NetApp Console 자동화 허브

NetApp Automation

NetApp
November 18, 2025

목차

NetApp Console 자동화 허브

NetApp Console 자동화 허브 개요

NetApp Console 자동화 허브는 NetApp 고객, 파트너 및 직원이 사용할 수 있는 자동화 솔루션 모음입니다. 자동화 허브에는 여러 가지 기능과 이점이 있습니다.

자동화를 위한 단일 위치

당신은 접근할 수 있습니다 "NetApp Console 자동화 허브" 콘솔 웹 사용자 인터페이스를 통해. 이를 통해 NetApp 제품과 서비스의 자동화와 운영을 개선하는 데 필요한 스크립트, 플레이북, 모듈을 단일 위치에서 관리할 수 있습니다.

솔루션은 **NetApp**에서 만들고 테스트합니다

모든 자동화 솔루션과 스크립트는 NetApp에 의해 생성 및 테스트되었습니다. 각 솔루션은 특정 고객 사용 사례 또는 요청을 대상으로 합니다. 대부분 NetApp 파일 및 데이터 서비스와의 통합에 중점을 둡니다.

문서화

각 자동화 솔루션에는 시작하는 데 도움이 되는 관련 문서가 포함되어 있습니다. 솔루션은 콘솔 웹 인터페이스를 통해 접근하지만 모든 문서는 이 사이트에서 확인할 수 있습니다. 문서는 NetApp 제품과 클라우드 서비스를 기준으로 구성됩니다.

미래를 위한 견고한 기반

NetApp 고객이 데이터 센터와 클라우드 환경의 자동화를 개선하고 간소화하도록 지원하는 데 전념하고 있습니다. 고객 요구 사항, 기술 변화, 지속적인 제품 통합에 대응하기 위해 콘솔 자동화 허브를 지속적으로 개선할 예정입니다.

NetApp은 파트너 의견을 환영합니다

NetApp 고객 경험 사무소(CXO) 자동화 팀이 연락을 드리면 됩니다. 피드백, 문제 또는 기능 요청이 있는 경우 cXO 자동화 팀 으로 이메일을 보내주시기 바랍니다.

Amazon FSx for NetApp ONTAP

Amazon FSx for NetApp ONTAP - 클라우드로의 버스트

이 자동화 솔루션을 사용하면 볼륨과 관련 FlexCache 를 사용하여 Amazon FSx for NetApp ONTAP 프로비저닝할 수 있습니다.

> Amazon FSx for NetApp ONTAP 관리 기능은 *FSx for ONTAP*라고도 합니다.

살펴봅니다

이 솔루션과 함께 제공되는 자동화 코드는 개략적으로 다음과 같은 작업을 수행합니다.

- 대상 FSx for ONTAP 파일 시스템을 프로비저닝합니다
- 파일 시스템에 대한 SVM(Storage Virtual Machine)을 프로비저닝합니다
- 소스 시스템과 타겟 시스템 간에 클러스터 피어링 관계를 생성합니다
- 소스 시스템과 FlexCache용 타겟 시스템 간에 SVM 피어링 관계를 생성합니다

- 선택적으로 FSx for ONTAP를 사용하여 FlexVol 볼륨을 생성합니다
- 온프레미스 스토리지를 가리키는 FlexCache 볼륨을 FSx for ONTAP에서 생성합니다

자동화는 아래 설명과 같이 Linux 가상 머신에 설치해야 하는 Docker 및 Docker Compose를 기반으로 합니다.

시작하기 전에

프로비저닝 및 구성을 완료하려면 다음이 있어야 합니다.

- 다운로드가 필요합니다 "Amazon FSx for NetApp ONTAP - 클라우드로의 버스트" NetApp Console 웹 UI를 통한 자동화 솔루션. 솔루션은 파일로 패키징됩니다. `AWS_FSxN_BTC.zip`.
- 소스 시스템과 대상 시스템 간의 네트워크 연결
- 다음과 같은 특성을 가진 Linux VM:
  - 데비안 기반 리눅스 배포판
  - FSx for ONTAP 프로비저닝에 사용되는 동일한 VPC 서브셋에 구축됩니다
- AWS 계정.

1단계**: Docker** 설치 및 구성

데비안 기반 Linux 가상 머신에 Docker를 설치하고 구성합니다.

단계

1. 환경을 준비합니다.

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl gnupg-
agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
```

2. Docker를 설치하고 설치를 확인합니다.

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker --version
```

3. 필요한 Linux 그룹을 관련 사용자와 추가합니다.

   먼저 Linux 시스템에 * docker * 그룹이 있는지 확인하십시오. 그렇지 않으면 그룹을 만들고 사용자를 추가합니다. 기본적으로 현재 셸 사용자가 그룹에 추가됩니다.

```
sudo groupadd docker
sudo usermod -aG docker $(whoami)
```

4. 새 그룹 및 사용자 정의를 활성화합니다

   사용자와 함께 새 그룹을 만든 경우 정의를 활성화해야 합니다. 이렇게 하려면 Linux에서 로그아웃했다가 다시 로그인하거나 다음 명령을 실행할 수 있습니다.

```
newgrp docker
```

2단계**: Docker** 구성 설치

Docker Compose를 데비안 기반 Linux 가상 머신에 설치합니다.

단계

1. Docker Compose를 설치합니다.

```
sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

2. 설치가 성공적으로 완료되었는지 확인합니다.

```
docker-compose --version
```

3단계**: Docker** 이미지 준비

자동화 솔루션과 함께 제공된 Docker 이미지를 추출하여 로드해야 합니다.

단계

1. 자동화 코드가 실행될 가상 머신에 솔루션 파일을 `AWS_FSxN_BTC.zip` 복사합니다.

```
scp -i ~/<private-key.pem> -r AWS_FSxN_BTC.zip user@<IP_ADDRESS_OF_VM>
```

   input 매개 변수는 `private-key.pem` AWS 가상 머신 인증(EC2 인스턴스)에 사용되는 개인 키 파일입니다.

2. 솔루션 파일이 있는 올바른 폴더로 이동하여 파일의 압축을 풉니다.

```
unzip AWS_FSxN_BTC.zip
```

3. 압축 풀기 작업으로 생성된 새 폴더로 `AWS_FSxN_BTC` 이동하여 파일을 나열합니다. 파일을 볼 수 `aws_fsxn_flexcache_image_latest.tar.gz` 있습니다.

```
ls -la
```

4. Docker 이미지 파일을 로드합니다. 로드 작업은 일반적으로 몇 초 내에 완료됩니다.

```
docker load -i aws_fsxn_flexcache_image_latest.tar.gz
```

5. Docker 이미지가 로드되었는지 확인합니다.

```
docker images
```

Docker `aws_fsxn_flexcache_image` 이미지와 태그가 `latest`표시됩니다.

```
   REPOSITORY                TAG      IMAGE ID       CREATED       SIZE
aws_fsxn_flexcahce_image  latest   ay98y7853769  2 weeks ago  1.19GB
```

4단계**: AWS** 자격 증명에 대한 환경 파일을 생성합니다

액세스 및 비밀 키를 사용하여 인증할 로컬 변수 파일을 만들어야 합니다. 그런 다음 파일을 `.env` 파일에 추가합니다.

단계

1. 다음 위치에 파일을 만듭니다 `awsauth.env`.

   `path/to/env-file/awsauth.env`

2. 파일에 다음 내용을 추가합니다.

```
access_key=<>
secret_key=<>
```

   형식은 * 와(과 `value`) 사이의 공백을 제외하고 위에 표시된 것과 정확히 `key` 일치해야 합니다.

3. 변수를 사용하여 절대 파일 경로를 파일에 `AWS_CREDS` 추가합니다 `.env`. 예를 들면 다음과 같습니다.

   `AWS_CREDS=path/to/env-file/awsauth.env`

5단계**:** 외부 볼륨을 만듭니다

Terraform 상태 파일과 기타 중요한 파일이 영구적으로 유지되도록 하려면 외부 볼륨이 필요합니다. Terraform에서 워크플로우 및 배포를 실행하려면 이러한 파일을 사용할 수 있어야 합니다.

단계

1. Docker Compose 외부에서 외부 볼륨을 생성합니다.

   명령을 실행하기 전에 볼륨 이름(마지막 매개 변수)을 적절한 값으로 업데이트해야 합니다.

   ```
   docker volume create aws_fsxn_volume
   ```

2. 다음 명령을 사용하여 외부 볼륨의 경로를 환경 파일에 추가합니다 `.env`.

   `PERSISTENT_VOL=path/to/external/volume:/volume_name`

   기존 파일 내용과 콜론 서식을 유지해야 합니다. 예를 들면 다음과 같습니다.

   ```
   PERSISTENT_VOL=aws_fsxn_volume:/aws_fsxn_flexcache
   ```

   대신 다음과 같은 명령을 사용하여 NFS 공유를 외부 볼륨으로 추가할 수 있습니다.

   `PERSISTENT_VOL=nfs/mnt/document:/aws_fsx_flexcache`

3. Terraform 변수를 업데이트합니다.
   a. 폴더로 `aws_fsxn_variables`이동합니다.
   b. 다음 두 파일이 있는지 `terraform.tfvars` 확인합니다. 및 `variables.tf`.
   c. 사용자 환경에 필요한 대로 의 값을 `terraform.tfvars` 업데이트합니다.

      자세한 내용은 을 "Terraform 리소스: AWS_FSX_ONTAP_FILE_SYSTEM" 참조하십시오.

6단계: NetApp ONTAP 관리 및 FlexCache Amazon FSx for NetApp ONTAP 프로비저닝

NetApp ONTAP 관리 및 FlexCache 를 Amazon FSx for NetApp ONTAP 프로비저닝할 수 있습니다.

단계

1. 폴더 루트(AWS_FSXN_BTC)로 이동하여 provisioning 명령을 실행합니다.

   ```
   docker-compose -f docker-compose-provision.yml up
   ```

   이 명령은 두 개의 컨테이너를 만듭니다. 첫 번째 컨테이너에서 FSx for ONTAP를 구축하고 두 번째 컨테이너에서 클러스터 피어링, SVM 피어링, 타겟 볼륨 및 FlexCache를 생성합니다.

2. 프로비저닝 프로세스를 모니터링합니다.

   ```
   docker-compose -f docker-compose-provision.yml logs -f
   ```

   이 명령은 실시간으로 출력을 제공하지만 파일을 통해 로그를 캡처하도록 `deployment.log` 구성되었습니다.

파일을 편집하고 변수를 업데이트하여 `DEPLOYMENT_LOGS` 이러한 로그 파일의 이름을 변경할 수 `.env` 있습니다.

7단계: NetApp ONTAP 관리 및 FlexCache Amazon FSx for NetApp ONTAP 삭제

선택적으로 Amazon FSx for NetApp ONTAP 관리 및 FlexCache 삭제하고 제거할 수 있습니다.

1. `terraform.tfvars`파일의 변수를 `flexcache_operation` "destroy"로 설정합니다.
2. 루트 폴더(AWS_FSXN_BTC)로 이동하여 다음 명령을 실행합니다.

```
docker-compose -f docker-compose-destroy.yml up
```

이 명령은 두 개의 컨테이너를 만듭니다. 첫 번째 컨테이너에서 FlexCache을 삭제하고 두 번째 컨테이너에서 FSx for ONTAP을 삭제합니다.

3. 프로비저닝 프로세스를 모니터링합니다.

```
docker-compose -f docker-compose-destroy.yml logs -f
```

Amazon FSx for NetApp ONTAP 관리 - 재해 복구

Amazon FSx for NetApp ONTAP 관리를 사용하여 소스 시스템의 재해 복구 백업을 수행하는 데 이 자동화 솔루션을 사용할 수 있습니다.

> Amazon FSx for NetApp ONTAP 관리 기능은 *FSx for ONTAP*라고도 합니다.

살펴봅니다

이 솔루션과 함께 제공되는 자동화 코드는 개략적으로 다음과 같은 작업을 수행합니다.

- 대상 FSx for ONTAP 파일 시스템을 프로비저닝합니다
- 파일 시스템에 대한 SVM(Storage Virtual Machine)을 프로비저닝합니다
- 소스 시스템과 타겟 시스템 간에 클러스터 피어링 관계를 생성합니다
- 소스 시스템과 SnapMirror용 타겟 시스템 간에 SVM 피어링 관계를 생성합니다
- 대상 볼륨을 생성합니다
- 소스 볼륨 및 타겟 볼륨 간에 SnapMirror 관계를 생성합니다
- 소스 볼륨과 타겟 볼륨 간에 SnapMirror 전송을 시작한다

자동화는 아래 설명과 같이 Linux 가상 머신에 설치해야 하는 Docker 및 Docker Compose를 기반으로 합니다.

시작하기 전에

프로비저닝 및 구성을 완료하려면 다음이 있어야 합니다.

- 다운로드가 필요합니다 "Amazon FSx for NetApp ONTAP 관리 - 재해 복구" NetApp Console 웹 UI를 통한

자동화 솔루션. 솔루션은 다음과 같이 패키지됩니다. `FSxN_DR.zip`. 이 zip에는 다음이 포함되어 있습니다. `AWS_FSxN_Bck_Prov.zip` 이 문서에 설명된 솔루션을 배포하는 데 사용할 파일입니다.

- 소스 시스템과 대상 시스템 간의 네트워크 연결
- 다음과 같은 특성을 가진 Linux VM:
  - 데비안 기반 리눅스 배포판
  - FSx for ONTAP 프로비저닝에 사용되는 동일한 VPC 서브셋에 구축됩니다
- AWS 계정.

1단계**: Docker** 설치 및 구성

데비안 기반 Linux 가상 머신에 Docker를 설치하고 구성합니다.

단계

1. 환경을 준비합니다.

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl gnupg-
agent softwareproperties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
```

2. Docker를 설치하고 설치를 확인합니다.

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker --version
```

3. 필요한 Linux 그룹을 관련 사용자와 추가합니다.

   먼저 Linux 시스템에 * docker * 그룹이 있는지 확인하십시오. 그룹이 없는 경우 그룹을 만들고 사용자를 추가합니다. 기본적으로 현재 셸 사용자가 그룹에 추가됩니다.

```
sudo groupadd docker
sudo usermod -aG docker $(whoami)
```

4. 새 그룹 및 사용자 정의를 활성화합니다

   사용자와 함께 새 그룹을 만든 경우 정의를 활성화해야 합니다. 이렇게 하려면 Linux에서 로그아웃했다가 다시 로그인하거나 다음 명령을 실행할 수 있습니다.

```
newgrp docker
```

2단계: Docker 구성 설치

Docker Compose를 데비안 기반 Linux 가상 머신에 설치합니다.

단계

1. Docker Compose를 설치합니다.

```
sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

2. 설치가 성공적으로 완료되었는지 확인합니다.

```
docker-compose --version
```

3단계: Docker 이미지 준비

자동화 솔루션과 함께 제공된 Docker 이미지를 추출하여 로드해야 합니다.

단계

1. 자동화 코드가 실행될 가상 머신에 솔루션 파일을 `AWS_FSxN_Bck_Prov.zip` 복사합니다.

```
scp -i ~/<private-key.pem> -r AWS_FSxN_Bck_Prov.zip
user@<IP_ADDRESS_OF_VM>
```

input 매개 변수는 `private-key.pem` AWS 가상 머신 인증(EC2 인스턴스)에 사용되는 개인 키 파일입니다.

2. 솔루션 파일이 있는 올바른 폴더로 이동하여 파일의 압축을 풉니다.

```
unzip AWS_FSxN_Bck_Prov.zip
```

3. 압축 풀기 작업으로 생성된 새 폴더로 `AWS_FSxN_Bck_Prov` 이동하여 파일을 나열합니다. 파일을 볼 수 `aws_fsxn_bck_image_latest.tar.gz` 있습니다.

```
ls -la
```

4. Docker 이미지 파일을 로드합니다. 로드 작업은 일반적으로 몇 초 내에 완료됩니다.

```
docker load -i aws_fsxn_bck_image_latest.tar.gz
```

5. Docker 이미지가 로드되었는지 확인합니다.

```
docker images
```

Docker `aws_fsxn_bck_image` 이미지와 태그가 `latest`표시됩니다.

```
   REPOSITORY        TAG     IMAGE ID      CREATED      SIZE
aws_fsxn_bck_image  latest  da87d4974306  2 weeks ago  1.19GB
```

4단계**: AWS** 자격 증명에 대한 환경 파일을 생성합니다

액세스 및 비밀 키를 사용하여 인증할 로컬 변수 파일을 만들어야 합니다. 그런 다음 파일을 `.env` 파일에 추가합니다.

단계

1. 다음 위치에 파일을 만듭니다 `awsauth.env`.

   `path/to/env-file/awsauth.env`

2. 파일에 다음 내용을 추가합니다.

```
access_key=<>
secret_key=<>
```

   형식은 * 와(과 `value`) 사이의 공백을 제외하고 위에 표시된 것과 정확히 `key` 일치해야 합니다.

3. 변수를 사용하여 절대 파일 경로를 파일에 `AWS_CREDS` 추가합니다 `.env`. 예를 들면 다음과 같습니다.

   `AWS_CREDS=path/to/env-file/awsauth.env`

5단계**:** 외부 볼륨을 만듭니다

Terraform 상태 파일과 기타 중요한 파일이 영구적으로 유지되도록 하려면 외부 볼륨이 필요합니다. Terraform에서 워크플로우 및 배포를 실행하려면 이러한 파일을 사용할 수 있어야 합니다.

단계

1. Docker Compose 외부에서 외부 볼륨을 생성합니다.

   명령을 실행하기 전에 볼륨 이름(마지막 매개 변수)을 적절한 값으로 업데이트해야 합니다.

```
docker volume create aws_fsxn_volume
```

2. 다음 명령을 사용하여 외부 볼륨의 경로를 환경 파일에 추가합니다 `.env`.

   `PERSISTENT_VOL=path/to/external/volume:/volume_name`

   기존 파일 내용과 콜론 서식을 유지해야 합니다. 예를 들면 다음과 같습니다.

   ```
   PERSISTENT_VOL=aws_fsxn_volume:/aws_fsxn_bck
   ```

   대신 다음과 같은 명령을 사용하여 NFS 공유를 외부 볼륨으로 추가할 수 있습니다.

   `PERSISTENT_VOL=nfs/mnt/document:/aws_fsx_bck`

3. Terraform 변수를 업데이트합니다.
   a. 폴더로 `aws_fsxn_variables`이동합니다.
   b. 다음 두 파일이 있는지 `terraform.tfvars` 확인합니다. 및 `variables.tf`.
   c. 사용자 환경에 필요한 대로 의 값을 `terraform.tfvars` 업데이트합니다.

      자세한 내용은 을 "Terraform 리소스: AWS_FSX_ONTAP_FILE_SYSTEM" 참조하십시오.

6단계: 백업 솔루션을 배포합니다

재해 복구 백업 솔루션을 배포하고 프로비저닝할 수 있습니다.

단계

1. 루트 폴더(AWS_FSxN_Bck_Prov)로 이동하여 provisioning 명령을 실행합니다.

   ```
   docker-compose up -d
   ```

   이 명령은 세 개의 컨테이너를 만듭니다. 첫 번째 컨테이너에서 FSx for ONTAP을 구축합니다. 두 번째 컨테이너는 클러스터 피어링, SVM 피어링 및 대상 볼륨을 생성합니다. 세 번째 컨테이너에서 SnapMirror 관계를 생성하고 SnapMirror 전송을 시작합니다.

2. 프로비저닝 프로세스를 모니터링합니다.

   ```
   docker-compose logs -f
   ```

   이 명령은 실시간으로 출력을 제공하지만 파일을 통해 로그를 캡처하도록 `deployment.log` 구성되었습니다. 파일을 편집하고 변수를 업데이트하여 `DEPLOYMENT_LOGS` 이러한 로그 파일의 이름을 변경할 수 `.env` 있습니다.

Azure NetApp Files

Azure NetApp Files를 사용하여 Oracle을 설치합니다

이 자동화 솔루션을 사용하여 Azure NetApp Files 볼륨을 프로비저닝하고 사용 가능한 가상 머신에 Oracle을 설치할 수 있습니다. 그런 다음 Oracle은 이 볼륨을 데이터 스토리지에 사용합니다.

살펴봅니다

이 솔루션과 함께 제공되는 자동화 코드는 개략적으로 다음과 같은 작업을 수행합니다.

- Azure에서 NetApp 계정을 설정합니다
- Azure에서 스토리지 용량 풀 설정
- 정의에 따라 Azure NetApp Files 볼륨을 프로비저닝합니다
- 마운트 지점을 생성합니다
- Azure NetApp Files 볼륨을 마운트 지점에 마운트합니다
- Linux 서버에 Oracle을 설치합니다
- 수신기 및 데이터베이스를 만듭니다
- 플러그형 데이터베이스(PDB) 만들기
- Listener 및 Oracle 인스턴스를 시작합니다
- 스냅샷을 생성하도록 유틸리티를 설치하고 구성합니다 `azacsnap`

시작하기 전에

설치를 완료하려면 다음이 필요합니다.

- 다운로드가 필요합니다 "Azure NetApp Files를 사용하는 Oracle" NetApp Console 웹 UI를 통한 자동화 솔루션. 솔루션은 파일로 패키징됩니다. `na_oracle19c_deploy-master.zip`.
- 다음과 같은 특성을 가진 Linux VM:
  - RHEL 8(Standard_D8s_v3-RHEL-8)
  - Azure NetApp Files 프로비저닝에 사용되는 것과 동일한 Azure 가상 네트워크에 배포됩니다
- Azure 계정

자동화 솔루션은 이미지로 제공되며 Docker 및 Docker Compose를 사용하여 실행됩니다. 아래 설명과 같이 Linux 가상 머신에 이 두 가지를 모두 설치해야 합니다.

또한 명령을 사용하여 RedHat에 VM을 등록해야 `sudo subscription-manager register` 합니다. 이 명령을 실행하면 계정 자격 증명을 입력하라는 메시지가 표시됩니다. 필요한 경우 https://developers.redhat.com/.에서 계정을 만들 수 있습니다

1단계: Docker 설치 및 구성

RHEL 8 Linux 가상 머신에 Docker를 설치하고 구성합니다.

단계

1. 다음 명령을 사용하여 Docker 소프트웨어를 설치합니다.

```
dnf config-manager --add
-repo=https://download.docker.com/linux/centos/docker-ce.repo
dnf install docker-ce --nobest -y
```

2. Docker를 시작하고 버전을 표시하여 설치가 성공적으로 완료되었는지 확인합니다.

```
systemctl start docker
systemctl enable docker
docker --version
```

3. 필요한 Linux 그룹을 관련 사용자와 추가합니다.

   먼저 Linux 시스템에 * docker * 그룹이 있는지 확인하십시오. 그렇지 않으면 그룹을 만들고 사용자를 추가합니다. 기본적으로 현재 셸 사용자가 그룹에 추가됩니다.

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

4. 새 그룹 및 사용자 정의를 활성화합니다

   사용자와 함께 새 그룹을 만든 경우 정의를 활성화해야 합니다. 이렇게 하려면 Linux에서 로그아웃했다가 다시 로그인하거나 다음 명령을 실행할 수 있습니다.

```
newgrp docker
```

2단계**: Docker Compose** 및 **NFS** 유틸리티 설치

Docker Compose를 NFS 유틸리티 패키지와 함께 설치 및 구성합니다.

단계

1. Docker를 설치하고 버전을 표시하여 설치가 성공적으로 완료되었는지 확인합니다.

```
dnf install curl -y
curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

2. NFS 유틸리티 패키지를 설치합니다.

```
sudo yum install nfs-utils
```

3단계: Oracle 설치 파일을 다운로드합니다

필요한 Oracle 설치 및 패치 파일과 `azacsnap` 유틸리티를 다운로드합니다.

단계

1. 필요한 경우 Oracle 계정에 로그인합니다.
2. 다음 파일을 다운로드합니다.

파일	설명
LINUX.X64_193000_db_home.zip	19.3 기본 장착 공구
p31281355_190000_Linux-x86-64.zip	19.8RU 패치
p6880880_190000_Linux-x86-64.zip	OPatch 버전 12.2.0.1.23
azacsnap_installer_v5.0.ru n 을 참조하십시오	azacsnap 설치 프로그램

3. 모든 설치 파일을 폴더에 `/tmp/archive`넣습니다.
4. 데이터베이스 서버의 모든 사용자가 폴더에 대한 모든 액세스 권한(읽기, 쓰기, 실행)을 갖는지 확인합니다. `/tmp/archive`

4단계: Docker 이미지 준비

자동화 솔루션과 함께 제공된 Docker 이미지를 추출하여 로드해야 합니다.

단계

1. 자동화 코드가 실행될 가상 머신에 솔루션 파일을 `na_oracle19c_deploy-master.zip` 복사합니다.

```
scp -i ~/<private-key.pem> -r na_oracle19c_deploy-master.zip
user@<IP_ADDRESS_OF_VM>
```

   입력 매개 `private-key.pem` 변수는 Azure 가상 머신 인증에 사용되는 개인 키 파일입니다.

2. 솔루션 파일이 있는 올바른 폴더로 이동하여 파일의 압축을 풉니다.

```
unzip na_oracle19c_deploy-master.zip
```

3. 압축 풀기 작업으로 생성된 새 폴더로 `na_oracle19c_deploy-master` 이동하여 파일을 나열합니다. 파일을 볼 수 `ora_anf_bck_image.tar` 있습니다.

```
ls -lt
```

4. Docker 이미지 파일을 로드합니다. 로드 작업은 일반적으로 몇 초 내에 완료됩니다.

```
docker load -i ora_anf_bck_image.tar
```

5. Docker 이미지가 로드되었는지 확인합니다.

```
docker images
```

Docker `ora_anf_bck_image` 이미지와 태그가 `latest`표시됩니다.

```
   REPOSITORY          TAG       IMAGE ID      CREATED      SIZE
ora_anf_bck_image    latest   ay98y7853769   1 week ago   2.58GB
```

5단계**:** 외부 볼륨을 만듭니다

Terraform 상태 파일과 기타 중요한 파일이 영구적으로 유지되도록 하려면 외부 볼륨이 필요합니다. Terraform에서 워크플로우 및 배포를 실행하려면 이러한 파일을 사용할 수 있어야 합니다.

단계

1. Docker Compose 외부에서 외부 볼륨을 생성합니다.

   명령을 실행하기 전에 볼륨 이름을 업데이트하십시오.

```
docker volume create <VOLUME_NAME>
```

2. 다음 명령을 사용하여 외부 볼륨의 경로를 환경 파일에 추가합니다 `.env`.

   `PERSISTENT_VOL=path/to/external/volume:/ora_anf_prov`..

   기존 파일 내용과 콜론 서식을 유지해야 합니다. 예를 들면 다음과 같습니다.

```
PERSISTENT_VOL= ora_anf _volume:/ora_anf_prov
```

3. Terraform 변수를 업데이트합니다.
   a. 폴더로 `ora_anf_variables`이동합니다.
   b. 다음 두 파일이 있는지 `terraform.tfvars` 확인합니다. 및 `variables.tf`.
   c. 사용자 환경에 필요한 대로 의 값을 `terraform.tfvars` 업데이트합니다.

6단계**: Oracle**을 설치합니다

이제 Oracle을 프로비저닝하고 설치할 수 있습니다.

단계

1. 다음 명령 시퀀스를 사용하여 Oracle을 설치합니다.

```
docker-compose up terraform_ora_anf
bash /ora_anf_variables/setup.sh
docker-compose up linux_config
bash /ora_anf_variables/permissions.sh
docker-compose up oracle_install
```

2. Bash 변수를 다시 로드하고 의 값을 표시하여 `ORACLE_HOME` 확인합니다.
   a. `cd /home/oracle`
   b. `source .bash_profile`
   c. `echo $ORACLE_HOME`
3. Oracle에 로그인할 수 있어야 합니다.

```
sudo su oracle
```

7단계**: Oracle** 설치를 검증합니다

Oracle 설치가 성공적으로 완료되었는지 확인해야 합니다.

단계

1. Linux Oracle 서버에 로그인하고 Oracle 프로세스 목록을 표시합니다. 이렇게 하면 설치가 예상대로 완료되었고 Oracle 데이터베이스가 실행 중임을 확인할 수 있습니다.

```
ps -ef | grep ora
```

2. 데이터베이스에 로그인하여 데이터베이스 구성을 검사하고 PDB가 제대로 생성되었는지 확인합니다.

```
sqlplus / as sysdba
```

   다음과 유사한 출력이 표시됩니다.

```
SQL*Plus: Release 19.0.0.0.0 - Production on Thu May 6 12:52:51 2021
Version 19.8.0.0.0

Copyright (c) 1982, 2019, Oracle. All rights reserved.

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.8.0.0.0
```

3. 몇 가지 간단한 SQL 명령을 실행하여 데이터베이스를 사용할 수 있는지 확인합니다.

```
select name, log_mode from v$database;
show pdbs.
```

단계 **8: azacsnap** 유틸리티를 설치하고 스냅샷 백업을 수행합니다

스냅샷 백업을 수행하려면 유틸리티를 설치하고 실행해야 `azacsnap` 합니다.

단계

1. 컨테이너를 설치합니다.

```
docker-compose up azacsnap_install
```

2. 스냅샷 사용자 계정으로 전환합니다.

```
su - azacsnap
execute /tmp/archive/ora_wallet.sh
```

3. 스토리지 백업 세부 정보 파일을 구성합니다. 그러면 구성 파일이 `azacsnap.json` 생성됩니다.

```
cd /home/azacsnap/bin/
azacsnap -c configure --configuration new
```

4. 스냅샷 백업을 수행합니다.

```
azacsnap -c backup --other data --prefix ora_test --retention=1
```

9단계: 필요에 따라 온프레미스 PDB를 클라우드로 마이그레이션합니다

필요에 따라 온프레미스 PDB를 클라우드로 마이그레이션할 수 있습니다.

단계

1. 환경에 필요한 대로 파일의 변수를 `tfvars` 설정합니다.
2. PDB를 마이그레이션합니다.

```
docker-compose -f docker-compose-relocate.yml up
```

AWS용 Cloud Volumes ONTAP

AWS용 Cloud Volumes ONTAP - 클라우드 사용 급증

이 문서에서는 NetApp 고객이 NetApp Console 자동화 허브에서 사용할 수 있는 NetApp Cloud Volumes ONTAP for AWS 자동화 솔루션을 지원합니다.

AWS 환경을 위한 Cloud Volumes ONTAP 자동화 솔루션은 Terraform을 사용하여 Cloud Volumes ONTAP for AWS 환경을 컨테이너화된 구축을 자동화하므로 수동 개입 없이 AWS용 Cloud Volumes ONTAP를 빠르게 구축할 수 있습니다.

시작하기 전에

- 다운로드해야 합니다 "Cloud Volumes ONTAP AWS - 클라우드 사용 급증" 콘솔 웹 UI를 통한 자동화 솔루션. 솔루션은 다음과 같이 패키지됩니다. `cvo_aws_flexcache.zip`.
- Cloud Volumes ONTAP와 동일한 네트워크에 Linux VM을 설치해야 합니다.
- Linux VM을 설치한 후에는 이 솔루션의 단계에 따라 필요한 종속성을 설치해야 합니다.

1단계: Docker 및 Docker Compose를 설치합니다

Docker를 설치합니다

다음 단계에서는 Ubuntu 20.04 Debian Linux 배포 소프트웨어를 예로 사용합니다. 실행하는 명령은 사용 중인 Linux 배포 소프트웨어에 따라 다릅니다. 해당 구성에 대한 자세한 내용은 해당 Linux 배포 소프트웨어 설명서를 참조하십시오.

단계

1. 다음 명령을 실행하여 Docker를 `sudo` 설치합니다.

```
sudo apt-get update
sudo apt-get install apt-transport-https cacertificates curl gnupg-agent
software-properties-common curl -fsSL
https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add -
sudo add-apt-repository “deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable”
sudo apt-get update
sudo apt-get install dockerce docker-ce-cli containerd.io
```

2. 설치를 확인합니다.

```
docker –version
```

3. Linux 시스템에 "docker"라는 그룹이 생성되었는지 확인합니다. 필요한 경우 그룹을 생성합니다.

```
sudo groupadd docker
```

4. Docker에 액세스해야 하는 사용자를 그룹에 추가합니다.

```
sudo usermod -aG docker $(whoami)
```

5. 변경 사항은 로그아웃한 후 터미널에 다시 로그인하면 적용됩니다. 또는 변경 사항을 즉시 적용할 수 있습니다.

```
newgrp docker
```

Docker Compose를 설치합니다

단계

1. 다음 명령을 실행하여 Docker Compose를 `sudo` 설치합니다.

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

2. 설치를 확인합니다.

```
docker-compose -version
```

2단계: Docker 이미지 준비

단계

1. `cvo_aws_flexcache.zip`Cloud Volumes ONTAP를 배포하는 데 사용할 Linux VM에 폴더를 복사합니다.

```
scp -i ~/<private-key>.pem -r cvo_aws_flexcache.zip
<awsuser>@<IP_ADDRESS_OF_VM>:<LOCATION_TO_BE_COPIED>
```

   - `private-key.pem` 은(는) 암호 없이 로그인할 수 있는 개인 키 파일입니다.
   - `awsuser` VM 사용자 이름입니다.
   - `IP_ADDRESS_OF_VM` VM IP 주소입니다.
   - `LOCATION_TO_BE_COPIED` 폴더가 복사될 위치입니다.

2. 폴더 압축을 `cvo_aws_flexcache.zip` 풉니다. 현재 디렉터리나 사용자 지정 위치에서 폴더의 압축을 풀 수 있습니다.

   현재 디렉터리에서 폴더를 추출하려면 다음을 실행합니다.

```
unzip cvo_aws_flexcache.zip
```

   사용자 지정 위치에서 폴더를 추출하려면 다음을 실행합니다.

```
unzip cvo_aws_flexcache.zip -d ~/<your_folder_name>
```

3. 콘텐츠의 압축을 푼 후 폴더로 `CVO_Aws_Deployment` 이동하고 다음 명령을 실행하여 파일을 봅니다.

```
 ls -la
```

   다음 예제와 유사한 파일 목록이 표시됩니다.

```
total 32
    drwxr-xr-x   8 user1  staff   256 Mar 23 12:26 .
    drwxr-xr-x   6 user1  staff   192 Mar 22 08:04 ..
    -rw-r--r--   1 user1  staff   324 Apr 12 21:37 .env
    -rw-r--r--   1 user1  staff  1449 Mar 23 13:19 Dockerfile
    drwxr-xr-x  15 user1  staff   480 Mar 23 13:19 cvo_Aws_source_code
    drwxr-xr-x   4 user1  staff   128 Apr 27 13:43 cvo_Aws_variables
    -rw-r--r--   1 user1  staff   996 Mar 24 04:06 docker-compose-
deploy.yml
    -rw-r--r--   1 user1  staff  1041 Mar 24 04:06 docker-compose-
destroy.yml
```

4. 파일을 찾습니다 `cvo_aws_flexcache_ubuntu_image.tar`. Cloud Volumes ONTAP for AWS를 구축하는 데 필요한 Docker 이미지가 포함되어 있습니다.
5. 파일의 압축을 풉니다.

```
docker load -i cvo_aws_flexcache_ubuntu_image.tar
```

6. Docker 이미지가 로드될 때까지 몇 분 기다린 다음 Docker 이미지가 성공적으로 로드되었는지 확인합니다.

```
docker images
```

다음 예와 같이 태그와 함께 `latest` 이름이 지정된 Docker 이미지가 표시됩니다 `cvo_aws_flexcache_ubuntu_image`.

```
REPOSITORY                                TAG        IMAGE ID       CREATED
SIZE
 cvo_aws_flexcache_ubuntu_image     latest    18db15a4d59c   2 weeks ago
1.14GB
```

> 필요한 경우 Docker 이미지 이름을 변경할 수 있습니다. Docker 이미지 이름을 변경하는 경우 및 `docker-compose-destroy` 파일에서 Docker 이미지 이름을 업데이트해야 `docker-compose-deploy` 합니다.

3단계**:** 환경 변수 파일을 만듭니다

이 단계에서는 두 개의 환경 변수 파일을 만들어야 합니다. 한 파일은 AWS 액세스 키와 비밀 키를 사용하여 AWS Resource Manager API를 인증하기 위한 것입니다. 두 번째 파일은 콘솔 Terraform 모듈이 AWS API를 찾아 인증할 수 있도록 환경 변수를 설정하는 데 사용됩니다.

단계

1. 다음 위치에 파일을 만듭니다 `awsauth.env`.

`path/to/env-file/awsauth.env`

a. 파일에 다음 내용을 `awsauth.env` 추가합니다.

   access_key=<> secret_key=<>

형식은 * 위에 표시된 것과 정확히 일치해야 합니다.

2. 절대 파일 경로를 `.env` 파일에 추가합니다.

   환경 변수에 해당하는 환경 파일의 `AWS_CREDS` 절대 경로를 `awsauth.env` 입력합니다.

   `AWS_CREDS=path/to/env-file/awsauth.env`

3. `cvo_aws_variable`폴더로 이동하여 자격 증명 파일에서 액세스 및 비밀 키를 업데이트합니다.

   파일에 다음 내용을 추가합니다.

   AWS_ACCESS_KEY_ID = <> AWS_SYSTEM_ACCESS_KEY=<>

   형식은 * 위에 표시된 것과 정확히 일치해야 합니다.

4단계**: NetApp Intelligent Services**에 가입하세요

클라우드 공급업체를 통해 NetApp Intelligent Services에 가입하여 시간당 요금(PAYGO)을 지불하거나 연간 계약을 맺으세요. NetApp Intelligent Services에는 NetApp 백업 및 복구, Cloud Volumes ONTAP, NetApp 클라우드 계층화, NetApp 랜섬웨어 복원력 및 NetApp 재해 복구가 포함됩니다. NetApp 데이터 분류는 추가 비용 없이 구독에 포함되어 있습니다.

단계

1. Amazon Web Services(AWS) 포털에서 *SaaS*로 이동하여 * NetApp Intelligent Services 구독*을 선택합니다.

   Cloud Volumes ONTAP와 동일한 리소스 그룹을 사용하거나 다른 리소스 그룹을 사용할 수 있습니다.

2. NetApp 콘솔 포털을 구성하여 SaaS 구독을 콘솔로 가져옵니다.

   이 구성은 AWS 포털에서 직접 구성할 수 있습니다.

   구성을 확인하기 위해 콘솔 포털로 리디렉션됩니다.

3. 콘솔 포털에서 *저장*을 선택하여 구성을 확인합니다.

5단계**:** 외부 볼륨을 만듭니다

Terraform 상태 파일과 기타 중요한 파일을 영구적으로 유지하려면 외부 볼륨을 생성해야 합니다. Terraform에서 워크플로 및 배포를 실행하려면 파일을 사용할 수 있는지 확인해야 합니다.

단계

1. Docker 외부에서 외부 볼륨 생성 Compose:

```
docker volume create <volume_name>
```

예:

```
docker volume create cvo_aws_volume_dst
```

2. 다음 옵션 중 하나를 사용합니다.

   a. 환경 파일에 외부 볼륨 경로를 추가합니다 `.env`.

      아래 표시된 형식을 정확히 따라야 합니다.

      형식:

      `PERSISTENT_VOL=path/to/external/volume:/cvo_aws`

      예:
      `PERSISTENT_VOL=cvo_aws_volume_dst:/cvo_aws`

   b. NFS 공유를 외부 볼륨으로 추가합니다.

      Docker 컨테이너가 NFS 공유와 통신할 수 있으며 읽기/쓰기와 같은 올바른 권한이 구성되어 있는지 확인합니다.

      i. 다음과 같이 NFS 공유 경로를 Docker Compose 파일의 외부 볼륨에 대한 경로로 추가합니다. 형식:

         `PERSISTENT_VOL=path/to/nfs/volume:/cvo_aws`

   예:
   `PERSISTENT_VOL=nfs/mnt/document:/cvo_aws`

3. `cvo_aws_variables`폴더로 이동합니다.

   폴더에 다음 변수 파일이 표시됩니다.

   - `terraform.tfvars`
   - `variables.tf`

4. 요구 사항에 따라 파일 내의 값을 `terraform.tfvars` 변경합니다.

   파일의 변수 값을 수정할 때는 특정 지원 문서를 읽어야 `terraform.tfvars` 합니다. 값은 지역, 가용 영역 및 Cloud Volumes ONTAP for AWS에서 지원하는 기타 요인에 따라 달라질 수 있습니다. 여기에는 단일 노드에 대한 라이센스, 디스크 크기, VM 크기 및 고가용성(HA) 쌍이 포함됩니다.

   Console 에이전트 및 Cloud Volumes ONTAP Terraform 모듈에 대한 모든 지원 변수는 이미 정의되어 있습니다. `variables.tf` 파일. 변수 이름을 참조해야 합니다. `variables.tf` 추가하기 전에 파일 `terraform.tfvars` 파일.

5. 요구 사항에 따라 다음 옵션을 또는 `false` 로 설정하여 FlexCache 및 FlexClone를 활성화하거나 비활성화할 수

`true` 있습니다.

다음 예에서는 FlexCache 및 FlexClone를 사용합니다.

- `is_flexcache_required = true`
- `is_flexclone_required = true`

6단계: AWS 환경을 위한 Cloud Volumes ONTAP를 구축합니다

다음 단계에 따라 Cloud Volumes ONTAP for AWS를 구축하십시오.

단계

1. 루트 폴더에서 다음 명령을 실행하여 배포를 트리거합니다.

```
docker-compose -f docker-compose-deploy.yml up -d
```

두 개의 컨테이너가 트리거되고 첫 번째 컨테이너가 Cloud Volumes ONTAP를 배포하며 두 번째 컨테이너가 원격 측정 데이터를 AutoSupport로 전송합니다.

두 번째 컨테이너는 첫 번째 컨테이너가 모든 단계를 성공적으로 완료할 때까지 대기합니다.

2. 로그 파일을 사용하여 배포 프로세스의 진행 상황을 모니터링합니다.

```
docker-compose -f docker-compose-deploy.yml logs -f
```

이 명령은 실시간으로 출력을 제공하고 다음 로그 파일에 데이터를 캡처합니다.
`deployment.log`

`telemetry_asup.log`

다음 환경 변수를 사용하여 파일을 편집하여 이러한 로그 파일의 이름을 변경할 수 `.env` 있습니다.

`DEPLOYMENT_LOGS`

`TELEMETRY_ASUP_LOGS`

다음 예제에서는 로그 파일 이름을 변경하는 방법을 보여 줍니다.

`DEPLOYMENT_LOGS=<your_deployment_log_filename>.log`

`TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log`

작업을 마친 후

다음 단계에 따라 임시 환경을 제거하고 배포 프로세스 중에 만든 항목을 정리할 수 있습니다.

단계

1. FlexCache를 배포한 경우 변수 파일에서 다음 옵션을 `terraform.tfvars` 설정하면 FlexCache 볼륨이

정리되고 이전에 생성된 임시 환경이 제거됩니다.

```
flexcache_operation = "destroy"
```

> 가능한 옵션은 `deploy`및 `destroy`입니다.

2. FlexClone를 배포한 경우 변수 파일에서 다음 옵션을 `terraform.tfvars` 설정하면 FlexClone 볼륨이 정리되고 이전에 생성된 임시 환경이 제거됩니다.

```
flexclone_operation = "destroy"
```

> 가능한 옵션은 `deploy` 및 `destroy`입니다.

Azure용 Cloud Volumes ONTAP

Cloud Volumes ONTAP for Azure - 클라우드 사용 급증

이 문서에서는 NetApp 고객이 NetApp Console 자동화 허브에서 사용할 수 있는 Azure Automation 솔루션용 NetApp Cloud Volumes ONTAP 지원합니다.

Cloud Volumes ONTAP for Azure Automation 솔루션은 Terraform을 사용하여 Cloud Volumes ONTAP for Azure 컨테이너식 구축을 자동화하므로 수동 개입 없이 Azure용 Cloud Volumes ONTAP를 빠르게 배포할 수 있습니다.

시작하기 전에

- 다운로드해야 합니다 "Cloud Volumes ONTAP Azure - 클라우드 사용 급증" 콘솔 웹 UI를 통한 자동화 솔루션. 솔루션은 다음과 같이 패키지됩니다. `CVO-Azure-Burst-To-Cloud.zip`.
- Cloud Volumes ONTAP와 동일한 네트워크에 Linux VM을 설치해야 합니다.
- Linux VM을 설치한 후에는 이 솔루션의 단계에 따라 필요한 종속성을 설치해야 합니다.

1단계: Docker 및 Docker Compose를 설치합니다

Docker를 설치합니다

다음 단계에서는 Ubuntu 20.04 Debian Linux 배포 소프트웨어를 예로 사용합니다. 실행하는 명령은 사용 중인 Linux 배포 소프트웨어에 따라 다릅니다. 해당 구성에 대한 자세한 내용은 해당 Linux 배포 소프트웨어 설명서를 참조하십시오.

단계

1. 다음 명령을 실행하여 Docker를 `sudo` 설치합니다.

```
sudo apt-get update
sudo apt-get install apt-transport-https cacertificates curl gnupg-agent
software-properties-common curl -fsSL
https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add -
sudo add-apt-repository “deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable”
sudo apt-get update
sudo apt-get install dockerce docker-ce-cli containerd.io
```

2. 설치를 확인합니다.

```
docker –version
```

3. Linux 시스템에 "docker"라는 그룹이 생성되었는지 확인합니다. 필요한 경우 그룹을 생성합니다.

```
sudo groupadd docker
```

4. Docker에 액세스해야 하는 사용자를 그룹에 추가합니다.

```
sudo usermod -aG docker $(whoami)
```

5. 변경 사항은 로그아웃한 후 터미널에 다시 로그인하면 적용됩니다. 또는 변경 사항을 즉시 적용할 수 있습니다.

```
newgrp docker
```

Docker Compose를 설치합니다

단계

1. 다음 명령을 실행하여 Docker Compose를 `sudo` 설치합니다.

```
sudo curl -L
“https://github.com/docker/compose/releases/download/1.29.2/dockercompos
e-(□□□□□ - □)-(uname -m)” -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

2. 설치를 확인합니다.

```
docker-compose –version
```

2단계: Docker 이미지 준비

단계

1. `CVO-Azure-Burst-To-Cloud.zip`Cloud Volumes ONTAP를 배포하는 데 사용할 Linux VM에 폴더를 복사합니다.

```
scp -i ~/<private-key>.pem -r CVO-Azure-Burst-To-Cloud.zip
<azureuser>@<IP_ADDRESS_OF_VM>:<LOCATION_TO_BE_COPIED>
```

   - `private-key.pem` 은(는) 암호 없이 로그인할 수 있는 개인 키 파일입니다.
   - `azureuser` VM 사용자 이름입니다.
   - `IP_ADDRESS_OF_VM` VM IP 주소입니다.
   - `LOCATION_TO_BE_COPIED` 폴더가 복사될 위치입니다.

2. 폴더 압축을 `CVO-Azure-Burst-To-Cloud.zip` 풉니다. 현재 디렉터리나 사용자 지정 위치에서 폴더의 압축을 풀 수 있습니다.

   현재 디렉터리에서 폴더를 추출하려면 다음을 실행합니다.

```
unzip CVO-Azure-Burst-To-Cloud.zip
```

   사용자 지정 위치에서 폴더를 추출하려면 다음을 실행합니다.

```
unzip CVO-Azure-Burst-To-Cloud.zip -d ~/<your_folder_name>
```

3. 콘텐츠의 압축을 푼 후 폴더로 `CVO_Azure_Deployment` 이동하고 다음 명령을 실행하여 파일을 봅니다.

```
 ls -la
```

   다음 예제와 유사한 파일 목록이 표시됩니다.

```
drwxr-xr-x@ 11 user1 staff 352 May 5 13:56 .
drwxr-xr-x@ 5 user1 staff 160 May 5 14:24 ..
-rw-r--r--@ 1 user1 staff 324 May 5 13:18 .env
-rw-r--r--@ 1 user1 staff 1449 May 5 13:18 Dockerfile
-rw-r--r--@ 1 user1 staff 35149 May 5 13:18 LICENSE
-rw-r--r--@ 1 user1 staff 13356 May 5 14:26 README.md
-rw-r--r-- 1  user1 staff 354318151 May 5 13:51
cvo_azure_flexcache_ubuntu_image_latest
drwxr-xr-x@ 4 user1 staff 128 May 5 13:18 cvo_azure_variables
-rw-r--r--@ 1 user1 staff 996 May 5 13:18 docker-compose-deploy.yml
-rw-r--r--@ 1 user1 staff 1041 May 5 13:18 docker-compose-destroy.yml
-rw-r--r--@ 1 user1 staff 4771 May 5 13:18 sp_role.json
```

4. 파일을 찾습니다 `cvo_azure_flexcache_ubuntu_image_latest.tar.gz`. Cloud Volumes ONTAP for Azure를 구축하는 데 필요한 Docker 이미지가 포함되어 있습니다.
5. 파일의 압축을 풉니다.

```
docker load -i cvo_azure_flexcache_ubuntu_image_latest.tar.gz
```

6. Docker 이미지가 로드될 때까지 몇 분 기다린 다음 Docker 이미지가 성공적으로 로드되었는지 확인합니다.

```
docker images
```

다음 예와 같이 태그와 함께 `latest` 이름이 지정된 Docker 이미지가 표시됩니다 `cvo_azure_flexcache_ubuntu_image_latest`.

```
REPOSITORY TAG IMAGE ID CREATED SIZE
cvo_azure_flexcache_ubuntu_image latest 18db15a4d59c 2 weeks ago 1.14GB
```

3단계**:** 환경 변수 파일을 만듭니다

이 단계에서는 두 개의 환경 변수 파일을 만들어야 합니다. 한 파일은 서비스 주체 자격 증명을 사용하여 Azure Resource Manager API를 인증하기 위한 것입니다. 두 번째 파일은 콘솔 Terraform 모듈이 Azure API를 찾아 인증할 수 있도록 환경 변수를 설정하는 데 사용됩니다.

단계

1. 서비스 보안 주체를 만듭니다.

   환경 변수 파일을 만들려면 의 단계에 따라 서비스 보안 주체를 만들어야 "리소스에 액세스할 수 있는 Azure Active Directory 응용 프로그램 및 서비스 보안 주체를 만듭니다"합니다.

2. 새로 만든 서비스 사용자에게 * Contributor * 역할을 할당합니다.

3. 사용자 지정 역할을 만듭니다.
   a. `sp_role.json`파일을 찾아 나열된 작업 아래에서 필요한 권한을 확인합니다.
   b. 이러한 권한을 삽입하고 사용자 지정 역할을 새로 만든 서비스 보안 주체에 연결합니다.
4. 인증서 및 암호 * 로 이동하고 * 새 클라이언트 암호 * 를 선택하여 클라이언트 암호를 만듭니다.

   클라이언트 암호를 만들 때 이 값을 다시 볼 수 없으므로 * 값 * 열의 세부 정보를 기록해야 합니다. 또한 다음 정보를 기록해야 합니다.

   - 클라이언트 ID입니다
   - 구독 ID입니다
   - 테넌트 ID입니다

     환경 변수를 생성하려면 이 정보가 필요합니다. 클라이언트 ID 및 테넌트 ID 정보는 서비스 사용자 UI의 * 개요 * 섹션에서 찾을 수 있습니다.
5. 환경 파일을 만듭니다.
   a. 다음 위치에 파일을 만듭니다 `azureauth.env`.

      `path/to/env-file/azureauth.env`

      i. 파일에 다음 내용을 추가합니다.

         클라이언트 ID = <> clientSecret = <> 아래 첨자 ID = <> tenantId = <>

         - 형식은 키와 값 사이에 공백을 제외하고 위에 표시된 것과 정확히 일치해야 합니다.
   b. 다음 위치에 파일을 만듭니다 `credentials.env`.

      `path/to/env-file/credentials.env`

      i. 파일에 다음 내용을 추가합니다.

         Azure_Tenant_ID=<> Azure_client_secret=<> Azure_client_ID=<> Azure_subscription_ID=<>

         - 형식은 키와 값 사이에 공백을 제외하고 위에 표시된 것과 정확히 일치해야 합니다.
6. 절대 파일 경로를 `.env` 파일에 추가합니다.

   환경 변수에 해당하는 파일에 `AZURE_RM_CREDS` 환경 파일의 `.env` 절대 경로를 입력합니다 `azureauth.env`.

   `AZURE_RM_CREDS=path/to/env-file/azureauth.env`

   환경 변수에 해당하는 파일에 `BLUEXP_TF_AZURE_CREDS` 환경 파일의 `.env` 절대 경로를 입력합니다 `credentials.env`.

   `BLUEXP_TF_AZURE_CREDS=path/to/env-file/credentials.env`

4단계: NetApp Intelligent Services에 가입하세요

클라우드 공급업체를 통해 NetApp Intelligent Services에 가입하여 시간당 요금(PAYGO)을 지불하거나 연간 계약을 맺으세요. NetApp Intelligent Services에는 NetApp 백업 및 복구, Cloud Volumes ONTAP, NetApp 클라우드 계층화, NetApp 랜섬웨어 복원력 및 NetApp 재해 복구가 포함됩니다. NetApp 데이터 분류는 추가 비용 없이 구독에 포함됩니다.

단계

1. Azure Portal에서 *SaaS*로 이동하여 * NetApp Intelligent Services 구독*을 선택합니다.
2. Cloud Manager(시간당 Cap PYGO, WORM 및 데이터 서비스 기준) * 계획을 선택합니다.

   Cloud Volumes ONTAP와 동일한 리소스 그룹을 사용하거나 다른 리소스 그룹을 사용할 수 있습니다.

3. 콘솔 포털을 구성하여 SaaS 구독을 콘솔로 가져옵니다.

   Azure 포털에서 * 제품 및 계획 세부 정보 * 로 이동하고 * 지금 계정 구성 * 옵션을 선택하여 직접 구성할 수 있습니다.

   그러면 구성을 확인하기 위해 콘솔 포털로 리디렉션됩니다.

4. 콘솔 포털에서 *저장*을 선택하여 구성을 확인합니다.

5단계: 외부 볼륨을 만듭니다

Terraform 상태 파일과 기타 중요한 파일을 영구적으로 유지하려면 외부 볼륨을 생성해야 합니다. Terraform에서 워크플로 및 배포를 실행하려면 파일을 사용할 수 있는지 확인해야 합니다.

단계

1. Docker 외부에서 외부 볼륨 생성 Compose:

   ```
   docker volume create « volume_name »
   ```

   예:

   ```
   docker volume create cvo_azure_volume_dst
   ```

2. 다음 옵션 중 하나를 사용합니다.

   a. 환경 파일에 외부 볼륨 경로를 추가합니다 `.env`.

      아래 표시된 형식을 정확히 따라야 합니다.

      형식:

      ```
      PERSISTENT_VOL=path/to/external/volume:/cvo_azure
      ```

      예:
      `PERSISTENT_VOL=cvo_azure_volume_dst:/cvo_azure`

b. NFS 공유를 외부 볼륨으로 추가합니다.

Docker 컨테이너가 NFS 공유와 통신할 수 있으며 읽기/쓰기와 같은 올바른 권한이 구성되어 있는지 확인합니다.

i. 다음과 같이 NFS 공유 경로를 Docker Compose 파일의 외부 볼륨에 대한 경로로 추가합니다. 형식:

```
PERSISTENT_VOL=path/to/nfs/volume:/cvo_azure
```

예:
`PERSISTENT_VOL=nfs/mnt/document:/cvo_azure`

3. `cvo_azure_variables`폴더로 이동합니다.

폴더에 다음 변수 파일이 표시됩니다.

`terraform.tfvars`

`variables.tf`

4. 요구 사항에 따라 파일 내의 값을 `terraform.tfvars` 변경합니다.

파일의 변수 값을 수정할 때는 특정 지원 문서를 읽어야 `terraform.tfvars` 합니다. 값은 지역, 가용 영역 및 Cloud Volumes ONTAP for Azure에서 지원하는 기타 요인에 따라 달라질 수 있습니다. 여기에는 단일 노드에 대한 라이센스, 디스크 크기, VM 크기 및 고가용성(HA) 쌍이 포함됩니다.

Console 에이전트 및 Cloud Volumes ONTAP Terraform 모듈에 대한 모든 지원 변수는 이미 정의되어 있습니다. `variables.tf` 파일. 변수 이름을 참조해야 합니다. `variables.tf` 추가하기 전에 파일 `terraform.tfvars` 파일.

5. 요구 사항에 따라 다음 옵션을 또는 `false` 로 설정하여 FlexCache 및 FlexClone를 활성화하거나 비활성화할 수 `true` 있습니다.

다음 예에서는 FlexCache 및 FlexClone를 사용합니다.

- `is_flexcache_required = true`
- `is_flexclone_required = true`

6. 필요한 경우 Azure Active Directory 서비스에서 Terraform 변수의 값을 검색할 수 있습니다 `az_service_principal_object_id`.

a. 엔터프라이즈 애플리케이션 –> 모든 애플리케이션 * 으로 이동하고 앞서 생성한 서비스 사용자 이름을 선택합니다.

b. 객체 ID를 복사하고 Terraform 변수의 값을 삽입합니다.

```
az_service_principal_object_id
```

6단계**: Azure**용 **Cloud Volumes ONTAP**를 배포합니다

다음 단계에 따라 Azure용 Cloud Volumes ONTAP를 배포합니다.

단계

1. 루트 폴더에서 다음 명령을 실행하여 배포를 트리거합니다.

```
docker-compose up -d
```

두 개의 컨테이너가 트리거되고 첫 번째 컨테이너가 Cloud Volumes ONTAP를 배포하며 두 번째 컨테이너가 원격 측정 데이터를 AutoSupport로 전송합니다.

두 번째 컨테이너는 첫 번째 컨테이너가 모든 단계를 성공적으로 완료할 때까지 대기합니다.

2. 로그 파일을 사용하여 배포 프로세스의 진행 상황을 모니터링합니다.

```
docker-compose logs -f
```

이 명령은 실시간으로 출력을 제공하고 다음 로그 파일에 데이터를 캡처합니다.

`deployment.log`

`telemetry_asup.log`

다음 환경 변수를 사용하여 파일을 편집하여 이러한 로그 파일의 이름을 변경할 수 `.env` 있습니다.

`DEPLOYMENT_LOGS`

`TELEMETRY_ASUP_LOGS`

다음 예제에서는 로그 파일 이름을 변경하는 방법을 보여 줍니다.

`DEPLOYMENT_LOGS=<your_deployment_log_filename>.log`

`TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log`

작업을 마친 후

다음 단계에 따라 임시 환경을 제거하고 배포 프로세스 중에 만든 항목을 정리할 수 있습니다.

단계

1. FlexCache를 배포한 경우 파일에 다음 옵션을 `terraform.tfvars` 설정하면 FlexCache 볼륨이 지워지고 이전에 생성된 임시 환경이 제거됩니다.

   `flexcache_operation = "destroy"`

   > 가능한 옵션은 `deploy`및 `destroy`입니다.

2. FlexClone를 배포한 경우 파일에 다음 옵션을 `terraform.tfvars` 설정하면 FlexClone 볼륨이 지워지고 이전에 생성된 임시 환경이 제거됩니다.

   `flexclone_operation = "destroy"`

가능한 옵션은 `deploy` 및 `destroy`입니다.

Google Cloud용 Cloud Volumes ONTAP

Google Cloud용 Cloud Volumes ONTAP - 클라우드 사용 급증

이 문서에서는 NetApp 고객이 NetApp Console 자동화 허브에서 사용할 수 있는 Google Cloud Automation 솔루션용 NetApp Cloud Volumes ONTAP 지원합니다.

Google Cloud Automation용 Cloud Volumes ONTAP 솔루션은 Cloud Volumes ONTAP for Google Cloud를 컨테이너식으로 구축하여 수동 개입 없이 Google Cloud용 Cloud Volumes ONTAP를 빠르게 배포할 수 있도록 합니다.

시작하기 전에

- 다운로드해야 합니다 "Google Cloud용 Cloud Volumes ONTAP - 클라우드 사용 급증" 콘솔 웹 UI를 통한 자동화 솔루션. 솔루션은 다음과 같이 패키지됩니다. `cvo_gcp_flexcache.zip`.
- Cloud Volumes ONTAP와 동일한 네트워크에 Linux VM을 설치해야 합니다.
- Linux VM을 설치한 후에는 이 솔루션의 단계에 따라 필요한 종속성을 설치해야 합니다.

1단계: Docker 및 Docker Compose를 설치합니다

Docker를 설치합니다

다음 단계에서는 Ubuntu 20.04 Debian Linux 배포 소프트웨어를 예로 사용합니다. 실행하는 명령은 사용 중인 Linux 배포 소프트웨어에 따라 다릅니다. 해당 구성에 대한 자세한 내용은 해당 Linux 배포 소프트웨어 설명서를 참조하십시오.

단계

1. 다음 명령을 실행하여 Docker를 설치합니다.

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl gnupg-
agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

2. 설치를 확인합니다.

```
docker -version
```

3. Linux 시스템에 "docker"라는 그룹이 생성되었는지 확인합니다. 필요한 경우 그룹을 생성합니다.

```
sudo groupadd docker
```

4. Docker에 액세스해야 하는 사용자를 그룹에 추가합니다.

```
sudo usermod -aG docker $(whoami)
```

5. 변경 사항은 로그아웃한 후 터미널에 다시 로그인하면 적용됩니다. 또는 변경 사항을 즉시 적용할 수 있습니다.

```
newgrp docker
```

Docker Compose를 설치합니다

단계

1. 다음 명령을 실행하여 Docker Compose를 `sudo` 설치합니다.

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

2. 설치를 확인합니다.

```
docker-compose –version
```

2단계: Docker 이미지 준비

단계

1. `cvo_gcp_flexcache.zip`Cloud Volumes ONTAP를 배포하는 데 사용할 Linux VM에 폴더를 복사합니다.

```
scp -i ~/private-key.pem -r cvo_gcp_flexcache.zip
gcpuser@IP_ADDRESS_OF_VM:LOCATION_TO_BE_COPIED
```

   - `private-key.pem` 은(는) 암호 없이 로그인할 수 있는 개인 키 파일입니다.
   - `gcpuser` VM 사용자 이름입니다.
   - `IP_ADDRESS_OF_VM` VM IP 주소입니다.

- LOCATION_TO_BE_COPIED 폴더가 복사될 위치입니다.

2. 폴더 압축을 cvo_gcp_flexcache.zip 풉니다. 현재 디렉터리나 사용자 지정 위치에서 폴더의 압축을 풀 수 있습니다.

   현재 디렉터리에서 폴더를 추출하려면 다음을 실행합니다.

```
unzip cvo_gcp_flexcache.zip
```

   사용자 지정 위치에서 폴더를 추출하려면 다음을 실행합니다.

```
unzip cvo_gcp_flexcache.zip -d ~/<your_folder_name>
```

3. 콘텐츠를 추출한 후 다음 명령을 실행하여 파일을 봅니다.

```
 ls -la
```

   다음 예제와 유사한 파일 목록이 표시됩니다.

```
    total 32
    drwxr-xr-x   8 user  staff   256 Mar 23 12:26 .
    drwxr-xr-x   6 user  staff   192 Mar 22 08:04 ..
    -rw-r--r--   1 user  staff   324 Apr 12 21:37 .env
    -rw-r--r--   1 user  staff  1449 Mar 23 13:19 Dockerfile
    drwxr-xr-x  15 user  staff   480 Mar 23 13:19 cvo_gcp_source_code
    drwxr-xr-x   4 user  staff   128 Apr 27 13:43 cvo_gcp_variables
    -rw-r--r--   1 user  staff   996 Mar 24 04:06 docker-compose-
deploy.yml
    -rw-r--r--   1 user  staff  1041 Mar 24 04:06 docker-compose-
destroy.yml
```

4. 파일을 찾습니다 cvo_gcp_flexcache_ubuntu_image.tar. Cloud Volumes ONTAP for Google Cloud를 구축하는 데 필요한 Docker 이미지가 포함되어 있습니다.
5. 파일의 압축을 풉니다.

```
docker load -i cvo_gcp_flexcache_ubuntu_image.tar
```

6. Docker 이미지가 로드될 때까지 몇 분 기다린 다음 Docker 이미지가 성공적으로 로드되었는지 확인합니다.

```
docker images
```

다음 예와 같이 태그와 함께 `latest` 이름이 지정된 Docker 이미지가 표시됩니다 `cvo_gcp_flexcache_ubuntu_image`.

```
REPOSITORY                                TAG         IMAGE ID       CREATED
SIZE
    cvo_gcp_flexcache_ubuntu_image      latest     18db15a4d59c    2 weeks
ago   1.14GB
```

> 필요한 경우 Docker 이미지 이름을 변경할 수 있습니다. Docker 이미지 이름을 변경하는 경우 및 `docker-compose-destroy` 파일에서 Docker 이미지 이름을 업데이트해야 `docker-compose-deploy` 합니다.

단계 **3: JSON** 파일을 업데이트합니다

이 단계에서는 Google Cloud 공급자를 인증하기 위해 서비스 계정 키로 파일을 업데이트해야 `cxo-automation-gcp.json` 합니다.

1. Cloud Volumes ONTAP 및 콘솔 에이전트를 배포할 수 있는 권한이 있는 서비스 계정을 만듭니다."서비스 계정 만들기에 대해 자세히 알아보십시오."
2. 계정의 키 파일을 다운로드하고 주요 파일 정보로 파일을 업데이트합니다 `cxo-automation-gcp.json`. `cxo-automation-gcp.json`파일이 `cvo_gcp_variables` 폴더에 있습니다.

   예

```
{
  "type": "service_account",
  "project_id": "",
  "private_key_id": "",
  "private_key": "",
  "client_email": "",
  "client_id": "",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "",
  "universe_domain": "googleapis.com"
}
```

   파일 형식은 위에 표시된 것과 정확히 일치해야 합니다.

4단계**: NetApp Intelligent Services**에 가입하세요

클라우드 공급업체를 통해 NetApp Intelligent Services에 가입하여 시간당 요금(PAYGO)을 지불하거나 연간 계약을 맺으세요. NetApp Intelligent Services에는 NetApp 백업 및 복구, Cloud Volumes ONTAP, NetApp 클라우드 계층화,

NetApp 랜섬웨어 복원력 및 NetApp 재해 복구가 포함됩니다. NetApp 데이터 분류는 추가 비용 없이 구독에 포함되어 있습니다.

단계

1. 로 이동합니다"Google Cloud 콘솔" * NetApp Intelligent Services 구독*을 선택하세요.
2. NetApp 콘솔 포털을 구성하여 SaaS 구독을 콘솔로 가져옵니다.

   Google Cloud Platform에서 직접 구성할 수 있습니다. 구성을 확인하기 위해 콘솔 포털로 리디렉션됩니다.

3. 콘솔 포털에서 *저장*을 선택하여 구성을 확인합니다.

자세한 내용은 다음을 참조하세요. "NetApp 콘솔에 대한 Google Cloud 자격 증명 및 구독 관리" .

5단계**:** 필요한 **Google Cloud API**를 활성화합니다

Cloud Volumes ONTAP 과 콘솔 에이전트를 배포하려면 프로젝트에서 다음 Google Cloud API를 활성화해야 합니다.

- Cloud Deployment Manager V2 API
- 클라우드 로깅 API
- Cloud Resource Manager API를 참조하십시오
- 컴퓨팅 엔진 API
- IAM(Identity and Access Management) API

"API 사용에 대해 자세히 알아보십시오"

6단계**:** 외부 볼륨을 만듭니다

Terraform 상태 파일 및 기타 중요한 파일을 영구적으로 유지하려면 외부 볼륨을 만들어야 합니다. Terraform에서 워크플로 및 배포를 실행하려면 파일을 사용할 수 있는지 확인해야 합니다.

단계

1. Docker 외부에서 외부 볼륨 생성 Compose:

   ```
   docker volume create <volume_name>
   ```

   예:

   ```
   docker volume create cvo_gcp_volume_dst
   ```

2. 다음 옵션 중 하나를 사용합니다.

   a. 환경 파일에 외부 볼륨 경로를 추가합니다 `.env`.

      아래 표시된 형식을 정확히 따라야 합니다.

      형식:

```
PERSISTENT_VOL=path/to/external/volume:/cvo_gcp
```

예:
`PERSISTENT_VOL=cvo_gcp_volume_dst:/cvo_gcp`

b. NFS 공유를 외부 볼륨으로 추가합니다.

Docker 컨테이너가 NFS 공유와 통신할 수 있으며 읽기/쓰기와 같은 올바른 권한이 구성되어 있는지 확인합니다.

i. 다음과 같이 NFS 공유 경로를 Docker Compose 파일의 외부 볼륨에 대한 경로로 추가합니다. 형식:

```
PERSISTENT_VOL=path/to/nfs/volume:/cvo_gcp
```

예:
`PERSISTENT_VOL=nfs/mnt/document:/cvo_gcp`

3. `cvo_gcp_variables`폴더로 이동합니다.

폴더에 다음 파일이 표시됩니다.

- `terraform.tfvars`
- `variables.tf`

4. 요구 사항에 따라 파일 내의 값을 `terraform.tfvars` 변경합니다.

파일의 변수 값을 수정할 때는 특정 지원 문서를 읽어야 `terraform.tfvars` 합니다. 값은 지역, 가용 영역 및 Cloud Volumes ONTAP for Google Cloud에서 지원하는 기타 요인에 따라 달라질 수 있습니다. 여기에는 단일 노드에 대한 라이센스, 디스크 크기, VM 크기 및 고가용성(HA) 쌍이 포함됩니다.

Console 에이전트 및 Cloud Volumes ONTAP Terraform 모듈에 대한 모든 지원 변수는 이미 정의되어 있습니다. `variables.tf` 파일. 변수 이름을 참조해야 합니다. `variables.tf` 추가하기 전에 파일 `terraform.tfvars` 파일.

5. 요구 사항에 따라 다음 옵션을 또는 `false` 로 설정하여 FlexCache 및 FlexClone를 활성화하거나 비활성화할 수 `true` 있습니다.

다음 예에서는 FlexCache 및 FlexClone를 사용합니다.

- `is_flexcache_required = true`
- `is_flexclone_required = true`

7단계**: Google Cloud**용 **Cloud Volumes ONTAP**를 배포합니다

다음 단계에 따라 Google Cloud용 Cloud Volumes ONTAP를 구축하십시오.

단계

1. 루트 폴더에서 다음 명령을 실행하여 배포를 트리거합니다.

```
docker-compose -f docker-compose-deploy.yml up -d
```

두 개의 컨테이너가 트리거되고 첫 번째 컨테이너가 Cloud Volumes ONTAP를 배포하며 두 번째 컨테이너가 원격 측정 데이터를 AutoSupport로 전송합니다.

두 번째 컨테이너는 첫 번째 컨테이너가 모든 단계를 성공적으로 완료할 때까지 대기합니다.

2. 로그 파일을 사용하여 배포 프로세스의 진행 상황을 모니터링합니다.

```
docker-compose -f docker-compose-deploy.yml logs -f
```

이 명령은 실시간으로 출력을 제공하고 다음 로그 파일에 데이터를 캡처합니다.
`deployment.log`

`telemetry_asup.log`

다음 환경 변수를 사용하여 파일을 편집하여 이러한 로그 파일의 이름을 변경할 수 `.env` 있습니다.

`DEPLOYMENT_LOGS`

`TELEMETRY_ASUP_LOGS`

다음 예제에서는 로그 파일 이름을 변경하는 방법을 보여 줍니다.

`DEPLOYMENT_LOGS=<your_deployment_log_filename>.log`

`TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log`

작업을 마친 후

다음 단계에 따라 임시 환경을 제거하고 배포 프로세스 중에 만든 항목을 정리할 수 있습니다.

단계

1. FlexCache를 배포한 경우 파일에 다음 옵션을 `terraform.tfvars` 설정하면 FlexCache 볼륨이 지워지고 이전에 생성된 임시 환경이 제거됩니다.

   `flexcache_operation = "destroy"`

   > 가능한 옵션은 `deploy`및 `destroy`입니다.

2. FlexClone를 배포한 경우 파일에 다음 옵션을 `terraform.tfvars` 설정하면 FlexClone 볼륨이 지워지고 이전에 생성된 임시 환경이 제거됩니다.

   `flexclone_operation = "destroy"`

   > 가능한 옵션은 `deploy` 및 `destroy`입니다.

ONTAP

1일 차

ONTAP Day 0/1 솔루션 개요

ONTAP 0/1일 자동화 솔루션을 사용하면 Ansible을 사용하여 ONTAP 클러스터를 배포하고 구성할 수 있습니다. 해당 솔루션은 다음에서 제공됩니다. "NetApp Console 자동화 허브".

유연한 ONTAP 구축 옵션

요구사항에 따라 온프레미스 하드웨어를 사용하거나 ONTAP를 시뮬레이션하여 Ansible을 사용하여 ONTAP 클러스터를 구축 및 구성할 수 있습니다.

온프레미스 하드웨어

FAS 또는 AFF 시스템과 같이 ONTAP을 실행하는 온프레미스 하드웨어를 사용하여 이 솔루션을 배포할 수 있습니다. Ansible을 사용하여 ONTAP 클러스터를 구축하고 구성하려면 Linux VM을 사용해야 합니다.

ONTAP 시뮬레이션

ONTAP 시뮬레이터를 사용하여 이 솔루션을 구축하려면 NetApp Support 사이트에서 Simulate ONTAP의 최신 버전을 다운로드해야 합니다. Simulate ONTAP는 ONTAP 소프트웨어용 가상 시뮬레이터입니다. ONTAP는 Windows, Linux 또는 Mac 시스템의 VMware 하이퍼바이저에서 실행됩니다. Windows 및 Linux 호스트의 경우 VMware Workstation 하이퍼바이저를 사용하여 이 솔루션을 실행해야 합니다. Mac OS를 사용하는 경우 VMware Fusion 하이퍼바이저를 사용하십시오.

레이어드 디자인

Ansible 프레임워크는 자동화 실행 및 논리 작업의 개발과 재사용을 간소화합니다. 프레임워크는 의사 결정 작업(논리 계층)과 자동화의 실행 단계(실행 계층)를 구별합니다. 이러한 레이어의 작동 방식을 이해하면 구성을 사용자 정의할 수 있습니다.

Ansible "플레이북"은 처음부터 끝까지 일련의 작업을 실행합니다. `site.yml`플레이북에는 플레이북 및 `execution.yml` 플레이북이 포함되어 `logic.yml` 있습니다.

요청이 실행되면 `site.yml` Playbook은 먼저 Playbook을 호출한 `logic.yml` 다음 `execution.yml` Playbook을 호출하여 서비스 요청을 실행합니다.

프레임워크의 논리 계층을 사용할 필요는 없습니다. 논리 계층은 실행을 위해 하드 코드된 값 이상으로 프레임워크의 기능을 확장하는 옵션을 제공합니다. 따라서 필요한 경우 프레임워크 기능을 사용자 지정할 수 있습니다.

로직 레이어

논리 계층은 다음과 같이 구성됩니다.

- 플레이북 `logic.yml`
- 디렉토리 내의 논리 작업 파일 `logic-tasks`

논리 계층은 중대한 맞춤형 통합(예: ServiceNow에 연결) 없이도 복잡한 의사 결정을 위한 기능을 제공합니다. 논리 계층은 구성 가능하며 마이크로서비스에 입력을 제공합니다.

논리 계층을 우회하는 기능도 제공됩니다. 논리 계층을 무시하려면 변수를 정의하지 마십시오 `logic_operation`.

플레이북을 직접 호출하면 `logic.yml` 실행 없이 일부 수준의 디버깅을 수행할 수 있습니다. "debug" 문을 사용하여 의 값이 올바른지 확인할 수 `raw_service_request` 있습니다.

중요 고려 사항:

- `logic.yml`플레이북에서 `logic_operation` 변수를 검색합니다. 변수가 요청에 정의되어 있으면 디렉터리에서 작업 파일을 `logic-tasks` 로드합니다. 작업 파일은 .yml 파일이어야 합니다. 일치하는 작업 파일이 없고 변수가 정의된 경우 `logic_operation` 논리 계층이 실패합니다.
- 변수의 `logic_operation` 기본값은 입니다 `no-op`. 변수가 명시적으로 정의되지 않은 경우 기본값은 로 설정되며, 이 경우 `no-op` 아무 작업도 실행되지 않습니다.
- 변수가 이미 정의되어 있으면 `raw_service_request` 실행이 실행 계층으로 진행됩니다. 변수가 정의되지 않으면 논리 계층이 실패합니다.

실행 계층

실행 계층은 다음과 같이 구성됩니다.

- 플레이북 `execution.yml`

실행 계층은 ONTAP 클러스터를 구성하기 위해 API를 호출합니다. `execution.yml`Playbook을 사용하려면 실행 시 변수를 정의해야 `raw_service_request` 합니다.

사용자 지정 지원

요구 사항에 따라 다양한 방법으로 이 솔루션을 사용자 지정할 수 있습니다.

사용자 지정 옵션은 다음과 같습니다.

- Ansible 플레이북 수정
- 역할을 추가하는 중입니다

Ansible 파일 사용자 지정

다음 표에서는 이 솔루션에 포함된 사용자 지정 가능한 Ansible 파일을 설명합니다.

위치	설명
`playbooks/inventory/hosts`	호스트 및 그룹 목록이 포함된 단일 파일을 포함합니다.
`playbooks/group_vars/all/*`	Ansible을 사용하면 여러 호스트에 변수를 한 번에 간편하게 적용할 수 있습니다. 이 폴더에 있는 , `clusters.yml`, `defaults.yml`, , `services.yml`, `standards.yml` 및 `vault.yml` 를 포함한 모든 파일을 수정할 수 `cfg.yml` 있습니다.
`playbooks/logic-tasks`	Ansible 내에서 의사 결정 작업을 지원하고 논리와 실행의 분리를 유지합니다. 관련 서비스에 해당하는 파일을 이 폴더에 추가할 수 있습니다.
`playbooks/vars/*`	Ansible 플레이북 및 역할에 사용되는 동적인 가치를 제공하여 커스터마이징, 유연성 및 구성 재사용을 지원합니다. 필요한 경우 이 폴더의 모든 파일을 수정할 수 있습니다.

역할을 사용자 정의합니다

마이크로서비스라고도 하는 Ansible 역할을 추가 또는 변경하여 솔루션을 사용자 지정할 수도 있습니다. 자세한 내용은 을 참조하십시오"사용자 정의".

ONTAP Day 0/1 솔루션을 사용할 준비를 하십시오

자동화 솔루션을 구축하기 전에 ONTAP 환경을 준비하고 Ansible을 설치 및 구성해야 합니다.

초기 계획 고려 사항

이 솔루션을 사용하여 ONTAP 클러스터를 구축하기 전에 다음 요구 사항 및 고려 사항을 검토해야 합니다.

기본 요구 사항

이 솔루션을 사용하려면 다음 기본 요구 사항을 충족해야 합니다.

- 온프레미스 또는 ONTAP 시뮬레이터를 통해 ONTAP 소프트웨어에 액세스할 수 있어야 합니다.
- ONTAP 소프트웨어 사용 방법을 알고 있어야 합니다.
- Ansible 자동화 소프트웨어 툴을 사용하는 방법을 알아야 합니다.

계획 고려 사항

이 자동화 솔루션을 배포하기 전에 다음을 결정해야 합니다.

- Ansible 제어 노드를 실행할 위치입니다.
- ONTAP 시스템의 온프레미스 하드웨어 또는 ONTAP 시뮬레이터입니다.
- 사용자 지정 필요 여부.

ONTAP 시스템을 준비합니다

온프레미스 ONTAP 시스템을 사용하든 ONTAP를 시뮬레이션하든 상관없이 자동화 솔루션을 구축하기 전에 환경을 준비해야 합니다.

필요에 따라 **Simulate ONTAP**를 설치 및 구성합니다

ONTAP 시뮬레이터를 통해 이 솔루션을 배포하려면 Simulate ONTAP를 다운로드하여 실행해야 합니다.

시작하기 전에

- Simulate ONTAP를 실행하는 데 사용할 VMware 하이퍼바이저를 다운로드하고 설치해야 합니다.
  - Windows 또는 Linux OS를 사용하는 경우 VMware Workstation을 사용하십시오.
  - Mac OS를 사용하는 경우 VMware Fusion를 사용합니다.

> Mac OS를 사용하는 경우 Intel 프로세서가 있어야 합니다.

단계

로컬 환경에 두 개의 ONTAP 시뮬레이터를 설치하려면 다음 절차를 따르십시오.

1. 에서 Simulate ONTAP 를 "NetApp Support 사이트"다운로드합니다.

두 개의 ONTAP 시뮬레이터를 설치하더라도 소프트웨어 복사본 하나만 다운로드하면 됩니다.

2. 아직 실행 중이 아닌 경우 VMware 애플리케이션을 시작합니다.
3. 다운로드한 시뮬레이터 파일을 찾아 마우스 오른쪽 버튼을 클릭하여 VMware 애플리케이션에서 엽니다.
4. 첫 번째 ONTAP 인스턴스의 이름을 설정합니다.
5. 시뮬레이터가 부팅될 때까지 기다린 다음 지침에 따라 단일 노드 클러스터를 생성합니다.

   두 번째 ONTAP 인스턴스에 대해 이 단계를 반복합니다.

6. 필요에 따라 전체 디스크 보완을 추가합니다.

   각 클러스터에서 다음 명령을 실행합니다.

```
security unlock -username <user_01>
security login password -username <user_01>
set -priv advanced
systemshell local
disk assign -all -node <Cluster-01>-01
```

ONTAP 시스템의 상태입니다

사내 시스템 또는 ONTAP 시뮬레이터를 통해 ONTAP 시스템의 초기 상태를 확인해야 합니다.

다음 ONTAP 시스템 요구 사항이 충족되는지 확인합니다.

- 아직 정의된 클러스터가 없는 상태에서 ONTAP가 설치되고 실행 중입니다.
- ONTAP가 부팅되고 클러스터에 액세스하기 위한 IP 주소가 표시됩니다.
- 네트워크에 연결할 수 있습니다.
- 관리자 자격 증명이 있습니다.
- MOTD(Message of the Day) 배너에 관리 주소가 표시됩니다.

필요한 자동화 소프트웨어를 설치합니다

이 섹션에서는 Ansible을 설치하고 구축을 위한 자동화 솔루션을 준비하는 방법에 대한 정보를 제공합니다.

Ansible 설치

Linux 또는 Windows 시스템에 Ansible을 설치할 수 있습니다.

Ansible이 ONTAP 클러스터와 통신하기 위해 사용하는 기본 통신 방법은 SSH입니다.

Ansible 설치에 대해서는 을 "NetApp 및 Ansible 시작하기: Ansible을 설치하십시오"참조하십시오.

시스템의 제어 노드에 Ansible을 설치해야 합니다.

자동화 솔루션을 다운로드하고 준비합니다

다음 단계에 따라 배포를 위한 자동화 솔루션을 다운로드하고 준비할 수 있습니다.

1. 다운로드 "ONTAP - 1일차 및 1일차; 상태 점검" 콘솔 웹 UI를 통한 자동화 솔루션. 솔루션은 다음과 같이 패키지됩니다. `ONTAP_DAY0_DAY1.zip`.
2. zip 폴더의 압축을 풀고 Ansible 환경 내의 제어 노드에서 원하는 위치로 파일을 복사합니다.

초기 **Ansible** 프레임워크 구성

Ansible 프레임워크의 초기 구성 수행:

1. 로 이동합니다 `playbooks/inventory/group_vars/all`.
2. 파일 암호 해독 `vault.yml`:

   ```
   ansible-vault decrypt playbooks/inventory/group_vars/all/vault.yml
   ```

   볼트 암호를 입력하라는 메시지가 나타나면 다음 임시 암호를 입력합니다.

   ```
   NetApp123!
   ```

   > "NetApp123!"는 파일과 해당 볼트 암호를 해독하기 위한 임시 암호입니다. `vault.yml` 처음 사용한 후에는 * 자신의 암호를 사용하여 * 파일을 암호화해야 합니다.

3. 다음 Ansible 파일을 수정합니다.
   - `clusters.yml` - 환경에 맞게 이 파일의 값을 수정합니다.
   - `vault.yml` - 파일을 해독한 후 사용자 환경에 맞게 ONTAP 클러스터, 사용자 이름 및 암호 값을 수정합니다.
   - `cfg.yml` - 에 대한 파일 경로를 `log2file` 설정하고 `cfg` 을(를) 표시하려면 을 `raw_service_request` (를 `True`) 로 설정합니다 `show_request`.

     ```
     `raw_service_request`로그 파일과 실행 중에 변수가 표시됩니다.
     ```

     > 나열된 각 파일에는 요구 사항에 따라 수정하는 방법에 대한 지침이 포함된 설명이 포함되어 있습니다.

4. 파일 다시 암호화 `vault.yml`:

   ```
   ansible-vault encrypt playbooks/inventory/group_vars/all/vault.yml
   ```

   > 암호화 시 볼트에 대한 새 암호를 선택하라는 메시지가 표시됩니다.

5. 유효한 Python 인터프리터를 찾아 `playbooks/inventory/hosts` 설정합니다.
6. 서비스 구축 `framework_test`:

   다음 명령을 실행하면 `na_ontap_info` 값이 인 `cluster_identity_info` 모듈이 `gather_subset`

실행됩니다. 이렇게 하면 기본 구성이 올바른지 확인하고 클러스터와 통신할 수 있는지 확인합니다.

```
ansible-playbook -i inventory/hosts site.yml -e
cluster_name=<CLUSTER_NAME>
-e logic_operation=framework-test
```

각 클러스터에 대해 명령을 실행합니다.

성공하면 다음 예와 유사한 출력이 표시됩니다.

```
PLAY RECAP
*********************************************************************
*********
localhost : ok=12 changed=1 unreachable=0 failed=0 skipped=6
The key is 'rescued=0' and 'failed=0'..
```

솔루션을 사용하여 **ONTAP** 클러스터를 구축합니다

준비 및 계획을 완료하면 ONTAP Day 0/1 솔루션을 사용하여 Ansible을 사용하여 ONTAP 클러스터를 신속하게 구성할 수 있습니다.

이 섹션의 단계 중 언제든지 요청을 실제로 실행하는 대신 테스트할 수 있습니다. 요청을 테스트하려면 명령줄에서 플레이북을 로 `logic.yml` 변경합니다 `site.yml`.

> `docs/tutorial-requests.txt`위치에는 이 절차 전반에 걸쳐 사용된 모든 서비스 요청의 최종 버전이 포함되어 있습니다. 서비스 요청을 실행하는 데 문제가 있는 경우 파일에서 `playbooks/inventory/group_vars/all/tutorial-requests.yml` 위치로 관련 요청을 복사하고 필요에 따라 하드 코드된 값(IP 주소, 집계 이름 등)을 수정할 수 `tutorial-requests.txt` 있습니다. 그러면 요청을 성공적으로 실행할 수 있습니다.

시작하기 전에

- Ansible을 설치해야 합니다.
- ONTAP DAY 0/1 솔루션을 다운로드하고 Ansible 제어 노드의 원하는 위치에 폴더를 추출해야 합니다.
- ONTAP 시스템 상태는 요구사항을 충족해야 하고 필요한 자격 증명이 있어야 합니다.
- 섹션에 요약된 모든 필수 작업을 완료해야 "준비"합니다.

> 이 솔루션의 예에서는 두 클러스터의 이름으로 "Cluster_01"과 "Cluster_02"를 사용합니다. 이러한 값은 사용자 환경에서 클러스터의 이름으로 대체해야 합니다.

1단계**:** 초기 클러스터 구성

이 단계에서는 몇 가지 초기 클러스터 구성 단계를 수행해야 합니다.

단계

1. 위치로 `playbooks/inventory/group_vars/all/tutorial-requests.yml` 이동하여 `cluster_initial` 파일의 요청을 검토합니다. 사용자 환경에 필요한 변경 작업을 수행합니다.
2. 서비스 요청에 대한 폴더에 파일을 만듭니다 `logic-tasks`. 예를 들어, 이라는 파일을 `cluster_initial.yml` 만듭니다.

   다음 줄을 새 파일에 복사합니다.

   ```
   - name: Validate required inputs
     ansible.builtin.assert:
       that:
       - service is defined

   - name: Include data files
     ansible.builtin.include_vars:
       file:   "{{ data_file_name }}.yml"
     loop:
     - common-site-stds
     - user-inputs
     - cluster-platform-stds
     - vserver-common-stds
     loop_control:
       loop_var:    data_file_name

   - name: Initial cluster configuration
     set_fact:
       raw_service_request:
   ```

3. `raw_service_request`변수를 정의합니다.

   다음 옵션 중 하나를 사용하여 폴더에 만든 파일의 `logic-tasks` 변수를 `cluster_initial.yml` 정의할 수 `raw_service_request` 있습니다.

   - * 옵션 1 *: 변수를 수동으로 `raw_service_request` 정의합니다.

     `tutorial-requests.yml`편집기를 사용하여 파일을 열고 11줄의 콘텐츠를 165줄로 복사합니다. 다음 예제와 같이 새 파일의 변수 `cluster_initial.yml` 아래에 내용을 붙여 넣습니다 `raw service request`.

```
3   # This file contains the final version of the various service
4   # requests used throughout the tutorial in TUTORIAL.md.
5   #-----------------------------------------------------------------
6   #-----------------------------------------------------------------
7   # cluster_initial:
8   #
9   #-----------------------------------------------------------------
10  cluster_initial:
11    service:          cluster_initial
12    operation:        create
13    std_name:         none
14    req_details:
15
16      ontap_aggr:
17      - hostname:                   "{{ cluster_name }}"
18        disk_count:                 24
19        name:                       n01_aggr1
20        nodes:                      "{{ cluster_name }}-01"
```

예제 보기

`cluster_initial.yml`예제 파일:

```
- name: Validate required inputs
  ansible.builtin.assert:
    that:
    - service is defined

- name: Include data files
  ansible.builtin.include_vars:
    file:   "{{ data_file_name }}.yml"
  loop:
  - common-site-stds
  - user-inputs
  - cluster-platform-stds
  - vserver-common-stds
  loop_control:
    loop_var:    data_file_name

- name: Initial cluster configuration
  set_fact:
    raw_service_request:
     service:          cluster_initial
     operation:         create
     std_name:           none
     req_details:

      ontap_aggr:
      - hostname:                   "{{ cluster_name }}"
        disk_count:                 24
        name:                       n01_aggr1
        nodes:                      "{{ cluster_name }}-01"
        raid_type:                  raid4

      - hostname:                   "{{ peer_cluster_name }}"
        disk_count:                 24
        name:                       n01_aggr1
        nodes:                      "{{ peer_cluster_name }}-01"
        raid_type:                  raid4

      ontap_license:
      - hostname:                   "{{ cluster_name }}"
        license_codes:
```

```
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA

    - hostname:                   "{{ peer_cluster_name }}"
      license_codes:
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
```

```
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA

ontap_motd:
- hostname:                   "{{ cluster_name }}"
  vserver:                    "{{ cluster_name }}"
  message:                    "New MOTD"

- hostname:                   "{{ peer_cluster_name }}"
  vserver:                    "{{ peer_cluster_name }}"
  message:                    "New MOTD"

ontap_interface:
- hostname:                   "{{ cluster_name }}"
  vserver:                    "{{ cluster_name }}"
  interface_name:             ic01
  role:                       intercluster
  address:                    10.0.0.101
  netmask:                    255.255.255.0
  home_node:                  "{{ cluster_name }}-01"
  home_port:                  e0c
  ipspace:                    Default
  use_rest:                   never

- hostname:                   "{{ cluster_name }}"
  vserver:                    "{{ cluster_name }}"
  interface_name:             ic02
  role:                       intercluster
  address:                    10.0.0.101
  netmask:                    255.255.255.0
```

```
      home_node:                    "{{ cluster_name }}-01"
      home_port:                    e0c
      ipspace:                      Default
      use_rest:                     never

    - hostname:                     "{{ peer_cluster_name }}"
      vserver:                      "{{ peer_cluster_name }}"
      interface_name:               ic01
      role:                         intercluster
      address:                      10.0.0.101
      netmask:                      255.255.255.0
      home_node:                    "{{ peer_cluster_name }}-01"
      home_port:                    e0c
      ipspace:                      Default
      use_rest:                     never

    - hostname:                     "{{ peer_cluster_name }}"
      vserver:                      "{{ peer_cluster_name }}"
      interface_name:               ic02
      role:                         intercluster
      address:                      10.0.0.101
      netmask:                      255.255.255.0
      home_node:                    "{{ peer_cluster_name }}-01"
      home_port:                    e0c
      ipspace:                      Default
      use_rest:                     never

    ontap_cluster_peer:
    - hostname:                     "{{ cluster_name }}"
      dest_cluster_name:            "{{ peer_cluster_name }}"
      dest_intercluster_lifs:       "{{ peer_lifs }}"
      source_cluster_name:          "{{ cluster_name }}"
      source_intercluster_lifs:     "{{ cluster_lifs }}"
      peer_options:
        hostname:                   "{{ peer_cluster_name }}"
```

- * 옵션 2 * : Jinja 템플릿을 사용하여 요청을 정의합니다.

  다음 Jinja 템플릿 형식을 사용하여 값을 가져올 수도 `raw_service_request` 있습니다.

```
raw_service_request: "{{ cluster_initial }}"
```

4. 첫 번째 클러스터에 대한 초기 클러스터 구성을 수행합니다.

```
ansible-playbook -i inventory/hosts site.yml -e
cluster_name=<Cluster_01>
```

계속하기 전에 오류가 없는지 확인하십시오.

5. 두 번째 클러스터에 대해 명령을 반복합니다.

```
ansible-playbook -i inventory/hosts site.yml -e
cluster_name=<Cluster_02>
```

두 번째 클러스터에 오류가 없는지 확인합니다.

Ansible 출력 시작 부분으로 스크롤하면 다음 예제에 표시된 것처럼 프레임워크로 전송된 요청을 볼 수 있습니다.

예제 보기

```
TASK [Show the raw_service_request]
********************************************************************
****************************************
ok: [localhost] => {
    "raw_service_request": {
        "operation": "create",
        "req_details": {
            "ontap_aggr": [
                {
                    "disk_count": 24,
                    "hostname": "Cluster_01",
                    "name": "n01_aggr1",
                    "nodes": "Cluster_01-01",
                    "raid_type": "raid4"
                }
            ],
            "ontap_license": [
                {
                    "hostname": "Cluster_01",
                    "license_codes": [
                        "XXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
```

```
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA"
                    ]
                }
            ],
            "ontap_motd": [
                {
                    "hostname": "Cluster_01",
                    "message": "New MOTD",
                    "vserver": "Cluster_01"
                }
            ]
        },
        "service": "cluster_initial",
        "std_name": "none"
    }
}
```

6. 각 ONTAP 인스턴스에 로그인하고 요청이 성공했는지 확인합니다.

2단계: 인터클러스터 **LIF**를 구성합니다

이제 LIF 정의를 요청에 추가하고 `ontap_interface` 마이크로서비스를 정의하여 인터클러스터 LIF를 구성할 수 있습니다 `cluster_initial`.

서비스 정의와 요청이 함께 작동하여 조치를 결정합니다.

- 서비스 정의에 없는 마이크로서비스에 대한 서비스 요청을 제공하면 요청이 실행되지 않습니다.
- 서비스 정의에 정의된 마이크로서비스가 하나 이상 있지만 요청에서 생략된 서비스 요청을 제공하는 경우 요청이 실행되지 않습니다.

  `execution.yml`Playbook은 나열된 순서대로 마이크로서비스 목록을 스캔하여 서비스 정의를 평가합니다.

- 요청에 마이크로서비스 정의에 포함된 항목과 일치하는 사전 키가 있는 항목이 있으면 `args` 요청이 실행됩니다.

- 서비스 요청에 일치하는 항목이 없으면 오류 없이 요청을 건너뜁니다.

단계

1. `cluster_initial.yml`이전에 만든 파일로 이동하고 요청 정의에 다음 행을 추가하여 요청을 수정합니다.

```
    ontap_interface:
    - hostname:                    "{{ cluster_name }}"
      vserver:                     "{{ cluster_name }}"
      interface_name:              ic01
      role:                        intercluster
      address:                     <ip_address>
      netmask:                     <netmask_address>
      home_node:                   "{{ cluster_name }}-01"
      home_port:                   e0c
      ipspace:                     Default
      use_rest:                    never

    - hostname:                    "{{ cluster_name }}"
      vserver:                     "{{ cluster_name }}"
      interface_name:              ic02
      role:                        intercluster
      address:                     <ip_address>
      netmask:                     <netmask_address>
      home_node:                   "{{ cluster_name }}-01"
      home_port:                   e0c
      ipspace:                     Default
      use_rest:                    never

    - hostname:                    "{{ peer_cluster_name }}"
      vserver:                     "{{ peer_cluster_name }}"
      interface_name:              ic01
      role:                        intercluster
      address:                     <ip_address>
      netmask:                     <netmask_address>
      home_node:                   "{{ peer_cluster_name }}-01"
      home_port:                   e0c
      ipspace:                     Default
      use_rest:                    never

    - hostname:                    "{{ peer_cluster_name }}"
      vserver:                     "{{ peer_cluster_name }}"
      interface_name:              ic02
      role:                        intercluster
      address:                     <ip_address>
      netmask:                     <netmask_address>
      home_node:                   "{{ peer_cluster_name }}-01"
      home_port:                   e0c
      ipspace:                     Default
      use_rest:                    never
```

2. 다음 명령을 실행합니다.

```
ansible-playbook -i inventory/hosts  site.yml -e
cluster_name=<Cluster_01> -e peer_cluster_name=<Cluster_02>
```

3. 각 인스턴스에 로그인하여 LIF가 클러스터에 추가되었는지 확인합니다.

예제 보기

```
Cluster_01::> net int show
  (network interface show)
            Logical    Status     Network            Current
Current Is
Vserver     Interface  Admin/Oper Address/Mask       Node
Port    Home
----------- ---------- ---------- ------------------ -------------
------- ----
Cluster_01
            Cluster_01-01_mgmt up/up 10.0.0.101/24   Cluster_01-01
e0c     true
            Cluster_01-01_mgmt_auto up/up 10.101.101.101/24
Cluster_01-01 e0c true
            cluster_mgmt up/up    10.0.0.110/24      Cluster_01-01
e0c     true
5 entries were displayed.
```

출력은 LIF가 추가되지 않았음을 보여줍니다 *. 이는 마이크로 서비스를 `services.yml` 파일에 정의해야 하기 `ontap_interface` 때문입니다.

4. LIF가 변수에 추가되었는지 확인 `raw_service_request`

예제 보기

다음 예제는 LIF가 요청에 추가되었음을 보여줍니다.


```
            "ontap_interface": [
                 {
                     "address": "10.0.0.101",
                     "home_node": "Cluster_01-01",
                     "home_port": "e0c",
                     "hostname": "Cluster_01",
                     "interface_name": "ic01",
                     "ipspace": "Default",
                     "netmask": "255.255.255.0",
                     "role": "intercluster",
                     "use_rest": "never",
                     "vserver": "Cluster_01"
                 },
                 {
                     "address": "10.0.0.101",
                     "home_node": "Cluster_01-01",
                     "home_port": "e0c",
                     "hostname": "Cluster_01",
                     "interface_name": "ic02",
                     "ipspace": "Default",
                     "netmask": "255.255.255.0",
                     "role": "intercluster",
                     "use_rest": "never",
                     "vserver": "Cluster_01"
                 },
                 {
                     "address": "10.0.0.101",
                     "home_node": "Cluster_02-01",
                     "home_port": "e0c",
                     "hostname": "Cluster_02",
                     "interface_name": "ic01",
                     "ipspace": "Default",
                     "netmask": "255.255.255.0",
                     "role": "intercluster",
                     "use_rest": "never",
                     "vserver": "Cluster_02"
                 },
                 {
                     "address": "10.0.0.126",
                     "home_node": "Cluster_02-01",
                     "home_port": "e0c",
                     "hostname": "Cluster_02",
```

```
                    "interface_name": "ic02",
                    "ipspace": "Default",
                    "netmask": "255.255.255.0",
                    "role": "intercluster",
                    "use_rest": "never",
                    "vserver": "Cluster_02"
                }
            ],
```

5. `ontap_interface services.yml`파일에서 마이크로서비스를 `cluster_initial` 정의합니다.

   마이크로서비스를 정의하려면 파일에 다음 줄을 복사합니다.

```
        - name: ontap_interface
          args: ontap_interface
          role: na/ontap_interface
```

6. 이제 마이크로 서비스가 요청과 `services.yml` 파일에 정의되었으므로 `ontap_interface` 요청을 다시 실행합니다.

```
ansible-playbook -i inventory/hosts  site.yml -e
cluster_name=<Cluster_01> -e peer_cluster_name=<Cluster_02>
```

7. 각 ONTAP 인스턴스에 로그인하여 LIF가 추가되었는지 확인합니다.

3단계: 필요에 따라 여러 클러스터를 구성합니다

필요한 경우 동일한 요청으로 여러 클러스터를 구성할 수 있습니다. 요청을 정의할 때 각 클러스터에 대해 가변 이름을 제공해야 합니다.

단계

1. 파일에 두 번째 클러스터에 대한 항목을 `cluster_initial.yml` 추가하여 동일한 요청에서 두 클러스터를 구성합니다.

   다음 예제에서는 `ontap_aggr` 두 번째 항목이 추가된 후 필드를 표시합니다.

```
ontap_aggr:
 - hostname:                    "{{ cluster_name }}"
   disk_count:                  24
   name:                        n01_aggr1
   nodes:                       "{{ cluster_name }}-01"
   raid_type:                   raid4

 - hostname:                    "{{ peer_cluster_name }}"
   disk_count:                  24
   name:                        n01_aggr1
   nodes:                       "{{ peer_cluster_name }}-01"
   raid_type:                   raid4
```

2. 에서 다른 모든 항목에 대한 변경 내용을 `cluster_initial` 적용합니다.
3. 다음 줄을 파일에 복사하여 요청에 클러스터 피어링을 추가합니다.

```
ontap_cluster_peer:
- hostname:                    "{{ cluster_name }}"
  dest_cluster_name:            "{{ cluster_peer }}"
  dest_intercluster_lifs:       "{{ peer_lifs }}"
  source_cluster_name:          "{{ cluster_name }}"
  source_intercluster_lifs:     "{{ cluster_lifs }}"
  peer_options:
    hostname:                   "{{ cluster_peer }}"
```

4. Ansible 요청 실행:

```
ansible-playbook -i inventory/hosts -e cluster_name=<Cluster_01>
site.yml -e peer_cluster_name=<Cluster_02> -e
cluster_lifs=<cluster_lif_1_IP_address,cluster_lif_2_IP_address>
-e peer_lifs=<peer_lif_1_IP_address,peer_lif_2_IP_address>
```

4단계: 초기 SVM 구성

절차의 이 단계에서는 클러스터에서 SVM을 구성합니다.

단계

1. `svm_initial`파일에서 요청을 `tutorial-requests.yml` 업데이트하여 SVM 및 SVM 피어 관계 구성

   다음을 구성해야 합니다.

   - SVM은

- SVM 피어 관계
- 각 SVM의 SVM 인터페이스

2. 요청 정의에서 변수 정의를 `svm_initial` 업데이트합니다. 다음 변수 정의를 수정해야 합니다.
   - `cluster_name`
   - `vserver_name`
   - `peer_cluster_name`
   - `peer_vserver`

     정의를 업데이트하려면 정의에 대한 `svm_initial` 뒤에 * '{}' * 를 `req_details` 제거하고 올바른 정의를 추가하십시오.

3. 서비스 요청에 대한 폴더에 파일을 만듭니다 `logic-tasks`. 예를 들어, 이라는 파일을 `svm_initial.yml` 만듭니다.

   다음 줄을 파일에 복사합니다.

```
- name: Validate required inputs
  ansible.builtin.assert:
    that:
    - service is defined

- name: Include data files
  ansible.builtin.include_vars:
    file:   "{{ data_file_name }}.yml"
  loop:
  - common-site-stds
  - user-inputs
  - cluster-platform-stds
  - vserver-common-stds
  loop_control:
    loop_var:    data_file_name

- name: Initial SVM configuration
  set_fact:
    raw_service_request:
```

4. `raw_service_request`변수를 정의합니다.

   다음 옵션 중 하나를 사용하여 `logic-tasks` 폴더에서 변수를 `svm_initial` 정의할 수 `raw_service_request` 있습니다.

   - * 옵션 1 *: 변수를 수동으로 `raw_service_request` 정의합니다.

`tutorial-requests.yml`편집기를 사용하여 파일을 열고 179줄의 내용을 222줄로 복사합니다. 다음 예제와 같이 새 파일의 변수 `svm_initial.yml` 아래에 내용을 붙여 넣습니다 `raw service request`.

```
179     service:          svm_initial
181     std_name:         none
182     req_details:

184       ontap_vserver:
185       - hostname:                   "{{ cluster_name }}"
186         name:                       "{{ vserver_name }}"
187         root_volume_aggregate:      n01_aggr1

189       - hostname:                   "{{ peer_cluster_name }}"
190         name:                       "{{ peer_vserver }}"
191         root_volume_aggregate:      n01_aggr1
```

예제 보기

`svm_initial.yml`예제 파일:

```
- name: Validate required inputs
  ansible.builtin.assert:
    that:
    - service is defined

- name: Include data files
  ansible.builtin.include_vars:
    file:   "{{ data_file_name }}.yml"
  loop:
  - common-site-stds
  - user-inputs
  - cluster-platform-stds
  - vserver-common-stds
  loop_control:
    loop_var:    data_file_name

- name: Initial SVM configuration
  set_fact:
    raw_service_request:
     service:          svm_initial
     operation:        create
     std_name:         none
     req_details:

      ontap_vserver:
      - hostname:                   "{{ cluster_name }}"
        name:                       "{{ vserver_name }}"
        root_volume_aggregate:      n01_aggr1

      - hostname:                   "{{ peer_cluster_name }}"
       name:                       "{{ peer_vserver }}"
       root_volume_aggregate:      n01_aggr1

      ontap_vserver_peer:
      - hostname:                   "{{ cluster_name }}"
        vserver:                    "{{ vserver_name }}"
        peer_vserver:               "{{ peer_vserver }}"
        applications:               snapmirror
        peer_options:
          hostname:                 "{{ peer_cluster_name }}"
```

```
      ontap_interface:
      - hostname:                    "{{ cluster_name }}"
        vserver:                     "{{ vserver_name }}"
        interface_name:              data01
        role:                        data
        address:                     10.0.0.200
        netmask:                     255.255.255.0
        home_node:                   "{{ cluster_name }}-01"
        home_port:                   e0c
        ipspace:                     Default
        use_rest:                    never

      - hostname:                    "{{ peer_cluster_name }}"
        vserver:                     "{{ peer_vserver }}"
        interface_name:              data01
        role:                        data
        address:                     10.0.0.201
        netmask:                     255.255.255.0
        home_node:                   "{{ peer_cluster_name }}-01"
        home_port:                   e0c
        ipspace:                     Default
        use_rest:                    never
```

- * 옵션 2 * : Jinja 템플릿을 사용하여 요청을 정의합니다.

  다음 Jinja 템플릿 형식을 사용하여 값을 가져올 수도 `raw_service_request` 있습니다.

```
raw_service_request: "{{ svm_initial }}"
```

5. 다음 요청을 실행합니다.

```
ansible-playbook -i inventory/hosts -e cluster_name=<Cluster_01> -e
peer_cluster_name=<Cluster_02> -e peer_vserver=<SVM_02>  -e
vserver_name=<SVM_01> site.yml
```

6. 각 ONTAP 인스턴스에 로그인하고 구성을 검증합니다.
7. SVM 인터페이스를 추가합니다.

`ontap_interface` `services.yml`파일에서 서비스를 `svm_initial` 정의하고 요청을 다시 실행합니다.

```
ansible-playbook -i inventory/hosts -e cluster_name=<Cluster_01> -e
peer_cluster_name=<Cluster_02> -e peer_vserver=<SVM_02>  -e
vserver_name=<SVM_01> site.yml
```

8. 각 ONTAP 인스턴스에 로그인하고 SVM 인터페이스가 구성되었는지 확인합니다.

5단계**:** 필요에 따라 서비스 요청을 동적으로 정의합니다

이전 단계에서 `raw_service_request` 변수는 하드 코딩됩니다. 이 기능은 학습, 개발 및 테스트에 유용합니다. 서비스 요청을 동적으로 생성할 수도 있습니다.

다음 섹션에서는 Required 를 상위 시스템과 통합하지 않으려는 경우 동적으로 생성할 수 있는 옵션을 `raw_service_request` 제공합니다.

- 명령에서 변수가 정의되지 않은 `logic.yml` 경우 `logic_operation` 파일은 폴더에서 파일을 가져오지 `logic-tasks` 않습니다. 이는 `raw_service_request` Ansible 외부에서 정의되고 실행 프레임워크에 제공되어야 함을 의미합니다.
- 폴더의 작업 파일 이름은 `logic-tasks` .yml 확장자가 없는 변수 값과 일치해야 `logic_operation` 합니다.
- 폴더의 작업 파일이 `logic-tasks` 동적으로 을 `raw_service_request`정의합니다. 단, 유효한 를 `raw_service_request` 관련 파일의 마지막 작업으로 정의해야 합니다.

서비스 요청을 동적으로 정의하는 방법

서비스 요청을 동적으로 정의하기 위해 논리 작업을 적용하는 방법에는 여러 가지가 있습니다. 이러한 옵션 중 일부는 다음과 같습니다.

- 폴더의 Ansible 작업 파일 사용 `logic-tasks`
- varaible 로 변환하기에 적합한 데이터를 반환하는 사용자 지정 역할 호출 `raw_service_request`
- Ansible 환경 외부에서 다른 툴을 호출하여 필요한 데이터를 제공합니다. 예를 들어, Active IQ Unified Manager에 대한 REST API 호출

다음 명령 예는 파일을 사용하여 각 클러스터에 대한 서비스 요청을 동적으로 `tutorial-requests.yml` 정의합니다.

```
ansible-playbook -i inventory/hosts -e cluster2provision=Cluster_01
-e logic_operation=tutorial-requests site.yml
```

```
ansible-playbook -i inventory/hosts -e cluster2provision=Cluster_02
-e logic_operation=tutorial-requests site.yml
```

6단계**: ONTAP Day 0/1** 솔루션을 배포합니다

이 단계에서는 다음을 이미 완료해야 합니다.

- 요구 사항에 따라 의 모든 파일을 검토하고 수정했습니다. `playbooks/inventory/group_vars/all` 각 파일에는 변경 작업에 도움이 되는 자세한 설명이 있습니다.
- 필요한 작업 파일을 `logic-tasks` 디렉토리에 추가했습니다.
- 필요한 데이터 파일을 `playbook/vars` 디렉터리에 추가했습니다.

다음 명령을 사용하여 ONTAP DAY 0/1 솔루션을 배포하고 배포 상태를 확인합니다.

> 이 단계에서는 이미 파일을 암호 해독하고 수정했으며 `vault.yml` 새 암호로 암호화해야 합니다.

- ONTAP Day 0 서비스 실행:

```
ansible-playbook -i playbooks/inventory/hosts playbooks/site.yml -e
logic_operation=cluster_day_0 -e service=cluster_day_0 -vvvv --ask-vault
-pass <your_vault_password>
```

- ONTAP Day 1 서비스 실행:

```
ansible-playbook -i playbooks/inventory/hosts playbooks/site.yml -e
logic_operation=cluster_day_1 -e service=cluster_day_0 -vvvv --ask-vault
-pass <your_vault_password>
```

- 클러스터 전체 설정 적용:

```
ansible-playbook -i playbooks/inventory/hosts playbooks/site.yml -e
logic_operation=cluster_wide_settings -e service=cluster_wide_settings
-vvvv --ask-vault-pass <your_vault_password>
```

- 상태 점검 실행:

```
ansible-playbook -i playbooks/inventory/hosts playbooks/site.yml -e
logic_operation=health_checks -e service=health_checks -e
enable_health_reports=true -vvvv --ask-vault-pass <your_vault_password>
```

ONTAP Day 0/1 솔루션을 사용자 지정합니다

요구사항에 맞게 ONTAP Day 0/1 솔루션을 사용자 정의하려면 Ansible 역할을 추가하거나 변경할 수 있습니다.

역할은 Ansible 프레임워크 내의 마이크로서비스를 나타냅니다. 각 마이크로서비스는 하나의 작업을 수행합니다. 예를 들어 ONTAP Day 0은 여러 마이크로서비스를 포함하는 서비스입니다.

Ansible 역할을 추가합니다

Ansible 역할을 추가하여 환경에 맞게 솔루션을 사용자 지정할 수 있습니다. 필수 역할은 Ansible 프레임워크 내의 서비스 정의에 의해 정의됩니다.

마이크로서비스로 사용하려면 역할이 다음 요구 사항을 충족해야 합니다.

- 변수의 인수 목록을 수락합니다 `args`.
- 각 블록에 대한 특정 요구 사항이 있는 Ansible "블록, 구조, 항상" 구조를 사용합니다.
- 단일 Ansible 모듈을 사용하여 블록 내의 단일 작업을 정의합니다.
- 이 섹션에 설명된 요구 사항에 따라 사용 가능한 모든 모듈 매개 변수를 구현합니다.

필수 마이크로서비스 구조

각 역할은 다음 변수를 지원해야 합니다.

- `mode`: 모드가 역할로 설정된 경우 `test` 을(를) 가져오려고 시도하면 `test.yml` 실제로 해당 역할이 실행되지 않고 어떤 작업을 수행하는지를 보여줍니다.

  > 특정 상호 의존성 때문에 이를 항상 구현할 수는 없습니다.

- `status`: Playbook 실행의 전반적인 상태. 값이 역할로 설정되지 않으면 `success` 실행되지 않습니다.
- `args` : 역할 매개 변수 이름과 일치하는 키가 있는 역할별 사전 목록입니다.
- `global_log_messages`: 플레이북 실행 중 로그 메시지를 수집합니다. 역할이 실행될 때마다 하나의 항목이 생성됩니다.
- `log_name`: 항목 내에서 역할을 참조하는 데 사용되는 이름입니다 `global_log_messages`.
- `task_descr`: 역할에 대한 간단한 설명입니다.
- `service_start_time`: 각 역할이 실행되는 시간을 추적하는 데 사용되는 타임 스탬프입니다.
- `playbook_status`: Ansible 플레이북의 상태.
- `role_result`: 역할 출력을 포함하고 항목 내의 각 메시지에 포함되는 변수입니다. `global_log_messages`

역할 구조의 예

다음 예제에서는 마이크로서비스를 구현하는 역할의 기본 구조를 제공합니다. 이 예제에서는 설정을 위해 변수를 변경해야 합니다.

예제 보기

기본 역할 구조:

```
- name:  Set some role attributes
  set_fact:
    log_name:     "<LOG_NAME>"
    task_descr:   "<TASK_DESCRIPTION>"

-  name: "{{ log_name }}"
   block:
      -  set_fact:
            service_start_time: "{{ lookup('pipe', 'date
+%Y%m%d%H%M%S') }}"

      -  name: "Provision the new user"
         <MODULE_NAME>:

#-------------------------------------------------------------
            # COMMON ATTRIBUTES

#-------------------------------------------------------------
            hostname:            "{{
clusters[loop_arg['hostname']]['mgmt_ip'] }}"
            username:            "{{
clusters[loop_arg['hostname']]['username'] }}"
            password:            "{{
clusters[loop_arg['hostname']]['password'] }}"

            cert_filepath:       "{{ loop_arg['cert_filepath']
| default(omit) }}"
            feature_flags:       "{{ loop_arg['feature_flags']
| default(omit) }}"
            http_port:           "{{ loop_arg['http_port']
| default(omit) }}"
            https:               "{{ loop_arg['https']
| default('true') }}"
            ontapi:              "{{ loop_arg['ontapi']
| default(omit) }}"
            key_filepath:        "{{ loop_arg['key_filepath']
| default(omit) }}"
            use_rest:            "{{ loop_arg['use_rest']
| default(omit) }}"
            validate_certs:      "{{ loop_arg['validate_certs']
| default('false') }}"
```

```
            <MODULE_SPECIFIC_PARAMETERS>

#-----------------------------------------------------------------
            # REQUIRED ATTRIBUTES

#-----------------------------------------------------------------
            required_parameter:     "{{ loop_arg['required_parameter']
}}"

#-----------------------------------------------------------------
            # ATTRIBUTES w/ DEFAULTS

#-----------------------------------------------------------------
            defaulted_parameter:    "{{ loop_arg['defaulted_parameter']
| default('default_value') }}"

#-----------------------------------------------------------------
            # OPTIONAL ATTRIBUTES

#-----------------------------------------------------------------
            optional_parameter:     "{{ loop_arg['optional_parameter']
| default(omit) }}"
         loop:    "{{ args }}"
         loop_control:
            loop_var:   loop_arg
         register:   role_result

   rescue:
      -  name: Set role status to FAIL
         set_fact:
            playbook_status:   "failed"

   always:
      -  name: add log msg
         vars:
            role_log:
               role: "{{ log_name }}"
               timestamp:
                  start_time: "{{service_start_time}}"
                  end_time: "{{ lookup('pipe', 'date +%Y-%m-
%d@%H:%M:%S') }}"
               service_status: "{{ playbook_status }}"
               result: "{{role_result}}"
         set_fact:
            global_log_msgs:   "{{ global_log_msgs + [ role_log ] }}"
```

예제 역할에 사용되는 변수:

- `<NAME>`: 각 마이크로서비스에 대해 제공되어야 하는 대체 가능한 값입니다.
- `<LOG_NAME>`: 로깅 목적으로 사용되는 역할의 짧은 형식 이름입니다. `ONTAP_VOLUME`예를 들어,
- `<TASK_DESCRIPTION>`: 마이크로 서비스가 수행하는 작업에 대한 간략한 설명.
- `<MODULE_NAME>`: 작업에 대한 Ansible 모듈 이름.

> 최상위 `execute.yml` Playbook에서 `netapp.ontap` 컬렉션을 지정합니다. 모듈이 컬렉션의 일부인 경우 `netapp.ontap` 모듈 이름을 완전히 지정할 필요가 없습니다.

- `<MODULE_SPECIFIC_PARAMETERS>`: 마이크로 서비스를 구현하는 데 사용되는 모듈에 고유한 Ansible 모듈 매개 변수입니다. 다음 목록에서는 매개 변수 유형과 매개 변수를 그룹화하는 방법을 설명합니다.
  - 필수 매개변수: 모든 필수 매개변수는 기본값 없이 지정됩니다.
  - 마이크로서비스에 고유한 기본값을 갖는 매개 변수(모듈 설명서에 지정된 기본값과 다름)
  - 나머지 모든 매개 변수는 `default(omit)` 기본값으로 사용됩니다.

다중 수준 사전을 모듈 매개 변수로 사용

일부 NetApp에서 제공하는 Ansible 모듈은 모듈 매개 변수에 대해 다중 수준 사전을 사용합니다(예: 고정 및 적응형 QoS 정책 그룹).

이러한 사전이 둘 이상 있고 서로 배타적인 경우에는 을 `default(omit)` 단독으로 사용할 수 없습니다.

다중 수준 사전을 모듈 매개 변수로 사용해야 하는 경우 각 사전이 관련 사전에 대해 최소 1개의 2단계 사전 값을 제공하도록 여러 마이크로서비스(역할)로 기능을 분할해야 합니다.

다음 예에서는 고정 및 적응형 QoS 정책 그룹이 두 마이크로서비스로 분할되는 것을 보여 줍니다.

첫 번째 마이크로서비스에는 고정된 QoS 정책 그룹 값이 포함되어 있습니다.

```
fixed_qos_options:
  capacity_shared:           "{{
loop_arg['fixed_qos_options']['capacity_shared']         | default(omit)
}}"
  max_throughput_iops:       "{{
loop_arg['fixed_qos_options']['max_throughput_iops']     | default(omit)
}}"
  min_throughput_iops:       "{{
loop_arg['fixed_qos_options']['min_throughput_iops']     | default(omit)
}}"
  max_throughput_mbps:       "{{
loop_arg['fixed_qos_options']['max_throughput_mbps']     | default(omit)
}}"
  min_throughput_mbps:       "{{
loop_arg['fixed_qos_options']['min_throughput_mbps']     | default(omit)
}}"
```

두 번째 마이크로서비스에는 적응형 QoS 정책 그룹 값이 포함됩니다.

```
adaptive_qos_options:
  absolute_min_iops:         "{{
loop_arg['adaptive_qos_options']['absolute_min_iops'] | default(omit) }}"
  expected_iops:             "{{
loop_arg['adaptive_qos_options']['expected_iops']     | default(omit) }}"
  peak_iops:                 "{{
loop_arg['adaptive_qos_options']['peak_iops']         | default(omit) }}"
```