Install Astra Control Center

Astra Control Center

NetApp
February 12, 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. You can use this procedure to install Astra Control Center in internet-connected or air-gapped environments.

Expand for other installation procedures

- **Install with RedHat Openshift OperatorHub**: Use this alternative procedure to install Astra Control Center on Openshift using OperatorHub.
- **Install in the public cloud with Cloud Volumes ONTAP backend**: Use these procedures to install Astra Control Center in Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure with a Cloud Volumes ONTAP storage backend.

For a demonstration of the Astra Control Center installation process, see this video.

Before you begin

- 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. Refer to pod security restrictions.
- 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 attempt to install its own cert manager. By default, Astra Control Center installs its own cert manager during installation.

Deploy Astra Control Center in a third fault domain or secondary site. This is recommended for app replication and seamless disaster recovery.

Steps

To install Astra Control Center, do the following steps:

- Download and extract Astra Control Center
- 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

Do not delete the Astra Control Center operator (for example, `kubectl delete -f astra_control_center_operator_deploy.yaml`) at any time during Astra Control Center installation or operation to avoid deleting pods.

Download and extract Astra Control Center

1. Download the bundle containing Astra Control Center (`astra-control-center-[version].tar.gz`) from the the Astra Control Center downloads page.
2. (Recommended but optional) Download the certificates and signatures bundle for Astra Control Center (`astra-control-center-certs-[version].tar.gz`) to verify the signature of the bundle.

   Expand for details

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

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

   The output will show `Verified OK` after successful verification.

3. Extract the images from the Astra Control Center bundle:

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

Install the NetApp Astra kubectl plugin

You can use the NetApp Astra kubectl command line plugin to push images to a local Docker repository.

Before you begin

NetApp provides plugin binaries for different CPU architectures and operating systems. You need to know which CPU and operating system you have before you perform this task.

If you already have the plugin installed from a previous installation, make sure you have the latest version before completing these steps.

Steps

1. List the available NetApp Astra kubectl plugin binaries:

   The kubectl plugin library is part of the tar bundle and is extracted into the folder `kubectl-astra`.

   ```
   ls kubectl-astra/
   ```

2. Move the file you need for your operating system and CPU architecture into the current path and rename it to `kubectl-astra`:

   ```
   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

a. Change to the root directory of the tarball. You should see the `acc.manifest.bundle.yaml` file and these directories:

```
acc/
kubectl-astra/
acc.manifest.bundle.yaml
```

b. Push the package images in the Astra Control Center image directory to your local registry. Make the following substitutions before running the `push-images` command:

- Replace <BUNDLE_FILE> with the name of the Astra Control bundle file (`acc.manifest.bundle.yaml`).
- Replace <MY_FULL_REGISTRY_PATH> with the URL of the Docker repository; for example, "https://<docker-registry>".
- 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_FULL_REGISTRY_PATH> -u <MY_REGISTRY_USER> -p
<MY_REGISTRY_TOKEN>
```

Podman

a. Change to the root directory of the tarball. You should see this file and directory:

```
acc.manifest.bundle.yaml
acc/
```

b. Log in to your registry:

```
podman login <YOUR_REGISTRY>
```

c. Prepare and run one of the following scripts that is customized for the version of Podman you use. Substitute <MY_FULL_REGISTRY_PATH> with the URL of your repository that includes any sub-directories.

```
<strong>Podman 4</strong>
```

```
export REGISTRY=<MY_FULL_REGISTRY_PATH>
export PACKAGENAME=acc
export PACKAGEVERSION=23.07.0-25
export DIRECTORYNAME=acc
for astraImageFile in $(ls ${DIRECTORYNAME}/images/*.tar) ; do
astraImage=$(podman load --input ${astraImageFile} | sed 's/Loaded
image: //')
astraImageNoPath=$(echo ${astraImage} | sed 's:.*/::')
podman tag ${astraImageNoPath} ${REGISTRY}/netapp/astra/
${PACKAGENAME}/${PACKAGEVERSION}/${astraImageNoPath}
podman push ${REGISTRY}/netapp/astra/${PACKAGENAME}/${
PACKAGEVERSION}/${astraImageNoPath}
done
```

```
<strong>Podman 3</strong>
```

```
export REGISTRY=<MY_FULL_REGISTRY_PATH>
export PACKAGENAME=acc
export PACKAGEVERSION=23.07.0-25
export DIRECTORYNAME=acc
for astraImageFile in $(ls ${DIRECTORYNAME}/images/*.tar) ; do
astraImage=$(podman load --input ${astraImageFile} | sed 's/Loaded
image: //')
astraImageNoPath=$(echo ${astraImage} | sed 's:.*/::')
podman tag ${astraImageNoPath} ${REGISTRY}/netapp/astra/
${PACKAGENAME}/${PACKAGEVERSION}/${astraImageNoPath}
podman push ${REGISTRY}/netapp/astra/${PACKAGENAME}/${
PACKAGEVERSION}/${astraImageNoPath}
done
```

The image path the script creates should resemble the following, depending on your registry configuration:

```
https://netappdownloads.jfrog.io/docker-astra-control-
prod/netapp/astra/acc/23.07.0-25/image:version
```

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]
```

> Before you complete the installation, be sure your kubeconfig is pointing to the cluster where you want to install Astra Control Center.

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

Expand for steps

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

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

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/23.07.0-25`).

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

> If you delete the namespace after the secret is generated, recreate the namespace and then regenerate the secret for the namespace.

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

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

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]
```

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 `ASTRA_IMAGE_REGISTRY` for the `kube-rbac-proxy` image to the registry path where you pushed the images in a previous step.

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

Expand for sample astra_control_center_operator_deploy.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
  strategy:
    type: Recreate
  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: ASTRA_IMAGE_REGISTRY/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
        env:
        - name: ACCOP_LOG_LEVEL
          value: "2"
        - name: ACCOP_HELM_INSTALLTIMEOUT
          value: 5m
        image: ASTRA_IMAGE_REGISTRY/acc-operator:23.07.25
        imagePullPolicy: IfNotPresent
        livenessProbe:
          httpGet:
            path: /healthz
```

```
            port: 8081
          initialDelaySeconds: 15
          periodSeconds: 20
        name: manager
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8081
          initialDelaySeconds: 5
          periodSeconds: 10
        resources:
          limits:
            cpu: 300m
            memory: 750Mi
          requests:
            cpu: 100m
            memory: 75Mi
        securityContext:
          allowPrivilegeEscalation: false
      imagePullSecrets: []
      securityContext:
        runAsUser: 65532
      terminationGracePeriodSeconds: 10
```

3. Install the Astra Control Center operator:

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

Expand for sample response:

```
namespace/netapp-acc-operator created
customresourcedefinition.apiextensions.k8s.io/astracontrolcenters.as
tra.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.yaml`) to make account, support, registry, and other necessary configurations:

```
vim astra_control_center.yaml
```

An annotated sample YAML follows these steps.

2. Modify or confirm the following settings:

`accountName`

Setting	Guidance	Type	Example
`accountName`	Change the `accountName` string to the name you want to associate with the Astra Control Center account. There can be only one accountName.	string	`Example`

`astraVersion`

Setting	Guidance	Type	Example
`astraVersion`	The version of Astra Control Center to deploy. No action is needed for this setting as the value will be pre-populated.	string	`23.07.0-25`

`astraAddress`

Setting	Guidance	Type	Example
`astraAddress`	Change the `astraAddress` string to the FQDN (recommended) or IP address you want to use in your browser to access Astra Control Center. This address defines how Astra Control Center will be found in your data center and is the same FQDN or IP address you provisioned from your load balancer when you completed Astra Control Center requirements. NOTE: Do not use `http://` or `https://` in the address. Copy this FQDN for use in a later step.	string	`astra.example.com`

`autoSupport`

Your selections in this section determine whether you will participate in NetApp's pro-active support application, NetApp Active IQ, and where data is sent. An internet connection is required (port 442), and all support data is anonymized.

Setting	Use	Guidance	Type	Example
`autoSupport.enrolled`	Either `enrolled` or `url` fields must be selected	Change `enrolled` for AutoSupport to `false` for sites without internet connectivity or retain `true` for connected sites. A setting of `true` enables anonymous data to be sent to NetApp for support purposes. The default election is `false` and indicates no support data will be sent to NetApp.	Boolean	`false` (this value is the default)
`autoSupport.url`	Either `enrolled` or `url` fields must be selected	This URL determines where the anonymous data will be sent.	string	`https://support.netapp.com/asupprod/post/1.0/postAsup`

`email`

Setting	Guidance	Type	Example
`email`	Change the `email` string to the default initial administrator address. Copy this email address for use in a later step. This email address will be used as the username for the initial account to log in to the UI and will be notified of events in Astra Control.	string	`admin@example.com`

`firstName`

Setting	Guidance	Type	Example
`firstName`	The first name of the default initial administrator associated with the Astra account. The name used here will be visible in a heading in the UI after your first login.	string	`SRE`

`LastName`

Setting	Guidance	Type	Example
`lastName`	The last name of the default initial administrator associated with the Astra account. The name used here will be visible in a heading in the UI after your first login.	string	`Admin`

`imageRegistry`

Your selections in this section define the container image registry that is hosting the Astra application images, Astra Control Center Operator, and Astra Control Center Helm repository.

Setting	Use	Guidance	Type	Example
`imageRegistry.name`	Required	The name of the image registry where you pushed the images in the previous step. Do not use `http://` or `https://` in the registry name.	string	`example.registry.com/astra`
`imageRegistry.secret`	Required if the string you entered for `imageRegistry.name' requires a secret.` `IMPORTANT: If you are using a registry that does not require authorization, you must delete this` `secret` line within `imageRegistry` or the installation will fail.	The name of the Kubernetes secret used to authenticate with the image registry.	string	`astra-registry-cred`

`storageClass`

Setting	Guidance	Type	Example
`storageClass`	Change the `storageClass` value from `ontap-gold` to another Astra Trident storageClass resource as required by your installation. Run the command `kubectl get sc` to determine your existing configured storage classes. One of the Astra Trident-based storage classes must be entered in the manifest file (`astra-control-center-<version>.manifest`) and will be used for Astra PVs. If it is not set, the default storage class will be used. NOTE: If a default storage class is configured, ensure that it is the only storage class that has the default annotation.	string	`ontap-gold`

`volumeReclaimPolicy`

Setting	Guidance	Type	Options
`volumeReclaimPolicy`	This sets the reclaim policy for Astra's PVs. Setting this policy to `Retain` retains persistent volumes after Astra is deleted. Setting this policy to `Delete` deletes persistent volumes after astra is deleted. If this value is not set, the PVs are retained.	string	• `Retain` (This is the default value) • `Delete`

ingressType

Setting	Guidance	Type	Options
`ingressType`	Use one of the following ingress types: **`Generic`** (`ingressType: "Generic"`) (Default) Use this option when you have another ingress controller in use or would prefer to use your own ingress controller. After Astra Control Center is deployed, you will need to configure the ingress controller to expose Astra Control Center with a URL. **`AccTraefik`** (`ingressType: "AccTraefik"`) Use this option when you would prefer not to configure an ingress controller. 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	string	• `Generic` (this is the default value) • `AccTraefik`

`scaleSize`

Setting	Guidance	Type	Options
`scaleSize`	By default, Astra will use High Availability (HA) `scaleSize` of `Medium`, which deploys most services in HA and deploys multiple replicas for redundancy. With `scaleSize` as `Small`, Astra will reduce the number of replicas for all services except for essential services to reduce consumption. TIP: `Medium` deployments consist of around 100 pods (not including transient workloads. 100 pods is based on a three master node and three worker node configuration). Be aware of per-pod network limit constraints that might be an issue in your environment, especially when considering disaster recovery scenarios.	string	• `Small` • `Medium` (This is the default value)

`astraResourcesScaler`

Setting	Guidance	Type	Options
`astraResourcesScaler`	Scaling options for AstraControlCenter Resource limits. By default, Astra Control Center deploys with resource requests set for most of the components within Astra. This configuration allows the Astra Control Center software stack to perform better in environments under increased application load and scale. However, in scenarios using smaller development or test clusters, the CR field `astraResourcesScalar` may be set to `Off`. This disables resource requests and allows for deployment on smaller clusters.	string	• `Default` (This is the default value) • `Off`

`additionalValues`

Add the following additional values to the Astra Control Center CR to prevent a known issue in the 23.07 installation:

```
additionalValues:
    polaris-keycloak:
      livenessProbe:
        initialDelaySeconds: 180
      readinessProbe:
        initialDelaySeconds: 180
```

- For Astral Control Center and Cloud Insights communication, TLS certificate verification is disabled by default. You can enable TLS certification verification for communication between Cloud Insights and both the Astra Control Center host cluster and managed cluster by adding the following section in `additionalValues`.

```
  additionalValues:
    netapp-monitoring-operator:
      config:
        ciSkipTlsVerify: false
    cloud-insights-service:
      config:
        ciSkipTlsVerify: false
    telemetry-service:
      config:
        ciSkipTlsVerify: false
```

`crds`

Your selections in this section determine how Astra Control Center should handle CRDs.

Setting	Guidance	Type	Example
`crds.externalCertManager`	If you use an external cert manager, change `externalCertManager` to `true`. The default `false` causes Astra Control Center to install its own cert manager CRDs during installation. CRDs are cluster-wide objects and installing them might have an impact on other parts of the cluster. You can use this flag to signal to Astra Control Center that these CRDs will be installed and managed by the cluster administrator outside of Astra Control Center.	Boolean	`False` (this value is the default)
`crds.externalTraefik`	By default, Astra Control Center will install required Traefik CRDs. CRDs are cluster-wide objects and installing them might have an impact on other parts of the cluster. You can use this flag to signal to Astra Control Center that these CRDs will be installed and managed by the cluster administrator outside of Astra Control Center.	Boolean	`False` (this value is the default)

Be sure that you have selected the correct storage class and ingress type for your configuration before completing installation.

Expand for sample astra_control_center.yaml

```
apiVersion: astra.netapp.io/v1
kind: AstraControlCenter
metadata:
  name: astra
spec:
  accountName: "Example"
  astraVersion: "ASTRA_VERSION"
  astraAddress: "astra.example.com"
  autoSupport:
    enrolled: true
  email: "[admin@example.com]"
  firstName: "SRE"
  lastName: "Admin"
  imageRegistry:
    name: "[your_registry_path]"
    secret: "astra-registry-cred"
  storageClass: "ontap-gold"
  volumeReclaimPolicy: "Retain"
  ingressType: "Generic"
  scaleSize: "Medium"
  astraResourcesScaler: "Default"
  additionalValues:
    polaris-keycloak:
      livenessProbe:
        initialDelaySeconds: 180
      readinessProbe:
        initialDelaySeconds: 180
  crds:
    externalTraefik: false
    externalCertManager: false
```

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]
```

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

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

The Astra Control Center operator will run an automatic check for environment requirements. Missing requirements can cause your installation to fail or Astra Control Center to not operate properly. See the next section to check for warning messages related to the automatic system check.

Verify system status

You can verify system status using kubectl commands. If you prefer to use OpenShift, you can use comparable oc commands for verification steps.

Steps

1. Verify that the installation process did not produce warnings messages related to the validation checks:

```
kubectl get acc [astra or custom Astra Control Center CR name] -n
[netapp-acc or custom namespace] -o yaml
```

Additional warning messages are also reported in the Astra Control Center operator logs.

2. Correct any issues with your environment that were reported by the automated requirements checks.

You can correct issues by ensuring that your environment meets the requirements for Astra Control Center.

3. 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.

Expand for sample response

```
NAME                                          READY   STATUS
RESTARTS     AGE
acc-helm-repo-6cc7696d8f-pmhm8                1/1     Running     0
9h
activity-597fb656dc-5rd4l                     1/1     Running     0
9h
activity-597fb656dc-mqmcw                     1/1     Running     0
9h
api-token-authentication-62f84                1/1     Running     0
9h
api-token-authentication-68nlf                1/1     Running     0
9h
api-token-authentication-ztgrm                1/1     Running     0
9h
asup-669d4ddbc4-fnmwp                         1/1     Running     1
(9h ago)   9h
authentication-78789d7549-lk686               1/1     Running     0
9h
bucketservice-65c7d95496-24x7l                1/1     Running     3
(9h ago)   9h
cert-manager-c9f9fbf9f-k8zq2                  1/1     Running     0
9h
cert-manager-c9f9fbf9f-qjlzm                  1/1     Running     0
9h
cert-manager-cainjector-dbbbd8447-b5qll       1/1     Running     0
9h
cert-manager-cainjector-dbbbd8447-p5whs       1/1     Running     0
9h
cert-manager-webhook-6f97bb7d84-4722b         1/1     Running     0
9h
cert-manager-webhook-6f97bb7d84-86kv5         1/1     Running     0
9h
certificates-59d9f6f4bd-2j899                 1/1     Running     0
9h
certificates-59d9f6f4bd-9d9k6                 1/1     Running     0
9h
certificates-expiry-check-28011180--1-8lkxz   0/1     Completed   0
9h
cloud-extension-5c9c9958f8-jdhrp              1/1     Running     0
9h
cloud-insights-service-5cdd5f7f-pp8r5         1/1     Running     0
9h
composite-compute-66585789f4-hxn5w            1/1     Running     0
9h
```

```
composite-volume-68649f68fd-tb7p4             1/1     Running     0
9h
credentials-dfc844c57-jsx92                   1/1     Running     0
9h
credentials-dfc844c57-xw26s                   1/1     Running     0
9h
entitlement-7b47769b87-4jb6c                  1/1     Running     0
9h
features-854d8444cc-c24b7                     1/1     Running     0
9h
features-854d8444cc-dv6sm                     1/1     Running     0
9h
fluent-bit-ds-9tlv4                           1/1     Running     0
9h
fluent-bit-ds-bpkcb                           1/1     Running     0
9h
fluent-bit-ds-cxmwx                           1/1     Running     0
9h
fluent-bit-ds-jgnhc                           1/1     Running     0
9h
fluent-bit-ds-vtr6k                           1/1     Running     0
9h
fluent-bit-ds-vxqd5                           1/1     Running     0
9h
graphql-server-7d4b9d44d5-zdbf5               1/1     Running     0
9h
identity-6655c48769-4pwk8                     1/1     Running     0
9h
influxdb2-0                                   1/1     Running     0
9h
keycloak-operator-55479d6fc6-slvmt            1/1     Running     0
9h
krakend-f487cb465-78679                       1/1     Running     0
9h
krakend-f487cb465-rjsxx                       1/1     Running     0
9h
license-64cbc7cd9c-qxsr8                      1/1     Running     0
9h
login-ui-5db89b5589-ndb96                     1/1     Running     0
9h
loki-0                                        1/1     Running     0
9h
metrics-facade-8446f64c94-x8h7b               1/1     Running     0
9h
monitoring-operator-6b44586965-pvcl4          2/2     Running     0
9h
```

```
nats-0                                          1/1     Running     0
9h
nats-1                                          1/1     Running     0
9h
nats-2                                          1/1     Running     0
9h
nautilus-85754d87d7-756qb                       1/1     Running     0
9h
nautilus-85754d87d7-q8j7d                       1/1     Running     0
9h
openapi-5f9cc76544-7fnjm                        1/1     Running     0
9h
openapi-5f9cc76544-vzr7b                        1/1     Running     0
9h
packages-5db49f8b5-lrzhd                        1/1     Running     0
9h
polaris-consul-consul-server-0                  1/1     Running     0
9h
polaris-consul-consul-server-1                  1/1     Running     0
9h
polaris-consul-consul-server-2                  1/1     Running     0
9h
polaris-keycloak-0                              1/1     Running     2
(9h ago)   9h
polaris-keycloak-1                              1/1     Running     0
9h
polaris-keycloak-2                              1/1     Running     0
9h
polaris-keycloak-db-0                           1/1     Running     0
9h
polaris-keycloak-db-1                           1/1     Running     0
9h
polaris-keycloak-db-2                           1/1     Running     0
9h
polaris-mongodb-0                               1/1     Running     0
9h
polaris-mongodb-1                               1/1     Running     0
9h
polaris-mongodb-2                               1/1     Running     0
9h
polaris-ui-66fb99479-qp9gq                      1/1     Running     0
9h
polaris-vault-0                                 1/1     Running     0
9h
polaris-vault-1                                 1/1     Running     0
9h
```

```
polaris-vault-2                                1/1     Running     0
9h
public-metrics-76fbf9594d-zmxzw                1/1     Running     0
9h
storage-backend-metrics-7d7fbc9cb9-lmd25       1/1     Running     0
9h
storage-provider-5bdd456c4b-2fftc              1/1     Running     0
9h
task-service-87575df85-dnn2q                   1/1     Running     3
(9h ago)   9h
task-service-task-purge-28011720--1-q6w4r      0/1     Completed   0
28m
task-service-task-purge-28011735--1-vk6pd      1/1     Running     0
13m
telegraf-ds-2r2kw                              1/1     Running     0
9h
telegraf-ds-6s9d5                              1/1     Running     0
9h
telegraf-ds-96jl7                              1/1     Running     0
9h
telegraf-ds-hbp84                              1/1     Running     0
9h
telegraf-ds-plwzv                              1/1     Running     0
9h
telegraf-ds-sr22c                              1/1     Running     0
9h
telegraf-rs-4sbg8                              1/1     Running     0
9h
telemetry-service-fb9559f7b-mk9l7              1/1     Running     3
(9h ago)   9h
tenancy-559bbc6b48-5msgg                       1/1     Running     0
9h
traefik-d997b8877-7xpf4                        1/1     Running     0
9h
traefik-d997b8877-9xv96                        1/1     Running     0
9h
trident-svc-585c97548c-d25z5                   1/1     Running     0
9h
vault-controller-88484b454-2d6sr               1/1     Running     0
9h
vault-controller-88484b454-fc5cz               1/1     Running     0
9h
vault-controller-88484b454-jktld               1/1     Running     0
9h
```

4. (Optional) Watch the `acc-operator` logs to monitor progress:

```
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.

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

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

Response:

```
NAME    UUID                                   VERSION     ADDRESS
READY
astra   9aa5fdae-4214-4cb7-9976-5d8b4c0ce27f  23.07.0-25
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. These procedures give setup examples for an ingress controller if you used the default of `ingressType: "Generic"` in the Astra Control Center custom resource (`astra_control_center.yaml`). You do not need to use this procedure if you specified `ingressType: "AccTraefik"` in the Astra Control Center custom resource (`astra_control_center.yaml`).

After Astra Control Center is deployed, you will need to configure the ingress controller to expose Astra Control Center with a URL.

Setup steps differ depending on the type of ingress controller you use. Astra Control Center supports many ingress controller types. These setup procedures provide example steps for some common ingress controller types.

Before you begin

- The required ingress controller should already be deployed.
- The ingress class corresponding to the ingress controller should already be created.

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 the `netapp-acc` (or custom-named) namespace using the v1 resource type for a schema (`istio-Ingress.yaml` is used in this example):

```
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: [netapp-acc or custom namespace]
spec:
  ingressClassName: istio
  tls:
  - hosts:
    - <ACC address>
    secretName: [tls secret name]
  rules:
  - host: [ACC address]
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: traefik
            port:
              number: 80
```

5. Apply the changes:

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

6. Check the status of the ingress:

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

Response:

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

7. Finish Astra Control Center installation.

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 the v1 resource type for a schema (`nginx-Ingress.yaml` is used in this example):

```
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 address>
    http:
      paths:
        - path:
          backend:
            service:
              name: traefik
              port:
                number: 80
          pathType: ImplementationSpecific
```

3. Apply the changes:

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

NetApp recommends installing the nginx controller as a deployment rather than a `daemonSet`.

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 (including the `https://` prefix) you used in the `astraAddress` in the `astra_control_center.yaml` CR when you installed Astra Control Center.
2. Accept the self-signed certificates if 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.yaml` CR when you installed Astra Control Center, followed by the initial setup 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.

Options

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

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

- To check the output of the Astra Control Center CR:

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

What's next

- (Optional) Depending on your environment, complete post-installation configuration steps.
- Complete the deployment by performing setup tasks.

Configure an external cert manager

If a cert manager already exists in your Kubernetes cluster, you need to perform some prerequisite steps so that Astra Control Center does not install its own cert manager.

Steps

1. Confirm that you have a cert manager installed:

```
kubectl get pods -A | grep 'cert-manager'
```

Sample response:

```
cert-manager   essential-cert-manager-84446f49d5-sf2zd              1/1
Running    0      6d5h
cert-manager   essential-cert-manager-cainjector-66dc99cc56-9ldmt   1/1
Running    0      6d5h
cert-manager   essential-cert-manager-webhook-56b76db9cc-fjqrq      1/1
Running    0      6d5h
```

2. Create a certificate/key pair for the `astraAddress` FQDN:

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

Sample response:

```
Generating a 2048 bit RSA private key
...................+++
.......................................+++
writing new private key to 'tls.key'
```

3. Create a secret with previously generated files:

```
kubectl create secret tls selfsigned-tls --key tls.key --cert tls.crt -n
<cert-manager-namespace>
```

Sample response:

```
secret/selfsigned-tls created
```

4. Create a `ClusterIssuer` file that is **exactly** the following but includes the namespace location where your `cert-manager` pods are installed:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: astra-ca-clusterissuer
  namespace: <cert-manager-namespace>
spec:
  ca:
    secretName: selfsigned-tls
```

```
kubectl apply -f ClusterIssuer.yaml
```

Sample response:

```
clusterissuer.cert-manager.io/astra-ca-clusterissuer created
```

5. Verify that the `ClusterIssuer` has come up correctly. `Ready` must be `True` before you can proceed:

```
kubectl get ClusterIssuer
```

Sample response:

```
NAME                        READY   AGE
astra-ca-clusterissuer      True    9s
```

6. Complete the Astra Control Center installation process. There is a required configuration step for the Astra Control Center cluster YAML in which you change the CRD value to indicate that the cert manager is externally installed. You must complete this step during installation so that Astra Control Center recognizes the external cert manager.

Copyright information

Trademark information