Install Astra Control Center

Astra Control Center

NetApp
August 29, 2024

Table of Contents

Install Astra Control Center using the standard process

To install Astra Control Center, download the installation bundle from the NetApp Support Site and perform the following steps to install Astra Control Center Operator and Astra Control Center in your environment. You can use this procedure to install Astra Control Center in internet-connected or air-gapped environments.

For Red Hat OpenShift environments, you can use an alternative procedure to install Astra Control Center using OpenShift OperatorHub.

What you'll need

- Before you begin installation, prepare your environment for Astra Control Center deployment.
- If you have configured or want to configure pod security policies in your environment, familiarize yourself with pod security policies and how they affect Astra Control Center installation. See Understand pod security policy restrictions.
- Ensure all cluster operators are in a healthy state and available.

```
kubectl get clusteroperators
```

- Ensure all API services are in a healthy state and available:

```
kubectl get apiservices
```

- Ensure the Astra FQDN you plan to use is routable to this cluster. This means that you either have a DNS entry in your internal DNS server or you are using a core URL route that is already registered.
- If a cert-manager already exists in the cluster, you need to perform some prerequisite steps so that Astra Control Center does not install its own cert-manager.

About this task

The Astra Control Center installation process does the following:

- Installs the Astra components into the `netapp-acc` (or custom-named) namespace.
- Creates a default account.
- Establishes a default administrative user email address and default one-time password. This user is assigned the Owner role in the system that is needed for first time login to the UI.
- Helps you determine that all Astra Control Center pods are running.
- Installs the Astra UI.

(Applies to the Astra Data Store Early Access Program (EAP) release only) If you intend to manage Astra Data Store using Astra Control Center and enable VMware workflows, deploy Astra Control Center only on the `pcloud` namespace and not on the `netapp-acc` namespace or a custom namespace described in the steps of this procedure.

Do not execute the following command during the entirety of the installation process to avoid deleting all Astra Control Center pods: `kubectl delete -f astra_control_center_operator_deploy.yaml`

If you are using Red Hat's Podman instead of Docker Engine, Podman commands can be used in place of Docker commands.

Steps

To install Astra Control Center, do the following steps:

- Download and unpack the Astra Control Center bundle
- Install the NetApp Astra kubectl plugin
- Add the images to your local registry
- Set up namespace and secret for registries with auth requirements
- Install the Astra Control Center operator
- Configure Astra Control Center
- Complete Astra Control Center and operator installation
- Verify system status
- Set up ingress for load balancing
- Log in to the Astra Control Center UI

Download and unpack the Astra Control Center bundle

1. Download the Astra Control Center bundle (`astra-control-center-[version].tar.gz`) from the NetApp Support Site.
2. Download the zip of Astra Control Center certificates and keys from the NetApp Support Site.
3. (Optional) Use the following command to verify the signature of the bundle:

```
openssl dgst -sha256 -verify AstraControlCenter-public.pub -signature
astra-control-center-[version].tar.gz.sig astra-control-center-
[version].tar.gz
```

4. Extract the images:

```
tar -vxzf astra-control-center-[version].tar.gz
```

Install the NetApp Astra kubectl plugin

The NetApp Astra `kubectl` command line plugin saves time when performing common tasks associated with deploying and upgrading Astra Control Center.

What you'll need

NetApp provides binaries for the plugin for different CPU architectures and operating systems. You need to know which CPU and operating system you have before you perform this task. On Linux and Mac operating systems, you can use the `uname -a` command to gather this information.

Steps

1. List the available NetApp Astra `kubectl` plugin binaries, and note the name of the file you need for your operating system and CPU architecture:

```
ls kubectl-astra/
```

2. Copy the file to the same location as the standard `kubectl` utility. In this example, the `kubectl` utility is located in the `/usr/local/bin` directory. Replace `<binary-name>` with the name of the file you need:

```
cp kubectl-astra/<binary-name> /usr/local/bin/kubectl-astra
```

Add the images to your local registry

1. Complete the appropriate step sequence for your container engine:

Docker

1. Change to the Astra directory:

```
cd acc
```

2. Push the package images in the Astra Control Center image directory to your local registry. Make the following substitutions before running the command:
   - Replace BUNDLE_FILE with the name of the Astra Control bundle file (for example, `acc.manifest.yaml`).
   - Replace MY_REGISTRY with the URL of the Docker repository.
   - Replace MY_REGISTRY_USER with the user name.
   - Replace MY_REGISTRY_TOKEN with an authorized token for the registry.

```
kubectl astra packages push-images -m BUNDLE_FILE -r MY_REGISTRY
-u MY_REGISTRY_USER -p MY_REGISTRY_TOKEN
```

Podman

1. Log in to your registry:

```
podman login [your_registry_path]
```

2. Run the following script, making the <YOUR_REGISTRY> substitution as noted in the comments:

```
# You need to be at the root of the tarball.
# You should see these files to confirm correct location:
#   acc.manifest.yaml
#   acc/

# Replace <YOUR_REGISTRY> with your own registry (e.g
registry.customer.com or registry.customer.com/testing, etc..)
export REGISTRY=<YOUR_REGISTRY>
export PACKAGENAME=acc
export PACKAGEVERSION=22.08.1-26
export DIRECTORYNAME=acc
for astraImageFile in $(ls ${DIRECTORYNAME}/images/*.tar) ; do
  # Load to local cache
  astraImage=$(podman load --input ${astraImageFile} | sed 's/Loaded
image(s): //')

  # Remove path and keep imageName.
  astraImageNoPath=$(echo ${astraImage} | sed 's:.*/::')

  # Tag with local image repo.
  podman tag ${astraImage} ${REGISTRY}/netapp/astra/${PACKAGENAME}
/${PACKAGEVERSION}/${astraImageNoPath}

  # Push to the local repo.
  podman push ${REGISTRY}/netapp/astra/${PACKAGENAME}/
${PACKAGEVERSION}/${astraImageNoPath}
done
```

Set up namespace and secret for registries with auth requirements

1. Export the KUBECONFIG for the Astra Control Center host cluster:

```
export KUBECONFIG=[file path]
```

2. If you use a registry that requires authentication, you need to do the following:

   a. Create the `netapp-acc-operator` namespace:

```
kubectl create ns netapp-acc-operator
```

   Response:

```
namespace/netapp-acc-operator created
```

b. Create a secret for the `netapp-acc-operator` namespace. Add Docker information and run the following command:

> The placeholder `your_registry_path` should match the location of the images that you uploaded earlier (for example, `[Registry_URL]/netapp/astra/astracc/22.08.1-26`).

```
kubectl create secret docker-registry astra-registry-cred -n netapp-
acc-operator --docker-server=[your_registry_path] --docker-username
=[username] --docker-password=[token]
```

Sample response:

```
secret/astra-registry-cred created
```

> If you delete the namespace after the secret is generated, you need to regenerate the secret for the namespace after the namespace is recreated.

c. Create the `netapp-acc` (or custom named) namespace.

```
kubectl create ns [netapp-acc or custom namespace]
```

Sample response:

```
namespace/netapp-acc created
```

d. Create a secret for the `netapp-acc` (or custom named) namespace. Add Docker information and run the following command:

```
kubectl create secret docker-registry astra-registry-cred -n [netapp-
acc or custom namespace] --docker-server=[your_registry_path]
--docker-username=[username] --docker-password=[token]
```

Response

```
secret/astra-registry-cred created
```

e. (Optional) If you want the cluster to be automatically managed by Astra Control Center after

installation, make sure that you provide the kubeconfig as a secret within the Astra Control Center namespace you intend to deploy into using this command:

```
kubectl create secret generic [acc-kubeconfig-cred or custom secret name] --from-file=<path-to-your-kubeconfig> -n [netapp-acc or custom namespace]
```

Install the Astra Control Center operator

1. Change the directory:

```
cd manifests
```

2. Edit the Astra Control Center operator deployment YAML (`astra_control_center_operator_deploy.yaml`) to refer to your local registry and secret.

```
vim astra_control_center_operator_deploy.yaml
```

An annotated sample YAML follows these steps.

a. If you use a registry that requires authentication, replace the default line of `imagePullSecrets: []` with the following:

```
imagePullSecrets:
- name: <astra-registry-cred>
```

b. Change `[your_registry_path]` for the `kube-rbac-proxy` image to the registry path where you pushed the images in a previous step.

c. Change `[your_registry_path]` for the `acc-operator-controller-manager` image to the registry path where you pushed the images in a previous step.

d. (For installations using Astra Data Store preview) See this known issue regarding storage class provisioners and additional changes you will need to make to the YAML.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    control-plane: controller-manager
  name: acc-operator-controller-manager
  namespace: netapp-acc-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      control-plane: controller-manager
  template:
    metadata:
      labels:
        control-plane: controller-manager
    spec:
      containers:
      - args:
        - --secure-listen-address=0.0.0.0:8443
        - --upstream=http://127.0.0.1:8080/
        - --logtostderr=true
        - --v=10
        image: [your_registry_path]/kube-rbac-proxy:v4.8.0
        name: kube-rbac-proxy
        ports:
        - containerPort: 8443
          name: https
      - args:
        - --health-probe-bind-address=:8081
        - --metrics-bind-address=127.0.0.1:8080
        - --leader-elect
        command:
        - /manager
        env:
        - name: ACCOP_LOG_LEVEL
          value: "2"
        image: [your_registry_path]/acc-operator:[version x.y.z]
        imagePullPolicy: IfNotPresent
      imagePullSecrets: []
```

3. Install the Astra Control Center operator:

```
kubectl apply -f astra_control_center_operator_deploy.yaml
```

Sample response:

```
namespace/netapp-acc-operator created
customresourcedefinition.apiextensions.k8s.io/astracontrolcenters.astra.
netapp.io created
role.rbac.authorization.k8s.io/acc-operator-leader-election-role created
clusterrole.rbac.authorization.k8s.io/acc-operator-manager-role created
clusterrole.rbac.authorization.k8s.io/acc-operator-metrics-reader
created
clusterrole.rbac.authorization.k8s.io/acc-operator-proxy-role created
rolebinding.rbac.authorization.k8s.io/acc-operator-leader-election-
rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/acc-operator-manager-
rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/acc-operator-proxy-
rolebinding created
configmap/acc-operator-manager-config created
service/acc-operator-controller-manager-metrics-service created
deployment.apps/acc-operator-controller-manager created
```

4. Verify pods are running:

```
kubectl get pods -n netapp-acc-operator
```

Configure Astra Control Center

1. Edit the Astra Control Center custom resource (CR) file (`astra_control_center_min.yaml`) to make account, autoSupport, registry, and other necessary configurations:

> `astra_control_center_min.yaml` is the default CR and is suitable for most installations. Familiarize yourself with all CR options and their potential values to ensure you deploy Astra Control Center correctly for your environment. If additional customizations are required for your environment, you can use `astra_control_center.yaml` as an alternative CR.

```
vim astra_control_center_min.yaml
```

> If you are using a registry that does not require authorization, you must delete the `secret` line within `imageRegistry` or the installation will fail.

   a. Change `[your_registry_path]` to the registry path where you pushed the images in the previous step.

   b. Change the `accountName` string to the name you want to associate with the account.

c. Change the `astraAddress` string to the FQDN you want to use in your browser to access Astra. Do not use `http://` or `https://` in the address. Copy this FQDN for use in a later step.

d. Change the `email` string to the default initial administrator address. Copy this email address for use in a later step.

e. Change `enrolled` for AutoSupport to `false` for sites without internet connectivity or retain `true` for connected sites.

f. If you use an external cert-manager, add the following lines to `spec`:

```
spec:
  crds:
    externalCertManager: true
```

g. (Optional) Add a first name `firstName` and last name `lastName` of the user associated with the account. You can perform this step now or later within the UI.

h. (Optional) Change the `storageClass` value to another Trident storageClass resource if required by your installation.

i. (Optional) If you want the cluster to be automatically managed by Astra Control Center after installation and you have already created the secret containing the kubeconfig for this cluster, provide the name of the secret by adding a new field to this YAML file called `astraKubeConfigSecret: "acc-kubeconfig-cred or custom secret name"`

j. Complete one of the following steps:

- **Other ingress controller (ingressType:Generic)**: This is the default action with Astra Control Center. After Astra Control Center is deployed, you will need to configure the ingress controller to expose Astra Control Center with a URL.

  The default Astra Control Center installation sets up its gateway (`service/traefik`) to be of the type `ClusterIP`. This default installation requires you to additionally set up a Kubernetes IngressController/Ingress to route traffic to it. If you want to use an ingress, see Set up ingress for load balancing.

- **Service load balancer (ingressType:AccTraefik)**: If you don't want to install an IngressController or create an Ingress resource, set `ingressType` to `AccTraefik`.

  This deploys the Astra Control Center `traefik` gateway as a Kubernetes LoadBalancer type service.

  Astra Control Center uses a service of the type "LoadBalancer" (`svc/traefik` in the Astra Control Center namespace), and requires that it be assigned an accessible external IP address. If load balancers are permitted in your environment and you don't already have one configured, you can use MetalLB or another external service load balancer to assign an external IP address to the service. In the internal DNS server configuration, you should point the chosen DNS name for Astra Control Center to the load-balanced IP address.

  For details about the service type of "LoadBalancer" and ingress, see Requirements.

```
apiVersion: astra.netapp.io/v1
kind: AstraControlCenter
metadata:
  name: astra
spec:
  accountName: "Example"
  astraVersion: "ASTRA_VERSION"
  astraAddress: "astra.example.com"
  astraKubeConfigSecret: "acc-kubeconfig-cred or custom secret name"
  ingressType: "Generic"
  autoSupport:
    enrolled: true
  email: "[admin@example.com]"
  firstName: "SRE"
  lastName: "Admin"
  imageRegistry:
    name: "[your_registry_path]"
    secret: "astra-registry-cred"
  storageClass: "ontap-gold"
```

Complete Astra Control Center and operator installation

1. If you didn't already do so in a previous step, create the `netapp-acc` (or custom) namespace:

   ```
   kubectl create ns [netapp-acc or custom namespace]
   ```

   Sample response:

   ```
   namespace/netapp-acc created
   ```

2. Install Astra Control Center in the `netapp-acc` (or your custom) namespace:

   ```
   kubectl apply -f astra_control_center_min.yaml -n [netapp-acc or custom
   namespace]
   ```

   Sample response:

   ```
   astracontrolcenter.astra.netapp.io/astra created
   ```

Verify system status

If you prefer to use OpenShift, you can use comparable oc commands for verification steps.

1. Verify that all system components installed successfully.

```
kubectl get pods -n [netapp-acc or custom namespace]
```

Each pod should have a status of `Running`. It may take several minutes before the system pods are deployed.

Sample response

```
NAME                                        READY   STATUS    RESTARTS
AGE
acc-helm-repo-6b44d68d94-d8m55              1/1     Running   0
13m
activity-78f99ddf8-hltct                    1/1     Running   0
10m
api-token-authentication-457nl              1/1     Running   0
9m28s
api-token-authentication-dgwsz              1/1     Running   0
9m28s
api-token-authentication-hmqqc              1/1     Running   0
9m28s
asup-75fd554dc6-m6qzh                       1/1     Running   0
9m38s
authentication-6779b4c85d-92gds             1/1     Running   0
8m11s
bucketservice-7cc767f8f8-lqwr8              1/1     Running   0
9m31s
certificates-549fd5d6cb-5kmd6               1/1     Running   0
9m56s
certificates-549fd5d6cb-bkjh9               1/1     Running   0
9m56s
cloud-extension-7bcb7948b-hn8h2             1/1     Running   0
10m
cloud-insights-service-56ccf86647-fgg69     1/1     Running   0
9m46s
composite-compute-677685b9bb-7vgsf          1/1     Running   0
10m
composite-volume-657d6c5585-dnq79           1/1     Running   0
9m49s
credentials-755fd867c8-vrlmt                1/1     Running   0
11m
entitlement-86495cdf5b-nwhh2                1/1     Running   2
10m
features-5684fb8b56-8d6s8                   1/1     Running   0
10m
fluent-bit-ds-rhx7v                         1/1     Running   0
7m48s
fluent-bit-ds-rjms4                         1/1     Running   0
7m48s
fluent-bit-ds-zf5ph                         1/1     Running   0
7m48s
graphql-server-66d895f544-w6hjd             1/1     Running   0
3m29s
```

```
identity-744df448d5-rlcmm                1/1    Running  0
10m
influxdb2-0                              1/1    Running  0
13m
keycloak-operator-75c965cc54-z7csw       1/1    Running  0
8m16s
krakend-798d6df96f-9z2sk                 1/1    Running  0
3m26s
license-5fb7d75765-f8mjg                 1/1    Running  0
9m50s
login-ui-7d5b7df85d-l2s7s                1/1    Running  0
3m20s
loki-0                                   1/1    Running  0
13m
metrics-facade-599b9d7fcc-gtmgl          1/1    Running  0
9m40s
monitoring-operator-67cc74f844-cdplp     2/2    Running  0
8m11s
nats-0                                   1/1    Running  0
13m
nats-1                                   1/1    Running  0
13m
nats-2                                   1/1    Running  0
12m
nautilus-769f5b74cd-k5jxm                1/1    Running  0
9m42s
nautilus-769f5b74cd-kd9gd                1/1    Running  0
8m59s
openapi-84f6ccd8ff-76kvp                 1/1    Running  0
9m34s
packages-6f59fc67dc-4g2f5                1/1    Running  0
9m52s
polaris-consul-consul-server-0           1/1    Running  0
13m
polaris-consul-consul-server-1           1/1    Running  0
13m
polaris-consul-consul-server-2           1/1    Running  0
13m
polaris-keycloak-0                       1/1    Running  0
8m7s
polaris-keycloak-1                       1/1    Running  0
5m49s
polaris-keycloak-2                       1/1    Running  0
5m15s
polaris-keycloak-db-0                    1/1    Running  0
8m6s
```

```
polaris-keycloak-db-1                     1/1    Running  0
5m49s
polaris-keycloak-db-2                     1/1    Running  0
4m57s
polaris-mongodb-0                         2/2    Running  0
13m
polaris-mongodb-1                         2/2    Running  0
12m
polaris-mongodb-2                         2/2    Running  0
12m
polaris-ui-565f56bf7b-zwr8b               1/1    Running  0
3m19s
polaris-vault-0                           1/1    Running  0
13m
polaris-vault-1                           1/1    Running  0
13m
polaris-vault-2                           1/1    Running  0
13m
public-metrics-6d86d66444-2wbzl           1/1    Running  0
9m30s
storage-backend-metrics-77c5d98dcd-dbhg5 1/1    Running  0
9m44s
storage-provider-78c885f57c-6zcv4         1/1    Running  0
9m36s
telegraf-ds-2l2m9                         1/1    Running  0
7m48s
telegraf-ds-qfzgh                         1/1    Running  0
7m48s
telegraf-ds-shrms                         1/1    Running  0
7m48s
telegraf-rs-bjpkt                         1/1    Running  0
7m48s
telemetry-service-6684696c64-qzfdf        1/1    Running  0
10m
tenancy-6596b6c54d-vmpsm                  1/1    Running  0
10m
traefik-7489dc59f9-6mnst                  1/1    Running  0
3m19s
traefik-7489dc59f9-xrkgg                  1/1    Running  0
3m4s
trident-svc-6c8dc458f5-jswcl              1/1    Running  0
10m
vault-controller-6b954f9b76-gz9nm         1/1    Running  0
11m
```

2. (Optional) To ensure the installation is completed, you can watch the `acc-operator` logs using the following command.

```
kubectl logs deploy/acc-operator-controller-manager -n netapp-acc-
operator -c manager -f
```

> `accHost` cluster registration is one of the last operations, and if it fails it will not cause deployment to fail. In the event of a cluster registration failure indicated in the logs, you can attempt registration again through the add cluster workflow in the UI or API.

3. When all the pods are running, verify that the installation was successful (`READY` is `True`) and get the one-time password you will use when you log in to Astra Control Center:

```
kubectl get AstraControlCenter -n netapp-acc
```

Response:

```
NAME    UUID                                    VERSION     ADDRESS
READY
astra   ACC-9aa5fdae-4214-4cb7-9976-5d8b4c0ce27f  22.08.1-26
10.111.111.111  True
```

> Copy the UUID value. The password is `ACC-` followed by the UUID value (`ACC-[UUID]` or, in this example, `ACC-9aa5fdae-4214-4cb7-9976-5d8b4c0ce27f`).

Set up ingress for load balancing

You can set up a Kubernetes ingress controller that manages external access to services, such as load balancing in a cluster.

This procedure explains how to set up an ingress controller (`ingressType:Generic`). This is the default action with Astra Control Center. After Astra Control Center is deployed, you will need to configure the ingress controller to expose Astra Control Center with a URL.

> If you don't want to set up an ingress controller, you can set `ingressType:AccTraefik`). Astra Control Center uses a service of the type "LoadBalancer" (`svc/traefik` in the Astra Control Center namespace), and requires that it be assigned an accessible external IP address. If load balancers are permitted in your environment and you don't already have one configured, you can use MetalLB or another external service load balancer to assign an external IP address to the service. In the internal DNS server configuration, you should point the chosen DNS name for Astra Control Center to the load-balanced IP address. For details about the service type of "LoadBalancer" and ingress, see Requirements.

The steps differ depending on the type of ingress controller you use:

- Istio ingress
- Nginx ingress controller
- OpenShift ingress controller

What you'll need

- The required ingress controller should already be deployed.
- The ingress class corresponding to the ingress controller should already be created.
- You are using Kubernetes versions between and including v1.19 and v1.22.

Steps for Istio ingress

1. Configure Istio ingress.

   This procedure assumes that Istio is deployed using the "default" configuration profile.

2. Gather or create the desired certificate and private key file for the Ingress Gateway.

   You can use a CA-signed or self-signed certificate. The common name must be the Astra address (FQDN).

   Sample command:

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048
   -keyout tls.key -out tls.crt
   ```

3. Create a secret `tls secret name` of type `kubernetes.io/tls` for a TLS private key and certificate in the `istio-system namespace` as described in TLS secrets.

   Sample command:

   ```
   kubectl create secret tls [tls secret name]
   --key="tls.key"
   --cert="tls.crt" -n istio-system
   ```

   The name of the secret should match the `spec.tls.secretName` provided in `istio-ingress.yaml` file.

4. Deploy an ingress resource in `netapp-acc` (or custom-named) namespace using either the v1beta1 (deprecated in Kubernetes version less than or 1.22) or v1 resource type for either a deprecated or a new schema:

   Output:

```
apiVersion: networking.k8s.io/v1beta1
kind: IngressClass
metadata:
  name: istio
spec:
  controller: istio.io/ingress-controller
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress
  namespace: istio-system
spec:
  ingressClassName: istio
  tls:
  - hosts:
    - <ACC addess>
    secretName: [tls secret name]
  rules:
  - host: [ACC addess]
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          serviceName: traefik
          servicePort: 80
```

For the v1 new schema, follow this sample:

```
kubectl apply -f istio-Ingress.yaml
```

Output:

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: istio
spec:
  controller: istio.io/ingress-controller
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
  namespace: istio-system
spec:
  ingressClassName: istio
  tls:
  - hosts:
    - <ACC addess>
    secretName: [tls secret name]
  rules:
  - host: [ACC addess]
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: traefik
            port:
              number: 80
```

5. Deploy Astra Control Center as usual.
6. Check the status of the ingress:

```
kubectl get ingress -n netapp-acc
```

Response:

```
NAME    CLASS HOSTS             ADDRESS         PORTS   AGE
ingress istio astra.example.com 172.16.103.248  80, 443 1h
```

Steps for Nginx ingress controller

1. Create a secret of type `kubernetes.io/tls` for a TLS private key and certificate in `netapp-acc` (or custom-named) namespace as described in TLS secrets.

2. Deploy an ingress resource in `netapp-acc` (or custom-named) namespace using either the `v1beta1` (deprecated in Kubernetes version less than or 1.22) or `v1` resource type for either a deprecated or a new schema:

   a. For a `v1beta1` deprecated schema, follow this sample:

```
apiVersion: extensions/v1beta1
Kind: IngressClass
metadata:
  name: ingress-acc
  namespace: [netapp-acc or custom namespace]
  annotations:
    kubernetes.io/ingress.class: [class name for nginx controller]
spec:
  tls:
  - hosts:
    - <ACC address>
    secretName: [tls secret name]
  rules:
  - host: [ACC address]
    http:
      paths:
      - backend:
        serviceName: traefik
        servicePort: 80
        pathType: ImplementationSpecific
```

   b. For the `v1` new schema, follow this sample:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: netapp-acc-ingress
  namespace: [netapp-acc or custom namespace]
spec:
  ingressClassName: [class name for nginx controller]
  tls:
  - hosts:
    - <ACC address>
    secretName: [tls secret name]
  rules:
  - host: <ACC addess>
    http:
      paths:
        - path:
          backend:
            service:
              name: traefik
              port:
                number: 80
          pathType: ImplementationSpecific
```

Steps for OpenShift ingress controller

1. Procure your certificate and get the key, certificate, and CA files ready for use by the OpenShift route.
2. Create the OpenShift route:

```
oc create route edge --service=traefik
--port=web -n [netapp-acc or custom namespace]
--insecure-policy=Redirect --hostname=<ACC address>
--cert=cert.pem --key=key.pem
```

Log in to the Astra Control Center UI

After installing Astra Control Center, you will change the password for the default administrator and log in to the Astra Control Center UI dashboard.

Steps

1. In a browser, enter the FQDN you used in the `astraAddress` in the `astra_control_center_min.yaml` CR when you installed Astra Control Center.
2. Accept the self-signed certificates when prompted.

> You can create a custom certificate after login.

3. At the Astra Control Center login page, enter the value you used for `email` in `astra_control_center_min.yaml` CR when you installed Astra Control Center, followed by the one-time password (`ACC-[UUID]`).

   If you enter an incorrect password three times, the admin account will be locked for 15 minutes.

4. Select **Login**.
5. Change the password when prompted.

   If this is your first login and you forget the password and no other administrative user accounts have yet been created, contact NetApp Support for password recovery assistance.

6. (Optional) Remove the existing self-signed TLS certificate and replace it with a custom TLS certificate signed by a Certificate Authority (CA).

Troubleshoot the installation

If any of the services are in `Error` status, you can inspect the logs. Look for API response codes in the 400 to 500 range. Those indicate the place where a failure happened.

Steps

1. To inspect the Astra Control Center operator logs, enter the following:

```
kubectl logs --follow -n netapp-acc-operator $(kubectl get pods -n
netapp-acc-operator -o name) -c manager
```

What's next

Complete the deployment by performing setup tasks.

Understand pod security policy restrictions

Astra Control Center supports privilege limitation through pod security policies (PSPs). Pod security policies enable you to limit what users or groups are able to run containers and what privileges those containers can have.

Some Kubernetes distributions, such as RKE2, have a default pod security policy that is too restrictive, and causes problems when installing Astra Control Center.

You can use the information and examples included here to understand the pod security policies that Astra Control Center creates, and configure pod security policies that provide the protection you need without interfering with Astra Control Center functions.

PSPs installed by Astra Control Center

Astra Control Center creates several pod security policies during installation. Some of these are permanent, and some of them are created during certain operations and are removed once the operation is complete.

PSPs created during installation

During Astra Control Center installation, the Astra Control Center operator installs a custom pod security policy, a Role object, and a RoleBinding object to support the deployment of Astra Control Center services in the Astra Control Center namespace.

The new policy and objects have the following attributes:

```
kubectl get psp

NAME                            PRIV    CAPS          SELINUX    RUNASUSER
FSGROUP     SUPGROUP    READONLYROOTFS   VOLUMES
avp-psp                         false                 RunAsAny   RunAsAny
RunAsAny    RunAsAny    false            *
netapp-astra-deployment-psp     false                 RunAsAny   RunAsAny
RunAsAny    RunAsAny    false            *

kubectl get role

NAME                                    CREATED AT
netapp-astra-deployment-role            2022-06-27T19:34:58Z

kubectl get rolebinding

NAME                                    ROLE
AGE
netapp-astra-deployment-rb              Role/netapp-astra-deployment-role
32m
```

PSPs created during backup operations

During backup operations, Astra Control Center creates a dynamic pod security policy, a ClusterRole object, and a RoleBinding object. These support the backup process, which happens in a separate namespace.

The new policy and objects have the following attributes:

```
kubectl get psp

NAME                            PRIV    CAPS
SELINUX    RUNASUSER          FSGROUP     SUPGROUP    READONLYROOTFS
VOLUMES
netapp-astra-backup             false   DAC_READ_SEARCH
RunAsAny   RunAsAny           RunAsAny    RunAsAny    false           *

kubectl get role

NAME                   CREATED AT
netapp-astra-backup   2022-07-21T00:00:00Z

kubectl get rolebinding

NAME                   ROLE                         AGE
netapp-astra-backup   Role/netapp-astra-backup   62s
```

PSPs created during cluster management

When you manage a cluster, Astra Control Center installs the netapp-monitoring operator in the managed cluster. This operator creates a pod security policy, a ClusterRole object, and a RoleBinding object to deploy telemetry services in the Astra Control Center namespace.

The new policy and objects have the following attributes:

```
kubectl get psp

NAME                            PRIV    CAPS
SELINUX    RUNASUSER          FSGROUP     SUPGROUP    READONLYROOTFS
VOLUMES
netapp-monitoring-psp-nkmo      true    AUDIT_WRITE,NET_ADMIN,NET_RAW
RunAsAny   RunAsAny           RunAsAny    RunAsAny    false           *

kubectl get role

NAME                                          CREATED AT
netapp-monitoring-role-privileged             2022-07-21T00:00:00Z

kubectl get rolebinding

NAME                                                ROLE
AGE
netapp-monitoring-role-binding-privileged           Role/netapp-
monitoring-role-privileged             2m5s
```

Enable network communication between namespaces

Some environments use NetworkPolicy constructs to restrict traffic between namespaces. The Astra Control Center operator, Astra Control Center, and the Astra Plugin for VMware vSphere are all in different namespaces. The services in these different namespaces need to be able to communicate with one another. To enable this communication, follow these steps.

Steps

1. Delete any NetworkPolicy resources that exist in the Astra Control Center namespace:

```
kubectl get networkpolicy -n netapp-acc
```

2. For each NetworkPolicy object that is returned by the preceding command, use the following command to delete it. Replace <OBJECT_NAME> with the name of the returned object:

```
kubectl delete networkpolicy <OBJECT_NAME> -n netapp-acc
```

3. Apply the following resource file to configure the acc-avp-network-policy object to allow Astra Plugin for VMware vSphere services to make requests to Astra Control Center services. Replace the information in brackets <> with information from your environment:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: acc-avp-network-policy
  namespace: <ACC_NAMESPACE_NAME> # REPLACE THIS WITH THE ASTRA CONTROL
CENTER NAMESPACE NAME
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: <PLUGIN_NAMESPACE_NAME> #
REPLACE THIS WITH THE ASTRA PLUGIN FOR VMWARE VSPHERE NAMESPACE NAME
```

4. Apply the following resource file to configure the acc-operator-network-policy object to allow the Astra Control Center operator to communicate with Astra Control Center services. Replace the information in brackets <> with information from your environment:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: acc-operator-network-policy
  namespace: <ACC_NAMESPACE_NAME> # REPLACE THIS WITH THE ASTRA CONTROL
CENTER NAMESPACE NAME
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: <NETAPP-ACC-OPERATOR> #
REPLACE THIS WITH THE OPERATOR NAMESPACE NAME
```

Remove resource limitations

Some environments use the ResourceQuotas and LimitRanges objects to prevent the resources in a namespace from consuming all available CPU and memory on the cluster. Astra Control Center does not set maximum limits, so it will not be in compliance with those resources. You need to remove them from the namespaces where you plan to install Astra Control Center.

You can use the following steps to retrieve and remove these quotas and limits. In these examples, the command output is shown immediately after the command.

Steps

1. Get the resource quotas in the netapp-acc namespace:

```
kubectl get quota -n netapp-acc
```

Response:

```
NAME          AGE   REQUEST                                          LIMIT
pods-high     16s   requests.cpu: 0/20, requests.memory: 0/100Gi
limits.cpu: 0/200, limits.memory: 0/1000Gi
pods-low      15s   requests.cpu: 0/1, requests.memory: 0/1Gi
limits.cpu: 0/2, limits.memory: 0/2Gi
pods-medium   16s   requests.cpu: 0/10, requests.memory: 0/20Gi
limits.cpu: 0/20, limits.memory: 0/200Gi
```

2. Delete all of the resource quotas by name:

```
kubectl delete resourcequota  pods-high -n netapp-acc
```

```
kubectl delete resourcequota  pods-low -n netapp-acc
```

```
kubectl delete resourcequota  pods-medium -n netapp-acc
```

3. Get the limit ranges in the netapp-acc namespace:

```
kubectl get limits -n netapp-acc
```

Response:

```
NAME              CREATED AT
cpu-limit-range   2022-06-27T19:01:23Z
```

4. Delete the limit ranges by name:

```
kubectl delete limitrange cpu-limit-range -n netapp-acc
```

Copyright information

Trademark information