NetApp Astra Trident 개요

NetApp Solutions

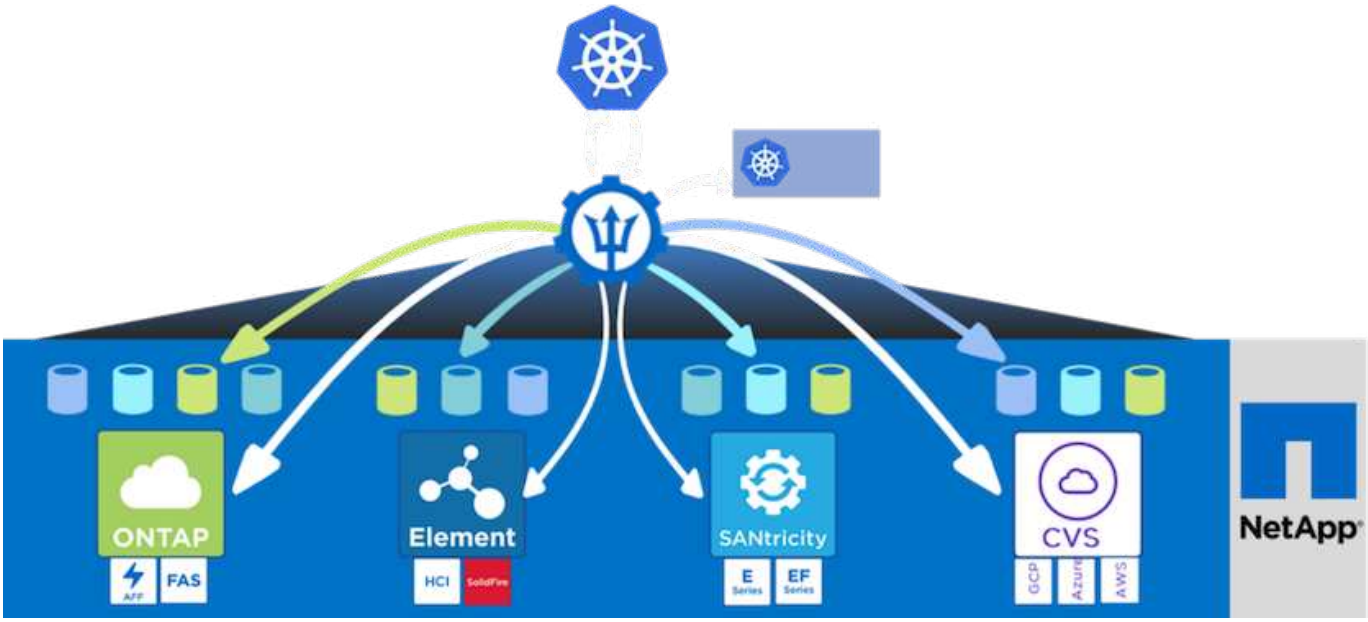
NetApp
April 20, 2024

목차

Astra Trident 개요

Astra Trident는 Red Hat OpenShift를 포함하여 컨테이너 및 Kubernetes 배포를 위한 오픈 소스 및 완전 지원되는 스토리지 오케스트레이터입니다. Trident는 NetApp ONTAP 및 Element 스토리지 시스템을 포함한 전체 NetApp 스토리지 포트폴리오와 연동되며 NFS 및 iSCSI 연결도 지원합니다. Trident는 최종 사용자가 스토리지 관리자의 개입 없이 NetApp 스토리지 시스템에서 스토리지를 프로비저닝 및 관리할 수 있도록 하여 DevOps 워크플로우를 가속합니다.

관리자는 특정 수준의 성능을 보장하는 압축, 특정 디스크 유형 또는 QoS 수준을 비롯한 고급 스토리지 기능을 지원하는 스토리지 시스템 모델과 프로젝트 요구사항에 따라 여러 스토리지 백엔드를 구성할 수 있습니다. 이러한 백엔드를 정의한 후, 개발자는 프로젝트의 이러한 백엔드를 사용하여 지속적인 PVC(Volume Claim)를 생성하고 필요에 따라 컨테이너에 영구 저장소를 연결할 수 있습니다.



Astra Trident는 빠른 개발 주기를 제공하며, Kubernetes와 마찬가지로 1년에 4회 릴리즈됩니다.

Astra Trident의 최신 버전은 2022년 1월 22.01입니다. Kubernetes 배포를 찾을 수 있는 Trident의 버전에 대한 지원 매트릭스입니다 "여기".

20.04 릴리즈부터 Trident 운영자가 Trident 설정을 수행합니다. 운영자는 대규모 구축을 용이하게 하고 Trident 설치의 일부로 배포된 Pod에 대한 자동 복구를 포함한 추가 지원을 제공합니다.

21.01 릴리즈를 통해 Trident Operator의 설치를 용이하게 하는 제어 차트를 사용할 수 있게 되었습니다.

Astra Trident를 다운로드하십시오

구축된 사용자 클러스터에 Trident를 설치하고 영구 볼륨을 프로비저닝하려면 다음 단계를 완료하십시오.

1. 설치 아카이브를 관리 워크스테이션에 다운로드하고 압축을 풉니다. Trident의 현재 버전은 22.01이며 다운로드할 수 있습니다 "여기".

```
[netapp-user@rhel7 ~]$ wget
https://github.com/NetApp/trident/releases/download/v22.01.0/trident-
```

```
installer-22.01.0.tar.gz
--2021-05-06 15:17:30--
https://github.com/NetApp/trident/releases/download/v22.01.0/trident-
installer-22.01.0.tar.gz
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-
releases.githubusercontent.com/77179634/a4fa9f00-a9f2-11eb-9053-
98e8e573d4ae?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=AKIAIWNJYAX4CSVEH53A%2F20210506%2Fus-east-
1%2Fs3%2Faws4_request&X-Amz-Date=20210506T191643Z&X-Amz-Expires=300&X-
Amz-
Signature=8a49a2a1e08c147d1ddd8149ce45a5714f9853fee19bb1c507989b9543eb36
30&X-Amz-
SignedHeaders=host&actor_id=0&key_id=0&repo_id=77179634&response-
content-disposition=attachment%3B%20filename%3Dtrident-installer-
22.01.0.tar.gz&response-content-type=application%2Foctet-stream
[following]
--2021-05-06 15:17:30--  https://github-
releases.githubusercontent.com/77179634/a4fa9f00-a9f2-11eb-9053-
98e8e573d4ae?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=AKIAIWNJYAX4CSVEH53A%2F20210506%2Fus-east-
1%2Fs3%2Faws4_request&X-Amz-Date=20210506T191643Z&X-Amz-Expires=300&X-
Amz-
Signature=8a49a2a1e08c147d1ddd8149ce45a5714f9853fee19bb1c507989b9543eb36
30&X-Amz-
SignedHeaders=host&actor_id=0&key_id=0&repo_id=77179634&response-
content-disposition=attachment%3B%20filename%3Dtrident-installer-
22.01.0.tar.gz&response-content-type=application%2Foctet-stream
Resolving github-releases.githubusercontent.com (github-
releases.githubusercontent.com)... 185.199.108.154, 185.199.109.154,
185.199.110.154, ...
Connecting to github-releases.githubusercontent.com (github-
releases.githubusercontent.com)|185.199.108.154|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 38349341 (37M) [application/octet-stream]
Saving to: 'trident-installer-22.01.0.tar.gz'

100%[=====================================================================
==============================================>] 38,349,341  88.5MB/s
in 0.4s

2021-05-06 15:17:30 (88.5 MB/s) - 'trident-installer-22.01.0.tar.gz'
saved [38349341/38349341]
```

2. 다운로드한 번들에서 Trident 설치를 추출합니다.

```
[netapp-user@rhel7 ~]$ tar -xzf trident-installer-22.01.0.tar.gz
[netapp-user@rhel7 ~]$ cd trident-installer/
[netapp-user@rhel7 trident-installer]$
```

Hrom을 사용하여 Trident 연산자를 설치합니다

1. 먼저 사용자 클러스터의 "kubecononfig" 파일 위치를 환경 변수로 설정하여 Trident에 이 파일을 전달할 수 있는 옵션이 없으므로 참조할 필요가 없습니다.

```
[netapp-user@rhel7 trident-installer]$ export KUBECONFIG=~/ocp-
install/auth/kubeconfig
```

2. Helm 명령을 실행하여 사용자 클러스터에 삼중덴트 네임스페이스를 생성하는 동안 Helm 디렉토리의 tarball에서 Trident 연산자를 설치합니다.

```
[netapp-user@rhel7 trident-installer]$ helm install trident
helm/trident-operator-22.01.0.tgz --create-namespace --namespace trident
NAME: trident
LAST DEPLOYED: Fri May  7 12:54:25 2021
NAMESPACE: trident
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing trident-operator, which will deploy and manage
NetApp's Trident CSI
storage provisioner for Kubernetes.

Your release is named 'trident' and is installed into the 'trident'
namespace.
Please note that there must be only one instance of Trident (and
trident-operator) in a Kubernetes cluster.

To configure Trident to manage storage resources, you will need a copy
of tridentctl, which is
available in pre-packaged Trident releases.  You may find all Trident
releases and source code
online at https://github.com/NetApp/trident.

To learn more about the release, try:

  $ helm status trident
  $ helm get all trident
```

3. 네임스페이스에서 실행 중인 포드를 확인하거나 tridentctl 바이너리를 사용하여 설치된 버전을 확인하여 Trident가 성공적으로 설치되었는지 확인할 수 있습니다.

```
[netapp-user@rhel7 trident-installer]$ oc get pods -n trident
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-5z45l                   1/2     Running   2          30s
trident-csi-696b685cf8-htdb2        6/6     Running   0          30s
trident-csi-b74p2                   2/2     Running   0          30s
trident-csi-lrw4n                   2/2     Running   0          30s
trident-operator-7c748d957-gr2gw    1/1     Running   0          36s

[netapp-user@rhel7 trident-installer]$ ./tridentctl -n trident version
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 22.01.0          | 22.01.0          |
+----------------+----------------+
```

경우에 따라 고객 환경에 Trident 구축의 사용자 지정이 필요할 수 있습니다. 이러한 경우 Trident 연산자를 수동으로 설치하고 포함된 매니페스트를 업데이트하여 배포를 사용자 지정할 수도 있습니다.

Trident 연산자를 수동으로 설치합니다

1. 먼저, Trident에 이 파일을 전달할 수 있는 옵션이 없기 때문에 사용자 클러스터의 "kubecononfig" 파일을 참조할 필요가 없도록 환경 변수로 설정합니다.

```
[netapp-user@rhel7 trident-installer]$ export KUBECONFIG=~/ocp-
install/auth/kubeconfig
```

2. 트리덴트 설치 프로그램 디렉토리에는 필요한 모든 리소스를 정의하기 위한 매니페스트가 들어 있습니다. 적절한 매니페스트를 사용하여 '트리엔오케스트레이터' 사용자 지정 리소스 정의를 만듭니다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f
deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml
customresourcedefinition.apiextensions.k8s.io/tridentorchestrators.tride
nt.netapp.io created
```

3. Trident 네임스페이스가 없으면 제공된 매니페스트를 사용하여 클러스터에 Trident 네임스페이스를 만듭니다.

```
[netapp-user@rhel7 trident-installer]$ oc apply -f deploy/namespace.yaml
namespace/trident created
```

4. 연산자에 대한 'erviceAccount', 'clusterRole', 'ClusterRoleBinding', 'erviceAccount', 'PodSecurityPolicy', 또는 연산자 자체에 대한 'erviceAccount' 등 Trident 운용자 구축에 필요한 리소스를 생성한다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f deploy/bundle.yaml
serviceaccount/trident-operator created
clusterrole.rbac.authorization.k8s.io/trident-operator created
clusterrolebinding.rbac.authorization.k8s.io/trident-operator created
deployment.apps/trident-operator created
podsecuritypolicy.policy/tridentoperatorpods created
```

5. 다음 명령을 사용하여 운영자 배포 후 상태를 확인할 수 있습니다.

```
[netapp-user@rhel7 trident-installer]$ oc get deployment -n trident
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
trident-operator   1/1     1            1           23s
[netapp-user@rhel7 trident-installer]$ oc get pods -n trident
NAME                                READY   STATUS    RESTARTS   AGE
trident-operator-66f48895cc-lzczk   1/1     Running   0          41s
```

6. 운영자가 구축되었으므로 이제 Trident를 설치할 수 있습니다. 이를 위해서는 '트리엔오케스트레이터'를 만들어야 합니다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f
deploy/crds/tridentorchestrator_cr.yaml
tridentorchestrator.trident.netapp.io/trident created
[netapp-user@rhel7 trident-installer]$ oc describe torc trident
Name:         trident
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  trident.netapp.io/v1
Kind:         TridentOrchestrator
Metadata:
  Creation Timestamp:  2021-05-07T17:00:28Z
  Generation:          1
  Managed Fields:
    API Version:  trident.netapp.io/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:spec:
        .:
        f:debug:
        f:namespace:
    Manager:      kubectl-create
    Operation:    Update
    Time:         2021-05-07T17:00:28Z
    API Version:  trident.netapp.io/v1
```

```
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:currentInstallationParams:
          .:
          f:IPv6:
          f:autosupportHostname:
          f:autosupportImage:
          f:autosupportProxy:
          f:autosupportSerialNumber:
          f:debug:
          f:enableNodePrep:
          f:imagePullSecrets:
          f:imageRegistry:
          f:k8sTimeout:
          f:kubeletDir:
          f:logFormat:
          f:silenceAutosupport:
          f:tridentImage:
        f:message:
        f:namespace:
        f:status:
        f:version:
    Manager:         trident-operator
    Operation:       Update
    Time:            2021-05-07T17:00:28Z
  Resource Version:  931421
  Self Link:
/apis/trident.netapp.io/v1/tridentorchestrators/trident
  UID:               8a26a7a6-dde8-4d55-9b66-a7126754d81f
Spec:
  Debug:      true
  Namespace:  trident
Status:
  Current Installation Params:
    IPv6:                       false
    Autosupport Hostname:
    Autosupport Image:          netapp/trident-autosupport:21.01
    Autosupport Proxy:
    Autosupport Serial Number:
    Debug:                      true
    Enable Node Prep:           false
    Image Pull Secrets:
    Image Registry:
    k8sTimeout:           30
```

```
    Kubelet Dir:          /var/lib/kubelet
    Log Format:           text
    Silence Autosupport:  false
    Trident Image:        netapp/trident:22.01.0
  Message:                Trident installed
  Namespace:              trident
  Status:                 Installed
  Version:                v22.01.0
Events:
  Type    Reason      Age   From                        Message
  ----    ------      ----  ----                        -------
  Normal  Installing  80s   trident-operator.netapp.io  Installing
Trident
  Normal  Installed   68s   trident-operator.netapp.io  Trident
installed
```

7. 네임스페이스에서 실행 중인 포드를 확인하거나 tridentctl 바이너리를 사용하여 설치된 버전을 확인하여 Trident가 성공적으로 설치되었는지 확인할 수 있습니다.

```
[netapp-user@rhel7 trident-installer]$ oc get pods -n trident
NAME                                READY   STATUS    RESTARTS   AGE
trident-csi-bb64c6cb4-lmd6h         6/6     Running   0          82s
trident-csi-gn59q                   2/2     Running   0          82s
trident-csi-m4szj                   2/2     Running   0          82s
trident-csi-sb9k9                   2/2     Running   0          82s
trident-operator-66f48895cc-lzczk   1/1     Running   0          2m39s

[netapp-user@rhel7 trident-installer]$ ./tridentctl -n trident version
+----------------+----------------+
| SERVER VERSION | CLIENT VERSION |
+----------------+----------------+
| 22.01.0          | 22.01.0          |
+----------------+----------------+
```

스토리지에 대한 작업자 노드 준비

NFS 를 참조하십시오

대부분의 Kubernetes 배포판에는 Red Hat OpenShift를 포함하여 기본적으로 설치된 NFS 백엔드를 마운트하는 패키지와 유틸리티가 함께 제공됩니다.

그러나 NFSv3의 경우 클라이언트와 서버 간에 동시성을 협상하는 메커니즘이 없습니다. 따라서 서버에서 지원되는 값을 사용하여 수동으로 클라이언트 측 sunrpc 슬롯 테이블 항목의 최대 수를 동기화해야 서버의 창 크기를 줄일 필요 없이 NFS 연결에 대한 최상의 성능을 보장할 수 있습니다.

ONTAP의 경우 지원되는 최대 sunrpc 슬롯 테이블 항목 수는 128개입니다. 즉, ONTAP는 한 번에 128개의 동시 NFS 요청을 지원할 수 있습니다. 그러나 기본적으로 Red Hat CoreOS/Red Hat Enterprise Linux는 연결당 최대 65,536개의 sunrpc 슬롯 테이블 항목을 갖습니다. 이 값은 128로 설정해야 하며, OpenShift에서 Machine Config Operator(MCO)를 사용하여 설정할 수 있습니다.

OpenShift 작업자 노드에서 최대 sunrpc 슬롯 테이블 항목을 수정하려면 다음 단계를 완료하십시오.

1. OCP 웹 콘솔에 로그인하여 Compute(컴퓨팅) > Machine Configs(장비 구성) 로 이동합니다. Create Machine Config 를 클릭합니다. YAML 파일을 복사하여 붙여넣은 다음 생성 을 클릭합니다.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 98-worker-nfs-rpc-slot-tables
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8; base64,b3B0aW9ucyBzdW5ycGMgdGNwX21heF9zbG90X3RhYmxlX2VudHJpZXM9MTI4Cg=
=
          filesystem: root
          mode: 420
          path: /etc/modprobe.d/sunrpc.conf
```

2. MCO를 생성한 후에는 모든 작업자 노드에 구성을 적용하고 하나씩 재부팅해야 합니다. 전체 과정은 약 20-30분 정도 소요됩니다. 'OC Get MCP'를 사용하여 기계 설정이 적용되었는지 확인하고 작업자에 대한 기계 구성 풀이 업데이트되었는지 확인합니다.

```
[netapp-user@rhel7 openshift-deploy]$ oc get mcp
NAME     CONFIG                                     UPDATED   UPDATING
DEGRADED
master   rendered-master-a520ae930e1d135e0dee7168   True      False
False
worker   rendered-worker-de321b36eeba62df41feb7bc   True      False
False
```

iSCSI

iSCSI 프로토콜을 통해 블록 스토리지 볼륨을 매핑할 수 있도록 작업자 노드를 준비하려면 해당 기능을 지원하는 데 필요한 패키지를 설치해야 합니다.

Red Hat OpenShift에서는 MCO(Machine Config Operator)를 배포된 후 클러스터에 적용하여 처리됩니다.

작업자 노드가 iSCSI 서비스를 실행하도록 구성하려면 다음 단계를 수행하십시오.

1. OCP 웹 콘솔에 로그인하여 Compute(컴퓨팅) > Machine Configs(장비 구성) 로 이동합니다. Create Machine Config 를 클릭합니다. YAML 파일을 복사하여 붙여넣은 다음 생성 을 클릭합니다.

   다중 경로를 사용하지 않는 경우:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker
     name: 99-worker-element-iscsi
   spec:
     config:
       ignition:
         version: 3.2.0
       systemd:
         units:
           - name: iscsid.service
             enabled: true
             state: started
     osImageURL: ""
   ```

   다중 경로 사용 시:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 99-worker-ontap-iscsi
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8; base64,ZGVmYXVsdHMgewogICAgICAgIHVzZXJfZnJpZW5kbHlfbmFtZXMgbm8KICAgICA
gICBmaW5kX211bHRpcGF0aHMgbm8KfQoKYmxhY2tsaXN0X2V4Y2VwdGlvbnMgewogICAgICA
gIHByb3BlcnR5ICIoU0NTSV9JREVOVF98SURfV1dOKSIKfQoKYmxhY2tsaXN0IHsKfQoK
          verification: {}
        filesystem: root
        mode: 400
        path: /etc/multipath.conf
    systemd:
      units:
        - name: iscsid.service
          enabled: true
          state: started
        - name: multipathd.service
          enabled: true
          state: started
  osImageURL: ""
```

2. 구성을 생성한 후 작업자 노드에 구성을 적용하고 다시 로드하는 데 약 20~30분이 걸립니다. 'OC Get MCP'를 사용하여 기계 설정이 적용되었는지 확인하고 작업자에 대한 기계 구성 풀이 업데이트되었는지 확인합니다. 작업자 노드에 로그인하여 iscsid 서비스가 실행 중인지 확인할 수도 있습니다(다중 경로를 사용하는 경우 multipathd 서비스가 실행 중인지 확인).

```
[netapp-user@rhel7 openshift-deploy]$ oc get mcp
NAME     CONFIG                                     UPDATED   UPDATING
DEGRADED
master   rendered-master-a520ae930e1d135e0dee7168   True      False
False
worker   rendered-worker-de321b36eeba62df41feb7bc   True      False
False

[netapp-user@rhel7 openshift-deploy]$ ssh core@10.61.181.22 sudo
systemctl status iscsid
● iscsid.service - Open-iSCSI
   Loaded: loaded (/usr/lib/systemd/system/iscsid.service; enabled;
vendor preset: disabled)
   Active: active (running) since Tue 2021-05-26 13:36:22 UTC; 3 min ago
     Docs: man:iscsid(8)
           man:iscsiadm(8)
 Main PID: 1242 (iscsid)
   Status: "Ready to process requests"
    Tasks: 1
   Memory: 4.9M
      CPU: 9ms
   CGroup: /system.slice/iscsid.service
           └─1242 /usr/sbin/iscsid -f

[netapp-user@rhel7 openshift-deploy]$ ssh core@10.61.181.22 sudo
systemctl status multipathd
 ● multipathd.service - Device-Mapper Multipath Device Controller
   Loaded: loaded (/usr/lib/systemd/system/multipathd.service; enabled;
vendor preset: enabled)
   Active: active (running) since Tue 2021-05-26 13:36:22 UTC; 3 min ago
  Main PID: 918 (multipathd)
    Status: "up"
    Tasks: 7
    Memory: 13.7M
    CPU: 57ms
    CGroup: /system.slice/multipathd.service
            └─918 /sbin/multipathd -d -s
```

또한 MachineConfig가 성공적으로 적용되고 서비스가 예상대로 시작되었는지 확인할 수 있는 것은 적절한 플래그를 사용하여 OC debug 명령을 실행하는 것입니다.

스토리지 시스템 백엔드를 생성합니다

Astra Trident Operator 설치를 완료한 후에는 사용 중인 특정 NetApp 스토리지 플랫폼에 대한 백엔드를 구성해야

합니다. Astra Trident의 설정 및 구성을 계속하려면 아래 링크를 따라가십시오.

- "NetApp ONTAP NFS 를 참조하십시오"
- "NetApp ONTAP iSCSI를 참조하십시오"
- "NetApp Element iSCSI 를 참조하십시오"

NetApp ONTAP NFS 구성

NetApp ONTAP 스토리지 시스템과의 Trident 통합을 활성화하려면 스토리지 시스템과의 통신을 지원하는 백엔드를 생성해야 합니다.

1. 다운로드한 설치 아카이브에서 사용할 수 있는 예제 백엔드 파일이 'ample-input' 폴더 계층에 있습니다. NFS를 지원하는 NetApp ONTAP 시스템의 경우 'backend-ontap-nas.json' 파일을 작업 디렉토리에 복사하고 파일을 편집하십시오.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/backends-
samples/ontap-nas/backend-ontap-nas.json ./
[netapp-user@rhel7 trident-installer]$ vi backend-ontap-nas.json
```

2. backendName, managedLIF, dataLIF, svm, 사용자 이름, 및 암호 값을 입력합니다.

```
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "ontap-nas+10.61.181.221",
    "managementLIF": "172.21.224.201",
    "dataLIF": "10.61.181.221",
    "svm": "trident_svm",
    "username": "cluster-admin",
    "password": "password"
}
```

> 사용자 지정 backendName 값을 storageDriverName 과 NFS를 함께 사용하여 쉽게 식별할 수 있도록 하는 데이터 LIF를 함께 정의하는 것이 좋습니다.

3. 이 백엔드 파일을 배치하고 다음 명령을 실행하여 첫 번째 백엔드를 생성합니다.

```
[netapp-user@rhel7 trident-installer]$ ./tridentctl -n trident create
backend -f backend-ontap-nas.json
+------------------------+----------------
+--------------------------------------+--------+---------+
|          NAME          | STORAGE DRIVER |                  UUID
| STATE  | VOLUMES |
+------------------------+----------------
+--------------------------------------+--------+---------+
| ontap-nas+10.61.181.221 | ontap-nas      | be7a619d-c81d-445c-b80c-
5c87a73c5b1e | online |       0 |
+------------------------+----------------
+--------------------------------------+--------+---------+
```

4. 백엔드가 생성되면 다음 번에 스토리지 클래스를 생성해야 합니다. 백엔드와 마찬가지로 샘플 입력 폴더에서 사용 가능한 환경에 대해 편집할 수 있는 샘플 스토리지 클래스 파일이 있습니다. 작업 디렉토리에 복사하고 생성된 백엔드를 반영하기 위해 필요한 편집을 수행합니다.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/storage-class-
samples/storage-class-csi.yaml.templ ./storage-class-basic.yaml
[netapp-user@rhel7 trident-installer]$ vi storage-class-basic.yaml
```

5. 이 파일에 대해 편집해야 하는 유일한 방법은 새로 생성된 백엔드에서 스토리지 드라이버의 이름으로 'backendType' 값을 정의하는 것입니다. 또한 이름 필드 값을 기록해 둡니다. 이 값은 나중에 참조해야 합니다.

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

> 이 파일에 정의된 fschType이라는 선택적 필드가 있습니다. 이 라인은 NFS 백엔드에서 삭제할 수 있습니다.

6. "OC" 명령을 실행하여 스토리지 클래스를 생성합니다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f storage-class-
basic.yaml
storageclass.storage.k8s.io/basic-csi created
```

7. 스토리지 클래스를 생성한 후 첫 번째 영구 볼륨 클레임(PVC)을 생성해야 합니다. 샘플 입력에도 이 작업을

수행하는 데 사용할 수 있는 PVC-BASIC.YAML 파일이 있습니다.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/pvc-samples/pvc-
basic.yaml ./
[netapp-user@rhel7 trident-installer]$ vi pvc-basic.yaml
```

8. 이 파일에 대해 편집해야 하는 유일한 내용은 'torageClassName' 필드가 방금 만든 필드와 일치한다는 것입니다. PVC 정의는 프로비저닝할 작업 부하에 따라 필요에 따라 추가로 사용자 정의할 수 있습니다.

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

9. OC 명령을 실행하여 PVC를 생성한다. 생성 중인 백업 볼륨의 크기에 따라 생성 시간이 다소 걸릴 수 있으므로 완료 시 프로세스를 확인할 수 있습니다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f pvc-basic.yaml
persistentvolumeclaim/basic created

[netapp-user@rhel7 trident-installer]$ oc get pvc
NAME    STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
basic   Bound    pvc-b4370d37-0fa4-4c17-bd86-94f96c94b42d   1Gi
RWO            basic-csi      7s
```

NetApp ONTAP iSCSI 구성

NetApp ONTAP 스토리지 시스템과의 Trident 통합을 활성화하려면 스토리지 시스템과의 통신을 지원하는 백엔드를 생성해야 합니다.

1. 다운로드한 설치 아카이브에서 사용할 수 있는 예제 백엔드 파일이 'ample-input' 폴더 계층에 있습니다. iSCSI를 지원하는 NetApp ONTAP 시스템의 경우 'backend-ontap-san.json' 파일을 작업 디렉토리에 복사하고 파일을 편집합니다.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/backends-
samples/ontap-san/backend-ontap-san.json ./
[netapp-user@rhel7 trident-installer]$ vi backend-ontap-san.json
```

2. 이 파일에서 관리 LIF, dataLIF, svm, 사용자 이름 및 암호 값을 편집합니다.

```
{
  "version": 1,
  "storageDriverName": "ontap-san",
  "managementLIF": "172.21.224.201",
  "dataLIF": "10.61.181.240",
  "svm": "trident_svm",
  "username": "admin",
  "password": "password"
}
```

3. 이 백엔드 파일을 배치하고 다음 명령을 실행하여 첫 번째 백엔드를 생성합니다.

```
[netapp-user@rhel7 trident-installer]$ ./tridentctl -n trident create
backend -f backend-ontap-san.json
+------------------------+----------------
+--------------------------------------+--------+---------+
|          NAME          | STORAGE DRIVER |                  UUID
| STATE  | VOLUMES |
+------------------------+----------------
+--------------------------------------+--------+---------+
| ontapsan_10.61.181.241 | ontap-san      | 6788533c-7fea-4a35-b797-
fb9bb3322b91 | online |       0 |
+------------------------+----------------
+--------------------------------------+--------+---------+
```

4. 백엔드가 생성되면 다음 번에 스토리지 클래스를 생성해야 합니다. 백엔드와 마찬가지로 샘플 입력 폴더에서 사용 가능한 환경에 대해 편집할 수 있는 샘플 스토리지 클래스 파일이 있습니다. 작업 디렉토리에 복사하고 생성된 백엔드를 반영하기 위해 필요한 편집을 수행합니다.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/storage-class-
samples/storage-class-csi.yaml.templ ./storage-class-basic.yaml
[netapp-user@rhel7 trident-installer]$ vi storage-class-basic.yaml
```

5. 이 파일에 대해 편집해야 하는 유일한 방법은 새로 생성된 백엔드에서 스토리지 드라이버의 이름으로 'backendType' 값을 정의하는 것입니다. 또한 이름 필드 값을 기록해 둡니다. 이 값은 나중에 참조해야 합니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: basic-csi
provisioner: csi.trident.netapp.io
parameters:
  backendType: "ontap-san"
```

> 이 파일에 정의된 fschType이라는 선택적 필드가 있습니다. iSCSI 백엔드에서 이 값은 특정 Linux 파일 시스템 유형(XFS, ext4 등)으로 설정하거나, OpenShift가 사용할 파일 시스템을 결정할 수 있도록 삭제할 수 있습니다.

6. "OC" 명령을 실행하여 스토리지 클래스를 생성합니다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f storage-class-basic.yaml
storageclass.storage.k8s.io/basic-csi created
```

7. 스토리지 클래스를 생성한 후 첫 번째 영구 볼륨 클레임(PVC)을 생성해야 합니다. 샘플 입력에도 이 작업을 수행하는 데 사용할 수 있는 PVC-BASIC.YAML 파일이 있습니다.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/pvc-samples/pvc-basic.yaml ./
[netapp-user@rhel7 trident-installer]$ vi pvc-basic.yaml
```

8. 이 파일에 대해 편집해야 하는 유일한 내용은 'torageClassName' 필드가 방금 만든 필드와 일치한다는 것입니다. PVC 정의는 프로비저닝할 작업 부하에 따라 필요에 따라 추가로 사용자 정의할 수 있습니다.

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

9. OC 명령을 실행하여 PVC를 생성한다. 생성 중인 백업 볼륨의 크기에 따라 생성 시간이 다소 걸릴 수 있으므로 완료 시 프로세스를 확인할 수 있습니다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f pvc-basic.yaml
persistentvolumeclaim/basic created

[netapp-user@rhel7 trident-installer]$ oc get pvc
NAME    STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
basic   Bound    pvc-7ceac1ba-0189-43c7-8f98-094719f7956c   1Gi
RWO            basic-csi      3s
```

NetApp Element iSCSI 구성

NetApp Element 스토리지 시스템과의 Trident 통합을 활성화하려면 iSCSI 프로토콜을 사용하여 스토리지 시스템과의 통신을 지원하는 백엔드를 생성해야 합니다.

1. 다운로드한 설치 아카이브에서 사용할 수 있는 예제 백엔드 파일이 'ample-input' 폴더 계층에 있습니다. iSCSI를 지원하는 NetApp Element 시스템의 경우 'backend-solidfire.json' 파일을 작업 디렉토리로 복사하고 파일을 편집합니다.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/backends-
samples/solidfire/backend-solidfire.json ./
[netapp-user@rhel7 trident-installer]$ vi ./backend-solidfire.json
```

   a. 끝점 줄에서 사용자, 암호, MVIP 값을 편집합니다.
   b. VIP 값을 편집합니다.

```
 {
    "version": 1,
    "storageDriverName": "solidfire-san",
    "Endpoint": "https://trident:password@172.21.224.150/json-
rpc/8.0",
    "SVIP": "10.61.180.200:3260",
    "TenantName": "trident",
    "Types": [{"Type": "Bronze", "Qos": {"minIOPS": 1000, "maxIOPS":
2000, "burstIOPS": 4000}},
              {"Type": "Silver", "Qos": {"minIOPS": 4000, "maxIOPS":
6000, "burstIOPS": 8000}},
              {"Type": "Gold", "Qos": {"minIOPS": 6000, "maxIOPS":
8000, "burstIOPS": 10000}}]
}
```

2. 이 백엔드 파일을 배치하고 다음 명령을 실행하여 첫 번째 백엔드를 생성합니다.

```
[netapp-user@rhel7 trident-installer]$ ./tridentctl -n trident create
backend -f backend-solidfire.json
+------------------------+----------------
+--------------------------------------+--------+---------+
|          NAME          | STORAGE DRIVER |                  UUID
| STATE  | VOLUMES |
+------------------------+----------------
+--------------------------------------+--------+---------+
| solidfire_10.61.180.200 | solidfire-san  | b90783ee-e0c9-49af-8d26-
3ea87ce2efdf | online |       0 |
+------------------------+----------------
+--------------------------------------+--------+---------+
```

3. 백엔드가 생성되면 다음 번에 스토리지 클래스를 생성해야 합니다. 백엔드와 마찬가지로 샘플 입력 폴더에서 사용 가능한 환경에 대해 편집할 수 있는 샘플 스토리지 클래스 파일이 있습니다. 작업 디렉토리에 복사하고 생성된 백엔드를 반영하기 위해 필요한 편집을 수행합니다.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/storage-class-
samples/storage-class-csi.yaml.templ ./storage-class-basic.yaml
[netapp-user@rhel7 trident-installer]$ vi storage-class-basic.yaml
```

4. 이 파일에 대해 편집해야 하는 유일한 방법은 새로 생성된 백엔드에서 스토리지 드라이버의 이름으로 'backendType' 값을 정의하는 것입니다. 또한 이름 필드 값을 기록해 둡니다. 이 값은 나중에 참조해야 합니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: basic-csi
provisioner: csi.trident.netapp.io
parameters:
  backendType: "solidfire-san"
```

> 이 파일에 정의된 fschType이라는 선택적 필드가 있습니다. iSCSI 백엔드에서 이 값은 특정 Linux 파일 시스템 유형(XFS, ext4 등)으로 설정하거나 OpenShift가 사용할 파일 시스템을 결정할 수 있도록 삭제할 수 있습니다.

5. "OC" 명령을 실행하여 스토리지 클래스를 생성합니다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f storage-class-
basic.yaml
storageclass.storage.k8s.io/basic-csi created
```

6. 스토리지 클래스를 생성한 후 첫 번째 영구 볼륨 클레임(PVC)을 생성해야 합니다. 샘플 입력에도 이 작업을 수행하는 데 사용할 수 있는 PVC-BASIC.YAML 파일이 있습니다.

```
[netapp-user@rhel7 trident-installer]$ cp sample-input/pvc-samples/pvc-
basic.yaml ./
[netapp-user@rhel7 trident-installer]$ vi pvc-basic.yaml
```

7. 이 파일에 대해 편집해야 하는 유일한 내용은 'torageClassName' 필드가 방금 만든 필드와 일치한다는 것입니다. PVC 정의는 프로비저닝할 작업 부하에 따라 필요에 따라 추가로 사용자 정의할 수 있습니다.

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

8. OC 명령을 실행하여 PVC를 생성한다. 생성 중인 백업 볼륨의 크기에 따라 생성 시간이 다소 걸릴 수 있으므로 완료 시 프로세스를 확인할 수 있습니다.

```
[netapp-user@rhel7 trident-installer]$ oc create -f pvc-basic.yaml
persistentvolumeclaim/basic created

[netapp-user@rhel7 trident-installer]$ oc get pvc
NAME    STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
basic   Bound    pvc-3445b5cc-df24-453d-a1e6-b484e874349d   1Gi
RWO            basic-csi      5s
```