



配置和管理卷 Trident

NetApp
October 29, 2026

目录

配置和管理卷	1
预配 Trident 卷	1
概述	1
创建 PVC	1
移动卷	4
要求	4
工作原理	4
移动卷	4
扩展 Trident 卷	7
扩展 iSCSI 卷	7
扩展 FC 卷	10
扩展 NFS 卷	13
了解 RWX NVMe 子系统限制	16
了解 64 节点限制	16
了解 NVMe 子系统模型	17
识别错误症状	17
解决子系统限制错误	17
升级 Trident 以应用超级子系统模型	17
控制器可扩展性	18
关键概念和定义	18
默认行为（串行模式）	18
控制器可扩展性支持	18
enableConcurrency 的工作原理	19
启用之前	19
启用控制器可扩展性	19
并发行为	21
注意事项和限制	22
建议	22
自动卷扩展	22
要求	23
限制	23
使用自动增长策略配置卷	23
创建 Autogrow 策略	23
创建 Autogrow 策略	24
策略状态	25
将策略与 StorageClass 关联	25
策略优先级	26
配置示例	26
管理 Autogrow 策略	29

查看 Autogrow 策略	29
更新 Autogrow 策略	29
删除 Autogrow 策略	30
监控 Autogrow 策略使用情况	31
支持的协议	31
支持的后端	31
已知限制	32
常见问题解答	32
导入 Trident 卷	37
概述和注意事项	37
导入卷	38
示例	40
ONTAP SAN-economy 示例	45
自定义 Trident 卷名和标签	48
开始之前	48
限制	48
可定制卷名的关键行为	48
使用名称模板和标签的后端配置示例	48
命名模板示例	50
需要考虑的要点	50
跨命名空间共享 Trident NFS 卷	50
功能	50
快速入门	51
配置源和目标命名空间	52
删除共享卷	53
使用 <code>tridentctl get</code> 查询从属卷	53
限制	54
了解更多信息	54
跨命名空间克隆 Trident 卷	54
前提条件	54
快速入门	54
配置源和目标命名空间	55
限制	56
使用 SnapMirror 复制卷	56
复制先决条件	57
创建镜像 PVC	57
卷复制状态	60
在计划外故障转移期间提升辅助 PVC	60
在计划的故障转移期间推广辅助 PVC	61
故障转移后恢复镜像关系	61
其他操作	61

ONTAP 联机时更新镜像关系	62
ONTAP 离线时更新镜像关系	62
使用 CSI 拓扑	62
概述	63
步骤 1: 创建可感知拓扑的后端	64
步骤 2: 定义拓扑感知的 StorageClasses	66
步骤 3: 创建和使用 PVC	66
更新后端以包括 supportedTopologies	69
查找更多信息	69
使用 Trident 快照	69
概述	69
创建卷快照	69
从卷快照创建 PVC	71
导入卷快照	72
使用快照恢复卷数据	74
从快照就地还原卷	74
删除具有关联快照的 PV	76
部署卷快照控制器	76
相关链接	77
使用 Trident 卷组快照	77
创建卷组快照	77
使用组快照恢复卷数据	79
从快照就地还原卷	79
删除具有关联组快照的 PV	79
部署卷快照控制器	79
相关链接	80

配置和管理卷

预配 Trident 卷

创建一个 PersistentVolumeClaim (PVC)，该 PVC 使用配置的 Kubernetes StorageClass 来请求对 PV 的访问权限。然后，您可以将 PV 挂载到 pod 上。

概述

<https://kubernetes.io/docs/concepts/storage/persistent-volumes>["_PersistentVolumeClaim_"] (PVC) 是访问集群上 PersistentVolume 的请求。

PVC 可以配置为请求特定大小或访问模式的存储。使用关联的 StorageClass，集群管理员可以控制超出 PersistentVolume 大小和访问模式的更多内容——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. 在 Pod 中挂载卷。

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



您可以使用 `kubectl get pod --watch` 监控进度。

2. 验证卷是否已装入 `/my/mount/path`。

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

3. 现在您可以删除 Pod 了。Pod 应用程序将不再存在，但卷将保持不变。

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

示例清单

▼ PersistentVolumeClaim 示例清单

这些示例显示了基本的 PVC 配置选项。

带 RWO 访问的 PVC

此示例显示了一个与名为 `basic-csi` 的 StorageClass 相关联的具有 RWO 访问权限的基本 PVC。

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

带 NVMe/TCP 的 PVC

此示例显示了一个与名为 `protection-gold` 的 StorageClass 相关联的具有 RWO 访问权限的 NVMe/TCP 基本 PVC。

```
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
```

请参阅["Kubernetes 和 Trident 对象"](#)，了解存储类如何与 `PersistentVolumeClaim` 交互，以及用于控制 Trident 配置卷的参数详情。

移动卷

从 Trident 26.06 开始，存储管理员可以无中断地将 iSCSI 卷移动到同一 ONTAP 集群中的不同 ONTAP 节点和聚合，同时使用该卷的工作负载保持运行。您可以使用 `TridentVolumeMove` 自定义资源或 `tridentctl` 等效资源以声明方式请求移动，Trident 将协调 ONTAP 和主机，确保 I/O 不会中断。

要求

卷移动仅适用于 iSCSI 卷的 `ontap-san` 驱动程序。不支持 NVMe 和 NAS 协议。

在移动卷之前，请确保满足以下要求：

- 后端使用 `ontap-san` 驱动程序与 ONTAP REST API。
- ONTAP 集群至少有两个节点。
- 您拥有以下凭据集之一：
 - 如果使用专用自定义 ONTAP 角色，则为 ONTAP 9.19.1 或更高版本。
 - 早期 ONTAP 版本上的集群管理员凭据。

工作原理

Trident 在同一 ONTAP 集群内的两个位置之间移动卷 (LUN)。在数据移动之前，Trident 会在现有路径旁边向目标添加新的 iSCSI 路径，以便主机多路径设备永远不会失去访问权限。ONTAP 完成移动后，Trident 会删除陈旧的源路径。

Trident 通过一组有序状态驱动移动，这些状态在 `TridentVolumeMove` 资源中可见，以便您可以跟踪移动进度。

移动卷

识别卷和位置

1. 查找要移动的 PV 的名称。PV 名称也是移动资源的名称。
2. 在 ONTAP 集群中，记录该卷的当前（源）节点和聚合，并选择目标节点和聚合。

开始移动

使用任一方法。两者都创建相同的 `TridentVolumeMove` 资源，其名称为 PV 名称。

使用 `tridentctl`

```
tridentctl create volume-move \  
  --volume      pvc-8a814d62-bd58-4253-b0d1-82f2885db671 \  
  --source-node ontap-node-01 \  
  --source-pool aggr1_node01 \  
  --target-node ontap-node-02 \  

```

```
--target-pool aggr1_node02 \  
-n trident
```

tvm 是 volume-move 的别名。添加 `--delete-after-success 10m` 以在移动成功后自动删除资源。

使用 **TridentVolumeMove** 资源

```
apiVersion: trident.netapp.io/v1  
kind: TridentVolumeMove  
metadata:  
  name: pvc-8a814d62-bd58-4253-b0d1-82f2885db671    # the PV to move  
  namespace: trident  
spec:  
  sourceNode: ontap-node-01  
  sourcePool: aggr1_node01  
  targetNode: ontap-node-02  
  targetPool: aggr1_node02  
  # deleteAfterSuccess: 10m    # optional
```

应用资源：

```
kubectl apply -f tridentvolumemove.yaml
```

规格字段

sourceNode / targetNode

ONTAP 节点名称（非 UUID）。

sourcePool / targetPool

ONTAP 聚合名称。

deleteAfterSuccess

可选持续时间，例如 `30s` 或 `10m`。设置为 `0` 可立即删除资源，或省略此选项以保留资源。

监控移动

使用 `tvm` 简称或 `tridentvolumemove` 跟踪进度：

```
kubectl get tridentvolumemove -n trident
```

要查看当前状态和每个节点的进度，请描述资源：

```
kubectl describe tridentvolumemove <pv-name> -n trident
```

卷可以在 `Moving` 状态期间具有并发附件，因为 ONTAP 可能需要一段时间才能完成大卷的移动。为了确保无缝移动，Trident 在 Trident 控制器和 Trident 节点之间进行协调。每个状态按顺序执行以下步骤，最终以 `Succeeded` 或 `Failed` 结束。

州/省	发生了什么
Pending	Trident接受并验证移动请求，并在开始任何存储更改之前收集所需的信息。
ControllerStaging	Trident 准备目标位置。它将目标节点的存储接口映射到 LUN 并更新 Kubernetes，以便将来节点重启时重新连接到正确的位置。
NodeStaging	在每个已挂载卷的主机上，Trident 会在现有路径旁边添加来自目标节点的新存储路径，以便 Pod 的 I/O 在两个路径集上继续进行。
Moving	ONTAP 将卷数据复制并转移到目标聚合。这是最长的阶段，ONTAP 全程处理，而 Trident 持续轮询直至完成。在此状态下允许新发布。
ControllerUnstaging	数据移动完成。Trident 更新 Kubernetes 以从所需状态中删除源路径，然后从 LUN 中取消源节点的存储接口的映射。
NodeUnstaging	在每个主机上，Trident 会删除过时的源路径。主机只保留目标路径，多路径设备继续提供 I/O 服务而不会中断。
Succeeded	此卷在目标位置已完全移动并处于联机状态。Trident 会保留 <code>TridentVolumeMove</code> 资源，如果您设置了 <code>deleteAfterSuccess</code> ，则会自动将其删除。
Failed	无法恢复的错误导致移动停止或无法启动。Trident 重置卷状态，以便恢复正常操作，并保留带有错误消息的资源。请查看此消息，修复根本原因，删除此资源，然后重试。

该 `.status.attachments` 列表报告了每个附加节点在其 `state` 字段中的进度：

附加状态	含义
Pending	节点被标识为已连接；工作尚未启动。
Bridged	在现有路径旁边添加新的目标路径。
Migrated	现在通过新路径提供流量。
Cleaned	已从节点中删除过时的源路径。
Detached	节点在移动过程中分离，在 Trident 评估附加操作是否已准备好继续下一步时，该节点被排除在进一步考虑范围之外。
Failed	此节点上出现终端错误。查看消息。

清理移动资源

如果您设置了 `deleteAfterSuccess`，Trident 将自动删除该资源。否则，您可以手动将其删除。在请求对同一卷进行另一次移动之前，请确保该卷不存在处于 `Succeeded` 或 `Failed` 状态的 `TridentVolumeMove` 资源。



Trident不会自动删除失败的移动。Trident保留 `Failed` 资源作为审计记录。修复根本原因后，请手动删除资源，然后重试。

扩展 Trident 卷

Trident 为 Kubernetes 用户提供了在创建卷后扩展卷的功能。查找有关扩展 iSCSI、NFS、SMB、NVMe/TCP 和 FC 卷所需配置的信息。

扩展 iSCSI 卷

您可以使用 CSI 置备程序扩展 iSCSI 永久卷 (PV)。



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

步骤 1：配置 **StorageClass** 以支持卷扩展

编辑 StorageClass 定义以将 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含 allowVolumeExpansion 参数。

步骤 2：使用创建的 **StorageClass** 创建 **PVC**

编辑 PVC 定义并更新 `spec.resources.requests.storage` 以反映新期望的尺寸，该尺寸必须大于原始尺寸。

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

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: san-pvc
spec:
  accessModes:
    - ReadWriteOnce
```

```
resources:
  requests:
    storage: 1Gi
storageClassName: ontap-san
```

Trident 创建持久卷 (PV) 并将其与此持久卷声明 (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
```

步骤 3: 定义一个连接 PVC 的 pod

将 PV 连接到 pod 以调整其大小。调整 iSCSI PV 大小时有两种情况：

- 如果 PV 连接到 pod，Trident 会扩展存储后端上的卷，重新扫描设备，并调整文件系统的大小。
- 在尝试调整未连接 PV 的大小时，Trident 扩展存储后端上的卷。在 PVC 绑定到 pod 之后，Trident 重新扫描设备并调整文件系统的大小。然后，Kubernetes 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 san-pvc 的 Pod。

```
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
```

步骤 4: 扩展 PV

要将已创建的 PV 从 1Gi 调整为 2Gi，请编辑 PVC 定义并将 `spec.resources.requests.storage` 更新为 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
# ...
```

步骤 5: 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工作：

```
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 |
+-----+-----+-----+-----+-----+-----+
|          BACKEND UUID  | STATE | MANAGED  |
+-----+-----+-----+-----+-----+-----+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san |
+-----+-----+-----+-----+-----+-----+
| block | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true  |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
```

扩展 FC 卷

您可以使用 CSI 配置程序扩展 FC 持久卷 (PV)。



ontap-san 驱动程序支持 FC 卷扩展，需要 Kubernetes 1.16 及更高版本。

步骤 1: 配置 **StorageClass** 以支持卷扩展

编辑 **StorageClass** 定义以将 `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含 allowVolumeExpansion 参数。

步骤 2：使用创建的 StorageClass 创建 PVC

编辑 PVC 定义并更新 `spec.resources.requests.storage` 以反映新期望的尺寸，该尺寸必须大于原始尺寸。

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

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

Trident 创建持久卷 (PV) 并将其与此持久卷声明 (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
```

步骤 3：定义一个连接 PVC 的 pod

将 PV 连接到 pod 以调整其大小。调整 FC PV 大小时有两种情况：

- 如果 PV 连接到 pod，Trident 会扩展存储后端上的卷，重新扫描设备，并调整文件系统的大小。
- 在尝试调整未连接 PV 的大小时，Trident 扩展存储后端上的卷。在 PVC 绑定到 pod 之后，Trident 重新扫描设备并调整文件系统的大小。然后，Kubernetes 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 san-pvc 的 Pod。

```
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

```

步骤 4: 扩展 PV

要将已创建的 PV 从 1Gi 调整为 2Gi，请编辑 PVC 定义并将 `spec.resources.requests.storage` 更新为 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
# ...

```

步骤 5: 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工作:

```

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 |
+-----+-----+-----+-----+
|          BACKEND UUID  | STATE | MANAGED  |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| pvc-8a814d62-bd58-4253-b0d1-82f2885db671 | 2.0 GiB | ontap-san    |
| block          | a9b7bfff-0505-4e31-b6c5-59f492e02d33 | online | true      |
+-----+-----+-----+-----+
+-----+-----+-----+-----+

```

扩展 NFS 卷

Trident 支持在 ontap-nas、ontap-nas-economy、`ontap-nas-flexgroup` 和 `azure-netapp-files` 后端上配置的 NFS PV 的卷扩展。

步骤 1: 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 `allowVolumeExpansion` 字段设置为 `true` 来配置存储类以允许卷扩展：

```
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
```

如果您已经创建了一个没有此选项的存储类，则可以通过使用 `kubectl edit storageclass` 来允许卷扩展，从而编辑现有存储类。

步骤 2: 使用创建的 **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 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				

步骤 3: 扩展 PV

要将新创建的 20 MiB PV 调整为 1 GiB, 请编辑 PVC 并设置 `spec.resources.requests.storage` 为 1 GiB:

```
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
# ...
```

步骤 4：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确：

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

了解 RWX NVMe 子系统限制

使用 NVMe 协议的 ReadWriteMany (RWX) 卷的可扩展性限制为每卷 64 个节点。以下包括限制，解释了所涉及的 NVMe 子系统体系结构，并概述了所需的解决步骤。

了解 64 节点限制

如果您计划将 ReadWriteMany (RWX) 卷与 NVMe 协议一起使用，则单个 RWX NVMe 卷不能由 Kubernetes 集群中超过 64 个节点挂载。

请勿计划在 64 个以上节点上挂载相同 RWX NVMe PersistentVolumeClaim 的工作负载。

此限制仅适用于使用 NVMe 协议的 RWX 卷。

了解 NVMe 子系统模型

每卷子系统模型（**Trident** 版本早于 **26.02**）

在 26.02 之前的 Trident 版本中，RWX NVMe 卷使用每个卷的子系统模型进行调配。每个 RWX NVMe 卷都映射到 ONTAP 上其自己的专用 NVMe 子系统。

此模型很简单，但具有较低的可扩展性限制。在大型 Kubernetes 集群中，由于每个 RWX 卷消耗一个专用子系统，因此子系统控制器限制很快就会达到。

超级子系统模型（在 **Trident 26.02** 中引入）

从 Trident 26.02 开始，RWX NVMe 卷使用共享超级子系统模型。多个 RWX NVMe 卷共享同一个 NVMe 子系统。

每个超级子系统最多支持 1024 个命名空间（卷）。该模型显著提高了 RWX 工作负载的可扩展性，并降低了达到 ONTAP 子系统限制的可能性。

每个 RWX NVMe 卷最多支持 64 个节点。

识别错误症状

如果大规模创建或附加 RWX NVMe 卷，您可能会发现类似于以下错误：

```
Maximum number of controllers reached. No more controllers can be created.
```

此错误表示已达到 ONTAP NVMe 子系统控制器限制。

解决子系统限制错误

要超越每卷子系统的限制并利用超级子系统模型，请升级到 Trident 26.02 或更高版本。

升级 **Trident** 以应用超级子系统模型

要为 RWX NVMe 卷应用超级子系统模型：

1. 将 Trident 升级到 26.02 或更高版本。
2. 将使用 RWX NVMe 卷的所有 Pod 缩小到零副本。
3. 确认没有工作负载正在使用 RWX NVMe 卷。
4. 将 pod 重新扩容。

此重新启动顺序可确保使用超级子系统模型连接 RWX NVMe 卷。

- 此限制仅适用于使用 NVMe 协议的 RWX 卷。
- 64 节点限制适用于每个 RWX NVMe 卷。
- 其他访问模式和其他协议不受影响。

控制器可扩展性

Trident 通过提高多个存储驱动程序之间的并发性来提升控制器可扩展性。当您启用控制器可扩展性时，Trident 控制器会并行处理存储操作，而不是对其进行序列化，从而在具有大量并发操作的 Kubernetes 环境中提高吞吐量。

在部署之前，请确定哪些 Trident 驱动程序在正式发布时支持控制器可扩展性，以及哪些驱动程序在 Trident 26.06 中作为技术预览版提供。这有助于您做出明智的部署决策并管理风险。控制器可扩展性默认处于禁用状态。

关键概念和定义

控制器可扩展性

控制器可扩展性是指 Trident 控制器并行处理多个存储操作的能力，而不是在单个锁后对其进行序列化。这些操作包括卷创建、删除和调整大小；快照创建和删除；卷发布和取消发布；以及后端管理。

启用控制器可扩展性时，不同卷和后端上的操作会同时进行。这增加了吞吐量，并减少了具有大量并发 PersistentVolumeClaim 和 VolumeSnapshot 操作的环境中的端到端操作时间。

默认行为（串行模式）

默认情况下，Trident 控制器一次处理一个操作。每个创建、删除、调整大小或快照请求在 Trident 启动下一个之前完成。串行模式是所有安装支持的默认值，无需配置。

串行模式足以满足大多数工作负载的要求。仅当操作量在串行处理下产生积压时，才启用控制器可扩展性。

控制器可扩展性支持

Trident 支持针对不同存储驱动程序的不同成熟度级别的控制器可扩展性。

正式发布

以下驱动程序在 Trident 26.06 正式上市时支持控制器可扩展性：

- `ontap-san`
- `ontap-nas`
- `ontap-nas-economy`
- `ontap-san-economy`
- `google-cloud-netapp-volumes`
- `azure-netapp-files`
- `solidfire-san`



The `google-cloud-netapp-volumes` 和 `google-cloud-netapp-volumes-san` 驱动程序不同。Trident 仅支持 `google-cloud-netapp-volumes`。请勿在后端配置或示例中使用 `google-cloud-netapp-volumes-san`。

技术预览

以下驱动程序支持将控制器可扩展性作为 Trident 26.06 的技术预览功能：

- `asa-r2` (SAN 和 NVMe)

此驱动程序具有以下限制：

- 控制器并发性仅可用于评估和测试。
- 行为可能因版本不同而有所变化。
- NetApp 不建议在生产环境中使用。

enableConcurrency 的工作原理

当您将 `enableConcurrency` 设置为 `true` 时，Trident 会对控制器管理的所有后端应用并发处理。该设置同时应用于每个后端。您不能为单个后端或单个驱动程序启用它。

每个配置的后端都必须使用正式发布或技术预览列表中的驱动程序。如果任何后端使用不受支持的驱动程序，则 Trident 不会启动，并且 Trident 不会添加使用不受支持的驱动程序的后端。

要恢复启动，请删除或重新配置使用不受支持的驱动程序的任何后端，或禁用控制器可扩展性。

启用之前



在启用控制器可扩展性之前，请确认每个已配置的后端均使用正式发布或技术预览列表中的驱动程序。如果任何后端使用不受支持的驱动程序，则在启用该功能后，Trident 将无法启动。

使用下表来决定是否启用控制器可扩展性。

如果您的环境	操作
仅使用受支持的驱动程序，并遇到控制器操作积压	启用控制器可扩展性。
使用任何不受支持的驱动程序	保留默认串行模式。不要启用控制器可扩展性。
处理低操作量且无积压	保留默认串行模式。

启用控制器可扩展性

`enableConcurrency` 配置选项控制控制器的可扩展性。在 Trident 安装期间或更新现有部署时，必须显式启用此选项。

Trident 操作员部署

要使用 Trident 运算符启用控制器可扩展性，请在 `TridentOrchestrator` 自定义资源 (CR) 中将 `enableConcurrency` 设置为 `true`。

新安装

创建或编辑 TridentOrchestrator CR 并将 enableConcurrency 设置为 true:

```
apiVersion: trident.netapp.io/v1
kind: TridentOrchestrator
metadata:
  name: trident
spec:
  namespace: trident
  enableConcurrency: true
```

应用 CR:

```
kubectl apply -f tridentorchestrator_cr.yaml
```

现有安装

修补现有 TridentOrchestrator CR 以启用控制器可扩展性:

```
kubectl patch torc trident --type=merge -p
'{"spec":{"enableConcurrency":true}}'
```

验证 Trident 是否应用了设置:

```
kubectl get torc trident -o
jsonpath='{.status.currentInstallationParams.enableConcurrency}'
```

Helm 部署

要使用 Helm 启用控制器可扩展性, 请将 enableConcurrency 值设置为 true。

新安装

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace --set enableConcurrency=true
```

现有安装

```
helm upgrade trident netapp-trident/trident-operator --namespace trident
--set enableConcurrency=true
```

或者，在自定义 `values.yaml` 文件中将 `enableConcurrency` 设置为 `true`：

```
# values.yaml
enableConcurrency: true
```

然后使用 `values` 文件安装或升级：

```
helm install trident netapp-trident/trident-operator --namespace trident
--create-namespace -f values.yaml
```

tridentctl 部署

要使用 `tridentctl` 启用控制器可扩展性，请在安装过程中传递 `--enable-concurrency` 标志。

新安装

```
tridentctl install -n trident --enable-concurrency
```

现有安装

要在现有 `'tridentctl'` 部署上启用控制器可扩展性，请卸载 Trident 并使用以下标志重新安装：

```
tridentctl uninstall -n trident
tridentctl install -n trident --enable-concurrency
```

验证是否已启用控制器可扩展性

启用控制器可扩展性后，验证 Trident 控制器是否在启用并发的情况下运行。检查控制器 Pod 日志：

```
kubectl logs -n trident deploy/trident-controller | grep -i concurrency
```

输出包含一个确认启用并发的日志条目。

并发行为

启用控制器可扩展性后，Trident 控制器将应用以下行为：

- Trident 将单个全局锁替换为细粒度的按资源锁定。
- Trident 对修改同一资源的操作进行序列化，以保持数据一致性。
- 仅从资源读取的操作与该资源上的其他读取操作同时进行。
- Trident 将并发 ONTAP API 请求限制为每个管理 LIF 20 个，以防止后端存储系统过载。

- 如果多个后端共享相同的管理 LIF，则它们共享此 20 个请求的限制。

注意事项和限制

以下注意事项适用于 Trident 26.06 中的控制器可扩展性：

- 控制器可扩展性仅支持一般可用性和技术预览列表中的驱动程序。有关详细信息，请参见 [启用之前](#)。
- Trident 控制器在内部管理并发。此版本不提供用户可配置的并发限制。
- 整体吞吐量取决于正在使用的存储驱动程序、后端响应能力和 Kubernetes API 服务器性能。
- 高并发可能会增加后端存储系统的负载。
- 控制器的可扩展性行为在所有驱动程序中并不相同。
- 技术预览驱动程序在高负载下可能表现出不一致的性能，并且可能在不同版本之间改变行为。
- 由于并行执行，调试并发操作可能会更加复杂。指标和日志可以显示交错操作输出。

建议

在启用控制器可扩展性之前，请完成以下步骤：

1. 确认每个配置的后端都使用一般可用性或技术预览列表中的驱动程序。
2. 在将更改应用于生产之前，请先在非生产集群中测试更改。
3. 应用更改后，验证是否已启用控制器可扩展性。

使用控制器可扩展性操作时，请应用以下一般建议：

- 对于需要高可扩展性的生产环境，请使用正式发布版驱动程序。
- 仅在非生产环境中评估技术预览驱动程序。
- 大规模运行时监控后端和控制器性能。
- 请勿在自动化脚本中假设操作顺序。

自动卷扩展

自动卷扩展使 Trident 配置的持久卷能够在使用的容量达到定义的阈值时自动增长。此功能可降低运营开销，并有助于防止因容量耗尽而导致的应用程序中断。使用 Autogrow Policies 实现自动卷扩展。Autogrow Policy 定义：

- 触发扩展的利用率阈值
- 卷增长的量
- 卷可以达到的最大大小



当超过定义的利用率阈值时，卷的大小会自动增加。卷永远不会自动缩减。

要求

在配置自动卷扩展之前，请确保满足以下要求：

- Trident 26.02 或更高版本
- 用于创建 `TridentAutogrowPolicy` 自定义资源的基于角色的访问控制权限
- StorageClasses 配置为 `allowVolumeExpansion: true`
- 支持的 ONTAP 协议：
 - 网络文件系统 (NFS)
 - Internet 小型计算机系统接口 (iSCSI)
 - 非易失性内存快速通道 (NVMe)
 - 光纤通道协议 (FCP)

限制

- 早于 ONTAP 9.16.1 的 ONTAP Non-Volatile Memory Express 原始块卷不支持自动扩展。
- 对于存储区域网络卷，如果 `growthAmount`` 小于或等于 50 mebibytes，则 Trident 会在调整大小之前自动将值增加到 51 mebibytes，前提是生成的大小不超过 ``maxSize``。
- 在棕地环境中，由于卷发布迁移行为，某些现有卷可能无法自动扩展。
- 当卷达到 ``maxSize`` 时，不会发生进一步的扩展。
- 支持自动卷扩展的协议：
 - 网络文件系统 (NFS)
 - Internet 小型计算机系统接口 (iSCSI)
 - 非易失性内存快速通道 (NVMe)
 - 光纤通道协议 (FCP)

使用自动增长策略配置卷

可以在两个级别配置自动增长策略：

- 存储类级别：为所有卷设置默认值（使用注释）
- PVC 级别：覆盖存储类默认值（使用注释）

创建 **Autogrow** 策略

当卷达到定义的容量阈值时，Autogrow Policies 启用自动卷扩展。

请确保您拥有：

- 已安装 Trident 26.02 或更高版本
- 基于角色的访问控制权限以创建 `TridentAutogrowPolicy` 资源

- 了解工作负载增长需求

Autogrow Policy 定义了卷在达到定义的容量阈值时如何自动扩展。

您可以在工作流程中随时创建 Autogrow Policies：

- 在创建 StorageClasses 和卷之前
- 在 StorageClasses 存在后
- 配置卷后

这种灵活性使您能够引入自动扩展，而无需重新创建现有资源。

Autogrow 策略规范

Autogrow Policies 是 Kubernetes 自定义资源，定义如下：

字段	说明	格式	必填项	示例	默认
name	唯一策略标识符	字符串	是	production-db-policy	无
usedThresh hold	触发扩展的容量百分比	百分比字 符串	是	"80%"	无
growthAm ount	达到阈值时要增长的金额	百分比或 大小	否	"10%" 或 "5Gi"	"10%"
maxSize	最大卷大小限制	Kubernete s 数量	否	"500Gi"	无限制

创建 Autogrow 策略

步骤

1. 创建定义自动增长策略的 YAML 文件：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

2. 将策略应用于您的集群：

```
kubectl apply -f autogrow-policy.yaml
```

3. 验证策略是否已创建：

```
kubectl get tridentautogrowpolicy standard-autogrow
```

预期输出

NAME	USED THRESHOLD	GROWTH AMOUNT	STATE
standard-autogrow	80%	10%	Success

策略状态

创建策略后，Trident 验证规范并分配以下状态之一：

州/省	说明	需采取行动
成功	策略已验证并可供使用。	无。
失败	检测到验证错误。	查看并修复规范。
正在删除	正在执行删除。	等待完成。

将策略与 **StorageClass** 关联

您可以使用 `trident.netapp.io/autogrowPolicy` 批注将自动增长策略与 StorageClass 关联起来。从该 StorageClass 配置的所有卷都会继承该策略。



StorageClass 必须具有 `allowVolumeExpansion: true`。

步骤

1. 使用 Autogrow Policy 注释创建或修改 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

2. 应用 StorageClass：

```
kubectl apply -f storageclass.yaml
```

3. 验证标注：

```
kubectl get storageclass ontap-gold -o
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

预期输出

```
production-db-policy
```

策略优先级

当在 StorageClass 和 PVC 上设置 Autogrow Policy 注释时，Trident 应用以下优先级规则：

1. *PVC 注释优先。*如果 PVC 设置 `trident.netapp.io/autogrowPolicy`，则始终使用该值。
2. **StorageClass** 注释仅适用于 **PVC** 没有注释的情况。
3. 如果两者都没有注释，则不会应用 Autogrow Policy。

StorageClass 标注	PVC 标注	有效行为
trident.netapp.io/autogrowPolicy: standard-agp	未设置	用途 standard-agp。
trident.netapp.io/autogrowPolicy: standard-agp	trident.netapp.io/autogrowPolicy: logs-policy	使用 logs-policy (PVC 覆盖 StorageClass)。
trident.netapp.io/autogrowPolicy: standard-agp	trident.netapp.io/autogrowPolicy: "none"	无自动增长策略 (PVC 禁用自动增长)。
未设置	trident.netapp.io/autogrowPolicy: dev-policy	用途 dev-policy。
未设置	未设置	无自动增长策略。

配置示例

以下示例显示了不同用例的常见 Autogrow Policy 配置。

生产数据库的保守策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: production-db-policy
spec:
```

```
usedThreshold: "75%"
growthAmount: "20%"
maxSize: "5Ti"
```

具有固定增长增量的日志存储

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: log-storage-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

具有默认值的最小策略

当您忽略 `growthAmount` 和 `maxSize` 时, Trident 使用默认值(`10%` 增长, 无限大小) :

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: logs-policy
spec:
  usedThreshold: "85%"
```

具有自定义 **maxSize** 和默认 **growthAmount** 的策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: default-ga-policy
spec:
  usedThreshold: "70%"
  maxSize: "100Gi"
```

无限制 **maxSize** 的激进增长

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: aggressive-growth-policy
```

```
spec:
  usedThreshold: "80%"
  growthAmount: "150%"
```

具有分数百分比的策略

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: precise-policy
spec:
  usedThreshold: "80.28%"
  growthAmount: "10.65%"
  maxSize: "100Gi"
```



支持分数百分比。如果指定的小数位数超过三位，Trident 会将值舍入到小数点后三位。

NAS StorageClass 使用 Autogrow

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN StorageClass 使用 Autogrow

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
```

```
fsType: "ext4"
allowVolumeExpansion: true
```

管理 Autogrow 策略

创建 Autogrow Policies 后，可以查看、更新和删除它们。您还可以监视使用给定策略的卷。

查看 Autogrow 策略

列出所有策略

使用 `kubectl` 列出集群中的所有 Autogrow Policies：

```
kubectl get tridentautogrowpolicy
```

或者，也可以使用 `tridentctl`：

```
tridentctl get autogrowpolicy
```

查看策略详细信息

要查看策略的完整规范和状态：

```
kubectl describe tridentautogrowpolicy production-db-policy
```

要以 YAML 格式查看策略及其关联的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

更新 Autogrow 策略

您可以修改现有策略以更改其阈值、增长量或最大大小。更改将立即对使用该策略的所有卷生效。



更改会影响使用此策略的所有卷。如果可能，请首先在非生产环境中测试更改。

步骤

1. 编辑策略：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

2. 根据需要修改 `spec` 字段：

```
spec:
  usedThreshold: "75%"      # Changed from 80%
  growthAmount: "20%"       # Changed from 10%
  maxSize: "1Ti"           # Changed from 500Gi
```

3. 保存并退出。更改立即生效。

更新注意事项

- 即时效果：所有使用该策略的卷将在下一次增长评估时采用新参数。
- *无需重新启动卷：*更改适用于下一次增长操作。
- 先测试：尽可能在非生产环境中验证更改。
- *沟通更改：*修改共享策略时通知团队。

删除 Autogrow 策略

Autogrow Policies 使用终结器保护，以防止在卷主动使用时意外删除。

步骤

1. 删除策略：

```
kubectl delete tridentautogrowpolicy production-db-policy
```

2. 如果卷仍在策略，则删除将进入 `Deleting` 状态。检查受影响的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

3. 从每个受影响的卷中删除策略。从下列选项中选择一项：

- 选项 **A**：通过将注释设置为 `"none"` 来显式禁用自动增长：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

- 选项 **B**：完全删除注释：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy-
```

删除行为

场景	行为
没有卷使用该策略	策略将立即删除。
卷使用此策略	策略进入 `Deleting` 状态。终结器阻止完成，直到删除所有卷。
所有卷均已从策略中删除	终结器将被删除，策略也将被删除。

监控 Autogrow 策略使用情况

使用策略检查卷

```
tridentctl get autogrowpolicy production-db-policy -o json | jq '.volumes'
```

查找卷使用的策略

```
kubectl get pvc database-pvc -o  
jsonpath='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

监控策略事件

```
kubectl get events --field-selector  
involvedObject.kind=TridentAutogrowPolicy
```

支持的协议

Autogrow 支持以下存储协议：

- NFS
- iSCSI
- FCP
- NVMe



对于 SAN 卷，如果配置的 `growthAmount` 为 50 MiB 或更少，Trident 会自动将调整大小操作的增长量增加到 51 MB，前提是结果大小不超过 `maxSize`。

支持的后端

Autogrow 支持 ONTAP 后端。Autogrow 还支持 NAS 和 SAN 卷的 Google Cloud NetApp Volumes (GCNV)。

已知限制

- **ONTAP NVMe** 原始块卷：使用早于 9.16.1 的 ONTAP 版本创建的卷不支持自动增长。
- 现有卷（棕地部署）：即使应用了有效的自动增长策略，自动增长可能也不适用于现有卷。此限制是由于卷发布的持续迁移所致。要确认迁移已完成，请检查 Trident 控制器日志中的 `"Migration completed"` 消息。

常见问题解答

Trident 何时评估阈值？

Trident 持续监控卷使用情况。当使用的容量超过 `usedThreshold` 时，Trident 会创建一个内部调整大小请求，并按配置的 `growthAmount` 扩展卷。

例如，此策略将以 80% 的容量触发扩展，并每次将卷增长 10%，最多可达 500 GiB：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: standard-autogrow
spec:
  usedThreshold: "80%"
  growthAmount: "10%"
  maxSize: "500Gi"
```

是否可以在已配置卷后应用策略？

是您可以随时创建 Autogrow Policy，并通过添加或更新 `trident.netapp.io/autogrowPolicy` 注释将其应用于现有 PVC。您无需重新创建 PVC 或 StorageClass。

将策略应用于现有 PVC：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

要将策略应用于现有 StorageClass：

```
kubectl annotate storageclass ontap-gold \
  trident.netapp.io/autogrowPolicy="production-db-policy" \
  --overwrite
```

如果我同时在 **StorageClass** 和 **PVC** 上设置自动增长策略，会发生什么？

PVC 注释始终优先。如果 PVC 具有 `trident.netapp.io/autogrowPolicy` 注释，则无论 StorageClass 指定什么，Trident 都会使用该值。有关详细信息，请参阅 [策略优先级](#)。

例如，假设此 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-agp"
provisioner: csi.trident.netapp.io
allowVolumeExpansion: true
```

此 PVC 覆盖 StorageClass 策略：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-pvc
  annotations:
    trident.netapp.io/autogrowPolicy: "logs-policy"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: ontap-gold
```

Trident 使用 logs-policy 用于 database-pvc，而不是 standard-agp。

如何禁用特定卷的自动增长？

将 PVC 注释设置为 "none"。此设置将覆盖该卷的任何 StorageClass 级别策略：

```
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite
```

您可以验证是否已禁用自动增长：

```
kubectl get pvc <pvc-name> -o jsonpath
='{.metadata.annotations.trident\.netapp\.io/autogrowPolicy}'
```

预期输出

```
none
```

当卷达到 **maxSize** 时会发生什么？

Trident 停止扩展该卷。即使使用量继续增加超过 `usedThreshold`，它也不会为该卷创建进一步的调整大小请求。

例如，使用此策略，Trident 在达到 100 GiB 后停止扩展卷：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: capped-policy
spec:
  usedThreshold: "90%"
  growthAmount: "10Gi"
  maxSize: "100Gi"
```

要允许无限增长，请忽略 `maxSize` 或将其设置为 0：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: unlimited-policy
spec:
  usedThreshold: "85%"
  growthAmount: "10%"
```

我可以在不重新启动卷的情况下更改策略吗？

是更新策略时，使用该策略的所有卷将在下一次增长评估时采用新参数。不需要重新启动卷。

要就地更新策略：

```
kubectl edit tridentautogrowpolicy production-db-policy
```

根据需要修改字段：

```
spec:
  usedThreshold: "75%"      # Changed from 80%
  growthAmount: "20%"      # Changed from 10%
```

```
maxSize: "1Ti" # Changed from 500Gi
```

保存并退出。验证更新后的策略：

```
kubectl get tridentautogrowpolicy production-db-policy
```

预期输出

NAME	USED THRESHOLD	GROWTH AMOUNT	STATE
production-db-policy	75%	20%	Success

为什么我的策略处于失败状态？

`Failed` 状态表示策略规范包含验证错误。运行以下命令来查看错误详细信息：

```
kubectl describe tridentautogrowpolicy <policy-name>
```

常见原因包括无效的 `usedThreshold`（必须为 1–99%）、`growthAmount` 超过 `maxSize`，或无效的 Kubernetes 数量格式。请更正规范并重新应用：

```
kubectl apply -f autogrow-policy.yaml
```

为什么我无法删除策略？

策略使用终结器保护。如果卷仍在使用该策略，则删除将进入 `Deleting` 状态并等待您从策略中删除所有卷。

识别受影响的卷：

```
tridentctl get autogrowpolicy production-db-policy -o yaml
```

然后从每个 PVC 中删除注释：

```
# Option A: Explicitly disable autogrow
kubectl annotate pvc <pvc-name> \
  trident.netapp.io/autogrowPolicy="none" \
  --overwrite

# Option B: Remove the annotation entirely
kubectl annotate pvc <pvc-name> \
```

```
trident.netapp.io/autogrowPolicy-
```

删除所有卷后，Trident 释放终结器并删除此策略。

Autogrow 支持哪些后端？

Autogrow 支持 ONTAP 后端。它还支持 NAS 和 SAN 卷的 Google Cloud NetApp Volumes (GCNV)。

Autogrow 支持 NFS、iSCSI、FCP 和 NVMe 协议。NVMe 原始块卷需要 ONTAP 9.16.1 或更高版本。

棕色字段部署中的现有卷可能需要在自动扩展生效之前完成卷发布迁移。要验证迁移状态，请检查 Trident 控制器日志：

```
kubectl logs -l app=trident-controller -n trident | grep "Migration completed"
```

以下 StorageClass 示例显示为 NAS 和 SAN 后端配置的自动增长：

NAS 后端

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-autogrow
  annotations:
    trident.netapp.io/autogrowPolicy: "standard-autogrow"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-nas"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN 后端

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: database-storage
  annotations:
    trident.netapp.io/autogrowPolicy: "production-db-policy"
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
  fsType: "ext4"
allowVolumeExpansion: true
```

SAN 卷的最低增长量是多少？

对于 SAN 卷，有效最低增长量为 51 MB。如果将 `growthAmount` 配置为 50 MiB 或更少，Trident 会自动将调整大小操作的增长增加到 51 MB。

例如，此策略设置 `growthAmount` 的 `"40Mi"`，但 Trident 对使用它的任何 SAN 卷应用 51 MB 增长：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-minimal-policy
spec:
  usedThreshold: "85%"
  growthAmount: "40Mi"
  maxSize: "100Gi"
```

要避免这种自动调整，请将 `growthAmount` 设置为大于 50 MiB 的值：

```
apiVersion: trident.netapp.io/v1
kind: TridentAutogrowPolicy
metadata:
  name: san-policy
spec:
  usedThreshold: "85%"
  growthAmount: "100Mi"
  maxSize: "500Gi"
```

导入 Trident 卷

您可以使用 `tridentctl import` 或通过创建带有 Trident 导入注释的持久卷声明 (PVC) 将现有存储卷导入为 Kubernetes PV。

概述和注意事项

您可以将卷导入 Trident 以：

- 容器化应用程序并重用其现有数据集
- 将数据集的克隆用于临时应用程序
- 重建失败的 Kubernetes 集群
- 在灾难恢复期间迁移应用程序数据

注意事项

在导入卷之前，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）类型卷是 SnapMirror 目标卷。在将卷导入 Trident 之前，您应该中断镜像关系。
- 我们建议导入没有活动连接的卷。要导入正在使用的卷，请克隆该卷，然后执行导入。



这对于块卷尤其重要，因为 Kubernetes 不会意识到以前的连接，并且可以轻松地将活动卷附加到 pod。这可能会导致数据损坏。

- 虽然 `StorageClass` 必须在 PVC 上指定，但 Trident 在导入过程中不使用此参数。存储类用于在卷创建期间根据存储特性从可用池中进行选择。由于卷已存在，因此在导入过程中不需要选择池。因此，即使卷存在于与 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。卷由存储驱动程序导入后，PV 将使用 ClaimRef 创建到 PVC。
 - 回收策略最初在 PV 中设置为 `retain`。Kubernetes 成功绑定 PVC 和 PV 后，回收策略将更新为与 Storage Class 的回收策略匹配。
 - 如果 Storage Class 的回收策略为 `delete`，Trident 会在删除 PV 时删除存储卷。
- 默认情况下，Trident 管理 PVC 并重命名 FlexVol 卷以及后端上的 LUN 或 qtree。您可以传递 `--no-manage` 标记以导入非托管卷，并传递 `--no-rename` 标记以保留卷名称。
 - `--no-manage*` - 如果您使用 `--no-manage` 标志，Trident 不会在对象生命周期中对 PVC 或 PV 执行任何附加操作。删除 PV 时不会删除存储卷，也会忽略其他操作，如卷克隆和卷大小调整。
 - `--no-rename*` - 如果使用 `--no-rename` 标志，Trident 将保留现有卷名并管理卷的生命周期。此选项仅支持 `ontap-nas`、`ontap-san`（包括 ASA r2 系统）、`ontap-san-economy` 和 `ontap-nas-economy`（qtree 导入）驱动程序。



如果要将 Kubernetes 用于容器化工作负载，但要管理 Kubernetes 之外的存储卷的生命周期，则这些选项非常有用。

- 在 PVC 和 PV 上添加了一个注释，该注释具有双重目的，用于指示已导入该卷以及是否已管理 PVC 和 PV。不应修改或删除此注释。

导入卷

您可以使用 `tridentctl import` 或通过创建带有 Trident 导入注释的 PVC 来导入卷。



如果使用 PVC 注释，则无需下载或使用 `tridentctl` 来导入卷。

使用 tridentctl

步骤

1. 创建一个 PVC 文件（例如，`pvc.yaml`），该文件将用于创建 PVC。PVC 文件应包括 `name`、`namespace`、`accessModes` 和 `storageClassName`。或者，您可以在 PVC 定义中指定 `unixPermissions`。

以下是最小规格的示例：

```
kind: PersistentVolumeClaim
apiVersion: v1
```

```

metadata:
  name: my_claim
  namespace: my_namespace
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: my_storage_class

```



仅包括所需的参数。PV 名称或卷大小等附加参数可能导致导入命令失败。

2. 使用 `tridentctl import` 命令可指定包含卷的 Trident 后端的名称以及唯一标识存储上卷的名称（例如：ONTAP FlexVol、Element Volume）。指定 PVC 文件的路径需要 `-f` 参数。

```

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

```

使用 PVC 注释

步骤

1. 使用所需的 Trident 导入注释创建 PVC YAML 文件（例如，pvc.yaml）。PVC 文件应包括：

- name 和 namespace 在元数据中
- accessModes、resources.requests.storage 和 storageClassName 在规格中
- 标注：
 - trident.netapp.io/importOriginalName: 后端上的卷名称
 - trident.netapp.io/importBackendUUID: 卷存在的后端 UUID
 - trident.netapp.io/notManaged (*Optional*): 为非托管卷设置为 "true"。默认值为 "false"。

下面是导入托管卷的示例规范：

```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "<volume-name>"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteOnce
  resources:

```

```
requests:
  storage: <size>
storageClassName: <storage-class-name>
```

2. 将 PVC YAML 文件应用到 Kubernetes 集群：

```
kubectl apply -f <pvc-file>.yaml
```

Trident 将自动导入卷并将其绑定到 PVC。

示例

查看以下受支持驱动程序的卷导入示例。

ONTAP NAS 和 ONTAP NAS FlexGroup

Trident 支持使用 `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序进行卷导入。



- 该 ``ontap-nas-economy`` 驱动程序在 `qtree` 级别管理存储，因此不支持单独导入 FlexVol 卷。要使用 ``ontap-nas-economy`` 驱动程序导入 `qtree`，请参阅[导入 qtree](#)（技术预览）。当 Trident 导入 `qtree` 时，它还会将父 FlexVol 作为包含该 `qtree` 的存储桶一并导入。
- `ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序不允许使用重复的卷名。

使用 `ontap-nas` 驱动程序创建的每个卷都是 ONTAP 集群上的一个 FlexVol 卷。使用 `ontap-nas` 驱动程序导入 FlexVol 卷的工作原理相同。ONTAP 集群上已存在的 FlexVol 卷可以作为 `ontap-nas` PVC 导入。同样，FlexGroup 卷可以作为 `ontap-nas-flexgroup` PVC 导入。

使用 `tridentctl` 的 ONTAP NAS 示例

以下示例演示如何使用 `tridentctl` 导入托管卷和非托管卷。

托管卷

以下示例导入名为 `managed_volume` 的卷到名为 `ontap_nas` 的后端：

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

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

```
+-----+-----+-----+
+-----+-----+-----+-----+-----+
```

非受管卷

使用 `--no-manage` 参数时，Trident 不会重命名卷。

以下示例在 `ontap_nas` 后端导入 `unmanaged_volume`：

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

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

使用 **PVC** 注释的 **ONTAP NAS** 示例

以下示例演示如何使用 PVC 注释导入托管和非托管卷。

托管卷

以下示例从后端 `81abcb27-ea63-49bb-b606-0a5315ac5f21` 导入名为 `ontap_volume1` 的 1GiB `ontap-nas` 卷，使用 PVC 注释设置了 RWO 访问模式：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <managed-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap_volume1"
    trident.netapp.io/importBackendUUID: "81abcb27-ea63-49bb-b606-0a5315ac5f21"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
```

```
storage: 1Gi
storageClassName: <storage-class-name>
```

非受管卷

以下示例使用 PVC 注释从后端 34abcb27-ea63-49bb-b606-0a5315ac5f34 导入名为 `ontap-volume2` 的 1Gi `ontap-nas` 卷，并设置 RWO 访问模式：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <unmanaged-imported-volume>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "ontap-volume2"
    trident.netapp.io/importBackendUUID: "34abcb27-ea63-49bb-b606-0a5315ac5f34"
    trident.netapp.io/notManaged: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: <storage-class-name>
```

导入 qtree



Qtree 导入是 Trident 26.06 中的技术预览功能。技术预览功能不支持用于生产环境。

从 Trident 26.06 开始，您可以使用 `ontap-nas-economy` 驱动程序将现有的 ONTAP qtree 导入 Trident。通过 FlexVol volume 和 qtree 名称以 `flexvol/qtree` 格式标识 qtree，以及包含该 qtree 的后端的 UUID。[概述和注意事项]中的一般导入注意事项也适用于 qtree 导入。

使用 tridentctl 导入 qtree

以 flexvol/qtree 格式将 qtree 指定为卷名称。以下示例从后端 ontap_nas_economy 上的 FlexVol volume flexvol1 导入 qtree qtrees1：

```
tridentctl import volume ontap_nas_economy flexvol1/qtrees1 -f <path-to-pvc-file>
```

要导入 qtree 而不重命名它，请添加 `--no-rename` 标志。要导入 qtree 而不让 Trident 管理其生命周期，请添加 `--no-manage` 标志。

使用 PVC 注释导入 qtree

使用以下注释创建 PVC：

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <pvc-name>
  namespace: <namespace>
  annotations:
    trident.netapp.io/importOriginalName: "flexvoll1/qtree1"
    trident.netapp.io/importBackendUUID: "<backend-uuid>"
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: <size>
  storageClassName: <storage-class-name>
```

要保留现有 FlexVol 卷和 qtree 名称，请添加 `trident.netapp.io/importNoRename: "true"` 注释。要将 qtree 导入为非托管，请添加 `trident.netapp.io/notManaged: "true"` 注释。

命名和管理行为

托管导入(默认)

Trident 重命名 qtree 以匹配 PVC 名称，并将 FlexVol 卷重命名为 `ontap-nas-economy` 命名约定。接合路径不变。

--no-rename

Trident 保留现有的 FlexVol volume 和 qtree 名称。名称必须已遵循 Trident 命名约定，否则 Trident 无法管理 qtree 的删除或挂载操作。

--no-manage

Trident 引用 qtree 但不管理其生命周期。名称必须遵循 Trident 命名约定，Trident 不使用或管理 qtree 的自动导出策略。

导出策略行为

如果后端使用自动导出策略，则导入的 qtree 使用每 qtree 导出策略模型，其中每个 qtree 都有自己的导出策略。Trident 在导入时为 qtree 分配一个空导出策略。在发布时，如果启用了自动导出且现有 FlexVol 导出策略为 `default` 或为空，则 Trident 会将 FlexVol 卷更新为每个后端导出策略。



更新 FlexVol 卷导出策略可能会中断当前挂载的同一 FlexVol 卷中的其他 qtree。仅在 FlexVol 卷中的 qtree 已卸载时才导入 qtree。



不要将两个 Kubernetes 集群指向同一个活动 SVM，也不要导入另一个集群仍在管理的 qtree。如果两个集群管理同一个卷，冲突的操作可能会损坏或破坏数据。

ONTAP SAN

Trident 支持使用 `ontap-san` (iSCSI、NVMe/TCP 和 FC) 和 `ontap-san-economy` 驱动程序进行卷导入。

Trident 可以导入包含单个 LUN 的 ONTAP SAN FlexVol 卷。这与 `ontap-san` 驱动程序一致，该驱动程序为每个 PVC 创建一个 FlexVol 卷，并在 FlexVol 卷中创建一个 LUN。Trident 导入 FlexVol 卷并将其与 PVC 定义相关联。Trident 可以导入包含多个 LUN 的 `ontap-san-economy` 卷。

以下示例显示了如何导入托管卷和非托管卷：

托管卷

对于托管卷，Trident 会将 FlexVol 卷重命名为 `pvc-<uuid>` 格式，并将 FlexVol 卷中的 LUN 重命名为 `lun0`。

以下示例导入存在于 `ontap_san_default` 后端的 `ontap-san-managed` FlexVol volume：

```
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
pvc-d6ee4f54-4e40-4454-92fd-d00fc228d74a	20 MiB	basic
block	cd394786-ddd5-4470-adc3-10c5ce4ca757	online

非受管卷

以下示例在 `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
pvc-1fc999c9-ce8c-459c-82e4-ed4380a4b228	1.0 GiB	san-blog
block	e3275890-7d80-4af6-90cc-c7a0759f555a	online

如果您的 LUN 映射到与 Kubernetes 节点 IQN 共享 IQN 的 igroup，如以下示例所示，您将收到错误：LUN already mapped to initiator(s) in this group。您需要删除启动器或取消映射 LUN 以导入卷。

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

ONTAP SAN-economy 示例

以下示例演示如何为 ontap-san-economy 后端导入托管和非托管卷。

托管卷

当您导入托管卷时，Trident 将获得 FlexVol 的所有权并重命名它。当您从同一个 FlexVol 导入多个 LUN 时，您必须考虑此重命名。

以下示例将 `lun1` 从 FlexVol `toimport` 导入为名为 `vol-managed-saneco` 的受管卷：

```
tridentctl import volume vol-managed-saneco toimport/lun1 -f
import1.yaml
```

导入 `lun1` 后，Trident 将 FlexVol 重命名（例如，重命名为 `trident\_lun\_pool\_xyz`）。要从相同的 FlexVol 导入其他 LUN，请使用新的 FlexVol 名称：

```
tridentctl import volume vol-managed-saneco trident_lun_pool_xyz/lun2
-f import2.yaml
```



ontap-san-economy 后端一次导入一个 LUN。您可以使用脚本自动执行多个导入。

非受管卷

当您导入非托管卷时，Trident 不拥有 FlexVol。但是，FlexVol 和 LUN 必须遵循 Trident 命名约定。

FlexVol 命名格式

```
trident_lun_pool_STORAGEPREFIX_RANDOMSTRING
```

- STORAGEPREFIX 是后端配置中 storagePrefix 的值。默认值为 trident。
- RANDOMSTRING 是您选择的任何字符串。

LUN 命名要求

必须为 LUN 命名 lun0。

示例

如果您的 storagePrefix 是 `xyz`，则 LUN 的完整路径为：

```
trident_lun_pool_xyz_randomstring/lun0
```

Element

Trident 支持 NetApp Element 软件和 NetApp HCI 卷导入，使用 `solidfire-san` 驱动程序。



Element 驱动程序支持重复的卷名。但是，如果存在重复的卷名，Trident 返回错误。作为一种解决方法，请克隆该卷，提供唯一的卷名，然后导入克隆的卷。

以下示例将在后端 `element\_default` 导入 `element-managed` 卷。

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

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

Azure NetApp Files

Trident 支持使用 `azure-netapp-files` 驱动程序进行卷导入。



若要导入 Azure NetApp Files 卷，请通过其卷路径识别卷。卷路径是卷的导出路径在 `:/` 之后的部分。例如，如果装载路径为 `10.0.0.2:/importvol1`，则卷路径为 `importvol1`。

以下示例将在后端 `azurenetafiles\_40517` 上导入一个 `azure-netapp-files` 卷，卷路径为 `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      |
+-----+-----+-----+
+-----+-----+-----+-----+
```

Google Cloud NetApp Volumes

Trident 支持使用 `google-cloud-netapp-volumes` 驱动程序进行卷导入。

以下示例在后端 backend-tbc-gcnv1 上导入卷 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
|
+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+
```

当两个卷位于同一区域时，以下示例导入 `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 Trident 卷名和标签

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映射到相应的 Kubernetes 资源 (PVC)。您还可以在后端级别定义模板，用于创建自定义卷名和自定义标签；您创建、导入或克隆的任何卷都将遵守模板。

开始之前

可定制的卷名和标签支持：

- 卷创建、导入和克隆操作。
- 对于 `ontap-nas-economy` 驱动程序，只有 Qtree 卷的名称符合名称模板。
- 对于 `ontap-san-economy` 驱动程序，只有 LUN 名称符合名称模板。

限制

- 自定义卷名仅与 ONTAP 本地驱动程序兼容。
- 只有 `ontap-san`、`ontap-nas` 和 `ontap-nas-flexgroup` 驱动程序才支持自定义标签。
- 自定义卷名称不适用于现有卷。

可定制卷名的关键行为

- 如果由于名称模板中的语法无效而发生故障，则后端创建将失败。但是，如果模板应用失败，将根据现有的命名约定命名卷。
- 当卷使用后端配置中的名称模板命名时，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。

使用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.RequestName}}"
  },
  "labels": {
    "cluster": "ClusterA",
    "PVC": "{{.volume.Namespace}}_{{.volume.RequestName}}"
  }
}
```

Pool 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，仅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. 对于托管卷导入，卷名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切片操作符。
4. 如果模板没有产生唯一的卷名，Trident 将附加几个随机字符以创建唯一的卷名。
5. 如果 NAS 经济卷的自定义名称长度超过 64 个字符，Trident 将根据现有的命名约定命名卷。对于所有其他 ONTAP 驱动程序，如果卷名超过名称限制，则卷创建过程将失败。

跨命名空间共享 Trident NFS 卷

使用 Trident，您可以在主命名空间中创建卷，并在一个或多个辅助命名空间中共享它。

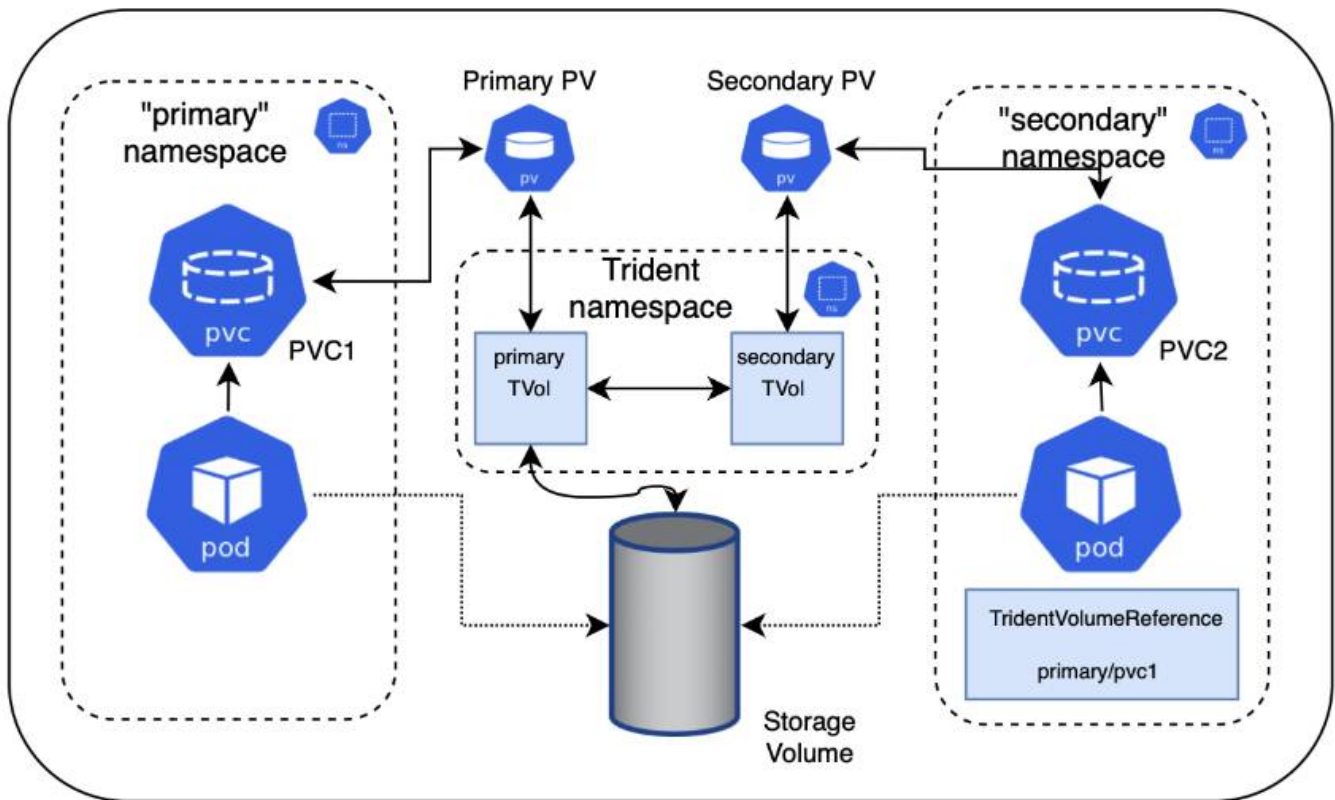
功能

TridentVolumeReference CR 允许您在一个或多个 Kubernetes 命名空间之间安全地共享 ReadWriteMany (RWX) NFS 卷。此 Kubernetes 原生解决方案具有以下优势：

- 多级访问控制，确保安全

- 适用于所有 Trident NFS 卷驱动程序
- 不依赖 tridentctl 或任何其他非本地 Kubernetes 功能

此图说明了两个 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 中的数据的数据的权限。

2

授予在目标命名空间中创建 **CR** 的权限

群集管理员授予目标命名空间的所有者创建 **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 以使用来自源 PVC 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。在每个步骤中指定用户角色。

步骤

1. 源命名空间所有者：在源命名空间中创建 PVC (pvc1)，通过使用 `shareToNamespace`` 注解，授予与目标命名空间 (``namespace2`) 共享的权限。

```
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 创建 PV 及其后端 NFS 存储卷。



- 您可以使用逗号分隔的列表将 PVC 共享到多个命名空间。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: *`
- 您可以随时更新 PVC 以包含 ``shareToNamespace`` 注释。

2. *集群管理员：*确保已设置正确的 RBAC，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：在目标命名空间 (namespace2) 中创建一个 PVC (pvc2)，并使用 shareFromPVC 注解来指定源 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 读取目标 PVC 上的 `shareFromPVC` 注释，并将目标 PV 创建为没有自己的存储资源的从属卷，该卷指向源 PV 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。Trident 将删除对源命名空间上卷的访问权限，并保留对共享该卷的其他命名空间的访问权限。当引用该卷的所有命名空间都被删除时，Trident 会删除该卷。

使用 `tridentctl get` 查询从属卷

使用该[tridentctl`实用程序，您可以运行该 `get` 命令来获取从属卷。有关详细信息，请参见 `tridentctl` 命令和选项。

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

标记：

- `-h, --help`: 卷的帮助。

- `--parentOfSubordinate string`: 将查询限制为从属源卷。
- `--subordinateOf string`: 将查询限制为卷的下属。

限制

- Trident 无法阻止目标命名空间写入共享卷。您应该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 `shareToNamespace`或`shareFromNamespace`注释或删除`TridentVolumeReference CR` 来撤销对源 PVC 的访问权限。要撤销访问权限，您必须删除从属 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更多信息：

- 观看 ["NetAppTV"](#) 上的演示。

跨命名空间克隆 Trident 卷

使用 Trident，您可以使用来自同一 Kubernetes 集群内不同命名空间的现有卷或卷快照创建新卷。

前提条件

在克隆卷之前，请确保源和目标后端的类型相同并且具有相同的存储类。



只有 `ontap-san` 和 `ontap-nas` 存储驱动程序才支持跨命名空间克隆。不支持只读克隆。

快速入门

只需几步即可设置卷克隆。

1

配置源 **PVC** 以克隆卷

源命名空间所有者授予访问源 PVC 中的数据权限。

2

授予在目标命名空间中创建 **CR** 的权限

群集管理员授予目标命名空间的所有者创建 `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创建 `TridentVolumeReference CR` 以引用源 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的协作和操作。在每个步骤中指定用户角色。

步骤

1. 源命名空间所有者：在源命名空间(namespace1)中创建 PVC(pvc1)，通过使用 cloneToNamespace 注解，授予与目标命名空间(namespace2)共享的权限。

```
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 创建 PV 及其后端存储卷。



- 您可以使用逗号分隔的列表将 PVC 共享到多个命名空间。例如，
trident.netapp.io/cloneToNamespace: namespace2, namespace3, namespace4。
- 您可以使用 * 共享到所有命名空间。例如，
trident.netapp.io/cloneToNamespace: *
- 您可以随时更新 PVC 以包含 cloneToNamespace 注释。

2. 集群管理员：确保已设置正确的 RBAC，以授予目标命名空间所有者为目标命名空间中创建 TridentVolumeReference 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一个引用源 PVC 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：在目标命名空间 (namespace2) 中创建一个 PVC (pvc2)，使用 `cloneFromPVC` 或 `cloneFromSnapshot`，以及 `cloneFromNamespace` 注解来指定源 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 `ontap-nas-economy` 驱动程序配置的 PVC，不支持只读克隆。

使用 SnapMirror 复制卷

Trident 支持一个集群上的源卷与对等集群上的目标卷之间的镜像关系，用于复制数据以进行灾难恢复。您可以使用命名空间的自定义资源定义 (CRD)，称为 Trident Mirror Relationship (TMR) 来执行以下操作：

- 创建卷 (PVC) 之间的镜像关系
- 删除卷之间的镜像关系
- 中断镜像关系
- 在灾难情况（故障转移）期间提升辅助卷
- 执行应用程序从集群到集群的无损转换（在计划的故障转移或迁移期间）

复制先决条件

在开始之前，请确保满足以下先决条件：

ONTAP 集群

- **Trident**：Trident 版本 22.10 或更高版本必须存在于使用 ONTAP 作为后端的源和目标 Kubernetes 集群上。
- **许可证**：必须在源和目标 ONTAP 集群上启用使用数据保护捆绑包的 ONTAP SnapMirror 异步许可证。请参阅 ["ONTAP 中的 SnapMirror 许可概述"](#) 以了解更多信息。

从 ONTAP 9.10.1 开始，所有许可证都以 NetApp 许可证文件 (NLF) 的形式交付，该文件是启用多个功能的单个文件。有关详细信息，请参见["ONTAP One 附带的许可证"](#)。



仅支持 SnapMirror 异步保护。

对等

- **集群和 SVM**：必须对等 ONTAP 存储后端。请参阅 ["集群和 SVM 对等概述"](#) 以了解更多信息。



确保在两个 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 必须可用于目标集群上的 Trident。

支持的驱动程序

NetApp Trident 支持使用由以下驱动程序支持的存储类的 NetApp SnapMirror 技术进行卷复制：**ontap-nas**：**NFS** ontap-san: iSCSI **ontap-san: FC** ontap-san: NVMe/TCP（需要最低 ONTAP 版本 9.15.1）



ASA r2 系统不支持使用 SnapMirror 进行卷复制。有关 ASA r2 系统的信息，请参见 ["了解 ASA r2 存储系统"](#)。

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

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 MirrorRelationship CR。

示例

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

Trident 获取卷的内部信息和卷的当前数据保护 (DP) 状态，然后填充 MirrorRelationship 的 status 字段。

- d. 获取 TridentMirrorRelationship 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. 请在辅助 Kubernetes 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. 创建包含目标和源信息的 `MirrorRelationship CR`。

示例

```

kind: TridentMirrorRelationship
apiVersion: trident.netapp.io/v1
metadata:
  name: csi-nas
spec:
  state: established
  replicationPolicy: MirrorAllSnapshots
  replicationSchedule: 5min
  volumeMappings:
  - localPVCName: csi-nas
    remoteVolumeHandle:
"datavserver:trident_pvc_3bedd23c_46a8_4384_b12b_3c38b313c1e1"

```

TMR spec 有四个顶级字段：`state`，用于设置所需的关系状态（请参阅 [\[卷复制状态\]](#)）；`volumeMappings`，用于将本地 PVC 与其远程卷句柄相关联；以及下面描述的可选 `replicationPolicy` 和 `replicationSchedule` 字段。

`replicationPolicy` 和 `replicationSchedule` 字段是可选的。将 `replicationPolicy` 设置为 ONTAP 上的异步 SnapMirror 策略的名称，并将 `replicationSchedule` 设置为 ONTAP 计划的名称。如果省略任一字段，Trident 将使用在后端配置的相应复制策略或计划。在创建关系时设置这些字段；建立关系后无法更改策略或计划。要使用不同的设置，请删除并重新创建 TMR。

Trident 使用配置的复制策略和计划创建并初始化 SnapMirror 关系。

1. 使用先前创建的 StorageClass 创建 PVC 以充当辅助（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 TridentMirrorRelationship CRD，如果关系不存在，则无法创建卷。如果存在此关系，Trident 将确保将新 FlexVol 卷放置到与 MirrorRelationship 中定义的远程 SVM 对等的 SVM 上。

卷复制状态

Trident 镜像关系（TMR）是表示 PVC 之间复制关系的一端的 CRD。目标 TMR 有一个状态，它告诉 Trident 所需的状态是什么。目标 TMR 具有以下状态：

- **Established**：本地 PVC 是镜像关系的目标卷，这是一种新的关系。
- **Promoted**：本地 PVC 是 ReadWrite 且可挂载的，目前没有镜像关系生效。
- **已重新建立**：本地 PVC 是镜像关系的目标卷，此前也处于该镜像关系中。
 - 如果目标卷与源卷之间曾经存在关系，则必须使用重新建立的状态，因为它会覆盖目标卷的内容。
 - 如果卷以前未与源建立关系，则重新建立状态将失败。

在计划外故障转移期间提升辅助 PVC

在辅助 Kubernetes 集群上执行以下步骤：

- 将 TridentMirrorRelationship 的 *spec.state* 字段更新为 `promoted`。

在计划的故障转移期间推广辅助 PVC

在计划的故障转移（迁移）期间，请执行以下步骤来升级辅助 PVC：

步骤

1. 在主 Kubernetes 集群上，创建 PVC 的快照，并等待创建快照。
2. 在主 Kubernetes 集群上，创建 TridentTridentSnapshotInfoCR 以获取内部详细信息。

示例

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

- a. 在辅助 Kubernetes 集群上，将 *TridentMirrorRelationship* CR 的 *spec.state* 字段更新为 *promoted*，并将 *spec.volumeMappings[].promotedSnapshotHandle* 更新为快照的 *internalName*。
3. 在辅助 Kubernetes 集群上，确认 *TridentMirrorRelationship* 的状态（*status.state* 字段）为 *promoted*。

故障转移后恢复镜像关系

在恢复镜像关系之前，请选择要作为新主要关系的一侧。

步骤

1. 在辅助 Kubernetes 集群上，请确保更新了 *TridentMirrorRelationship* 上的 *spec.remoteVolumeHandle* 字段的值。
2. 在辅助 Kubernetes 集群上，将 *TridentMirrorRelationship* 的 *spec.state* 字段更新为 *reestablished*。

其他操作

Trident 支持主卷和二级卷上的以下操作：

将主要 **PVC** 复制到新的次要 **PVC**

请确保已有一个主要 PVC 和一个次要 PVC。

步骤

1. 从已建立的辅助（目标）集群中删除 *PersistentVolumeClaim* 和 *TridentMirrorRelationship* CRD。
2. 从主（源）集群中删除 *TridentMirrorRelationship* CRD。
3. 在要建立的新辅助（目标）PVC 的主（源）集群上创建新的 *TridentMirrorRelationship* CRD。

调整镜像、主 **PVC** 或辅助 **PVC** 的大小

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

从 PVC 中删除复制

要删除复制，请在当前辅助卷上执行以下操作之一：

- 删除辅助 PVC 上的 `MirrorRelationship`。这会破坏复制关系。
- 或者，将 `spec.state` 字段更新为 *promoted*。

删除 PVC（先前已镜像）

Trident 检查复制的 PVC，并在尝试删除卷之前释放复制关系。

删除 TMR

删除镜像关系一侧的 TMR 会导致剩余的 TMR 在 Trident 完成删除之前过渡到 *promoted* 状态。如果选择删除的 TMR 已经处于 *promoted* 状态，则不存在现有的镜像关系，TMR 将被删除，Trident 会将本地 PVC 提升为 *ReadWrite*。此删除将释放 ONTAP 中本地卷的 `SnapMirror` 元数据。如果将来在镜像关系中使用此卷，则在创建新的镜像关系时，必须使用具有 *established* 卷复制状态的新 TMR。

ONTAP 联机时更新镜像关系

镜像关系建立后可以随时更新。您可以使用 ``state: promoted`` 或 ``state: reestablished`` 字段更新关系。将目标卷升级到常规 *ReadWrite* 卷时，您可以使用 *promotedSnapshotHandle* 指定要将当前卷还原到的特定快照。

ONTAP 离线时更新镜像关系

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 `TridentActionMirrorUpdate` 格式示例：

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` 反映了 `TridentActionMirrorUpdate` CRD 的状态。它可以取自 *Succeeded*、*In Progress* 或 *Failed* 的值。

使用 CSI 拓扑

Trident 可以通过使用 "CSI 拓扑功能" 选择性地创建卷并将其附加到 Kubernetes 集群中的节点。

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节点的子集。如今，云提供商使 Kubernetes 管理员能够生成基于区域的节点。节点可以位于一个区域内的不同可用区，也可以位于不同的区域。为了便于在多区域架构中为工作负载配置卷，Trident 使用了 CSI 拓扑。



详细了解 CSI Topology 功能 ["此处"](#)。

Kubernetes 提供两种独特的卷绑定模式：

- 当 VolumeBindingMode 设置为 Immediate 时，Trident 会在没有任何拓扑感知的情况下创建卷。创建 PVC 时会处理卷绑定和动态配置。这是默认 VolumeBindingMode，适用于不强制执行拓扑约束的集群。创建持久卷时不依赖于请求 Pod 的调度要求。
- 将 VolumeBindingMode 设置为 WaitForFirstConsumer 时，PVC 的 Persistent Volume 的创建和绑定会被延迟，直到使用该 PVC 的 pod 被调度和创建。通过这种方式，创建的卷可以满足拓扑要求所实施的调度约束。



The WaitForFirstConsumer 绑定模式不需要拓扑标签。这可以独立于 CSI 拓扑功能使用。

您需要什么

要使用 CSI 拓扑，您需要以下内容：

- 运行 ["支持的 Kubernetes 版本"](#) 的 Kubernetes 集群

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"19",
GitVersion:"v1.19.3",
GitCommit:"1e11e4a2108024935ecfcb2912226cedaafd99df",
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:"1e11e4a2108024935ecfcb2912226cedaafd99df",
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引入拓扑感知(topology.kubernetes.io/region`和`topology.kubernetes.io/zone)的标签。在安装 Trident 之前，这些标签*应该存在于集群中的节点上*，以便 Trident 具有拓扑感知能力。

```
kubectl get nodes -o=jsonpath='{range .items[*]}[{.metadata.name},
{.metadata.labels}]{"\n"}{end}' | grep --color "topology.kubernetes.io"
[nodel,
{"beta.kubernetes.io/arch":"amd64","beta.kubernetes.io/os":"linux","kubernetes.io/arch":"amd64","kubernetes.io/hostname":"nodel","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","kubernetes.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","kubernetes.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"}}]

```

步骤 1: 创建可感知拓扑的后端

Trident 存储后端可以设计为基于可用性区域有选择地配置卷。每个后端都可以携带一个可选 `supportedTopologies` 块，该块表示受支持的区域和地区列表。对于使用此类后端的 StorageClasses，只有在支持的地区/区域中调度的应用程序请求时，才会创建卷。

以下是后端定义示例：

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` 用于为每个后端提供区域和可用区的列表。这些区域和可用区表示可以在 `StorageClass` 中提供的允许值列表。对于包含后端提供的区域和可用区子集的 `StorageClasses`, `Trident` 会在后端上创建卷。

您也可以为每个存储池定义 `supportedTopologies`。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 标签代表存储池的位置。topology.kubernetes.io/region 和 topology.kubernetes.io/zone 指示可以从哪里使用存储池。

步骤 2：定义拓扑感知的 StorageClasses

根据提供给集群中节点的拓扑标签，StorageClasses 可以定义为包含拓扑信息。这将确定用作 PVC 请求候选项的存储池，以及可以使用 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
```

在上面提供的 StorageClass 定义中，volumeBindingMode 设置为 WaitForFirstConsumer。使用此 StorageClass 请求的 PVC 在 pod 中引用之前不会被执行。并且，allowedTopologies 提供要使用的区域和地区。netapp-san-us-east1 StorageClass 在上面定义的 san-backend-us-east1 后端上创建 PVC。

步骤 3：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
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: voll
  persistentVolumeClaim:
    claimName: pvc-san
containers:
- name: sec-ctx-demo
  image: busybox
  command: [ "sh", "-c", "sleep 1h" ]
  volumeMounts:
    - name: voll
      mountPath: /data/demo
  securityContext:
    allowPrivilegeEscalation: false

```

这个 podSpec 指示 Kubernetes 在 us-east1 区域中的节点上调度 pod，并从 us-east1-a 或 us-east1-b 可用区中的任何节点中进行选择。

请参阅以下输出：

```

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**

可以更新预先存在的后端，以包含 `supportedTopologies` 使用 `tridentctl backend update` 的列表。这不会影响已配置的卷，仅用于后续 PVC。

查找更多信息

- "管理容器的资源"
- "nodeSelector"
- "亲和性和反亲和性"
- "污点和容忍"

使用 Trident 快照

持久卷 (PV) 的 Kubernetes 卷快照支持卷的时间点副本。您可以创建使用 Trident 创建的卷的快照，导入在 Trident 外部创建的快照，从现有快照创建新卷，并从快照中恢复卷数据。

概述

``ontap-nas``、``ontap-nas-flexgroup``、``ontap-san``、``ontap-san-economy``、``solidfire-san``、``azure-netapp-files`` 和 ``google-cloud-netapp-volumes`` 驱动程序支持卷快照。

开始之前

必须具有外部快照控制器和自定义资源定义 (CRD) 才能使用快照。这是 Kubernetes 编排器的职责（例如：Kubeadm、GKE、OpenShift）。

如果您的 Kubernetes 发行版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. 创建一个 `VolumeSnapshotClass`。有关详细信息，请参见 ["VolumeSnapshotClass"](#)。
 - `driver` 指向 Trident CSI 驱动程序。

- `deletionPolicy` 可以为 `Delete` 或 `Retain`。当设置为 `Retain` 时，即使删除 `VolumeSnapshot` 对象，也会保留存储集群上的底层物理快照。

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
```

- 此示例为名为 `pvc1` 的 PVC 创建卷快照对象，快照的名称设置为 `pvc1-snap`。VolumeSnapshot 类似于 PVC，并与表示实际快照的 `VolumeSnapshotContent` 对象相关联。

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

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

- 您可以通过描述 `VolumeSnapshotContent` 对象来识别 `pvc1-snap` VolumeSnapshot。`Snapshot Content Name` 标识为此快照提供服务的 `VolumeSnapshotContent` 对象。`Ready To Use` 参数表示快照可用于创建新的 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 PVC，使用名为 `` 的 VolumeSnapshot 作为数据源。创建 PVC 后，它可以连接到 pod 并像任何其他 PVC 一样使用。



PVC 将在与源卷相同的后端创建。请参阅 ["KB: 无法在备用后端从 Trident PVC 快照创建 PVC"](#)。

以下示例使用 pvc1-snap 作为数据源创建 PVC。

```
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 "[Kubernetes 预置 Snapshot 流程](#)" 使集群管理员能够创建 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：创建引用后端快照的 VolumeSnapshotContent 对象。这将启动 Trident 中的快照工作流。
 - 将 annotations 中的后端快照的名称指定为 trident.netapp.io/internalSnapshotName: <"backend-snapshot-name">。
 - 在 <name-of-parent-volume-in-trident>/<volume-snapshot-content-name> 中指定 `snapshotHandle`。这是外部快照程序在 `ListSnapshots` 调用中提供给 Trident 的唯一信息。



由于 CR 命名限制，<volumeSnapshotContentName> 无法始终匹配后端快照名称。

示例

以下示例创建一个 VolumeSnapshotContent 对象，该对象引用后端快照 `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. 集群管理员：创建引用 VolumeSnapshot 对象的 `VolumeSnapshotContent` CR。这将请求在指定命名空间中使用 `VolumeSnapshot` 的访问权限。

示例

以下示例创建了一个 VolumeSnapshot 名为 `import-snap` 的 CR，并引用了名为 `import-snap-content` 的 VolumeSnapshotContent。

```
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）：外部快照器识别新创建的 VolumeSnapshotContent 并运行 `ListSnapshots` 调用。Trident 创建 TridentSnapshot。
 - 外部快照器将 VolumeSnapshotContent 设置为 readyToUse，将 VolumeSnapshot 设置为 true。
 - Trident 返回 readyToUse=true。
4. 任何用户：创建 PersistentVolumeClaim 以引用新 VolumeSnapshot，其中 spec.dataSource（或 spec.dataSourceRef）名称是 VolumeSnapshot 名称。

示例

以下示例创建了一个 PVC，引用名为 VolumeSnapshot 的 `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

默认情况下隐藏 snapshot 目录，以促进使用 `ontap-nas` 和 `ontap-nas-economy` 驱动程序配置的卷的最大兼容性。启用 `.snapshot` 目录以直接从快照恢复数据。

使用卷快照还原 ONTAP CLI 将卷还原至先前快照中记录的状态。

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



还原快照副本时，将覆盖现有卷配置。创建快照副本后对卷数据所做的更改将丢失。

从快照就地还原卷

Trident 使用 `TridentActionSnapshotRestore` (TASR) CR 提供快速的就地卷恢复功能。此 CR 作为命令式 Kubernetes 操作，在操作完成后不会持续存在。

Trident 支持在 `ontap-san`、`ontap-san-economy`、`ontap-nas`、`ontap-nas-flexgroup`、`azure-netapp-files`、`google-cloud-netapp-volumes` 和 `solidfire-san` 驱动程序上还原快照。

开始之前

您必须具有绑定的 PVC 和可用的卷快照。

- 确认 PVC 状态已绑定。

```
kubectl get pvc
```

- 验证卷快照是否可以使用。

```
kubectl get vs
```

步骤

1. 创建 TASR CR。此示例为 PVC `pvc1` 和卷快照 `pvc1-snapshot` 创建 CR。



TASR CR 必须位于存在 PVC 和 VS 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 从快照中恢复。此示例从快照中还原 pvc1。

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

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

结果

Trident 从快照中恢复数据。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，出现故障时，Trident 不会自动重试此操作。您需要再次执行此操作。
- 没有管理员访问权的 Kubernetes 用户可能需要管理员授予权限才能在其应用程序命名空间中

删除具有关联快照的 **PV**

删除具有关联快照的永久卷时，相应的 Trident 卷将更新为"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 发行版不包括快照控制器和 CRD，您可以按以下方式部署它们。

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.yaml
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，请打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` 并更新 `namespace` 到您的命名空间。

相关链接

- ["卷快照"](#)
- ["VolumeSnapshotClass"](#)

使用 Trident 卷组快照

持久卷 (PV) 的 Kubernetes 卷组快照 NetApp Trident 提供了创建多个卷快照（一组卷快照）的功能。此卷组快照表示在同一时间点拍摄的多个卷的副本。



VolumeGroupSnapshot 是 Kubernetes 中带有 beta API 的 beta 功能。Kubernetes 1.32 是 VolumeGroupSnapshot 所需的最低版本。

创建卷组快照

以下存储驱动程序支持卷组快照：

- `ontap-san` 驱动程序 - 仅适用于 iSCSI 和 FC 协议，不适用于 NVMe/TCP 协议。
- `ontap-san-economy` - 仅适用于 iSCSI 协议。
- `ontap-nas`



NetApp ASA r2 或 AFX 存储系统不支持卷组快照。

开始之前

- 请确保您的 Kubernetes 版本为 K8s 1.32 或更高版本。
- 必须具有外部快照控制器和自定义资源定义 (CRD) 才能使用快照。这是 Kubernetes 编排器的职责（例如：Kubeadm、GKE、OpenShift）。

如果您的 Kubernetes 发行版不包括外部快照控制器和 CRD，请参阅 [\[部署卷快照控制器\]](#)。



如果在 GKE 环境中创建按需卷组快照，请勿创建快照控制器。GKE 使用内置的隐藏快照控制器。

- 在快照控制器 YAML 中，将 `CSIVolumeGroupSnapshot` 功能门设置为 `'true'`，以确保启用卷组快照。
- 在创建卷组快照之前创建所需的卷组快照类。
- 确保所有 PVC/卷都在同一个 SVM 上，以便能够创建 `VolumeGroupSnapshot`。

步骤

- 在创建 `VolumeGroupSnapshot` 之前，先创建 `VolumeGroupSnapshotClass`。有关详细信息，请参阅 ["VolumeGroupSnapshotClass"](#)。

```
apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshotClass
metadata:
```

```

name: csi-group-snap-class
annotations:
  kubernetes.io/description: "Trident group snapshot class"
driver: csi.trident.netapp.io
deletionPolicy: Delete

```

- 使用现有存储类创建具有所需标签的 PVC，或将这些标签添加到现有 PVC。

以下示例使用 `pvc1-group-snap` 作为数据源和标签 `consistentGroupSnapshot: groupA` 创建 PVC。根据您的要求定义标签键和值。

```

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc1-group-snap
  labels:
    consistentGroupSnapshot: groupA
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: sc1-1

```

- 创建具有相同标签 (`consistentGroupSnapshot: groupA` 的 `VolumeGroupSnapshot`，该标签在 PVC 中指定。

此示例创建卷组快照：

```

apiVersion: groupsnapshot.storage.k8s.io/v1beta1
kind: VolumeGroupSnapshot
metadata:
  name: "vgs1"
  namespace: trident
spec:
  volumeGroupSnapshotClassName: csi-group-snap-class
  source:
    selector:
      matchLabels:
        consistentGroupSnapshot: groupA

```

使用组快照恢复卷数据

您可以使用作为 Volume Group Snapshot 一部分创建的单个快照来恢复单个 Persistent Volume。您无法将 Volume Group Snapshot 作为一个单位进行恢复。

使用卷快照还原 ONTAP CLI 将卷还原至先前快照中记录的状态。

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



还原快照副本时，将覆盖现有卷配置。创建快照副本后对卷数据所做的更改将丢失。

从快照就地还原卷

Trident 使用 TridentActionSnapshotRestore (TASR) CR 提供快速的就地卷恢复功能。此 CR 作为命令式 Kubernetes 操作，在操作完成后不会持续存在。

有关详细信息，请参见 ["从快照就地还原卷"](#)。

删除具有关联组快照的 PV

删除组卷快照时：

- 您可以整体删除 VolumeGroupSnapshots，而不是删除组中的单个快照。
- 如果在存在该 PersistentVolume 的快照时删除 PersistentVolumes，Trident 会将该卷移动到“deleting”状态，因为必须先删除快照，才能安全地删除该卷。
- 如果已使用分组快照创建克隆，然后要删除该组，则将开始克隆拆分操作，并且在完成拆分之前无法删除该组。

部署卷快照控制器

如果您的 Kubernetes 发行版不包括快照控制器和 CRD，您可以按以下方式部署它们。

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-  
8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcl  
asses.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshotcontents.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/client/config/crd/groupsnapshot.storage.k8s.io_volumegroupsnapshots.yaml
```

2. 创建快照控制器。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-8.2/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```



如有必要，请打开 `deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml` 并更新 `namespace` 到您的命名空间。

相关链接

- ["VolumeGroupSnapshotClass"](#)
- ["卷快照"](#)

版权信息

版权所有 © 2026 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

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

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

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

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

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

商标信息

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