BlueXP自动化目录

NetApp Automation

NetApp
October 23, 2024

目录

BlueXP自动化目录

BlueXP 自动化目录概述

BlueXP 自动化目录是一组可供NetApp客户、合作伙伴和员工使用的自动化解决方案。该目录具有多个功能和优势。

只需一个位置即可满足您的自动化需求

您可以通过BlueXP Web用户界面访问 "BlueXP自动化目录"。这样、您就可以在一个位置找到增强NetApp产品和服务的自动化和操作所需的脚本、操作手册和模块。

解决方案由**NetApp**创建和测试

所有自动化解决方案和脚本均由NetApp创建并进行了测试。每个解决方案都针对特定的客户用例或请求。大多数人都关注与NetApp文件和数据服务的集成。

文档

每个自动化解决方案都包含相关文档、可帮助您快速入门。虽然可以通过BlueXP Web界面访问这些解决方案、但所有文档均可从此站点获得。本文档根据NetApp产品和云服务进行组织。

为未来奠定坚实的基础

NetApp致力于帮助客户改进和简化数据中心和云环境的自动化。我们希望继续改进BlueXP 自动化目录、以满足客户需求、技术变更和持续的产品集成。

我们希望听到您的心声

NetApp客户体验办公室(CXO)自动化团队希望听到您的心声。如果您有任何反馈、问题或功能请求、请发送电子邮件至mailto：ng) CxO-automance-admins@NetApp．com[CXO自动化团队]。

适用于 NetApp ONTAP 的 Amazon FSX

Amazon FSx for NetApp ONTAP—突发到云

您可以使用此自动化解决方案为包含卷和关联FlexCache的NetApp ONTAP配置Amazon FSx。

Amazon FSx for NetApp ONTAP也称为*FSx for ONTAP。

关于该解决方案

概括地说、此解决方案提供的自动化代码将执行以下操作：

- 为ONTAP文件系统配置目标FSx
- 为文件系统配置Storage Virtual Machine (SVM)
- 在源系统和目标系统之间创建集群对等关系
- 在FlexCache的源系统和目标系统之间创建SVM对等关系
- (可选)使用FSx for ONTAP创建FlexVol卷

- 在FSx for ONTAP中创建一个FlexCache卷、其中源卷指向内置存储

此自动化操作基于Docker和Docker编制、必须按如下所述将其安装在Linux虚拟机上。

开始之前

要完成配置和配置、您必须满足以下条件：

- 您需要通过BlueXP Web UI下载 "Amazon FSx for NetApp ONTAP—突发到云"自动化解决方案。该解决方案打包为file `AWS_FSxN_BTC.zip`。
- 源系统和目标系统之间的网络连接。
- 具有以下特征的Linux VM：
  - 基于Debian的Linux分发版
  - 部署在用于FSx for ONTAP配置的同一VPC子集上
- AWS帐户。

第1步：安装和配置Docker

在基于Debian的Linux虚拟机中安装和配置Docker。

步骤

1. 准备环境。

```
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. 安装Docker并验证安装。

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

3. 添加具有关联用户的所需Linux组。

   首先检查Linux系统中是否存在组*Docker *。如果没有、请创建组并添加用户。默认情况下、当前shell用户将添加到组中。

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

4. 激活新的组和用户定义

   如果您使用用户创建了新组、则需要激活这些定义。要执行此操作、您可以注销Linux、然后重新进入。或者、您可以运行以下命令。

```
newgrp docker
```

第2步：安装Docker配置

在基于Debian的Linux虚拟机中安装Docker编制。

步骤

1. 安装Docker配置。

```
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. 验证安装是否成功。

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

第3步：准备Docker映像

您需要提取并加载随自动化解决方案提供的Docker映像。

步骤

1. 将解决方案文件复制 `AWS_FSxN_BTC.zip`到要运行自动化代码的虚拟机。

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

   输入参数 `private-key.pem`是用于AWS虚拟机身份验证(EC2实例)的专用密钥文件。

2. 导航到包含解决方案文件的正确文件夹、然后解压缩该文件。

```
unzip AWS_FSxN_BTC.zip
```

3. 导航到通过解压缩操作创建的新文件夹 `AWS_FSxN_BTC`、并列出文件。您应看到文件 `aws_fsxn_flexcache_image_latest.tar.gz`。

```
ls -la
```

4. 加载Docker映像文件。加载操作通常应在几秒钟内完成。

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

5. 确认Docker映像已加载。

```
docker images
```

您应看到标记为的 `latest`Docker映像 `aws_fsxn_flexcache_image。

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

第4步：为AWS凭据创建环境文件

您必须使用访问和机密密钥创建一个用于身份验证的本地变量文件。然后将该文件添加到该文件中 `.env`。

步骤

1. 在以下位置创建 `awsauth.env`文件：

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

2. 将以下内容添加到文件中：

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

格式“必须”与上面所示完全相同，并且和 `value`之间没有任何空格 `key。

3. 使用变量将绝对文件路径添加到此文件 `AWS_CREDS`中 `.env。例如：

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

第5步：创建外部卷

您需要一个外部卷来确保Terraform状态文件和其他重要文件是永久性的。必须为Terraform提供这些文件、才能运行工作流和部署。

步骤

1. 在Docker撰写之外创建外部卷。

请确保在运行命令之前将卷名称(Last参数)更新为适当的值。

```
docker volume create aws_fsxn_volume
```

2. 使用命令将外部卷的路径添加到环境文件中 `.env`：

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

   请务必保留现有文件内容和冒号格式。例如：

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

   而是可以使用以下命令将NFS共享添加为外部卷：

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

3. 更新Terraform变量。
   a. 导航到文件夹 `aws_fsxn_variables`。
   b. 确认存在以下两个文件： `terraform.tfvars`和 `variables.tf`。
   c. 根据环境需要更新中的值 `terraform.tfvars`。

      有关详细信息、请参见 "Terraform资源：aws_FSX_raf_File_system ONTAP" 。

第6步：为NetApp ONTAP和FlexCache配置Amazon FSx

您可以为NetApp ONTAP和FlexCache配置Amazon FSx。

步骤

1. 导航到文件夹根目录(aws_fs_bTC)、然后发出配置命令。

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

   此命令将创建两个容器。第一个容器部署FSx for ONTAP、第二个容器创建集群对等、SVM对等、目标卷和FlexCache。

2. 监控配置过程。

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

   此命令可实时提供输出，但已配置为通过文件捕获日志 `deployment.log`。您可以通过编辑这些日志文件并更新变量来更改这些文件的 `DEPLOYMENT_LOGS`名称 `.env`。

第7步：销毁**Amazon FSx for NetApp ONTAP**和**FlexCache**

您可以选择删除和删除Amazon FSx for NetApp ONTAP和FlexCache。

1. 将文件中的 `terraform.tfvars`变量设置 `flexcache_operation`为"Destroy"。
2. 导航到文件夹根目录(aws_fs_bTC)、然后发出以下命令。

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

此命令将创建两个容器。第一个容器删除FlexCache、第二个容器删除FSx for ONTAP。

3. 监控配置过程。

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

Amazon FSx for NetApp ONTAP—灾难恢复

您可以使用此自动化解决方案通过Amazon FSx for NetApp ONTAP为源系统创建灾难恢复备份。

Amazon FSx for NetApp ONTAP也称为*FSx for ONTAP。

关于该解决方案

概括地说、此解决方案提供的自动化代码将执行以下操作：

- 为ONTAP文件系统配置目标FSx
- 为文件系统配置Storage Virtual Machine (SVM)
- 在源系统和目标系统之间创建集群对等关系
- 在SnapMirror的源系统和目标系统之间创建SVM对等关系
- 创建目标卷
- 在源卷和目标卷之间创建SnapMirror关系
- 在源卷和目标卷之间启动SnapMirror传输

此自动化操作基于Docker和Docker编制、必须按如下所述将其安装在Linux虚拟机上。

开始之前

要完成配置和配置、您必须满足以下条件：

- 您需要通过BlueXP Web UI下载 "Amazon FSx for NetApp ONTAP—灾难恢复"自动化解决方案。该解决方案打包为 `FSxN_DR.zip`。此压缩文件包含 `AWS_FSxN_Bck_Prov.zip`用于部署本文档所述解决方案的文件。
- 源系统和目标系统之间的网络连接。

- 具有以下特征的Linux VM：
  - 基于Debian的Linux分发版
  - 部署在用于FSx for ONTAP配置的同一VPC子集上
- AWS帐户。

第1步：安装和配置Docker

在基于Debian的Linux虚拟机中安装和配置Docker。

步骤

1. 准备环境。

```
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. 安装Docker并验证安装。

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

3. 添加具有关联用户的所需Linux组。

   首先检查Linux系统中是否存在组*Docker *。如果不存在、请创建组并添加用户。默认情况下、当前shell用户将添加到组中。

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

4. 激活新的组和用户定义

   如果您使用用户创建了新组、则需要激活这些定义。要执行此操作、您可以注销Linux、然后重新进入。或者、您可以运行以下命令。

```
newgrp docker
```

第2步：安装Docker配置

在基于Debian的Linux虚拟机中安装Docker编制。

步骤

1. 安装Docker配置。

```
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. 验证安装是否成功。

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

第3步：准备Docker映像

您需要提取并加载随自动化解决方案提供的Docker映像。

步骤

1. 将解决方案文件复制 `AWS_FSxN_Bck_Prov.zip`到要运行自动化代码的虚拟机。

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

   输入参数 `private-key.pem`是用于AWS虚拟机身份验证(EC2实例)的专用密钥文件。

2. 导航到包含解决方案文件的正确文件夹、然后解压缩该文件。

```
unzip AWS_FSxN_Bck_Prov.zip
```

3. 导航到通过解压缩操作创建的新文件夹 `AWS_FSxN_Bck_Prov`、并列出文件。您应看到文件 `aws_fsxn_bck_image_latest.tar.gz`。

```
ls -la
```

4. 加载Docker映像文件。加载操作通常应在几秒钟内完成。

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

5. 确认Docker映像已加载。

```
docker images
```

您应看到标记为的 `latest`Docker映像 `aws_fsxn_bck_image。

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

第4步：为AWS凭据创建环境文件

您必须使用访问和机密密钥创建一个用于身份验证的本地变量文件。然后将该文件添加到该文件中 `.env`。

步骤

1. 在以下位置创建 `awsauth.env`文件：

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

2. 将以下内容添加到文件中：

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

   格式"必须"与上面所示完全相同，并且和 `value`之间没有任何空格 `key。

3. 使用变量将绝对文件路径添加到此文件 `AWS_CREDS`中 `.env。例如：

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

第5步：创建外部卷

您需要一个外部卷来确保Terraform状态文件和其他重要文件是永久性的。必须为Terraform提供这些文件、才能运行工作流和部署。

步骤

1. 在Docker撰写之外创建外部卷。

   请确保在运行命令之前将卷名称(Last参数)更新为适当的值。

   ```
   docker volume create aws_fsxn_volume
   ```

2. 使用命令将外部卷的路径添加到环境文件中 `.env`：

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

请务必保留现有文件内容和冒号格式。例如：

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

而是可以使用以下命令将NFS共享添加为外部卷：

PERSISTENT_VOL=nfs/mnt/document:/aws_fsx_bck

3. 更新Terraform变量。
   a. 导航到文件夹 aws_fsxn_variables。
   b. 确认存在以下两个文件： terraform.tfvars`和 `variables.tf。
   c. 根据环境需要更新中的值 terraform.tfvars。

      有关详细信息、请参见 "Terraform资源：aws_FSX_raf_File_system ONTAP" 。

第6步：部署备份解决方案

您可以部署和配置灾难恢复备份解决方案。

步骤

1. 导航到文件夹根(aws_fsxN_Bck_Prov)、然后发出配置命令。

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

   此命令可创建三个容器。第一个容器部署FSx for ONTAP。第二个容器将创建集群对等、SVM对等和目标卷。第三个容器将创建SnapMirror关系并启动SnapMirror传输。

2. 监控配置过程。

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

   此命令可实时提供输出，但已配置为通过文件捕获日志 deployment.log。您可以通过编辑这些日志文件并更新变量来更改这些文件的 DEPLOYMENT_LOGS`名称 `.env。

Azure NetApp Files

使用Azure NetApp Files安装Oracle

您可以使用此自动化解决方案配置Azure NetApp Files卷并在可用虚拟机上安装Oracle。然后、Oracle会使用这些卷进行数据存储。

关于该解决方案

概括地说、此解决方案提供的自动化代码将执行以下操作：

- 在Azure上设置NetApp帐户
- 在Azure上设置存储容量池
- 根据定义配置Azure NetApp Files卷
- 创建挂载点
- 将Azure NetApp Files卷挂载到挂载点
- 在Linux服务器上安装Oracle
- 创建侦听器和数据库
- 创建可插拔数据库(PDB)
- 启动侦听器和Oracle实例
- 安装并配置 `azacsnap`实用程序以创建快照

开始之前

要完成安装、您必须满足以下条件：

- 您需要通过BlueXP Web UI下载 "使用Azure NetApp Files的Oracle"自动化解决方案。该解决方案打包为file `na_oracle19c_deploy-master.zip`。
- 具有以下特征的Linux VM：
  - RHEL 8 (Standard"(标准) D8s_v3-rRHEL 8)
  - 部署在用于Azure NetApp Files配置的同一个Azure虚拟网络上
- Azure帐户

该自动化解决方案以映像形式提供、并使用Docker和Docker构成运行。您需要按照如下所述在Linux虚拟机上安装这两个组件。

您还应使用命令向RedHat注册此虚拟机 `sudo subscription-manager register`。命令将提示您输入帐户凭据。如果需要、您可以在\https://developers.redhat.com/.上创建帐户

第**1**步：安装和配置**Docker**

在RHEL 8 Linux虚拟机中安装和配置Docker。

步骤

1. 使用以下命令安装Docker软件。

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

2. 启动Docker并显示版本以确认安装成功。

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

3. 添加具有关联用户的所需Linux组。

   首先检查Linux系统中是否存在组*Docker *。如果没有、请创建组并添加用户。默认情况下、当前shell用户将添加到组中。

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

4. 激活新的组和用户定义

   如果您使用用户创建了新组、则需要激活这些定义。要执行此操作、您可以注销Linux、然后重新进入。或者、您可以运行以下命令。

```
newgrp docker
```

第2步：安装Docker配置和NFS实用程序

安装和配置Docker配置以及NFS实用程序软件包。

步骤

1. 安装Docker配置并显示版本以确认安装成功。

```
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. 安装NFS实用程序软件包。

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

第3步：下载Oracle安装文件

下载所需的Oracle安装和修补程序文件以及 `azacsnap`实用程序。

步骤

1. 根据需要登录到Oracle帐户。
2. 下载以下文件。

文件	说明
LINUX.X64_193000_db_home.zip	基础安装程序
p31281355_190000_Linux-x86-64.zip	安装了一个插片
p6880880_190000_Linux-x86-64.zip	请选择12.2.0.1.23版
azacsnap_installer_v5.0.run	azacsnap"安装程序

3. 将所有安装文件放在文件夹中 `/tmp/archive`。
4. 确保数据库服务器上的所有用户都对文件夹具有完全访问权限(读取、写入、执行) `/tmp/archive`。

第**4**步：准备**Docker**映像

您需要提取并加载随自动化解决方案提供的Docker映像。

步骤

1. 将解决方案文件复制 `na_oracle19c_deploy-master.zip`到要运行自动化代码的虚拟机。

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

输入参数 `private-key.pem`是用于Azure虚拟机身份验证的私钥文件。

2. 导航到包含解决方案文件的正确文件夹、然后解压缩该文件。

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

3. 导航到通过解压缩操作创建的新文件夹 `na_oracle19c_deploy-master`、并列出文件。您应看到文件 `ora_anf_bck_image.tar`。

```
ls -lt
```

4. 加载Docker映像文件。加载操作通常应在几秒钟内完成。

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

5. 确认Docker映像已加载。

```
docker images
```

您应看到标记为的 `latest`Docker映像 `ora_anf_bck_image。

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

第5步：创建外部卷

您需要一个外部卷来确保Terraform状态文件和其他重要文件是永久性的。必须为Terraform提供这些文件、才能运行工作流和部署。

步骤

1. 在Docker撰写之外创建外部卷。

   请确保先更新卷名称、然后再运行命令。

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

2. 使用命令将外部卷的路径添加到环境文件中 `.env`：

   `PERSISTENT_VOL=path/to/external/volume:/ora_anf_prov`(英文)

   请务必保留现有文件内容和冒号格式。例如：

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

3. 更新Terraform变量。
   a. 导航到文件夹 `ora_anf_variables`。
   b. 确认存在以下两个文件： `terraform.tfvars`和 `variables.tf`。
   c. 根据环境需要更新中的值 `terraform.tfvars`。

第6步：安装Oracle

现在、您可以配置和安装Oracle。

步骤

1. 使用以下命令序列安装Oracle。

   ```
   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. 重新加载您的Bash变量，并通过显示的值进行确认 `ORACLE_HOME`。
   a. `cd /home/oracle`
   b. `source .bash_profile`
   c. `echo $ORACLE_HOME`
3. 您应该能够登录到Oracle。

```
sudo su oracle
```

第7步：验证Oracle安装

您应确认Oracle安装成功。

步骤

1. 登录到Linux Oracle服务器并显示Oracle进程列表。这将确认安装按预期完成、并且Oracle数据库正在运行。

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

2. 登录到数据库以检查数据库配置并确认正确创建了PDB。

```
sqlplus / as sysdba
```

   您应看到类似于以下内容的输出：

```
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. 执行几个简单的SQL命令以确认数据库可用。

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

第8步：安装**azacsnap"**实用程序并执行快照备份

要执行快照备份、您需要安装并运行此 `azacsnap`实用程序。

步骤

1. 安装容器。

```
docker-compose up azacsnap_install
```

2. 切换到Snapshot用户帐户。

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

3. 配置存储备份详细信息文件。这将创建 `azacsnap.json`配置文件。

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

4. 执行快照备份。

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

第9步：(可选)将内部**PDB**迁移到云

您可以选择将内部PDB迁移到云。

步骤

1. 根据环境需要在文件中设置变量 `tfvars`。
2. 迁移PDB。

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

适用于 AWS 的 Cloud Volumes ONTAP

适用于**AWS**的**Cloud Volumes ONTAP—**突发到云

本文支持适用于AWS的NetApp Cloud Volumes ONTAP自动化解决方案、该解决方案可供NetApp客户从BlueXP 自动化目录中获取。

适用于AWS的Cloud Volumes ONTAP自动化解决方案可使用Terraform自动执行适用于AWS的Cloud Volumes ONTAP容器化部署、让您无需任何手动干预即可快速部署适用于AWS的Cloud Volumes ONTAP。

开始之前

- 您必须通过BlueXP Web UI下载"Cloud Volumes ONTAP AWS—突发到云"自动化解决方案。该解决方案打包为 `cvo_aws_flexcache.zip`。
- 您必须在与Cloud Volumes ONTAP相同的网络上安装Linux VM。
- 安装Linux VM后、必须按照本解决方案中的步骤安装所需的依赖项。

第1步：安装Docker和Docker构建

安装 Docker

以下步骤以Ubuntu 20.04 Debian Linux分发软件为例。您运行的命令取决于您使用的Linux分发软件。请参阅适用于您的配置的特定Linux分发软件文档。

步骤

1. 运行以下命令以安装Docker `sudo`：

```
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. 验证安装。

```
docker –version
```

3. 验证是否已在Linux系统上创建名为"Docker "的组。如有必要、请创建组：

```
sudo groupadd docker
```

4. 将需要访问Docker的用户添加到组：

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

5. 您的更改将在注销并重新登录到终端后应用。或者、您也可以立即应用更改：

```
newgrp docker
```

安装Docker配置

步骤

1. 运行以下命令以安装Docker配置 `sudo`：

```
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. 验证安装。

```
docker-compose –version
```

第2步：准备Docker映像

步骤

1. 将此文件夹复制 `cvo_aws_flexcache.zip`到要用于部署Cloud Volumes ONTAP的Linux VM：

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

   - `private-key.pem`是用于无密码登录的私钥文件。
   - `awsuser`是虚拟机用户名。
   - `IP_ADDRESS_OF_VM`是VM IP地址。
   - `LOCATION_TO_BE_COPIED`是复制文件夹的位置。

2. 提取 `cvo_aws_flexcache.zip`文件夹。您可以提取当前目录或自定义位置中的文件夹。

   要解压缩当前目录中的文件夹、请运行：

```
unzip cvo_aws_flexcache.zip
```

   要在自定义位置提取文件夹、请运行：

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

3. 解压缩内容后、导航到 `CVO_Aws_Deployment`文件夹并运行以下命令以查看文件：

```
 ls -la
```

您应看到一个文件列表、类似于以下示例：

```
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. 找到 `cvo_aws_flexcache_ubuntu_image.tar`文件。其中包含部署Cloud Volumes ONTAP for AWS所需的Docker映像。
5. 解压缩文件：

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

6. 等待几分钟、等待Docker映像加载完毕、然后验证是否已成功加载Docker映像：

```
docker images
```

您应看到一个名为且带有 `latest`标记的Docker映像 `cvo_aws_flexcache_ubuntu_image`、如以下示例所示：

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

> 您可以根据需要更改Docker映像名称。如果更改Docker映像名称、请确保在和 `docker-compose-destroy`文件中更新Docker映像名称 `docker-compose-deploy`。

第3步：创建环境变量文件

在此阶段、您必须创建两个环境变量文件。其中一个文件用于使用AWS访问密钥和机密密钥对AWS Resource Manager API进行身份验证。第二个文件用于设置环境变量、以使BlueXP Terraform模块能够找到AWS API并对其进行身份验证。

步骤

1. 在以下位置创建 `awsauth.env`文件：

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

   a. 将以下内容添加到文件中 `awsauth.env`：

      access_key=<> key_key=<>

   格式*必须*与上面显示的格式完全相同。

2. 将绝对文件路径添加到 `.env`文件中。

   输入与环境变量对应的环境文件的 `AWS_CREDS`**绝对路径** `awsauth.env`。

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

3. 导航到该 `cvo_aws_variable`文件夹并更新凭据文件中的访问权限和机密密钥。

   将以下内容添加到文件中：

   aws_access_key_id <>aws_key_access_key=<>

   格式*必须*与上面显示的格式完全相同。

第4步：将**Cloud Volumes ONTAP**许可证添加到**BlueXP** 或订阅**BlueXP**

您可以将Cloud Volumes ONTAP许可证添加到BlueXP 或订阅AWS Marketplace中的NetApp BlueXP 。

步骤

1. 在AWS门户中、导航到* SaaS 并选择*订阅**NetApp BlueXP** 。

   您可以使用与Cloud Volumes ONTAP相同的资源组，也可以使用不同的资源组。

2. 配置BlueXP 门户以将SaaS订阅导入到BlueXP 。

   您可以直接从AWS门户配置此配置。

   系统会将您重定向到BlueXP 门户以确认配置。

3. 选择*保存*，确认BlueXP 门户中的配置。

第5步：创建外部卷

您应创建一个外部卷、以保留Terraform状态文件和其他重要文件。您必须确保文件可供Terraform运行工作流和部署。

步骤

1. 在Docker撰写之外创建外部卷：

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

示例：

```
docker volume create cvo_aws_volume_dst
```

2. 使用以下选项之一：

   a. 向环境文件添加外部卷路径 `.env`。

      您必须遵循以下所示的确切格式。

      格式。

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

      示例：
      `PERSISTENT_VOL=cvo_aws_volume_dst:/cvo_aws`

   b. 将NFS共享添加为外部卷。

      请确保Docker容器可以与NFS共享进行通信、并且已配置正确的权限(例如读/写权限)。

      i. 在Docker编制文件中添加NFS共享路径作为外部卷的路径、如下所示：格式：

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

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

3. 导航到 `cvo_aws_variables`文件夹。

   您应在文件夹中看到以下变量文件：

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

4. 根据需要更改文件中的值 `terraform.tfvars`。

   修改文件中的任何变量值时、您必须阅读特定的支持文档 `terraform.tfvars`。根据地区、可用性区域以及Cloud Volumes ONTAP for AWS支持的其他因素、这些值可能会有所不同。其中包括单个节点和高可用性(HA)对的许可证、磁盘大小和VM大小。

   文件中已定义连接器和Cloud Volumes ONTAP Terraform模块的所有支持变量 `variables.tf`。在添加到文件之前、必须引用文件 `terraform.tfvars`中的变量名称 `variables.tf`。

5. 根据您的要求，您可以通过将以下选项设置为或来启用或 `false`禁用FlexCache和FlexClone `true`。

   以下示例将启用FlexCache和FlexClone：

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

第6步：部署Cloud Volumes ONTAP for AWS

按照以下步骤部署Cloud Volumes ONTAP for AWS。

步骤

1. 从根文件夹中、运行以下命令以触发部署：

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

   此时将触发两个容器、第一个容器部署Cloud Volumes ONTAP、第二个容器将遥测数据发送到AutoSupport。

   第二个容器将等待、直到第一个容器成功完成所有步骤。

2. 使用日志文件监控部署过程的进度：

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

   此命令可实时提供输出并捕获以下日志文件中的数据：
   `deployment.log`

   `telemetry_asup.log`

   您可以通过使用以下环境变量编辑这些日志文件来更改其名称 `.env`：

   `DEPLOYMENT_LOGS`

   `TELEMETRY_ASUP_LOGS`

   以下示例显示了如何更改日志文件名：

   `DEPLOYMENT_LOGS=<your_deployment_log_filename>.log`

   `TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log`

完成后

您可以使用以下步骤删除临时环境并清理在部署过程中创建的项目。

步骤

1. 如果您已部署FlexCache、请在变量文件中设置以下选项 `terraform.tfvars`、这样将清理FlexCache卷

并删除先前创建的临时环境。

```
flexcache_operation = "destroy"
```

可能的选项包括 `deploy`和 `destroy`。

2. 如果您已部署FlexClone、请在变量文件中设置以下选项 `terraform.tfvars`、这样将清理FlexClone卷并删除先前创建的临时环境。

```
flexclone_operation = "destroy"
```

可能的选项包括 `deploy`和 `destroy`。

适用于 Azure 的 Cloud Volumes ONTAP

适用于Azure的Cloud Volumes ONTAP—突发到云

本文支持适用于Azure的NetApp Cloud Volumes ONTAP自动化解决方案、该解决方案可供NetApp客户从BlueXP 自动化目录获得。

适用于Azure的Cloud Volumes ONTAP自动化解决方案使用Terraform自动执行适用于Azure的Cloud Volumes ONTAP容器化部署、让您无需任何手动干预即可快速部署适用于Azure的Cloud Volumes ONTAP。

开始之前

- 您必须通过BlueXP Web UI下载"Cloud Volumes ONTAP Azure—突发到云"自动化解决方案。该解决方案打包为 `CVO-Azure-Burst-To-Cloud.zip`。
- 您必须在与Cloud Volumes ONTAP相同的网络上安装Linux VM。
- 安装Linux VM后、必须按照本解决方案中的步骤安装所需的依赖项。

第1步：安装Docker和Docker构建

安装 Docker

以下步骤以Ubuntu 20.04 Debian Linux分发软件为例。您运行的命令取决于您使用的Linux分发软件。请参阅适用于您的配置的特定Linux分发软件文档。

步骤

1. 运行以下命令以安装Docker `sudo`：

```
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. 验证安装。

```
docker –version
```

3. 验证是否已在Linux系统上创建名为"Docker "的组。如有必要、请创建组：

```
sudo groupadd docker
```

4. 将需要访问Docker的用户添加到组：

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

5. 您的更改将在注销并重新登录到终端后应用。或者、您也可以立即应用更改：

```
newgrp docker
```

安装Docker配置

步骤

1. 运行以下命令以安装Docker配置 `sudo`：

```
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. 验证安装。

```
docker-compose –version
```

第2步：准备Docker映像

步骤

1. 将此文件夹复制 `CVO-Azure-Burst-To-Cloud.zip`到要用于部署Cloud Volumes ONTAP的Linux VM：

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

   - `private-key.pem`是用于无密码登录的私钥文件。
   - `azureuser`是虚拟机用户名。
   - `IP_ADDRESS_OF_VM`是VM IP地址。
   - `LOCATION_TO_BE_COPIED`是复制文件夹的位置。

2. 提取 `CVO-Azure-Burst-To-Cloud.zip`文件夹。您可以提取当前目录或自定义位置中的文件夹。

   要解压缩当前目录中的文件夹、请运行：

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

   要在自定义位置提取文件夹、请运行：

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

3. 解压缩内容后、导航到 `CVO_Azure_Deployment`文件夹并运行以下命令以查看文件：

```
 ls -la
```

   您应看到一个文件列表、类似于以下示例：

```
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. 找到 `cvo_azure_flexcache_ubuntu_image_latest.tar.gz`文件。其中包含部署Cloud Volumes ONTAP for Azure所需的Docker映像。
5. 解压缩文件：

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

6. 等待几分钟、等待Docker映像加载完毕、然后验证是否已成功加载Docker映像：

```
docker images
```

您应看到一个名为且带有 `latest`标记的Docker映像 `cvo_azure_flexcache_ubuntu_image_latest、如以下示例所示：

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

第3步：创建环境变量文件

在此阶段、您必须创建两个环境变量文件。其中一个文件用于使用服务主体凭据对Azure Resource Manager API进行身份验证。第二个文件用于设置环境变量、以使BlueXP Terraform模块能够找到Azure API并对其进行身份验证。

步骤

1. 创建服务主体。

   在创建环境变量文件之前，必须按照中的步骤创建服务主体"创建可以访问资源的Azure Active Directory应用程序和服务主体"。

2. 将*贡献方*角色分配给新创建的服务主体。

3. 创建自定义角色。
   a. 找到该 `sp_role.json`文件、然后在列出的操作下检查所需的权限。
   b. 插入这些权限并将自定义角色附加到新创建的服务主体。
4. 导航到*Certificates & Secretes*并选择*New client Secret*以创建客户端机密。

   创建客户端密钥时，必须记录*value*列中的详细信息，因为您将无法再看到此值。您还必须记录以下信息：

   - 客户端 ID
   - 订阅ID
   - 租户ID

   创建环境变量时需要此信息。您可以在服务主体UI的*Overview*部分中找到客户端ID和租户ID信息。
5. 创建环境文件。
   a. 在以下位置创建 `azureauth.env`文件：

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

      i. 将以下内容添加到文件中：

      ClientID=<>clientSecret=<>下标Id=<>租户ID=<>

   格式“必须”与上面所示完全相同，键和值之间没有空格。

   b. 在以下位置创建 `credentials.env`文件：

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

      i. 将以下内容添加到文件中：

      Azue_租 户ID=<>Azue_client_SECLE=<>Azue_client_ID=<>Azue_Probation_ID=<>

   格式“必须”与上面所示完全相同，键和值之间没有空格。
6. 将绝对文件路径添加到文件中 `.env`。

   在与环境变量对应的文件中输入环境文件 `.env`的 `AZURE_RM_CREDS`绝对路径 `azureauth.env`。

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

   在与环境变量对应的文件中输入环境文件 `.env`的 `BLUEXP_TF_AZURE_CREDS`绝对路径 `credentials.env`。

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

第4步：将Cloud Volumes ONTAP许可证添加到BlueXP 或订阅BlueXP

您可以将Cloud Volumes ONTAP许可证添加到BlueXP 或订阅Azure Marketplace中的NetApp BlueXP 。

步骤

1. 在Azure门户中、导航到* SaaS 并选择*订阅**NetApp BlueXP** 。
2. 选择*云管理器(按小时、WORM和数据服务划分的容量PYGO)*计划。

   您可以使用与Cloud Volumes ONTAP相同的资源组，也可以使用不同的资源组。

3. 配置BlueXP 门户以将SaaS订阅导入到BlueXP 。

   您可以通过导航到*产品和计划详细信息*并选择*立即配置帐户*选项、直接从Azure门户配置此帐户。

   然后、您将重定向到BlueXP 门户以确认配置。

4. 选择*保存*，确认BlueXP 门户中的配置。

第**5**步：创建外部卷

您应创建一个外部卷、以保留Terraform状态文件和其他重要文件。您必须确保文件可供Terraform运行工作流和部署。

步骤

1. 在Docker撰写之外创建外部卷：

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

   示例：

   ```
   docker volume create cvo_azure_volume_dst
   ```

2. 使用以下选项之一：

   a. 向环境文件添加外部卷路径 `.env`。

      您必须遵循以下所示的确切格式。

      格式。

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

      示例：
      ```
      PERSISTENT_VOL=cvo_azure_volume_dst:/cvo_azure
      ```

   b. 将NFS共享添加为外部卷。

      请确保Docker容器可以与NFS共享进行通信、并且已配置正确的权限(例如读/写权限)。

      i. 在Docker编制文件中添加NFS共享路径作为外部卷的路径、如下所示：格式：

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

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

3. 导航到 `cvo_azure_variables`文件夹。

   您应在该文件夹中看到以下变量文件：

   `terraform.tfvars`

   `variables.tf`

4. 根据需要更改文件中的值 `terraform.tfvars`。

   修改文件中的任何变量值时、您必须阅读特定的支持文档 `terraform.tfvars`。根据地区、可用性区域以及适用于Azure的Cloud Volumes ONTAP支持的其他因素、这些值可能会有所不同。其中包括单个节点和高可用性(HA)对的许可证、磁盘大小和VM大小。

   文件中已定义连接器和Cloud Volumes ONTAP Terraform模块的所有支持变量 `variables.tf`。在添加到文件之前、必须引用文件 `terraform.tfvars`中的变量名称 `variables.tf`。

5. 根据您的要求，您可以通过将以下选项设置为或来启用或 `false`禁用`FlexCache`和`FlexClone` `true`。

   以下示例将启用FlexCache和FlexClone：

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

6. 如有必要、您可以从Azure Active Directory服务检索Terraform变量的值 `az_service_principal_object_id`：

   a. 导航到*企业应用程序–>所有应用程序*，然后选择您先前创建的服务主体的名称。

   b. 复制对象ID并插入Terraform变量的值：

      `az_service_principal_object_id`

第6步：部署Cloud Volumes ONTAP for Azure

请按照以下步骤部署Cloud Volumes ONTAP for Azure。

步骤

1. 从根文件夹中、运行以下命令以触发部署：

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

   此时将触发两个容器、第一个容器部署Cloud Volumes ONTAP、第二个容器将遥测数据发送到AutoSupport。

   第二个容器将等待、直到第一个容器成功完成所有步骤。

2. 使用日志文件监控部署过程的进度：

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

此命令可实时提供输出并捕获以下日志文件中的数据：

`deployment.log`

`telemetry_asup.log`

您可以通过使用以下环境变量编辑这些日志文件来更改其名称 `.env`：

`DEPLOYMENT_LOGS`

`TELEMETRY_ASUP_LOGS`

以下示例显示了如何更改日志文件名：

`DEPLOYMENT_LOGS=<your_deployment_log_filename>.log`

`TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log`

完成后

您可以使用以下步骤删除临时环境并清理在部署过程中创建的项目。

步骤

1. 如果您已部署FlexCache、请在文件中设置以下选项 `terraform.tfvars`、这样将清理FlexCache卷并删除先前创建的临时环境。

```
flexcache_operation = "destroy"
```

可能的选项包括 `deploy`和 `destroy`。

2. 如果您已部署FlexClone、请在文件中设置以下选项 `terraform.tfvars`、这样将清理FlexClone卷并删除先前创建的临时环境。

```
flexclone_operation = "destroy"
```

可能的选项包括 `deploy`和 `destroy`。

适用于 Google Cloud 的 Cloud Volumes ONTAP

适用于Google Cloud的Cloud Volumes ONTAP—突发到云

本文支持适用于Google云自动化的NetApp Cloud Volumes ONTAP解决方案、该解决方案可供NetApp客户从BlueXP 自动化目录获得。

适用于Google Cloud的Cloud Volumes ONTAP自动化解决方案可自动执行适用于Google Cloud的Cloud Volumes ONTAP容器化部署、让您无需任何手动干预即可快速部署适用于Google Cloud的Cloud Volumes ONTAP。

开始之前

- 您必须通过BlueXP Web UI下载"适用于Google Cloud的Cloud Volumes ONTAP—突发到云"自动化解决方案。该解决方案打包为 `cvo_gcp_flexcache.zip`。
- 您必须在与Cloud Volumes ONTAP相同的网络上安装Linux VM。
- 安装Linux VM后、必须按照本解决方案中的步骤安装所需的依赖项。

第1步：安装Docker和Docker构建

安装 Docker

以下步骤以Ubuntu 20.04 Debian Linux分发软件为例。您运行的命令取决于您使用的Linux分发软件。请参阅适用于您的配置的特定Linux分发软件文档。

步骤

1. 运行以下命令以安装Docker：

```
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. 验证安装。

```
docker –version
```

3. 验证是否已在Linux系统上创建名为"Docker "的组。如有必要、请创建组：

```
sudo groupadd docker
```

4. 将需要访问Docker的用户添加到组：

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

5. 您的更改将在注销并重新登录到终端后应用。或者、您也可以立即应用更改：

```
newgrp docker
```

安装Docker配置

步骤

1. 运行以下命令以安装Docker配置 `sudo`：

```
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. 验证安装。

```
docker-compose –version
```

第2步：准备Docker映像

步骤

1. 将此文件夹复制 `cvo_gcp_flexcache.zip`到要用于部署Cloud Volumes ONTAP的Linux VM：

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

   - `private-key.pem`是用于无密码登录的私钥文件。
   - `gcpuser`是虚拟机用户名。
   - `IP_ADDRESS_OF_VM`是VM IP地址。
   - `LOCATION_TO_BE_COPIED`是复制文件夹的位置。

2. 提取 `cvo_gcp_flexcache.zip`文件夹。您可以提取当前目录或自定义位置中的文件夹。

   要解压缩当前目录中的文件夹、请运行：

```
unzip cvo_gcp_flexcache.zip
```

   要在自定义位置提取文件夹、请运行：

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

3. 解压缩内容后、运行以下命令以查看文件：

```
 ls -la
```

您应看到一个文件列表、类似于以下示例：

```
    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. 找到 `cvo_gcp_flexcache_ubuntu_image.tar`文件。其中包含部署适用于Google Cloud的Cloud Volumes ONTAP所需的Docker映像。
5. 解压缩文件：

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

6. 等待几分钟、等待Docker映像加载完毕、然后验证是否已成功加载Docker映像：

```
docker images
```

您应看到一个名为且带有 `latest`标记的Docker映像 `cvo_gcp_flexcache_ubuntu_image`、如以下示例所示：

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

> 您可以根据需要更改Docker映像名称。如果更改Docker映像名称、请确保在和 `docker-compose-destroy`文件中更新Docker映像名称 `docker-compose-deploy`。

第3步：更新JSON文件

在此阶段、您必须使用服务帐户密钥更新此 `cxo-automation-gcp.json`文件、以便对Google Cloud提供程序进行身份验证。

1. 创建一个具有部署Cloud Volumes ONTAP和BlueXP 连接器权限的服务帐户。"了解有关创建服务帐户的更多信息。"
2. 下载帐户的密钥文件并使用密钥文件信息更新此 `cxo-automation-gcp.json`文件。 `cxo-automation-gcp.json`文件位于文件夹中 `cvo_gcp_variables。

   示例

```
{
  "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"
}
```

   文件格式必须与上述格式完全相同。

第4步：订阅BlueXP

您可以在Google云市场中订阅NetApp BlueXP 。

步骤

1. 导航到"Google Cloud控制台"并选择*订阅NetApp BlueXP *。
2. 配置BlueXP 门户以将SaaS订阅导入到BlueXP 。

   您可以直接从Google Cloud Platform配置此功能。系统将重定向到BlueXP 门户以确认配置。

3. 选择*保存*，确认BlueXP 门户中的配置。

有关详细信息，请参见 "管理BlueXP的Google Cloud凭据和订阅"。

第5步：启用所需的Google Cloud API

要部署Cloud Volumes ONTAP和连接器、您必须在项目中启用以下Google Cloud API。

- Cloud Deployment Manager V2 API

- 云日志记录 API
- Cloud Resource Manager API
- 计算引擎 API
- 身份和访问管理（ IAM ） API

"了解有关启用 API 的更多信息"

第6步：创建外部卷

您应创建一个外部卷、以保留Terraform状态文件和其他重要文件。您必须确保文件可供Terraform运行工作流和部署。

步骤

1. 在Docker撰写之外创建外部卷：

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

示例：

```
docker volume create cvo_gcp_volume_dst
```

2. 使用以下选项之一：

   a. 向环境文件添加外部卷路径 `.env`。

   您必须遵循以下所示的确切格式。

   格式。

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

   示例：
   `PERSISTENT_VOL=cvo_gcp_volume_dst:/cvo_gcp`

   b. 将NFS共享添加为外部卷。

   请确保Docker容器可以与NFS共享进行通信、并且已配置正确的权限(例如读/写权限)。

      i. 在Docker编制文件中添加NFS共享路径作为外部卷的路径、如下所示：格式：

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

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

3. 导航到 `cvo_gcp_variables`文件夹。

您应在该文件夹中看到以下文件：

- terraform.tfvars
- variables.tf

4. 根据需要更改文件中的值 terraform.tfvars。

   修改文件中的任何变量值时、您必须阅读特定的支持文档 terraform.tfvars。根据地区、可用性区域以及适用于Google Cloud的Cloud Volumes ONTAP支持的其他因素、这些值可能会有所不同。其中包括单个节点和高可用性(HA)对的许可证、磁盘大小和VM大小。

   文件中已定义连接器和Cloud Volumes ONTAP Terraform模块的所有支持变量 variables.tf。在添加到文件之前、必须引用文件 terraform.tfvars`中的变量名称 `variables.tf。

5. 根据您的要求，您可以通过将以下选项设置为或来启用或 false`禁用FlexCache和FlexClone `true。

   以下示例将启用FlexCache和FlexClone：

   - is_flexcache_required = true
   - is_flexclone_required = true

第7步：部署适用于Google Cloud的Cloud Volumes ONTAP

按照以下步骤部署适用于Google Cloud的Cloud Volumes ONTAP。

步骤

1. 从根文件夹中、运行以下命令以触发部署：

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

   此时将触发两个容器、第一个容器部署Cloud Volumes ONTAP、第二个容器将遥测数据发送到AutoSupport。

   第二个容器将等待、直到第一个容器成功完成所有步骤。

2. 使用日志文件监控部署过程的进度：

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

   此命令可实时提供输出并捕获以下日志文件中的数据：
   deployment.log

   telemetry_asup.log

   您可以通过使用以下环境变量编辑这些日志文件来更改其名称 .env：

   DEPLOYMENT_LOGS

```
TELEMETRY_ASUP_LOGS
```

以下示例显示了如何更改日志文件名：

```
DEPLOYMENT_LOGS=<your_deployment_log_filename>.log

TELEMETRY_ASUP_LOGS=<your_telemetry_asup_log_filename>.log
```

完成后

您可以使用以下步骤删除临时环境并清理在部署过程中创建的项目。

步骤

1. 如果您已部署FlexCache、请在文件中设置以下选项 `terraform.tfvars`、这样将清理FlexCache卷并删除先前创建的临时环境。

```
flexcache_operation = "destroy"
```

> 可能的选项包括 `deploy`和 `destroy`。

2. 如果您已部署FlexClone、请在文件中设置以下选项 `terraform.tfvars`、这样将清理FlexClone卷并删除先前创建的临时环境。

```
flexclone_operation = "destroy"
```

> 可能的选项包括 `deploy`和 `destroy`。

ONTAP

Day 1

ONTAP Day 1解决方案概述

您可以使用Day 1自动化解决方案使用ONTAP部署和配置ONTAP集群。该解决方案可从获得"BlueXP自动化目录"。

灵活的ONTAP部署选项

根据您的要求、您可以使用内部硬件或模拟ONTAP来使用ONTAP部署和配置。

内部硬件

您可以使用运行ONTAP的内部硬件(例如FAS或AFF系统)部署此解决方案。您必须使用Linux VM使用ONTAP来部署和配置Storage Virtual Machine集群。

模拟ONTAP

要使用ONTAP模拟器部署此解决方案、您必须从NetApp支持站点下载最新版本的sim模拟ONTAP。模拟ONTAP是ONTAP软件的虚拟模拟器。模拟ONTAP在Windows、Linux或Mac系统上的VMware虚拟机管理程序中运行。对于Windows和Linux主机、必须使用VMware Workstation虚拟机管理程序来运行此解决方案。如果您使用的是Mac操作系统、请使用VMware Fusion虚拟机管理程序。

分层设计

通过该框架、可以简化自动化执行和逻辑任务的开发和重复使用。该框架区分了自动化中的决策任务(逻辑层)和执行步骤(执行层)。通过了解这些层的工作原理、您可以自定义配置。

一本安可赛"操作手册"从头到尾执行一系列任务。该 `site.yml`手册包含该手册 `logic.yml`和该 `execution.yml`手册。

运行请求时，该 `site.yml`操作手册首先调用该操作手册，然后调用该 `logic.yml`操作手册 `execution.yml`以执行服务请求。

您不需要使用框架的逻辑层。逻辑层提供了一些选项、可将框架的功能扩展到硬编码值之外、以供执行。这样、您可以根据需要自定义框架功能。

逻辑层

逻辑层由以下部分组成：

- 这是一本操作 `logic.yml`手册
- 目录中的逻辑任务文件 `logic-tasks`

逻辑层提供了复杂决策的功能、而无需进行大量的自定义集成(例如、连接到ServiceNow)。逻辑层是可配置的、可为微服务提供输入。

此外、还可以绕过逻辑层。如果要绕过逻辑层、请勿定义 `logic_operation`变量。通过直接调用该 `logic.yml`操作手册、可以在不执行的情况下执行某种级别的调试。您可以使用"debug "语句验证的值是否 `raw_service_request`正确。

重要注意事项：

- 该手册将 `logic.yml`搜索 `logic_operation`变量。如果在请求中定义了变量、则它会从目录中加载任务文件 `logic-tasks`。任务文件必须是.yml文件。如果没有匹配的任务文件且 `logic_operation`已定义变量、则逻辑层将失败。
- 变量的默认值 `logic_operation`为 `no-op`。如果未明确定义变量，则默认为 `no-op`，而不运行任何操作。
- 如果 `raw_service_request`已定义变量、则执行将继续到执行层。如果未定义此变量、则逻辑层将失败。

执行层

执行层由以下部分组成：

- 这是一本操作 `execution.yml`手册

执行层通过API调用来配置ONTAP集群。该 `execution.yml`手册要求 `raw_service_request`在执行时定义变量。

支持自定义

您可以根据自己的要求以各种方式自定义此解决方案。

自定义选项包括：

- 正在修改《安赛尔操作手册》

- 添加角色

自定义**Ans**｝文件

下表介绍了此解决方案中包含的可自定义的Ansable文件。

位置	说明
`playbooks/inventory/hosts`	包含一个包含主机和组列表的文件。
`playbooks/group_vars/all/*`	通过Ans可 轻松地一次性将变量应用于多个主机。您可以修改此文件夹中的任何或所有文件，包括 `cfg.yml`、、 `clusters.yml` `defaults.yml`、`services.yml`、 `standards.yml`和 `vault.yml`。
`playbooks/logic-tasks`	支持在Ans可 内执行决策任务、并保持逻辑与执行的分离。您可以向此文件夹添加与相关服务对应的文件。
`playbooks/vars/*`	在《Andsability操作手册》和角色中使用动态值、以实现配置的自定义、灵活性和可重用性。如有必要、您可以修改此文件夹中的任何或所有文件。

自定义角色

此外、您还可以通过添加或更改Ans担负 的角色(也称为微服务)来自定义解决方案。有关详细信息，请参见"自定义"。

准备使用**ONTAP Day 1**解决方案

在部署自动化解决方案之前、您必须准备ONTAP环境并安装和配置Ans得。

初始规划注意事项

在使用此解决方案部署ONTAP集群之前、您应查看以下要求和注意事项。

基本要求

要使用此解决方案、您必须满足以下基本要求：

- 您必须能够在内部或通过ONTAP模拟器访问ONTAP软件。
- 您必须了解如何使用ONTAP软件。
- 您必须了解如何使用Ans得 自动化软件工具。

规划注意事项

在部署此自动化解决方案之前、您必须确定：

- 要运行Ans得 可控制节点的位置。
- ONTAP系统、内部硬件或ONTAP模拟器。
- 是否需要自定义。

无论您是使用内部ONTAP系统还是模拟ONTAP、都必须先准备好环境、然后才能部署自动化解决方案。

(可选)安装和配置模拟ONTAP

如果要通过ONTAP模拟器部署此解决方案、则必须下载并运行模拟ONTAP。

开始之前

- 您必须下载并安装要用于运行模拟ONTAP的VMware虚拟机管理程序。
  - 如果您使用的是Windows或Linux操作系统、请使用VMware Workstation。
  - 如果您使用的是Mac操作系统、请使用VMware Fusion。

> 如果您使用的是Mac OS、则必须使用Intel处理器。

步骤

使用以下过程在本地环境中安装两个ONTAP管理器：

1. 从下载模拟ONTAP"NetApp 支持站点"。

   > 尽管您安装了两个ONTAP联机器、但只需下载一份软件副本即可。

2. 如果尚未运行、请启动VMware应用程序。
3. 找到已下载的模拟器文件、然后右键单击以使用VMware应用程序打开该文件。
4. 设置第一个ONTAP实例的名称。
5. 等待模拟器启动、然后按照说明创建单节点集群。

   对第二个ONTAP实例重复上述步骤。

6. (可选)添加完整磁盘补充。

   从每个集群中、运行以下命令：

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

ONTAP系统的状态

您必须验证ONTAP系统的初始状态、无论是内部还是通过ONTAP模拟器运行。

验证是否满足以下ONTAP系统要求：

- ONTAP已安装并正在运行、但尚未定义集群。
- 此时、ONTAP将启动并显示用于访问集群的IP地址。
- 网络可访问。
- 您具有管理员凭据。
- 此时将显示每日消息(Message of the Day、MOTD)横幅以及管理地址。

安装所需的自动化软件

本节介绍有关如何安装Ands还是 准备部署自动化解决方案的信息。

安装**Ands**器

可以在Linux或Windows系统上安装Ans得。

Ands得以 与ONTAP集群进行通信的默认通信方法是SSH。

请参阅"NetApp 和 Ansible 入门：安装 Ansible"安装Ands得以 安装。

必须在系统的控制节点上安装有Ans得。

下载并准备自动化解决方案

您可以使用以下步骤下载并准备要部署的自动化解决方案。

1. 通过BlueXP Web UI下载"ONTAP - Day 1和amp；运行状况检查"自动化解决方案。该解决方案打包为 `ONTAP_DAY0_DAY1.zip`。
2. 提取zip文件夹、并将文件复制到您的Ands得以 环境中控制节点上的所需位置。

初始的**Ands**处理 框架配置

执行Ans担负 框架的初始配置：

1. 导航到 `playbooks/inventory/group_vars/all`。
2. 解密 `vault.yml`文件：

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

   当系统提示您输入存储密码时、输入以下临时密码：

   ```
   NetApp123!
   ```

   "NetApp 123！"是用于对文件和相应的存储密码进行解密的临时 `vault.yml`密码。首次使用后，*必须*使用您自己的密码对文件进行加密。

3. 修改以下的Ans的 文件：

   - `clusters.yml`修改此文件中的值以适合您的环境。

- vault.yml-解密文件后、根据您的环境修改ONTAP集群、用户名和密码值。
- cfg.yml-设置的文件路径 log2file，并在下 cfg`设置 `show_request`为 `True`以显示 `raw_service_request。

  此 `raw_service_request`变量将显示在日志文件中以及执行期间。

  列出的每个文件都包含注释、并说明如何根据您的要求对其进行修改。

4. 重新加密 `vault.yml`文件：

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

  系统会提示您在加密时为存储选择新密码。

5. 导航到 `playbooks/inventory/hosts`并设置有效的Python解释器。
6. 部署 `framework_test`服务：

  以下命令运行值为 cluster_identity_info`的 `na_ontap_info`模块 `gather_subset。这将验证基本配置是否正确、并验证您是否可以与集群进行通信。

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

  对每个集群运行命令。

  如果成功、您应看到类似于以下示例的输出：

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

使用该解决方案部署**ONTAP**集群

完成准备和规划后、您便可以使用ONTAP Day 1解决方案使用ONTAP来快速配置使用Ans得 集群了。

在执行本节中的步骤期间、您可以随时选择测试请求、而不是实际执行请求。要测试请求，请将命令行上的操作手册更 site.yml`改为 `logic.yml。

该 `docs/tutorial-requests.txt`位置包含此过程中使用的所有服务请求的最终版本。如果您在运行服务请求时遇到困难、可以将相关请求从文件复制 `tutorial-requests.txt`到该 `playbooks/inventory/group_vars/all/tutorial-requests.yml`位置、并根据需要修改硬编码值(IP地址、聚合名称等)。然后、您应该能够成功运行此请求。

开始之前

- 您必须安装了Ans得。
- 您必须已下载ONTAP Day 1解决方案并将该文件夹解压缩到了Ands得以 控制的节点上的所需位置。
- ONTAP系统状态必须满足要求、并且您必须具有必要的凭据。
- 您必须已完成本节中所述的所有必需任务"准备"。

本解决方案中的示例使用"Cluster_01"和"Cluster_02"作为两个集群的名称。您必须将这些值替换为环境中集群的名称。

第1步：初始集群配置

在此阶段、您必须执行一些初始集群配置步骤。

步骤

1. 导航到该 `playbooks/inventory/group_vars/all/tutorial-requests.yml`位置并查看 `cluster_initial`文件中的请求。对您的环境进行任何必要的更改。
2. 在文件夹中为服务请求创建一个文件 `logic-tasks`。例如，创建一个名为的文件 `cluster_initial.yml`。

   将以下行复制到新文件：

```
- 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. 定义 `raw_service_request`变量。

   您可以使用以下选项之一在文件夹中创建的文件 `logic-tasks`中定义 `raw_service_request`变量 `cluster_initial.yml`：

   - 选项**1**:手动定义 `raw_service_request`变量。

     使用编辑器打开 `tutorial-requests.yml`文件、并将内容从第11行复制到第165行。将内容粘贴到新文件中的变量 `cluster_initial.yml`下 `raw service request`、如以下示例所示：

显示示例

示例 `cluster_initial.yml`文件：

```
- 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:
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
```

```
        - 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 }}"
```

- 选项**2**:使用Jinja模板定义请求:

  您也可以使用以下Jinja模板格式获取该 `raw_service_request`值。

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

4. 对第一个集群执行初始集群配置：

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

继续操作前、请确认没有错误。

5. 对第二个集群重复此命令：

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

确认第二个集群没有错误。

向上滚动到Andsent输出的开头时、您应看到发送到框架的请求、如以下示例所示：

显示示例

```
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. 登录到每个ONTAP实例并验证请求是否成功。

第2步：配置集群间的生命周期

现在、您可以通过向请求添加LIF定义并定义 `ontap_interface`微服务来配置集群间LIF `cluster_initial`。

服务定义和请求共同确定操作：

- 如果您为服务定义中未包含的微服务提供服务请求、则不会执行此请求。
- 如果您为服务请求提供服务定义中定义的一个或多个微服务、但在请求中省略了该服务、则不会执行该请求。

该 `execution.yml`操作手册将按所列顺序扫描微服务列表、以评估服务定义：

- 如果请求中有一个条目，其词典密钥与微服务定义中包含的条目匹配 `args`，则执行该请求。
- 如果服务请求中没有匹配的条目、则会跳过此请求、并且不会出现错误。

步骤

1. 导航到 `cluster_initial.yml`先前创建的文件，并通过向请求定义添加以下行来修改请求：

```
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. 运行命令：

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

3. 登录到每个实例以检查是否已将这些RIF添加到集群：

显示示例

```
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.
```

输出显示已*未*添加Lifs。这是因为 `ontap_interface`仍需要在文件中定义微服务 `services.yml`。

4. 验证是否已将这些生命周期添加到此变量中 `raw_service_request`。

显示示例

以下示例显示已将这些生命周期管理器添加到请求中：

```
          "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. 在文件中 `services.yml`的下定义 `ontap_interface`微服务 `cluster_initial。

   将以下行复制到文件中以定义微服务：

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

6. `ontap_interface`已在请求和文件中定义微服务 `services.yml、请再次运行请求：

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

7. 登录到每个ONTAP实例并验证是否已添加这些LUN。

第3步：(可选)配置多个集群

如果需要、您可以在同一请求中配置多个集群。定义请求时、必须为每个集群提供变量名称。

步骤

1. 在文件中为第二个集群添加一个条目 `cluster_initial.yml`、以便在同一请求中配置这两个集群。

   以下示例将在添加第二个条目后显示 `ontap_aggr`字段。

```
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. 将更改应用于下的所有其他项目 `cluster_initial`。
3. 通过将以下行复制到文件来向请求添加集群对等关系：

```
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. 运行Ands处理 请求：

```
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>
```

第4步：初始SVM配置

在此过程的此阶段、您需要配置集群中的SVM。

步骤

1. 更新 `svm_initial`文件中的请求 `tutorial-requests.yml`以配置SVM和SVM对等关系。

   您必须配置以下内容：

   - SVM
   - SVM对等关系

- 每个SVM的SVM接口

2. 更新请求定义中的变量定义 `svm_initial`。您必须修改以下变量定义：
   - `cluster_name`
   - `vserver_name`
   - `peer_cluster_name`
   - `peer_vserver`

   要更新定义，请删除 `svm_initial`定义后的*‘{}'* `req_details`并添加正确的定义。

3. 在文件夹中为服务请求创建一个文件 `logic-tasks`。例如，创建一个名为的文件 `svm_initial.yml`。

   将以下行复制到文件：

```
- 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. 定义 `raw_service_request`变量。

   您可以使用以下选项之一为文件夹中 `logic-tasks`的定义 `raw_service_request`变量 `svm_initial`：

   - 选项**1**:手动定义 `raw_service_request`变量。

     使用编辑器打开 `tutorial-requests.yml`文件、并将内容从第179行复制到第222行。将内容粘贴到新文件中的变量 `svm_initial.yml`下 `raw service request`、如以下示例所示：


service: svm_initial
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

显示示例

示例 `svm_initial.yml`文件：

```
- 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
```

- 选项**2**:使用Jinja模板定义请求:

  您也可以使用以下Jinja模板格式获取该 `raw_service_request`值。

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

5. 运行请求：

```
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. 登录到每个ONTAP实例并验证配置。
7. 添加SVM接口。

   在文件中 `services.yml`的下定义 `ontap_interface`服务 `svm_initial`、然后再次运行请求：

```
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. 登录到每个ONTAP实例并验证是否已配置SVM接口。

第5步：(可选)动态定义服务请求

在前面的步骤中、 `raw_service_request`变量是硬编码的。这对于学习、开发和测试非常有用。您还可以动态生成服务请求。

如果您不想将所需的与更高级别的系统集成、则下一节提供了一个动态生成所需的选项 `raw_service_request`。

- 如果 `logic_operation`未在命令中定义变量、则该 `logic.yml`文件不会从文件夹导入任何文件 `logic-tasks`。这意味着`raw_service_request`必须在Ands还是 外部定义、并在执行时提供给框架。
- 文件夹中的任务文件名 `logic-tasks`必须与不带`.yml`扩展名的变量值匹配 `logic_operation`。
- 文件夹中的任务文件 `logic-tasks`会动态定义 `raw_service_request`。唯一的要求是将有效定义为相关文件中的最后一个任务。 `raw_service_request`

如何动态定义服务请求

可以通过多种方法应用逻辑任务来动态定义服务请求。下面列出了其中一些选项：

- 使用文件夹中的一个Ans｝任务文件 `logic-tasks`
- 调用返回适合转换为变量的数据的自定义角色 `raw_service_request`。
- 调用Ands得以 环境外部的另一个工具以提供所需数据。例如、对Active IQ Unified Manager的REST API调用。

以下示例命令使用文件为每个集群动态定义服务请求 `tutorial-requests.yml`：

```
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
```

第6步：部署ONTAP Day 1解决方案

在此阶段、您应已完成以下操作：

- 已根据您的要求查看和修改中的所有文件 `playbooks/inventory/group_vars/all`。每个文件中都有详细的注释、以帮助您进行更改。
- 已将任何所需的任务文件添加到 `logic-tasks`目录。
- 已将任何所需的数据文件添加到 `playbook/vars`目录。

使用以下命令部署ONTAP Day 1解决方案并验证部署的运行状况：

在此阶段、您应已解密并修改 `vault.yml`文件、并且必须使用新密码对其进行加密。

- 运行ONTAP Day 0服务：

```
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>
```

- 运行ONTAP Day 1服务：

```
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>
```

- 应用集群范围设置：

```
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>
```

- 运行运行状况检查：

```
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>
```

定制**ONTAP Day 1**解决方案

要根据您的需求自定义ONTAP Day 1解决方案、您可以添加或更改Ands还是 角色。

角色表示在Andsent框架内的微服务。每个微服务执行一个操作。例如、ONTAP Day 0是包含多个微服务的服务。

添加**Ands**处理 角色

您可以添加Ans可 通过角色为您的环境自定义解决方案。所需角色由在Ands还是 在框架中定义的服务定义来定义。

角色必须满足以下要求才能用作微服务：

- 接受变量中的参数列表 `args`。
- 对于每个块、使用具有特定要求的"Andsing"结构。

- 使用单个Ans还是 模块并在块中定义单个任务。
- 根据本节详细介绍的要求实施每个可用的模块参数。

所需的微服务结构

每个角色都必须支持以下变量：

- `mode`：如果将模式设置为角色，则 `test`尝试导入显示该角色所执行的 `test.yml`操作而不实际执行该角色。

  

  由于某些相互依赖关系、并非总是可以实现这一点。

- `status`：执行手册的整体状态。如果未将此值设置为 `success`、则不会执行此角色。
- `args`：具有与角色参数名称匹配的关键字的角色专用词典列表。
- `global_log_messages`：在执行播放手册期间收集日志消息。每次执行角色时都会生成一个条目。
- `log_name`：用于引用条目中的角色的名称 `global_log_messages`。
- `task_descr`：有关该角色的作用的简要说明。
- `service_start_time`：用于跟踪每个角色执行时间的时间戳。
- `playbook_status`：Ans负责人 的状态。
- `role_result`：包含角色输出并包含在条目中的每条消息中的变量 `global_log_messages`。

示例角色结构

以下示例提供了实施微服务的角色的基本结构。您必须更改此示例中适用于您的配置的变量。

显示示例

基本角色结构：

```
- 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 ] }}"
```

示例角色中使用的变量：

- <NAME>：必须为每个微服务提供的可替换值。
- <LOG_NAME>：用于日志记录的角色的简称。例如， ONTAP_VOLUME。
- <TASK_DESCRIPTION>：微服务功能的简要说明。
- <MODULE_NAME>：任务的Ans担负 模块名称。

> 顶级 execute.yml`操作手册用于指定 `netapp.ontap`集合。如果模块是集合的一部分 `netapp.ontap，则无需完全指定模块名称。

- <MODULE_SPECIFIC_PARAMETERS>:特定于用于实施微服务的模块的Ans得 模块参数。以下列表介绍了参数的类型以及应如何对其进行分组。
  - 必需参数：所有必需参数均指定、无默认值。
  - 具有特定于微服务的默认值的参数(与模块文档指定的默认值不同)。
  - 所有剩余参数均 `default(omit)`用作默认值。

使用多层词典作为模块参数

某些NetApp提供的Ans典 模块使用多级别词典作为模块参数(例如、固定和自适应QoS策略组)。

如果使用这些词典、则单独使用 `default(omit)`不起作用、尤其是当有多个词典且它们互斥时。

如果您需要使用多级别词典作为模块参数、则应将此功能拆分为多个微服务(角色)、以确保每个微服务都能为相关词典至少提供一个二级词典值。

以下示例显示了固定和自适应QoS策略组、它们分别分布在两个微服务中。

第一个微服务包含固定的QoS策略组值：

```
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)
}}"
```

第二个微服务包含自适应QoS策略组值：

```
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) }}"
```