配置和管理卷
Trident

NetApp
January 14, 2026

目录

配置和管理卷

配置卷

创建一个使用已配置的Kubernetes StorageClass来请求PV访问权限的永久性卷请求(PVC)。然后、您可以将PV挂载到POD。

概述

A "*PersigentVolumeClaim*" (PVC)是指请求访问集群上的永久卷。

可以将PVC配置为请求特定大小的存储或访问模式。通过使用关联的StorageClass，集群管理员可以控制不限于持续卷大小和访问模式(例如性能或服务级别)。

创建PVC后、您可以将卷挂载到Pod中。

创建**PVC**

步骤

1. 创建 PVC。

```
kubectl create -f pvc.yaml
```

2. 验证PVC状态。

```
kubectl get pvc
```

```
NAME          STATUS   VOLUME      CAPACITY    ACCESS MODES   STORAGECLASS AGE
pvc-storage   Bound    pv-name     1Gi         RWO                    5m
```

1. 将卷挂载到Pod中。

```
kubectl create -f pv-pod.yaml
```

> 您可以使用监控进度 `kubectl get pod --watch`。

2. 验证卷是否已挂载在上 `/my/mount/path`。

```
kubectl exec -it task-pv-pod -- df -h /my/mount/path
```

3. 现在、您可以删除Pod。Pod应用程序将不再存在、但卷将保留。

```
kubectl delete pod pv-pod
```

示例清单

PersentVolumeClaim示例清单

这些示例显示了基本的PVC配置选项。

PVC、带读取器

此示例显示了一个具有读取权限的基本PVC，该PVC与名为的StorageClass关联 `basic-csi`。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: basic-csi
```

采用**NVMe/TCP**的**PVC**

此示例显示了与名为的StorageClass关联的具有读取权限的NVMe/TCP的基本PVC `protection-gold`。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: pvc-san-nvme
spec:
accessModes:
  - ReadWriteOnce
resources:
  requests:
    storage: 300Mi
storageClassName: protection-gold
```

POD清单示例

这些示例显示了将PVC连接到POD的基本配置。

基本配置

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
       claimName: pvc-storage
  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: storage
```

基本**NVMe/TCP**配置

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: basic-pvc
      persistentVolumeClaim:
        claimName: pvc-san-nvme
  containers:
    - name: task-pv-container
      image: nginx
      volumeMounts:
        - mountPath: "/my/mount/path"
          name: basic-pvc
```

有关存储类如何与和参数交互以控制Trident如何配置卷的详细信息 `PersistentVolumeClaim`、请参见"Kubernetes 和 Trident 对象"。

展开卷

Trident使Kubnetes用户能够在创建卷后对其进行扩展。查找有关扩展iSCSI、NFS和FC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 ）。

> iSCSI卷扩展受、、 `ontap-san-economy` `solidfire-san`驱动程序支持 `ontap-san`，并且需要Kubernetes 1.16及更高版本。

第 1 步：配置 StorageClass 以支持卷扩展

编辑StorageClass定义以将字段设置 `allowVolumeExpansion`为 `true`。

```
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
```

对于已有的StorageClass、请对其进行编辑以包含 `allowVolumeExpansion`参数。

第 2 步：使用您创建的 StorageClass 创建 PVC

编辑PVC定义并更新以反映新需要的 `spec.resources.requests.storage`大小、该大小必须大于原始大小。

```
cat pvc-ontapsan.yaml
```

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

Trident会创建一个永久性卷(PV)并将其与此永久性卷请求(PVC)相关联。

```
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**

将PV连接到POD以调整大小。调整 iSCSI PV 大小时，有两种情况：

- 如果PV连接到Pod、则Trident会扩展存储后端的卷、重新扫描设备并重新设置文件系统大小。
- 尝试调整未连接PV的大小时、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。

在此示例中，创建了一个使用的POD `san-pvc`。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-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:    ubuntu-pod
```

第4步：展开PV

要将已创建的PV从1Gi调整为2Gi、请编辑PVC定义并将更新 `spec.resources.requests.storage`为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Trident卷的大小来验证扩展是否正常：

```
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 FC 卷

您可以使用CSI配置程序扩展FC持久卷(PV)。

驱动程序支持FC卷扩展 `ontap-san`、并且需要Kubernetes 1.16及更高版本。

第 1 步：配置 StorageClass 以支持卷扩展

编辑StorageClass定义以将字段设置 `allowVolumeExpansion`为 `true`。

```
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
```

对于已有的StorageClass、请对其进行编辑以包含 `allowVolumeExpansion`参数。

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

编辑PVC定义并更新以反映新需要的 `spec.resources.requests.storage`大小、该大小必须大于原始大小。

```
cat pvc-ontapsan.yaml
```

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

Trident会创建一个永久性卷(PV)并将其与此永久性卷请求(PVC)相关联。

```
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**

将PV连接到POD以调整大小。调整FC PV大小有两种情形：

- 如果PV连接到Pod、则Trident会扩展存储后端的卷、重新扫描设备并重新设置文件系统大小。
- 尝试调整未连接PV的大小时、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。

在此示例中，创建了一个使用的POD `san-pvc`。

```
 kubectl get pod
NAME         READY   STATUS    RESTARTS   AGE
ubuntu-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:    ubuntu-pod
```

第4步：展开PV

要将已创建的PV从1Gi调整为2Gi、请编辑PVC定义并将更新 `spec.resources.requests.storage`为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Trident卷的大小来验证扩展是否正常：

```
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

Trident支持对、 ontap-nas-economy、 ontap-nas-flexgroup、 gcp-cvs`和 `azure-netapp-files`后端配置的NFS PV进行卷扩展 `ontap-nas。

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将字段设置为来将存储类配置为 true`允许卷扩展 `allowVolumeExpansion：

```
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
```

如果您已在不使用此选项的情况下创建了存储类、则只需使用编辑现有存储类即可 `kubectl edit storageclass` 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
```

Trident应为此PVC创建一个20MiB 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并设置 `spec.resources.requests.storage`为1GiB：

```
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Trident卷的大小来验证调整大小是否正常：

```
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导入为Kubbernetes PV `tridentctl import`。

概述和注意事项

您可以将卷导入到Trident中、以便：

- 将应用程序容器化并重复使用其现有数据集
- 对一个应用程序使用数据集的克隆
- 重建发生故障的Kubrenetes集群
- 在灾难恢复期间迁移应用程序数据

注意事项

导入卷之前、请查看以下注意事项。

- Trident只能导入RW (读写)类型的ONTAP卷。DP (数据保护)类型的卷是SnapMirror目标卷。在将卷导入Trident之前、应先中断镜像关系。

- 我们建议导入没有活动连接的卷。要导入当前使用的卷、请克隆此卷、然后执行导入。

这对于块卷尤其重要、因为Kubnetes不会意识到先前的连接、并且可以轻松地将活动卷连接到Pod。这可能会导致数据损坏。

- 虽然 `StorageClass`必须在PVC上指定、但Trident在导入期间不使用此参数。创建卷期间会使用存储类根据存储特征从可用池中进行选择。由于卷已存在、因此导入期间不需要选择池。因此、即使卷位于与PVC中指定的存储类不匹配的后端或池中、导入也不会失败。
- 现有卷大小在PVC中确定和设置。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 。
  - 在PV中、回收策略最初设置为 `retain`。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。
  - 如果存储类的回收策略为 `delete`，则删除PV时将删除存储卷。
- 默认情况下、Trident管理PVC并在后端重命名FlexVol volume和LUN。您可以传递此 `--no-manage`标志以导入非受管卷。如果使用 `--no-manage`，则在对象的生命周期内，Trident不会对PVC或PV执行任何附加操作。删除PV后、不会删除存储卷、并且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。

导入卷

您可以使用 `tridentctl import`导入卷。

步骤

1. 创建用于创建PVC的永久性卷请求(PVC)文件(例如 `pvc.yaml`)。PVC文件应包括 `name`、、 `namespace` `accessModes`和 `storageClassName`。您也可以在PVC定义中指定 `unixPermissions`。

   以下是最低规格示例：

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

   请勿包含PV名称或卷大小等其他参数。这可能发生原因会使导入命令失败。

2. 使用 `tridentctl import`命令指定包含卷的Trident后端的名称以及在存储上唯一标识卷的名称(例如：ONTAP FlexVol、Element卷、Cloud Volumes Service路径)。 `-f`指定PVC文件的路径需要参数。

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

示例

查看以下卷导入示例、了解受支持的驱动程序。

ONTAP NAS和ONTAP NAS FlexGroup

Trident支持使用和 `ontap-nas-flexgroup`驱动程序导入卷 `ontap-nas`。

- `ontap-nas-economy`驱动程序无法导入和管理qtrees。
- `ontap-nas`和 `ontap-nas-flexgroup`驱动程序不允许卷名称重复。

使用驱动程序创建的每个卷 `ontap-nas`都是ONTAP集群上的一个FlexVol volume。使用驱动程序导入FlexVol卷的 `ontap-nas`工作原理相同。可以将ONTAP集群上已存在的FlexVol卷作为PVC导入 `ontap-nas`。同样、FlexGroup vols也可以作为PVC导入 `ontap-nas-flexgroup`。

ONTAP NAS示例

以下是受管卷和非受管卷导入的示例。

受管卷

以下示例将在名为的后端导入名为的 `ontap_nas`卷 `managed_volume`：

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

非受管卷

使用参数时 `--no-manage`、Trident不会重命名卷。

以下示例将在 `ontap_nas`后端导入 `unmanaged_volume`：

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

ONTAP SAN

Trident支持使用和 `ontap-san-economy`驱动程序导入卷 `ontap-san`。

Trident可以导入包含单个LUN的ONTAP SAN FlexVol卷。这与驱动程序一致 `ontap-san`、该驱动程序会为FlexVol volume中的每个PVC和LUN创建一个FlexVol volume。Trident导入FlexVol volume并将其与PVC定义关联。

ONTAP SAN示例

以下是受管卷和非受管卷导入的示例。

受管卷

对于受管卷，Trident会将FlexVol volume重命名为格式，并将FlexVol volume中的LUN重命名 `pvc-<uuid>`为 `lun0`。

以下示例将导入 `ontap-san-managed`后端上的`FlexVol volume `ontap_san_default`：

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

非受管卷

以下示例将在 `ontap_san`后端导入 `unmanaged_example_volume`：

```
tridentctl import volume -n trident san_blog unmanaged_example_volume
-f pvc-import.yaml --no-manage
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
|                   NAME                  |  SIZE   | STORAGE CLASS |
PROTOCOL |             BACKEND UUID              | STATE  | MANAGED |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
| pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228 | 1.0 GiB | san-blog      |
block    | e3275890-7d80-4af6-90cc-c7a0759f555a | online | false   |
+-----------------------------------------+---------+---------------
+----------+-----------------------------------------+--------+---------+
```

如果您将LUN映射到与Kubornetes节点IQN共享IQN的igroups (如以下示例所示)，则会收到错误消息： `LUN already mapped to initiator(s) in this group`。您需要删除启动程序或取消映射LUN才能导入卷。

```
Vserver   Igroup       Protocol OS Type  Initiators
--------- ------------ -------- -------- ------------------------------------
svm0      k8s-nodename.example.com-fe5d36f2-cded-4f38-9eb0-c7719fc2f9f3
                       iscsi    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
svm0      unmanaged-example-igroup
                       mixed    linux    iqn.1994-05.com.redhat:4c2e1cf35e0
```

Element

Trident支持使用驱动程序导入NetApp Element软件和NetApp HCI卷 `solidfire-san`。

> Element 驱动程序支持重复的卷名称。但是、如果存在重复的卷名称、Trident将返回错误。作为临时解决策、克隆卷、提供唯一的卷名称并导入克隆的卷。

元素示例

以下示例将在后端导入 `element-managed`卷 `element_default`。

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

Google Cloud Platform

Trident支持使用驱动程序导入卷 `gcp-cvs`。

> 要在Google云平台中导入NetApp Cloud Volumes Service支持的卷、请按卷路径确定该卷。卷路径是卷的导出路径中在之后的部分 `:/`。例如，如果导出路径为 `10.0.0.1:/adroit-jolly-swift`，则卷路径为 `adroit-jolly-swift`。

Google Cloud Platform示例

以下示例将在后端导入 `gcp-cvs`卷路径为的 `adroit-jolly-swift`卷 `gcpcvs_YEppr`。

```
tridentctl import volume gcpcvs_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 | gcp-storage   | file
| e1a6e65b-299e-4568-ad05-4f0a105c888f | online | true    |
+----------------------------------------+--------+--------------
+----------+----------------------------------------+--------+---------+
```

Azure NetApp Files

Trident支持使用驱动程序导入卷 `azure-netapp-files`。

> 要导入Azure NetApp Files卷、请按卷路径确定该卷。卷路径是卷的导出路径中在之后的部分 `:/`。例如，如果挂载路径为 `10.0.0.2:/importvol1`，则卷路径为 `importvol1`。

Azure NetApp Files示例

以下示例将在后端导入 `azure-netapp-files`卷路径为的 `importvol1`卷 `azurenetappfiles_40517`。

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

Google Cloud NetApp卷

Trident支持使用驱动程序导入卷 `google-cloud-netapp-volumes`。

Google Cloud NetApp卷示例

以下示例将使用卷在后端 backend-tbc-gcnv1`导入 `google-cloud-netapp-volumes`卷 `testvoleasiaeast1。

```
tridentctl import volume backend-tbc-gcnv1 "testvoleasiaeast1" -f < path-
to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE   | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

以下示例将在两个卷位于同一区域时导入 `google-cloud-netapp-volumes`卷：

```
tridentctl import volume backend-tbc-gcnv1
"projects/123456789100/locations/asia-east1-a/volumes/testvoleasiaeast1"
-f <path-to-pvc> -n trident

+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
|                   NAME                   |  SIZE   | STORAGE CLASS
| PROTOCOL |             BACKEND UUID             | STATE   | MANAGED |
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
| pvc-a69cda19-218c-4ca9-a941-aea05dd13dc0 |  10 GiB | gcnv-nfs-sc-
identity | file     | 8c18cdf1-0770-4bc0-bcc5-c6295fe6d837 | online | true
|
+------------------------------------------+---------
+----------------------+----------+--------------------------------------
+--------+---------+
```

自定义卷名称和标签

使用Trident、您可以为创建的卷分配有意义的名称和标签。这有助于您识别卷并轻松地将其映射到其各自的Kubnetes资源(PVC)。您还可以在后端级别定义用于创建自定义卷名称和自定义标签的模板；您创建、导入或克隆的任何卷都将遵循这些模板。

开始之前

可自定义的卷名称和标签支持：

1. 卷创建、导入和克隆操作。
2. 如果使用的是ONTA-NAS经济型驱动程序、则只有qtree卷的名称符合此名称模板。
3. 如果使用的是ONONTAP SAN经济版驱动程序、则只有LUN名称符合此名称模板。

限制

1. 可自定义的卷名称仅与ONTAP内部部署驱动程序兼容。
2. 可自定义的卷名称不适用于现有卷。

可自定义卷名称的关键行为

1. 如果因名称模板中的语法无效而导致失败、则后端创建将失败。但是、如果模板应用程序失败、则会根据现有命名约定对卷进行命名。
2. 如果使用后端配置中的名称模板来命名卷、则存储前缀不适用。任何所需的前缀值都可以直接添加到模板中。

具有名称模板和标签的后端配置示例

可以在根和/或池级别定义自定义名称模板。

根级别示例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "defaults": {
    "nameTemplate":
"{{.volume.Name}}_{{.labels.cluster}}_{{.volume.Namespace}}_{{.volume.Requ
estName}}"
  },
  "labels": {
    "cluster": "ClusterA",
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"
  }
}
```

池级别示例

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "backendName": "ontap-nfs-backend",
  "managementLIF": "<ip address>",
  "svm": "svm0",
  "username": "<admin>",
  "password": "<password>",
  "useREST": true,
  "storage": [
    {
      "labels": {
        "labelname": "label1",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool01_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    },
    {
      "labels": {
        "cluster": "label2",
        "name": "{{ .volume.Name }}"
      },
      "defaults": {
        "nameTemplate": "pool02_{{ .volume.Name }}_{{ .labels.cluster
}}_{{ .volume.Namespace }}_{{ .volume.RequestName }}"
      }
    }
  ]
}
```

命名模板示例

示例**1**：

```
"nameTemplate": "{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
.config.BackendName }}"
```

示例**2**：

```
"nameTemplate": "pool_{{ .config.StoragePrefix }}_{{ .volume.Name }}_{{
slice .volume.RequestName 1 5 }}""
```

需要考虑的要点

1. 对于卷导入、只有当现有卷具有特定格式的标签时、才会更新标签。例如：`{"provisioning":{"Cluster":"ClusterA", "PVC": "pvcname"}}`。
2. 对于受管卷导入、卷名称遵循后端定义中根级别定义的名称模板。
3. Trident不支持使用带有存储前缀的分区操作符。
4. 如果这些模板不会生成唯一的卷名称、则Trident将附加一些随机字符来创建唯一的卷名称。
5. 如果NAS经济型卷的自定义名称长度超过64个字符、则Trident将根据现有命名约定为卷命名。对于所有其他ONTAP驱动程序、如果卷名称超过名称限制、则卷创建过程将失败。

在命名空间之间共享NFS卷

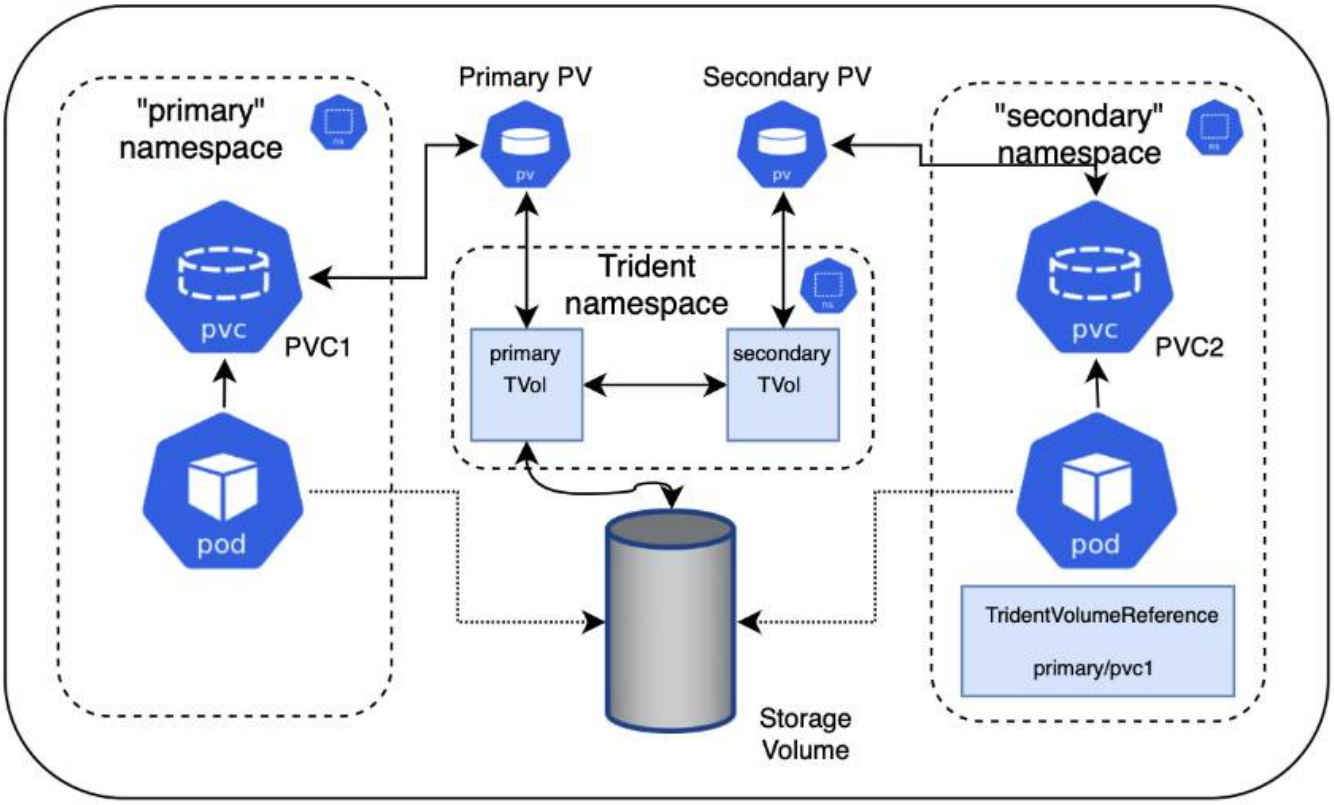
使用Trident、您可以在主命名空间中创建卷、并在一个或多个二级命名空间中共享该卷。

功能

通过TridentvolumeReference CR、您可以在一个或多个Kubernetes名空间之间安全地共享ReadwriteMany (rwx) NFS卷。此Kubernetes本机解决方案 具有以下优势：

- 可通过多个级别的访问控制来确保安全性
- 适用于所有Trident NFS卷驱动程序
- 不依赖于tridentctl或任何其他非本机Kubernetes功能

此图显示了两个Kubernetes命名空间之间的NFS卷共享。



快速入门

只需几个步骤即可设置NFS卷共享。

1 配置源**PVC**以共享卷

源命名空间所有者授予访问源PVC中数据的权限。

2 授予在目标命名空间中创建**CR**的权限

集群管理员向目标命名空间的所有者授予创建TridentVolumeReference CR的权限。

3 在目标命名空间中创建**Trident**卷 引用

目标命名空间的所有者将创建TridentVolumeReference CR以引用源PVC。

4 在目标命名空间中创建从属**PVC**

目标命名空间的所有者创建从属PVC以使用源PVC中的数据源。

配置源和目标命名空间

为了确保安全性、跨命名空间共享需要源命名空间所有者、集群管理员和目标命名空间所有者的协作和操作。每个步骤都会指定用户角色。

步骤

1. **Source命名空间owner:**(`pvc1`在源命名空间中创建PVC，该PVC用于授予与目标命名空间共享的权限(`namespace2)。 shareToNamespace

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/shareToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident将创建PV及其后端NFS存储卷。

- 您可以使用逗号分隔列表将PVC共享给多个命名空间。例如，`trident.netapp.io/shareToNamespace: namespace2,namespace3,namespace4`。
- 您可以使用共享到所有`*`的文件。例如、`trident.netapp.io/shareToNamespace: *`
- 您可以随时更新PVC以包含`shareToNamespace`标注。

2. *集群管理员：*创建自定义角色并执行kubeconfig、以授予目标命名空间所有者在目标命名空间中创建TridentVolumeReference CR的权限。
3. *目标命名空间所有者:*在目标命名空间中创建引用源命名空间的trident卷 引用CR `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. **Destination命名空间owner:**(`pvc2`在目标命名空间中创建PVC(`namespace2)使用`shareFromPVC`标注指定源PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/shareFromPVC: namespace1/pvc1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

目标PVC的大小必须小于或等于源PVC。

结果

Trident会读取 `shareFromPVC`目标PVC上的标注、并将目标PV创建为其自身没有指向源PV的存储资源的子卷、同时共享源PV存储资源。目标PVC和PV显示为正常绑定。

删除共享卷

您可以删除跨多个命名空间共享的卷。Trident将删除对源命名空间上的卷的访问权限、并保留共享该卷的其他命名空间的访问权限。删除引用卷的所有名称空间后、Trident将删除该卷。

`tridentctl get`用于查询子卷

您可以使用[`tridentctl`实用程序运行 `get`命令以获取子卷。有关详细信息、请参阅链接：`Trident tridentctl.html[`tridentctl` commands and options]。

```
Usage:
  tridentctl get [option]
```

flags

- `-h, --help`：卷帮助。
- `--parentOfSubordinate string`：将查询限制为从属源卷。
- `--subordinateOf string`：将查询限制为卷的子卷。

限制

- Trident无法阻止目标名称空间写入共享卷。您应使用文件锁定或其他进程来防止覆盖共享卷数据。

- 您不能通过删除或 `shareFromNamespace`标注或删除CR来撤消对源PVC的 `TridentVolumeReference`访问 `shareToNamespace`。要撤消访问、必须删除从属PVC。
- 无法在从属卷上执行快照、克隆和镜像。

了解更多信息

要了解有关跨命名空间卷访问的详细信息、请执行以下操作：

- 请访问。"在命名空间之间共享卷：对跨命名空间卷访问说Hello"
- 观看上的演示 "NetAppTV"。

跨命名空间克隆卷

通过使用Trident、您可以使用同一个Kubornetes集群中不同命名空间的现有卷或卷快照创建新卷。

前提条件

克隆卷之前、请确保源后端和目标后端的类型相同、并且具有相同的存储类。

快速入门

只需几个步骤即可设置卷克隆。

1 配置源**PVC**以克隆卷

源命名空间所有者授予访问源PVC中数据的权限。

2 授予在目标命名空间中创建**CR**的权限

集群管理员向目标命名空间的所有者授予创建TridentVolumeReference CR的权限。

3 在目标命名空间中创建**Trident**卷 引用

目标命名空间的所有者将创建TridentVolumeReference CR以引用源PVC。

4 在目标命名空间中创建克隆**PVC**

目标命名空间的所有者创建PVC以从源命名空间克隆PVC。

配置源和目标命名空间

为确保安全性、跨命名空间克隆卷需要源命名空间所有者、集群管理员和目标命名空间所有者进行协作并采取相应操作。每个步骤都会指定用户角色。

步骤

1. **Source命名空间owner:**(`pvc1`在源命名空间中创建PVC(`namespace1`)，用于(`namespace2`使用标注授予与目标命名空间共享的权限。 `cloneToNamespace`

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1
  namespace: namespace1
  annotations:
    trident.netapp.io/cloneToNamespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

Trident将创建PV及其后端存储卷。

- 您可以使用逗号分隔列表将PVC共享给多个命名空间。例如，`trident.netapp.io/cloneToNamespace: namespace2,namespace3,namespace4`。
- 您可以使用共享到所有`*`的文件。例如、`trident.netapp.io/cloneToNamespace: *`
- 您可以随时更新PVC以包含`cloneToNamespace`标注。

2. *Cluster admin:*创建自定义角色和kubeconfig,以向目标命名空间所有者授予在目标命名空间中创建Trident卷 参考CR的权限(`namespace2`)。
3. *目标命名空间所有者:*在目标命名空间中创建引用源命名空间的trident卷 引用CR `pvc1`。

```
apiVersion: trident.netapp.io/v1
kind: TridentVolumeReference
metadata:
  name: my-first-tvr
  namespace: namespace2
spec:
  pvcName: pvc1
  pvcNamespace: namespace1
```

4. **Destination命名空间owner:**(`pvc2`在目标命名空间中创建PVC(`namespace2`)，使用`cloneFromPVC`或`cloneFromSnapshot`和`cloneFromNamespace`标注指定源PVC。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  annotations:
    trident.netapp.io/cloneFromPVC: pvc1
    trident.netapp.io/cloneFromNamespace: namespace1
  name: pvc2
  namespace: namespace2
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: trident-csi
  resources:
    requests:
      storage: 100Gi
```

限制

- 对于使用PV-NAS经济型驱动程序配置的ONTAP、不支持只读克隆。

使用SnapMirror复制卷

Trident支持在一个集群上的源卷与对等集群上的目标卷之间建立镜像关系、以便为灾难恢复复制数据。您可以使用具有名称流的自定义资源定义(CRD)执行以下操作：

- 在卷之间创建镜像关系(PVC)
- 删除卷之间的镜像关系
- 中断镜像关系
- 在灾难情况下提升二级卷(故障转移)
- 在集群之间执行应用程序无中断过渡(在计划内故障转移或迁移期间)

复制前提条件

开始之前、请确保满足以下前提条件：

ONTAP 集群

- **Kubernet**：使用ONTAP作为后端的源和目标Trident集群上必须存在Trident版本22.10或更高版本。
- 许可证：必须在源和目标ONTAP集群上启用使用数据保护包的ONTAP SnapMirror异步许可证。有关详细信息、请参见 "ONTAP 中的SnapMirror许可概述" 。

对等

- 集群和**SVM**：ONTAP存储后端必须建立对等状态。有关详细信息、请参见 "集群和 SVM 对等概述" 。

确保两个ONTAP集群之间的复制关系中使用的SVM名称是唯一的。

- * Trident和SVM*：对等远程SVM必须可供目标集群上的Trident使用。

支持的驱动程序

- ONTAP -NAS和ONTAP SAN驱动程序支持卷复制。

创建镜像PVC

按照以下步骤并使用CRD示例在主卷和二级卷之间创建镜像关系。

步骤

1. 在主Kubbernetes集群上执行以下步骤：
   a. 使用参数创建StorageClass对象 `trident.netapp.io/replication: true` 。

      示例

      ```
      apiVersion: storage.k8s.io/v1
      kind: StorageClass
      metadata:
        name: csi-nas
      provisioner: csi.trident.netapp.io
      parameters:
        backendType: "ontap-nas"
        fsType: "nfs"
        trident.netapp.io/replication: "true"
      ```

   b. 使用先前创建的StorageClass创建PVC。

      示例

      ```
      kind: PersistentVolumeClaim
      apiVersion: v1
      metadata:
        name: csi-nas
      spec:
        accessModes:
        - ReadWriteMany
        resources:
          requests:
            storage: 1Gi
        storageClassName: csi-nas
      ```

   c. 使用本地信息创建镜像关系CR。

示例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
```

Trident会提取卷的内部信息以及卷的当前数据保护(DP)状态、然后填充镜像关系的状态字段。

d. 获取TridentMirorRelationship CR以获取PVC的内部名称和SVM。

```
kubectl get tmr csi-nas
```

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
  generation: 1
spec:
  state: promoted
  volumeMappings:
  - localPVCName: csi-nas
status:
  conditions:
  - state: promoted
    localVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
    localPVCName: csi-nas
    observedGeneration: 1
```

2. 在二级Kubbernetes集群上执行以下步骤：

a. 使用trident.netapp.io/replication: true参数创建StorageClass。

示例

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-nas
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/replication: true
```

b. 使用目标和源信息创建镜像关系CR。

示例

```
kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"
```

Trident将使用配置的关系策略名称(或ONTAP的默认策略)创建SnapMirror关系并对其进行初始化。

c. 使用先前创建的StorageClass创建一个PVC以用作二级(SnapMirror目标)。

示例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: csi-nas
  annotations:
    trident.netapp.io/mirrorRelationship: csi-nas
spec:
  accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
storageClassName: csi-nas
```

Trident将检查是否存在TridentMirorRelationship CRD、如果此关系不存在、则无法创建卷。如果存在此

关系、Trident将确保将新FlexVol volume放置到与镜像关系中定义的远程SVM建立对等关系的SVM上。

卷复制状态

三级镜像关系(TCR)是一种CRD、表示PVC之间复制关系的一端。目标T关系 管理器具有一个状态、此状态会告知Trident所需的状态。目标T关系 管理器具有以下状态：

- 已建立：本地PVC是镜像关系的目标卷、这是一个新关系。
- 提升：本地PVC可读写并可挂载、当前未建立任何有效的镜像关系。
- 重新建立：本地PVC是镜像关系的目标卷、以前也位于该镜像关系中。
  - 如果目标卷曾经与源卷建立关系、因为它会覆盖目标卷的内容、则必须使用重新建立的状态。
  - 如果卷之前未与源建立关系、则重新建立的状态将失败。

在计划外故障转移期间提升辅助**PVC**

在二级Kubbernetes集群上执行以下步骤：

- 将TridentMirorRelationship的_spec.state_字 段更新到 `promoted`。

在计划内故障转移期间提升辅助**PVC**

在计划内故障转移(迁移)期间、执行以下步骤以提升二级PVC：

步骤

1. 在主Kubbernetes集群上、创建PVC的快照、并等待创建快照。
2. 在主Kubnetes集群上、创建SnapshotInfo CR以获取内部详细信息。

   示例

   ```
   kind: SnapshotInfo
   apiVersion: trident.netapp.io/v1
   metadata:
     name: csi-nas
   spec:
     snapshot-name: csi-nas-snapshot
   ```

3. 在二级Kubernetes集群上、将_TridentMirorRelationship_ CR的_spec.state_字 段更新为_promoted_和_spec.promotedSnapshotHandle_、以成为快照的内部名称。
4. 在二级Kubernetes集群上、确认Trident镜像 关系的状态(stats.state字段)为已提升。

在故障转移后还原镜像关系

在还原镜像关系之前、请选择要用作新主卷的那一端。

步骤

1. 在二级Kubernetes集群上、确保已更新TundentMirorRelationship上的_spic.netVolumeHandle_字段的值。
2. 在二级Kubernetes集群上，将Trident镜像 关系的_spec.mirector_字段更新到 `reestablished`。

其他操作

Trident支持对主卷和二级卷执行以下操作：

将主**PVC**复制到新的二级**PVC**

确保您已有一个主PVC和一个次要PVC。

步骤

1. 从已建立的二级(目标)集群中删除PerbestentVolumeClaim和TridentMirorRelationship CRD。
2. 从主(源)集群中删除TridentMirorRelationship CRD。
3. 在主(源)集群上为要建立的新二级(目标) PVC创建新的TridentMirorRelationship CRD。

调整镜像、主**PVC**或二级**PVC**的大小

可以正常调整PVC的大小、如果数据量超过当前大小、ONTAP将自动扩展任何目标flevxvol。

从**PVC**中删除复制

要删除复制、请对当前二级卷执行以下操作之一：

- 删除次要PVC上的镜像关系。此操作将中断复制关系。
- 或者、将spec.state字段更新为_promoted_。

删除**PVC (**之前已镜像**)**

Trident会检查是否存在复制的PVC、并在尝试删除卷之前释放复制关系。

删除**TMR**

删除镜像关系一端的T磁 还原会导致剩余的T磁 还原在Trident完成删除之前过渡到_promoted 状态。如果选择删除的*TMirror*已处于*_Promved"*状态、则不存在现有镜像关系、此时*TMirror*将被删除、*Trident*会将本地*PVC*提升为*_ReadWrite*。此删除操作将释放ONTAP中本地卷的SnapMirror元数据。如果此卷将来要在镜像关系中使用、则在创建新镜像关系时、它必须使用具有_re设立_卷复制状态的新TMirror。

在**ONTAP**联机时更新镜像关系

建立镜像关系后、可以随时更新这些关系。您可以使用 `state: promoted` 或 `state: reestablished` 字段更新关系。将目标卷提升为常规ReadWrite卷时、可以使用_promotedSnapshotHandle_指定要将当前卷还原到的特定快照。

在**ONTAP**脱机时更新镜像关系

您可以使用CRD执行SnapMirror更新、而无需Trident直接连接到ONTAP集群。请参阅以下TridentAction镜像 更新的示例格式：

示例

```
apiVersion: trident.netapp.io/v1
kind: TridentActionMirrorUpdate
metadata:
  name: update-mirror-b
spec:
  snapshotHandle: "pvc-1234/snapshot-1234"
  tridentMirrorRelationshipName: mirror-b
```

`status.state` 反映TridentAction镜像 更新CRD的状态。它可以从_suced_、_in Progress _或_failed中获取值。

使用 CSI 拓扑

Trident可以通过使用有选择地创建卷并将其连接到Kubbernetes集群中的节点 "CSI 拓扑功能"。

概述

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。节点可以位于一个区域内的不同可用性区域中，也可以位于不同区域之间。为了便于在多区域架构中为工作负载配置卷、Trident使用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：

- 将设置为 `Immediate`时，`Trident`创建卷时 `VolumeBindingMode`不具有任何拓扑感知功能。创建 PVC 时会处理卷绑定和动态配置。这是默认设置 `VolumeBindingMode`、适合不强制实施拓扑限制的集群。创建永久性卷时、不会依赖于发出请求的POD的计划要求。
- 将设置为 `WaitForFirstConsumer`时，为PVC创建和绑定永久性卷的操作将延迟到计划和创建使用PVC的Pod时 `VolumeBindingMode`才进行。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：

- 运行的Kub并 网集群"支持的Kubernetes版本"

```
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Trident之前，这些标签*应出现在群集中的节点上*，以便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

Trident存储后端可以设计为根据可用性区域选择性地配置卷。每个后端都可以包含一个可选 `supportedTopologies` 块、该块代表受支持的分区和区域列表。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。

下面是一个后端定义示例：

YAML

```yaml
---
version: 1
storageDriverName: ontap-san
backendName: san-backend-us-east1
managementLIF: 192.168.27.5
svm: iscsi_svm
username: admin
password: password
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
```

JSON

```json
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "backendName": "san-backend-us-east1",
  "managementLIF": "192.168.27.5",
  "svm": "iscsi_svm",
  "username": "admin",
  "password": "password",
  "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、Trident会在后端创建一个卷。

您也可以定义 `supportedTopologies`每个存储池。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: password
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
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-a
  - labels:
      workload: dev
    supportedTopologies:
      - topology.kubernetes.io/region: us-central1
        topology.kubernetes.io/zone: us-central1-b
```

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

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: null
name: netapp-san-us-east1
provisioner: csi.trident.netapp.io
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions: null
  - 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`将设置为 `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`提供了要使用的分区和区域。StorageClass会 `netapp-san-us-east1`在上述定义的后端创建PVC `san-backend-us-east1`。

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 。

请参见以下示例 `spec`：

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata: null
name: pvc-san
spec: null
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Kubornetes在区域中的节点上计划POD `us-east1`、并从或 `us-east1-b`区域中的任何节点中进行选择 `us-east1-a`。

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
```

更新后端以包含 supportedTopologies

可以更新已有的后端以包括使用 `tridentctl backend update`列表 `supportedTopologies。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

持久卷(PVs)的Kubbernetes卷快照支持卷的时间点副本。您可以为使用Trident创建的卷创建快照、导入在Trident外部创建的快照、从现有快照创建新卷以及从快照恢复卷数据。

概述

卷快照支持 `ontap-nas` ， `ontap-nas-flexgroup` ， `ontap-san` ， `ontap-san-economy` ， `solidfire-san` ， `gcp-cvs` ， `azure-netapp-files` ， 和 `google-cloud-netapp-volumes`司机。

开始之前

要使用快照、您必须具有外部快照控制器和自定义资源定义(CRD)。这是Kubernetes流程编排程序(例如：Kubeadm、GKE、OpenShift)的职责。

如果您的Kubnetes分发不包括快照控制器和CRD，请参阅[部署卷快照控制器]。

> 如果在GKE环境中创建按需卷快照、请勿创建快照控制器。GKE-使用内置的隐藏快照控制器。

创建卷快照

步骤

1. 创建 VolumeSnapshotClass。有关详细信息，请参阅。"VolumeSnapshotClass"
   - `driver`指向Trident CSI驱动程序。
   - deletionPolicy`可以是 `Delete`或 `Retain。如果设置为 Retain，则即使删除对象，存储集群上的底层物理快照也会保留 VolumeSnapshot。

   示例

   ```
   cat snap-sc.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshotClass
   metadata:
     name: csi-snapclass
   driver: csi.trident.netapp.io
   deletionPolicy: Delete
   ```

2. 创建现有PVC的快照。

   示例

   - 此示例将创建现有PVC的快照。

   ```
   cat snap.yaml
   ```

   ```
   apiVersion: snapshot.storage.k8s.io/v1
   kind: VolumeSnapshot
   metadata:
     name: pvc1-snap
   spec:
     volumeSnapshotClassName: csi-snapclass
     source:
       persistentVolumeClaimName: pvc1
   ```

   - 以下示例将为名为的PVC创建卷快照对象，并且快照 pvc1`的名称设置为 `pvc1-snap。卷快照类似于PVC、并与表示实际快照的对象相关联 VolumeSnapshotContent。

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

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

- 您可以通过对卷快照对象进行描述来确定 `VolumeSnapshotContent`该对象 `pvc1-snap`。`Snapshot Content Name`标识提供此快照的卷SnapshotContent对象。`Ready To Use`参数表示快照可用于创建新PVC。

```
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
...
```

从卷快照创建PVC

您可以使用 `dataSource`创建使用名为作为数据源的卷快照的`PVC` `<pvc-name>`。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。

> PVC将与源卷在同一后端创建。请参阅 "知识库文章：无法在备用后端创建从三端PVC Snapshot创建PVC"。

以下示例将使用作为数据源创建PVC `pvc1-snap`。

```
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
```

导入卷快照

Trident支持通过"Kubbernetes预配置快照过程"、集群管理员可以创建 `VolumeSnapshotContent`对象并导入在Trident外部创建的快照。

开始之前

Trident必须已创建或导入快照的父卷。

步骤

1. *集群管理员：*创建 `VolumeSnapshotContent`引用后端快照的对象。这将在Trident中启动快照工作流。
   - 在中将后端快照的名称指定 `annotations`为 `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - 在中指定 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>`。这是调用中 `snapshotHandle`外部快照程序向Trident提供的唯一信息。 `ListSnapshots`

     `<volumeSnapshotContentName>`由于CR命名限制、不能始终与后端快照名称匹配。

     示例

     以下示例将创建一个 `VolumeSnapshotContent`引用后端Snapshot的对象 `snap-01`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotContent
metadata:
  name: import-snap-content
  annotations:
    trident.netapp.io/internalSnapshotName: "snap-01"  # This is the
name of the snapshot on the backend
spec:
  deletionPolicy: Retain
  driver: csi.trident.netapp.io
  source:
    snapshotHandle: pvc-f71223b5-23b9-4235-bbfe-e269ac7b84b0/import-
snap-content # <import PV name or source PV name>/<volume-snapshot-
content-name>
  volumeSnapshotRef:
    name: import-snap
    namespace: default
```

2. *Cluster admin:*创建引用对象的 `VolumeSnapshot`CR `VolumeSnapshotContent`。此操作将请求访问以在给定命名空间中使用 `VolumeSnapshot`。

   示例

   以下示例将创建一个 `VolumeSnapshot`名为的CR，该CR引用名为 `import-snap`的 `VolumeSnapshotContent import-snap-content`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: import-snap
spec:
  # volumeSnapshotClassName: csi-snapclass (not required for pre-
provisioned or imported snapshots)
  source:
    volumeSnapshotContentName: import-snap-content
```

3. *内部处理(无需执行任何操作)：*外部快照程序识别新创建的 `VolumeSnapshotContent`并运行 `ListSnapshots`调用。`Trident`将创建 `TridentSnapshot`。
   - 外部快照程序将设置为，将 `VolumeSnapshot`设置 `VolumeSnapshotContent`为 `readyToUse true`。
   - Trident返回 `readyToUse=true`。
4. *any user:*创建 `PersistentVolumeClaim`引用新的的 `VolumeSnapshot`，其中 `spec.dataSource`(或 `spec.dataSourceRef`)名是 `VolumeSnapshot`名称。

示例

以下示例将创建一个引用名为 `import-snap`的的PVC `VolumeSnapshot。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-from-snap
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: simple-sc
  resources:
    requests:
      storage: 1Gi
  dataSource:
    name: import-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

使用快照恢复卷数据

默认情况下、快照目录处于隐藏状态、以便最大程度地兼容使用和 `ontap-nas-economy`驱动程序配置的卷 `ontap-nas。启用 `.snapshot`目录以直接从快照恢复数据。

使用volume Snapshot restore ONTAP命令行界面将卷还原到先前快照中记录的状态。

```
cluster1::*> volume snapshot restore -vserver vs0 -volume vol3 -snapshot
vol3_snap_archive
```

还原Snapshot副本时、现有卷配置将被覆盖。创建Snapshot副本后对卷数据所做的更改将丢失。

从快照原位还原卷

Trident可使用(TSR) CR从快照快速原位还原卷 `TridentActionSnapshotRestore`。此CR用作要务Kubbernetes操作、在操作完成后不会持久保留。

Trident支持在 `ontap-san`、、 `ontap-san-economy` `ontap-nas`、 `ontap-nas-flexgroup` `azure-netapp-files`、、 `gcp-cvs` `google-cloud-netapp-volumes`和 `solidfire-san`驱动程序。

开始之前

您必须具有绑定的PVC和可用的卷快照。

- 验证PVC状态是否已绑定。

```
kubectl get pvc
```

- 确认卷快照已准备就绪、可以使用。

```
kubectl get vs
```

步骤

1. 创建TSR CR。此示例将为PVC和卷快照创建CR `pvc1` `pvc1-snapshot`。

   TSR CR必须位于PVC和VS所在的命名空间中。

```
cat tasr-pvc1-snapshot.yaml
```

```
apiVersion: trident.netapp.io/v1
kind: TridentActionSnapshotRestore
metadata:
  name: trident-snap
  namespace: trident
spec:
  pvcName: pvc1
  volumeSnapshotName: pvc1-snapshot
```

2. 应用CR以从快照还原。此示例将从Snapshot恢复 `pvc1`。

```
kubectl create -f tasr-pvc1-snapshot.yaml
```

```
tridentactionsnapshotrestore.trident.netapp.io/trident-snap created
```

结果

Trident将从快照还原数据。您可以验证快照还原状态：

```
kubectl get tasr -o yaml
```

```
apiVersion: trident.netapp.io/v1
items:
- apiVersion: trident.netapp.io/v1
  kind: TridentActionSnapshotRestore
  metadata:
    creationTimestamp: "2023-04-14T00:20:33Z"
    generation: 3
    name: trident-snap
    namespace: trident
    resourceVersion: "3453847"
    uid: <uid>
  spec:
    pvcName: pvc1
    volumeSnapshotName: pvc1-snapshot
  status:
    startTime: "2023-04-14T00:20:34Z"
    completionTime: "2023-04-14T00:20:37Z"
    state: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

- 在大多数情况下、如果出现故障、Trident不会自动重试此操作。您需要再次执行此操作。
- 没有管理员访问权限的Kubbernetes用户可能必须获得管理员授予的权限、才能在其应用程序命名空间中创建TSR CR。

删除具有关联快照的PV

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。删除卷快照以删除Trident卷。

部署卷快照控制器

如果您的Kubernetes分发版不包含快照控制器和CRD、则可以按如下所示进行部署。

步骤

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-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yam
l
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-
6.1/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

2. 创建快照控制器。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
csi/external-snapshotter/release-6.1/deploy/kubernetes/snapshot-
controller/setup-snapshot-controller.yaml
```

如有必要、打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml`并更新 `namespace`命名空间。

相关链接

- "卷快照"
- "VolumeSnapshotClass"