执行卷操作

Astra Trident

NetApp
November 20, 2023

This PDF was generated from https://docs.netapp.com/zh-cn/trident-2107/trident-use/csi-topology.html on November 20, 2023. Always check docs.netapp.com for the latest.

目录

执行卷操作 1
使用 CSI 拓扑 1
使用快照 8
展开卷 12
导入卷 19

执行卷操作

了解 Astra Trident 为管理卷提供的功能。

- "使用 CSI 拓扑"
- "使用快照"
- "展开卷"
- "导入卷"

使用 CSI 拓扑

Astra Trident 可以通过使用有选择地创建卷并将其附加到 Kubernetes 集群中的节点 "CSI 拓扑功能"。使用 CSI 拓扑功能，可以根据区域和可用性区域将对卷的访问限制为一小部分节点。如今，借助云提供商， Kubernetes 管理员可以生成基于分区的节点。节点可以位于可用性区域内的不同区域，也可以位于不同可用性区域之间。为了便于在多区域架构中为工作负载配置卷， Astra Trident 使用了 CSI 拓扑。

> 了解有关 CSI 拓扑功能的更多信息 "此处"。

Kubernetes 提供了两种唯一的卷绑定模式：

- 使用 `VolumeBindingMode` 设置为 `Immediate`、Astra Trident将创建卷、而不会感知任何拓扑。创建 PVC 时会处理卷绑定和动态配置。这是默认值 `VolumeBindingMode` 和适用于不强制实施拓扑限制的集群。创建永久性卷时，不会依赖于请求的 Pod 的计划要求。
- 使用 `VolumeBindingMode` 设置为 `WaitForFirstConsumer`、在计划和创建使用PVC的Pod之前、将延迟为PVC创建和绑定永久性卷。这样，卷就会根据拓扑要求强制实施的计划限制来创建。

> 。 `WaitForFirstConsumer` 绑定模式不需要拓扑标签。此功能可独立于 CSI 拓扑功能使用。

您需要的内容

要使用 CSI 拓扑，您需要满足以下条件：

- 运行 1.17 或更高版本的 Kubernetes 集群。

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:50:19Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedeafd99df",
GitTreeState:"clean", BuildDate:"2020-10-14T12:41:49Z",
GoVersion:"go1.15.2", Compiler:"gc", Platform:"linux/amd64"}
```

- 集群中的节点应具有可引入拓扑感知的标签 (`topology.kubernetes.io/region` 和 `topology.kubernetes.io/zone`）。在安装 Astra Trident 之前，集群中的节点上应存在这些标签 * ，以使 Astra Trident 能够识别拓扑。

```
$ kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[node1,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node1","kubernetes.io/
os":"linux","node-
role.kubernetes.io/master":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-a"}]
[node2,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node2","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-b"}]
[node3,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kube
rnetes.io/arch":"amd64","kubernetes.io/hostname":"node3","kubernetes.io/
os":"linux","node-
role.kubernetes.io/worker":"","topology.kubernetes.io/region":"us-
east1","topology.kubernetes.io/zone":"us-east1-c"}]
```

第 1 步：创建可感知拓扑的后端

可以设计 Astra Trident 存储后端，以便根据可用性区域有选择地配置卷。每个后端都可以具有一个可选的 `supportedTopologies` 表示必须支持的分区和区域列表的块。对于使用此后端的 StorageClasses ，只有在受支持区域 / 区域中计划的应用程序请求时，才会创建卷。

下面是后端定义示例：

```
{
 "version": 1,
 "storageDriverName": "ontap-san",
 "backendName": "san-backend-us-east1",
 "managementLIF": "192.168.27.5",
 "svm": "iscsi_svm",
 "username": "admin",
 "password": "xxxxxxxxxxxx",
 "supportedTopologies": [
{"topology.kubernetes.io/region": "us-east1",
"topology.kubernetes.io/zone": "us-east1-a"},
{"topology.kubernetes.io/region": "us-east1",
"topology.kubernetes.io/zone": "us-east1-b"}
]
}
```

`supportedTopologies` 用于提供每个后端的区域和分区列表。这些区域和分区表示可在 StorageClass 中提供的允许值列表。对于包含后端提供的部分区域和分区的 StorageClasses，Astra Trident 将在后端创建卷。

您可以定义 `supportedTopologies` 也是每个存储池的一个。请参见以下示例：

```
{"version": 1,
"storageDriverName": "ontap-nas",
"backendName": "nas-backend-us-central1",
"managementLIF": "172.16.238.5",
"svm": "nfs_svm",
"username": "admin",
"password": "Netapp123",
"supportedTopologies": [
      {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"},
      {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
    ]
"storage": [
   {
       "labels": {"workload":"production"},
        "region": "Iowa-DC",
        "zone": "Iowa-DC-A",
        "supportedTopologies": [
            {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-a"}
        ]
    },
    {
        "labels": {"workload":"dev"},
         "region": "Iowa-DC",
         "zone": "Iowa-DC-B",
         "supportedTopologies": [
             {"topology.kubernetes.io/region": "us-central1",
"topology.kubernetes.io/zone": "us-central1-b"}
         ]
     }
]
}
```

在此示例中、将显示 `region` 和 `zone` 标签表示存储池的位置。 `topology.kubernetes.io/region` 和 `topology.kubernetes.io/zone` 指定存储池的使用位置。

第 2 步：定义可识别拓扑的 StorageClasses

根据为集群中的节点提供的拓扑标签，可以将 StorageClasses 定义为包含拓扑信息。这将确定用作 PVC 请求候选对象的存储池，以及可使用 Trident 配置的卷的节点子集。

请参见以下示例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
- matchLabelExpressions:
- key: topology.kubernetes.io/zone
  values:
  - us-east1-a
  - us-east1-b
- key: topology.kubernetes.io/region
  values:
  - us-east1
parameters:
  fsType: "ext4"
```

在上述StorageClass定义中、 `volumeBindingMode` 设置为 `WaitForFirstConsumer`。在此存储类中请求的PVC 在 Pod 中引用之前不会执行操作。和、 `allowedTopologies` 提供要使用的分区和区域。。 `netapp-san-us-east1` StorageClass将在上创建PVC `san-backend-us-east1` 上述定义的后端。

第 3 步：创建和使用 PVC

创建 StorageClass 并将其映射到后端后，您现在可以创建 PVC 。

请参见示例 `spec` 以下：

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: netapp-san-us-east1
```

使用此清单创建 PVC 将导致以下结果：

```
$ kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-san created
$ kubectl get pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
AGE
pvc-san   Pending                                      netapp-san-us-east1
2s
$ kubectl describe pvc
Name:          pvc-san
Namespace:     default
StorageClass:  netapp-san-us-east1
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Mounted By:    <none>
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  WaitForFirstConsumer  6s    persistentvolume-controller  waiting
for first consumer to be created before binding
```

要使 Trident 创建卷并将其绑定到 PVC ，请在 Pod 中使用 PVC 。请参见以下示例：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-pod-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/region
            operator: In
            values:
            - us-east1
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-east1-a
            - us-east1-b
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: vol1
    persistentVolumeClaim:
      claimName: pvc-san
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: vol1
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```

此podSpec指示Kubernetes在中的节点上计划Pod `us-east1` 区域、然后从中的任何节点中进行选择 `us-east1-a` 或 `us-east1-b` 分区。

请参见以下输出：

```
$ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE
NOMINATED NODE   READINESS GATES
app-pod-1   1/1     Running   0          19s   192.168.25.131   node2
<none>           <none>
$ kubectl get pvc -o wide
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS          AGE   VOLUMEMODE
pvc-san   Bound    pvc-ecb1e1a0-840c-463b-8b65-b3d033e2e62b   300Mi
RWO            netapp-san-us-east1   48s   Filesystem
```

更新后端以包括 supportedTopologies

可以更新已有后端以包括列表 `supportedTopologies` 使用 `tridentctl backend update`。这不会影响已配置的卷，并且仅用于后续的 PVC 。

了解更多信息

- "管理容器的资源"
- "节点选择器"
- "关联性和反关联性"
- "损害和公差"

使用快照

从 Astra Trident 的 20.01 版开始，您可以在 Kubernetes 层创建 PV 快照。您可以使用这些快照维护由 Astra Trident 创建的卷的时间点副本，并计划创建其他卷（克隆）。支持卷快照 `ontap-nas`， `ontap-san`， `ontap-san-economy`， `solidfire-san`， `aws-cvs`， `gcp-cvs`，和 `azure-netapp-files` 驱动程序。

> 此功能可从 Kubernetes 1.17 （测试版）获得，从 1.20 开始正式上市。要了解从测试版迁移到 GA 所涉及的变化，请参见 "发布博客"。随着毕业到GA `v1` API版本已推出、可向后兼容 `v1beta1` 快照。

您需要的内容

- 要创建卷快照，需要创建外部快照控制器以及一些自定义资源定义（ Custom Resource Definitions ， CRD ）。这是所使用的 Kubernetes 流程编排程序（例如： Kubeadm ， GKEE ， OpenShift ）的职责。

您可以按如下所示创建外部快照控制器和快照 CRD ：

1. 创建卷快照 CRD ：

```
$ cat snapshot-setup.sh
#!/bin/bash
# Create volume snapshot CRDs
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
3.0/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. 在所需命名空间中创建 snapshot-controller 。编辑以下 YAML 清单以修改命名空间。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-3.0/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-3.0/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

> CSI Snapshotter 提供了 "正在验证 webhook" 帮助用户验证现有 v1bea1 快照并确认它们是有效的资源对象。正在验证的 webhook 会自动标记无效的快照对象，并防止将来创建无效对象。验证 webhook 由 Kubernetes 流程编排程序部署。请参见有关手动部署验证 webhook 的说明 "此处"。查找无效快照清单的示例 "此处"。

下面详细介绍的示例说明了使用快照所需的构造，并说明了如何创建和使用快照。

第1步：设置 `VolumeSnapshotClass`

在创建卷快照之前、请设置一个链接：./trident -referation/objects.html[`VolumeSnapshotClass`^。

```
$ cat snap-sc.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.17 -
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-snapclass
driver: csi.trident.netapp.io
deletionPolicy: Delete
```

。 `driver` 指向Astra Trident的CSI驱动程序。 `deletionPolicy` 可以是 `Delete` 或 `Retain`。设置为时 `Retain`、存储集群上的底层物理快照会保留、即使在使用时也是如此 `VolumeSnapshot` 对象已删除。

第 2 步：创建现有 PVC 的快照

```
$ cat snap.yaml
#Use apiVersion v1 for Kubernetes 1.20 and above. For Kubernetes 1.17 -
1.19, use apiVersion v1beta1.
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvc1-snap
spec:
  volumeSnapshotClassName: csi-snapclass
  source:
    persistentVolumeClaimName: pvc1
```

正在为名为的PVC创建快照 `pvc1`、快照的名称设置为 `pvc1-snap`。

```
$ kubectl create -f snap.yaml
volumesnapshot.snapshot.storage.k8s.io/pvc1-snap created

$ kubectl get volumesnapshots
NAME                   AGE
pvc1-snap              50s
```

这就创建了 `VolumeSnapshot` 对象。VolumeSnapshot类似于PVC、并与关联 `VolumeSnapshotContent` 表示实际快照的对象。

可以标识 `VolumeSnapshotContent` 的对象 `pvc1-snap` VolumeSnapshot的说明。

```
$ kubectl describe volumesnapshots pvc1-snap
Name:         pvc1-snap
Namespace:    default
.
.
.
Spec:
  Snapshot Class Name:    pvc1-snap
  Snapshot Content Name:  snapcontent-e8d8a0ca-9826-11e9-9807-525400f3f660
  Source:
    API Group:
    Kind:       PersistentVolumeClaim
    Name:       pvc1
Status:
  Creation Time:  2019-06-26T15:27:29Z
  Ready To Use:   true
  Restore Size:   3Gi
.
.
```

。 `Snapshot Content Name` 标识提供此快照的VolumeSnapshotContent对象。。 `Ready To Use` 参数表示可使用Snapshot创建新的PVC。

第 3 步：从 VolumeSnapshots 创建 PVC

有关使用快照创建 PVC 的示例，请参见以下示例：

```
$ cat pvc-from-snap.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: golden
  resources:
    requests:
      storage: 3Gi
  dataSource:
    name: pvc1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

`dataSource` 显示必须使用名为的VolumeSnapshot创建PVC `pvc1-snap` 作为数据源。此操作将指示 Astra Trident 从快照创建 PVC 。创建 PVC 后，可以将其附加到 Pod 上，并像使用任何其他 PVC 一样使用。

> 删除具有关联快照的永久性卷时，相应的 Trident 卷将更新为 " 正在删除 " 状态。要删除 Astra Trident 卷，应删除该卷的快照。

了解更多信息

- "卷快照"
- 链接：./trident参考/objects.html[`VolumeSnapshotClass`^

展开卷

通过 Astra Trident ， Kubernetes 用户可以在创建卷后对其进行扩展。查找有关扩展 iSCSI 和 NFS 卷所需配置的信息。

展开 iSCSI 卷

您可以使用 CSI 配置程序扩展 iSCSI 永久性卷（ PV ）。

> 支持iSCSI卷扩展 `ontap-san`， `ontap-san-economy`， `solidfire-san` 驱动程序并需要Kubernetes 1.16及更高版本。

概述

扩展 iSCSI PV 包括以下步骤：

- 编辑StorageClass定义以设置 `allowVolumeExpansion` 字段设置为 `true`。
- 编辑PVC定义并更新 `spec.resources.requests.storage` 以反映新需要的大小、该大小必须大于原始大小。
- 要调整 PV 大小，必须将 PV 连接到 Pod 。调整 iSCSI PV 大小时，有两种情况：
  - 如果 PV 连接到 Pod ，则 Astra Trident 会扩展存储后端的卷，重新扫描设备并调整文件系统大小。
  - 尝试调整未连接 PV 的大小时， Astra Trident 会扩展存储后端的卷。将 PVC 绑定到 Pod 后， Trident 会重新扫描设备并调整文件系统大小。然后， Kubernetes 会在扩展操作成功完成后更新 PVC 大小。

以下示例显示了扩展 iSCSI PV 的工作原理。

第 **1** 步：配置 **StorageClass** 以支持卷扩展

```
$ cat storageclass-ontapsan.yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-san
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
allowVolumeExpansion: True
```

对于已存在的StorageClass、请对其进行编辑以包括 `allowVolumeExpansion` 参数。

第 **2** 步：使用您创建的 **StorageClass** 创建 **PVC**

```
$ cat pvc-ontapsan.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ontap-san
```

Astra Trident 会创建一个永久性卷（ PV ）并将其与此永久性卷声明（ PVC ）关联。

```
$ kubectl get pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi
RWO            ontap-san      8s

$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   1Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               10s
```

第 3 步：定义连接 PVC 的 POD

在此示例中、创建了一个使用的POD `san-pvc`。

```
$  kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
centos-pod   1/1     Running   0          65s

$  kubectl describe pvc san-pvc
Name:          san-pvc
Namespace:     default
StorageClass:  ontap-san
Status:        Bound
Volume:        pvc-8a814d62-bd58-4253-b0d1-82f2885db671
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner:
csi.trident.netapp.io
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Mounted By:    centos-pod
```

第 4 步：展开 PV

要将已创建的PV从1Gi调整为2Gi、请编辑PVC定义并更新 `spec.resources.requests.storage` 至2Gi。

```
$ kubectl edit pvc san-pvc
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: "2019-10-10T17:32:29Z"
  finalizers:
  - kubernetes.io/pvc-protection
  name: san-pvc
  namespace: default
  resourceVersion: "16609"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/san-pvc
  uid: 8a814d62-bd58-4253-b0d1-82f2885db671
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
 ...
```

第 **5** 步：验证扩展

您可以通过检查 PVC ， PV 和 Astra Trident 卷的大小来验证扩展是否正常运行：

```
$ kubectl get pvc san-pvc
NAME      STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
san-pvc   Bound    pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi
RWO            ontap-san      11m
$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
pvc-8a814d62-bd58-4253-b0d1-82f2885db671   2Gi        RWO
Delete           Bound    default/san-pvc   ontap-san               12m
$ tridentctl get volumes -n trident
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san     |
block    | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

展开 NFS 卷

Astra Trident支持对上配置的NFS PV进行卷扩展 `ontap-nas`， `ontap-nas-economy`， `ontap-nas-flexgroup`， `aws-cvs`， `gcp-cvs`，和 `azure-netapp-files` 后端。

第 1 步：配置 StorageClass 以支持卷扩展

要调整NFS PV的大小、管理员首先需要通过设置来配置存储类以允许卷扩展 `allowVolumeExpansion` 字段设置为 `true`：

```
$ cat storageclass-ontapnas.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontapnas
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
allowVolumeExpansion: true
```

如果您已创建没有此选项的存储类、则只需使用编辑现有存储类即可 `kubectl edit storageclass` 以允许卷扩展。

第 **2** 步：使用您创建的 **StorageClass** 创建 **PVC**

```
$ cat pvc-ontapnas.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ontapnas20mb
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Mi
  storageClassName: ontapnas
```

Astra Trident 应为此 PVC 创建一个 20 MiB NFS PV：

```
$ kubectl get pvc
NAME           STATUS   VOLUME
CAPACITY     ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi
RWO            ontapnas        9s

$ kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   20Mi       RWO
Delete           Bound    default/ontapnas20mb   ontapnas
2m42s
```

第 **3** 步：展开 **PV**

要将新创建的20MiB PV调整为1GiB、请编辑PVC并进行设置 `spec.resources.requests.storage` 到1 GB：

```
$ kubectl edit pvc ontapnas20mb
# Please edit the object below. Lines beginning with a '#' will be
ignored,
# and an empty file will abort the edit. If an error occurs while saving
this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io
  creationTimestamp: 2018-08-21T18:26:44Z
  finalizers:
  - kubernetes.io/pvc-protection
  name: ontapnas20mb
  namespace: default
  resourceVersion: "1958015"
  selfLink: /api/v1/namespaces/default/persistentvolumeclaims/ontapnas20mb
  uid: c1bd7fa5-a56f-11e8-b8d7-fa163e59eaab
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
...
```

第 **4** 步：验证扩展

您可以通过检查 PVC ， PV 和 Astra Trident 卷的大小来验证调整大小是否正常工作：

```
$ kubectl get pvc ontapnas20mb
NAME           STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
ontapnas20mb   Bound    pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi
RWO            ontapnas        4m44s

$ kubectl get pv pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS    REASON
AGE
pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7   1Gi        RWO
Delete           Bound    default/ontapnas20mb   ontapnas
5m35s

$ tridentctl get volume pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 -n
trident
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-08f3d561-b199-11e9-8d9f-5254004dfdb7 | 1.0 GiB | ontapnas      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

导入卷

您可以使用将现有存储卷作为Kubernetes PV导入 `tridentctl import`。

支持卷导入的驱动程序

下表介绍了支持导入卷的驱动程序及其引入的版本。

驱动程序	版本。
`ontap-nas`	19.04
`ontap-nas-flexgroup`	19.04
`solidfire-san`	19.04
`aws-cvs`	19.04

驱动程序	版本。
azure-netapp-files	19.04
gcp-cvs	19.04
ontap-san	19.04

为什么应导入卷?

将卷导入到 Trident 的使用情形有多种：

- 对应用程序进行容器化并重复使用其现有数据集
- 为临时应用程序使用数据集的克隆
- 重建发生故障的 Kubernetes 集群
- 在灾难恢复期间迁移应用程序数据

导入的工作原理是什么?

卷导入过程使用永久性卷声明（ PVC ）文件创建 PVC 。PVC 文件应至少包含 name ， namespace ， accessModes 和 storageClassName 字段，如以下示例所示。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my_claim
  namespace: my_namespace
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: my_storage_class
```

。 `tridentctl` 客户端用于导入现有存储卷。Trident 通过保留卷元数据并创建 PVC 和 PV 来导入卷。

```
$ tridentctl import volume <backendName> <volumeName> -f <path-to-pvc-
file>
```

要导入存储卷，请指定包含该卷的 Astra Trident 后端的名称以及用于唯一标识存储上的卷的名称（例如：ONTAP FlexVol ， Element Volume ， CVS 卷路径）。存储卷必须允许读 / 写访问，并可由指定的 Astra Trident 后端访问。。 `-f` 字符串参数为必填项、用于指定YAML或JSON PVC文件的路径。

当 Astra Trident 收到导入卷请求时，系统会在 PVC 中确定并设置现有卷大小。存储驱动程序导入卷后，系统将创建 PV ，并为其创建一个 Claims Ref 。回收策略最初设置为 `retain` 在PV中。Kubernetes 成功绑定 PVC 和 PV 后，将更新回收策略以匹配存储类的回收策略。存储类的回收策略为时 `delete`、删除PV时、存储卷将被删

除。

使用导入卷时 `--no-manage` 参数中、Trident不会在对象的生命周期内对PVC或PV执行任何其他操作。因为Trident会忽略的PV和PVC事件 `--no-manage` 对象、删除PV时不会删除存储卷。卷克隆和卷大小调整等其他操作也会被忽略。如果要对容器化工作负载使用 Kubernetes ，但希望在 Kubernetes 外部管理存储卷的生命周期，则此选项非常有用。

PVC 和 PV 中会添加一个标注，用于指示卷已导入以及 PVC 和 PV 是否已管理。不应修改或删除此标注。

Trident 19.07 及更高版本可处理 PV 的连接，并在导入卷时挂载该卷。对于使用早期版本的 Astra Trident 进行的导入，数据路径中不会执行任何操作，卷导入将不会验证是否可以挂载卷。如果卷导入出错(例如、StorageClass不正确)、您可以通过将PV上的回收策略更改为来恢复 `retain`、删除PVC和PV、然后重试volume import命令。

`ontap-nas` 和 `ontap-nas-flexgroup` 导入

使用创建的每个卷 `ontap-nas` 驱动程序是ONTAP 集群上的FlexVol。使用导入FlexVol `ontap-nas` 驱动程序的工作原理相同。ONTAP 集群上已存在的FlexVol 可以作为导入 `ontap-nas` PVC。同样、FlexGroup vols也可以作为导入 `ontap-nas-flexgroup` PVC。

> 要由 Trident 导入 ONTAP 卷，必须为 rw 类型。如果卷的类型为 DP ，则为 SnapMirror 目标卷；应先中断镜像关系，然后再将卷导入到 Trident 中。

> 。 `ontap-nas` 驱动程序无法导入和管理qtree。。 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序不允许使用重复的卷名称。

例如、导入名为的卷 `managed_volume` 位于名为的后端 `ontap_nas`、请使用以下命令：

```
$ tridentctl import volume ontap_nas managed_volume -f <path-to-pvc-file>

+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-bf5ad463-afbb-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

导入名为的卷 `unmanaged_volume` (在上 `ontap_nas backend`)、而Trident不会管理此项、请使用以下命令：

```
$ tridentctl import volume nas_blog unmanaged_volume -f <path-to-pvc-file>
--no-manage

+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-df07d542-afbc-11e9-8d9f-5254004dfdb7 | 1.0 GiB | standard      |
file     | c5a6f6a4-b052-423b-80d4-8fb491a14a22 | online | false   |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

使用时 `--no-manage` 参数、Trident不会重命名卷或验证卷是否已挂载。如果卷未手动挂载，则卷导入操作将失败。

> 先前存在的使用自定义 UnixPermissions 导入卷的错误已得到修复。您可以在 PVC 定义或后端配置中指定 unixPermissions ，并指示 Astra Trident 相应地导入卷。

`ontap-san` 导入

Astra Trident 还可以导入包含单个 LUN 的 ONTAP SAN FlexVol 。这与一致 `ontap-san` 驱动程序、用于为FlexVol 中的每个PVC和LUN创建FlexVol。您可以使用 `tridentctl import` 命令的方式与其他情况相同：

- 包括的名称 `ontap-san` 后端。
- 提供需要导入的 FlexVol 的名称。请记住，此 FlexVol 仅包含一个必须导入的 LUN 。
- 提供必须与结合使用的PVC定义路径 `-f` 标志。
- 可以选择对 PVC 进行管理，也可以选择不对其进行管理。默认情况下， Trident 将管理 PVC 并重命名后端的 FlexVol 和 LUN 。要作为非受管卷导入、请传递 `--no-manage` 标志。

> 导入非受管时 `ontap-san` 卷中的LUN、您应确保FlexVol 中的LUN名为 `lun0` 和映射到具有所需启动程序的igroup。Astra Trident 会自动为受管导入处理此问题。

然后， Astra Trident 将导入 FlexVol 并将其与 PVC 定义关联。Astra Trident还会将FlexVol 重命名为 `pvc-<uuid>` 将FlexVol 中的LUN格式化为 `lun0`。

> 建议导入没有活动连接的卷。如果要导入当前使用的卷，请先克隆该卷，然后再执行导入。

示例

以导入 `ontap-san-managed` 上存在的FlexVol `ontap_san_default` 后端、运行 `tridentctl import` 命令为：

```
$ tridentctl import volume ontapsan_san_default ontap-san-managed -f pvc-
basic-import.yaml -n trident -d

+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a | 20 MiB | basic         |
block    | cd394786-ddd5-4470-adc3-10c5ce4ca757 | online | true    |
+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
```

ONTAP 卷的类型必须为 RW ，才能由 Astra Trident 导入。如果卷的类型为 DP ，则为 SnapMirror 目标卷；在将卷导入到 Astra Trident 之前，应中断镜像关系。

element 导入

您可以使用 Trident 将 NetApp Element 软件 /NetApp HCI 卷导入到 Kubernetes 集群中。您需要提供Astra Trident后端的名称、以及作为参数的卷和PVC文件的唯一名称 `tridentctl import` 命令：

```
$ tridentctl import volume element_default element-managed -f pvc-basic-
import.yaml -n trident -d

+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-970ce1ca-2096-4ecd-8545-ac7edc24a8fe | 10 GiB | basic-element |
block    | d3ba047a-ea0b-43f9-9c42-e38e58301c49 | online | true    |
+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
```

Element 驱动程序支持重复的卷名称。如果卷名称重复，则 Trident 的卷导入过程将返回错误。作为临时解决策，克隆卷并提供唯一的卷名称。然后导入克隆的卷。

aws-cvs 导入

要在 AWS 中导入由 NetApp Cloud Volumes Service 支持的卷，请按卷路径而非名称来标识该卷。

导入 aws-cvs 后端上的卷称为 awscvs_YEppr 卷路径 adroit-jolly-swift、请使用以下命令：

```
$ tridentctl import volume awscvs_YEppr adroit-jolly-swift -f <path-to-
pvc-file> -n trident

+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE  | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-a46ccab7-44aa-4433-94b1-e47fc8c0fa55 | 93 GiB | aws-storage   | file
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true    |
+------------------------------------------+--------+---------------
+----------+--------------------------------------+--------+---------+
```

> 卷路径是卷导出路径中： / 之后的部分。例如、如果导出路径为 10.0.0.1:/adroit-jolly-swift、卷路径为 adroit-jolly-swift。

gcp-cvs 导入

导入 gcp-cvs 卷的工作方式与导入相同 aws-cvs 卷。

azure-netapp-files 导入

导入 azure-netapp-files 后端上的卷称为 azurenetappfiles_40517 卷路径 `importvol1`下，运行以下命令：

```
$ tridentctl import volume azurenetappfiles_40517 importvol1 -f <path-to-
pvc-file> -n trident

+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID             | STATE  | MANAGED |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
| pvc-0ee95d60-fd5c-448d-b505-b72901b3a4ab | 100 GiB | anf-storage   |
file     | 1c01274f-d94b-44a3-98a3-04c953c9a51e | online | true    |
+------------------------------------------+---------+---------------
+----------+--------------------------------------+--------+---------+
```

ANF 卷的卷路径位于： / 之后的挂载路径中。例如、如果挂载路径为 `10.0.0.2:/importvol1`、卷路径为 `importvol1`。

版权信息

版权所有 © 2023 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。