Manage and monitor Trident

Trident

NetApp
February 20, 2025

Table of Contents

Manage and monitor Trident

Upgrade Trident

Upgrade Trident

Beginning with the 24.02 release, Trident follows a four-month release cadence, delivering three major releases every calendar year. Each new release builds on the previous releases and provides new features, performance enhancements, bug fixes, and improvements. We encourage you to upgrade at least once a year to take advantage of the new features in Trident.

Considerations before upgrading

When upgrading to the latest release of Trident, consider the following:

- There should be only one Trident instance installed across all the namespaces in a given Kubernetes cluster.
- Trident 23.07 and later requires v1 volume snapshots and no longer supports alpha or beta snapshots.
- If you created Cloud Volumes Service for Google Cloud in the CVS service type, you must update the backend configuration to use the `standardsw` or `zoneredundantstandardsw` service level when upgrading from Trident 23.01. Failure to update the `serviceLevel` in the backend could cause volumes to fail. Refer to CVS service type samples for details.
- When upgrading, it is important you provide `parameter.fsType` in `StorageClasses` used by Trident. You can delete and re-create `StorageClasses` without disrupting pre-existing volumes.
  - This is a **requirement** for enforcing security contexts for SAN volumes.
  - The sample input directory contains examples, such as `storage-class-basic.yaml.templ` and `storage-class-bronze-default.yaml`.
  - For more information, refer to Known Issues.

Step 1: Select a version

Trident versions follow a date-based `YY.MM` naming convention, where "YY" is the last two digits of the year and "MM" is the month. Dot releases follow a `YY.MM.X` convention, where "X" is the patch level. You will select the version to upgrade to based on the version you are upgrading from.

- You can perform a direct upgrade to any target release that is within a four-release window of your installed version. For example, you can directly upgrade from 23.04 (or any 23.04 dot release) to 24.06.
- If you are upgrading from a release outside of the four-release window, perform a multi-step upgrade. Use the upgrade instructions for the earlier version you are upgrading from to upgrade to the most recent release that fits the four-release window. For example, if you are running 22.01 and want to upgrade to 24.06:
  1. First upgrade from 22.07 to 23.04.
  2. Then upgrade from 23.04 to 24.06.

When upgrading using the Trident operator on OpenShift Container Platform, you should upgrade to Trident 21.01.1 or later. The Trident operator released with 21.01.0 contains a known issue that has been fixed in 21.01.1. For more details, refer to the issue details on GitHub.

Step 2: Determine the original installation method

To determine which version you used to originally install Trident:

1. Use `kubectl get pods -n trident` to examine the pods.
   - If there is no operator pod, Trident was installed using `tridentctl`.
   - If there is an operator pod, Trident was installed using the Trident operator either manually or using Helm.
2. If there is an operator pod, use `kubectl describe torc` to determine if Trident was installed using Helm.
   - If there is a Helm label, Trident was installed using Helm.
   - If there is no Helm label, Trident was installed manually using the Trident operator.

Step 3: Select an upgrade method

Generally, you should upgrade using the same method you used for the initial installation, however you can move between installation methods. There are two options to upgrade Trident.

- Upgrade using the Trident operator

  We suggest you review Understand the operator upgrade workflow before upgrading with the operator.

- Upgrade using `tridentctl`

Upgrade with the operator

Understand the operator upgrade workflow

Before using the Trident operator to upgrade Trident, you should understand the background processes that occur during upgrade. This includes changes to the Trident controller, controller Pod and node Pods, and node DaemonSet that enable rolling updates.

Trident operator upgrade handling

One of the many benefits of using the Trident operator to install and upgrade Trident is the automatic handling of Trident and Kubernetes objects without disrupting existing mounted volumes. In this way, Trident can support upgrades with zero downtime, or *rolling updates*. In particular, the Trident operator communicates with the Kubernetes cluster to:

- Delete and recreate the Trident Controller deployment and node DaemonSet.
- Replace the Trident Controller Pod and Trident Node Pods with new versions.
  - If a node is not updated, it does not prevent remaining nodes from being updated.

- Only nodes with a running Trident Node Pod can mount volumes.

For more information about Trident architecture on the Kubernetes cluster, refer to Trident architecture.

Operator upgrade workflow

When you initiate an upgrade using the Trident operator:

1. The **Trident operator**:
   a. Detects the currently installed version of Trident (version *n*).
   b. Updates all Kubernetes objects including CRDs, RBAC, and Trident SVC.
   c. Deletes the Trident Controller deployment for version *n*.
   d. Creates the Trident Controller deployment for version *n+1*.
2. **Kubernetes** creates Trident Controller Pod for *n+1*.
3. The **Trident operator**:
   a. Deletes the Trident Node DaemonSet for *n*. The operator does not wait for Node Pod termination.
   b. Creates the Trident Node Daemonset for *n+1*.
4. **Kubernetes** creates Trident Node Pods on nodes not running Trident Node Pod *n*. This ensures there is never more than one Trident Node Pod, of any version, on a node.

Upgrade a Trident installation using Trident operator or Helm

You can upgrade Trident using the Trident operator either manually or using Helm. You can upgrade from a Trident operator installation to another Trident operator installation or upgrade from a `tridentctl` installation to a Trident operator version. Review Select an upgrade method before upgrading a Trident operator installation.

Upgrade a manual installation

You can upgrade from a cluster-scoped Trident operator installation to another cluster-scoped Trident operator installation. All Trident versions 21.01 and above use a cluster-scoped operator.

To upgrade from Trident that was installed using the namespace-scoped operator (versions 20.07 through 20.10), use the upgrade instructions for your installed version of Trident.

About this task

Trident provides a bundle file you can use to install the operator and create associated objects for your Kubernetes version.

- For clusters running Kubernetes 1.24, use bundle_pre_1_25.yaml.
- For clusters running Kubernetes 1.25 or later, use bundle_post_1_25.yaml.

Before you begin

Ensure you are using a Kubernetes cluster running a supported Kubernetes version.

Steps

1. Verify your Trident version:

```
./tridentctl -n trident version
```

2. Delete the Trident operator that was used to install the current Trident instance. For example, if you are upgrading from 23.07, run the following command:

```
kubectl delete -f 23.07.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

3. If you customized your initial installation using `TridentOrchestrator` attributes, you can edit the `TridentOrchestrator` object to modify the installation parameters. This might include changes made to specify mirrored Trident and CSI image registries for offline mode, enable debug logs, or specify image pull secrets.
4. Install Trident using the correct bundle YAML file for your environment, where *<bundle.yaml>* is `bundle_pre_1_25.yaml` or `bundle_post_1_25.yaml` based on your Kubernetes version. For example, if you are installing Trident 24.10, run the following command:

```
kubectl create -f 24.10.0/trident-installer/deploy/<bundle.yaml> -n
trident
```

Upgrade a Helm installation

You can upgrade a Trident Helm installation.

> When upgrading a Kubernetes cluster from 1.24 to 1.25 or later that has Trident installed, you must update values.yaml to set `excludePodSecurityPolicy` to `true` or add `--set excludePodSecurityPolicy=true` to the `helm upgrade` command before you can upgrade the cluster.

If you have already upgraded your Kubernetes cluster from 1.24 to 1.25 without upgrading the Trident helm, the helm upgrade fails. For the helm upgrade to go through, perform these steps as pre-requisites:

1. Install the helm-mapkubeapis plugin from https://github.com/helm/helm-mapkubeapis.
2. Perform a dry run for the Trident release in the namespace where Trident is installed. This lists out the resources, which will be cleaned up.

```
helm mapkubeapis --dry-run trident  --namespace trident
```

3. Perform a full run with helm to do the cleanup.

```
helm mapkubeapis trident  --namespace trident
```

Steps

1. If you installed Trident using Helm, you can use `helm upgrade trident netapp-trident/trident-operator --version 100.2410.0` to upgrade in one step. If you did not add the Helm repo or cannot use it to upgrade:
   a. Download the latest Trident release from the *Assets* section on GitHub.
   b. Use the `helm upgrade` command where `trident-operator-24.10.0.tgz` reflects the version that you want to upgrade to.

```
helm upgrade <name> trident-operator-24.10.0.tgz
```

> If you set custom options during the initial installation (such as specifying private, mirrored registries for Trident and CSI images), append the `helm upgrade` command using `--set` to ensure those options are included in the upgrade command, otherwise the values will reset to default.

2. Run `helm list` to verify that the chart and app version have both been upgraded. Run `tridentctl logs` to review any debug messages.

Upgrade from a `tridentctl` installation to Trident operator

You can upgrade to the latest release of the Trident operator from a `tridentctl` installation. The existing backends and PVCs will automatically be available.

> Before switching between installation methods, review Moving between installation methods.

Steps

1. Download the latest Trident release.

```
# Download the release required [24.10.0]
mkdir 24.10.0
cd 24.10.0
wget
https://github.com/NetApp/trident/releases/download/v24.10.0/trident-installer-24.10.0.tar.gz
tar -xf trident-installer-24.10.0.tar.gz
cd trident-installer
```

2. Create the `tridentorchestrator` CRD from the manifest.

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

3. Deploy the cluster-scoped operator in the same namespace.

```
kubectl create -f deploy/<bundle-name.yaml>

serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created

#Examine the pods in the Trident namespace
NAME                                  READY   STATUS    RESTARTS   AGE
trident-controller-79df798bdc-m79dc   6/6     Running   0          150d
trident-node-linux-xrst8              2/2     Running   0          150d
trident-operator-5574dbbc68-nthjv     1/1     Running   0          1m30s
```

4. Create a `TridentOrchestrator` CR for installing Trident.

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

kubectl create -f deploy/crds/tridentorchestrator_cr.yaml

#Examine the pods in the Trident namespace
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-79df798bdc-m79dc        6/6     Running   0          1m
trident-csi-xrst8                   2/2     Running   0          1m
trident-operator-5574dbbc68-nthjv   1/1     Running   0          5m41s
```

5. Confirm Trident was upgraded to the intended version.

```
kubectl describe torc trident | grep Message -A 3

Message:                Trident installed
Namespace:              trident
Status:                 Installed
Version:                v24.10.0
```

Upgrade with tridentctl

You can easily upgrade an existing Trident installation using `tridentctl`.

About this task

Uninstalling and reinstalling Trident acts as an upgrade. When you uninstall Trident, the Persistent Volume Claim (PVC) and Persistent Volume (PV) used by the Trident deployment are not deleted. PVs that have already been provisioned will remain available while Trident is offline, and Trident will provision volumes for any PVCs that are created in the interim after it is back online.

Before you begin

Review Select an upgrade method before upgrading using `tridentctl`.

Steps

1. Run the uninstall command in `tridentctl` to remove all of the resources associated with Trident except for the CRDs and related objects.

```
./tridentctl uninstall -n <namespace>
```

2. Reinstall Trident. Refer to Install Trident using tridentctl.

> Do not interrupt the upgrade process. Ensure the installer runs to completion.

Manage Trident using tridentctl

The Trident installer bundle includes the `tridentctl` command-line utility to provide simple access to Trident. Kubernetes users with sufficient privileges can use it to install Trident or manage the namespace that contains the Trident pod.

Commands and global flags

You can run `tridentctl help` to get a list of available commands for `tridentctl` or append the `--help` flag to any command to get a list of options and flags for that specific command.

```
tridentctl [command] [--optional-flag]
```

The Trident `tridentctl` utility supports the following commands and global flags.

Commands

`create`
Add a resource to Trident.

`delete`
Remove one or more resources from Trident.

`get`
Get one or more resources from Trident.

`help`
Help about any command.

`images`
Print a table of the container images Trident needs.

`import`
Import an existing resource to Trident.

`install`
Install Trident.

`logs`
Print the logs from Trident.

`send`
Send a resource from Trident.

`uninstall`
Uninstall Trident.

`update`
Modify a resource in Trident.

`update backend state`
Temporarily suspend backend operations.

`upgrade`
Upgrade a resource in Trident.

`version`
Print the version of Trident.

Global flags

`-d, --debug`
Debug output.

`-h, --help`
Help for `tridentctl`.

`-k, --kubeconfig string`
Specify the `KUBECONFIG` path to run commands locally or from one Kubernetes cluster to another.

> Alternatively, you can export the `KUBECONFIG` variable to point to a specific Kubernetes cluster and issue `tridentctl` commands to that cluster.

`-n, --namespace string`
Namespace of Trident deployment.

`-o, --output string`
Output format. One of json|yaml|name|wide|ps (default).

`-s, --server string`
Address/port of Trident REST interface.

> Trident REST interface can be configured to listen and serve at 127.0.0.1 (for IPv4) or [::1] (for IPv6) only.

Command options and flags

create

Use the `create` command to add a resource to Trident.

```
tridentctl create [option]
```

Options

`backend`: Add a backend to Trident.

delete

Use the `delete` command to remove one or more resources from Trident.

```
tridentctl delete [option]
```

Options

`backend`: Delete one or more storage backends from Trident.
`snapshot`: Delete one or more volume snapshots from Trident.
`storageclass`: Delete one or more storage classes from Trident.
`volume`: Delete one or more storage volumes from Trident.

get

Use the `get` command to get one or more resources from Trident.

```
tridentctl get [option]
```

Options

`backend`: Get one or more storage backends from Trident.
`snapshot`: Get one or more snapshots from Trident.
`storageclass`: Get one or more storage classes from Trident.
`volume`: Get one or more volumes from Trident.

Flags

`-h`, `--help`: Help for volumes.
`--parentOfSubordinate string`: Limit query to subordinate source volume.
`--subordinateOf string`: Limit query to subordinates of volume.

images

Use `images` flags to print a table of the container images Trident needs.

```
tridentctl images [flags]
```

Flags

`-h`, `--help`: Help for images.
`-v`, `--k8s-version string`: Semantic version of Kubernetes cluster.

import volume

Use the `import volume` command to import an existing volume to Trident.

```
tridentctl import volume <backendName> <volumeName> [flags]
```

Aliases

`volume`, `v`

Flags

`-f`, `--filename string`: Path to YAML or JSON PVC file.
`-h`, `--help`: Help for volume.
`--no-manage`: Create PV/PVC only. Don’t assume volume lifecycle management.

install

Use the `install` flags to install Trident.

```
tridentctl install [flags]
```

Flags

`--autosupport-image string`: The container image for Autosupport Telemetry (default "netapp/trident autosupport:<current-version>").
`--autosupport-proxy string`: The address/port of a proxy for sending Autosupport Telemetry.

`--enable-node-prep`: Attempt to install required packages on nodes.
`--generate-custom-yaml`: Generate YAML files without installing anything.
`-h, --help`: Help for install.
`--http-request-timeout`: Override the HTTP request timeout for Trident controller's REST API (default 1m30s).
`--image-registry string`: The address/port of an internal image registry.
`--k8s-timeout duration`: The timeout for all Kubernetes operations (default 3m0s).
`--kubelet-dir string`: The host location of kubelet's internal state (default "/var/lib/kubelet").
`--log-format string`: The Trident logging format (text, json) (default "text").
`--node-prep`: Enables Trident to prepare the nodes of the Kubernetes cluster to manage volumes using the specified data storage protocol. **Currently, `iscsi` is the only value supported.**
`--pv string`: The name of the legacy PV used by Trident, makes sure this doesn't exist (default "trident").
`--pvc string`: The name of the legacy PVC used by Trident, makes sure this doesn't exist (default "trident").
`--silence-autosupport`: Don't send autosupport bundles to NetApp automatically (default true).
`--silent`: Disable most output during installation.
`--trident-image string`: The Trident image to install.
`--use-custom-yaml`: Use any existing YAML files that exist in setup directory.
`--use-ipv6`: Use IPv6 for Trident's communication.

logs

Use `logs` flags to print the logs from Trident.

```
tridentctl logs [flags]
```

Flags

`-a, --archive`: Create a support archive with all logs unless otherwise specified.
`-h, --help`: Help for logs.
`-l, --log string`: Trident log to display. One of trident|auto|trident-operator|all (default "auto").
`--node string`: The Kubernetes node name from which to gather node pod logs.
`-p, --previous`: Get the logs for the previous container instance if it exists.
`--sidecars`: Get the logs for the sidecar containers.

send

Use the `send` command to send a resource from Trident.

```
tridentctl send [option]
```

Options

`autosupport`: Send an Autosupport archive to NetApp.

uninstall

Use `uninstall` flags to uninstall Trident.

```
tridentctl uninstall [flags]
```

Flags

`-h, --help`: Help for uninstall.
`--silent`: Disable most output during uninstall.

update

Use the `update` command to modify a resource in Trident.

```
tridentctl update [option]
```

Options

`backend`: Update a backend in Trident.

update backend state

Use the `update backend state` command to suspend or resume backend operations.

```
tridentctl update backend state <backend-name> [flag]
```

Points to consider

- If a backend is created using a TridentBackendConfig (tbc), the backend cannot be updated using a `backend.json` file.
- If the `userState` has been set in a tbc, it cannot be modified using the `tridentctl update backend state <backend-name> --user-state suspended/normal` command.
- To regain the ability to set the `userState` via tridentctl after it has been set via tbc, the `userState` field must be removed from the tbc. This can be done using the `kubectl edit tbc` command. After the `userState` field is removed, you can use the `tridentctl update backend state` command to change the `userState` of a backend.
- Use the `tridentctl update backend state` to change the `userState`. You can also update the `userState` using `TridentBackendConfig` or `backend.json` file; this triggers a complete re-initialization of the backend and can be time-consuming.

  #### Flags

  `-h, --help`: Help for backend state.
  `--user-state`: Set to `suspended` to pause backend operations. Set to `normal` to resume backend operations. When set to `suspended`:

- `AddVolume` and `Import Volume` are paused.
- `CloneVolume`, `ResizeVolume`, `PublishVolume`, `UnPublishVolume`, `CreateSnapshot`, `GetSnapshot`, `RestoreSnapshot`, `DeleteSnapshot`, `RemoveVolume`, `GetVolumeExternal`, `ReconcileNodeAccess` remain available.

You can also update the backend state using `userState` field in the backend configuration file `TridentBackendConfig` or `backend.json`.
For more information, refer to Options for managing backends and Perform backend management with kubectl.

Example:

JSON

Follow these steps to update the `userState` using the `backend.json` file:

1. Edit the `backend.json` file to include the `userState` field with its value set to 'suspended'.
2. Update the backend using the `tridentctl backend update` command and the path to the updated `backend.json` file.

   Example: `tridentctl backend update -f /<path to backend JSON file>/backend.json`

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "managementLIF": "<redacted>",
    "svm": "nas-svm",
    "backendName": "customBackend",
    "username": "<redacted>",
    "password": "<redacted>",
    "userState": "suspended",
}
```

YAML

You can edit the tbc after it has been applied using the `kubectl edit <tbc-name> -n <namespace>` command.
The following example updates the backend state to suspend using the `userState: suspended` option:

```
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  backendName: customBackend
  storageDriverName: ontap-nas
  managementLIF: <redacted>
  svm: nas-svm
userState: suspended
  credentials:
    name: backend-tbc-ontap-nas-secret
```

version

Use `version` flags to print the version of `tridentctl` and the running Trident service.

```
tridentctl version [flags]
```

Flags

`--client`: Client version only (no server required).
`-h, --help`: Help for version.

Plugin support

Tridentctl supports plugins similar to kubectl. Tridentctl detects a plugin if the plugin binary file name follows the scheme "tridentctl-<plugin>", and the binary is located in a folder listed the PATH environment variable. All the detected plugins are listed in the plugin section of the tridentctl help. Optionally, you can also limit the search by specifying a plugin folder in the the enviornment variable TRIDENTCTL_PLUGIN_PATH (Example: `TRIDENTCTL_PLUGIN_PATH=~/tridentctl-plugins/`). If the variable is used, tridenctl searches only in the specified folder.

Monitor Trident

Trident provides a set of Prometheus metrics endpoints that you can use to monitor Trident performance.

Overview

The metrics provided by Trident enable you to do the following:

- Keep tabs on Trident's health and configuration. You can examine how successful operations are and if it can communicate with the backends as expected.
- Examine backend usage information and understand how many volumes are provisioned on a backend and the amount of space consumed, and so on.
- Maintain a mapping of the amount of volumes provisioned on available backends.
- Track performance. You can take a look at how long it takes for Trident to communicate to backends and perform operations.

By default, Trident's metrics are exposed on the target port `8001` at the `/metrics` endpoint. These metrics are **enabled by default** when Trident is installed.

What you'll need

- A Kubernetes cluster with Trident installed.
- A Prometheus instance. This can be a containerized Prometheus deployment or you can choose to run Prometheus as a native application.

Step 1: Define a Prometheus target

You should define a Prometheus target to gather the metrics and obtain information about the backends Trident manages, the volumes it creates, and so on. This blog explains how you can use Prometheus and Grafana with Trident to retrieve metrics. The blog explains how you can run Prometheus as an operator in your Kubernetes cluster and the creation of a ServiceMonitor to obtain Trident metrics.

Step 2: Create a Prometheus ServiceMonitor

To consume the Trident metrics, you should create a Prometheus ServiceMonitor that watches the `trident-csi` service and listens on the `metrics` port. A sample ServiceMonitor looks like this:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: trident-sm
  namespace: monitoring
  labels:
      release: prom-operator
spec:
  jobLabel: trident
  selector:
    matchLabels:
      app: controller.csi.trident.netapp.io
  namespaceSelector:
    matchNames:
    - trident
  endpoints:
  - port: metrics
    interval: 15s
```

This ServiceMonitor definition retrieves metrics returned by the `trident-csi` service and specifically looks for the `metrics` endpoint of the service. As a result, Prometheus is now configured to understand Trident's metrics.

In addition to metrics available directly from Trident, kubelet exposes many `kubelet_volume_*` metrics via it's own metrics endpoint. Kubelet can provide information about the volumes that are attached, and pods and other internal operations it handles. Refer to here.

Step 3: Query Trident metrics with PromQL

PromQL is good for creating expressions that return time-series or tabular data.

Here are some PromQL queries that you can use:

Get Trident health information

- **Percentage of HTTP 2XX responses from Trident**

```
(sum (trident_rest_ops_seconds_total_count{status_code=~"2.."} OR on()
vector(0)) / sum (trident_rest_ops_seconds_total_count)) * 100
```

- **Percentage of REST responses from Trident via status code**

```
(sum (trident_rest_ops_seconds_total_count) by (status_code)  / scalar
(sum (trident_rest_ops_seconds_total_count))) * 100
```

- **Average duration in ms of operations performed by Trident**

```
sum by (operation)
(trident_operation_duration_milliseconds_sum{success="true"}) / sum by
(operation)
(trident_operation_duration_milliseconds_count{success="true"})
```

Get Trident usage information

- **Average volume size**

```
trident_volume_allocated_bytes/trident_volume_count
```

- **Total volume space provisioned by each backend**

```
sum (trident_volume_allocated_bytes) by (backend_uuid)
```

Get individual volume usage

This is enabled only if kubelet metrics are also gathered.

- **Percentage of used space for each volume**

```
kubelet_volume_stats_used_bytes / kubelet_volume_stats_capacity_bytes *
100
```

Learn about Trident AutoSupport telemetry

By default, Trident sends Prometheus metrics and basic backend information to NetApp on a daily cadence.

- To stop Trident from sending Prometheus metrics and basic backend information to NetApp, pass the `--silence-autosupport` flag during Trident installation.
- Trident can also send container logs to NetApp Support on-demand via `tridentctl send autosupport`. You will need to trigger Trident to upload it's logs. Before you submit logs, you should accept NetApp's
  privacy policy.
- Unless specified, Trident fetches the logs from the past 24 hours.
- You can specify the log retention time frame with the `--since` flag. For example: `tridentctl send autosupport --since=1h`. This information is collected and sent via a `trident-autosupport`

container
that is installed alongside Trident. You can obtain the container image at Trident AutoSupport.

- Trident AutoSupport does not gather or transmit Personally Identifiable Information (PII) or Personal Information. It comes with a EULA that is not applicable to the Trident container image itself. You can learn more about NetApp's commitment to data security and trust here.

An example payload sent by Trident looks like this:

```
---
items:
- backendUUID: ff3852e1-18a5-4df4-b2d3-f59f829627ed
  protocol: file
  config:
    version: 1
    storageDriverName: ontap-nas
    debug: false
    debugTraceFlags:
    disableDelete: false
    serialNumbers:
    - nwkvzfanek_SN
    limitVolumeSize: ''
  state: online
  online: true
```

- The AutoSupport messages are sent to NetApp's AutoSupport endpoint. If you are using a private registry to store container images, you can use the `--image-registry` flag.
- You can also configure proxy URLs by generating the installation YAML files. This can be done by using `tridentctl install --generate-custom-yaml` to create the YAML files and adding the `--proxy-url` argument for the `trident-autosupport` container in `trident-deployment.yaml`.

Disable Trident metrics

To **disable** metrics from being reported, you should generate custom YAMLs (using the `--generate-custom-yaml` flag) and edit them to remove the `--metrics` flag from being invoked for the `trident-main` container.

Uninstall Trident

You should use the same method to uninstall Trident that you used to install Trident.

About this task

- If you need a fix for bugs observed after an upgrade, dependency issues, or an unsuccessful or incomplete upgrade, you should uninstall Trident and reinstall the earlier version using the specific instructions for that version. This is the only recommended way to *downgrade* to an earlier version.
- For easy upgrade and reinstallation, uninstalling Trident does not remove the CRDs or related objects created by Trident. If you need to completely remove Trident and all of its data, refer to Completely remove Trident and CRDs.

Before you begin

If you are decommissioning Kubernetes clusters, you must delete all applications that use volumes created by Trident prior to uninstalling. This ensures that PVCs are unpublished on Kubernetes nodes before they are deleted.

Determine the original installation method

You should use the same method to uninstall Trident that you used to install it. Before uninstalling, verify which version you used to originally install Trident.

1. Use `kubectl get pods -n trident` to examine the pods.
   - If there is no operator pod, Trident was installed using `tridentctl`.
   - If there is an operator pod, Trident was installed using the Trident operator either manually or using Helm.
2. If there is an operator pod, use `kubectl describe tproc trident` to determine if Trident was installed using Helm.
   - If there is a Helm label, Trident was installed using Helm.
   - If there is no Helm label, Trident was installed manually using the Trident operator.

Uninstall a Trident operator installation

You can uninstall a trident operator installation manually or using Helm.

Uninstall manual installation

If you installed Trident using the operator, you can uninstall it by doing one of the following:

1. **Edit `TridentOrchestrator` CR and set the uninstall flag**:

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

   When the `uninstall` flag is set to `true`, the Trident operator uninstalls Trident, but does not remove the TridentOrchestrator itself. You should clean up the TridentOrchestrator and create a new one if you want to install Trident again.

2. **Delete `TridentOrchestrator`**: By removing the `TridentOrchestrator` CR that was used to deploy Trident, you instruct the operator to uninstall Trident. The operator processes the removal of `TridentOrchestrator` and proceeds to remove the Trident deployment and daemonset, deleting the Trident pods it had created as part of the installation.

   ```
   kubectl delete -f deploy/<bundle.yaml> -n <namespace>
   ```

Uninstall Helm installation

If you installed Trident by using Helm, you can uninstall it by using `helm uninstall`.

```
#List the Helm release corresponding to the Trident install.
helm ls -n trident
NAME          NAMESPACE       REVISION        UPDATED
STATUS          CHART                           APP VERSION
trident       trident         1               2021-04-20
00:26:42.417764794 +0000 UTC deployed        trident-operator-21.07.1
21.07.1

#Uninstall Helm release to remove Trident
helm uninstall trident -n trident
release "trident" uninstalled
```

Uninstall a `tridentctl` installation

Use the `uninstall` command in `tridentctl` to remove all of the resources associated with Trident except for the CRDs and related objects:

```
./tridentctl uninstall -n <namespace>
```

Copyright information

Trademark information