DB Automation Toolkits

NetApp Solutions

NetApp
October 20, 2023

Table of Contents

DB Automation Toolkits

Automated Oracle Migration

NetApp Solutions Engineering Team

Purpose

This toolkit automates Oracle database migration from on-premises to AWS cloud with FSx ONTAP storage and EC2 compute instance as target infrastructure. It assumes the customer already has an on-premises Oracle database deployed in the CDB/PDB model. The toolkit will allow the customer to relocate a named PDB from a container database on an Oracle host using the Oracle PDB relocation procedure with a maximum availability option. That means the source PDB on any on-premises storage array relocates to a new container database with minimal service interruption. The Oracle relocation procedure will move the Oracle data files while database is online. It subsequently reroutes user sessions from on-premises to the relocated database services at the time of switching over when all data files move over to AWS cloud. The underlined technology is proven Oracle PDB hot clone methodology.

This solution addresses the following use cases:

- Create migration user and grant required privileges at on-prem source DB server.
- Relocate a PDB from on-premises CDB to a target CDB in cloud while the source PDB is online until switch over.

Audience

This solution is intended for the following people:

- A DBA who migrates Oracle databases from on-premisses to AWS cloud.
- A database solution architect who is interested in Oracle database migration from on-premisses to AWS cloud.
- A storage administrator who manages AWS FSx ONTAP storage that supports Oracle databases.
- An application owner who likes to migrate Oracle database from on-premisses to AWS cloud.

License

By accessing, downloading, installing or using the content in this GitHub repository, you agree the terms of the License laid out in License file.

There are certain restrictions around producing and/or sharing any derivative works with the content in this GitHub repository. Please make sure you read the terms of the License before using the content. If you do not agree to all of the terms, do not access, download or use the content in this repository.

Solution deployment

Prerequisites for deployment

Deployment requires the following prerequisites.

```
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)
```

Download the toolkit

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

Host variables configuration

Host variables are defined in host_vars directory named as {{ host_name }}.yml. An example host variable file host_name.yml is included to demonstrate typical configuration. Following are key considerations:

```
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
```

DB server host file configuration

AWS EC2 instance use IP address for host naming by default. If you use different name in hosts file for Ansible, setup host naming resolution in /etc/hosts file for both source and target server. Following is an example.

```
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
```

Playbook execution - executed in sequence

1. Install Ansible controller prerequisites.

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

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

2. Execute pre-migration tasks against on-prem server - assuming admin is ssh user for connection to on-prem Oracle host with sudo permission.

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

3. Execute Oracle PDB relocation from on-prem CDB to target CDB in AWS EC2 instance - assuming ec2-user for EC2 DB instance connection, and db1.pem with ec2-user ssh key pairs.

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

Where to find additional information

To learn more about the NetApp solution automation, review the following website NetApp Solution Automation

Automated Oracle HA/DR in AWS FSx ONTAP

NetApp Solutions Engineering Team

Purpose

This toolkit automates the tasks of setting up and managing a High Availability and Disaster Recovery (HR/DR) environment for Oracle database deployed in AWS cloud with FSx for ONTAP storage and EC2 compute instances.

This solution addresses the following use cases:

- Setup HA/DR target host - kernel configuration, Oracle configuration to match up with source server host.
- Setup FSx ONTAP - cluster peering, vserver peering, Oracle volumes snapmirror relationship setup from source to target.
- Backup Oracle database data via snapshot - execute off crontab
- Backup Oracle database archive log via snapshot - execute off crontab
- Run failover and recovery on HA/DR host - test and validate HA/DR environment

- Run resync after failover test - re-establish database volumes snapmirror relationship in HA/DR mode

Audience

This solution is intended for the following people:

- A DBA who set up Oracle database in AWS for high availability, data protection, and disaster recovery.
- A database solution architect who is interested in storage level Oracle HA/DR solution in the AWS cloud.
- A storage administrator who manages AWS FSx ONTAP storage that supports Oracle databases.
- An application owner who like to stand up Oracle database for HA/DR in AWS FSx/EC2 environment.

License

By accessing, downloading, installing or using the content in this GitHub repository, you agree the terms of the License laid out in License file.

There are certain restrictions around producing and/or sharing any derivative works with the content in this GitHub repository. Please make sure you read the terms of the License before using the content. If you do not agree to all of the terms, do not access, download or use the content in this repository.

Solution deployment

Prerequisites for deployment

Deployment requires the following prerequisites.

```
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
```

Download the toolkit

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

Global variables configuration

The Ansible playbooks are variable driven. An example global variable file fsx_vars_example.yml is included to demonstrate typical configuration. Following are key considerations:

```
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.
```

Host variables configuration

Host variables are defined in host_vars directory named as {{ host_name }}.yml. An example host variable file host_name.yml is included to demonstrate typical configuration. Following are key considerations:

```
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.
```

DB server host file configuration

AWS EC2 instance use IP address for host naming by default. If you use different name in hosts file for Ansible, setup host naming resolution in /etc/hosts file for both source and target servers. Following is an example.

```
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
```

Playbook execution - executed in sequence

1. Install Ansible controller prerequsites.

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

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

2. Setup target EC2 DB instance.

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

3. Setup FSx ONTAP snapmirror relationship between source and target database volumes.

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

4. Backup Oracle database data volumes via snapshot from 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 Oracle database archive log volumes via snapshot from 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. Run failover and recover Oracle database on target EC2 DB instance - test and validate HA/DR configuration.

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

7. Run resync after failover test - re-establish database volumes snapmirror relationship in replication mode.

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

Where to find additional information

To learn more about the NetApp solution automation, review the following website NetApp Solution Automation

AWS FSx ONTAP Cluster and EC2 Instance Provision

NetApp Solutions Engineering Team

Purpose

This toolkit automates the tasks of provisioning of an AWS FSx ONTAP storage cluster and an EC2 compute instance, which can be subsequently used for database deployment.

This solution addresses the following use cases:

- Provision an EC2 compute instance in AWS cloud in a predefined VPC subnet and set ssh key for EC2 instance access as ec2-user.
- Provision an AWS FSx ONTAP storage cluster in desired availability zones and configure a storage SVM and set cluster admin user fsxadmin password.

Audience

This solution is intended for the following people:

- A DBA who manages databases in AWS EC2 environment.
- A database solution architect who is interested in database deployment in AWS EC2 ecosystem.
- A storage administrator who manages AWS FSx ONTAP storage that supports databases.
- An application owner who likes to standup database in AWS EC2 ecosystem.

License

By accessing, downloading, installing or using the content in this GitHub repository, you agree the terms of the License laid out in License file.

There are certain restrictions around producing and/or sharing any derivative works with the content in this GitHub repository. Please make sure you read the terms of the License before using the content. If you do not agree to all of the terms, do not access, download or use the content in this repository.

Solution deployment

Prerequisites for deployment

Deployment requires the following prerequisites.

```
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
```

Download the toolkit

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

Connectivity and authentication

The toolkit is supposed to be executed from an AWS cloud shell. AWS cloud shell is a browser-based shell that makes it easy to securely manage, explore, and interact with your AWS resources. CloudShell is pre-authenticated with your console credentials. Common development and operations tools are pre-installed, so no local installation or configuration is required.

Terraform provider.tf and main.tf files configuration

The provider.tf defines the provider that Terraform is provisioning resources from via API calls. The main.tf defines the resources and attributes of resources that are to be provisioned. Following are some 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 and terraform.tfvars configuration

The variables.tf declares the variables to be used in main.tf. The terraform.tfvars contains the actual values for the variables. Following are some examples:

```
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
....
```

Step by step procedures - executed in sequence

1. Install Terraform in AWS cloud shell.

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

```
mkdir ~/bin
```

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

```
tfenv install
```

```
tfenv use 1.3.9
```

2. Download the toolkit from NetApp GitHub public site

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

3. Run init to initialize terraform

```
terraform init
```

4. Output the execution plan

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

5. Apply the execution plan

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

6. Run destroy to remove the resources when done

```
terraform destroy
```

Where to find additional information

To learn more about the NetApp solution automation, review the following website NetApp Solution Automation

Copyright information

Trademark information