Toolkits für DB-Automatisierung

NetApp Solutions

NetApp
April 26, 2024

Inhalt

Toolkits für DB-Automatisierung . . . 1
SnapCenter Lifecycle Automation für Oracle-Klone . . . 1
Automatisierte Oracle-Migration . . . 5
Automatisierte Oracle HA/DR in AWS FSX ONTAP . . . 9
Bereitstellung von AWS FSX ONTAP-Clustern und EC2-Instanzen . . . 15

Toolkits für DB-Automatisierung

SnapCenter Lifecycle Automation für Oracle-Klone

Allen Cao, Niyaz Mohamed, NetApp

Zweck

Kunden sind begeistert von der FlexClone Funktion von NetApp ONTAP Storage für Datenbanken, mit deutlichen Einsparungen bei den Storage-Kosten. Dieses Ansible-basierte Toolkit automatisiert die Einrichtung, das Klonen und die Aktualisierung von geklonten Oracle Datenbanken anhand der NetApp SnapCenter Befehlszeilen-Dienstprogramme für ein optimiertes Lifecycle Management. Das Toolkit ist auf Oracle-Datenbanken anwendbar, die auf ONTAP Storage entweder bei Vorliegen oder in der Public Cloud bereitgestellt und über das UI Tool NetApp SnapCenter gemanagt werden.

Diese Lösung eignet sich für folgende Anwendungsfälle:

- Richten Sie die Konfigurationsdatei für die Klonspezifikation der Oracle-Datenbank ein.
- Erstellen und aktualisieren Sie die Oracle-Datenbank nach benutzerdefiniertem Zeitplan.

Zielgruppe

Diese Lösung ist für folgende Personen gedacht:

- Ein DBA, der Oracle Datenbanken mit SnapCenter managt.
- Ein Storage-Administrator, der ONTAP Storage mit SnapCenter managt
- Ein Anwendungseigentümer, der Zugriff auf die SnapCenter-Benutzeroberfläche hat.

Lizenz

Durch den Zugriff auf, das Herunterladen, die Installation oder die Verwendung der Inhalte in diesem GitHub-Repository stimmen Sie den Bedingungen der in dargelegten Lizenz zu "Lizenzdatei".

> Es gibt bestimmte Beschränkungen bezüglich der Erstellung und/oder Freigabe von abgeleiteten Arbeiten mit dem Inhalt in diesem GitHub-Repository. Bitte lesen Sie die Lizenzbedingungen, bevor Sie den Inhalt verwenden. Wenn Sie nicht allen Bedingungen zustimmen, dürfen Sie nicht auf den Inhalt dieses Repositorys zugreifen, ihn herunterladen oder verwenden.

Lösungsimplementierung

Voraussetzungen für die Bereitstellung

Die Bereitstellung erfordert die folgenden Voraussetzungen.

```
Ansible controller:
  Ansible v.2.10 and higher
  ONTAP collection 21.19.1
  Python 3
  Python libraries:
    netapp-lib
    xmltodict
    jmespath
```

```
SnapCenter server:
  version 5.0
  backup policy configured
  Source database protected with a backup policy
```

```
Oracle servers:
  Source server managed by SnapCenter
  Target server managed by SnapCenter
  Target server with identical Oracle software stack as source server
installed and configured
```

Toolkit herunterladen

```
git clone https://bitbucket.ngage.netapp.com/scm/ns-
bb/na_oracle_clone_lifecycle.git
```

Dateikonfiguration der Ansible Ziel-Hosts

Das Toolkit enthält eine Host-Datei, die die Ziele definiert, für die ein Ansible-Playbook ausgeführt wird. In der Regel sind dies die Ziel-Clones-Hosts von Oracle. Im Folgenden finden Sie eine Beispieldatei. Ein Hosteintrag enthält die IP-Adresse des Zielhosts sowie den SSH-Schlüssel für den Zugriff eines Admin-Benutzers auf den Host, um den Klon- oder Aktualisierungsbefehl auszuführen.

#Oracle-Clone-Hosts

```
[clone_1]
ora_04.cie.netapp.com ansible_host=10.61.180.29
ansible_ssh_private_key_file=ora_04.pem
```

```
[clone_2]
[clone_3]
```

Konfiguration globaler Variablen

Die Ansible-Playbooks verwenden variable Eingaben aus mehreren variablen Dateien. Unten finden Sie ein Beispiel für die globale Variablendatei VARs.yml.

```
# ONTAP specific config variables
# SnapCtr specific config variables
```

```
snapctr_usr: xxxxxxxx
snapctr_pwd: 'xxxxxxxx'
```

```
backup_policy: 'Oracle Full offline Backup'
# Linux specific config variables
# Oracle specific config variables
```

Konfiguration der Host-Variablen

Hostvariablen werden im Verzeichnis Host_VARs mit dem Namen {{ Host_Name }}.yml definiert. Unten ist ein Beispiel für die Oracle-Zieldatei ora_04.cie.netapp.com.yml, die eine typische Konfiguration zeigt.

```
# User configurable Oracle clone db host specific parameters
```

```
# Source database to clone from
source_db_sid: NTAP1
source_db_host: ora_03.cie.netapp.com
```

```
# Clone database
clone_db_sid: NTAP1DEV
```

```
snapctr_obj_id: '{{ source_db_host }}\{{ source_db_sid }}'
```

Zusätzliche Clone-Ziel-Oracle-Serverkonfiguration

Der Oracle-Zielserver für Clones sollte denselben Oracle-Softwarestack aufweisen wie der Oracle-Quellserver, der installiert und gepatcht ist. Oracle-Benutzer .bash_profile hat ORACLE_BASE in Höhe von USD und ORACLE_HOME in Höhe von USD konfiguriert. Außerdem sollte die Variable „ORACLE_HOME“ mit der Oracle-Quellservereinstellung übereinstimmen. Hier ein Beispiel.

```
# .bash_profile
```

```
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
```

```
# User specific environment and startup programs
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/19.0.0/NTAP1
```

Ausführung des Playbook

Es gibt insgesamt drei Playbooks zur Ausführung des Lebenszyklus von Oracle Datenbankklonen mit SnapCenter CLI-Dienstprogrammen.

1. Einmalige Installation von Ansible-Controller-Voraussetzungen

```
ansible-playbook -i hosts ansible_requirements.yml
```

2. Spezifikationsdatei für Clone einrichten – nur einmalig.

```
ansible-playbook -i hosts clone_1_setup.yml -u admin -e
@vars/vars.yml
```

3. Erstellen und aktualisieren Sie die Klondatenbank regelmäßig von crontab mit einem Shell-Skript, um ein Aktualisierungs-Playbook aufzurufen.

```
0 */4 * * * /home/admin/na_oracle_clone_lifecycle/clone_1_refresh.sh
```

Erstellen Sie für eine zusätzliche Clone-Datenbank separate Clones_n_Setup.yml und Clone_n_refresh.yml sowie Clone_n_refresh.sh. Konfigurieren Sie die Ansible-Zielhosts und die Datei hostname.yml im Verzeichnis Host_vars entsprechend.

Wo Sie weitere Informationen finden

Weitere Informationen zur Automatisierung von NetApp Lösungen finden Sie auf der folgenden Website "Automatisierung der NetApp Lösung"

Automatisierte Oracle-Migration

NetApp Solutions Engineering Team

Zweck

Dieses Toolkit automatisiert die Migration von Oracle Datenbanken von lokalen Systemen in die AWS Cloud mit FSX ONTAP Storage und EC2 Computing-Instanz als Zielinfrastruktur. Er geht davon aus, dass der Kunde bereits über eine Oracle-Datenbank vor Ort im CDB/PDB-Modell verfügt. Mit dem Toolkit kann der Kunde eine benannte PDB aus einer Container-Datenbank auf einem Oracle-Host verschieben. Dabei wird das Verfahren für die Oracle-PDB-Verschiebung mit einer Option für maximale Verfügbarkeit verwendet. Das bedeutet, dass die Quell-PDB auf jedem lokalen Storage-Array mit minimaler Serviceunterbrechung in eine neue Container-Datenbank verlagert wird. Bei der Oracle-Verlagerung werden die Oracle-Datendateien verschoben, während die Datenbank online ist. Er leitet Benutzer-Sessions zum Zeitpunkt des Umschaltens von lokalen Systemen in die umgelagerten Datenbank-Services um, wenn alle Datendateien in die AWS-Cloud verschoben werden. Die unterstrichene Technologie ist die bewährte Oracle PDB Hot Clone-Methodik.

> Obwohl das Migrations-Toolkit für die AWS Cloud-Infrastruktur entwickelt und validiert wurde, basiert es auf Lösungen auf Applikationsebene von Oracle. Daher ist das Toolkit auch für andere Public-Cloud-Plattformen wie Azure, GCP usw. anwendbar

Diese Lösung eignet sich für folgende Anwendungsfälle:

- Erstellen Sie einen Migrationsbenutzer und gewähren Sie die erforderlichen Berechtigungen auf dem lokalen Quell-DB-Server.
- Verlagerung einer PDB von einer lokalen CDB auf eine Ziel-CDB in der Cloud, während die Quell-PDB bis zum Umschalten online ist.

Zielgruppe

Diese Lösung ist für folgende Personen gedacht:

- Ein DBA, der Oracle-Datenbanken von On-Premisses in die AWS Cloud migriert.
- Ein Database Solution Architect, der an der Migration der Oracle-Datenbank von On-Prepains zur AWS Cloud interessiert ist.
- Ein Storage-Administrator, der für das Management von AWS FSX ONTAP Storage zur Unterstützung von Oracle-Datenbanken zuständig ist
- Ein Applikationsinhaber, der Oracle Database von On-Premisses in die AWS Cloud migrieren möchte.

Lizenz

Durch den Zugriff auf, das Herunterladen, die Installation oder die Verwendung der Inhalte in diesem GitHub-Repository stimmen Sie den Bedingungen der in dargelegten Lizenz zu "Lizenzdatei".

> Es gibt bestimmte Beschränkungen bezüglich der Erstellung und/oder Freigabe von abgeleiteten Arbeiten mit dem Inhalt in diesem GitHub-Repository. Bitte lesen Sie die Lizenzbedingungen, bevor Sie den Inhalt verwenden. Wenn Sie nicht allen Bedingungen zustimmen, dürfen Sie nicht auf den Inhalt dieses Repositorys zugreifen, ihn herunterladen oder verwenden.

Lösungsimplementierung

Voraussetzungen für die Bereitstellung

Die Bereitstellung erfordert die folgenden Voraussetzungen.

```
Ansible v.2.10 and higher
ONTAP collection 21.19.1
Python 3
Python libraries:
  netapp-lib
  xmltodict
  jmespath
```

```
Source Oracle CDB with PDBs on-premises
Target Oracle CDB in AWS hosted on FSx and EC2 instance
Source and target CDB on same version and with same options installed
```

```
Network connectivity
  Ansible controller to source CDB
  Ansible controller to target CDB
  Source CDB to target CDB on Oracle listener port (typical 1521)
```

Toolkit herunterladen

```
git clone https://github.com/NetApp/na_ora_aws_migration.git
```

Konfiguration der Host-Variablen

Hostvariablen werden im Verzeichnis Host_VARs mit dem Namen {{ Host_Name }}.yml definiert. Eine Beispiel-Host-Variable Datei Host_Name.yml ist enthalten, um die typische Konfiguration zu demonstrieren. Wichtige Überlegungen:

```
Source Oracle CDB - define host specific variables for the on-prem CDB
  ansible_host: IP address of source database server host
  source_oracle_sid: source Oracle CDB instance ID
  source_pdb_name: source PDB name to migrate to cloud
  source_file_directory: file directory of source PDB data files
  target_file_directory: file directory of migrated PDB data files
```

```
Target Oracle CDB - define host specific variables for the target CDB
including some variables for on-prem CDB
  ansible_host: IP address of target database server host
  target_oracle_sid: target Oracle CDB instance ID
  target_pdb_name: target PDB name to be migrated to cloud (for max
availability option, the source and target PDB name must be the same)
  source_oracle_sid: source Oracle CDB instance ID
  source_pdb_name: source PDB name to be migrated to cloud
  source_port: source Oracle CDB listener port
  source_oracle_domain: source Oracle database domain name
  source_file_directory: file directory of source PDB data files
  target_file_directory: file directory of migrated PDB data files
```

Konfiguration der Hostdatei des DB-Servers

AWS EC2-Instanz verwenden standardmäßig die IP-Adresse für die Hostbenennung. Wenn Sie einen anderen Namen in der Hostdatei für Ansible verwenden, richten Sie die Auflösung der Hostbenennung in der Datei /etc/Hosts sowohl für den Quell- als auch für den Zielserver ein. Hier ein Beispiel.

```
127.0.0.1   localhost localhost.localdomain localhost4
localhost4.localdomain4
::1         localhost localhost.localdomain localhost6
localhost6.localdomain6
172.30.15.96 source_db_server
172.30.15.107 target_db_server
```

Ausführung des Playbooks – in der Reihenfolge ausgeführt

1. Voraussetzungen für die Installation des Ansible-Controllers

```
ansible-playbook -i hosts requirements.yml
```

```
ansible-galaxy collection install -r collections/requirements.yml
--force
```

2. Führen Sie Aufgaben vor der Migration auf dem lokalen Server aus. Dabei wird davon ausgegangen, dass Admin ssh-Benutzer für die Verbindung zum lokalen Oracle-Host mit sudo-Berechtigung ist.

```
ansible-playbook -i hosts ora_pdb_relocate.yml -u admin -k -K -t
ora_pdb_relo_onprem
```

3. Führen Sie die Oracle-PDB-Verlagerung von der lokalen CDB zur Ziel-CDB in der AWS ec2-Instanz aus, wobei ec2-User für die Verbindung mit der ec2-DB-Instanz und db1.pem mit SSH-Schlüsselpaaren für ec2-Benutzer vorausgesetzt werden.

```
ansible-playbook -i hosts ora_pdb_relocate.yml -u ec2-user --private
-key db1.pem -t ora_pdb_relo_primary
```

Wo Sie weitere Informationen finden

Weitere Informationen zur Automatisierung von NetApp Lösungen finden Sie auf der folgenden Website "Automatisierung der NetApp Lösung"

Automatisierte Oracle HA/DR in AWS FSX ONTAP

NetApp Solutions Engineering Team

Zweck

Dieses Toolkit automatisiert die Aufgaben beim Einrichten und Managen einer Umgebung für Hochverfügbarkeit und Disaster Recovery (HR/DR) für Oracle Database, die in der AWS Cloud mit FSX für ONTAP Storage- und EC2 Computing-Instanzen implementiert wird.

Diese Lösung eignet sich für folgende Anwendungsfälle:

- Richten Sie HA/DR-Ziel-Host ein – Kernel-Konfiguration, Oracle-Konfiguration, die mit dem Quell-Server-Host übereinstimmt.
- Einrichtung von FSX ONTAP – Einrichtung von Cluster-Peering, vServer-Peering, Einrichtung der Oracle Volumes snapmirror Beziehung von der Quelle zum Ziel.
- Sichern Sie Oracle-Datenbankdaten per Snapshot - Ausführen von crontab

- Backup Oracle Datenbank Archiv Protokoll per Snapshot - Ausführen von crontab
- Failover und Recovery auf HA/DR-Host – testen und validieren der HA/DR-Umgebung
- Resynchronisierung nach Failover-Test durchführen - Datenbank-Volumes snapmirror-Beziehung wieder im HA-/DR-Modus einrichten

Zielgruppe

Diese Lösung ist für folgende Personen gedacht:

- Ein DBA, der Oracle-Datenbanken in AWS für Hochverfügbarkeit, Datensicherung und Disaster Recovery eingerichtet hat.
- Ein Datenbanklösungsarchitekt, der an einer Oracle HA/DR-Lösung auf Storage-Ebene in der AWS-Cloud interessiert ist.
- Ein Storage-Administrator, der für das Management von AWS FSX ONTAP Storage zur Unterstützung von Oracle-Datenbanken zuständig ist
- Ein Applikationseigentümer, der Oracle Database für HA/DR in einer AWS FSX/EC2-Umgebung einrichten möchte.

Lizenz

Durch den Zugriff auf, das Herunterladen, die Installation oder die Verwendung der Inhalte in diesem GitHub-Repository stimmen Sie den Bedingungen der in dargelegten Lizenz zu "Lizenzdatei".

> Es gibt bestimmte Beschränkungen bezüglich der Erstellung und/oder Freigabe von abgeleiteten Arbeiten mit dem Inhalt in diesem GitHub-Repository. Bitte lesen Sie die Lizenzbedingungen, bevor Sie den Inhalt verwenden. Wenn Sie nicht allen Bedingungen zustimmen, dürfen Sie nicht auf den Inhalt dieses Repositorys zugreifen, ihn herunterladen oder verwenden.

Lösungsimplementierung

Voraussetzungen für die Bereitstellung

Die Bereitstellung erfordert die folgenden Voraussetzungen.

```
Ansible v.2.10 and higher
ONTAP collection 21.19.1
Python 3
Python libraries:
  netapp-lib
  xmltodict
  jmespath
```

```
AWS FSx storage as is available
```

```
AWS EC2 Instance
  RHEL 7/8, Oracle Linux 7/8
  Network interfaces for NFS, public (internet) and optional management
  Existing Oracle environment on source, and the equivalent Linux
operating system at the target
```

Toolkit herunterladen

```
git clone https://github.com/NetApp/na_ora_hadr_failover_resync.git
```

Konfiguration globaler Variablen

Die Ansible-Playbooks sind variablengetrieben. Eine globale Beispieldatei fsx_VARs_example.yml ist enthalten, um eine typische Konfiguration zu demonstrieren. Wichtige Überlegungen:

```
ONTAP - retrieve FSx storage parameters using AWS FSx console for both
source and target FSx clusters.
  cluster name: source/destination
  cluster management IP: source/destination
  inter-cluster IP: source/destination
  vserver name: source/destination
  vserver management IP: source/destination
  NFS lifs: source/destination
  cluster credentials: fsxadmin and vsadmin pwd to be updated in
roles/ontap_setup/defaults/main.yml file
```

```
Oracle database volumes - they should have been created from AWS FSx
console, volume naming should follow strictly with following standard:
  Oracle binary: {{ host_name }}_bin, generally one lun/volume
  Oracle data: {{ host_name }}_data, can be multiple luns/volume, add
additional line for each additional lun/volume in variable such as {{
host_name }}_data_01, {{ host_name }}_data_02 ...
  Oracle log: {{ host_name }}_log, can be multiple luns/volume, add
additional line for each additional lun/volume in variable such as {{
host_name }}_log_01, {{ host_name }}_log_02 ...
  host_name: as defined in hosts file in root directory, the code is
written to be specifically matched up with host name defined in host
file.
```

```
Linux and DB specific global variables - keep it as is.
  Enter redhat subscription if you have one, otherwise leave it black.
```

Konfiguration der Host-Variablen

Hostvariablen werden im Verzeichnis Host_VARs mit dem Namen {{ Host_Name }}.yml definiert. Eine Beispiel-Host-Variable Datei Host_Name.yml ist enthalten, um die typische Konfiguration zu demonstrieren. Wichtige Überlegungen:

```
Oracle - define host specific variables when deploying Oracle in
multiple hosts concurrently
  ansible_host: IP address of database server host
  log_archive_mode: enable archive log archiving (true) or not (false)
  oracle_sid: Oracle instance identifier
  pdb: Oracle in a container configuration, name pdb_name string and
number of pdbs (Oracle allows 3 pdbs free of multitenant license fee)
  listener_port: Oracle listener port, default 1521
  memory_limit: set Oracle SGA size, normally up to 75% RAM
  host_datastores_nfs: combining of all Oracle volumes (binary, data,
and log) as defined in global vars file. If multi luns/volumes, keep
exactly the same number of luns/volumes in host_var file
```

```
Linux - define host specific variables at Linux level
  hugepages_nr: set hugepage for large DB with large SGA for
performance
  swap_blocks: add swap space to EC2 instance. If swap exist, it will
be ignored.
```

Konfiguration der Hostdatei des DB-Servers

AWS EC2-Instanz verwenden standardmäßig die IP-Adresse für die Hostbenennung. Wenn Sie einen anderen Namen in der Hostdatei für Ansible verwenden, richten Sie die Auflösung der Hostbenennung in der Datei /etc/Hosts sowohl für Quell- als auch für Zielserver ein. Hier ein Beispiel.

```
127.0.0.1   localhost localhost.localdomain localhost4
localhost4.localdomain4
::1         localhost localhost.localdomain localhost6
localhost6.localdomain6
172.30.15.96 db1
172.30.15.107 db2
```

Ausführung des Playbooks – in der Reihenfolge ausgeführt

1. Controller-Prerequsites Ansible installieren

```
ansible-playbook -i hosts requirements.yml
```

```
ansible-galaxy collection install -r collections/requirements.yml
--force
```

2. Einrichtung der Ziel-EC2 DB-Instanz.

```
ansible-playbook -i hosts ora_dr_setup.yml -u ec2-user --private-key
db2.pem -e @vars/fsx_vars.yml
```

3. FSX ONTAP-snapmirror-Beziehung zwischen Quell- und Ziel-Datenbank-Volumes einrichten

```
ansible-playbook -i hosts ontap_setup.yml -u ec2-user --private-key
db2.pem -e @vars/fsx_vars.yml
```

4. Sichern Sie die Datenvolumes der Oracle-Datenbank per Snapshot von crontab.

```
10 * * * * cd /home/admin/na_ora_hadr_failover_resync &&
/usr/bin/ansible-playbook -i hosts ora_replication_cg.yml -u ec2-
user --private-key db1.pem -e @vars/fsx_vars.yml >>
logs/snap_data_`date +"%Y-%m%d-%H%M%S"`.log 2>&1
```

5. Backup von Protokollvolumes der Oracle-Datenbank per Snapshot von crontab.

```
0,20,30,40,50 * * * * cd /home/admin/na_ora_hadr_failover_resync &&
/usr/bin/ansible-playbook -i hosts ora_replication_logs.yml -u ec2-
user --private-key db1.pem -e @vars/fsx_vars.yml >>
logs/snap_log_`date +"%Y-%m%d-%H%M%S"`.log 2>&1
```

6. Failover und Recovery von Oracle Datenbanken auf EC2 Ziel-DB-Instanz – HA/DR-Konfiguration testen und validieren

```
ansible-playbook -i hosts ora_recovery.yml -u ec2-user --private-key
db2.pem -e @vars/fsx_vars.yml
```

7. Resynchronisierung nach Failover-Test durchführen - Datenbank-Volumes snapmirror-Beziehung im Replizierungsmodus wiederherstellen.

```
ansible-playbook -i hosts ontap_ora_resync.yml -u ec2-user --private
-key db2.pem -e @vars/fsx_vars.yml
```

Wo Sie weitere Informationen finden

Weitere Informationen zur Automatisierung von NetApp Lösungen finden Sie auf der folgenden Website "Automatisierung der NetApp Lösung"

Bereitstellung von AWS FSX ONTAP-Clustern und EC2-Instanzen

NetApp Solutions Engineering Team

Zweck

Dieses Toolkit automatisiert die Aufgaben der Bereitstellung eines AWS FSX ONTAP Storage-Clusters und einer EC2 Computing-Instanz, die anschließend für die Datenbankimplementierung verwendet werden können.

Diese Lösung eignet sich für folgende Anwendungsfälle:

- Stellen Sie eine EC2 Computing-Instanz in der AWS Cloud in einem vordefinierten VPC-Subnetz bereit und legen Sie ssh-Schlüssel für den EC2-Instanzzugriff als ec2-User fest.
- Stellen Sie ein AWS FSX ONTAP Storage-Cluster in gewünschten Verfügbarkeitszonen bereit, konfigurieren Sie eine Storage-SVM und legen Sie das Cluster-Admin-Benutzerpasswort fsxadmin fest.

Zielgruppe

Diese Lösung ist für folgende Personen gedacht:

- Ein DBA, der Datenbanken in der AWS EC2 Umgebung verwaltet.
- Ein Database Solution Architect, der an einer Datenbankimplementierung im AWS EC2 Ecosystem interessiert ist.
- Ein Storage-Administrator, der für das Management von AWS FSX ONTAP Storage, der Datenbanken unterstützt, zuständig ist
- Ein Applikationseigentümer, der eine Datenbank in AWS EC2 Ecosystem aufbauen möchte.

Lizenz

Durch den Zugriff auf, das Herunterladen, die Installation oder die Verwendung der Inhalte in diesem GitHub-Repository stimmen Sie den Bedingungen der in dargelegten Lizenz zu "Lizenzdatei".

> Es gibt bestimmte Beschränkungen bezüglich der Erstellung und/oder Freigabe von abgeleiteten Arbeiten mit dem Inhalt in diesem GitHub-Repository. Bitte lesen Sie die Lizenzbedingungen, bevor Sie den Inhalt verwenden. Wenn Sie nicht allen Bedingungen zustimmen, dürfen Sie nicht auf den Inhalt dieses Repositorys zugreifen, ihn herunterladen oder verwenden.

Lösungsimplementierung

Voraussetzungen für die Bereitstellung

Die Bereitstellung erfordert die folgenden Voraussetzungen.

```
An Organization and AWS account has been setup in AWS public cloud
  An user to run the deployment has been created
  IAM roles has been configured
  IAM roles granted to user to permit provisioning the resources
```

```
VPC and security configuration
  A VPC has been created to host the resources to be provisioned
  A security group has been configured for the VPC
  A ssh key pair has been created for EC2 instance access
```

```
Network configuration
  Subnets has been created for VPC with network segments assigned
  Route tables and network ACL configured
  NAT gateways or internet gateways configured for internet access
```

Toolkit herunterladen

```
git clone https://github.com/NetApp/na_aws_fsx_ec2_deploy.git
```

Konnektivität und Authentifizierung

Das Toolkit soll von einer AWS Cloud-Shell ausgeführt werden. AWS Cloud Shell ist eine browserbasierte Shell, die es Ihnen leicht macht, Ihre AWS-Ressourcen sicher zu managen, zu erkunden und mit ihnen zu interagieren. CloudShell ist mit Ihren Konsolenanmeldeinformationen vorauthentifiziert. Allgemeine Entwicklungs- und Betriebstools sind vorinstalliert, sodass keine lokale Installation oder Konfiguration erforderlich ist.

Konfiguration der Dateien Terraform Provider.tf und main.tf

Der Provider.tf definiert den Provider, von dem Terraform Ressourcen über API-Aufrufe bereitstellt. Die main.tf definiert die Ressourcen und Attribute der Ressourcen, die bereitgestellt werden sollen. Im Folgenden finden Sie einige Details:

```
provider.tf:
  terraform {
    required_providers {
      aws = {
        source  = "hashicorp/aws"
        version = "~> 4.54.0"
      }
    }
  }
```

```
main.tf:
  resource "aws_instance" "ora_01" {
    ami                           = var.ami
    instance_type                 = var.instance_type
    subnet_id                     = var.subnet_id
    key_name                      = var.ssh_key_name
    root_block_device {
      volume_type                 = "gp3"
      volume_size                 = var.root_volume_size
    }
    tags = {
      Name                        = var.ec2_tag
    }
  }
  ....
```

Terraform Variables.tf und terraform.tfvars Konfiguration

Die Variablen.tf deklariert die Variablen, die in main.tf verwendet werden sollen. Die terraform.tfvars enthält die tatsächlichen Werte für die Variablen. Im Folgenden einige Beispiele:

```
variables.tf:
  ### EC2 instance variables ###
```

```
variable "ami" {
  type        = string
  description = "EC2 AMI image to be deployed"
}
```

```
variable "instance_type" {
  type        = string
  description = "EC2 instance type"
}
```

```

```

terraform.tfvars:
EC2 instance variables

```

```

ami = "ami-06640050dc3f556bb" //RedHat 8.6 AMI
instance_type = "t2.micro"
ec2_tag = "ora_01"
subnet_id = "subnet-04f5fe7073ff514fb"
ssh_key_name = "sufi_new"
root_volume_size = 30

```

```

Schritt-für-Schritt-Verfahren - nacheinander ausgeführt

1. Terraform in der AWS-Cloud-Shell installieren.

```
git clone https://github.com/tfutils/tfenv.git ~/.tfenv
```

```
mkdir ~/bin
```

```
ln -s ~/.tfenv/bin/* ~/bin/
```

```
tfenv install
```

```
tfenv use 1.3.9
```

2. Laden Sie das Toolkit von der öffentlichen NetApp GitHub Website herunter

```
git clone https://github.com/NetApp-
Automation/na_aws_fsx_ec2_deploy.git
```

3. Führen Sie init aus, um Terraform zu initialisieren

```
terraform init
```

4. Testsuite ausgeben

```
terraform plan -out=main.plan
```

5. Anwenden der Testsuite

```
terraform apply "main.plan"
```

6. Führen Sie „Destroy“ aus, um die Ressourcen nach Abschluss zu entfernen

```
terraform destroy
```

Wo Sie weitere Informationen finden

Weitere Informationen zur Automatisierung von NetApp Lösungen finden Sie auf der folgenden Website "Automatisierung der NetApp Lösung"