Get started

Astra Trident

NetApp
June 27, 2022

Table of Contents

Get started

Try it out

NetApp provides a ready-to-use lab image that you can request through NetApp Test Drive. The Test Drive provides you with a sandbox environment that comes with a three-node Kubernetes cluster and Astra Trident installed and configured. It is a great way to familiarize yourself with Astra Trident and explore its features.

Another option is to see the kubeadm Install Guide provided by Kubernetes.

> You should not use the Kubernetes cluster that you build using these instructions in production. Use the production deployment guides provided by your distribution for creating clusters that are production ready.

If this is the first time you're using Kubernetes, familiarize yourself with the concepts and tools here.

Requirements

Get started by reviewing the supported frontends, backends, and host configuration.

> To learn about the ports that Astra Trident uses, see here.

Supported frontends (orchestrators)

Astra Trident supports multiple container engines and orchestrators, including the following:

- Kubernetes 1.17 or later (latest: 1.22)
- Mirantis Kubernetes Engine 3.4
- OpenShift OpenShift 4.7, 4.8, 4.9 (latest 4.9)

The Trident operator is supported with these releases:

- Kubernetes 1.17 or later (latest: 1.22)
- OpenShift OpenShift 4.7, 4.8, 4.9 (latest 4.9)

> Red Hat OpenShift Container Platform users might observe their initiatorname.iscsi file to be blank if using any version below 4.6.8. This is a bug that has been identified by RedHat to be fixed with OpenShift 4.6.8. See this bug fix announcement. NetApp recommends that you use Astra Trident on OpenShift 4.7 and later.

Astra Trident also works with a host of other fully managed and self-managed Kubernetes offerings, including Google Cloud's Google Kubernetes Engine (GKE), AWS's Elastic Kubernetes Services (EKS), Azure's Azure Kubernetes Service (AKS), and Rancher.

Supported backends (storage)

To use Astra Trident, you need one or more of the following supported backends:

- Amazon FSx for NetApp ONTAP

- Azure NetApp Files
- Astra Data Store
- Cloud Volumes ONTAP
- Cloud Volumes Service for AWS
- Cloud Volumes Service for GCP
- FAS/AFF/Select 9.3 or later
- NetApp All SAN Array (ASA)
- NetApp HCI/Element software 8 or later

Feature requirements

The table below summarizes the features available with this release of Astra Trident and the versions of Kubernetes it supports.

Feature	Kubernetes version	Feature gates required?
CSI Trident	1.17 and later	No
Volume Snapshots	1.17 and later	No
PVC from Volume Snapshots	1.17 and later	No
iSCSI PV resize	1.17 and later	No
ONTAP Bidirectional CHAP	1.17 and later	No
Dynamic Export Policies	1.17 and later	No
Trident Operator	1.17 and later	No
Auto Worker Node Prep (beta)	1.17 and later	No
CSI Topology	1.17 and later	No

Tested host operating systems

By default, Astra Trident runs in a container and will, therefore, run on any Linux worker. However, those workers need to be able to mount the volumes that Astra Trident provides using the standard NFS client or iSCSI initiator, depending on the backends you are using.

Though Astra Trident does not officially "support" specific operating systems, the following Linux distributions are known to work:

- RedHat CoreOS 4.2 and 4.3
- RHEL or CentOS 7.4 or later

- Ubuntu 18.04 or later

The `tridentctl` utility also runs on any of these distributions of Linux.

Host configuration

Depending on the backend(s) in use, NFS and/or iSCSI utilities should be installed on all of the workers in the cluster. See here for more information.

Storage system configuration

Astra Trident might require some changes to a storage system before a backend configuration can use it. See here for details.

Container images and corresponding Kubernetes versions

For air-gapped installations, the following list is a reference of container images needed to install Astra Trident. Use the `tridentctl images` command to verify the list of needed container images.

Kubernetes version	Container image
v1.17.0	• netapp/trident:21.10.0 • netapp/trident-autosupport:21.10 • k8s.gcr.io/sig-storage/csi-provisioner:v2.2.2 • k8s.gcr.io/sig-storage/csi-attacher:v3.3.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.3.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v3.0.3 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.3.0 • netapp/trident-operator:21.10.0 (optional)
v1.18.0	• netapp/trident:21.10.0 • netapp/trident-autosupport:21.10 • k8s.gcr.io/sig-storage/csi-provisioner:v2.2.2 • k8s.gcr.io/sig-storage/csi-attacher:v3.3.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.3.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v3.0.3 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.3.0 • netapp/trident-operator:21.10.0 (optional)

Kubernetes version	Container image
v1.19.0	• netapp/trident:21.10.0 • netapp/trident-autosupport:21.10 • k8s.gcr.io/sig-storage/csi-provisioner:v2.2.2 • k8s.gcr.io/sig-storage/csi-attacher:v3.3.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.3.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v3.0.3 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.3.0 • netapp/trident-operator:21.10.0 (optional)
v1.20.0	• netapp/trident:21.10.0 • netapp/trident-autosupport:21.10 • k8s.gcr.io/sig-storage/csi-provisioner:v3.0.0 • k8s.gcr.io/sig-storage/csi-attacher:v3.3.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.3.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v3.0.3 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.3.0 • netapp/trident-operator:21.10.0 (optional)
v1.21.0	• netapp/trident:21.10.0 • netapp/trident-autosupport:21.10 • k8s.gcr.io/sig-storage/csi-provisioner:v3.0.0 • k8s.gcr.io/sig-storage/csi-attacher:v3.3.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.3.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v3.0.3 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.3.0 • netapp/trident-operator:21.10.0 (optional)

Kubernetes version	Container image
v1.22.0	• netapp/trident:21.10.0 • netapp/trident-autosupport:21.10 • k8s.gcr.io/sig-storage/csi-provisioner:v3.0.0 • k8s.gcr.io/sig-storage/csi-attacher:v3.3.0 • k8s.gcr.io/sig-storage/csi-resizer:v1.3.0 • k8s.gcr.io/sig-storage/csi-snapshotter:v3.0.3 • k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.3.0 • netapp/trident-operator:21.10.0 (optional)

On Kubernetes version 1.20 and later, use the validated `k8s.gcr.io/sig-storage/csi-snapshotter:v4.x` image only if the `v1` version is serving the `volumesnapshots.snapshot.storage.k8s.io` CRD. If the `v1beta1` version is serving the CRD with/without the `v1` version, use the validated `k8s.gcr.io/sig-storage/csi-snapshotter:v3.x` image.

Deployment overview

You can deploy Astra Trident using the Trident operator or with `tridentctl`.

Choose the deployment method

To determine which deployment method to use, consider the following:

Why should I use the Trident operator?

The Trident operator is a great way to dynamically manage Astra Trident resources and automate the setup phase. There are some prerequisites that must be satisfied. See the requirements.

The Trident operator provides several benefits as outlined below.

Self-healing capability

You can monitor an Astra Trident installation and actively take measures to address issues, such as when the deployment is deleted or if it is modified accidentally. When the operator is set up as a deployment, a `trident-operator-<generated-id>` pod is created. This pod associates a `TridentOrchestrator` CR with an Astra Trident installation and always ensures there is only one active `TridentOrchestrator`. In other words, the operator ensures that there is only one instance of Astra Trident in the cluster and controls its setup, making sure the installation is idempotent. When changes are made to the installation (such as, deleting the deployment or node daemonset), the operator identifies them and fixes them individually.

Easy updates to existing installations

You can easily update an existing deployment with the operator. You only need to edit the `TridentOrchestrator` CR to make updates to an installation.
For example, consider a scenario where you need to enable Astra Trident to generate debug logs.

To do this, patch your `TridentOrchestrator` to set `spec.debug` to `true`:

```
kubectl patch torc <trident-orchestrator-name> -n trident --type=merge -p
'{"spec":{"debug":true}}'
```

After `TridentOrchestrator` is updated, the operator processes the updates and patches the existing installation. This might triggers the creation of new pods to modify the installation accordingly.

Automatically handles Kubernetes upgrades

When the Kubernetes version of the cluster is upgraded to a supported version, the operator updates an existing Astra Trident installation automatically and changes it to ensure that it meets the requirements of the Kubernetes version.

> If the cluster is upgraded to an unsupported version, the operator prevents installing Astra Trident. If Astra Trident has already been installed with the operator, a warning is displayed to indicate that Astra Trident is installed on an unsupported Kubernetes version.

Why should I use Helm?

If you have other applications that you are managing using Helm, starting with Astra Trident 21.01, you can manage your deployment also using Helm.

When should I use `tridenctl`?

If you have an existing deployment that must be upgraded to or if you are looking to highly customize your deployment, you should take a look at using tridentctl. This is the conventional method of deploying Astra Trident.

Considerations for moving between deployment methods

It is not hard to imagine a scenario where moving between deployment methods is desired. You should consider the following before attempting to move from a `tridentctl` deployment to an operator-based deployment, or vice-versa:

- Always use the same method for uninstalling Astra Trident. If you have deployed with `tridentctl`, you should use the appropriate version of the `tridentctl` binary to uninstall Astra Trident. Similarly, if you are deploying with the operator, you should edit the `TridentOrchestrator` CR and set `spec.uninstall=true` to uninstall Astra Trident.
- If you have an operator-based deployment that you want to remove and use `tridentctl` to deploy Astra Trident, you should first edit `TridentOrchestrator` and set `spec.uninstall=true` to uninstall Astra Trident. Then delete `TridentOrchestrator` and the operator deployment. You can then install using `tridentctl`.
- If you have a manual operator-based deployment, and you want to use Helm-based Trident operator deployment, you should manually uninstall the operator first, and then do the Helm install. This enables Helm to deploy the Trident operator with the required labels and annotations. If you do not do this, your Helm-based Trident operator deployment will fail with label validation error and annotation validation error. If you have a `tridentctl`-based deployment, you can use Helm-based deployment without running into issues.

Understand the deployment modes

There are three ways to deploy Astra Trident.

Standard deployment

Deploying Trident on a Kubernetes cluster results in the Astra Trident installer doing two things:

- Fetching the container images over the Internet
- Creating a deployment and/or node daemonset, which spins up Astra Trident pods on all the eligible nodes in the Kubernetes cluster.

A standard deployment such as this can be performed in two different ways:

- Using `tridentctl install`
- Using the Trident operator. You can deploy Trident operator either manually or by using Helm.

This mode of installing is the easiest way to install Astra Trident and works for most environments that do not impose network restrictions.

Offline deployment

To perform an air-gapped deployment, you can use the `--image-registry` flag when invoking `tridentctl install` to point to a private image registry. If deploying with the Trident operator, you can alternatively specify `spec.imageRegistry` in your `TridentOrchestrator`. This registry should contain the Trident image, the Trident Autosupport image, and the CSI sidecar images as required by your Kubernetes version.

To customize your deployment, you can use `tridentctl` to generate the manifests for Trident's resources. This includes the deployment, daemonset, service account, and the cluster role that Astra Trident creates as part of its installation.

See these links for more information about customizing your deployment:

- Customize your operator-based deployment
- Customize your `tridentctl`-based deployment

> If you are using a private image repository, you should add `/k8scsi` for Kubernetes versions earlier than 1.17 or `/sig-storage` for Kubernetes versions later than 1.17 to the end of the private registry URL. When using a private registry for `tridentctl` deployment, you should use `--trident-image` and `--autosupport-image` in conjunction with `--image-registry`. If you are deploying Astra Trident by using the Trident operator, ensure that the orchestrator CR includes `tridentImage` and `autosupportImage` in the installation parameters.

Remote deployment

Here is a high-level overview of the remote deployment process:

- Deploy the appropriate version of `kubectl` on the remote machine from where you want to deploy Astra Trident.
- Copy the configuration files from the Kubernetes cluster and set the `KUBECONFIG` environment variable on the remote machine.

- Initiate a `kubectl get nodes` command to verify that you can connect to the required Kubernetes cluster.
- Complete the deployment from the remote machine by using the standard installation steps.

Deploy with Trident operator

You can deploy Astra Trident with the Trident operator. You can deploy the Trident operator either manually or using Helm.

If you have not already familiarized yourself with the basic concepts, now is a great time to do that.

What you'll need

To deploy Astra Trident, the following prerequisites should be met:

- You have full privileges to a supported Kubernetes cluster running Kubernetes 1.17 and above.
- You have access to a supported NetApp storage system.
- You have the capability to mount volumes from all of the Kubernetes worker nodes.
- You have a Linux host with `kubectl` (or `oc`, if you are using OpenShift) installed and configured to manage the Kubernetes cluster that you want to use.
- You have set the `KUBECONFIG` environment variable to point to your Kubernetes cluster configuration.
- You have enabled the feature gates required by Astra Trident.
- If you are using Kubernetes with Docker Enterprise, follow their steps to enable CLI access.

Got all that? Great! Let's get started.

Deploy the Trident operator by using Helm

Perform the steps listed to deploy the Trident operator by using Helm.

What you'll need

In addition to the prerequisites listed above, to deploy Trident operator by using Helm, you need the following:

- Kubernetes 1.17 and later
- Helm version 3

Steps

1. Download the installer bundle from the Trident GitHub page. The installer bundle includes the Helm chart in the `/helm` directory.
2. Use the `helm install` command and specify a name for your deployment.
   See the following example:

   ```
   helm install <name> trident-operator-21.07.1.tgz --namespace <namespace
   you want to use for Trident>
   ```

   If you did not already create a namespace for Trident, you can add the `--create-namespace` parameter

to the `helm install` command. Helm will then automatically create the namespace for you.

There are two ways to pass configuration data during the install:

- `--values` (or `-f`): Specify a YAML file with overrides. This can be specified multiple times and the rightmost file will take precedence.
- `--set`: Specify overrides on the command line.

For example, to change the default value of `debug`, run the following `--set` command:

```
$ helm install <name> trident-operator-21.07.1.tgz --set tridentDebug=true
```

The `values.yaml` file, which is part of the Helm chart provides the list of keys and their default values.

`helm list` shows you details about the installation, such as name, namespace, chart, status, app version, revision number, and so on.

Deploy the Trident operator manually

Perform the steps listed to manually deploy the Trident operator.

Step 1: Qualify your Kubernetes cluster

The first thing you need to do is log in to the Linux host and verify that it is managing a *working*, supported Kubernetes cluster that you have the necessary privileges to.

> With OpenShift, use `oc` instead of `kubectl` in all of the examples that follow, and log in as **system:admin** first by running `oc login -u system:admin` or `oc login -u kube-admin`.

To see if your Kubernetes version is later than 1.17, run the following command:

```
kubectl version
```

To see if you have Kubernetes cluster administrator privileges, run the following command:

```
kubectl auth can-i '*' '*' --all-namespaces
```

To verify if you can launch a pod that uses an image from Docker Hub and reach your storage system over the pod network, run the following command:

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
  ping <management IP>
```

Step 2: Download and set up the operator

> Beginning with 21.01, the Trident operator is cluster scoped. Using the Trident operator to install Trident requires creating the `TridentOrchestrator` Custom Resource Definition (CRD) and defining other resources. You should perform these steps to set up the operator before you can install Astra Trident.

1. Download the latest version of the Trident installer bundle from the *Downloads* section and extract it.

```
wget https://github.com/NetApp/trident/releases/download/v21.04/trident-installer-21.04.tar.gz
tar -xf trident-installer-21.04.tar.gz
cd trident-installer
```

2. Use the appropriate CRD manifest to create the `TridentOrchestrator` CRD. You then create a `TridentOrchestrator` Custom Resource later on to instantiate an installation by the operator.

   Run the following command:

```
kubectl create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
```

3. After the `TridentOrchestrator` CRD is created, create the following resources required for the operator deployment:
   - A ServiceAccount for the operator
   - A ClusterRole and ClusterRoleBinding to the ServiceAccount
   - A dedicated PodSecurityPolicy
   - The operator itself

   The Trident installer contains manifests for defining these resources. By default, the operator is deployed in the `trident` namespace. If the `trident` namespace does not exist, use the following manifest to create one.

```
$ kubectl apply -f deploy/namespace.yaml
```

4. To deploy the operator in a namespace other than the default `trident` namespace, you should update the `serviceaccount.yaml`, `clusterrolebinding.yaml` and `operator.yaml` manifests and generate your `bundle.yaml`.

   Run the following command to update the YAML manifests and generate your `bundle.yaml` using the `kustomization.yaml`:

```
kubectl kustomize deploy/ > deploy/bundle.yaml
```

Run the following command to create the resources and deploy the operator:

```
kubectl create -f deploy/bundle.yaml
```

5. To verify the status of the operator after you have deployed, do the following:

```
$ kubectl get deployment -n <operator-namespace>
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
trident-operator   1/1     1            1           3m

$ kubectl get pods -n <operator-namespace>
NAME                                READY   STATUS              RESTARTS
AGE
trident-operator-54cb664d-lnjxh   1/1     Running             0
3m
```

The operator deployment successfully creates a pod running on one of the worker nodes in your cluster.

> There should only be **one instance** of the operator in a Kubernetes cluster. Do not create multiple deployments of the Trident operator.

Step 3: Create `TridentOrchestrator` and install Trident

You are now ready to install Astra Trident using the operator! This will require creating `TridentOrchestrator`. The Trident installer comes with example definitions for creating `TridentOrchestrator`. This kicks off an installation in the `trident` namespace.

```
$ kubectl create -f deploy/crds/tridentorchestrator_cr.yaml
tridentorchestrator.trident.netapp.io/trident created

$ kubectl describe torc trident
Name:        trident
Namespace:
Labels:      <none>
Annotations: <none>
API Version: trident.netapp.io/v1
Kind:        TridentOrchestrator
...
Spec:
  Debug:     true
  Namespace: trident
Status:
  Current Installation Params:
    IPv6:                      false
    Autosupport Hostname:
    Autosupport Image:         netapp/trident-autosupport:21.04
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                     true
    Enable Node Prep:          false
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:21.04.0
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v21.04.0
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

The Trident operator enables you to customize the manner in which Astra Trident is installed by using the attributes in the `TridentOrchestrator` spec. See Customize your Trident deployment.

The Status of `TridentOrchestrator` indicates if the installation was successful and displays the version of Trident installed.

Status	Description
Installing	The operator is installing Astra Trident using this `TridentOrchestrator` CR.
Installed	Astra Trident has successfully installed.
Uninstalling	The operator is uninstalling Astra Trident, because `spec.uninstall=true`.
Uninstalled	Astra Trident is uninstalled.
Failed	The operator could not install, patch, update or uninstall Astra Trident; the operator will automatically try to recover from this state. If this state persists you will require troubleshooting.
Updating	The operator is updating an existing installation.
Error	The `TridentOrchestrator` is not used. Another one already exists.

During the installation, the status of `TridentOrchestrator` changes from `Installing` to `Installed`. If you observe the `Failed` status and the operator is unable to recover by itself, you should check the logs of the operator. See the troubleshooting section.

You can confirm if the Astra Trident installation completed by taking a look at the pods that have been created:

```
$ kubectl get pod -n trident
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-7d466bf5c7-v4cpw        5/5     Running   0          1m
trident-csi-mr6zc                   2/2     Running   0          1m
trident-csi-xrp7w                   2/2     Running   0          1m
trident-csi-zh2jt                   2/2     Running   0          1m
trident-operator-766f7b8658-ldzsv   1/1     Running   0          3m
```

You can also use `tridentctl` to check the version of Astra Trident installed.

```
$ ./tridentctl -n trident version
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 21.04.0        | 21.04.0        |
+----------------+----------------+
```

Now you can go ahead and create a backend. See post-deployment tasks.

For troubleshooting issues during deployment, see the troubleshooting section.

Customize Trident operator deployment

The Trident operator provides enables you to customize the manner in which Astra Trident is installed by using the attributes in the `TridentOrchestrator` spec.

See the following table for the list of attributes:

Parameter	Description	Default
`namespace`	Namespace to install Astra Trident in	"default"
`debug`	Enable debugging for Astra Trident	false
`IPv6`	Install Astra Trident over IPv6	false
`k8sTimeout`	Timeout for Kubernetes operations	30sec
`silenceAutosupport`	Don't send autosupport bundles to NetApp automatically	false
`enableNodePrep`	Manage worker node dependencies automatically (**BETA**)	false
`autosupportImage`	The container image for Autosupport Telemetry	"netapp/trident-autosupport:21.04.0"
`autosupportProxy`	The address/port of a proxy for sending Autosupport Telemetry	"http://proxy.example.com:8888"
`uninstall`	A flag used to uninstall Astra Trident	false
`logFormat`	Astra Trident logging format to be used [text,json]	"text"
`tridentImage`	Astra Trident image to install	"netapp/trident:21.04"
`imageRegistry`	Path to internal registry, of the format `<registry FQDN>[:port][/subpath]`	"k8s.gcr.io/sig-storage (k8s 1.17+) or quay.io/k8scsi"
`kubeletDir`	Path to the kubelet directory on the host	"/var/lib/kubelet"
`wipeout`	A list of resources to delete to perform a complete removal of Astra Trident	
`imagePullSecrets`	Secrets to pull images from an internal registry	

`spec.namespace` is specified in `TridentOrchestrator` to signify which namespace Astra Trident is installed in. This parameter **cannot be updated after Astra Trident is installed**. Attempting to do so causes the status of `TridentOrchestrator` to change to `Failed`. Astra Trident is not meant to be migrated across namespaces.

Automatic worker node prep is a **beta feature** meant to be used in non-production environments only.

You can use the attributes mentioned above when defining `TridentOrchestrator` to customize your installation. Here's an example:

```
$ cat deploy/crds/tridentorchestrator_cr_imagepullsecrets.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  tridentImage: netapp/trident:21.04.0
  imagePullSecrets:
  - thisisasecret
```

If you are looking to customize the installation beyond what `TridentOrchestrator` arguments allow, you should consider using `tridentctl` to generate custom YAML manifests that you can modify as needed.

Deploy with `tridentctl`

You can deploy Astra Trident by using `tridentctl`.

If you have not already familiarized yourself with the basic concepts, now is a great time to do that.

To customize your deployment, see here.

What you'll need

To deploy Astra Trident, the following prerequisites should be met:

- You have full privileges to a supported Kubernetes cluster.
- You have access to a supported NetApp storage system.
- You have the capability to mount volumes from all of the Kubernetes worker nodes.
- You have a Linux host with `kubectl` (or `oc`, if you are using OpenShift) installed and configured to manage the Kubernetes cluster that you want to use.
- You have set the `KUBECONFIG` environment variable to point to your Kubernetes cluster configuration.

- You have enabled the feature gates required by Astra Trident.
- If you are using Kubernetes with Docker Enterprise, follow their steps to enable CLI access.

Got all that? Great! Let's get started.

For information about customizing your deployment, see here.

Step 1: Qualify your Kubernetes cluster

The first thing you need to do is log into the Linux host and verify that it is managing a *working*, supported Kubernetes cluster that you have the necessary privileges to.

With OpenShift, you use `oc` instead of `kubectl` in all of the examples that follow, and you should log in as **system:admin** first by running `oc login -u system:admin` or `oc login -u kube-admin`.

To check your Kubernetes version, run the following command:

```
kubectl version
```

To see if you have Kubernetes cluster administrator privileges, run the following command:

```
kubectl auth can-i '*' '*' --all-namespaces
```

To verify if you can launch a pod that uses an image from Docker Hub and reach your storage system over the pod network, run the following command:

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- \
  ping <management IP>
```

Identify your Kubernetes server version. You will use it when you install Astra Trident.

Step 2: Download and extract the installer

The Trident installer creates a Trident pod, configures the CRD objects that are used to maintain its state, and initializes the CSI sidecars that perform actions, such as provisioning and attaching volumes to the cluster hosts.

You can download the latest version of the Trident installer bundle from the *Downloads* section, and extract it.

For example, if the latest version is 21.07.1:

```
wget https://github.com/NetApp/trident/releases/download/v21.07.1/trident-
installer-21.07.1.tar.gz
tar -xf trident-installer-21.07.1.tar.gz
cd trident-installer
```

Step 3: Install Astra Trident

Install Astra Trident in the desired namespace by executing the `tridentctl install` command.

```
$ ./tridentctl install -n trident
....
INFO Starting Trident installation.                namespace=trident
INFO Created service account.
INFO Created cluster role.
INFO Created cluster role binding.
INFO Added finalizers to custom resource definitions.
INFO Created Trident service.
INFO Created Trident secret.
INFO Created Trident deployment.
INFO Created Trident daemonset.
INFO Waiting for Trident pod to start.
INFO Trident pod started.                          namespace=trident
pod=trident-csi-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=21.07.1
INFO Trident installation succeeded.
....
```

It will look like this when the installer is complete. Depending on the number of nodes in your Kubernetes cluster, you might observe more pods:

```
$ kubectl get pod -n trident
NAME                              READY   STATUS    RESTARTS   AGE
trident-csi-679648bd45-cv2mx   4/4     Running   0          5m29s
trident-csi-vgc8n              2/2     Running   0          5m29s

$ ./tridentctl -n trident version
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 21.07.1        | 21.07.1        |
+----------------+----------------+
```

If you see output similar to the above example, you've completed this step, but Astra Trident is not yet fully configured. Go ahead and continue to the next step. See post-deployment tasks.

However, if the installer does not complete successfully or you don't see a **Running** `trident-csi-<generated id>`, the platform was not installed.

For troubleshooting issues during deployment, see the troubleshooting section.

Customize `tridentctl` deployment

Trident installer enables you to customize attributes. For example, if you have copied the Trident image to a private repository, you can specify the image name by using `--trident-image`. If you have copied the Trident image as well as the needed CSI sidecar images to a private repository, it might be preferable to specify the location of that repository by using the `--image-registry` switch, which takes the form `<registry FQDN>[:port]`.

To have Astra Trident automatically configure worker nodes for you, use `--enable-node-prep`. For more details on how it works, see here.

Automatic worker node preparation is a **beta feature** meant to be used in non-production environments only.

If you are using a distribution of Kubernetes, where `kubelet` keeps its data on a path other than the usual `/var/lib/kubelet`, you can specify the alternate path by using `--kubelet-dir`.

If you need to customize the installation beyond what the installer's arguments allow, you can also customize the deployment files. Using the `--generate-custom-yaml` parameter creates the following YAML files in the installer's `setup` directory:

- `trident-clusterrolebinding.yaml`
- `trident-deployment.yaml`
- `trident-crds.yaml`
- `trident-clusterrole.yaml`
- `trident-daemonset.yaml`
- `trident-service.yaml`
- `trident-namespace.yaml`
- `trident-serviceaccount.yaml`

After you have generated these files, you can modify them according to your needs and then use `--use-custom-yaml` to install your custom deployment.

```
./tridentctl install -n trident --use-custom-yaml
```

What's next?

After you deploy Astra Trident, you can proceed with creating a backend, creating a storage class, provisioning a volume, and mounting the volume in a pod.

Step 1: Create a backend

You can now go ahead and create a backend that will be used by Astra Trident to provision volumes. To do this, create a `backend.json` file that contains the necessary parameters. Sample configuration files for different backend types can be found in the `sample-input` directory.

See here for more details about how to configure the file for your backend type.

```
cp sample-input/<backend template>.json backend.json
vi backend.json
```

```
./tridentctl -n trident create backend -f backend.json
+-------------+----------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE   | VOLUMES |
+-------------+----------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+----------------+--------------------------------------
+--------+---------+
```

If the creation fails, something was wrong with the backend configuration. You can view the logs to determine the cause by running the following command:

```
./tridentctl -n trident logs
```

After you address the problem, simply go back to the beginning of this step and try again. For more troubleshooting tips, see the troubleshooting section.

Step 2: Create a storage class

Kubernetes users provision volumes by using persistent volume claims (PVCs) that specify a storage class by name. The details are hidden from the users, but a storage class identifies the provisioner that is used for that class (in this case, Trident), and what that class means to the provisioner.

Create a storage class Kubernetes users will specify when they want a volume. The configuration of the class needs to model the backend that you created in the previous step, so that Astra Trident will use it to provision new volumes.

The simplest storage class to start with is one based on the `sample-input/storage-class-csi.yaml.templ` file that comes with the installer, replacing *`BACKEND_TYPE`* with the storage driver name.

```
./tridentctl -n trident get backend
+-------------+----------------+--------------------------------------
+--------+---------+
|    NAME     | STORAGE DRIVER |                 UUID                 |
STATE  | VOLUMES |
+-------------+----------------+--------------------------------------
+--------+---------+
| nas-backend | ontap-nas      | 98e19b74-aec7-4a3d-8dcf-128e5033b214 |
online |       0 |
+-------------+----------------+--------------------------------------
+--------+---------+

cp sample-input/storage-class-csi.yaml.templ sample-input/storage-class-
basic-csi.yaml

# Modify __BACKEND_TYPE__ with the storage driver field above (e.g.,
ontap-nas)
vi sample-input/storage-class-basic-csi.yaml
```

This is a Kubernetes object, so you use `kubectl` to create it in Kubernetes.

```
kubectl create -f sample-input/storage-class-basic-csi.yaml
```

You should now see a **basic-csi** storage class in both Kubernetes and Astra Trident, and Astra Trident should have discovered the pools on the backend.

```
kubectl get sc basic-csi
NAME         PROVISIONER             AGE
basic-csi    csi.trident.netapp.io   15h

./tridentctl -n trident get storageclass basic-csi -o json
{
  "items": [
    {
      "Config": {
        "version": "1",
        "name": "basic-csi",
        "attributes": {
          "backendType": "ontap-nas"
        },
        "storagePools": null,
        "additionalStoragePools": null
      },
      "storage": {
        "ontapnas_10.0.0.1": [
          "aggr1",
          "aggr2",
          "aggr3",
          "aggr4"
        ]
      }
    }
  ]
}
```

Step 3: Provision your first volume

Now you are ready to dynamically provision your first volume. This is done by creating a Kubernetes persistent volume claim (PVC) object.

Create a PVC for a volume that uses the storage class that you just created.

See `sample-input/pvc-basic-csi.yaml` for an example. Make sure the storage class name matches the one that you created.

```
kubectl create -f sample-input/pvc-basic-csi.yaml

kubectl get pvc --watch
NAME      STATUS    VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS   AGE
basic     Pending
basic          1s
basic     Pending   pvc-3acb0d1c-b1ae-11e9-8d9f-5254004dfdb7   0
basic          5s
basic     Bound     pvc-3acb0d1c-b1ae-11e9-8d9f-5254004dfdb7   1Gi
RWO           basic          7s
```

Step 4: Mount the volumes in a pod

Now let us mount the volume. We will launch an nginx pod that mounts the PV under `/usr/share/nginx/html`.

```
cat << EOF > task-pv-pod.yaml
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
       claimName: basic
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
EOF
kubectl create -f task-pv-pod.yaml
```

```
# Wait for the pod to start
kubectl get pod --watch

# Verify that the volume is mounted on /usr/share/nginx/html
kubectl exec -it task-pv-pod -- df -h /usr/share/nginx/html

# Delete the pod
kubectl delete pod task-pv-pod
```

At this point, the pod (application) no longer exists but the volume is still there. You can use it from another pod if you want to.

To delete the volume, delete the claim:

```
kubectl delete pvc basic
```

You can now do additional tasks, such as the following:

- Configure additional backends.
- Create additional storage classes.