account-token
```

7. Apply the token secret:

```
kubectl apply -f astracontrol-secret.yaml
```

8. Retrieve the token secret:

```
kubectl get secret astra-admin-account -n kube-system -o  
jsonpath='{.data.token}' | base64 -d
```

9. Replace the `user` section of the AWS EKS kubeconfig file with the token, as shown in the following example:

```
user: token: k8s-aws-  
v1.aHR0cHM6Ly9zdHMudXMtd2Vzdc0yLmFtYXpvbmF3cy5jb20vP0FjdGlvbj1HZXRDYWxsZ  
XJJZGVudGl0eSZWZXJzaW9uPTIwMTETMDYtMTUmWC1BbXotQWxnbn3JpdGhtPUFXUzQtSE1BQ  
y1TSEEyNTYmWC1BbXotQ3JlZGVudGlhbD1BS01BM1JEWddkU0haWU9LSEQ2SyUyRjIwMjMwN  
DAzJTJGdXMtd2Vzdc0yJTJGc3RzJTJGYXdzNF9yZXFlZlZlZW0xJltGTQW16LURhdGU9MjAyMzA0M  
DNUMjA0MzQwWiZYLUFtei1FeHBpcmVzPTYwJlgtQTQW16LVNpZ25lZEhlYWRLcnM9aG9zdCUzQ  
ngtazhzLWF3cy1pZCZYLUFtei1TaWduYXRlcMU9YjU4ZWM0NzdiM2NkZGYxNGRhNmU4MGFI2Z  
WQ2zy2NzI2YWIwM2UyNThjMjRhNTJjNmVhNjc4MTRlNjJkOTg2Mg
```

Create a kubeconfig file for Red Hat OpenShift Service on AWS (ROSA) clusters

Follow these instructions to create a kubeconfig file for Red Hat OpenShift Service on AWS (ROSA) clusters.

Steps

1. Log in to the ROSA cluster.
2. Create a service account:

```
oc create sa astracontrol-service-account
```

3. Add a cluster role:

```
oc adm policy add-cluster-role-to-user cluster-admin -z astracontrol-  
service-account
```

4. Using the following example, create a service account secret configuration file:

```
<strong>secret-astra-sa.yaml</strong>
```

```
apiVersion: v1  
kind: Secret  
metadata:  
  name: secret-astracontrol-service-account  
  annotations:  
    kubernetes.io/service-account.name: "astracontrol-service-account"  
type: kubernetes.io/service-account-token
```

5. Create the secret:

```
oc create -f secret-astra-sa.yaml
```

6. Edit the service account that you created, and add the Astra Control service account secret name to the secrets section:

```
oc edit sa astracontrol-service-account
```

```
apiVersion: v1  
imagePullSecrets:  
- name: astracontrol-service-account-dockercfg-dvfcd  
kind: ServiceAccount  
metadata:  
  creationTimestamp: "2023-08-04T04:18:30Z"  
  name: astracontrol-service-account  
  namespace: default  
  resourceVersion: "169770"  
  uid: 965fa151-923f-4fbd-9289-30cad15998ac  
secrets:  
- name: astracontrol-service-account-dockercfg-dvfcd  
- name: secret-astracontrol-service-account ####ADD THIS ONLY####
```

7. List the service account secrets, replacing <CONTEXT> with the correct context for your installation:

```
kubectl get serviceaccount astracontrol-service-account --context  
<CONTEXT> --namespace default -o json
```

The end of the output should look similar to the following:

```
"secrets": [  
  { "name": "astracontrol-service-account-dockercfg-dvfcf"},  
  { "name": "secret-astracontrol-service-account"}  
]
```

The indices for each element in the `secrets` array begin with 0. In the above example, the index for `astracontrol-service-account-dockercfg-dvfcf` would be 0 and the index for `secret-astracontrol-service-account` would be 1. In your output, make note of the index number for the service account secret. You will need this index number in the next step.

8. Generate the kubeconfig as follows:

- a. Create a `create-kubeconfig.sh` file. Replace `TOKEN_INDEX` in the beginning of the following script with the correct value.

create-kubeconfig.sh

```
# Update these to match your environment.  
# Replace TOKEN_INDEX with the correct value  
# from the output in the previous step. If you  
# didn't change anything else above, don't change  
# anything else here.  
  
SERVICE_ACCOUNT_NAME=astracontrol-service-account  
NAMESPACE=default  
NEW_CONTEXT=astracontrol  
KUBECONFIG_FILE='kubeconfig-sa'  
  
CONTEXT=$(kubectl config current-context)  
  
SECRET_NAME=$(kubectl get serviceaccount ${SERVICE_ACCOUNT_NAME} \  
  --context ${CONTEXT} \  
  --namespace ${NAMESPACE} \  
  -o jsonpath='{.secrets[TOKEN_INDEX].name}')
```

```
TOKEN_DATA=$(kubectl get secret ${SECRET_NAME} \  
  --context ${CONTEXT} \  
  --namespace ${NAMESPACE} \  
  -o jsonpath='{.data}'
```

```

-o jsonpath='{.data.token}')
```

```

TOKEN=$(echo ${TOKEN_DATA} | base64 -d)

# Create dedicated kubeconfig
# Create a full copy
kubectl config view --raw > ${KUBECONFIG_FILE}.full.tmp

# Switch working context to correct context
kubectl --kubeconfig ${KUBECONFIG_FILE}.full.tmp config use-context
${CONTEXT}

# Minify
kubectl --kubeconfig ${KUBECONFIG_FILE}.full.tmp \
  config view --flatten --minify > ${KUBECONFIG_FILE}.tmp

# Rename context
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  rename-context ${CONTEXT} ${NEW_CONTEXT}

# Create token user
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-credentials ${CONTEXT}-${NAMESPACE}-token-user \
  --token ${TOKEN}

# Set context to use token user
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-context ${NEW_CONTEXT} --user ${CONTEXT}-${NAMESPACE}-token
-user

# Set context to correct namespace
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-context ${NEW_CONTEXT} --namespace ${NAMESPACE}

# Flatten/minify kubeconfig
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  view --flatten --minify > ${KUBECONFIG_FILE}

# Remove tmp
rm ${KUBECONFIG_FILE}.full.tmp
rm ${KUBECONFIG_FILE}.tmp
```

b. Source the commands to apply them to your Kubernetes cluster.

```
source create-kubeconfig.sh
```

9. (Optional) Rename the kubeconfig to a meaningful name for your cluster.

```
mv kubeconfig-sa YOUR_CLUSTER_NAME_kubeconfig
```

Create a kubeconfig file for other types of clusters

Follow these instructions to create a kubeconfig file for Rancher, Upstream Kubernetes, and Red Hat OpenShift clusters.

Before you begin

Ensure that you have the following on your machine before you start:

- kubectl v1.26 or later installed
- An active kubeconfig for the cluster you intend to manage with cluster admin rights for the active context

Steps

1. Create a service account:

a. Create a service account file called `astracontrol-service-account.yaml`.

Adjust the name and namespace as needed. If changes are made here, you should apply the same changes in the following steps.

```
<strong>astracontrol-service-account.yaml</strong>
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: astracontrol-service-account
  namespace: default
```

b. Apply the service account:

```
kubectl apply -f astracontrol-service-account.yaml
```

2. Create one of the following cluster roles with sufficient permissions for a cluster to be managed by Astra Control:

- **Limited cluster role:** This role contains the minimum permissions necessary for a cluster to be managed by Astra Control:

Expand for steps

- a. Create a ClusterRole file called, for example, `astra-admin-account.yaml`.

Adjust the name and namespace as needed. If changes are made here, you should apply the same changes in the following steps.

```
<strong>astra-admin-account.yaml</strong>
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: astra-admin-account
rules:

# Justification for resource permissions:

# Astra Control needs to be able to discover (list) resources
# of all types within your application.
# These permissions are required to discover, back up, and
# restore your application resources including
# secrets.
# For example, if your application contains custom resources
# or cluster-scoped resources, Astra Control
# needs '*' to discover, back up, and restore your application
# resources.

# Justification for Verbs:
# - "List" enables discovery.
# - "Get" enables resource backups and enables users to define
#   apps using GVK.
# - "Create" enables restoring an application from a snapshot
#   or backup using Astra Control.
# - "Delete" enables application resource clean-up as part of
#   an in-place restore of an application or clones.
# - "Patch" enables maintaining owner references and updating
#   labels on some resources.
# - "Update" enables replica scaling in case of operations
#   like in-place restores of your application.
# - "Watch" enables Astra Control to keep an up to date view
#   of resources.

# Manage all resources
# Necessary to back up and restore all resources in an app
- apiGroups:
```

```

- '*'
resources:
- '*'
verbs:
- get
- list
- create
- patch
- delete
- watch
- update

- nonResourceURLs:
- /metrics
verbs:
- get
- watch
- list

# Use PodSecurityPolicies
- apiGroups:
- extensions
- policy
resources:
- podsecuritypolicies
verbs:
- use

# OpenShift security - uncomment the following lines for Red
# Hat OpenShift clusters
#- apiGroups:
# - security.openshift.io
# resources:
# - securitycontextconstraints
# verbs:
# - use

```

- b. (For OpenShift clusters only) If you are creating a kubeconfig for an OpenShift cluster, uncomment the final lines in the `astra-admin-account.yaml` file after the `# Use PodSecurityPolicies` section:

```
# OpenShift security
- apiGroups:
  - security.openshift.io
  resources:
  - securitycontextconstraints
  verbs:
  - use
```

c. Apply the cluster role:

```
kubectl apply -f astra-admin-account.yaml
```

- **Expanded cluster role:** This role contains expanded permissions for a cluster to be managed by Astra Control. You might use this role if you use multiple contexts and cannot use the default Astra Control kubeconfig configured during installation or a limited role with a single context won't work in your environment:



The following `ClusterRole` steps are a general Kubernetes example. Refer to the documentation for your Kubernetes distribution for instructions specific to your environment.

Expand for steps

- a. Create a `ClusterRole` file called, for example, `astra-admin-account.yaml`.

Adjust the name and namespace as needed. If changes are made here, you should apply the same changes in the following steps.

```
<strong>astra-admin-account.yaml</strong>
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: astra-admin-account
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

- b. Apply the cluster role:

```
kubectl apply -f astra-admin-account.yaml
```

3. Create the cluster role binding for the cluster role to the service account:

- a. Create a `ClusterRoleBinding` file called `astracontrol-clusterrolebinding.yaml`.

Adjust any names and namespaces modified when creating the service account as needed.

```
<strong>astracontrol-clusterrolebinding.yaml</strong>
```

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: astracontrol-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: astra-admin-account
subjects:
- kind: ServiceAccount
  name: astracontrol-service-account
  namespace: default

```

b. Apply the cluster role binding:

```
kubectl apply -f astracontrol-clusterrolebinding.yaml
```

4. Create and apply the token secret:

a. Create a token secret file called `secret-astracontrol-service-account.yaml`.

```
<strong>secret-astracontrol-service-account.yaml</strong>
```

```

apiVersion: v1
kind: Secret
metadata:
  name: secret-astracontrol-service-account
  namespace: default
  annotations:
    kubernetes.io/service-account.name: "astracontrol-service-
account"
type: kubernetes.io/service-account-token

```

b. Apply the token secret:

```
kubectl apply -f secret-astracontrol-service-account.yaml
```

5. Add the token secret to the service account by adding its name to the `secrets` array (the last line in the following example):

```
kubectl edit sa astracontrol-service-account
```

```

apiVersion: v1
imagePullSecrets:
- name: astracontrol-service-account-dockercfg-48xhx
kind: ServiceAccount
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"v1","kind":"ServiceAccount","metadata":{"annotations":{},"name":"astracontrol-service-account","namespace":"default"},"creationTimestamp":"2023-06-14T15:25:45Z","name":"astracontrol-service-account","namespace":"default","resourceVersion":"2767069","uid":"2ce068c4-810e-4a96-ada3-49cbf9ec3f89"}
secrets:
- name: astracontrol-service-account-dockercfg-48xhx
<strong>- name: secret-astracontrol-service-account</strong>

```

6. List the service account secrets, replacing <CONTEXT> with the correct context for your installation:

```

kubectl get serviceaccount astracontrol-service-account --context
<CONTEXT> --namespace default -o json

```

The end of the output should look similar to the following:

```

"secrets": [
{ "name": "astracontrol-service-account-dockercfg-48xhx"},
{ "name": "secret-astracontrol-service-account"}
]

```

The indices for each element in the `secrets` array begin with 0. In the above example, the index for `astracontrol-service-account-dockercfg-48xhx` would be 0 and the index for `secret-astracontrol-service-account` would be 1. In your output, make note of the index number for the service account secret. You will need this index number in the next step.

7. Generate the kubeconfig as follows:

- a. Create a `create-kubeconfig.sh` file. Replace `TOKEN_INDEX` in the beginning of the following script with the correct value.

```

<strong>create-kubeconfig.sh</strong>

```

```

# Update these to match your environment.
# Replace TOKEN_INDEX with the correct value
# from the output in the previous step. If you
# didn't change anything else above, don't change
# anything else here.

SERVICE_ACCOUNT_NAME=astraccontrol-service-account
NAMESPACE=default
NEW_CONTEXT=astraccontrol
KUBECONFIG_FILE='kubeconfig-sa'

CONTEXT=$(kubectl config current-context)

SECRET_NAME=$(kubectl get serviceaccount ${SERVICE_ACCOUNT_NAME} \
  --context ${CONTEXT} \
  --namespace ${NAMESPACE} \
  -o jsonpath='{.secrets[TOKEN_INDEX].name}')
TOKEN_DATA=$(kubectl get secret ${SECRET_NAME} \
  --context ${CONTEXT} \
  --namespace ${NAMESPACE} \
  -o jsonpath='{.data.token}')

TOKEN=$(echo ${TOKEN_DATA} | base64 -d)

# Create dedicated kubeconfig
# Create a full copy
kubectl config view --raw > ${KUBECONFIG_FILE}.full.tmp

# Switch working context to correct context
kubectl --kubeconfig ${KUBECONFIG_FILE}.full.tmp config use-context
${CONTEXT}

# Minify
kubectl --kubeconfig ${KUBECONFIG_FILE}.full.tmp \
  config view --flatten --minify > ${KUBECONFIG_FILE}.tmp

# Rename context
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  rename-context ${CONTEXT} ${NEW_CONTEXT}

# Create token user
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-credentials ${CONTEXT}-${NAMESPACE}-token-user \
  --token ${TOKEN}

# Set context to use token user
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \

```

```
set-context ${NEW_CONTEXT} --user ${CONTEXT}-${NAMESPACE}-token
-user

# Set context to correct namespace
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  set-context ${NEW_CONTEXT} --namespace ${NAMESPACE}

# Flatten/minify kubeconfig
kubectl config --kubeconfig ${KUBECONFIG_FILE}.tmp \
  view --flatten --minify > ${KUBECONFIG_FILE}

# Remove tmp
rm ${KUBECONFIG_FILE}.full.tmp
rm ${KUBECONFIG_FILE}.tmp
```

b. Source the commands to apply them to your Kubernetes cluster.

```
source create-kubeconfig.sh
```

8. (Optional) Rename the kubeconfig to a meaningful name for your cluster.

```
mv kubeconfig-sa YOUR_CLUSTER_NAME_kubeconfig
```

Copyright information

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

LIMITED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data -Noncommercial Items at DFARS 252.227-7013 (FEB 2014) and FAR 52.227-19 (DEC 2007).

Data contained herein pertains to a commercial product and/or commercial service (as defined in FAR 2.101) and is proprietary to NetApp, Inc. All NetApp technical data and computer software provided under this Agreement is commercial in nature and developed solely at private expense. The U.S. Government has a non-exclusive, non-transferrable, nonsublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b) (FEB 2014).

Trademark information

NETAPP, the NETAPP logo, and the marks listed at <http://www.netapp.com/TM> are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.