配置和管理卷

Astra Trident

NetApp
April 03, 2024

目录

配置和管理卷

配置卷

创建一个使用已配置的Kubernetes StorageClass来请求对PV的访问的永久性卷(PV)和永久性卷克莱姆(PVC)。然后、您可以将PV挂载到POD。

概述

答 "*PersigentVolume*" (PV)是由集群管理员在Kubbernetes集群上配置的物理存储资源。。
"*PersigentVolumeClaim*" (PVC)是对集群上的永久卷的访问请求。

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PV和PVC后、您可以将卷挂载到Pod中。

示例清单

PerfsentVolume示例清单

此示例清单显示了与StorageClass关联的10gi的基本PV `basic-csi`。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
  labels:
    type: local
spec:
  storageClassName: basic-csi
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/my/host/path"
```

PersentVolumeClaim示例清单

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
```

POD清单示例

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

创建**PV**和**PVC**

步骤

1. 创建PV。

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

2. 验证PV状态。

```
kubectl get pv
NAME         CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM
STORAGECLASS  REASON  AGE
pv-storage  4Gi       RWO           Retain          Available
7s
```

3. 创建 PVC 。

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

4. 验证PVC状态。

```
kubectl get pvc
NAME         STATUS VOLUME     CAPACITY ACCESS MODES STORAGECLASS AGE
pvc-storage Bound  pv-name 2Gi      RWO                          5m
```

5. 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`。

6. 验证卷是否已挂载到上 `/my/mount/path`。

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

7. 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 task-pv-pod
```

请参见 "Kubernetes 和 Trident 对象" 有关存储类如何与交互的详细信息 `PersistentVolumeClaim` 和用于控制Asta Trident配置卷的方式的参数。

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。

第 1 步：配置 StorageClass 以支持卷扩展

编辑StorageClass定义以设置 `allowVolumeExpansion` 字段设置为 `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

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。

```
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**

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 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。

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`， `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`。

概述和注意事项

您可以将卷导入到Astra三端到以下位置：

- 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。

- Asta三端磁盘只能导入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Astra三端存储之前、您应先中断镜像关系。

- 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。

- 不过 `StorageClass` 必须在PVC上指定、A作用 是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 。
  - 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。
- 默认情况下、Asta三端存储管理PVC、并在后端重命名FlexVol和LUN。您可以通过 `--no-manage` 用于导入非受管卷的标志。如果您使用 `--no-manage`中，A作用 是在对象的生命周期内不对PVC或PV执行任何其他操作。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` 以导入卷。

步骤

1. 创建永久性卷请求(PVC)文件(例如、 `pvc.yaml`)。PVC文件应包括 `name`， `namespace`， `accessModes` ，和 `storageClassName`。您也可以指定 `unixPermissions` 在PVC定义中。

   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` 命令以指定包含卷的Asta三元数据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

Astra三项功能支持使用导入卷 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序。

- 。 `ontap-nas-economy` 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。

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。

ONTAP NAS示例

以下是受管卷和非受管卷导入的示例。

受管卷

以下示例将导入名为的卷 `managed_volume` 位于名为的后端 `ontap_nas`：

```
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

使用时 `--no-manage` 参数、A作用 是不对卷进行重命名。

以下示例导入 `unmanaged_volume` 在上 `ontap_nas` 后端：

```
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

Astra三项功能支持使用导入卷 `ontap-san` 驱动程序。不支持使用导入卷 `ontap-san-economy` 驱动程序。

Astra三端存储可以导入包含单个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。Asta三进位导入FlexVol并将其与PVC定义关联起来。

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、Asta三端存储将FlexVol重命名为 `pvc-<uuid>` 将FlexVol 中的LUN格式化为 `lun0`。

以下示例将导入 `ontap-san-managed` 上存在的FlexVol `ontap_san_default` 后端：

```
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

以下示例导入 `unmanaged_example_volume` 在上 `ontap_san` 后端：

```
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igroux、如以下示例所示、您将收到错误： `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

Astra三端存储支持使用NetApp Element软件和NetApp HCI卷导入 `solidfire-san` 驱动程序。

> 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、Asta Dent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

以下示例将导入 `element-managed` 后端上的卷 `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 云平台

Astra三项功能支持使用导入卷 `gcp-cvs` 驱动程序。

> 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的一部分、位于之后 `:/`。例如、如果导出路径为 `10.0.0.1:/adroit-jolly-swift`、卷路径为 `adroit-jolly-swift`。

Google Cloud Platform示例

以下示例将导入 `gcp-cvs` 后端上的卷 `gcpcvs_YEppr` 卷路径 `adroit-jolly-swift`。

```
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

Astra三项功能支持使用导入卷 `azure-netapp-files` 驱动程序。

> 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的一部分、位于之后 `:/`。例如、如果挂载路径为 `10.0.0.2:/importvol1`、卷路径为 `importvol1`。

Azure NetApp Files示例

以下示例将导入 `azure-netapp-files` 后端上的卷 `azurenetappfiles_40517` 卷路径 `importvol1`。

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

在命名空间之间共享NFS卷

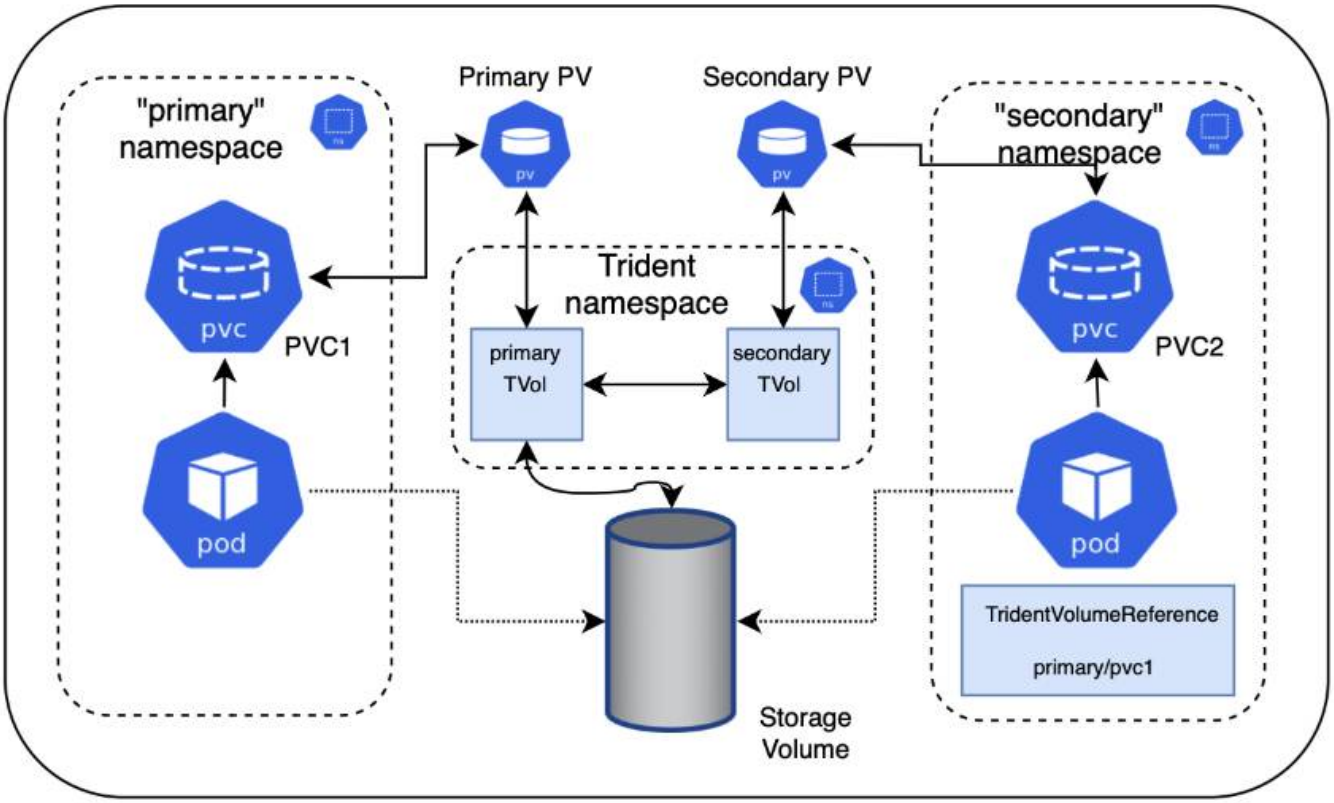
使用Astra Trident、您可以在主命名空间中创建卷、并在一个或多个二级命名空间中共享该卷。

功能

使用Astra TridentVolumeReference CR、您可以在一个或多个Kubernetes命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**TridentVolumeReference**

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. *源命名空间所有者：*创建PVC (`pvc1`)、以授予与目标命名空间共享的权限 (`namespace2`) `shareToNamespace` 标注。

```
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
```

Astra Trident会创建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命名空间 `*`。例如：`trident.netapp.io/shareToNamespace: *`
- 您可以更新PVC以包括 `shareToNamespace` 随时添加标注。

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TridentVolumeReference CR `pvc1`。

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

4. *目标命名空间所有者：*创建PVC (`pvc2`) (`namespace2`) `shareFromPVC` 用于指定源PVC的标注。

```
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

Astra Trident读取 `shareFromPVC` 在目标PVC上添加标注、并将目标PV创建为一个从属卷、而其自身没有指向源PV的存储资源、并共享源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。Astra Trident将删除对源命名空间上卷的访问、并保持对共享该卷的其他命名空间的访问。删除引用卷的所有命名空间后、Astra Trident将删除该卷。

使用 ... `tridentctl get` 查询从属卷

使用[`tridentctl` 实用程序中、您可以运行 `get` 用于获取从属卷的命令。有关详细信息、请参见链接：./trident referation/tridentctl.html[`tridentctl` 命令和选项]。

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

flags

- `-h, --help：卷帮助。
- --parentOfSubordinate string：将查询限制为从源卷。
- --subordinateOf string：将查询限制为卷的下属。

限制

- Astra Trident无法阻止目标命名空间写入共享卷。您应使用文件锁定或其他进程来防止覆盖共享卷数据。
- 您不能通过删除来撤消对源PVC的访问 shareToNamespace 或 shareFromNamespace 标注或删除 TridentVolumeReference CR.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。

有关详细信息 ...

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"。
- 观看演示 "NetAppTV"。

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"。

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， 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、不会依赖于发出请求的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：

- 运行的Kubernetes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 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。

下面是一个后端定义示例：

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

```
{
 "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 ，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: 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
  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：

```
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

持久卷(PVs)的Kubbernetes卷快照支持卷的时间点副本。您可以为使用Asta Trident创建的卷创建快照、导入在Asta 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

支持卷快照 `ontap-nas`， `ontap-nas-flexgroup`， `ontap-san`， `ontap-san-economy`， `solidfire-san`， `gcp-cvs`，和 `azure-netapp-files` 驱动程序。

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Kubernetes分发版不包含快照控制器和CRD、请参见 [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，请参见 "VolumeSnapshotClass"。
   - 。 `driver` 指向A作用 力三端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。

     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
     ```

   - 此示例将为名为的PVC创建卷快照对象 `pvc1` 快照的名称设置为 `pvc1-snap`。VolumeSnapshot类似于PVC、并与关联 `VolumeSnapshotContent` 表示实际快照的对象。

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

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

   - 您可以确定 `VolumeSnapshotContent` 的对象 `pvc1-snap` VolumeSnapshot的说明。。 `Snapshot`

`Content Name` 标识提供此快照的VolumeSnapshotContent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` 使用名为的卷快照创建PVC `<pvc-name>` 作为数据源。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。请参见 "知识库文章：无法在备用后端创建从三端PVC Snapshot创建PVC"。

以下示例将使用创建PVC `pvc1-snap` 作为数据源。

```
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

Asta三项功能支持 "Kubbernetes预配置快照过程" 以使集群管理员能够创建 `VolumeSnapshotContent` 在Asta Trident外部创建的对象和导入快照。

开始之前

Asta三端存储必须已创建或导入快照的父卷。

步骤

1. *集群管理员：*创建一个 `VolumeSnapshotContent` 引用后端快照的对象。这将在Asta Trdent中启动快照工作流。
   - 在中指定后端快照的名称 `annotations` 作为 `trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">`。
   - 指定 `<name-of-parent-volume-in-trident>/<volume-snapshot-content-name>` 在中 `snapshotHandle`。这是外部快照程序在中向Asta Trident提供的唯一信息 `ListSnapshots` 致电。

     。 `<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 `VolumeSnapshotContent` 引用后端快照的对象 `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>
```

2. *集群管理员：*创建 `VolumeSnapshot` 引用的CR `VolumeSnapshotContent` 对象。此操作将请求访问以使用 `VolumeSnapshot` 在给定命名空间中。

   示例

   以下示例将创建 `VolumeSnapshot` CR已命名 `import-snap` 引用的 `VolumeSnapshotContent` 已命名 `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)：*外部快照程序可识别新创建的 `VolumeSnapshotContent` 并运行 `ListSnapshots` 致电。Asta三项功能可创建 `TridentSnapshot`。
   - 外部快照程序用于设置 `VolumeSnapshotContent` to `readyToUse` 和 `VolumeSnapshot` to `true`。
   - TRIdent返回 `readyToUse=true`。
4. *任何用户:*创建一个 `PersistentVolumeClaim` 以引用新的 `VolumeSnapshot`、其中 `spec.dataSource` (或 `spec.dataSourceRef`)名称为 `VolumeSnapshot` 名称。

   示例

   以下示例将创建一个引用的PVC `VolumeSnapshot` 已命名 `import-snap`。

```
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、以便最大程度地提高使用配置的卷的兼容性 `ontap-nas` 和 `ontap-nas-economy` 驱动程序。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。

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Asta Trdent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"

版权信息

商标信息