入门

Astra Trident

NetApp
April 16, 2024

目录

入门

试用

NetApp 提供了一个可随时使用的实验室映像，您可以通过该映像进行请求 "NetApp 试用"。

了解测试活动

此测试驱动器为您提供了一个沙盒环境，该环境附带安装和配置了三节点 Kubernetes 集群和 Astra Trident 。这是熟悉 Astra Trident 并了解其功能的好方法。

另一个选项是查看 "《 kubeadm 安装指南》" 由 Kubernetes 提供。

> 您不应在生产环境中使用使用这些说明构建的 Kubernetes 集群。使用您的分发版提供的生产部署指南创建可随时投入生产的集群。

如果这是您第一次使用 Kubernetes ，请熟悉相关概念和工具 "此处"。

要求

在安装Astra Trident之前、您应查看这些常规系统要求。特定后端可能有其他要求。

有关**Astra Trident 23.01**的关键信息

您必须阅读以下有关**Astra Trident**的重要信息。

**** 中有关**Astra ** 的信息

- Trident现在支持Kubernetes 1.26。在升级Kubernetes之前升级Trident。
- Astra Trident会严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

支持的前端（编排程序）

Astra Trident 支持多个容器引擎和流程编排程序，其中包括：

- Anthos on-Prem (VMware)和Anthos on bare metal 1.9、1.10、1.11
- Kubernetes 1.21 - 1.26
- Mirantis Kubernetes Engine 3.5
- OpenShift 4.9 - 4.12

以下版本支持 Trident 操作符：

- Anthos on-Prem (VMware)和Anthos on bare metal 1.9、1.10、1.11
- Kubernetes 1.21 - 1.26
- OpenShift 4.9 - 4.12

Astra Trident 还可与许多其他完全托管和自我管理的 Kubernetes 产品结合使用，包括 Google Kubernetes Engine （ GKEE ）， Amazon Elastic Kubernetes Services （ EKS ）， Azure Kubernetes Service （ AKS ）， Rancher 和 VMware Tanzu Portfolio 。

在将已安装Astra Trident的Kubernetes集群从1.24升级到1.25或更高版本之前、请参见 "升级基于 Helm 的操作员安装"。

支持的后端（存储）

要使用 Astra Trident ，您需要以下一个或多个受支持的后端：

- 适用于 NetApp ONTAP 的 Amazon FSX
- Azure NetApp Files
- Cloud Volumes ONTAP
- 适用于 GCP 的 Cloud Volumes Service
- FAS/AFF/Select 9.5或更高版本
- NetApp 全 SAN 阵列（ ASA ）
- NetApp HCI/ Element软件11或更高版本

功能要求

下表总结了此版本的 Astra Trident 及其支持的 Kubernetes 版本提供的功能。

功能	**Kubernetes** 版本	是否需要功能安全门?
CSI Trident	1.21 - 1.26	否
卷快照	1.21 - 1.26	否
卷快照中的 PVC	1.21 - 1.26	否
iSCSI PV 调整大小	1.21 - 1.26	否
ONTAP 双向 CHAP	1.21 - 1.26	否
动态导出策略	1.21 - 1.26	否
Trident 运算符	1.21 - 1.26	否

功能	**Kubernetes** 版本	是否需要功能安全门?
CSI 拓扑	1.21 - 1.26	否

已测试主机操作系统

虽然Astra Trident不正式支持特定的操作系统、但已知以下功能有效：

- OpenShift 容器平台支持的 RedHat CoreOS （ RHCOS ）版本
- RHEL 8+
- Ubuntu 22.04或更高版本
- Windows Server 2019

默认情况下， Astra Trident 在容器中运行，因此将在任何 Linux 工作程序上运行。但是，根据您使用的后端，这些员工需要能够使用标准 NFS 客户端或 iSCSI 启动程序挂载 Astra Trident 提供的卷。

。 `tridentctl` 实用程序还可以在Linux的任何这些分发版上运行。

主机配置

Kubernetes集群中的所有工作节点都必须能够挂载为Pod配置的卷。要准备工作节点、必须根据您选择的驱动程序安装NFS或iSCSI工具。

"准备工作节点"

存储系统配置：

Astra Trident可能需要先更改存储系统、然后后端配置才能使用它。

"配置后端"

Astra Trident 端口

Astra Trident需要访问特定端口才能进行通信。

"Astra Trident 端口"

容器映像以及相应的 Kubernetes 版本

对于带气的安装，下面列出了安装 Astra Trident 所需的容器映像。使用 `tridentctl images` 用于验证所需容器映像列表的命令。

Kubernetes 版本	容器映像
v1.21.0	• dDocker .io/NetApp/trident：23.01.1 • docer.io/NetApp/trident-autostsupport：23.01 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.4.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.1.0 • 注册表.k8s.io/sig-storage/CsI-6泽 程序：v1.7.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.2.2 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.7.0 • dDocker .io/NetApp/trident-operator：23.01.1 (可选)
v1.22.0	• dDocker .io/NetApp/trident：23.01.1 • docer.io/NetApp/trident-autostsupport：23.01 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.4.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.1.0 • 注册表.k8s.io/sig-storage/CsI-6泽 程序：v1.7.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.2.2 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.7.0 • dDocker .io/NetApp/trident-operator：23.01.1 (可选)
v1.23.0	• dDocker .io/NetApp/trident：23.01.1 • docer.io/NetApp/trident-autostsupport：23.01 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.4.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.1.0 • 注册表.k8s.io/sig-storage/CsI-6泽 程序：v1.7.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.2.2 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.7.0 • dDocker .io/NetApp/trident-operator：23.01.1 (可选)

Kubernetes 版本	容器映像
v1.24.0	• dDocker .io/NetApp/trident：23.01.1 • docer.io/NetApp/trident-autostsupport：23.01 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.4.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.1.0 • 注册表.k8s.io/sig-storage/CsI-6泽 程序：v1.7.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.2.2 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.7.0 • dDocker .io/NetApp/trident-operator：23.01.1 (可选)
v1.25.0	• dDocker .io/NetApp/trident：23.01.1 • docer.io/NetApp/trident-autostsupport：23.01 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.4.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.1.0 • 注册表.k8s.io/sig-storage/CsI-6泽 程序：v1.7.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.2.2 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.7.0 • dDocker .io/NetApp/trident-operator：23.01.1 (可选)
v1.26.0	• dDocker .io/NetApp/trident：23.01.1 • docer.io/NetApp/trident-autostsupport：23.01 • 注册表.k8s.io/sig-storage/CsI-置 配置程序：v3.4.0 • 注册表.k8s.io/sig-storage/CsI-attacher：v4.1.0 • 注册表.k8s.io/sig-storage/CsI-6泽 程序：v1.7.0 • 注册表.k8s.io/sig-storage/CsI-snapshotter：v6.2.2 • 注册表.k8s.io/sig-storage/CsI-N节点 驱动程序注册器：v2.7.0 • dDocker .io/NetApp/trident-operator：23.01.1 (可选)

在Kubernetes 1.21及更高版本上、使用经验证的 `registry.k8s.gcr.io/sig-storage/csi-snapshotter:v6.x` 仅当出现时才创建映像 `v1` 版本正在提供 `volumesnapshots.snapshot.storage.k8s.gcr.io` CRD。如果 `v1beta1` 版本正在为CRD提供支持/不提供 `v1` 版本、请使用已验证的 `registry.k8s.gcr.io/sig-storage/csi-snapshotter:v3.x` 图像。

安装 Astra Trident

了解有关Astra Trident安装的信息

为了确保Astra Trident能够安装在各种环境和组织中、NetApp提供了多种安装选项。您可以使用Tridentoperator(手动或使用Helm)或安装Astra Trident `tridentctl`。本主题提供了有关为您选择正确安装过程的重要信息。

有关Astra Trident 23.01的关键信息

您必须阅读以下有关Astra Trident的重要信息。

** 中有关Astra 的信息**

- Trident现在支持Kubernetes 1.26。在升级Kubernetes之前升级Trident。
- Astra Trident会严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

开始之前

无论您的安装路径如何、您都必须：

- 对运行受支持的Kubernetes版本并启用了功能要求的受支持Kubernetes集群的完全权限。查看 "要求" 了解详细信息。
- 访问受支持的NetApp存储系统。
- 能够从所有Kubernetes工作节点挂载卷。
- 具有的Linux主机 `kubectl` (或 `oc`(如果您使用的是OpenShift)已安装并配置为管理要使用的Kubernetes集群。
- 。 `KUBECONFIG` 环境变量设置为指向您的Kubernetes集群配置。
- 如果您将 Kubernetes 与 Docker Enterprise 结合使用， "按照其步骤启用 CLI 访问"。

如果您尚未熟悉 "基本概念"，现在是一个实现这一目标的好时机。

选择安装方法

选择适合您的安装方法。您还应查看的注意事项 "在方法之间移动" 在做出决定之前。

使用**Trident**运算符

无论是手动部署还是使用Helm部署、Trident操作员都是简化安装和动态管理Astra Trident资源的绝佳方式。您可以做到这一点 "自定义Trident操作员部署" 使用中的属性 `TridentOrchestrator` 自定义资源(CR)。

使用Trident运算符的优势包括：

** A Trident对象**

Trident操作符会自动为您的Kubernetes版本创建以下对象。

- 操作员的ServiceAccount
- ClusterRole和ClusterRoleBindingto the ServiceAccount
- 专用PodSecurityPolicy (适用于Kubernetes 1.25及更早版本)
- 运算符本身

** —修复功能—**

操作员监控Astra Trident的安装、并主动采取措施来解决问题、例如部署何时被删除或意外修改。答 `trident-operator-<generated-id>` 此时将创建与关联的POD `TridentOrchestrator` 安装了Astra Trident的CR。这样可以确保集群中只有一个Astra Trident实例并控制其设置、从而确保安装有效。对安装进行更改（例如删除部署或节点取消设置）时，操作员会识别这些更改并逐个修复它们。

**** 更新了现有安装的****

您可以使用操作员轻松更新现有部署。您只需编辑 `TridentOrchestrator` cr以更新安装。

例如，请考虑需要启用 Astra Trident 以生成调试日志的情形。为此、请修补 `TridentOrchestrator` 设置 `spec.debug` to `true`：

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

之后 `TridentOrchestrator` 更新后、操作员将处理更新并修补现有安装。这可能会触发创建新 Pod 以相应地修改安装。

** Kubernetes升级**

当集群的 Kubernetes 版本升级到受支持的版本时，操作员会自动更新现有的 Astra Trident 安装并进行更改，以确保其满足 Kubernetes 版本的要求。

如果集群升级到不受支持的版本，则操作员会阻止安装 Astra Trident 。如果已随操作员安装了 Astra Trident ，则会显示一条警告，指示 Astra Trident 安装在不受支持的 Kubernetes 版本上。

**** 将使用**BlueXP (**以前称为**Cloud Manager)** 进行集群管理

使用 "使用BlueXP的Astra Trident"、您可以升级到最新版本的Astra Trident、添加和管理存储类并将其连接到工作环境、以及使用Cloud Backup Service 备份永久性卷。BlueXP支持使用Trident操作员手动或使用Helm部署Astra Trident。

使用 `tridentctl`

如果您的现有部署必须升级、或者您希望对部署进行高度自定义、则应考虑 。这是部署 Astra Trident 的传统方法。

您可以 生成Trident资源的清单。其中包括 Astra Trident 在安装过程中创建的部署，取消设置，服务帐户和集群角色。

从 22.04 版开始，每次安装 Astra Trident 时， AES 密钥将不再重新生成。在此版本中， Astra Trident 将安装一个新的机密对象，该对象会在安装之间持续存在。这意味着、 `tridentctl` 在22.04中、可以卸载先前版本的Trident、但早期版本无法卸载22.04安装。选择适当的安装_method。

选择安装模式

根据您的组织所需的_installation mode"(标准)、"Offline"(脱机)或"Remote"(远程)来确定部署过程。

标准安装

这是安装Astra Trident的最简单方法、适用于大多数不会实施网络限制的环境。标准安装模式使用默认注册表来存储所需的Trident (`docker.io`)和CSI (`registry.k8s.io`)映像。

使用标准模式时、Astra Trident安装程序将：

- 通过Internet提取容器映像
- 创建部署或节点取消命名集、以便在Kubernetes集群中所有符合条件的节点上启动Astra Trident Pod

脱机安装

在带风口或安全位置可能需要脱机安装模式。在这种情况下、您可以创建一个专用的镜像注册表或两个镜像注册表来存储所需的Trident和CSI映像。

无论注册表配置如何、CSI映像都必须驻留在一个注册表中。

远程安装

下面简要概述了远程安装过程：

- 部署适当版本的 `kubectl` 在要部署Astra Trident的远程计算机上。
- 从Kubernetes集群复制配置文件并设置 `KUBECONFIG` 远程计算机上的环境变量。
- 启动 `kubectl get nodes` 命令以验证是否可以连接到所需的Kubernetes集群。
- 使用标准安装步骤从远程计算机完成部署。

根据您的方法和模式选择过程

做出决定后、请选择相应的流程。

方法	安装模式
Trident运算符(手动)	"标准安装" "脱机安装"
Trident运算符(Helm)	"标准安装" "脱机安装"
`tridentctl`	"标准或脱机安装"

在安装方法之间移动

您可以决定更改安装方法。在执行此操作之前、请考虑以下事项：

- 安装和卸载Astra Trident时、请始终使用相同的方法。如果您已使用部署 `tridentctl`、您应使用的相应版本 `tridentctl` 用于卸载Astra Trident的二进制文件。同样、如果要使用操作员进行部署、则应编辑

`TridentOrchestrator` CR和设置 `spec.uninstall=true` 卸载Astra Trident。

- 如果您的部署基于操作员、则要删除此部署并改用此部署 `tridentctl` 要部署Astra Trident、您应先编辑 `TridentOrchestrator` 并设置 `spec.uninstall=true` 卸载Astra Trident。然后删除 `TridentOrchestrator` 和操作员部署。然后、您可以使用安装 `tridentctl`。
- 如果您使用的是基于操作员的手动部署、并且要使用基于Helm的Trident操作员部署、则应先手动卸载此操作员、然后再执行Helm安装。这样， Helm 就可以使用所需的标签和标注来部署 Trident 操作员。如果不执行此操作，则基于 Helm 的 Trident 操作员部署将失败，并显示标签验证错误和标注验证错误。如果您有 `tridentctl`基于部署、您可以使用基于Helm的部署、而不会遇到问题。

其他已知配置选项

在 VMware Tanzu Portfolio 产品上安装 Astra Trident 时：

- 集群必须支持有权限的工作负载。
- 。 `--kubelet-dir` 标志应设置为kubelet目录的位置。默认情况下、此值为 `/var/vcap/data/kubelet`。

  使用指定kubelet位置 `--kubelet-dir` 已知适用于Trident操作员、Helm和 `tridentctl` 部署。

使用Trident操作员安装

手动部署Trident操作员(标准模式)

您可以手动部署Trident操作员以安装Astra Trident。此过程将处理适用场景 安装、其中、Astra Trident所需的容器映像不会存储在专用注册表中。如果您有专用映像注册表、请使用 "脱机部署过程"。

有关Astra Trident 23.01的关键信息

您必须阅读以下有关Astra Trident的重要信息。

 中有关Astra 的信息

- Trident现在支持Kubernetes 1.26。在升级Kubernetes之前升级Trident。
- Astra Trident会严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

手动部署Trident操作员并安装Trident

请查看 "安装概述" 以确保满足安装前提条件并为您的环境选择正确的安装选项。

开始之前

开始安装之前、请登录到Linux主机并验证它是否正在管理一个正常运行的、 "支持的 Kubernetes 集群" 并且您拥有必要的特权。

使用OpenShift `oc` 而不是 `kubectl` 在下面的所有示例中、运行以*系统：admin*身份登录 `oc login -u system:admin` 或 `oc login -u kube-admin`。

1. 验证Kubernetes版本：

```
kubectl version
```

2. 验证集群管理员权限：

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

3. 验证您是否可以从Docker Hub启动使用映像的POD并通过POD网络访问存储系统：

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

第**1**步：下载**Trident**安装程序包

Astra Trident安装程序包包含部署Trident操作员和安装Astra Trident所需的所有内容。从下载并提取最新版本的Trident安装程序 "GitHub上的_assets_部分"。

```
wget https://github.com/NetApp/trident/releases/download/v23.01.1/trident-installer-23.01.1.tar.gz
tar -xf trident-installer-23.01.1.tar.gz
cd trident-installer
```

第**2**步：创建 `TridentOrchestrator` **CRD**

创建 `TridentOrchestrator` 自定义资源定义(CRD)。您将创建 `TridentOrchestrator` 稍后自定义资源。使用中相应的CRD YAML版本 `deploy/crds` 以创建 `TridentOrchestrator` CRD。

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

第**3**步：部署**Trident**操作员

Astra Trident安装程序提供了一个包文件、可用于安装操作员和创建关联对象。使用此捆绑包文件可以轻松地部署操作员并使用默认配置安装Astra Trident。

- 对于运行Kubernetes 1.24或更低版本的集群、请使用 `bundle_pre_1_25.yaml`。

- 对于运行Kubernetes 1.25或更高版本的集群、请使用 `bundle_post_1_25.yaml`。

Trident安装程序在中部署操作员 `trident` 命名空间。如果 `trident` 命名空间不存在、请使用 `kubectl apply -f deploy/namespace.yaml` 以创建它。

步骤

1. 创建资源并部署操作员：

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

> 在非命名空间中部署操作员 `trident` 命名空间、更新 `serviceaccount.yaml`，`clusterrolebinding.yaml` 和 `operator.yaml` 并使用生成捆绑包文件 `kustomization.yaml`：
>
> ```
> kubectl kustomize deploy/ > deploy/<bundle>.yaml
> ```

2. 验证是否已部署此操作员。

```
kubectl get deployment -n <operator-namespace>

NAME                READY   UP-TO-DATE   AVAILABLE   AGE
trident-operator    1/1     1            1           3m
```

> 在 Kubernetes 集群中只能有 * 一个操作符实例 * 。请勿创建 Trident 操作员的多个部署。

第4步：创建 `TridentOrchestrator` 并安装Trident

现在、您可以创建 `TridentOrchestrator` 并安装Astra Trident。您也可以选择 "自定义Trident安装" 使用中的属性 `TridentOrchestrator` 规格

```
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:23.01
    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:23.01.1
  Message:                  Trident installed  Namespace:
trident
  Status:                   Installed
  Version:                  v23.01.1
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

验证安装。

可以通过多种方法验证您的安装。

使用 `TridentOrchestrator` **status**

的状态 `TridentOrchestrator` 指示安装是否成功、并显示已安装的Trident版本。在安装期间、的状态 `TridentOrchestrator` 更改自 `Installing` to `Installed`。如果您观察到 `Failed` 状态、并且操作员无法自行恢复、 "检查日志"。

Status	Description
安装	操作员正在使用此安装Astra Trident `TridentOrchestrator` CR.
已安装	Astra Trident 已成功安装。
正在卸载	操作员正在卸载Astra Trident、因为 `spec.uninstall=true`。
已卸载	Astra Trident 已卸载。
失败	操作员无法安装，修补，更新或卸载 Astra Trident ；操作员将自动尝试从此状态恢复。如果此状态仍然存在，则需要进行故障排除。
正在更新	操作员正在更新现有安装。
error	。 `TridentOrchestrator` 未使用。另一个已存在。

正在使用**POD**创建状态

您可以通过查看已创建Pod的状态来确认Astra Trident安装是否已完成：

```
kubectl get pods -n trident

NAME                                READY   STATUS    RESTARTS
AGE
trident-controller-7d466bf5c7-v4cpw        6/6     Running   0
1m
trident-node-linux-mr6zc                   2/2     Running   0
1m
trident-node-linux-xrp7w                   2/2     Running   0
1m
trident-node-linux-zh2jt                   2/2     Running   0
1m
trident-operator-766f7b8658-ldzsv          1/1     Running   0
3m
```

使用 `tridentctl`

您可以使用 `tridentctl` 检查安装的Astra Trident版本。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 23.01.1        | 23.01.1        |
+----------------+----------------+
```

下一步行动

现在可以了 "创建创建后端和存储类、配置卷并将卷挂载到Pod中"。

手动部署Trident操作员(脱机模式)

您可以手动部署Trident操作员以安装Astra Trident。此过程将处理适用场景 安装、其中、Astra Trident所需的容器映像存储在专用注册表中。如果您没有专用映像注册表、请使用 "标准部署流程"。

有关Astra Trident 23.01的关键信息

您必须阅读以下有关Astra Trident的重要信息。

** 中有关Astra 的信息**

- Trident现在支持Kubernetes 1.26。在升级Kubernetes之前升级Trident。
- Astra Trident会严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

手动部署Trident操作员并安装Trident

请查看 "安装概述" 以确保满足安装前提条件并为您的环境选择正确的安装选项。

开始之前

登录到Linux主机并验证它是否正在管理正常工作的和 "支持的 Kubernetes 集群" 并且您拥有必要的特权。

> 使用OpenShift `oc` 而不是 `kubectl` 在下面的所有示例中、运行以*系统：admin*身份登录 `oc login -u system:admin` 或 `oc login -u kube-admin`。

1. 验证Kubernetes版本：

```
kubectl version
```

2. 验证集群管理员权限：

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

3. 验证您是否可以从Docker Hub启动使用映像的POD并通过POD网络访问存储系统：

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

第**1**步：下载**Trident**安装程序包

Astra Trident安装程序包包含部署Trident操作员和安装Astra Trident所需的所有内容。从下载并提取最新版本的Trident安装程序 "GitHub上的_assets_部分"。

```
wget https://github.com/NetApp/trident/releases/download/v23.01.1/trident-installer-23.01.1.tar.gz
tar -xf trident-installer-23.01.1.tar.gz
cd trident-installer
```

第**2**步：创建 `TridentOrchestrator` **CRD**

创建 `TridentOrchestrator` 自定义资源定义(CRD)。您将创建 `TridentOrchestrator` 稍后自定义资源。使用中相应的CRD YAML版本 `deploy/crds` 以创建 `TridentOrchestrator` CRD：

```
kubectl create -f deploy/crds/<VERSION>.yaml
```

第**3**步：更新操作符中的注册表位置

在中 `/deploy/operator.yaml`、更新 `image: docker.io/netapp/trident-operator:23.01.1` 以反映映像注册表的位置。Your "Trident和CSI映像" 可以位于一个注册表或不同的注册表中、但所有CSI映像都必须位于同一注册表中。例如：

- `image: <your-registry>/trident-operator:23.01.1` 如果您的映像全部位于一个注册表中。
- `image: <your-registry>/netapp/trident-operator:23.01.1` 如果Trident映像与CSI映像位于不同的注册表中。

第4步：部署Trident操作员

Trident安装程序在中部署操作员 `trident` 命名空间。如果 `trident` 命名空间不存在、请使用 `kubectl apply -f deploy/namespace.yaml` 以创建它。

在非命名空间中部署操作员 `trident` 命名空间、更新 `serviceaccount.yaml`，`clusterrolebinding.yaml` 和 `operator.yaml` 部署操作员之前。

1. 创建资源并部署操作员：

```
kubectl kustomize deploy/ > deploy/<BUNDLE>.yaml
```

Astra Trident安装程序提供了一个包文件、可用于安装操作员和创建关联对象。使用此捆绑包文件可以轻松地部署操作员并使用默认配置安装Astra Trident。

- 对于运行Kubernetes 1.24或更低版本的集群、请使用 `bundle_pre_1_25.yaml`。
- 对于运行Kubernetes 1.25或更高版本的集群、请使用 `bundle_post_1_25.yaml`。

2. 验证是否已部署此操作员。

```
kubectl get deployment -n <operator-namespace>

NAME                READY   UP-TO-DATE   AVAILABLE   AGE
trident-operator    1/1     1            1           3m
```

在 Kubernetes 集群中只能有 * 一个操作符实例 * 。请勿创建 Trident 操作员的多个部署。

第5步：更新中的映像注册表位置 `TridentOrchestrator`

Your "Trident和CSI映像" 可以位于一个注册表或不同的注册表中、但所有CSI映像都必须位于同一注册表中。更新 `deploy/crds/tridentorchestrator_cr.yaml` 根据注册表配置添加其他位置规格。

一个注册表中的映像

```
imageRegistry: "<your-registry>"
autosupportImage: "<your-registry>/trident-autosupport:23.01"
tridentImage: "<your-registry>/trident:23.01.1"
```

不同注册表中的映像

您必须附加 `sig-storage` 到 `imageRegistry` 使用不同的注册表位置。

```
imageRegistry: "<your-registry>/sig-storage"
autosupportImage: "<your-registry>/netapp/trident-autosupport:23.01"
tridentImage: "<your-registry>/netapp/trident:23.01.1"
```

第6步：创建 `TridentOrchestrator` 并安装Trident

现在、您可以创建 `TridentOrchestrator` 并安装Astra Trident。您也可以选择继续操作 "自定义Trident安装" 使用中的属性 `TridentOrchestrator` 规格以下示例显示了Trident和CSI映像位于不同注册表中的安装。

```
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:
  Autosupport Image:  <your-registry>/netapp/trident-autosupport:23.01
  Debug:              true
  Image Registry:     <your-registry>/sig-storage
  Namespace:          trident
  Trident Image:      <your-registry>/netapp/trident:23.01.1
Status:
  Current Installation Params:
    IPv6:                       false
    Autosupport Hostname:
    Autosupport Image:          <your-registry>/netapp/trident-
autosupport:23.01
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                      true
    Http Request Timeout:       90s
    Image Pull Secrets:
    Image Registry:       <your-registry>/sig-storage
    k8sTimeout:           30
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Probe Port:           17546
    Silence Autosupport:  false
    Trident Image:        <your-registry>/netapp/trident:23.01.1
  Message:                Trident installed
  Namespace:              trident
  Status:                 Installed
  Version:                v23.01.1
Events:
    Type Reason Age From Message ---- ------ ---- ---- -------Normal
    Installing 74s trident-operator.netapp.io Installing Trident Normal
    Installed 67s trident-operator.netapp.io Trident installed
```

验证安装。

可以通过多种方法验证您的安装。

使用 `TridentOrchestrator` status

的状态 `TridentOrchestrator` 指示安装是否成功、并显示已安装的Trident版本。在安装期间、的状态 `TridentOrchestrator` 更改自 `Installing` to `Installed`。如果您观察到 `Failed` 状态、并且操作员无法自行恢复、 "检查日志"。

Status	Description
安装	操作员正在使用此安装Astra Trident `TridentOrchestrator` CR.
已安装	Astra Trident 已成功安装。
正在卸载	操作员正在卸载Astra Trident、因为 `spec.uninstall=true`。
已卸载	Astra Trident 已卸载。
失败	操作员无法安装，修补，更新或卸载 Astra Trident ；操作员将自动尝试从此状态恢复。如果此状态仍然存在，则需要进行故障排除。
正在更新	操作员正在更新现有安装。
error	。 `TridentOrchestrator` 未使用。另一个已存在。

正在使用POD创建状态

您可以通过查看已创建Pod的状态来确认Astra Trident安装是否已完成：

```
kubectl get pods -n trident

NAME                                READY   STATUS    RESTARTS
AGE
trident-controller-7d466bf5c7-v4cpw        6/6     Running   0
1m
trident-node-linux-mr6zc                   2/2     Running   0
1m
trident-node-linux-xrp7w                   2/2     Running   0
1m
trident-node-linux-zh2jt                   2/2     Running   0
1m
trident-operator-766f7b8658-ldzsv          1/1     Running   0
3m
```

使用 `tridentctl`

您可以使用 `tridentctl` 检查安装的Astra Trident版本。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 23.01.1        | 23.01.1        |
+----------------+----------------+
```

下一步行动

现在可以了 "创建创建后端和存储类、配置卷并将卷挂载到Pod中"。

使用**Helm**部署**Trident**操作员(标准模式)

您可以使用Helm部署Trident操作员并安装Astra Trident。此过程将处理适用场景 安装、其中、Astra Trident所需的容器映像不会存储在专用注册表中。如果您有专用映像注册表、请使用 "脱机部署过程"。

有关**Astra Trident 23.01**的关键信息

您必须阅读以下有关**Astra Trident**的重要信息。

**** 中有关**Astra ** 的信息

- Trident现在支持Kubernetes 1.26。在升级Kubernetes之前升级Trident。
- Astra Trident会严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

部署**Trident**操作员并使用**Helm**安装**Astra Trident**

使用Trident "Helm图表" 您可以一步部署Trident操作员并安装Trident。

请查看 "安装概述" 以确保满足安装前提条件并为您的环境选择正确的安装选项。

开始之前

除了 "部署前提条件" 您需要 "Helm 版本 3"。

步骤

1. 添加Astra Trident Helm存储库：

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

2. 使用 ... `helm install` 并为您的部署指定一个名称、如以下示例中所示 `23.01.1` 是您要安装的Astra Trident版本。

```
helm install <name> netapp-trident/trident-operator --version 23.01.1
--create-namespace --namespace <trident-namespace>
```

> 如果您已为Trident创建命名空间、则会显示 `--create-namespace` 参数不会创建其他命名空间。

您可以使用 `helm list` 查看安装详细信息、例如名称、命名空间、图表、状态、应用程序版本、和修订版号。

在安装期间传递配置数据

在安装期间，可以通过两种方式传递配置数据：

选项	Description
`--values` (或 `-f`)	指定包含覆盖的YAML文件。可以多次指定此值，最右侧的文件将优先。
`--set`	在命令行上指定覆盖。

例如、要更改的默认值 `debug`、运行以下命令 `--set` 命令位置 `23.01.1` 是您要安装的Astra Trident版本：

```
helm install <name> netapp-trident/trident-operator --version 23.01.1
--create-namespace --namespace --set tridentDebug=true
```

配置选项

此表和 `values.yaml` 文件(属于Helm图表的一部分)提供了键列表及其默认值。

选项	Description	Default
`nodeSelector`	用于POD分配的节点标签	
`podAnnotations`	POD标注	
`deploymentAnnotations`	部署标注	
`tolerations`	POD分配的差值	
`affinity`	用于Pod分配的相关性	

选项	Description	Default
tridentControllerPluginNodeSelector	Pod的其他节点选择器。请参见 了解控制器Pod和节点Pod 了解详细信息。	
tridentControllerPluginTolerations	覆盖Kubernetes对Pod的容错。请参见 了解控制器Pod和节点Pod 了解详细信息。	
tridentNodePluginNodeSelector	Pod的其他节点选择器。请参见 了解控制器Pod和节点Pod 了解详细信息。	
tridentNodePluginTolerations	覆盖Kubernetes对Pod的容错。请参见 了解控制器Pod和节点Pod 了解详细信息。	
imageRegistry	标识的注册表 trident-operator，`trident`和其他图像。留空以接受默认值。	""
imagePullPolicy	设置的映像提取策略 trident-operator。	IfNotPresent
imagePullSecrets	设置的映像提取密钥 trident-operator，`trident`和其他图像。	
kubeletDir	允许覆盖kubelet内部状态的主机位置。	"/var/lib/kubelet"`
operatorLogLevel	允许将Trident操作符的日志级别设置为： trace， debug， info， warn， error`或 `fatal。	"info"
operatorDebug	允许将Trident操作符的日志级别设置为DEBUG。	true
operatorImage	允许完全覆盖的映像 trident-operator。	""
operatorImageTag	允许覆盖的标记 trident-operator 图像。	""
tridentIPv6	允许在IPv6集群中使用Astra Trident。	false
tridentK8sTimeout	覆盖大多数Kubernetes API操作的默认30秒超时(如果不为零、则以秒为单位)。	0
tridentHttpRequestTimeout	使用覆盖HTTP请求的默认90秒超时 0s 为超时的无限持续时间。不允许使用负值。	"90s"
tridentSilenceAutosupport	允许禁用Astra Trident定期AutoSupport 报告。	false
tridentAutosupportImageTag	允许覆盖Astra Trident AutoSupport容器的映像标记。	<version>

选项	Description	Default
tridentAutosupportProxy	允许Astra Trident AutoSupport 容器通过HTTP代理进行回拨。	""
tridentLogFormat	设置Astra Trident日志记录格式(text 或 json）。	"text"
tridentDisableAuditLog	禁用Astra Trident审核日志程序。	true
tridentLogLevel	允许将Astra Trident的日志级别设置为： trace， debug， info， warn， error`或 `fatal。	"info"
tridentDebug	允许将Astra Trident的日志级别设置为 debug。	false
tridentLogWorkflows	允许为跟踪日志记录或日志禁止启用特定的Astra Trident工作流。	""
tridentLogLayers	允许为跟踪日志记录或日志禁止启用特定的Astra Trident层。	""
tridentImage	允许完全覆盖Astra Trident的映像。	""
tridentImageTag	允许覆盖Astra Trident的映像标记。	""
tridentProbePort	允许覆盖用于Kubernetes活动/就绪性探测的默认端口。	""
windows	允许在Windows工作节点上安装Astra Trident。	false
enableForceDetach	允许启用强制分离功能。	false
excludePodSecurityPolicy	从创建过程中排除操作员POD安全策略。	false

了解控制器**Pod**和节点**Pod**

Astra Trident作为一个控制器POD运行、并在集群中的每个工作节点上运行一个节点POD。节点POD必须在任何可能要挂载Astra Trident卷的主机上运行。

Kubernetes "节点选择器" 和 "容忍和损害" 用于限制Pod在特定节点或首选节点上运行。使用`ControllerPlugin`和 NodePlugin、您可以指定约束和覆盖。

- 控制器插件负责卷配置和管理、例如快照和调整大小。
- 节点插件负责将存储连接到节点。

下一步行动

现在可以了 "创建创建后端和存储类、配置卷并将卷挂载到Pod中"。

使用**Helm**部署**Trident**操作员(脱机模式)

您可以使用Helm部署Trident操作员并安装Astra Trident。此过程将处理适用场景 安装、其中、Astra Trident所需的容器映像存储在专用注册表中。如果您没有专用映像注册表、请

使用 "标准部署流程"。

有关Astra Trident 23.01的关键信息

您必须阅读以下有关Astra Trident的重要信息。

** 中有关Astra 的信息**

- Trident现在支持Kubernetes 1.26。在升级Kubernetes之前升级Trident。
- Astra Trident会严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

部署Trident操作员并使用Helm安装Astra Trident

使用Trident "Helm图表" 您可以一步部署Trident操作员并安装Trident。

请查看 "安装概述" 以确保满足安装前提条件并为您的环境选择正确的安装选项。

开始之前

除了 "部署前提条件" 您需要 "Helm 版本 3"。

步骤

1. 添加Astra Trident Helm存储库：

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

2. 使用 ... `helm install` 并为部署和映像注册表位置指定一个名称。Your "Trident和CSI映像" 可以位于一个注册表或不同的注册表中、但所有CSI映像都必须位于同一注册表中。在示例中、 `23.01.1` 是您要安装的Astra Trident版本。

一个注册表中的映像

```
helm install <name> netapp-trident/trident-operator --version
23.01.1 --set imageRegistry=<your-registry> --create-namespace
--namespace <trident-namespace>
```

不同注册表中的映像

您必须附加 `sig-storage` 到 `imageRegistry` 使用不同的注册表位置。

```
helm install <name> netapp-trident/trident-operator --version
23.01.1 --set imageRegistry=<your-registry>/sig-storage --set
operatorImage=<your-registry>/netapp/trident-operator:23.01.1 --set
tridentAutosupportImage=<your-registry>/netapp/trident-
autosupport:23.01 --set tridentImage=<your-
registry>/netapp/trident:23.01.1 --create-namespace --namespace
<trident-namespace>
```

如果您已为Trident创建命名空间、则会显示 `--create-namespace` 参数不会创建其他命名空间。

您可以使用 `helm list` 查看安装详细信息、例如名称、命名空间、图表、状态、应用程序版本、和修订版号。

在安装期间传递配置数据

在安装期间，可以通过两种方式传递配置数据：

选项	Description
`--values` (或 `-f`)	指定包含覆盖的YAML文件。可以多次指定此值，最右侧的文件将优先。
`--set`	在命令行上指定覆盖。

例如、要更改的默认值 `debug`、运行以下命令 `--set` 命令位置 `23.01.1` 是您要安装的Astra Trident版本：

```
helm install <name> netapp-trident/trident-operator --version 23.01.1
--create-namespace --namespace --set tridentDebug=true
```

配置选项

此表和 `values.yaml` 文件(属于Helm图表的一部分)提供了键列表及其默认值。

选项	Description	Default
nodeSelector	用于POD分配的节点标签	
podAnnotations	POD标注	
deploymentAnnotations	部署标注	
tolerations	POD分配的差值	
affinity	用于Pod分配的相关性	
tridentControllerPluginNodeSelector	Pod的其他节点选择器。请参见 了解控制器Pod和节点Pod 了解详细信息。	
tridentControllerPluginTolerations	覆盖Kubernetes对Pod的容错。请参见 了解控制器Pod和节点Pod 了解详细信息。	
tridentNodePluginNodeSelector	Pod的其他节点选择器。请参见 了解控制器Pod和节点Pod 了解详细信息。	
tridentNodePluginTolerations	覆盖Kubernetes对Pod的容错。请参见 了解控制器Pod和节点Pod 了解详细信息。	
imageRegistry	标识的注册表 trident-operator，`trident`和其他图像。留空以接受默认值。	""
imagePullPolicy	设置的映像提取策略 trident-operator。	IfNotPresent
imagePullSecrets	设置的映像提取密钥 trident-operator，`trident`和其他图像。	
kubeletDir	允许覆盖kubelet内部状态的主机位置。	"/var/lib/kubelet"`
operatorLogLevel	允许将Trident操作符的日志级别设置为： trace，debug，info，warn，error`或 `fatal。	"info"
operatorDebug	允许将Trident操作符的日志级别设置为DEBUG。	true
operatorImage	允许完全覆盖的映像 trident-operator。	""
operatorImageTag	允许覆盖的标记 trident-operator 图像。	""
tridentIPv6	允许在IPv6集群中使用Astra Trident。	false
tridentK8sTimeout	覆盖大多数Kubernetes API操作的默认30秒超时(如果不为零、则以秒为单位)。	0

选项	Description	Default
tridentHttpRequestTimeout	使用覆盖HTTP请求的默认90秒超时 0s 为超时的无限持续时间。不允许使用负值。	"90s"
tridentSilenceAutosupport	允许禁用Astra Trident定期AutoSupport 报告。	false
tridentAutosupportImageTag	允许覆盖Astra Trident AutoSupport容器的映像标记。	<version>
tridentAutosupportProxy	允许Astra Trident AutoSupport 容器通过HTTP代理进行回拨。	""
tridentLogFormat	设置Astra Trident日志记录格式(text 或 json）。	"text"
tridentDisableAuditLog	禁用Astra Trident审核日志程序。	true
tridentLogLevel	允许将Astra Trident的日志级别设置为： trace， debug， info， warn， error`或 `fatal。	"info"
tridentDebug	允许将Astra Trident的日志级别设置为 debug。	false
tridentLogWorkflows	允许为跟踪日志记录或日志禁止启用特定的Astra Trident工作流。	""
tridentLogLayers	允许为跟踪日志记录或日志禁止启用特定的Astra Trident层。	""
tridentImage	允许完全覆盖Astra Trident的映像。	""
tridentImageTag	允许覆盖Astra Trident的映像标记。	""
tridentProbePort	允许覆盖用于Kubernetes活动/就绪性探测的默认端口。	""
windows	允许在Windows工作节点上安装Astra Trident。	false
enableForceDetach	允许启用强制分离功能。	false
excludePodSecurityPolicy	从创建过程中排除操作员POD安全策略。	false

了解控制器Pod和节点Pod

Astra Trident作为一个控制器POD运行、并在集群中的每个工作节点上运行一个节点POD。节点POD必须在任何可能要挂载Astra Trident卷的主机上运行。

Kubernetes "节点选择器" 和 "容忍和损害" 用于限制Pod在特定节点或首选节点上运行。使用`ControllerPlugin`和 NodePlugin、您可以指定约束和覆盖。

- 控制器插件负责卷配置和管理、例如快照和调整大小。
- 节点插件负责将存储连接到节点。

下一步行动

现在可以了 "创建创建后端和存储类、配置卷并将卷挂载到Pod中"。

自定义**Trident**操作员安装

使用Trident操作员可以使用中的属性自定义Astra Trident安装 `TridentOrchestrator` 规格如果您要对安装进行自定义、使其超出预期范围 `TridentOrchestrator` 参数允许、请考虑使用 `tridentctl` 生成自定义YAML清单以根据需要进行修改。

了解控制器**Pod**和节点**Pod**

Astra Trident作为一个控制器POD运行、并在集群中的每个工作节点上运行一个节点POD。节点POD必须在任何可能要挂载Astra Trident卷的主机上运行。

Kubernetes "节点选择器" 和 "容忍和损害" 用于限制Pod在特定节点或首选节点上运行。使用`ControllerPlugin`和 `NodePlugin`、您可以指定约束和覆盖。

- 控制器插件负责卷配置和管理、例如快照和调整大小。
- 节点插件负责将存储连接到节点。

配置选项

> `spec.namespace` 在中指定 `TridentOrchestrator` 表示安装了Astra Trident的命名空间。此参数 * 安装 Astra Trident 后无法更新 * 。如果尝试执行此操作、则会导致 `TridentOrchestrator` 要更改为的状态 `Failed`。Astra Trident不能跨命名空间迁移。

此表详细介绍了相关信息 `TridentOrchestrator` 属性。

参数	Description	Default
`namespace`	用于安装 Astra Trident 的命名空间	default
`debug`	为 Astra Trident 启用调试	false
`windows`	设置为 `true` 启用在Windows工作节点上的安装。	false
`IPv6`	安装基于 IPv6 的 Astra Trident	false
`k8sTimeout`	Kubernetes 操作超时	30 秒
`silenceAutosupport`	不要自动向 NetApp 发送 AutoSupport 捆绑包	false
`enableNodePrep`	自动管理工作节点依赖关系（ * 测试版 * ）	false
`autosupportImage`	AutoSupport 遥测的容器映像	"NetApp/trident自动支持：23.01"

参数	Description	Default
`autosupportProxy`	用于发送 AutoSupport 遥测的代理的地址 / 端口	"http://proxy.example.com:8888"
`uninstall`	用于卸载 Astra Trident 的标志	false
`logFormat`	要使用的 Astra Trident 日志记录格式 [text ， json]	文本
`tridentImage`	要安装的 Astra Trident 映像	"NetApp/Trident： 21.04"
`imageRegistry`	内部注册表的路径、格式 `<registry FQDN>[:port][/subpath]`	"K8s.gcr.io/SIG-storage (K8s 1.19+)或quay.io/k8scsi "
`kubeletDir`	主机上的 kubelet 目录的路径	"/var/lib/kubelet"
`wipeout`	要删除以执行 Astra Trident 完全删除的资源列表	
`imagePullSecrets`	从内部注册表中提取映像的机密信息	
`imagePullPolicy`	设置Trident运算符的映像提取策略。有效值为：`Always` 以始终提取映像。`IfNotPresent` 仅当节点上尚不存在映像时才提取该映像。`Never` 从不提取映像。	`IfNotPresent`
`controllerPluginNodeSelector`	Pod的其他节点选择器。格式与 pod.spec.nodeSelector 相同。	无默认值；可选
`controllerPluginTolerations`	覆盖Kubernetes对Pod的容错。格式与 po.spec.Tolerations 相同。	无默认值；可选
`nodePluginNodeSelector`	Pod的其他节点选择器。格式与 pod.spec.nodeSelector 相同。	无默认值；可选
`nodePluginTolerations`	覆盖Kubernetes对Pod的容错。格式与 po.spec.Tolerations 相同。	无默认值；可选

有关格式化 POD 参数的详细信息，请参见 "将 Pod 分配给节点"。

配置示例

您可以在定义时使用上述属性 `TridentOrchestrator` 自定义安装。

示例**1**：基本自定义配置

这是一个基本自定义配置示例。

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

示例**2**：使用节点选择器部署

此示例说明了如何使用节点选择器部署Trident：

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

示例3：在Windows工作节点上部署

此示例说明了如何在Windows工作节点上部署。

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

使用tridentctl进行安装

使用tridentctl进行安装

您可以使用安装Astra Trident `tridentctl`。此过程将处理适用场景 安装、其中、Astra Trident所需的容器映像是否存储在专用注册表中。以自定义 `tridentctl` 部署、请参见 "自定义 tridentctl 部署"。

有关Astra Trident 23.01的关键信息

您必须阅读以下有关Astra Trident的重要信息。

 中有关Astra 的信息

- Trident现在支持Kubernetes 1.26。在升级Kubernetes之前升级Trident。
- Astra Trident会严格强制在SAN环境中使用多路径配置、建议值为 `find_multipaths: no` 在multipath.conf文件中。

  使用非多路径配置或 `find_multipaths: yes` 或 `find_multipaths: smart` multipath.conf文件中的值将导致挂载失败。Trident已建议使用 `find_multipaths: no` 自21.07版起。

使用安装Astra Trident `tridentctl`

请查看 "安装概述" 以确保满足安装前提条件并为您的环境选择正确的安装选项。

开始之前

开始安装之前、请登录到Linux主机并验证它是否正在管理一个正常运行的、 "支持的 Kubernetes 集群" 并且您拥有必要的特权。

使用OpenShift `oc` 而不是 `kubectl` 在下面的所有示例中、运行以*系统：admin*身份登录 `oc login -u system:admin` 或 `oc login -u kube-admin`。

1. 验证Kubernetes版本：

```
kubectl version
```

2. 验证集群管理员权限：

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

3. 验证您是否可以从Docker Hub启动使用映像的POD并通过POD网络访问存储系统：

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

第1步：下载Trident安装程序包

Astra Trident安装程序包可创建Trident Pod、配置用于保持其状态的CRD对象、并初始化CSI sidecars以执行配置卷以及将卷附加到集群主机等操作。从下载并提取最新版本的Trident安装程序 "GitHub上的_assets_部分"。在示例中、使用选定的<trident-installer-XX.XX.X.tar.gz> Trident版本更新_Astra Trident。

```
wget https://github.com/NetApp/trident/releases/download/v23.01.1/trident-
installer-23.01.1.tar.gz
tar -xf trident-installer-23.01.1.tar.gz
cd trident-installer
```

第2步：安装Astra Trident

通过执行在所需命名空间中安装Astra Trident `tridentctl install` 命令：您可以添加其他参数来指定映像注册表位置。

要使Astra Trident能够在Windows节点上运行、请添加 `--windows` 安装命令的标志： `$ ./tridentctl install --windows -n trident`。

标准模式

```
./tridentctl install -n trident
```

一个注册表中的映像

```
./tridentctl install -n trident --image-registry <your-registry>
--autosupport-image <your-registry>/trident-autosupport:23.01 --trident
-image <your-registry>/trident:23.01.1
```

不同注册表中的映像

您必须附加 `sig-storage` 到 `imageRegistry` 使用不同的注册表位置。

```
./tridentctl install -n trident --image-registry <your-registry>/sig-
storage --autosupport-image <your-registry>/netapp/trident-
autosupport:23.01 --trident-image <your-
registry>/netapp/trident:23.01.1
```

您的安装状态应如下所示。

```
....
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-controller-679648bd45-cv2mx
INFO Waiting for Trident REST interface.
INFO Trident REST interface is up.                 version=23.01.1
INFO Trident installation succeeded.
....
```

验证安装。

您可以使用POD创建状态或验证安装 `tridentctl`。

正在使用**POD**创建状态

您可以通过查看已创建Pod的状态来确认Astra Trident安装是否已完成：

```
kubectl get pods -n trident

NAME                                READY   STATUS    RESTARTS   AGE
trident-controller-679648bd45-cv2mx   6/6     Running   0          5m29s
trident-node-linux-vgc8n              2/2     Running   0          5m29s
```

如果安装程序未成功完成或 `trident-controller-<generated id>` (`trident-csi-<generated id>` 在23.01之前的版本中)不具有*正在运行*状态、表示未安装此平台。使用 `... -d` to "打开调试模式" 并对问题描述 进行故障排除。

使用 `tridentctl`

您可以使用 `tridentctl` 检查安装的Astra Trident版本。

```
./tridentctl -n trident version

+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 23.01.1        | 23.01.1        |
+----------------+----------------+
```

下一步行动

现在可以了 "创建创建后端和存储类、配置卷并将卷挂载到Pod中"。

自定义**tridentctl**安装

您可以使用Astra Trident安装程序自定义安装。

了解安装程序

使用Astra Trident安装程序可以自定义属性。例如、如果已将Trident映像复制到专用存储库、则可以使用指定映像名称 `--trident-image`。如果已将Trident映像以及所需的CSI sidecar映像复制到专用存储库、则最好使用指定该存储库的位置 `--image-registry` 交换机、其形式为 `<registry FQDN>[:port]`。

如果您使用的是Kubernetes的分发版、其中 `kubelet` 将其数据保留在非正常路径上 `/var/lib/kubelet`、您可以使用指定备用路径 `--kubelet-dir`。

如果您需要自定义安装，使其超出安装程序参数的允许范围，则还可以自定义部署文件。使用 `--generate-custom-yaml` 参数将在安装程序中创建以下YAML文件 `setup` 目录：

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

生成这些文件后、您可以根据需要进行修改、然后使用 `--use-custom-yaml` 安装自定义部署。

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

下一步是什么?

安装Astra Trident后、您可以继续创建后端、创建存储类、配置卷以及将卷挂载到Pod中。

第 1 步：创建后端

现在，您可以继续创建一个后端，供 Astra Trident 配置卷使用。为此、请创建 `backend.json` 包含必要参数的文件。可在中找到不同后端类型的示例配置文件 `sample-input` 目录。

请参见 "此处" 有关如何为后端类型配置文件的更多详细信息。

```
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 |
+-------------+----------------+--------------------------------------
+--------+---------+
```

如果创建失败，则后端配置出现问题。您可以运行以下命令来查看日志以确定发生原因：

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

解决问题后，只需返回到此步骤的开头并重试。有关更多故障排除提示，请参见 "故障排除" 部分。

第 2 步：创建存储类

Kubernetes 用户使用指定的永久性卷声明（ Persistent Volume Claim ， PVC ）配置卷 "存储类" 按名称。详细信息对用户隐藏，但存储类可标识用于该类的配置程序（在本例中为 Trident ）以及该类对配置程序的含义。

创建存储类 Kubernetes 用户将指定何时需要卷。该类的配置需要为上一步创建的后端建模，以便 Astra Trident 可以使用它来配置新卷。

首先要使用的最简单存储类是基于的存储类 `sample-input/storage-class-csi.yaml.templ` 安装程序随附的文件、替换 *`BACKEND_TYPE`* 和存储驱动程序名称。

```
./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
```

这是一个Kubernetes对象、因此您可以使用 `kubectl` 以在Kubernetes中创建。

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

现在， Kubernetes 和 Astra Trident 都应显示 * 基本 -CSI * 存储类， Astra Trident 应已发现后端的池。

```
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"
        ]
      }
    }
  ]
}
```

第 3 步：配置第一个卷

现在，您已准备好动态配置第一个卷。可通过创建 Kubernetes 来完成此操作 "永久性卷声明" （ PVC ）对象。

为使用刚刚创建的存储类的卷创建 PVC 。

请参见 `sample-input/pvc-basic-csi.yaml` 例如。确保存储类名称与您创建的名称匹配。

```
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
```

第 4 步：将卷挂载到 Pod 中

现在，让我们挂载卷。我们将启动一个nginx POD、将PV挂载到下 `/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
```

此时， Pod （应用程序）不再存在，但卷仍在。如果需要，您可以从另一个 POD 使用它。

要删除卷，请删除声明：

```
kubectl delete pvc basic
```

现在，您可以执行其他任务，例如：

- "配置其他后端。"
- "创建其他存储类。"