NetApp Console自動化ハブ

NetApp Automation

NetApp
November 18, 2025

目次

NetApp Console自動化ハブ

NetApp Console自動化ハブの概要

NetApp Console自動化ハブは、 NetApp の顧客、パートナー、従業員が利用できる自動化ソリューションのコレクションです。自動化ハブにはいくつかの機能と利点があります。

単一の場所で自動化のニーズに対応

アクセスできます "NetApp Console自動化ハブ" コンソールの Web ユーザー インターフェイスを介して。これにより、 NetApp製品とサービスの自動化と運用を強化するために必要なスクリプト、プレイブック、モジュールが 1 か所に集まります。

ソリューションは**NetApp**によって作成、テストされています。

すべての自動化ソリューションとスクリプトは、NetAppによって作成、テストされています。各ソリューションは、特定のお客様のユースケースまたは要求を対象としています。最も重点が置かれているのは、NetAppのファイルサービスやデータサービスとの統合です。

ドキュメント

各自動化ソリューションには、開始するのに役立つ関連ドキュメントが含まれています。ソリューションにはコンソールの Web インターフェイスからアクセスしますが、すべてのドキュメントはこのサイトから入手できます。ドキュメントは、 NetApp製品とクラウド サービスに基づいて構成されています。

将来を見据えた強固な基盤

NetApp は、お客様のデータセンターとクラウド環境の自動化の改善と合理化を支援することに尽力しています。当社は、顧客の要件、テクノロジーの変化、継続的な製品統合に対応するために、コンソール自動化ハブを継続的に強化していく予定です。

皆様の貴重なご意見をぜひお聞かせください

NetAppカスタマーエクスペリエンスオフィス（CXO）自動化チームが、皆様からのご意見をお待ちしています。フィードバック、問題、機能のリクエストについては、mailto：ng-cxo-automation-admins@cxo automation-admins@cxo NetApp .com [CXO automation team]までEメールでお問い合わせください。

Amazon FSx for NetApp ONTAP管理

Amazon FSx for NetApp ONTAP管理 - クラウドへのバースト

この自動化ソリューションを使用すると、ボリュームと関連するFlexCacheを使用してAmazon FSx for NetApp ONTAP管理をプロビジョニングできます。

Amazon FSx for NetApp ONTAP管理は、*FSx for ONTAP*とも呼ばれます。

このソリューションについて

このソリューションで提供される自動化コードでは、大まかに次の処理が実行されます。

- デスティネーションのFSx for ONTAPファイルシステムのプロビジョニング

- ファイルシステム用のStorage Virtual Machine（SVM）のプロビジョニング
- ソースシステムとデスティネーションシステム間にクラスタピア関係を作成する
- FlexCacheのソースシステムとデスティネーションシステム間にSVMピア関係を作成する
- 必要に応じて、FSx for ONTAPを使用してFlexVolボリュームを作成
- FSx for ONTAPでFlexCacheボリュームを作成し、ソースをオンプレミスのストレージに設定

この自動化はDockerとDocker Composeに基づいており、以下で説明するようにLinux仮想マシンにインストールする必要があります。

開始する前に

プロビジョニングと設定を完了するには、次の情報が必要です。

- ダウンロードする必要があります "Amazon FSx for NetApp ONTAP管理 - クラウドへのバースト" NetApp ConsoleWeb UI を介した自動化ソリューション。ソリューションはファイルとしてパッケージ化されています `AWS_FSxN_BTC.zip`。
- ソースシステムとデスティネーションシステム間のネットワーク接続。
- 次の特性を持つLinux VM：
  - DebianベースのLinuxディストリビューション
  - FSx for ONTAPのプロビジョニングと同じVPCサブセットに導入
- AWSアカウント。

手順**1**：**Docker**をインストールして設定する

DebianベースのLinux仮想マシンにDockerをインストールして設定します。

手順

1. 環境を準備

```
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. Dockerをインストールし、インストールを確認します。

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

3. 必要なLinuxグループをユーザと関連付けて追加します。

   最初に、グループ* Docker *がLinuxシステムに存在するかどうかを確認します。表示されない場合は、グループを作成してユーザを追加します。デフォルトでは、現在のシェルユーザがグループに追加されます。

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

4. 新しいグループとユーザ定義をアクティブ化する

   ユーザを使用して新しいグループを作成した場合は、定義をアクティブ化する必要があります。これを行うには、Linuxからログアウトしてから再度ログインします。または、次のコマンドを実行します。

   ```
   newgrp docker
   ```

手順**2**：**Docker Compose**をインストールする

DebianベースのLinux仮想マシンにDocker Composeをインストールします。

手順

1. Docker Composeをインストールします。

   ```
   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. インストールが正常に完了したことを確認します。

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

ステップ**3**：**Docker**イメージを準備する

自動化ソリューションに付属のDockerイメージを抽出してロードする必要があります。

手順

1. ソリューションファイルを、自動化コードを実行する仮想マシンにコピーし `AWS_FSxN_BTC.zip`ます。

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

   入力パラメータ `private-key.pem`は、AWS仮想マシン認証（EC2インスタンス）に使用する秘密鍵ファイ

ルです。

2. ソリューションファイルを含む適切なフォルダに移動し、ファイルを解凍します。

```
unzip AWS_FSxN_BTC.zip
```

3. 解凍操作で作成された新しいフォルダに移動し `AWS_FSxN_BTC`、ファイルを一覧表示します。ファイルが表示されます `aws_fsxn_flexcache_image_latest.tar.gz`。

```
ls -la
```

4. Dockerイメージファイルをロードします。ロード操作は通常数秒で完了します。

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

5. Dockerイメージがロードされたことを確認します。

```
docker images
```

タグが付いた `latest`Dockerイメージが表示されます `aws_fsxn_flexcache_image`。

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

手順**4**：**AWS**クレデンシャル用の環境ファイルを作成する

アクセスキーとシークレットキーを使用して認証用のローカル変数ファイルを作成する必要があります。次に、ファイルをファイルに追加し `.env`ます。

手順

1. 次の場所にファイルを作成し `awsauth.env`ます。

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

2. ファイルに次の内容を追加します。

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

形式*は、上記のとの `value`間にスペースを入れずに正確に指定する必要があります `key`。

3. 変数を使用して、ファイル `AWS_CREDS`への絶対ファイルパスを追加し `.env`ます。例：

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

手順**5**：外部ボリュームを作成する

Terraform状態ファイルやその他の重要なファイルが永続的であることを確認するには、外部ボリュームが必要です。ワークフローとデプロイメントを実行するには、Terraformでこれらのファイルが使用可能である必要があります。

手順

1. Docker Composeの外部に外部ボリュームを作成します。

   コマンドを実行する前に、ボリューム名（最後のパラメータ）を適切な値に更新してください。

   ```
   docker volume create aws_fsxn_volume
   ```

2. コマンドを使用して、外部ボリュームへのパスを環境ファイルに追加し `.env`ます。

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

   既存のファイルの内容とコロンの書式を維持することを忘れないでください。例：

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

   NFS共有を外部ボリュームとして追加するには、次のようなコマンドを使用します。

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

3. Terraform変数を更新します。
   a. フォルダに移動し `aws_fsxn_variables`ます。
   b. との `variables.tf`2つのファイルが存在することを確認します `terraform.tfvars`。
   c. 環境に応じて、の値を更新します `terraform.tfvars`。

      詳細については、を参照してください "Terraformリソース：AWS_FSX_APS_FILE_SYSTEM ONTAP"。

ステップ **6: Amazon FSx for NetApp ONTAP**管理と**FlexCache**用に **Amazon FSx** をプロビジョニングする

Amazon FSx for NetApp ONTAP管理およびFlexCache用にプロビジョニングできます。

手順

1. rootフォルダ（AWS_FSXN_BTC）に移動し、provisioningコマンドを実行します。

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

このコマンドは、2つのコンテナを作成します。1つ目のコンテナでFSx for ONTAPが導入され、2つ目のコンテナでクラスタピアリング、SVMピアリング、デスティネーションボリューム、FlexCacheが作成されます。

2. プロビジョニングプロセスを監視します。

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

このコマンドは出力をリアルタイムで表示しますが、ファイルを介してログをキャプチャするように設定されて `deployment.log`います。これらのログファイルの名前を変更するには、ファイルを編集し `.env`て変数を更新し `DEPLOYMENT_LOGS`ます。

ステップ **7: Amazon FSx for NetApp ONTAP**管理と**FlexCache** を破棄する

オプションで、 Amazon FSx for NetApp ONTAP管理とFlexCacheを削除および除去できます。

1. ファイル内の `terraform.tfvars`変数を「destroy」に設定し `flexcache_operation`ます。
2. rootフォルダ（AWS_FSXN_BTC）に移動し、次のコマンドを実行します。

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

このコマンドは、2つのコンテナを作成します。最初のコンテナはFlexCacheを削除し、2番目のコンテナはFSx for ONTAPを削除します。

3. プロビジョニングプロセスを監視します。

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

Amazon FSx for NetApp ONTAP管理 - 災害復旧

この自動化ソリューションを使用すると、 Amazon FSx for NetApp ONTAP管理を使用してソースシステムの災害復旧バックアップを取得できます。

Amazon FSx for NetApp ONTAP管理は、*FSx for ONTAP*とも呼ばれます。

このソリューションについて

このソリューションで提供される自動化コードでは、大まかに次の処理が実行されます。

- デスティネーションのFSx for ONTAPファイルシステムのプロビジョニング
- ファイルシステム用のStorage Virtual Machine（SVM）のプロビジョニング

- ソースシステムとデスティネーションシステム間にクラスタピア関係を作成する
- SnapMirrorのソースシステムとデスティネーションシステム間にSVMピア関係を作成する
- デスティネーションボリュームを作成
- ソースボリュームとデスティネーションボリューム間にSnapMirror関係を作成する
- ソースボリュームとデスティネーションボリューム間のSnapMirror転送を開始する

この自動化はDockerとDocker Composeに基づいており、以下で説明するようにLinux仮想マシンにインストールする必要があります。

開始する前に

プロビジョニングと設定を完了するには、次の情報が必要です。

- ダウンロードする必要があります "Amazon FSx for NetApp ONTAP管理 - 災害復旧" NetApp ConsoleWeb UI を介した自動化ソリューション。このソリューションは次のようにパッケージ化されています。 `FSxN_DR.zip`。このzipには、 `AWS_FSxN_Bck_Prov.zip` このドキュメントで説明されているソリューションを展開するために使用するファイルです。
- ソースシステムとデスティネーションシステム間のネットワーク接続。
- 次の特性を持つLinux VM：
  - DebianベースのLinuxディストリビューション
  - FSx for ONTAPのプロビジョニングと同じVPCサブセットに導入
- AWSアカウント。

手順**1**：**Docker**をインストールして設定する

DebianベースのLinux仮想マシンにDockerをインストールして設定します。

手順

1. 環境を準備

```
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. Dockerをインストールし、インストールを確認します。

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

3. 必要なLinuxグループをユーザと関連付けて追加します。

   最初に、グループ* Docker *がLinuxシステムに存在するかどうかを確認します。存在しない場合は、グループを作成してユーザを追加します。デフォルトでは、現在のシェルユーザがグループに追加されます。

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

4. 新しいグループとユーザ定義をアクティブ化する

   ユーザを使用して新しいグループを作成した場合は、定義をアクティブ化する必要があります。これを行うには、Linuxからログアウトしてから再度ログインします。または、次のコマンドを実行します。

   ```
   newgrp docker
   ```

手順**2**：**Docker Compose**をインストールする

DebianベースのLinux仮想マシンにDocker Composeをインストールします。

手順

1. Docker Composeをインストールします。

   ```
   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. インストールが正常に完了したことを確認します。

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

ステップ**3**：**Docker**イメージを準備する

自動化ソリューションに付属のDockerイメージを抽出してロードする必要があります。

手順

1. ソリューションファイルを、自動化コードを実行する仮想マシンにコピーし `AWS_FSxN_Bck_Prov.zip` ます。

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

入力パラメータ `private-key.pem`は、AWS仮想マシン認証（EC2インスタンス）に使用する秘密鍵ファイルです。

2. ソリューションファイルを含む適切なフォルダに移動し、ファイルを解凍します。

```
unzip AWS_FSxN_Bck_Prov.zip
```

3. 解凍操作で作成された新しいフォルダに移動し `AWS_FSxN_Bck_Prov`、ファイルを一覧表示します。ファイルが表示されます `aws_fsxn_bck_image_latest.tar.gz`。

```
ls -la
```

4. Dockerイメージファイルをロードします。ロード操作は通常数秒で完了します。

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

5. Dockerイメージがロードされたことを確認します。

```
docker images
```

タグが付いた `latest`Dockerイメージが表示されます `aws_fsxn_bck_image`。

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

手順**4**：**AWS**クレデンシャル用の環境ファイルを作成する

アクセスキーとシークレットキーを使用して認証用のローカル変数ファイルを作成する必要があります。次に、ファイルをファイルに追加し `.env`ます。

手順

1. 次の場所にファイルを作成し `awsauth.env`ます。

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

2. ファイルに次の内容を追加します。

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

形式*は、上記のとの `value`間にスペースを入れずに正確に指定する必要があります `key`。

3. 変数を使用して、ファイル `AWS_CREDS`への絶対ファイルパスを追加し `.env`ます。例：

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

手順**5**：外部ボリュームを作成する

Terraform状態ファイルやその他の重要なファイルが永続的であることを確認するには、外部ボリュームが必要です。ワークフローとデプロイメントを実行するには、Terraformでこれらのファイルが使用可能である必要があります。

手順

1. Docker Composeの外部に外部ボリュームを作成します。

   コマンドを実行する前に、ボリューム名（最後のパラメータ）を適切な値に更新してください。

   ```
   docker volume create aws_fsxn_volume
   ```

2. コマンドを使用して、外部ボリュームへのパスを環境ファイルに追加し `.env`ます。

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

   既存のファイルの内容とコロンの書式を維持することを忘れないでください。例：

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

   NFS共有を外部ボリュームとして追加するには、次のようなコマンドを使用します。

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

3. Terraform変数を更新します。
   a. フォルダに移動し `aws_fsxn_variables`ます。
   b. との `variables.tf`2つのファイルが存在することを確認します `terraform.tfvars`。
   c. 環境に応じて、の値を更新します `terraform.tfvars`。

      詳細については、を参照してください "Terraformリソース：AWS_FSX_APS_FILE_SYSTEM ONTAP"。

手順**6**：バックアップソリューションを導入する

ディザスタリカバリバックアップソリューションを導入してプロビジョニングできます。

手順

1. rootフォルダ（AWS_FSxN_BCK_Provisioning）に移動し、provisioningコマンドを実行します。

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

このコマンドは、3つのコンテナを作成します。1つ目のコンテナはFSx for ONTAPを導入します。2つ目のコンテナでは、クラスタピアリング、SVMピアリング、およびデスティネーションボリュームが作成されます。3番目のコンテナでSnapMirror関係が作成され、SnapMirror転送が開始されます。

2. プロビジョニングプロセスを監視します。

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

このコマンドは出力をリアルタイムで表示しますが、ファイルを介してログをキャプチャするように設定されて `deployment.log`います。これらのログファイルの名前を変更するには、ファイルを編集し `.env`て変数を更新し `DEPLOYMENT_LOGS`ます。

Azure NetApp Files

Azure NetApp Filesを使用したOracleのインストール

この自動化ソリューションを使用すると、Azure NetApp Filesボリュームをプロビジョニングし、使用可能な仮想マシンにOracleをインストールできます。その後、Oracleはこのボリュームをデータストレージに使用します。

このソリューションについて

このソリューションで提供される自動化コードでは、大まかに次の処理が実行されます。

- AzureでのNetAppアカウントのセットアップ
- Azureでストレージ容量プールをセットアップする
- 定義に基づいてAzure NetApp Filesをプロビジョニング
- マウントポイントの作成
- マウントポイントへのAzure NetApp Filesボリュームのマウント
- LinuxサーバへのOracleのインストール
- リスナーとデータベースの作成
- Pluggable Database（PDB）の作成
- リスナーとOracleインスタンスの起動
- ユーティリティをインストールし、スナップショットを作成するように設定します。 `azacsnap`

開始する前に

インストールを完了するには、次のものが必要です。

- ダウンロードする必要があります "Azure NetApp Filesを使用したOracle" NetApp ConsoleWeb UI を介した自動化ソリューション。ソリューションはファイルとしてパッケージ化されています `na_oracle19c_deploy-master.zip`。

- 次の特性を持つLinux VM：
  - RHEL 8（Standard_D8s_v3 - RHEL-8）
  - Azure NetApp Filesのプロビジョニングと同じAzure仮想ネットワークに導入
- Azureアカウント

自動化ソリューションはイメージとして提供され、DockerとDocker Composeを使用して実行されます。以下で説明するように、これらの両方をLinux仮想マシンにインストールする必要があります。

また、コマンドを使用してVMをRedHatに登録する必要があります `sudo subscription-manager register`。アカウントのクレデンシャルの入力を求めるプロンプトが表示されます。必要に応じて、\https://developers.redhat.com/.でアカウントを作成できます。

手順**1**：**Docker**をインストールして設定する

RHEL 8 Linux仮想マシンにDockerをインストールして設定します。

手順

1. 次のコマンドを使用してDockerソフトウェアをインストールします。

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

2. Dockerを起動してバージョンを表示し、インストールが正常に完了したことを確認します。

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

3. 必要なLinuxグループをユーザと関連付けて追加します。

   最初に、グループ* Docker *がLinuxシステムに存在するかどうかを確認します。表示されない場合は、グループを作成してユーザを追加します。デフォルトでは、現在のシェルユーザがグループに追加されます。

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

4. 新しいグループとユーザ定義をアクティブ化する

   ユーザを使用して新しいグループを作成した場合は、定義をアクティブ化する必要があります。これを行うには、Linuxからログアウトしてから再度ログインします。または、次のコマンドを実行します。

```
newgrp docker
```

手順**2**：**Docker Compose**と**NFS**ユーティリティをインストールする

NFSユーティリティパッケージと一緒にDocker Composeをインストールして設定します。

手順

1. Docker Composeをインストールしてバージョンを表示し、インストールが正常に完了したことを確認します。

```
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ユーティリティパッケージをインストールします。

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

手順**3**：**Oracle**インストールファイルをダウンロードする

必要なOracleインストールファイルとパッチファイル、およびユーティリティをダウンロードし `azacsnap` ます。

手順

1. 必要に応じてOracleアカウントにサインインします。
2. 次のファイルをダウンロードします。

ファイル	製品説明
LINUX.X64_193000_db_home.zip	19.3ベースインストーラ
p31281355_190000_Linux-x86-64.zip	19.8 RUパッチ
p6880880_190000_Linux-x86-64.zip	Opatchバージョン12.2.0.1.23
azacsnap_installer_v5.0.ru	azacsnapインストーラ

3. すべてのインストールファイルをフォルダに配置し `/tmp/archive` ます。
4. データベース・サーバ上のすべてのユーザが'フォルダに対するフル・アクセス（読み取り'書き込み'実行）を持っていることを確認し `/tmp/archive` ます

ステップ**4**：**Docker**イメージを準備する

自動化ソリューションに付属のDockerイメージを抽出してロードする必要があります。

手順

1. ソリューションファイルを、自動化コードを実行する仮想マシンにコピーし `na_oracle19c_deploy-master.zip`ます。

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

入力パラメータ `private-key.pem`は、Azure仮想マシンの認証に使用する秘密鍵ファイルです。

2. ソリューションファイルを含む適切なフォルダに移動し、ファイルを解凍します。

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

3. 解凍操作で作成された新しいフォルダに移動し `na_oracle19c_deploy-master`、ファイルを一覧表示します。ファイルが表示されます `ora_anf_bck_image.tar`。

```
ls -lt
```

4. Dockerイメージファイルをロードします。ロード操作は通常数秒で完了します。

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

5. Dockerイメージがロードされたことを確認します。

```
docker images
```

タグが付いた `latest`Dockerイメージが表示されます `ora_anf_bck_image`。

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

手順**5**：外部ボリュームを作成する

Terraform状態ファイルやその他の重要なファイルが永続的であることを確認するには、外部ボリュームが必要です。ワークフローとデプロイメントを実行するには、Terraformでこれらのファイルが使用可能である必要があります。

手順

1. Docker Composeの外部に外部ボリュームを作成します。

   コマンドを実行する前に、必ずボリューム名を更新してください。

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

2. コマンドを使用して、外部ボリュームへのパスを環境ファイルに追加し `.env`ます。

   `PERSISTENT_VOL=path/to/external/volume:/ora_anf_prov`です。

   既存のファイルの内容とコロンの書式を維持することを忘れないでください。例：

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

3. Terraform変数を更新します。
   a. フォルダに移動し `ora_anf_variables`ます。
   b. との `variables.tf`2つのファイルが存在することを確認します `terraform.tfvars`。
   c. 環境に応じて、の値を更新します `terraform.tfvars`。

手順**6**：**Oracle**をインストールする

これで、Oracleのプロビジョニングとインストールが可能になりました。

手順

1. 次の一連のコマンドを使用してOracleをインストールします。

   ```
   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変数をリロードし、の値を表示して確認します `ORACLE_HOME`。
   a. `cd /home/oracle`
   b. `source .bash_profile`
   c. `echo $ORACLE_HOME`
3. Oracleにログインできる必要があります。

   ```
   sudo su oracle
   ```

手順7：Oracleのインストールを検証する

Oracleのインストールが正常に完了したことを確認する必要があります。

手順

1. Linux Oracleサーバにログインし、Oracleプロセスのリストを表示します。これにより、インストールが想定どおりに完了し、Oracleデータベースが実行されていることが確認されます。

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

2. データベースにログインしてデータベース設定を調べ、PDBが正しく作成されたことを確認します。

```
sqlplus / as sysdba
```

次のような出力が表示されます。

```
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. いくつかの簡単なSQLコマンドを実行して、データベースが使用可能であることを確認します。

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

手順8：azacsnapユーティリティをインストールしてスナップショット・バックアップを実行する

スナップショットバックアップを実行するには、ユーティリティをインストールして実行する必要があり`azacsnap`ます。

手順

1. コンテナを取り付けます。

```
docker-compose up azacsnap_install
```

2. スナップショットユーザーアカウントに切り替えます。

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

3. ストレージバックアップの詳細ファイルを設定これにより、構成ファイルが作成され `azacsnap.json`ます。

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

4. スナップショットバックアップを実行します。

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

ステップ**9**：必要に応じてオンプレミスの**PDB**をクラウドに移行

必要に応じて、オンプレミスのPDBをクラウドに移行できます。

手順

1. 環境に応じて、ファイルに変数を設定し `tfvars`ます。
2. PDBを移行します。

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

Cloud Volumes ONTAP for AWS

Cloud Volumes ONTAP for AWS -クラウドへのバースト

この記事は、 NetApp Console自動化ハブからNetApp のお客様が利用できるNetApp Cloud Volumes ONTAP for AWS 自動化ソリューションをサポートしています。

Cloud Volumes ONTAP for AWS自動化ソリューションは、Terraformを使用してCloud Volumes ONTAP for AWSのコンテナ化された導入を自動化するため、手動操作なしでCloud Volumes ONTAP for AWSを迅速に導入できます。

開始する前に

- ダウンロードする必要があります "Cloud Volumes ONTAP AWS -クラウドへのバースト" コンソール Web UI を介した自動化ソリューション。このソリューションは次のようにパッケージ化されています。 `cvo_aws_flexcache.zip`。
- Cloud Volumes ONTAPと同じネットワークにLinux VMをインストールする必要があります。
- Linux VMをインストールしたら、このソリューションの手順に従って必要な依存関係をインストールする必要があります。

手順1：DockerとDocker Composeをインストールする

Docker をインストールする

次の手順では、例としてUbuntu 20.04 Debian Linuxディストリビューションソフトウェアを使用します。実行するコマンドは、使用しているLinuxディストリビューションソフトウェアによって異なります。使用している構成に対応するLinuxディストリビューションソフトウェアのマニュアルを参照してください。

手順

1. 次のコマンドを実行してDockerをインストールし `sudo`ます。

```
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. インストールを確認します。

```
docker –version
```

3. Linuxシステムに「docker」という名前のグループが作成されていることを確認します。必要に応じて、グループを作成します。

```
sudo groupadd docker
```

4. Dockerにアクセスする必要があるユーザをグループに追加します。

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

5. 変更内容は、ログアウトして端末に再度ログインした後に適用されます。または、変更をすぐに適用することもできます。

```
newgrp docker
```

Docker Composeのインストール

手順

1. 次のコマンドを実行して、Docker Composeをインストールし `sudo`ます。

```
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. インストールを確認します。

```
docker-compose –version
```

ステップ**2**：**Docker**イメージを準備する

手順

1. Cloud Volumes ONTAPの導入に使用するLinux VMにフォルダをコピーし `cvo_aws_flexcache.zip`ます。

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

   - `private-key.pem`は、パスワードなしでログインするための秘密鍵ファイルです。
   - `awsuser`はVMのユーザ名です。
   - `IP_ADDRESS_OF_VM`はVMのIPアドレスです。
   - `LOCATION_TO_BE_COPIED`は、フォルダがコピーされる場所です。

2. フォルダを展開し `cvo_aws_flexcache.zip`ます。フォルダは、カレントディレクトリまたはカスタムの場所に展開できます。

   現在のディレクトリにフォルダを展開するには、次のコマンドを実行します。

```
unzip cvo_aws_flexcache.zip
```

   カスタムの場所にフォルダを抽出するには、次のコマンドを実行します。

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

3. コンテンツを展開したら、フォルダに移動し `CVO_Aws_Deployment`、次のコマンドを実行してファイルを表示します。

```
 ls -la
```

次の例のようなファイルの一覧が表示されます。

```
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. ファイルを探します `cvo_aws_flexcache_ubuntu_image.tar`。これには、Cloud Volumes ONTAP for AWSの導入に必要なDockerイメージが含まれています。
5. ファイルを解凍します。

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

6. Dockerイメージがロードされるまで数分待ってから、Dockerイメージが正常にロードされたことを確認します。

```
docker images
```

次の例に示すように、タグで `latest`という名前のDockerイメージが表示され `cvo_aws_flexcache_ubuntu_image`ます。

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

必要に応じて、Dockerイメージの名前を変更できます。Dockerイメージ名を変更した場合は、ファイルと `docker-compose-destroy`ファイルでDockerイメージ名を更新して `docker-compose-deploy`ください。

手順**3:**環境変数ファイルを作成する

この段階では、2 つの環境変数ファイルを作成する必要があります。 1 つのファイルは、AWS アクセスキーと秘密キーを使用して AWS Resource Manager API を認証するためのものです。 2 番目のファイルは、コンソール Terraform モジュールが AWS API を見つけて認証できるように環境変数を設定するためのものです。

手順

1. 次の場所にファイルを作成し `awsauth.env`ます。

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

   a. ファイルに次の内容を追加し `awsauth.env`ます。

      access_key=<> secret_key=<>

   形式*は上記のとおりである必要があります。

2. ファイルに絶対ファイルパスを追加します `.env`。

   環境変数に対応する環境ファイル `AWS_CREDS`の絶対パスを入力し `awsauth.env`ます。

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

3. フォルダに移動し `cvo_aws_variable`、credentialsファイルのアクセスキーとシークレットキーを更新します。

   ファイルに次の内容を追加します。

   aws_access_key_id=<>aws_secret_access_key=<>

   形式*は上記のとおりである必要があります。

ステップ**4**： **NetApp**インテリジェントサービスにサインアップする

クラウド プロバイダーを通じてNetApp Intelligent Services にサインアップし、時間単位 (PAYGO) または年間契約で支払います。NetAppインテリジェント サービスには、 NetAppバックアップおよびリカバリ、 Cloud Volumes ONTAP、 NetAppクラウド階層化、 NetAppランサムウェア回復力、 NetAppディザスタ リカバリが含まれます。NetApp Data Classification は追加料金なしでサブスクリプションに含まれています。

手順

1. Amazon Web Services (AWS) ポータルから、**SaaS** に移動し、* NetApp Intelligent Services にサブスクライブ* を選択します。

   Cloud Volumes ONTAPと同じリソースグループを使用することも別のリソースグループを使用することもできます。

2. NetAppコンソール ポータルを構成して、SaaS サブスクリプションをコンソールにインポートします。

   これはAWSポータルから直接設定できます。

   構成を確認するためにコンソール ポータルにリダイレクトされます。

3. *保存*を選択して、コンソール ポータルで構成を確認します。

手順**5**：外部ボリュームを作成する

Terraform状態ファイルとその他の重要なファイルを永続的に保持するには、外部ボリュームを作成する必要があります。ワークフローと導入環境を実行するには、Terraformでファイルを使用できることを確認する必

要があります。

手順

1. Docker Composeの外部に外部ボリュームを作成します。

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

例：

```
docker volume create cvo_aws_volume_dst
```

2. 次のいずれかのオプションを使用します。

   a. 環境ファイルに外部ボリュームパスを追加します `.env`。

      以下に示す正確な形式に従う必要があります。

      形式：

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

      例：
      `PERSISTENT_VOL=cvo_aws_volume_dst:/cvo_aws`

   b. NFS共有を外部ボリュームとして追加

      DockerコンテナがNFS共有と通信できること、および読み取り/書き込みなどの適切な権限が設定されていることを確認します。

      i. 次のように、Docker Composeファイルで、外部ボリュームへのパスとしてNFS共有パスを追加します。Format：

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

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

3. フォルダに移動し `cvo_aws_variables`ます。

   フォルダに次の変数ファイルが表示されます。

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

4. 必要に応じて、ファイル内の値を変更し `terraform.tfvars`ます。

   ファイル内の変数値を変更する場合は、特定のサポートドキュメントを参照する必要があります `terraform.tfvars`。値は、リージョン、アベイラビリティゾーン、およびCloud Volumes ONTAP for AWSでサポートされるその他の要因によって異なります。これには、シングルノードおよびハイアベイラ

ビリティ（HA）ペアのライセンス、ディスクサイズ、VMサイズが含まれます。

コンソールエージェントとCloud Volumes ONTAP Terraformモジュールのすべてのサポート変数は、`variables.tf`ファイル。変数名を参照する必要があります `variables.tf`ファイルに追加する前に`terraform.tfvars`ファイル。

5. 要件に応じて、次のオプションをまたは `false`に設定することで、`FlexCache`および`FlexClone`を有効または無効にできます `true`。

   次に、FlexCacheとFlexCloneを有効にする例を示します。

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

ステップ**6**：**Cloud Volumes ONTAP for AWS**を導入する

Cloud Volumes ONTAP for AWSを導入するには、次の手順を実行します。

手順

1. ルートフォルダから次のコマンドを実行して導入を開始します。

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

   2つのコンテナがトリガーされます。1つ目のコンテナはCloud Volumes ONTAPを導入し、2つ目のコンテナはAutoSupportに計測データを送信します。

   2番目のコンテナは、最初のコンテナがすべてのステップを正常に完了するまで待機します。

2. ログファイルを使用して導入プロセスの進行状況を監視します。

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

   このコマンドは、出力をリアルタイムで提供し、次のログファイルのデータをキャプチャします。`deployment.log`

   `telemetry_asup.log`

   これらのログファイルの名前を変更するには、次の環境変数を使用してファイルを編集し`.env`ます。

   `DEPLOYMENT_LOGS`

   `TELEMETRY_ASUP_LOGS`

   次の例は、ログファイル名を変更する方法を示しています。

   `DEPLOYMENT_LOGS=<your_deployment_log_filename>.log`

   `TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log`

終了後

次の手順を使用して、一時的な環境を削除し、導入プロセス中に作成された項目をクリーンアップできます。

手順

1. FlexCacheを導入した場合は、変数ファイルで次のオプションを設定する `terraform.tfvars`と、FlexCacheボリュームがクリーンアップされ、前の手順で作成した一時環境が削除されます。

   ```
   flexcache_operation = "destroy"
   ```

   指定可能なオプションは `deploy`、および `destroy`です。

2. FlexCloneを導入した場合は、変数ファイルで次のオプションを設定する `terraform.tfvars`と、FlexCloneボリュームがクリーンアップされ、前の手順で作成した一時環境が削除されます。

   ```
   flexclone_operation = "destroy"
   ```

   指定可能なオプションは `deploy`、および `destroy`です。

Cloud Volumes ONTAP for Azure

Cloud Volumes ONTAP for Azure -クラウドへのバースト対応

この記事は、NetApp Console自動化ハブからNetApp のお客様が利用できるNetApp Cloud Volumes ONTAP for Azure Automation ソリューションをサポートしています。

Cloud Volumes ONTAP for Azure自動化ソリューションは、Terraformを使用してCloud Volumes ONTAP for Azureのコンテナ化された導入を自動化するため、手動操作なしでCloud Volumes ONTAP for Azureを迅速に導入できます。

開始する前に

- ダウンロードする必要があります "Cloud Volumes ONTAP Azure -クラウドへのバースト" コンソールWeb UI を介した自動化ソリューション。このソリューションは次のようにパッケージ化されています。`CVO-Azure-Burst-To-Cloud.zip`。
- Cloud Volumes ONTAPと同じネットワークにLinux VMをインストールする必要があります。
- Linux VMをインストールしたら、このソリューションの手順に従って必要な依存関係をインストールする必要があります。

手順1：DockerとDocker Composeをインストールする

Docker をインストールする

次の手順では、例としてUbuntu 20.04 Debian Linuxディストリビューションソフトウェアを使用します。実行するコマンドは、使用しているLinuxディストリビューションソフトウェアによって異なります。使用している構成に対応するLinuxディストリビューションソフトウェアのマニュアルを参照してください。

手順

1. 次のコマンドを実行してDockerをインストールし `sudo`ます。

```
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. インストールを確認します。

```
docker –version
```

3. Linuxシステムに「docker」という名前のグループが作成されていることを確認します。必要に応じて、グループを作成します。

```
sudo groupadd docker
```

4. Dockerにアクセスする必要があるユーザをグループに追加します。

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

5. 変更内容は、ログアウトして端末に再度ログインした後に適用されます。または、変更をすぐに適用することもできます。

```
newgrp docker
```

Docker Composeのインストール

手順

1. 次のコマンドを実行して、Docker Composeをインストールし `sudo`ます。

```
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. インストールを確認します。

```
docker-compose -version
```

ステップ**2**：**Docker**イメージを準備する

手順

1. Cloud Volumes ONTAPの導入に使用するLinux VMにフォルダをコピーし `CVO-Azure-Burst-To-Cloud.zip` ます。

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

   - `private-key.pem`は、パスワードなしでログインするための秘密鍵ファイルです。
   - `azureuser`はVMのユーザ名です。
   - `IP_ADDRESS_OF_VM`はVMのIPアドレスです。
   - `LOCATION_TO_BE_COPIED`は、フォルダがコピーされる場所です。

2. フォルダを展開し `CVO-Azure-Burst-To-Cloud.zip`ます。フォルダは、カレントディレクトリまたはカスタムの場所に展開できます。

   現在のディレクトリにフォルダを展開するには、次のコマンドを実行します。

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

   カスタムの場所にフォルダを抽出するには、次のコマンドを実行します。

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

3. コンテンツを展開したら、フォルダに移動し `CVO_Azure_Deployment`、次のコマンドを実行してファイルを表示します。

```
 ls -la
```

   次の例のようなファイルの一覧が表示されます。

```
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. ファイルを探します `cvo_azure_flexcache_ubuntu_image_latest.tar.gz`。これには、Cloud Volumes ONTAP for Azureの導入に必要なDockerイメージが含まれています。
5. ファイルを解凍します。

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

6. Dockerイメージがロードされるまで数分待ってから、Dockerイメージが正常にロードされたことを確認します。

```
docker images
```

次の例に示すように、タグで `latest`という名前のDockerイメージが表示され `cvo_azure_flexcache_ubuntu_image_latest`ます。

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

手順**3:**環境変数ファイルを作成する

この段階では、2 つの環境変数ファイルを作成する必要があります。 1 つのファイルは、サービス プリンシパル資格情報を使用して Azure Resource Manager API を認証するためのものです。 2 番目のファイルは、コンソール Terraform モジュールが Azure API を見つけて認証できるように環境変数を設定するためのものです。

手順

1. サービスプリンシパルを作成します。

環境変数ファイルを作成する前に、の手順に従ってサービスプリンシパルを作成する必要があります。"リソースにアクセスできるAzure Active Directoryアプリケーションとサービスプリンシパルを作成する"

2. 新しく作成したサービスプリンシパルに* Contributor *ロールを割り当てます。
3. カスタムロールを作成します。
   a. ファイルを探し `sp_role.json`、表示された操作で必要な権限を確認します。
   b. これらの権限を挿入し、新しく作成したサービスプリンシパルにカスタムロールを関連付けます。
4. に移動し、[新しいクライアントシークレット]*を選択してクライアントシークレットを作成します。

   クライアントシークレットを作成するときは、この値を再度表示できないため、* value *列の詳細を記録する必要があります。また、次の情報も記録する必要があります。

   - クライアントID
   - サブスクリプションID
   - テナントID

     この情報は、環境変数を作成する際に必要になります。クライアントIDとテナントIDの情報は、サービスプリンシパルUIの*[Overview]*セクションで確認できます。

5. 環境ファイルを作成します。
   a. 次の場所にファイルを作成し `azureauth.env`ます。

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

      i. ファイルに次の内容を追加します。

         ClientID=<>clientSecret=<>サブスクリプションID=<> tenantId=<>

      形式*は、キーと値の間にスペースを入れずに、上記のとおりにする必要があります。

   b. 次の場所にファイルを作成し `credentials.env`ます。

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

      i. ファイルに次の内容を追加します。

         azure_tenant_ID=<> azure_client_secret=<> azure_client_ID=<> azure_subscription_ID=<>

   形式*は、キーと値の間にスペースを入れずに、上記のとおりにする必要があります。

6. ファイルに絶対ファイルパスを追加します `.env`。

   環境変数に対応するファイル `AZURE_RM_CREDS`に、環境ファイル `.env`の絶対パスを入力し `azureauth.env`ます。

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

   環境変数に対応するファイル `BLUEXP_TF_AZURE_CREDS`に、環境ファイル `.env`の絶対パスを入力し `credentials.env`ます。

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

ステップ**4**： **NetApp**インテリジェントサービスにサインアップする

クラウド プロバイダーを通じてNetApp Intelligent Services にサインアップし、時間単位 (PAYGO) または年間契約で支払います。NetAppインテリジェント サービスには、 NetAppバックアップおよびリカバリ、 Cloud Volumes ONTAP、 NetAppクラウド階層化、 NetAppランサムウェア回復力、 NetAppディザスタ リカバリが含まれます。NetApp Data Classificationは追加料金なしでサブスクリプションに含まれています

手順

1. Azure ポータルから **SaaS** に移動し、* NetApp Intelligent Services をサブスクライブ* を選択します。
2. Cloud Manager（Cap PYGO by Hour、WORM and data services）*プランを選択します。

   Cloud Volumes ONTAPと同じリソースグループを使用することも別のリソースグループを使用することもできます。

3. コンソール ポータルを構成して、SaaS サブスクリプションをコンソールにインポートします。

   Azureポータルから直接構成するには、**[**製品とプランの詳細**]***に移動し、[今すぐアカウントを構成]*オプションを選択します。

   その後、構成を確認するためにコンソール ポータルにリダイレクトされます。

4. *保存*を選択して、コンソール ポータルで構成を確認します。

手順**5**：外部ボリュームを作成する

Terraform状態ファイルとその他の重要なファイルを永続的に保持するには、外部ボリュームを作成する必要があります。ワークフローと導入環境を実行するには、Terraformでファイルを使用できることを確認する必要があります。

手順

1. Docker Composeの外部に外部ボリュームを作成します。

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

   例：

   ```
   docker volume create cvo_azure_volume_dst
   ```

2. 次のいずれかのオプションを使用します。

   a. 環境ファイルに外部ボリュームパスを追加します `.env`。

      以下に示す正確な形式に従う必要があります。

      形式：

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

例：
`PERSISTENT_VOL=cvo_azure_volume_dst:/cvo_azure`

b. NFS共有を外部ボリュームとして追加

DockerコンテナがNFS共有と通信できること、および読み取り/書き込みなどの適切な権限が設定されていることを確認します。

i. 次のように、Docker Composeファイルで、外部ボリュームへのパスとしてNFS共有パスを追加します。Format：

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

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

3. フォルダに移動し `cvo_azure_variables`ます。

フォルダに次の変数ファイルが表示されます。

`terraform.tfvars`

`variables.tf`

4. 必要に応じて、ファイル内の値を変更し `terraform.tfvars`ます。

ファイル内の変数値を変更する場合は、特定のサポートドキュメントを参照する必要があります `terraform.tfvars`。値は、リージョン、アベイラビリティゾーン、およびCloud Volumes ONTAP for Azureでサポートされるその他の要因によって異なります。これには、シングルノードおよびハイアベイラビリティ（HA）ペアのライセンス、ディスクサイズ、VMサイズが含まれます。

コンソールエージェントとCloud Volumes ONTAP Terraformモジュールのすべてのサポート変数は、`variables.tf`ファイル。変数名を参照する必要があります `variables.tf`ファイルに追加する前に `terraform.tfvars`ファイル。

5. 要件に応じて、次のオプションをまたは `false`に設定することで、`FlexCache`および`FlexClone`を有効または無効にできます `true`。

次に、FlexCacheとFlexCloneを有効にする例を示します。

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

6. 必要に応じて、Azure Active Directory ServiceからTerraform変数の値を取得できます `az_service_principal_object_id`。

a. [エンタープライズアプリケーション]–>[すべてのアプリケーション]*に移動し、前の手順で作成したサービスプリンシパルの名前を選択します。

b. オブジェクトIDをコピーし、Terraform変数の値を挿入します。

`az_service_principal_object_id`

ステップ6：Cloud Volumes ONTAP for Azureを導入する

Cloud Volumes ONTAP for Azureを導入するには、次の手順を実行します。

手順

1. ルートフォルダから次のコマンドを実行して導入を開始します。

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

2つのコンテナがトリガーされます。1つ目のコンテナはCloud Volumes ONTAPを導入し、2つ目のコンテナはAutoSupportに計測データを送信します。

2番目のコンテナは、最初のコンテナがすべてのステップを正常に完了するまで待機します。

2. ログファイルを使用して導入プロセスの進行状況を監視します。

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

このコマンドは、出力をリアルタイムで提供し、次のログファイルのデータをキャプチャします。

`deployment.log`

`telemetry_asup.log`

これらのログファイルの名前を変更するには、次の環境変数を使用してファイルを編集し `.env`ます。

`DEPLOYMENT_LOGS`

`TELEMETRY_ASUP_LOGS`

次の例は、ログファイル名を変更する方法を示しています。

`DEPLOYMENT_LOGS=<your_deployment_log_filename>.log`

`TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log`

終了後

次の手順を使用して、一時的な環境を削除し、導入プロセス中に作成された項目をクリーンアップできます。

手順

1. FlexCacheを導入した場合は、ファイルで次のオプションを設定する `terraform.tfvars`と、FlexCacheボリュームがクリーンアップされ、前の手順で作成した一時環境が削除されます。

`flexcache_operation = "destroy"`

指定可能なオプションは `deploy`、および `destroy`です。

2. FlexCloneを導入した場合は、ファイルで次のオプションを設定する `terraform.tfvars`と、FlexCloneボリュームがクリーンアップされ、前の手順で作成した一時環境が削除されます。

```
flexclone_operation = "destroy"
```

指定可能なオプションは `deploy`、および `destroy`です。

Cloud Volumes ONTAP for Google Cloud

Cloud Volumes ONTAP for Google Cloud -クラウドへのバースト

この記事は、NetApp Console自動化ハブからNetApp のお客様が利用できる、Google Cloud Automation ソリューション向けのNetApp Cloud Volumes ONTAP をサポートしています。

Cloud Volumes ONTAP for Google Cloud自動化ソリューションは、Cloud Volumes ONTAP for Google Cloud のコンテナ化された導入を自動化し、手動操作なしでCloud Volumes ONTAP for Google Cloudを迅速に導入できるようにします。

開始する前に

- ダウンロードする必要があります "Cloud Volumes ONTAP for Google Cloud -クラウドへのバースト" コンソール Web UI を介した自動化ソリューション。このソリューションは次のようにパッケージ化されています。 `cvo_gcp_flexcache.zip`。
- Cloud Volumes ONTAPと同じネットワークにLinux VMをインストールする必要があります。
- Linux VMをインストールしたら、このソリューションの手順に従って必要な依存関係をインストールする必要があります。

手順**1**：**Docker**と**Docker Compose**をインストールする

Docker をインストールする

次の手順では、例としてUbuntu 20.04 Debian Linuxディストリビューションソフトウェアを使用します。実行するコマンドは、使用しているLinuxディストリビューションソフトウェアによって異なります。使用している構成に対応するLinuxディストリビューションソフトウェアのマニュアルを参照してください。

手順

1. 次のコマンドを実行してDockerをインストールします。

```
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. インストールを確認します。

```
docker –version
```

3. Linuxシステムに「docker」という名前のグループが作成されていることを確認します。必要に応じて、グループを作成します。

```
sudo groupadd docker
```

4. Dockerにアクセスする必要があるユーザをグループに追加します。

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

5. 変更内容は、ログアウトして端末に再度ログインした後に適用されます。または、変更をすぐに適用することもできます。

```
newgrp docker
```

Docker Composeのインストール

手順

1. 次のコマンドを実行して、Docker Composeをインストールし `sudo`ます。

```
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. インストールを確認します。

```
docker-compose -version
```

ステップ**2**：**Docker**イメージを準備する

手順

1. Cloud Volumes ONTAPの導入に使用するLinux VMにフォルダをコピーし `cvo_gcp_flexcache.zip`ます。

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

   - `private-key.pem`は、パスワードなしでログインするための秘密鍵ファイルです。
   - `gcpuser`はVMのユーザ名です。
   - `IP_ADDRESS_OF_VM`はVMのIPアドレスです。
   - `LOCATION_TO_BE_COPIED`は、フォルダがコピーされる場所です。

2. フォルダを展開し `cvo_gcp_flexcache.zip`ます。フォルダは、カレントディレクトリまたはカスタムの場所に展開できます。

   現在のディレクトリにフォルダを展開するには、次のコマンドを実行します。

```
unzip cvo_gcp_flexcache.zip
```

   カスタムの場所にフォルダを抽出するには、次のコマンドを実行します。

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

3. コンテンツを展開したら、次のコマンドを実行してファイルを表示します。

```
 ls -la
```

   次の例のようなファイルの一覧が表示されます。

```
    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. ファイルを探します `cvo_gcp_flexcache_ubuntu_image.tar`。これには、Cloud Volumes ONTAP for Google Cloudの導入に必要なDockerイメージが含まれています。
5. ファイルを解凍します。

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

6. Dockerイメージがロードされるまで数分待ってから、Dockerイメージが正常にロードされたことを確認します。

```
docker images
```

次の例に示すように、タグで `latest`という名前のDockerイメージが表示され `cvo_gcp_flexcache_ubuntu_image`ます。

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

必要に応じて、Dockerイメージの名前を変更できます。Dockerイメージ名を変更した場合は、ファイルと `docker-compose-destroy`ファイルでDockerイメージ名を更新して `docker-compose-deploy`ください。

手順**3**：**JSON**ファイルを更新する

この段階で、Google Cloudプロバイダを認証するためにサービスアカウントキーを使用してファイルを更新する必要があります `cxo-automation-gcp.json`。

1. Cloud Volumes ONTAPとコンソールエージェントをデプロイする権限を持つサービスアカウントを作成します。"サービスアカウントの作成について詳しくは、こちらをご覧ください。"

2. アカウントのキーファイルをダウンロードし、キーファイル情報でファイルを更新します `cxo-automation-gcp.json`。 `cxo-automation-gcp.json`ファイルはフォルダにあり `cvo_gcp_variables`ます。

   例

```
{
  "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"
}
```

   ファイル形式は上記のとおりである必要があります。

ステップ4： NetAppインテリジェントサービスにサインアップする

クラウド プロバイダーを通じてNetApp Intelligent Services にサインアップし、時間単位 (PAYGO) または年間契約で支払います。NetAppインテリジェント サービスには、 NetAppバックアップおよびリカバリ、 Cloud Volumes ONTAP、 NetAppクラウド階層化、 NetAppランサムウェア回復力、 NetAppディザスタ リカバリが含まれます。NetApp Data Classification は追加料金なしでサブスクリプションに含まれています。

手順

1. に移動"Google Cloudコンソール"* NetApp Intelligent Services にサブスクライブ*を選択します。
2. NetAppコンソール ポータルを構成して、SaaS サブスクリプションをコンソールにインポートします。

   これは Google Cloud Platform から直接設定できます。構成を確認するためにコンソール ポータルにリダイレクトされます。

3. *保存*を選択して、コンソール ポータルで構成を確認します。

詳細については、以下を参照してください。 "NetAppコンソールの Google Cloud 認証情報とサブスクリプションを管理する" 。

ステップ5：必要なGoogle Cloud APIを有効にする

Cloud Volumes ONTAPとコンソール エージェントをデプロイするには、プロジェクトで次の Google Cloud API を有効にする必要があります。

- Cloud Deployment Manager V2 API

- クラウドロギング API
- Cloud Resource Manager API の略
- Compute Engine API
- ID およびアクセス管理（ IAM ） API

"API の有効化の詳細をご覧ください"

手順**6**：外部ボリュームを作成する

Terraform状態ファイルやその他の重要なファイルを永続的に保持するには、外部ボリュームを作成する必要があります。ワークフローと導入環境を実行するには、Terraformでファイルを使用できることを確認する必要があります。

手順

1. Docker Composeの外部に外部ボリュームを作成します。

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

例：

```
docker volume create cvo_gcp_volume_dst
```

2. 次のいずれかのオプションを使用します。

a. 環境ファイルに外部ボリュームパスを追加します `.env`。

以下に示す正確な形式に従う必要があります。

形式：

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

例：
`PERSISTENT_VOL=cvo_gcp_volume_dst:/cvo_gcp`

b. NFS共有を外部ボリュームとして追加

Dockerコンテナが NFS共有と通信できること、および読み取り/書き込みなどの適切な権限が設定されていることを確認します。

i. 次のように、Docker Composeファイルで、外部ボリュームへのパスとしてNFS共有パスを追加します。Format：

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

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

3. フォルダに移動し `cvo_gcp_variables`ます。

   フォルダに次のファイルが表示されます。

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

4. 必要に応じて、ファイル内の値を変更し `terraform.tfvars`ます。

   ファイル内の変数値を変更する場合は、特定のサポートドキュメントを参照する必要があります `terraform.tfvars`。値は、リージョン、アベイラビリティゾーン、およびCloud Volumes ONTAP for Google Cloudでサポートされているその他の要因によって異なります。これには、シングルノードおよびハイアベイラビリティ（HA）ペアのライセンス、ディスクサイズ、VMサイズが含まれます。

   コンソールエージェントとCloud Volumes ONTAP Terraformモジュールのすべてのサポート変数は、`variables.tf`ファイル。変数名を参照する必要があります `variables.tf`ファイルに追加する前に `terraform.tfvars`ファイル。

5. 要件に応じて、次のオプションをまたは `false`に設定することで、`FlexCache`および`FlexClone`を有効または無効にできます `true`。

   次に、FlexCacheとFlexCloneを有効にする例を示します。

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

ステップ**7**：**Cloud Volumes ONTAP for Google Cloud**を導入する

Cloud Volumes ONTAP for Google Cloudを導入するには、次の手順を実行します。

手順

1. ルートフォルダから次のコマンドを実行して導入を開始します。

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

   2つのコンテナがトリガーされます。1つ目のコンテナはCloud Volumes ONTAPを導入し、2つ目のコンテナはAutoSupportに計測データを送信します。

   2番目のコンテナは、最初のコンテナがすべてのステップを正常に完了するまで待機します。

2. ログファイルを使用して導入プロセスの進行状況を監視します。

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

   このコマンドは、出力をリアルタイムで提供し、次のログファイルのデータをキャプチャします。`deployment.log`

   `telemetry_asup.log`

これらのログファイルの名前を変更するには、次の環境変数を使用してファイルを編集し `.env`ます。

```
DEPLOYMENT_LOGS

TELEMETRY_ASUP_LOGS
```

次の例は、ログファイル名を変更する方法を示しています。

```
DEPLOYMENT_LOGS=<your_deployment_log_filename>.log

TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log
```

終了後

次の手順を使用して、一時的な環境を削除し、導入プロセス中に作成された項目をクリーンアップできます。

手順

1. FlexCacheを導入した場合は、ファイルで次のオプションを設定する `terraform.tfvars`と、FlexCacheボリュームがクリーンアップされ、前の手順で作成した一時環境が削除されます。

   ```
   flexcache_operation = "destroy"
   ```

   指定可能なオプションは `deploy`、および `destroy`です。

2. FlexCloneを導入した場合は、ファイルで次のオプションを設定する `terraform.tfvars`と、FlexCloneボリュームがクリーンアップされ、前の手順で作成した一時環境が削除されます。

   ```
   flexclone_operation = "destroy"
   ```

   指定可能なオプションは `deploy`、および `destroy`です。

ONTAP

1日目

ONTAP Day 0/1ソリューションの概要

ONTAP Day 0/1 自動化ソリューションを使用すると、Ansible を使用してONTAPクラスターを導入および構成できます。このソリューションは、 "NetApp Console自動化ハブ"。

柔軟なONTAP導入オプション

要件に応じて、オンプレミスのハードウェアまたはSimulate ONTAPを使用して、Ansibleを使用してONTAPクラスタを導入および設定できます。

オンプレミスのハードウェア

このソリューションは、FASやAFFシステムなど、ONTAPを実行しているオンプレミスのハードウェアを使用して導入できます。Ansibleを使用してONTAPクラスタを導入および設定するには、Linux VMを使用する必

要があります。

Simulate ONTAP

ONTAPシミュレータを使用してこのソリューションを導入するには、NetAppサポートサイトからSimulate ONTAPの最新バージョンをダウンロードする必要があります。Simulate ONTAPは、ONTAPソフトウェアの仮想シミュレータです。Simulate ONTAPは、Windows、Linux、またはMacシステム上のVMwareハイパーバイザーで実行されます。WindowsホストおよびLinuxホストでは、VMware Workstationハイパーバイザーを使用してこのソリューションを実行する必要があります。Mac OSを使用している場合は、VMware Fusionハイパーバイザーを使用します。

レイヤーデザイン

Ansibleフレームワークは、自動化の実行タスクとロジックタスクの開発と再利用を簡易化します。このフレームワークは、意思決定タスク（ロジックレイヤ）と自動化における実行ステップ（実行レイヤ）を区別します。これらのレイヤの仕組みを理解することで、構成をカスタマイズできます。

Ansibleの「プレイブック」は、一連のタスクを最初から最後まで実行します。 `site.yml`Playbookにはplaybookと `execution.yml`Playbookが含まれてい `logic.yml`ます。

要求が実行されると、 `site.yml`Playbookは最初にPlaybookを呼び出し `logic.yml`、次にPlaybookを呼び出し `execution.yml`てサービス要求を実行します。

フレームワークのロジックレイヤを使用する必要はありません。ロジック層は、実行のためにハードコードされた値を超えてフレームワークの機能を拡張するためのオプションを提供します。これにより、必要に応じてフレームワークの機能をカスタマイズできます。

ロジック層

ロジックレイヤは次の要素で構成されています。

- `logic.yml`プレイブック
- ディレクトリ内のロジックタスクファイル `logic-tasks`

ロジックレイヤは、大幅なカスタム統合（ServiceNowへの接続など）を必要とせずに、複雑な意思決定を行う機能を提供します。ロジック層はコンフィグレーション可能で、マイクロサービスへの入力を提供します。

ロジック層をバイパスする機能も提供されています。ロジックレイヤをバイパスする場合は、変数を定義しない `logic_operation`でください。プレイブックを直接呼び出す `logic.yml`と、実行せずにある程度のレベルのデバッグを実行できます。「`debug`」ステートメントを使用して、の値が正しいことを確認できます `raw_service_request`。

重要な考慮事項：

- `logic.yml`プレイブックによって変数が検索され `logic_operation`ます。リクエストで変数が定義されている場合は、ディレクトリからタスクファイルをロードし `logic-tasks`ます。タスクファイルは.ymlファイルである必要があります。一致するタスクファイルがなく、変数が定義されている場合、 `logic_operation`ロジックレイヤは失敗します。
- 変数のデフォルト値 `logic_operation`はです `no-op`。変数が明示的に定義されていない場合は、デフォルトでが設定され `no-op`、操作は実行されません。
- 変数がすでに定義されている場合、 `raw_service_request`実行は実行レイヤに進みます。変数が定義されていない場合、ロジックレイヤは失敗します。

実行レイヤ

実行レイヤは次の要素で構成されます。

- `execution.yml`プレイブック

実行レイヤは、ONTAPクラスタを設定するためのAPI呼び出しを行います。プレイブックで `execution.yml` は、実行時に変数が定義されている必要があり `raw_service_request`ます。

カスタマイズのサポート

このソリューションは、要件に応じてさまざまな方法でカスタマイズできます。

カスタマイズオプションは次のとおりです。

- Ansibleプレイブックの変更
- ロールを追加する

Ansibleファイルのカスタマイズ

次の表に、このソリューションに含まれるカスタマイズ可能なAnsibleファイルを示します。

場所	製品説明
`playbooks/inventory/hosts`	ホストとグループのリストを含む単一のファイルが格納されます。
`playbooks/group_vars/all/*`	Ansibleを使用すると、変数を一度に複数のホストに適用できます。このフォルダ内のファイル（、、 `clusters.yml`、 `defaults.yml` `services.yml`、、 `standards.yml`など）の一部またはすべてを変更できます `cfg.yml` `vault.yml`。
`playbooks/logic-tasks`	Ansible内の意思決定タスクをサポートし、ロジックと実行の分離を維持します。該当するサービスに対応するファイルをこのフォルダに追加できます。
`playbooks/vars/*`	Ansibleのプレイブックとロールで使用される動的な値により、構成のカスタマイズ、柔軟性、再利用が可能になります。必要に応じて、このフォルダ内のファイルの一部またはすべてを変更できます。

ロールのカスタマイズ

Ansibleのロール（マイクロサービスとも呼ばれます）を追加または変更してソリューションをカスタマイズすることもできます。詳細については、を参照してください"カスタマイズ"。

ONTAP Day 0/1ソリューションの使用準備

自動化ソリューションを導入する前に、ONTAP環境を準備し、Ansibleをインストールして設定する必要があります。

初期計画に関する考慮事項

このソリューションを使用してONTAPクラスタを導入する前に、次の要件と考慮事項を確認しておく必要があります。

基本的な要件

このソリューションを使用するには、次の基本要件を満たす必要があります。

- オンプレミスまたはONTAPシミュレータを介してONTAPソフトウェアにアクセスできる必要があります。
- ONTAPソフトウェアの使用方法を理解しておく必要があります。
- Ansible自動化ソフトウェアツールの使用方法を理解しておく必要があります。

計画に関する考慮事項

この自動化ソリューションを導入する前に、次の事項を決定する必要があります。

- Ansibleコントロールノードを実行する場所。
- ONTAPシステム（オンプレミスのハードウェアまたはONTAPシミュレータ）。
- カスタマイズが必要かどうか。

ONTAPシステムの準備

オンプレミスのONTAPシステムを使用している場合でも、ONTAPをシミュレートしている場合でも、自動化ソリューションを導入する前に環境を準備する必要があります。

必要に応じて、Simulate ONTAPをインストールして設定

ONTAPシミュレータを使用してこのソリューションを導入する場合は、Simulate ONTAPをダウンロードして実行する必要があります。

開始する前に

- Simulate ONTAPの実行に使用するVMwareハイパーバイザーをダウンロードしてインストールする必要があります。
  - WindowsまたはLinux OSを使用している場合は、VMware Workstationを使用します。
  - Mac OSを使用している場合は、VMware Fusionを使用します。

> Mac OSを使用している場合は、Intelプロセッサが必要です。

手順

ローカル環境に2つのONTAPシミュレータをインストールするには、次の手順を実行します。

1. からSimulate ONTAPをダウンロードします"NetApp Support Site"。

> 2つのONTAPシミュレータをインストールしますが、ダウンロードする必要があるのはソフトウェアのコピーを1つだけです。

2. VMwareアプリケーションがまだ実行されていない場合は、起動します。
3. ダウンロードしたシミュレータファイルを見つけ、右クリックしてVMwareアプリケーションで開きます。
4. 最初のONTAPインスタンスの名前を設定します。

5. シミュレータがブートするまで待ち、指示に従ってシングルノードクラスタを作成します。

   2つ目のONTAPインスタンスに対して同じ手順を繰り返します。

6. 必要に応じて、フルディスク補完を追加します。

   各クラスタから、次のコマンドを実行します。

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

ONTAPシステムの状態

ONTAPシステムがオンプレミスで実行されているか、ONTAPシミュレータを介して実行されているかを問わず、システムの初期状態を確認する必要があります。

次のONTAPシステム要件を満たしていることを確認します。

- ONTAPがインストールされ、クラスタが定義されていない状態で実行されている。
- ONTAPがブートし、クラスタにアクセスするためのIPアドレスが表示されます。
- ネットワークに到達できます。
- 管理者クレデンシャルが必要です。
- Message of the Day（MOTD）バナーに管理アドレスが表示されます。

必要な自動化ソフトウェアのインストール

このセクションでは、Ansibleのインストール方法と導入のための自動化ソリューションの準備方法について説明します。

Ansibleをインストール

AnsibleはLinuxシステムまたはWindowsシステムにインストールできます。

AnsibleでONTAPクラスタとの通信に使用されるデフォルトの通信方法はSSHです。

Ansibleのインストールについては、を参照してください"ネットアップと Ansible の使用： Ansible のインストール"。

Ansibleはシステムの制御ノードにインストールする必要があります。

自動化ソリューションをダウンロードして準備する

導入用の自動化ソリューションをダウンロードして準備するには、次の手順を実行します。

1. ダウンロード "ONTAP - 1日目およびヘルスチェック" コンソール Web UI を介した自動化ソリューション。このソリューションは次のようにパッケージ化されています。 `ONTAP_DAY0_DAY1.zip`。
2. zipフォルダを展開し、Ansible環境内の制御ノード上の目的の場所にファイルをコピーします。

Ansibleフレームワークの初期構成

Ansibleフレームワークの初期設定を実行します。

1. に移動します `playbooks/inventory/group_vars/all`。
2. ファイルを復号化 `vault.yml`します。

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

ボルトパスワードの入力を求められたら、次の一時パスワードを入力します。

```
NetApp123!
```

「NetApp123！」は、ファイルとそれに対応するバックアップパスワードを復号化するための一時的なパスワード `vault.yml`です。最初に使用した後は、自分のパスワードを使用してファイルを*暗号化する必要があります。

3. 次のAnsibleファイルを変更します。
   - `clusters.yml`-このファイルの値を環境に合わせて変更します。
   - `vault.yml`-ファイルを復号化したら、ONTAPクラスタ、ユーザ名、およびパスワードの値を環境に合わせて変更します。
   - `cfg.yml`-のファイルパスを設定し `log2file`、で `cfg`をに設定し `show_request`てを `True`表示します `raw_service_request`。

```
`raw_service_request`変数は、ログファイルおよび実行中に表示されます。
```

リストされている各ファイルには、要件に応じて変更する方法に関するコメントが含まれています。

4. ファイルを再暗号化し `vault.yml`ます。

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

暗号化時にボルトの新しいパスワードを選択するように求められます。

5. 有効なPythonインタプリタに移動し `playbooks/inventory/hosts`て設定します。
6. サービスを導入し `framework_test`ます。

次のコマンドは、値を `cluster_identity_info`指定してモジュールを `gather_subset`実行し `na_ontap_info`ます。これにより、基本的な設定が正しいかどうか、およびクラスタと通信できるかどうかが検証されます。

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

クラスタごとにコマンドを実行します。

成功すると、次の例のような出力が表示されます。

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

ソリューションを使用した**ONTAP**クラスタの導入

準備と計画が完了すると、ONTAPの導入初日のソリューションを使用して、Ansibleを使用してONTAPクラスタを迅速に構成できるようになります。

このセクションの手順では、要求を実際に実行するのではなく、いつでもテストすることができます。要求をテストするには、コマンドラインのPlaybookをに `logic.yml`変更し `site.yml`ます。

ロケーションに `docs/tutorial-requests.txt`は、この手順で使用されるすべてのサービスリクエストの最終バージョンが含まれています。サービス要求の実行に問題がある場合は、関連する要求をファイルから `playbooks/inventory/group_vars/all/tutorial-requests.yml`場所にコピーし、必要に応じてハードコードされた値（IPアドレス、集約名など）を変更できます `tutorial-requests.txt`。これで、要求を正常に実行できるようになります。

開始する前に

- Ansibleをインストールしておく必要があります。
- ONTAP Day 0/1ソリューションをダウンロードし、Ansibleコントロールノードの目的の場所にフォルダを展開しておく必要があります。
- ONTAPシステムの状態が要件を満たし、必要なクレデンシャルが必要です。
- に記載されている必要なタスクをすべて完了しておく必要があります"準備"。

このソリューションの例では、2つのクラスタの名前に「Cluster_01」と「Cluster_02」を使用しています。これらの値は、環境内のクラスタの名前に置き換える必要があります。

手順**1**：クラスタの初期設定

この段階で、クラスタの初期設定手順をいくつか実行する必要があります。

手順

1. 場所に移動し `playbooks/inventory/group_vars/all/tutorial-requests.yml`、ファイル内の要求を確認します `cluster_initial`。環境に必要な変更を行います。
2. サービスリクエストのフォルダにファイルを作成し `logic-tasks`ます。たとえば、という名前のファイルを作成し `cluster_initial.yml`ます。

   次の行を新しいファイルにコピーします。

```
- 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. 変数を定義し `raw_service_request`ます。

   次のいずれかのオプションを使用して、フォルダに作成したファイル `logic-tasks`の変数を `cluster_initial.yml`定義でき `raw_service_request`ます。

   - *オプション1 *：変数を手動で定義し `raw_service_request`ます。

     エディタを使用してファイルを開き `tutorial-requests.yml`、11行目から165行目に内容をコピーします。次の例に示すように、新しいファイルの変数の `cluster_initial.yml`下に内容を貼り付け `raw service request`ます。


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 #---
11 service: cluster_initial
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"

例を示します

ファイル例 `cluster_initial.yml`：

```
- 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:
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAA
```

```
        - 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 }}"
```

- *オプション2 *：Jinjaテンプレートを使用してリクエストを定義します。

  次のJinjaテンプレート形式を使用して値を取得することもできます `raw_service_request`。

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

4. 最初のクラスタでクラスタの初期設定を実行します。

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

処理を続行する前に、エラーがないことを確認してください。

5. 2つ目のクラスタに対してコマンドを繰り返します。

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

2つ目のクラスタでエラーが発生していないことを確認します。

Ansibleの出力の先頭までスクロールすると、次の例に示すように、フレームワークに送信された要求が表示されます。

例を示します

```
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",
```

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

6. 各ONTAPインスタンスにログインし、要求が成功したことを確認します。

手順**2**：クラスタ間**LIF**を設定する

LIF定義を要求に追加し、マイクロサービスを定義する `ontap_interface`ことで、クラスタ間LIFを設定できるようになり `cluster_initial`ました。

サービス定義とリクエストが連携してアクションを決定します。

- サービス定義に含まれていないマイクロサービスのサービス要求を指定した場合、要求は実行されません。
- サービス定義で定義されている1つ以上のマイクロサービスをサービス要求に提供したが、要求から除外された場合、要求は実行されません。

Playbookで `execution.yml`は、マイクロサービスのリストが次の順序でスキャンされ、サービス定義が評価されます。

- マイクロサービス定義に含まれるエントリと一致するディクショナリキーを持つエントリが要求にある場合、 `args`要求が実行されます。

- サービス要求に一致するエントリがない場合、その要求はエラーなしでスキップされます。

手順

1. 前に作成したファイルに移動し `cluster_initial.yml`、リクエスト定義に次の行を追加してリクエストを変更します。

```
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. 次のコマンドを実行します。

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

3. 各インスタンスにログインして、LIFがクラスタに追加されているかどうかを確認します。

   例を示します

```
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.
```

   この出力は、LIFが*追加されなかったことを示しています。これは、マイクロサービスをファイルに定義する必要がある `services.yml`ため `ontap_interface`です。

4. LIFが変数に追加されたことを確認します `raw_service_request`。

例を示します

次の例は、LIFが要求に追加されたことを示しています。

```
           "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. ファイルの `services.yml`にマイクロサービス `cluster_initial`を定義し `ontap_interface`ます。

   次の行をファイルにコピーして、マイクロサービスを定義します。

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

6. 要求とファイルにマイクロサービスが定義され `services.yml`たので `ontap_interface`、要求を再度実行します。

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

7. 各ONTAPインスタンスにログインし、LIFが追加されたことを確認します。

手順**3**：必要に応じて複数のクラスタを構成

必要に応じて、同じ要求で複数のクラスタを設定できます。要求を定義するときは、各クラスタの変数名を指定する必要があります。

手順

1. ファイルに2番目のクラスタのエントリを追加し `cluster_initial.yml`て、同じ要求で両方のクラスタを設定します。

   次の例は、2番目のエントリを追加したあとのフィールドを表示し `ontap_aggr`ます。

```
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. の他のすべての項目に変更を適用し `cluster_initial`ます。
3. 次の行をファイルにコピーして、要求にクラスタピアリングを追加します。

```
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要求を実行します。

```
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>
```

手順4：SVMの初期設定

この手順のこのステージでは、クラスタ内のSVMを設定します。

手順

1. SVMとSVMのピア関係を設定するために、ファイル内の要求を `tutorial-requests.yml`更新し `svm_initial`ます。

   次の項目を設定する必要があります。

   - SVM

- SVMピア関係
- 各SVMのSVMインターフェイス

2. リクエスト定義の変数定義を更新し `svm_initial`ます。次の変数定義を変更する必要があります。
   - `cluster_name`
   - `vserver_name`
   - `peer_cluster_name`
   - `peer_vserver`

     定義を更新するには、 `svm_initial`定義の後に`*' {} '*`を削除し `req_details`、正しい定義を追加します。

3. サービスリクエストのフォルダにファイルを作成し `logic-tasks`ます。たとえば、という名前のファイルを作成し `svm_initial.yml`ます。

   次の行をファイルにコピーします。

```
- 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. 変数を定義し `raw_service_request`ます。

   次のいずれかのオプションを使用して、フォルダ内のの `logic-tasks`の変数を `svm_initial`定義でき `raw_service_request`ます。

   - *オプション1 *：変数を手動で定義し `raw_service_request`ます。

     エディタを使用してファイルを開き `tutorial-requests.yml`、179行目から222行目に内容をコピーします。次の例に示すように、新しいファイルの変数の `svm_initial.yml`下に内容を貼り付け `raw

service request`ます。

例を示します

ファイル例 `svm_initial.yml`：

```
- 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
```

- *オプション2 *：Jinjaテンプレートを使用してリクエストを定義します。

  次のJinjaテンプレート形式を使用して値を取得することもできます `raw_service_request`。

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

5. 要求を実行します。

```
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. 各ONTAPインスタンスにログインし、構成を検証します。
7. SVMインターフェイスを追加

   ファイルの `services.yml`でサービスを `svm_initial`定義し `ontap_interface`、要求を再実行します。

```
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. 各ONTAPインスタンスにログインし、SVMインターフェイスが設定されていることを確認します。

ステップ**5**：必要に応じて、サービスリクエストを動的に定義します。

前の手順では、 `raw_service_request`変数はハードコードされています。これは、学習、開発、テストに役立ちます。サービスリクエストを動的に生成することもできます。

次のセクションでは、上位レベルのシステムと統合しない場合に必要なを動的に生成するオプションを提供し `raw_service_request`ます。

- コマンドで変数が定義されていない `logic.yml`場合、 `logic_operation`ファイルはフォルダからファイルをインポートしません `logic-tasks`。つまり、は `raw_service_request`Ansibleの外部で定義し、実行時にフレームワークに提供する必要があります。
- フォルダ内のタスクファイル名は `logic-tasks`、拡張子.ymlを付けない変数の値と一致する必要があります `logic_operation`。
- フォルダ内のタスクファイルは `logic-tasks`、を動的に定義し `raw_service_request`ます。唯一の要件は、有効なを関連ファイルの最後のタスクとして定義することです。 `raw_service_request`

サービスリクエストを動的に定義する方法

ロジックタスクを適用してサービスリクエストを動的に定義する方法は複数あります。これらのオプションの一部を次に示します。

- フォルダのAnsibleタスクファイルの使用 `logic-tasks`
- 変数への変換に適したデータを返すカスタムロールを呼び出し `raw_service_request`ます。
- Ansible環境以外の別のツールを呼び出して、必要なデータを提供します。たとえば、Active IQ Unified ManagerへのREST API呼び出しなどです。

次のコマンド例では、ファイルを使用して、各クラスタのサービス要求を動的に定義し `tutorial-requests.yml`ます。

```
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
```

この段階では、次の作業を完了している必要があります。

- 要件に応じて、のすべてのファイルを確認して変更しまし `playbooks/inventory/group_vars/all`た。各ファイルには、変更に役立つ詳細なコメントが記載されています。
- 必要なタスクファイルをディレクトリに追加しました `logic-tasks`。
- 必要なデータファイルをディレクトリに追加します `playbook/vars`。

次のコマンドを使用して、ONTAP Day 0/1ソリューションを導入し、導入環境の健全性を確認します。

この段階では、ファイルを復号化して変更し、新しいパスワードで暗号化する必要があり `vault.yml`ます。

- ONTAP Day 0サービスを実行します。

```
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サービスを実行します。

```
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>
```

- クラスタ全体の設定を適用します。

```
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>
```

- 健全性チェックを実行します。

```
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ソリューションのカスタマイズ

ONTAPのDay 0/1ソリューションを要件に合わせてカスタマイズするには、Ansibleのロールを追加または変更します。

ロールは、Ansibleフレームワーク内のマイクロサービスを表します。各マイクロサービスが1つの処理を実行します。たとえば、ONTAPの0日目は、複数のマイクロサービスを含むサービスです。

Ansibleロールを追加

Ansibleのロールを追加して、環境に合わせてソリューションをカスタマイズできます。必要なロールは、Ansibleフレームワーク内のサービス定義によって定義されます。

あるロールをマイクロサービスとして使用するには、次の要件を満たす必要があります。

- 変数内の引数のリストを受け入れ `args`ます。
- Ansibleの「block、rescue、always」構造を使用して、各ブロックに特定の要件を設定します。
- 単一のAnsibleモジュールを使用し、ブロック内に1つのタスクを定義します。
- このセクションで説明する要件に従って、使用可能なすべてのモジュールパラメータを実装します。

必要なマイクロサービス構造

各ロールが次の変数をサポートしている必要があります。

- `mode`:モードがロールに設定されている場合は `test`、ロールが実際に実行されずに何を行うかを示すをインポートしようとし `test.yml`ます。

  特定の相互依存性のため、これを実装することは必ずしも可能ではありません。

- `status`:プレイブックの実行の全体的なステータス。値が設定されていない場合、 `success`ロールは実行されません。
- `args`:ロールパラメータ名と一致するキーを持つロール固有のディクショナリのリスト。
- `global_log_messages`：プレイブックの実行中にログメッセージを収集します。ロールが実行されるたびに1つのエントリが生成されます。
- `log_name`:エントリ内のロールを参照するために使用される名前 `global_log_messages`。
- `task_descr`:役割の機能の簡単な説明。
- `service_start_time`:各ロールが実行された時間を追跡するために使用されるタイムスタンプ。
- `playbook_status`：Ansible Playbookのステータス。
- `role_result`:ロール出力を含み、エントリ内の各メッセージに含まれる変数 `global_log_messages`。

ロール構造の例

次の例は、マイクロサービスを実装するロールの基本的な構造を示しています。この例では、構成に応じて変数を変更する必要があります。

例を示します

基本的な役割構造：

```
- 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 ] }}"
```

ロール例で使用されている変数は次のとおりです。

- <NAME>：マイクロサービスごとに指定する必要がある交換可能な値。
- <LOG_NAME>：ロギングに使用するロールの省略形の名前。たとえば、 `ONTAP_VOLUME`です。
- <TASK_DESCRIPTION>：マイクロサービスの機能の簡単な説明。
- <MODULE_NAME>：タスクのAnsibleモジュール名。

> トップレベルの `execute.yml`プレイブックでコレクションを指定します `netapp.ontap`。モジュールがコレクションの一部である場合、 `netapp.ontap`モジュール名を完全に指定する必要はありません。

- <MODULE_SPECIFIC_PARAMETERS>:マイクロサービスの実装に使用するモジュールに固有のAnsibleモジュールパラメータ。次のリストでは、パラメータのタイプとそのグループ化方法について説明します。
  - 必須パラメータ：すべての必須パラメータをデフォルト値なしで指定します。
  - マイクロサービス固有のデフォルト値を持つパラメータ(モジュールのドキュメントで指定されているデフォルト値とは異なる)。
  - 残りのパラメータはすべてデフォルト値として使用され `default(omit)`ます。

マルチレベルディクショナリをモジュールパラメータとして使用する

NetAppが提供する一部のAnsibleモジュールでは、モジュールパラメータ（固定QoSポリシーグループやアダプティブQoSポリシーグループなど）にマルチレベルディクショナリを使用します。

これらのディクショナリが使用されている場合、特に複数のディクショナリがあり、それらが相互に排他的である場合は、単独で使用することは `default(omit)`機能しません。

マルチレベルディクショナリをモジュールパラメータとして使用する必要がある場合は、機能を複数のマイクロサービス(ロール)に分割して、それぞれが関連するディクショナリに少なくとも1つのセカンドレベルディクショナリ値を確実に提供できるようにする必要があります。

次の例は、2つのマイクロサービスに分割された固定QoSポリシーグループとアダプティブQoSポリシーグループを示しています。

1つ目のマイクロサービスには、固定QoSポリシーグループ値が含まれています。

```
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)
}}"
```

2つ目のマイクロサービスには、アダプティブQoSポリシーグループの値が含まれています。

```
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) }}"
```

著作権に関する情報

商標に関する情報