Ansible Modules for Dell EMC PowerMax 1.0 – Best Practices Guide

Abstract

This document provides guidelines and tips for making use of the Ansible Modules for Dell EMC PowerMax. The information is relevant for the 1.0 release of the Ansible Modules. Use it along with the corresponding Product Guide and Release Notes.

August 2019

Revisions

Date	Revision	Description
18 July 2019	0.1	Initial version – for internal review
30 August 2019	1.0	Incorporated review changes

Acknowledgements

This paper was produced by the following:

Author: Anil Degwekar

Review comments by: Parasar Koditi, Bill Elliott, Paul Martin

Other:

Table of contents

Executive summary

Ansible is a popular open source software solution for configuration management and automation of large data center deployments. Ansible Modules for Dell EMC PowerMax are designed to help customers automate storage management operations on PowerMax and VMAX All Flash arrays.

This best practices guide provides guidelines on how to make best use of these modules for your automation needs. The information provided here can be considered as additional to the product guide and release notes for the product.

1 Introduction

The Ansible Modules for Dell EMC PowerMax provide the functions which can be used by Storage administrators to automate storage management tasks. Functionality provided includes:

- Listing arrays under management of Unisphere
- Listing array entities like volumes, storage groups, port groups, hosts, host groups and masking views
- Creating volumes in storage groups, and managing their lifecycle
- Data protection of volumes using snapshots
- Creating port groups (FC and iSCSI) using ports present on the array
- Registering hosts on the array
- Creating host groups
- Creating masking views to present storage to hosts

This Best Practices Guide can be considered as a companion document to the product documentation. Product documentation includes embedded examples within the modules, a Product Guide and Release Notes. While most administrators familiar with Ansible will be able to use the modules based on this documentation, some extra tips and guidelines which may be useful are presented in this document.

1.1 Objective

This document should help users to quickly write some simple playbooks to get started, and to get the best results out of the Ansible modules for PowerMax. To get detailed information about each possible option available with the modules, please refer to the embedded documentation within the modules, and to the Product Guide.

The objective is not to educate users about Ansible or about PowerMax. To learn more about Ansible, refer to the Ansible documentation. The learn more about PowerMax or Unisphere, refer to the Dell EMC documentation site for PowerMax (see references in Appendix C).

1.2 Intended Audience

The intended audience for this guide are the storage administrators who want to use Ansible to automate their provisioning tasks on PowerMax arrays. It is assumed that the users of these modules are familiar with PowerMax architecture (including concepts like storage groups, port groups, hosts, host groups and masking views), and are familiar with basic operations using Unisphere for PowerMax. Some familiarity with the python language will be useful, though not required.

1.3 Product Overview

There are nine Ansible Modules for PowerMax. Each module is aimed at one particular object type on the array. Together, these modules provide all the operations required for basic provisioning of volumes and storage groups, and for presenting storage to hosts or clusters.

1. The volume module allows creation of volumes and placing them into storage groups. The complete lifecycle of volumes can be managed – including modifying the size, renaming and deletion.
2. The Storage Group module allows creating of storage groups; and placing the storage groups inside masking views.
3. The Host module allows registering hosts on the array; and managing the initiators on the host. Both FC and iSCSI initiators are supported.

4. The Host Group module allows grouping multiple hosts into a group; and adding or removing hosts from the host group.
5. The Port module allows getting details of ports present on the array. Both FC and iSCSI ports can be queried.
6. The Port Group module allows grouping multiple ports into a group; and adding or removing ports from the port group.
7. The Masking View module brings all these entities together. A masking view consists of a storage group, a port group and a host or a host group. When a masking view is created, the hosts in the host group get access to the volumes in the storage group via the ports in the port group.
8. The Snapshot module allows creating snapshots of storage groups; and managing their lifecycle – including linking the snapshots to target storage groups, and managing the link state.
9. The Gather Facts module allows getting a list of volumes, storage groups, ports, port groups, hosts, host groups, masking views and storage resource pools on the array. It also has a mode in which the arrays under management of the Unisphere for PowerMax can be obtained.

1.4 Limitations

Some of the limitations of the Ansible Modules are as follows:

- There is no support for remote replication (SRDF).
- There is no support for zone management.
- There is no support for relink operations in snapshot management.

Some more details of module specific limitations are covered in the following chapters.

2 Installation

Your installation should follow the sequence as given below.

2.1 PowerMax / VMAX AFA setup

The Ansible Modules support PowerMax OS version 5978 – available on PowerMax or VMAX All Flash arrays.

2.2 Unisphere for PowerMax

The Ansible modules support version 9.0 of Unisphere for PowerMax, either embedded running on the array or installed standalone. A single Unisphere server can manage multiple arrays. The connection parameters within the Ansible playbooks can specify the array serial number to operate upon. Access controls can be configured via Unisphere restricting users to certain arrays, storage groups and functions.

2.3 Setting up the Linux host

Verify that Unisphere is accessible from the Linux server which you plan to use as your Ansible host. Refer to the Product Guide [1] on specific versions of supported operating systems.

2.4 Ansible setup

Refer to Ansible documentation [4] on how to setup Ansible on your Linux server.

2.5 Deploying the Python library for Unisphere

The python library for Unisphere is available for download from pypi.org. This library is required for the operation of Ansible Modules for PowerMax.

The Ansible modules support version 3.0.0.14 of the PyU4V library. Refer to the product guide on specific instructions on how to download and install the library on your Linux host.

2.6 Deploying the Ansible Modules for PowerMax

Refer to the Product Guide [1] on specific instructions on how to install the Ansible modules for PowerMax. These need to be deployed on the same Linux host where Ansible and the python library is installed.

2.7 Securing array credentials

The connection parameters to Unisphere include

1. **unispherehost**: This is the IP address or hostname used to connect to Unisphere
2. The version number of Unisphere (**universion**): This number is without the decimal point – for Unisphere 9.0, specify the version number as ‘**90**’.
3. The ‘**verifycert**’ parameter – should be **True** for production deployments. This tells the library whether to verify the HTTPS certificates. Refer to Unisphere documentation on how to import the certificates on the client operating system.

4. The **username** and **password**: Running any task requires authentication with the Unisphere for PowerMax server with a valid username and password. It is recommended that these be encrypted using Ansible vault. Refer to Ansible documentation [4] on how to use Ansible vault. It is out of scope of this document to provide specific instructions on how to use Ansible commands – the reader is referred to Ansible documentation for the same.

For best results, create a separate user for Ansible on Unisphere, and give that user the required permissions on the array that you want to provision using Ansible. Please note that the array credentials should be handled with care and should not be revealed to all users. Any user who can login to the Linux host will be able to provision storage using the Ansible modules, if that user has the array credentials.

Users require admin privileges to run playbooks that execute provisioning tasks. Users require replication privileges to operate snapshot operations. Authentication can be integrated with LDAP and Role Based Access via the Unisphere for PowerMax configuration on a per array basis. See the blog post on role based access controls in Unisphere [7] for more details.

3 General guidelines and hints

Some general guidelines for using the Ansible modules are given below.

3.1 Ansible is end-state based

One important thing to note about Ansible is that it is end-state based. User specifies the desired end state of an object (say, a volume) – either ‘present’ or ‘absent’. The module execution checks the state on the array (PowerMax) to see whether the object state is as per the request. If the state is already as per the specification, the module will not execute any other commands. Else, it will execute the module calls to bring the actual state to the desired state.

The following table shows the possible cases for a volume.

Table 1 Volume module actions

Desired state as per playbook	State on the array	Action taken by the module

`sg_name: "SG1"`
`vol_name: "Vol1"`
`size: 100`
`cap_unit: "GB"`
`state: 'present'`
	Volume is not found on array.	Module will create a new volume of the given name and size in the given storage group.
		
		(Action = Create volume; changed = true)

	Volume is found on array, and its size is 50 GB.	Module will modify the size and bring it to 100 GB as per the desired state in the playbook. (Action = Modify volume; changed = true)
	Volume is found on the array, and its size is 100 GB.	In the case, since the volume size is already as per the desired state, the module will return success but will not take any action. (Action = Get volume details; changed = false)
	Volume is found on the array, and its size is 200 GB.	In the case, since the volume size cannot be reduced, the module will return an error.

`vol_id: "XXXX"`
`state: 'absent'`
	Volume is found on array	The module will delete the volume, provided all other conditions are met – e.g. volume is not mapped to any host, and volume is not part of any storage group. (Note that these validations are done by the array, and not by the Ansible module)
		
		(Action = Delete volume; changed = true)

	Volume is not found on array	In the case, since the volume is already deleted, the module will not take any action. (No action; changed = false)

The same logic is followed in all other modules.

3.2 Object state and the state of children within a group

This section covers the difference between object state (covered by its ‘state’ variable) and the state of objects within a group.

Every object – whether it’s a stand-alone object (e.g. a volume, or a snapshot) or a group object (e.g. a storage group, a port group or a host group), has a ‘state’ variable. The ‘state’ variable can take only two values – ‘present’ or ‘absent’. The state variable is always a mandatory parameter to every task. As Ansible is end-state based, usually the state variable will have the value ‘present’. Use the value ‘absent’ only when you want to delete something.

There are three group objects:

1. A storage group (which is a group of volumes)
2. A host group (which is a group of hosts)
3. A port group (which is a group of ports)

All these group objects take in child objects, and the state of the children is governed by separate state variables as given in the table below.

In addition, a host has child objects in the form of initiators, and so it has a state variable to control the state of initiators within it.

A storage group can also take other storage groups as its children.

Table 2 Group objects and its children

Group object	Child	Possible states of the child

Storage group	Volume	`vol_state: "present-in-sg"`
		`vol_state: "absent-in-sg"`

	Storage group	`sg_state: "present-in-sg"` `sg_state: "present-in-sg"`

Port Group	Port	`port_state: "present-in-group"`
		`port_state: "absent-in-group"`

Host Group	Host	`host_state: "present-in-group"`
		`host_state: "absent-in-group"`

Host	Initiator	`initiator_state: "present-in-host"`
		`initiator_state: "absent-in-host"`

For each group object, one can add/remove the underlying objects – e.g. you can add volumes to a storage group, hosts to a host group, or ports to a port group. Similarly, you can add initiators to a host.

In a cascaded storage group, a child storage group can be added to a parent storage group. Volumes can be added to a child storage group, but not to the parent storage group.

A host group is essentially a cascaded initiator group on PowerMax. Whereas a host is an initiator group on PowerMax. Initiators can be added to a host, but not to a host group.

3.3 Idempotency in Ansible

Idempotency is an important requirement for all Ansible modules; and is handled for both create as well as delete operations. Because of this, almost all tasks can be executed again without causing any side effect. The only exception to this is the snapshot module – where repeated execution of the ‘create snapshot’ task will create further generations of the same snapshot. More details can be seen in the next section, where each module is covered in greater detail.

3.4 How Ansible Modules work with PowerMax

3.4.1 Volumes

Volumes on PowerMax have a unique identifier (ID) as well as a user-friendly label (or name). The volume ID is guaranteed to be unique on the array, whereas the name need not be unique.

The Ansible modules focus on the volume name rather than the ID to make them user-friendly. This is because the playbooks may be written once but executed multiple times.

3.4.2 Storage groups

Storage groups are a collection of devices stored on the array that are used by an application, a server, or a collection of servers. The storage group is the key management entity in PowerMax. Storage group is the unit of provisioning, and all the data services like snapshots and management tasks are at the storage group level. Most of the operations in the Ansible modules for PowerMax are storage group-centric. Snapshots operate on storage groups. Masking views contain storage groups. Service level is set at a storage group level which will control the response time expected for the application. A volume can be part of multiple storage groups.

A storage group can contain other storage groups in a parent child relationship. A parent storage group can have multiple child storage groups (but not volumes). A child storage group can have volumes (but not child storage groups). Service level should be set on the child group, but not on the parent group.

A complete description of PowerMax storage groups is out of scope of this document. Please refer to the array documentation to get more details.

The Ansible modules for PowerMax identify a volume based on two characteristics – its name and the name of its storage group. We do not support multiple volumes with the same name in a storage group. However, volumes from different storage groups can have the same name.

3.4.3 Hosts and Host groups

On PowerMax, there are two types of container for host initiators:

1. Host – Usually represents a single host and its initiators.
2. Host Group– Container for multiple hosts, usually represents a cluster.

Hosts and Host groups in PowerMax can be either of type FC or iSCSI. Mixed initiator groups are not allowed.

3.4.4 Ports and Port Groups

A Port group is a collection of related ports through which devices are accessed by a host usually via FC fabric or IP Network. All ports in a port group can be either of type FC or iSCSI – mixed types are not allowed inside a port group.

3.4.5 Masking Views

A masking view consists of

- A storage group,
- A port group, and
- A host or a Host group

At the time of creating a masking view, all three parameters must be specified.

A masking view once created cannot be modified – it has to be destroyed and re-created.

While creating a masking view, ensure that the host/host group and port group are of the same type – FC or iSCSI. Mixing a port group of type FC with a host of type iSCSI is an error (and vice-a-versa).

Once a masking view is created, all the volumes in the storage group are visible to the host / host group (assuming that zones are already in place for FC initiators and ports). Any volumes that get added to the storage group after this are seen by the host automatically. If the storage group is a cascaded storage group, then volumes belonging to all its child storage groups are visible to the host / host group.

3.4.6 Snapshots

SnapVX snapshots are created and controlled via storage groups in Unisphere for PowerMax. The Ansible module for snapshots also follows this practice.

A storage group snapshot can be linked to another (target) storage group. When a snapshot is linked in this manner, the target storage group will be automatically created, if it does not exist.

PowerMax has a concept of snapshot generations. When a snapshot is created, it gets a generation number of 0. When the same snapshot is created one more time, the previously created snapshot gets its generation number incremented, and the newly created snapshot gets generation number 0. Only the 0’th snapshot generation can be deleted. When a snapshot generation is deleted, all previous snapshots with the same name get their generation number decremented.

4 Naming Policy

As already mentioned, operations via Ansible modules for PowerMax are name-centric. All entities, including volumes, storage groups, snapshots and masking view are identified by their name.

Before starting to use the Ansible modules, define a naming policy to separate your ansible managed storage objects from manually created

In most environments, storage provisioning is either host-centric or application-centric.

An example of host-centric naming convention is as follows:

Host names: kdpr001, kdpr002, pgmn001, pgmn002

Storage group names: sg-kdpr001, sg-kdpr002, sg-pgmn001, sg-pgmn002

Masking view names: mv-kdpr001, mv-kdpr002, mv-pgmn001, mv-pgmn002

Within each storage group, volume names can be appropriately assigned.

Port groups may be shared across hosts.

An example of an application-centric provisioning environment as follows:

Application names: Oracle, SAP, Exchange, MySql

Storage group names: sg-Oracle, sg-SAP, sg-Exchange, sg-MySql

Masking view names: mv-Oracle, mv-SAP, mv-Exchange, mv-MySql

Within each storage group, volume names can be appropriately assigned.

Port groups may be shared across the applications.

5 Module specific guidelines and hints

In this section we cover guidelines specific to each module. For specific details on parameters and their syntax, please refer to the embedded documentation within each module, and to the Product Guide.

5.1 Volume module

Keep the following points in mind while using the **volume** module:

- The volume module operates on only one volume at a time.
- A volume can be identified in two ways – (a) by the volume name and the name of its storage group; or (b) by specifying the volume ID (device ID on PowerMax). Most of the volume operations are based on volume name. Some operations require using the volume ID – e.g. volume delete (where state='absent'). Note that just specifying the volume name is not sufficient to identify a volume – since there can be multiple volumes with the same name.
- Delete a volume only when you are absolutely sure that you do not need the data. There are some inherent protections provided by the array – e.g. volumes which are mapped to a host, or which are part of a storage group cannot be deleted.
- The volume module does not allow multiple volumes with the same name to be created within a storage group. This is because the Ansible module identifies the volume based on the volume name and its storage group name. If there are multiple volumes with the same name in a storage group, then Ansible will not know which one to work on.
- A volume can be part of multiple storage groups – however, for Ansible use cases, a volume should be part of only one storage group.
- Volume size can be specified in MB, GB or TB. The default unit is GB.
- Volumes can be expanded simply by specifying the new size which is larger than the existing size. Shrinking volumes is not allowed.
- Operations on existing volumes (e.g. to get volume details, or to modify its size) only need to specify the volume name and the storage group name. But while creating a new volume, its size also needs to be specified.
- Performance attributes of a volume (e.g. the FAST policy, compression, etc.) are determined by the containing storage group.

5.2 Storage group module

Please note the following when using the **storagegroup** module:

- Most operations on VMAX and PowerMax are storage group-centric. Among other things:
    - Volumes are mapped to host based on which storage group they are part of.
    - Snapshots operate on the storage group.
    - Compression, service level and the SRP for a volume is determined by the storage group.
- Name the storage group based on the application that it supports, or the host/cluster that it supports – see the section 4 on Naming policy.
- A cascaded storage group strategy can be used when a single host or cluster needs volumes with different service levels. In such a case, the parent storage group can be without a service level, whereas the child storage groups can each have a different service level. Note that the parent storage group cannot have volumes in it.
- By default, the default SRP ('SRP_1') is used. This should be applicable for most deployments.

- Compression on the storage groups is enabled by default.
- If the service level is not specified, the storage group will inherit all the default policies as set on the array.
- Attributes such as the service level, compression, storage group name can be changed any time.
- When creating snapshots of a cascaded storage group, create them either at the parent level, or at the child group level. A snapshot created at the parent storage group level includes the child storage groups.

5.3 Port group module

Please note the following when using the **portgroup** module:

- Port groups can be shared across different hosts / masking views.
- One port can be part of multiple port groups.
- Mixed port groups (consisting of FC and iSCSI ports) are not allowed.
- Be aware of different formats while specifying the ports for FC and iSCSI. FC ports need a WWN (without the ':' characters), while iSCSI ports need the IQN.

5.4 Host module

Please note the following when using the **host** module:

- A host is basically an initiator group on PowerMax, one which is a stand-alone initiator group (does not have any child initiator groups).
- When creating a host, it is mandatory to specify at least one initiator. An empty host (with no initiators) cannot be created.
- While creating a host, you can specify the host flags. These are optional, and so the recommendation is to not specify anything and let the defaults get used unless you are very clear about which flags you want to specify.
- Hosts can have either FC or iSCSI initiators, but not both types of initiators. If your host has both FC and iSCSI initiators, create two separate hosts for it on the array – on with FC initiators, and another with iSCSI initiators.

5.5 Host group module

Please note the following when using the **hostgroup** module:

- A host group is also an initiator group on the array – the only difference is that this is a cascaded initiator group (one which has only child initiator groups but no initiators of its own).
- A host group can have only one type of hosts – either FC hosts or iSCSI hosts, but not both.
- Sometimes a host group is referred to as a cluster.
- The recommendation about host flags applies to the host group too – leave them to be the default unless you are very clear about which flags you want to specify.

5.6 Masking view module

Please note the following when using the **maskingview** module:

- A masking view is a critical entity, since it controls the storage exposed to a host or cluster.

- Create a host-based masking view for a stand-alone host.
- Create a host group-based masking view for a cluster.

5.7 Snapshot module

Please note the following when using the **snapshot** module:

- Pay attention to snapshot generations, as explained in the previous section (3.4.6).
- A snapshot can be the source of a replication link, in which case the snapshot data is made available at another (target) storage group. Note that this is local replication (within the array).
- Each generation can be linked differently. When a snapshot is linked in this manner, the target storage group is automatically created, if it doesn’t exist already.
- If a storage group is a target of a snapshot link, it cannot be deleted.
- Snapshots should be unlinked and terminated before deleting source storage groups.
- The link state can be changed to ‘linked’ (active) or ‘unlinked’ (inactive).
- A single snapshot may be linked to multiple target storage groups.

5.8 Gather facts module

Please note the following when using the **gatherfacts** module:

- The gatherfacts module in intended for getting details of entities on the array. In most cases, users will know the entities that they have created (storage groups, masking views, etc.,) and so they do not need to use this module.
- The most common use case is to get the list of ports and port groups, since users typically reuse existing port groups.
- Although there is a provision to get the list of volumes on the array, use this only if it is really required. In large deployments, the volume list is likely to be quite large, and it takes a long time to get the complete list.

5.9 Summary

This document provides some suggestions and best practices when using the Ansible modules for PowerMax.

In the Appendix that follows, we have provided two sample playbooks. The first sample contains a typical provisioning workflow, while the second sample contains a typical de-provisioning workflow. Users can take these samples and modify them as per their environment.

Stay tuned for some more comprehensive samples that will be provided in future.

A Sample playbooks

Note: These playbooks should be considered purely as samples. They need to be modified to suit your environment. Do not use them directly.

We provide below listing of two sample playbooks. These playbooks show the typical flow for provisioning volumes to a host or a cluster, and for deprovisioning the volumes and related resources after they are no longer needed.

The playbooks use Ansible vault to store array credentials in a file called `‘get_password.yml’`. Details on how the use Ansible vault are not provided here – users can get those from Ansible documentation. Using Ansible vault is a recommended way to store array credentials.

A.1 Provisioning Playbook: Overview

The playbook performs provisioning using a typical provisioning flow. There is a ‘pause’ at the end of every task, which allows the user time to verify every action on the array.

- The common variables are declared initially in the section called `‘vars’.`
- It first creates a storage group `'Ansible_Testing_SG'` for the host. Service level **`‘Diamond’`** is used – you may change it to your desired service level. Note that the storage group service level can be changed any time.
- It then creates a new volume `'Ansible_Test_SG_Vol'` in this storage group. Volume size is 1GB. The volume size can be increased any time.
- It registers a host `'Ansible_Testing_host'` with two FC initiators.
- It registers another host `'Ansible_Testing_host2'` with two FC initiators.
- It then creates a host group `'Ansible_Testing_hostgroup'` consisting of these two hosts.
- It creates a port group `'Ansible_Testing_portgroup'` consisting of two FC ports.
- It then creates a masking view with the storage group, host group and port group. This completes the provision task.

Note: Creating zones on the FC fabric is out of scope of this workflow. Assuming that the zones are already created, the volumes will be available at the hosts at the end of this workflow.

A.1.1 Provisioning Playbook: listing

```
---
- name: Provisioning storage for Powermax
  hosts: localhost
  connection: local
  vars:
        unispherehost: '******'
        universion: "90"
        verifycert: False
        sg_name: 'Ansible_Testing_SG'
```

```
      serial_no: '**********'
      cap_unit: 'GB'
      vol_name: 'Ansible_Test_SG_Vol'
      mv_name: 'Ansible_Testing_mv'
      portgroup_name: 'Ansible_Testing_portgroup'
      hostgroup_name: 'Ansible_Testing_hostgroup'
      host_name: 'Ansible_Testing_host'
      host_name2: 'Ansible_Testing_host2'

tasks:

- name: Get encrypted powermax credentials
  no_log: True
  include_vars:
      file: get_password.yml
- set_fact:
      user : "{{ powermax_username }}"
      password : "{{ powermax_password }}"
  no_log: True

- name: Create Storage group
  dellemc_powermax_storagegroup:
    unispherehost: "{{unispherehost}}"
    universion: "{{universion}}"
    verifycert: "{{verifycert}}"
    user: "{{user}}"
    password: "{{password}}"
    serial_no: "{{serial_no}}"
    sg_name: "{{sg_name}}"
    service_level: "Diamond"
    srp: "SRP_1"
    state: 'present'

- pause:
    prompt: "Paused! Please 'Enter' to continue"

- name: Create volume
  dellemc_powermax_volume:
    unispherehost: "{{unispherehost}}"
    universion: "{{universion}}"
    verifycert: "{{verifycert}}"
    user: "{{user}}"
    password: "{{password}}"
    serial_no: "{{serial_no}}"
    sg_name: "{{sg_name}}"
    vol_name: "{{vol_name}}"
    size: 1
    cap_unit: "{{cap_unit}}"
    state: 'present'
- pause:
    prompt: "Paused! Please 'Enter' to continue"

- name: Create host "{{host_name}}"
  dellemc_powermax_host:
    unispherehost: "{{unispherehost}}"
    universion: "{{universion}}"
    verifycert: "{{verifycert}}"
    user: "{{user}}"
    password: "{{password}}"
    serial_no: "{{serial_no}}"
    host_name: "{{host_name}}"
    initiators:
    - 10000000c98ffeae
    - 10000000c98ffebf
    state: 'present'
    initiator_state: "present-in-host"

- pause:
    prompt: "Paused! Please 'Enter' to continue"

- name: Create host "{{host_name2}}"
  dellemc_powermax_host:
    unispherehost: "{{unispherehost}}"
    universion: "{{universion}}"
    verifycert: "{{verifycert}}"
    user: "{{user}}"
    password: "{{password}}"
    serial_no: "{{serial_no}}"
    host_name: "{{host_name2}}"
    initiators:
    - 10000000c98ffeaa
    - 10000000c98ffebb
    state: 'present'
    initiator_state: "present-in-host"

- pause:
    prompt: "Paused! Please 'Enter' to continue"

- name: Create host group "{{hostgroup_name}}"
  dellemc_powermax_hostgroup:
    unispherehost: "{{unispherehost}}"
    universion: "{{universion}}"
    verifycert: "{{verifycert}}"
    user: "{{user}}"
    password: "{{password}}"
    serial_no: "{{serial_no}}"
    hostgroup_name: "{{hostgroup_name}}"
    state: "present"
    hosts:
    - "{{host_name}}"
    - "{{host_name2}}"
    host_state: 'present-in-group'

- pause:
    prompt: "Paused! Please 'Enter' to continue"

- name: Create port group "{{portgroup_name}}"
  dellemc_powermax_portgroup:
    unispherehost: "{{unispherehost}}"
    universion: "{{universion}}"
    verifycert: "{{verifycert}}"
    user: "{{user}}"
    password: "{{password}}"
    serial_no: "{{serial_no}}"
    portgroup_name: "{{portgroup_name}}"
    state: "present"
    ports:
    - director_id: "FA-1D"
      port_id: "5"
    - director_id: "FA-2D"
      port_id: "5"
    port_state: 'present-in-group'

- pause:
    prompt: "Paused! Please 'Enter' to continue"

- name: Create MV "{{mv_name}}" with existing elements
  dellemc_powermax_maskingview:
    unispherehost: "{{unispherehost}}"
    universion: "{{universion}}"
    verifycert: "{{verifycert}}"
    user: "{{user}}"
    password: "{{password}}"
    serial_no: "{{serial_no}}"
    mv_name: "{{mv_name}}"
    portgroup_name: "{{portgroup_name}}"
    hostgroup_name: "{{hostgroup_name}}"
    sg_name: "{{sg_name}}"
    state: 'present'
```

A.2 Deprovisioning Playbook: overview

The playbook performs deprovisioning of the volumes – which is a reverse of the previous operation. Note that the deprovisioning needs to happen in the reverse order. Another thing to note is that the 'state' is marked as 'absent' for deprovisioning, whereas it was 'present' for provisioning.

- The masking view is deleted first. Unless the masking view is deleted, the storage group, port group and host group within it cannot be deleted.
- The next task obtains the ID of the volume **`'Ansible_Test_SG_Vol'`**. This id will be required to delete the volume. Note that a volume cannot be deleted using its name. Also, the volume cannot be deleted if it is part of a storage group.
- After this, the storage group is deleted. This makes the volume a stand-alone volume.
- The volume is then deleted using the volume ID which was obtained in the second step.
- The host group is then deleted, followed by deletion of the two hosts.
- The port group is deleted next. This completed the deprovisioning workflow. Note that ports are always present on the array, and so cannot be deleted.

A.2.1 Deprovisioning Playbook: listing

```
---
- name: DeProvisioning Storage for Powermax
  hosts: localhost
  connection: local
  vars:
        unispherehost: '********'
        universion: "90"
        verifycert: False
        sg_name: 'Ansible_Testing_SG'
        serial_no: '********'
        cap_unit: 'GB'
        vol_name: 'Ansible_Test_SG_Vol'
        mv_name: 'Ansible_Testing_mv'
        portgroup_name: 'Ansible_Testing_portgroup'
        hostgroup_name: 'Ansible_Testing_hostgroup'
        host_name: 'Ansible_Testing_host'
        host_name2: 'Ansible_Testing_host2'

  tasks:

  - name: Get encrypted powermax credentials
    no_log: True
    include_vars:
        file: get_password.yml
  - set_fact:
        user : "{{ powermax_username }}"
        password : "{{ powermax_password }}"
    no_log: True

  - name: Delete MV "{{mv_name}}"
    dellemc_powermax_maskingview:
      unispherehost: "{{unispherehost}}"
      universion: "{{universion}}"
      verifycert: "{{verifycert}}"
      user: "{{user}}"
      password: "{{password}}"
      serial_no: "{{serial_no}}"
      mv_name: "{{mv_name}}"
      state: 'absent'

  - pause:
      prompt: "Paused! Please 'Enter' to continue"

  - name: Get volume details
    register: result
    dellemc_powermax_volume:
      unispherehost: "{{unispherehost}}"
      universion: "{{universion}}"
      verifycert: "{{verifycert}}"
      user: "{{user}}"
      password: "{{password}}"
      serial_no: "{{serial_no}}"
      sg_name: "{{sg_name}}"
      vol_name: "Ansible_Test_SG_Vol"
      state: 'present'

  - pause:
      prompt: "Paused! Please 'Enter' to continue"

  - name: Delete Storage group "{{sg_name}}"
    dellemc_powermax_storagegroup:
      unispherehost: "{{unispherehost}}"
      universion: "{{universion}}"
      verifycert: "{{verifycert}}"
      user: "{{user}}"
      password: "{{password}}"
      serial_no: "{{serial_no}}"
      sg_name: "{{sg_name}}"
      state: 'absent'

  - pause:
      prompt: "Paused! Please 'Enter' to continue"

  - name: Delete volume "{{vol_name}}"
    dellemc_powermax_volume:
      unispherehost: "{{unispherehost}}"
      universion: "{{universion}}"
      verifycert: "{{verifycert}}"
      user: "{{user}}"
      password: "{{password}}"
      serial_no: "{{serial_no}}"
      vol_id: "{{result.volume_details.volumeId}}"
      state: 'absent'

  - pause:
      prompt: "Paused! Please 'Enter' to continue"

  - name: Delete host group "{{hostgroup_name}}"
    dellemc_powermax_hostgroup:
      unispherehost: "{{unispherehost}}"
      universion: "{{universion}}"
      verifycert: "{{verifycert}}"
      user: "{{user}}"
      password: "{{password}}"
      serial_no: "{{serial_no}}"
      hostgroup_name: "{{hostgroup_name}}"
      state: "absent"

  - pause:
      prompt: "Paused! Please 'Enter' to continue"

  - name: Delete host "{{host_name}}"
    dellemc_powermax_host:
      unispherehost: "{{unispherehost}}"
      universion: "{{universion}}"
      verifycert: "{{verifycert}}"
      user: "{{user}}"
      password: "{{password}}"
      serial_no: "{{serial_no}}"
      host_name: "{{host_name}}"
      state: 'absent'

  - pause:
      prompt: "Paused! Please 'Enter' to continue"

  - name: Delete host "{{host_name2}}"
    dellemc_powermax_host:
      unispherehost: "{{unispherehost}}"
      universion: "{{universion}}"
      verifycert: "{{verifycert}}"
      user: "{{user}}"
      password: "{{password}}"
      serial_no: "{{serial_no}}"
      host_name: "{{host_name2}}"
      state: 'absent'

  - pause:
      prompt: "Paused! Please 'Enter' to continue"
  - name: Delete port group "{{portgroup_name}}"
    dellemc_powermax_portgroup:
      unispherehost: "{{unispherehost}}"
      universion: "{{universion}}"
      verifycert: "{{verifycert}}"
      user: "{{user}}"
      password: "{{password}}"
      serial_no: "{{serial_no}}"
      portgroup_name: "{{portgroup_name}}"
      state: "absent"
```

B Technical support and resources

Dell.com/support is focused on meeting customer needs with proven services and support.

Storage technical documents and videos provide expertise that helps to ensure customer success on Dell EMC storage platforms.

C Related resources

1. Ansible Modules for Dell EMC PowerMax 1.0 – Product Guide
2. Ansible Modules for Dell EMC PowerMax 1.0 – Release Notes
3. White paper – VMAX Auto-provisioning groups
4. Ansible documentation
5. FAQ on TimeFinder SnapVX (snapshots)
6. Technical Note on TimeFinder SnapVX
7. Blog Post on Role Based Access Controls in Unisphere