



Deploy with Trident operator

Astra Trident

NetApp

November 03, 2022

This PDF was generated from <https://docs.netapp.com/us-en/trident/trident-get-started/kubernetes-customize-deploy.html> on November 03, 2022. Always check docs.netapp.com for the latest.

Table of Contents

- Deploy with Trident operator 1
 - Critical information about Astra Trident 22.10 1
 - Trident operator deployment options 1
 - Verify prerequisites 1
 - Deploy the Trident operator and install Astra Trident using Helm 2
 - Deploy the Trident operator manually and install Trident 2
 - Customize Trident operator deployment 8

Deploy with Trident operator

You can deploy Astra Trident using the Trident operator.

Critical information about Astra Trident 22.10

You must read the following critical information before upgrading to Astra Trident 22.10.



Critical information about Astra Trident 22.10

- Kubernetes 1.25 is now supported in Trident. You must upgrade to Astra Trident 22.10 prior to upgrading to Kubernetes 1.25.
- Astra Trident now strictly enforces the use of multipathing configuration in SAN environments, with a recommended value of `find_multipaths: no` in `multipath.conf` file.

Use of non-multipathing configuration or use of `find_multipaths: yes` or `find_multipaths: smart` value in `multipath.conf` file will result in mount failures. Trident has recommended the use of `find_multipaths: no` since the 21.07 release.

Trident operator deployment options

You can deploy the Trident operator in one of two ways:

- Using the Trident [Helm Chart](#): The Helm Chart deploys the Trident operator and installs Trident in one step.
- Manually: Trident provides a [bundle.yaml](#) file that can be used to install the operator and create associated objects.



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

Verify prerequisites

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

- You have full privileges to a supported Kubernetes cluster running Kubernetes 1.19 - 1.24.
- 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 and install Astra Trident 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 1.19 - 1.24
- Helm version 3

Steps

1. Add Trident's Helm repository:

```
helm repo add netapp-trident https://netapp.github.io/trident-helm-chart
```

2. Use the `helm install` command and specify a name for your deployment.
See the following example:

```
helm install <release-name> netapp-trident/trident-operator --version 22.4.0 --create-namespace <trident-namespace>
```



If you already created a namespace for Trident, the `--create-namespace` parameter will not create an additional namespace.

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> netapp-trident/trident-operator --version 22.7.0 --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 and install Trident

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 verify 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>
```

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 and extract the latest version of the Trident installer bundle from [the Assets section on GitHub](#).

```
wget
https://github.com/NetApp/trident/releases/download/v22.10.0/trident-
installer-22.10.0.tar.gz
tar -xf trident-installer-22.10.0.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:22.10
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:          true
    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 ---- -
  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 <code>TridentOrchestrator</code> CR.
Installed	Astra Trident has successfully installed.
Uninstalling	The operator is uninstalling Astra Trident, because <code>spec.uninstall=true</code> .
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 <code>TridentOrchestrator</code> 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 allows you to customize Astra Trident installation using the attributes in the `TridentOrchestrator` spec.

If you want 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.



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

Configuration options

This table details `TridentOrchestrator` attributes:

Parameter	Description	Default
<code>namespace</code>	Namespace to install Astra Trident in	"default"
<code>debug</code>	Enable debugging for Astra Trident	false
<code>windows</code>	Setting to <code>true</code> enables installation on Windows worker nodes.	false
<code>IPv6</code>	Install Astra Trident over IPv6	false
<code>k8sTimeout</code>	Timeout for Kubernetes operations	30sec
<code>silenceAutosupport</code>	Don't send autosupport bundles to NetApp automatically	false
<code>enableNodePrep</code>	Manage worker node dependencies automatically (BETA)	false
<code>autosupportImage</code>	The container image for Autosupport Telemetry	"netapp/trident-autosupport:22.10.0"
<code>autosupportProxy</code>	The address/port of a proxy for sending Autosupport Telemetry	"http://proxy.example.com:8888"
<code>uninstall</code>	A flag used to uninstall Astra Trident	false
<code>logFormat</code>	Astra Trident logging format to be used [text,json]	"text"
<code>tridentImage</code>	Astra Trident image to install	"netapp/trident:21.04"

Parameter	Description	Default
imageRegistry	Path to internal registry, of the format <registry FQDN>[:port] [/subpath]	"k8s.gcr.io/sig-storage (k8s 1.19+) 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	
controllerPluginNodeSelector	Additional node selectors for pods running the Trident Controller CSI Plugin. Follows same format as pod.spec.nodeSelector.	No default; optional
controllerPluginTolerations	Overrides tolerations for pods running the Trident Controller CSI Plugin. Follows the same format as pod.spec.Tolerations.	No default; optional
nodePluginNodeSelector	Additional node selectors for pods running the Trident Node CSI Plugin. Follows same format as pod.spec.nodeSelector.	No default; optional
nodePluginTolerations	Overrides tolerations for pods running the Trident Node CSI Plugin. Follows the same format as pod.spec.Tolerations.	No default; optional



For more information on formatting pod parameters, see [Assigning Pods to Nodes](#).

Sample configurations

You can use the attributes mentioned above when defining `TridentOrchestrator` to customize your installation.

Example 1: Basic custom configuration

This is an example for a basic custom configuration.

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

Example 2: Deploy with node selectors

This example illustrates how Trident can be deployed with node selectors:

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  controllerPluginNodeSelector:
    nodetype: master
  nodePluginNodeSelector:
    storage: netapp
```

Example 3: Deploy on Windows worker nodes

This example illustrates deployment on a Windows worker node.

```
$ cat deploy/crds/tridentorchestrator_cr.yaml
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  debug: true
  namespace: trident
  windows: true
```

Copyright information

Copyright © 2022 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

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

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

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

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

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

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

Trademark information

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