入门
Astra Trident

NetApp
April 16, 2024

This PDF was generated from https://docs.netapp.com/zh-cn/trident-2110/trident-get-started/quickstart.html on April 16, 2024. Always check docs.netapp.com for the latest.

目录

入门 1
试用 1
要求 1
部署概述 5
使用 Trident 操作员进行部署 7
使用 tridentctl 进行部署 15
下一步是什么? 18

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"。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

首先查看支持的前端，后端和主机配置。

> 要了解 Astra Trident 使用的端口，请参见 "此处"。

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：

- Kubernetes 1.17 或更高版本（最新版本： 1.22 ）
- Mirantis Kubernetes Engine 3.4
- OpenShift OpenShift 4.7 ， 4.8 ， 4.9 （最新版本 4.9 ）

以下版本支持 Trident 操作符：

- Kubernetes 1.17 或更高版本（最新版本： 1.22 ）
- OpenShift OpenShift 4.7 ， 4.8 ， 4.9 （最新版本 4.9 ）

> 如果使用的版本低于 4.6-8 ， Red Hat OpenShift 容器平台用户可能会发现其 initiatorname.iscsi 文件为空。这是 RedHat 已确定的一个错误，需要通过 OpenShift 4.6-8 进行修复。请参见此内容 "错误修复公告"。NetApp 建议您在 OpenShift 4.7 及更高版本上使用 Astra Trident 。

Astra Trident 还可与其他许多完全托管和自我管理的 Kubernetes 产品结合使用，包括 Google Cloud 的 Google Kubernetes Engine （ GKS ）， AWS 的 Elastic Kubernetes Services （ EKS ）， Azure 的 Azure Kubernetes Service （ AKS ）和 Rancher 。

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
- Astra 数据存储
- Cloud Volumes ONTAP
- 适用于 AWS 的 Cloud Volumes Service
- 适用于 GCP 的 Cloud Volumes Service
- FAS/AFF/ 选择 9.3 或更高版本
- NetApp 全 SAN 阵列（ASA）
- NetApp HCI/Element 软件 8 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	1.17 及更高版本	否
卷快照	1.17 及更高版本	否
卷快照中的 PVC	1.17 及更高版本	否
iSCSI PV 调整大小	1.17 及更高版本	否
ONTAP 双向 CHAP	1.17 及更高版本	否
动态导出策略	1.17 及更高版本	否
Trident 运算符	1.17 及更高版本	否
自动工作节点准备（测试版）	1.17 及更高版本	否
CSI 拓扑	1.17 及更高版本	否

已测试主机操作系统

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。

虽然 Astra Trident 不会正式 " 支持 " 特定的操作系统，但已知以下 Linux 版本可以正常运行：

- RedHat CoreOS 4.2 和 4.3
- RHEL 或 CentOS 7.4 或更高版本

- Ubuntu 18.04 或更高版本

`tridentctl` 实用程序也可在 Linux 的任何这些分发版上运行。

主机配置

根据所使用的后端，应在集群中的所有工作人员上安装 NFS 和 / 或 iSCSI 实用程序。请参见 "此处" 有关详细信息 …

存储系统配置：

Astra Trident 可能需要先对存储系统进行一些更改，然后后端配置才能使用它。请参见 "此处" 了解详细信息。

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` 命令验证所需容器映像的列表。

Kubernetes 版本	容器映像
v1.17.0	• NetApp/trident： 21.10.0 • netapp/trident autosupport： 21.10 • K8s.gcr.io/SIG-storage/Csl-provisioner： v2.2.2 • K8s.gcr.io/SIG-storage/Csl-attacher： v3.3.0 • K8s.gcr.io/SIG-storage/Csl-resizer： v1.3.0 • K8s.gcr.io/SIG-storage/Csl-snapshotter： v3.0.3 • k8s.gcr.io/sig-storage/Csl-node-driver-registry:v2.3.0 • netapp/trident 操作符： 21.10.0 （可选）
v1.18.0	• NetApp/trident： 21.10.0 • netapp/trident autosupport： 21.10 • K8s.gcr.io/SIG-storage/Csl-provisioner： v2.2.2 • K8s.gcr.io/SIG-storage/Csl-attacher： v3.3.0 • K8s.gcr.io/SIG-storage/Csl-resizer： v1.3.0 • K8s.gcr.io/SIG-storage/Csl-snapshotter： v3.0.3 • k8s.gcr.io/sig-storage/Csl-node-driver-registry:v2.3.0 • netapp/trident 操作符： 21.10.0 （可选）

Kubernetes 版本	容器映像
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-registry:v2.3.0 • netapp/trident 操作符： 21.10.0 （可选）
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-registry:v2.3.0 • netapp/trident 操作符： 21.10.0 （可选）
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-registry:v2.3.0 • netapp/trident 操作符： 21.10.0 （可选）

Kubernetes 版本	容器映像
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-registry:v2.3.0 • netapp/trident 操作符： 21.10.0 （可选）

> 在 Kubernetes 1.20 及更高版本上，只有当 `v1` 版本提供了 `volumesnapshots.snapshot.storage.k8s.io` CRD 时，才使用经过验证的 `K8s.gcr.io/SIG-storage/CsI-snapshotter ： v4.x` image 。如果 `v1bea1` 版本在使用 / 不使用 `v1` 版本的情况下为 CRD 提供服务，请使用经验证的 `K8s.gcr.io/SIG-storage/CsI-snapshotter ： v3.x` 映像。

部署概述

您可以使用 Trident 运算符或使用 `tridentctl` 来部署 Astra Trident 。

选择部署方法

要确定要使用的部署方法，请考虑以下几点：

为什么要使用 **Trident** 运算符？

。 "Trident 运算符" 是动态管理 Astra Trident 资源并自动完成设置阶段的绝佳方式。必须满足某些前提条件。请参见 "要求"。

Trident 运算符提供了以下几项优势。

自我修复功能

您可以监控 Astra Trident 安装并主动采取措施来解决问题，例如删除部署或意外修改部署的时间。将此操作员设置为部署时，将创建 `trident` 操作符 `-<generated -id>` Pod 。此 POD 会将 `Trident Orchestrator` CR 与 Astra Trident 安装关联起来，并始终确保只有一个活动的 `TridentOrchestrator` 。换言之，操作员可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。

轻松更新现有安装

您可以使用操作员轻松更新现有部署。您只需编辑 `TridentOrchestrator` CR 即可更新安装。例如，请考虑需要启用 Astra Trident 以生成调试日志的情形。

为此，请修补您的 `TridentOrchestrator` 以将 `spec.debug` 设置为 `true`：

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

更新 `TridentOrchestrator` 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。

自动处理 **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。

为什么要使用 **Helm**？

如果您还有其他要使用 Helm 管理的应用程序，从 Astra Trident 21.01 开始，您也可以使用 Helm 管理您的部署。

应在何时使用 `tridenctl`?

如果您的现有部署必须升级到，或者您希望对部署进行高度自定义，则应查看使用 "Tridentctl"。这是部署 Astra Trident 的传统方法。

在部署方法之间移动时的注意事项

不难想象需要在部署方法之间移动的情形。在尝试从 `tridentctl` 部署迁移到基于操作员的部署之前，应考虑以下事项，反之亦然：

- 卸载 Astra Trident 时，请始终使用相同的方法。如果已使用 `tridentctl` 进行部署，则应使用适当版本的 `tridentctl` 二进制文件卸载 Astra Trident 。同样，如果要使用操作员进行部署，则应编辑 `TridentOrchestrator` CR 并设置 `sPec.uninstall=true` 以卸载 Astra Trident 。
- 如果您要删除基于操作员的部署并使用 `tridentctl` 来部署 Astra Trident ，则应先编辑 `TridentOrchestrator` 并设置 `sPec.uninstall=true` 以卸载 Astra Trident 。然后删除 `TridentOrchestrator` 和操作员部署。然后，您可以使用 `tridentctl` 进行安装。
- 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。如果您使用的是基于 `tridentctl` 的部署，则可以使用基于 Helm 的部署，而不会出现问题。

了解部署模式

部署 Astra Trident 的方法有三种。

标准部署

在 Kubernetes 集群上部署 Trident 会导致 Astra Trident 安装程序执行以下两项操作：

- 通过 Internet 提取容器映像
- 创建部署和 / 或节点取消设置，以便在 Kubernetes 集群中所有符合条件的节点上启动 Astra Trident Pod 。

此类标准部署可以通过两种不同的方式执行：

- 使用 `tridentctl install`
- 使用 Trident 运算符。您可以手动或使用 Helm 部署 Trident 操作员。

此安装模式是安装 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。

脱机部署

要执行空映射部署，您可以在调用 `tridentctl install` 时使用 ` -image-regRegistry` 标志来指向私有映像注册表。如果使用 Trident 操作符进行部署，则也可以在 `TridentOrchestrator` 中指定 `sPec.imageRegistry` 。此注册表应包含 "Trident 映像"， "Trident AutoSupport 映像"和 Kubernetes 版本所需的 CSI sidecar 映像。

要自定义部署，您可以使用 `tridentctl` 为 Trident 的资源生成清单。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。

有关自定义部署的详细信息，请参见以下链接：

- "自定义基于操作员的部署"
- "85cda507040c129ee606d4d8df583b90"

> 如果您使用的是私有映像存储库，则应在专用注册表 URL 的末尾添加 ` /k8scsi` （对于 1.17 之前的 Kubernetes 版本）或 ` /SIG-storage` （对于 1.17 之后的 Kubernetes 版本）。在使用私有注册表进行 `tridentctl` 部署时，应将 ` -trident 映像` 和 ` -autosupport-image` 与 ` -image -regRegistry` 结合使用。如果要使用 Trident 操作符部署 Astra Trident ，请确保 Orchestrator CR 在安装参数中包含 `tridentImage` 和 `autosupportImage` 。

远程部署

下面简要概述了远程部署过程：

- 在要部署 Astra Trident 的远程计算机上部署适当版本的 `kubectl` 。
- 从 Kubernetes 集群复制配置文件，并在远程计算机上设置 `KUBECONFIG` 环境变量。
- 启动 `kubectl get nodes` 命令，验证您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。

使用 Trident 操作员进行部署

您可以使用 Trident 操作员部署 Astra Trident 。您可以手动或使用 Helm 部署 Trident 操作员。

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。

您需要的内容

要部署 Astra Trident ，应满足以下前提条件：

- 您对运行 Kubernetes 1.17 及更高版本的受支持 Kubernetes 集群拥有完全权限。
- 您可以访问受支持的 NetApp 存储系统。
- 您可以从所有 Kubernetes 工作节点挂载卷。
- 您安装了一个安装了 `kubectl` （如果使用的是 OpenShift ，则为 `oc` ）的 Linux 主机，并将其配置为管理要使用的 Kubernetes 集群。
- 您已将 `KUBECONFIG` 环境变量设置为指向 Kubernetes 集群配置。
- 您已启用 "Astra Trident 所需的功能门"。
- 如果您将 Kubernetes 与 Docker Enterprise 结合使用， "按照其步骤启用 CLI 访问"。

明白了吗？太棒了！让我们开始吧。

使用 Helm 部署 Trident 操作员

执行列出的步骤以使用 Helm 部署 Trident 操作员。

您需要的内容

除了上述前提条件之外，要使用 Helm 部署 Trident 操作员，还需要满足以下条件：

- Kubernetes 1.17 及更高版本
- Helm 版本 3

步骤

1. 从下载安装程序包 "Trident GitHub" 页面。安装程序包的 ` /helm` 目录中包含 Helm 图表。
2. 使用 `helm install` 命令并为您的部署指定一个名称。请参见以下示例：

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

   如果尚未为 Trident 创建命名空间，则可以将 ` -create-namespace` 参数添加到 `helm install` 命令中。然后， Helm 将自动为您创建命名空间。

   在安装期间，可以通过两种方式传递配置数据：

   - ` 值` （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，要更改默认值 `debug` ，请运行以下 ` -set` 命令：

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

Helm 图表中的 `values.yaml` 文件提供了键及其默认值的列表。

`Helm list` 显示有关安装的详细信息，例如名称，命名空间，图表，状态， 应用程序版本，修订版号等。

手动部署 Trident 操作员

执行列出的步骤以手动部署 Trident 操作员。

第 1 步：确定 Kubernetes 集群的资格

首先，您需要登录到 Linux 主机并验证它是否正在管理 _b工作 _ ， "支持的 Kubernetes 集群" 您具有所需权限。

> 使用 OpenShift 时，在以下所有示例中使用 `oc` 而不是 `kubectl` ，并首先以 * 系统： admin* 的身份运行 `oc login -u system ： admin` 或 `oc login -u Kube-admin` 进行登录。

要查看您的 Kubernetes 版本是否高于 1.17 ，请运行以下命令：

```
kubectl version
```

要查看您是否具有 Kubernetes 集群管理员权限，请运行以下命令：

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

要验证是否可以从 Docker Hub 启动使用映像的 POD 并通过 Pod 网络访问存储系统，请运行以下命令：

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

第 2 步：下载并设置操作员

> 从 21.01 开始， Trident 操作符的范围为集群范围。使用 Trident 操作符安装 Trident 需要创建 `TridentOrchestrator` Custom Resource Definition （ CRD ）并定义其他资源。在安装 Astra Trident 之前，您应执行以下步骤来设置操作员。

1. 下载最新版本的 "Trident 安装程序包" 从 _Downloads_部分 中提取该文件。

```
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. 使用适当的 CRD 清单创建 `TridentOrchestrator` CRD 。然后，您可以稍后创建一个 `TridentOrchestrator` Custom Resource ，以通过操作员实例化安装。

   运行以下命令：

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

3. 创建 `TridentOrchestrator` CRD 后，创建操作员部署所需的以下资源：

   - 操作员的 ServiceAccount
   - 对 ServiceAccount 执行 ClusterRole 和 ClusterRoleBinding
   - 专用的 PodSecurityPolicy
   - 运算符本身

     Trident 安装程序包含用于定义这些资源的清单。默认情况下，操作符部署在 `trident` 命名空间中。如果 `trident` 命名空间不存在，请使用以下清单创建一个。

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

4. 要在非默认 `trident` 命名空间中部署运算符，您应更新 `serviceaccount.yaml` ， `clusterrolebinding.yaml` 和 `operator.yaml` 清单并生成您的 `bundle.yaml` 。

   运行以下命令以更新 YAML 清单并使用 `kucstation.yaml` 生成您的 `bundle.yaml` ：

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

   运行以下命令以创建资源并部署操作员：

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

5. 要在部署后验证操作员的状态，请执行以下操作：

```
$ 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
```

操作员部署成功创建了一个在集群中的一个工作节点上运行的 POD 。

> 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3步：创建 `TridentOrchestrator` 并安装Trident

现在，您可以使用操作员安装 Astra Trident 了！这需要创建 `TridentOrchestrator` 。Trident 安装程序附带了用于创建 `TridentOrchestrator` 的示例定义。这将在 `trident` 命名空间中启动安装。

```
$ 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:
  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
```

使用 Trident 操作符，您可以使用 `TridentOrchestrator` 规范中的属性自定义 Astra Trident 的安装方式。请参见 "自定义 Trident 部署"。

状态 `TridentOrchestrator` 指示安装是否成功，并显示已安装的 Trident 版本。

Status	Description
安装	操作员正在使用此 `TridentOrchestrator` CR 安装 Astra Trident 。
已安装	Astra Trident 已成功安装。
正在卸载	操作符正在卸载 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	不使用 `TridentOrchestrator` 。另一个已存在。

在安装期间， `TridentOrchestrator` 的状态会从 `Installing` 更改为 `Installed` 。如果您观察到 `failed` 状态，并且操作员无法自行恢复，则应检查操作员的日志。请参见 "故障排除" 部分。

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

```
$ 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
```

您也可以使用 `tridentctl` 检查已安装的 Astra Trident 版本。

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

现在，您可以继续创建后端。请参见 "部署后任务"。

有关在部署期间排除问题的信息，请参见 "故障排除" 部分。

自定义 Trident 操作员部署

通过 Trident 操作符，您可以使用 `TridentOrchestrator` 规范中的属性自定义 Astra Trident 的安装方式。

有关属性列表，请参见下表：

参数	Description	Default
命名空间	用于安装 Astra Trident 的命名空间	default
`debug`	为 Astra Trident 启用调试	false
`IPv6`	安装基于 IPv6 的 Astra Trident	false
`k8sTimeout`	Kubernetes 操作超时	30 秒
`sileAutoSupport`	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
`autosupport` 映像	AutoSupport 遥测的容器映像	"NetApp/trident autosupport：21.04.0"
`autosupport` 代理	用于发送 AutoSupport 遥测的代理的地址 / 端口	"http://proxy.example.com:8888"
卸载	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，格式为 ` < 注册表 FQDN>[： 端口 ]/subpath]`	"K8s.gcr.io/SIG-storage （K8s 1.17+）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	
`imagePullSecs`	从内部注册表中提取映像的机密信息	

> 在 `TridentOrchestrator` 中指定 `spec.namespace`，以表示安装了哪个命名空间 Astra Trident。此参数 * 安装 Astra Trident 后无法更新 *。尝试执行此操作会导致 `TridentOrchestrator` 的状态更改为 `Failed`。Astra Trident 不适用于跨命名空间迁移。

> 自动员工节点准备是一项 * 测试版功能 *，仅用于非生产环境。

在定义 `TridentOrchestrator` 时，您可以使用上述属性自定义安装。以下是一个示例：

```
$ 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
```

如果您希望自定义安装超出 `TridentOrchestrator` 参数的允许范围，则应考虑使用 `tridentctl` 生成自定义 YAML 清单，以便您可以根据需要进行修改。

使用 tridentctl 进行部署

您可以使用 `tridentctl` 来部署 Astra Trident 。

> 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。

> 要自定义部署，请参见 "此处"。

您需要的内容

要部署 Astra Trident ，应满足以下前提条件：

- 您对受支持的 Kubernetes 集群具有完全权限。
- 您可以访问受支持的 NetApp 存储系统。
- 您可以从所有 Kubernetes 工作节点挂载卷。
- 您安装了一个安装了 `kubectl` （如果使用的是 OpenShift ，则为 `oc` ）的 Linux 主机，并将其配置为管理要使用的 Kubernetes 集群。
- 您已将 `KUBECONFIG` 环境变量设置为指向 Kubernetes 集群配置。
- 您已启用 "Astra Trident 所需的功能门"。
- 如果您将 Kubernetes 与 Docker Enterprise 结合使用， "按照其步骤启用 CLI 访问"。

明白了吗？太棒了！让我们开始吧。

> 有关自定义部署的信息，请参见 "此处"。

第 1 步：确定 Kubernetes 集群的资格

首先，您需要登录到 Linux 主机并验证它是否正在管理 _b工作 _ ， "支持的 Kubernetes 集群" 您具有所需权限。

> 使用 OpenShift 时，您可以在以下所有示例中使用 `oc` 而不是 `kubectl` ，并且应首先以 * 系统：admin* 的身份登录，方法是运行 `oc login -u system ： admin` 或 `oc login -u Kube-admin` 。

要检查 Kubernetes 版本，请运行以下命令：

```
kubectl version
```

要查看您是否具有 Kubernetes 集群管理员权限，请运行以下命令：

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

要验证是否可以从 Docker Hub 启动使用映像的 POD 并通过 Pod 网络访问存储系统，请运行以下命令：

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

确定 Kubernetes 服务器版本。您将在安装 Astra Trident 时使用它。

第 2 步：下载并提取安装程序

> Trident 安装程序将创建 Trident Pod ，配置用于保持其状态的 CRD 对象，并初始化执行配置卷和将卷附加到集群主机等操作的 CSI sidecars 。

您可以下载最新版本的 "Trident 安装程序包" 从 _Downloads_部分 中提取该文件。

例如，如果最新版本为 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
```

第 3 步：安装 Astra Trident

执行 `tridentctl install` 命令，在所需命名空间中安装 Astra Trident 。

```
$ ./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.
....
```

安装程序完成后，其外观将会如此。根据 Kubernetes 集群中的节点数，您可能会发现更多 Pod：

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

如果您看到与上述示例类似的输出，则表示您已完成此步骤，但尚未完全配置 Astra Trident 。请继续执行下一步。请参见 "部署后任务"。

但是，如果安装程序未成功完成或您未看到 * 正在运行 * `trident CSI -` ，则表示未安装此平台。

有关在部署期间排除问题的信息，请参见 "故障排除" 部分。

自定义 tridentctl 部署

使用 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 ` -trident 映像` 指定映像名称。如果您已将 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 `

-image-regRegistry` 开关指定该存储库的位置，该开关的格式为 ` < 注册表 FQDN>[： port]` 。

要让 Astra Trident 自动为您配置工作节点，请使用 ` -enable-node-prep` 。有关其工作原理的详细信息，请参见 "此处"。

> 自动员工节点准备是一项 * 测试版功能 * ，仅用于非生产环境。

如果您使用的是 Kubernetes 的分发版本，其中 `kubelet` 将其数据保存在通常的 ` /var/lib/kubelet` 以外的路径上，则可以使用 ` -kubelet-dir` 指定备用路径。

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` 参数可在安装程序的 `setup` 目录中创建以下 YAML 文件：

- `trident — clusterrolebinding 。 yaml`
- `trident` 部署。 `yaml`
- `trident — crds.yaml`
- `trident` 集群角色。 `yaml`
- `trident — demonset.yaml`
- `trident service.yaml`
- `trident` 命名空间 `.yaml`
- `trident` 服务帐户。 `yaml`

生成这些文件后，您可以根据需要对其进行修改，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？

部署 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。要执行此操作，请创建包含必要参数的 `backend.json` 文件。可以在 `sample-input` 目录中找到不同后端类型的示例配置文件。

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安装程序随附的 `sample-input/storage-class-csi 。 yaml.tempdl` 文件，将 `*backend_type _*` 替换为存储驱动程序名称。

```
./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` 在 Kubernetes 中创建该对象。

```
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 。

有关示例，请参见 `sample-input/vpva-basic CSI 。 yaml` 。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 ` /usr/share/nginx/html` 下。

```
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。"

版权信息

版权所有 © 2024 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

本软件由 NetApp 按“原样”提供，不含任何明示或暗示担保，包括但不限于适销性以及针对特定用途的适用性的隐含担保，特此声明不承担任何责任。在任何情况下，对于因使用本软件而以任何方式造成的任何直接性、间接性、偶然性、特殊性、惩罚性或后果性损失（包括但不限于购买替代商品或服务；使用、数据或利润方面的损失；或者业务中断），无论原因如何以及基于何种责任理论，无论出于合同、严格责任或侵权行为（包括疏忽或其他行为），NetApp 均不承担责任，即使已被告知存在上述损失的可能性。

NetApp 保留在不另行通知的情况下随时对本文档所述的任何产品进行更改的权利。除非 NetApp 以书面形式明确同意，否则 NetApp 不承担因使用本文档所述产品而产生的任何责任或义务。使用或购买本产品不表示获得 NetApp 的任何专利权、商标权或任何其他知识产权许可。

本手册中描述的产品可能受一项或多项美国专利、外国专利或正在申请的专利的保护。

有限权利说明：政府使用、复制或公开本文档受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中“技术数据权利 — 非商用”条款第 (b)(3) 条规定的限制条件的约束。

本文档中所含数据与商业产品和/或商业服务（定义见 FAR 2.101）相关，属于 NetApp, Inc. 的专有信息。根据本协议提供的所有 NetApp 技术数据和计算机软件具有商业性质，并完全由私人出资开发。 美国政府对这些数据的使用权具有非排他性、全球性、受限且不可撤销的许可，该许可既不可转让，也不可再许可，但仅限在与交付数据所依据的美国政府合同有关且受合同支持的情况下使用。除本文档规定的情形外，未经 NetApp, Inc. 事先书面批准，不得使用、披露、复制、修改、操作或显示这些数据。美国政府对国防部的授权仅限于 DFARS 的第 252.227-7015(b)（2014 年 2 月）条款中明确的权利。

商标信息

NetApp、NetApp 标识和 http://www.netapp.com/TM 上所列的商标是 NetApp, Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。