Add a self-managed cluster

Astra Control Service

NetApp
March 11, 2024

Table of Contents

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)	1.26 to 1.28 (with Astra Trident 23.04 or newer)
Rancher Kubernetes Engine (RKE)	RKE 1.3 with Rancher 2.6 RKE 1.4 with Rancher 2.7 RKE 2 (v1.23.x) with Rancher 2.6 RKE 2 (v1.24.x) with Rancher 2.7
Red Hat OpenShift Container Platform	4.11 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 Trident 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 Trident**: If you are using or plan to use NetApp storage, ensure that you have installed the latest version of Astra Trident. If Astra Trident is already installed, check to make sure it is the latest version.

  You can deploy Astra Trident using either Trident operator (manually or using Helm chart) or `tridentctl`. Prior to installing or upgrading Astra Trident, review the supported frontends, backends, and host configurations.

  - **Astra Trident storage backend configured**: At least one Astra Trident storage backend must be configured on the cluster.
  - **Astra Trident storage classes configured**: At least one Astra Trident storage class must be configured on the cluster. If a default storage class is configured, ensure that only one storage class has that annotation.
  - **Astra Trident volume snapshot controller and volume snapshot class installed and configured**: The volume snapshot controller must be installed so that snapshots can be created in Astra Control. At least one Astra Trident `VolumeSnapshotClass` has been set up by an

administrator.

- **Astra Control Provisioner**: To use Astra Control Provisioner advanced management and storage provisioning features that are only accessible to Astra Control users, you must install Astra Trident 23.10 or later and enable Astra Control Provisioner functionality.

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)	1.26 to 1.28
Rancher Kubernetes Engine (RKE)	RKE 1.3 with Rancher Manager 2.6 RKE 1.4 with Rancher Manager 2.7 RKE 2 (v1.24.x) with Rancher 2.6 RKE 2 (v1.26.x) with Rancher 2.7
Red Hat OpenShift Container Platform	4.11 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 Trident 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 Trident**: If you are using or plan to use NetApp storage, ensure that you have installed the latest version of Astra Trident. If Astra Trident is already installed, check to make sure it is the latest version.

  You can deploy Astra Trident using either Trident operator (manually or using Helm chart) or `tridentctl`. Prior to installing or upgrading Astra Trident, review the supported frontends, backends, and host configurations.

  - **Astra Trident storage backend configured**: At least one Astra Trident storage backend must be configured on the cluster.
  - **Astra Trident storage classes configured**: At least one Astra Trident storage class must be configured on the cluster. If a default storage class is configured, ensure that only one storage class has that annotation.
  - **Astra Trident volume snapshot controller and volume snapshot class installed and configured**: The volume snapshot controller must be installed so that snapshots can be created in Astra Control. At least one Astra Trident `VolumeSnapshotClass` has been set up by an

administrator.

- **Astra Control Provisioner**: To use Astra Control Provisioner advanced management and storage provisioning features that are only accessible to Astra Control users, you must install Astra Trident 23.10 or later and enable Astra Control Provisioner functionality.

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 Trident for storage services, ensure that the installed version of Astra Trident is the latest.

Steps

1. Check the Astra Trident version.

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

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

```
NAME      VERSION
trident   22.10.0
```

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

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

> If Astra Trident is not installed or not current, and you want your cluster to use Astra Trident for storage services, you need to install the latest version of Astra Trident before proceeding. Refer to the Astra Trident documentation for instructions.

2. Ensure that the pods are running:

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

3. 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:

   a. Create a service account file called `astracontrol-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
XJJZGVudGl0eSZWZXJzaW9uPTIwMTEtMDYtMTUmWC1BbXotQWxnb3JpdGhtPUFXUzQtSE1BQ
y1TSEEyNTYmWC1BbXotQ3JlZGVudGlhbD1BS0lBM1JEWDdKU0haWU9LSEQ2SyUyRjIwMjMwN
DAzJTJGdXMtd2VzdC0yJTJGc3RzJTJGYXdzNF9yZXF1ZXN0JlgtQW16LURhdGU9MjAyMzA0M
DNUMjA0MzQwWiZYLUFtei1FeHBpcmVzPTYwJlgtQW16LVNpZ25lZEhlYWRlcnM9aG9zdCUzQ
ngtazhzLWF3cy1pZCZYLUFtei1TaWduYXR1cmU9YjU4ZWM0NzdiM2NkZGYxNGRhNzU4MGI2Z
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-dvfcd"},
{ "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-dvfcd` 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.

```
<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=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.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=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.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

Trademark information