Table of Contents

Table of Figures

Table of Tables

Version History

Date	Version	Comment
30.08.2022	0.1	Initial Version
31.08.2022	0.2	Implemented changes proposed from Elmar
24.10.2022	0.3	Implemented suggestions from RH and answered open questions
22.11.2022	0.4	Changes according to remarks of NetApp colleagues

Table 1: Version history

Introduction

This document describes how to execute Ansible Playbooks against NetApp ONTAP systems.

Target audience and document goal

The document is intended to give SAP administrators an introduction to the Ansible automation of recurring administrative activities.

- This document includes
  - Brief description how to automate tasks on NetApp ONTAP systems
  - Example configuration of Ansible on a Ubuntu 20.04 server
    Instead of Ubuntu 20.094 any other OS can be used which supports Ansible and of course also systems which provide a complete Ansible orchestration like RedHat Ansible Automation Platform
  - Example Playbooks
- This document does not include
  - Any storage architecture related topics
  - Security related topics
    i.e. encrypting passwords

Getting started

This document has been written to demonstrate automating tasks using Ansible for NetApp ONTAP based systems. More details can be found in this documentation:
`https://netapp.io/2018/10/08/getting-started-with-netapp-and-ansible-install-ansible/`

This document can therefore be used for the following storage products:

- ONTAP (FAS/AFF)
- ONTAP Select
- Cloud Volumes ONTAP (CVO)
- Amazon FSx for NetApp ONTAP (FSxN)

To run Ansible Playbooks a prerequisite is a running Ansible version. There are a lot of tutorials how to install Ansible on MacOS, Linux, Windows are available on the internet. In our environment we decided to use Ubuntu as operating system and run Ansible. For completeness, we will also cover what needs to be configured on the Ubuntu system. This is described in the chapter Prepare Ubuntu system. If Ansible should run on another operating system, Ansible and the operating system need to be prepared accordingly.

What needs to be configured on the ONTAP based system will be described in the chapter Prepare ONTAP.

First, it is important to mention, that there are 4 ways to interact with ONTAP based systems.

Two frontends:

1) Web based System Manager
2) SSH

And two API interfaces:

3) ZAPI
4) REST API

The Web based System Manager and SSH cannot be used with Ansible!

The remaining APIs are REST API and ZAPI API. The last ONTAP release supporting ZAPI API is ONTAP 9.12.1. The API which should therefore be used to be future ready is REST API. To make sure, the default API being used is the REST API, we include the parameter `'use_rest: always'` in all Playbooks. If you have a look into the "NetApp.Ontap – Ansible Documentation" at `https://docs.ansible.com/ansible/latest/collections/netapp/ontap/index.html` you will recognize, that for each module certain parameters are documented. For some parameters it is explicitly mentioned that the parameter is only available using the REST API, for some parameters it is explicitly mentioned that the parameter ins only available using the ZAPI API. So usually, all other parameters are available for REST and ZAPI API.

There are four different scenarios depending on the API being used

1) We us a module and specify only parameters which are available using the REST API **and** ZAPI API. We **do not** specify any parameter which is exclusively available **only** with REST API or **only** with ZAPI API
   → This results in using the REST API
2) We us a module and specify parameters which are available using the REST API **and** ZAPI API. We specify at least one parameter which is available **only** with REST API.
   → This results in using the REST API
3) We us a module and specify parameters which are available using the REST API **and** ZAPI API. We specify at least one parameter available **only** with ZAPI API.
   → This results in using the ZAPI API. If ZAPI API is being used, Ansible will print a warning like this: `[WARNING]: Using ZAPI for na_ontap_command, ignoring 'use_rest: always'.`
4) We us a module and specify parameters which are available using the REST API and ZAPI API. We specify at least one parameter which is available **only** with ZAPI API **and** we specify at least one parameter which is available **only** with REST API.
   → This results in an error, because for a module only REST API or ZAPI API can be used

These four scenarios are listed for completeness and to make sure, users understand what will happen if they specify certain parameters. NetApp is releasing subsequently new versions of the netapp.ontap collection and parameters which are available with ZAPI only, will be supported step be step with REST API.

Since we must use REST API or ZAPI API to execute commands via Ansible on ONTAP based systems, SSH cannot be used as authentication mechanism. There are two options to connect using REST API or ZAPI API

1) User/Password based authentication
2) Certificate based authentication

User/Password based authentication is not the preferred way, because no one wants to enter a password in plain text in a config file. Certificate authentication is the preferred way to go. How to set up certificate-based authentication will be described in chapter Prepare ONTAP. Unfortunately, not all commands available via NetApp ONTAP command line are exposed through REST API and/or ZAPI API. ZAPI API offers a module to execute native ONTAP command line commands. When using this module, certificate authentication cannot be used. Instead, user/password authentication must be used.

Prepare the environment

In our environment we used an Ubuntu 20.04 system to run Ansible playbooks. This chapter describes what needs to be configured on the Ubuntu 20.04 system and on the NetApp ONTAP system to execute Ansible Playbooks. The basic requirements are:

1) Installed Ansible version
2) netapp.ontap collection
3) Necessary python dependencies

Prepare Ubuntu system

The preparation of the Ubuntu system with regards to the requirements described above consists of the following steps

1) Install Ansible version
    a. Add Ansible APT repository
    b. Install Ansible
2) netapp.ontap collection
    a. Verify collections and if necessary, upgrade netapp.ontap collection
3) Necessary python dependencies
    a. Install ZAPI Libs for Python

If you plan to run Ansible on a different operating system (i.e. MacOS), you have to implement the requirements accordingly.

Add Ansible APT repository

For adding the Ansible APT repository, execute the following steps:

- `holgerz@HOLGERZ02-PC:~/# mkdir ansible`
- `holgerz@HOLGERZ02-PC:~/# cd ansible`
- `holgerz@HOLGERZ02-PC:~/ansible# apt install software-properties-common`

- `holgerz@HOLGERZ02-PC:~/ansible# add-apt-repository --yes --update ppa:ansible/ansible`

Install Ansible

For installing Ansible, execute the following command:

- `holgerz@HOLGERZ02-PC:~/ansible# apt install ansible`

Verify successful installation by executing the following commands:

-
  ```
  holgerz@HOLGERZ02-PC:~/ansible# dpkg -l | grep ansible
  ii  ansible                        5.10.0-1ppa~focal
  all          batteries-included package providing a curated set
  of Ansible collections in addition to ansible-core
  ii  ansible-core                   2.12.7-1ppa~focal
  all          Ansible IT Automation
  ```

-
  ```
  holgerz@HOLGERZ02-PC:~/ansible# ansible –version
  ansible [core 2.12.8]
  config file = /etc/ansible/ansible.cfg
  configured module search path =
  ['/root/.ansible/plugins/modules',
  '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-
  packages/ansible
    ansible collection location =
  /root/.ansible/collections:/usr/share/ansible/collections
    executable location = /usr/bin/ansible
    python version = 3.8.10 (default, Jun 22 2022, 20:18:18) [GCC
  9.4.0]
    jinja version = 2.10.1
    libyaml = True
  ```

Verify collections and if necessary, upgrade `netapp.ontap` collection

Verify the actual version of the `netapp.ontap` collection as shown in Figure 1 at:

https://docs.ansible.com/ansible/latest/collections/netapp/ontap/index.html

Netapp.Ontap

Collection version 21.22.0

Figure 1: Current collection version from netapp.ontap

The current collection is version `21.22.0`. Now let's verify which collection version is installed:

- `holgerz@HOLGERZ02-PC:~/ansible# ansible-galaxy collection list | grep netapp`

```
# /usr/lib/python3/dist-packages/ansible_collections
Collection                    Version
----------------------------- -------
netapp.aws                    21.7.0
netapp.azure                  21.10.0
netapp.cloudmanager           21.18.0
netapp.elementsw              21.7.0
netapp.ontap                  21.20.0
netapp.storagegrid            21.10.0
netapp.um_info                21.8.0
netapp_eseries.santricity     1.3.0
```

The default collection search path points to `/usr/lib/python3/dist-packages/ansible_collections` and `netapp.ontap` collection is version `21.20.0`. Since this is not the current version, we need to upgrade to `netapp.ontap` version `21.22.0` and place the updated modules into the default collection search path. It is necessary to move the updated collections to the default search path since collections are installed as default in the home directory of the current user in `~/.ansible` and if any other user wants to execute Ansible playbooks on the same server he either needs to download the updated collection again or he well us a version of `netapp.ontap` which is not up to date.

To update the `netapp.ontap` collection for all users, the following steps need to be executed:

1) Install current version
```
holgerz@HOLGERZ02-PC:~/ansible#  ansible-galaxy collection
install netapp.ontap
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/netapp-ontap-
21.22.0.tar.gz to /root/.ansible/tmp/ansible-local-
107_ro3sp_1/tmp1mk2xjz8/netapp-ontap-21.22.0-85bxl69u
Installing 'netapp.ontap:21.22.0' to
'/root/.ansible/collections/ansible_collections/netapp/ontap'
netapp.ontap:21.22.0 was installed successfully
```
2) Remove `netapp.ontap` from default collection search path
```
holgerz@HOLGERZ02-PC:~/ansible# rm /usr/lib/python3/dist-
packages/ansible_collections/netapp/ontap
```
3) Move the newly installed netapp.ontap collection from `/root/.ansible/collections/ansible_collections/netapp/ontap` into the default collection search path
```
holgerz@HOLGERZ02-PC:~/ansible# mv
/root/.ansible/collections/ansible_collections/netapp/ontap
/usr/lib/python3/dist-packages/ansible_collections/netapp/
```

Install ZAPI Libs for Python

As mentioned in the Introduction, we can use the REST API or the ZAPI API when executing commands on the ONTAP system. For using the ZAPI API, there is the need to install the NetApp-Lib for Python. To achieve this, the following steps must be executed:

1) If necessary, install Python pip
```
holgerz@HOLGERZ02-PC:~/ansible# apt install python3-pip
```
2) Install NetApp ZAPI Python Libs
```
holgerz@HOLGERZ02-PC:~/ansible# pip install NetApp-Lib
```

Prepare ONTAP

Now we must prepare ONTAP to be able to access the REST API and ZAPI API using certificate base authentication. In addition, we must configure ONATP to access the ZAPI API using user/password for executing native ONTAP based command line interface commands.

The following steps are necessary:

- Create self-signed certificate on the Linux server on which you run the Ansible playbooks
  We will create a public key file and a private key file on the Linux server on which we plan to run the Ansible playbooks.
- Add certificates to ONTAP and configure users and passwords
  The public key file create in the step above will then be uploaded to the ONTAP system to allow authentication using the private key file, also created in the step above, to access the REST and/or ZAPI API.

Create self-signed certificate on the Linux server on which you run the Ansible playbooks

When you create the self-signed certificate on the Linux server, it is important to enter the username in "`Common Name`" which you are going to configure in ONTAP in the steps described in chapter Add certificates to ONTAP and configure users and passwords

- Create self-signed certificate
  The following command creates two files, `ontap.key` which is the private key and `ontap.pem` which is the public key:
```
holgerz@HOLGERZ02-PC:~/ansible# openssl req -x509 -nodes -days
1095 -newkey rsa:2048 -keyout ontap.key -out ontap.pem
Generating a RSA private key
.....................+++++
.....................................................+++++
writing new private key to 'ontap.key'
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
```

```
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:BW
Locality Name (eg, city) []:Stuttgart
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NetApp
Organizational Unit Name (eg, section) []:Testcenter
Common Name (e.g. server FQDN or YOUR name) []:holger
Email Address []:holger.zecha@netapp.com
```

Add certificates to ONTAP and configure users and passwords

We must add the private key of the certificate to ONTAP system and configure the user(s) who are allowed to log on with the corresponding public key file of the certificate. Some legacy commands unfortunately have the requirement to authenticate through the ZAPI API to the console using user/password. For completeness we therefore will also configure user/password access to run console commands through ZAPI API.

- SSH to the cluster management IP of your ONTAP Cluster and login with a user who has admin permissions
  ```
  holgerz@HOLGERZ02-PC:~/ansible# ssh admin@192.168.71.25
  (admin@192.168.71.25) Password:
  Last login time: 9/12/2022 15:04:18
  testcl1::>
  ```
- To install the public key file of your certificate execute the following command:
  ```
  testcl1::> security certificate install -type client-ca -vserver
  testcl1
  ```
  <<
  Insert content of the public key file `ontap.pem`
  >>
  ```
  You should keep a copy of the CA-signed digital certificate for
  future reference.
  The installed certificate's CA and serial number for reference:
  CA: holger
  serial: 5314F75B537821699ACB32C0CB85BBDC6EC3A472

  The certificate's generated name for reference: holger
  ```
- Create necessary user login information for REST API (application http) and ZAPI API (application ontapi) for certificate-based authentication
  ```
  testcl1::> security login create -user-or-group-name holger -
  application ontapi -authentication-method cert -vserver testcl1
  testcl1::> security login create -user-or-group-name holger -
  application http -authentication-method cert -vserver testcl1
  ```
- Since we also need user/password-based authentication for the ZAPI API to excute native console commands, we configure this authentication method using the following command
  ```
  testcl1::> security login create -user-or-group-name holger -
  application ontapi -authentication-method password -vserver
  testcl1
  ```
- To execute console commands via ZAPI API we also must configure a user for console access with the following command

```
testcl1::> security login create -user-or-group-name holger -
application console -authentication-method password -vserver
testcl1
```

Workflows

Now that everything is prepared on the Linux server and inside ONTAP we can start to configure the necessary Ansible Playbooks

- Day 1 automation
- Daily operation
- Day 2 automation
- In addition, we added Playbooks needed for Cleanup
- Querying ONTAP information is described in Querying ONTAP information using Playbooks

Day 1

The necessary steps for Day 1 automation are visualized in Figure 2. The following Playbooks are needed:

- create_svm.yml
  If necessary, create a new SVM which will be used for creating the needed volumes.
- set_svm_options.yml
  Set SVM options needed for optimal performance
- create_export_policy.yml
  If necessary, create a new export which will be assigned to the newly created volumes.
- create_volume.yml
  Create new volumes
- create_snapshot.yml
  Take a SnapShot from the newly created volumes

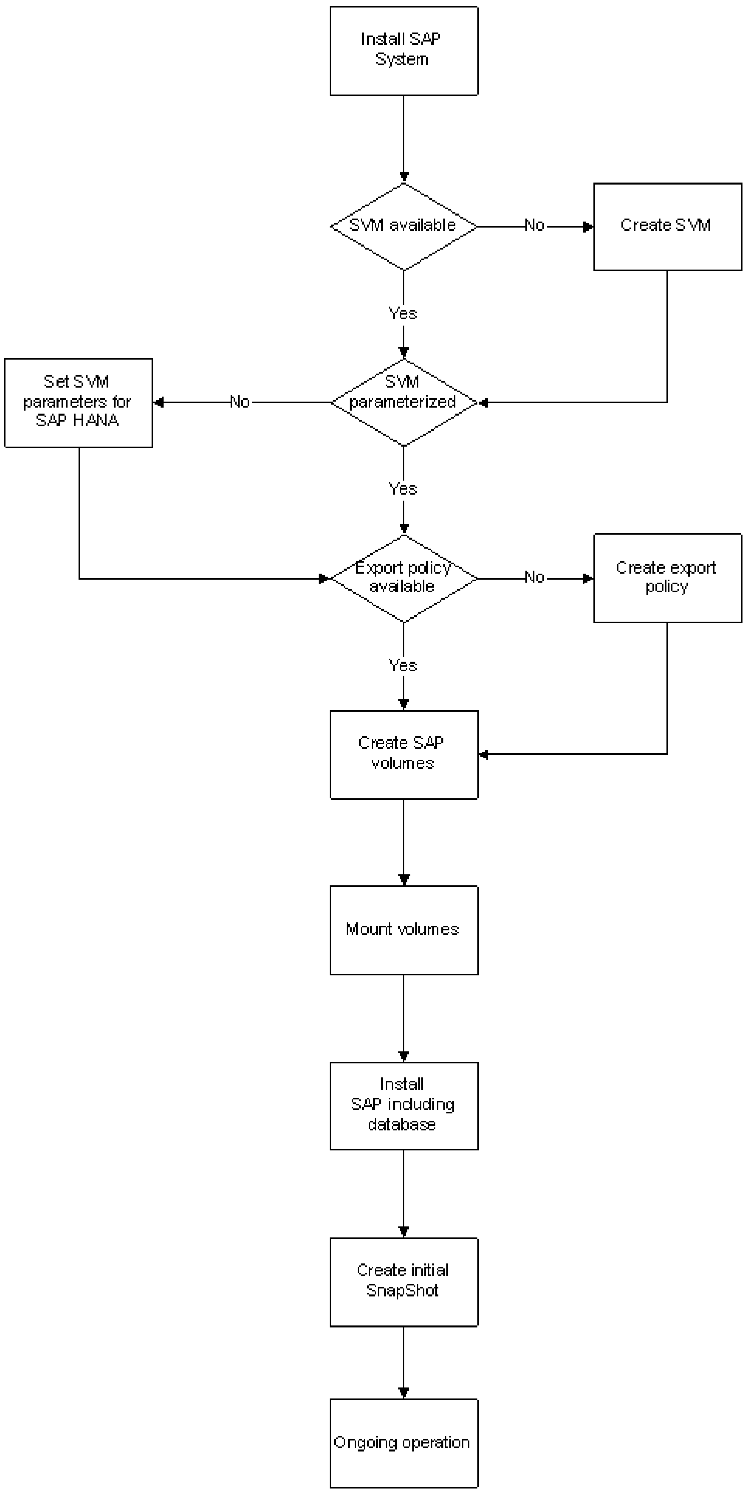

Figure 2: Workflow Day 1 automation

Daily operation

The necessary steps for daily operation are visualized in Figure 3. The following Playbooks are needed:

- restore_snapshot.yml
  Restore a SnapShot

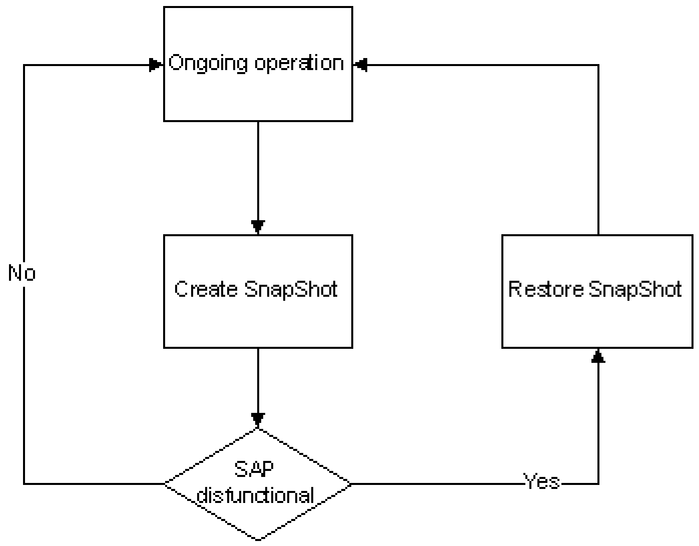

Figure 3: Workflow for daily operation

Day 2

For Day 2 automation we assume, that we need to do SAP system refreshes. The workflow for Day 2 automation is visualized in Figure 4. The following Playbooks are needed:

- create_snapshot.yml
  If necessary, create a SnapShot
- create_clone.yml
  Create FelxClone

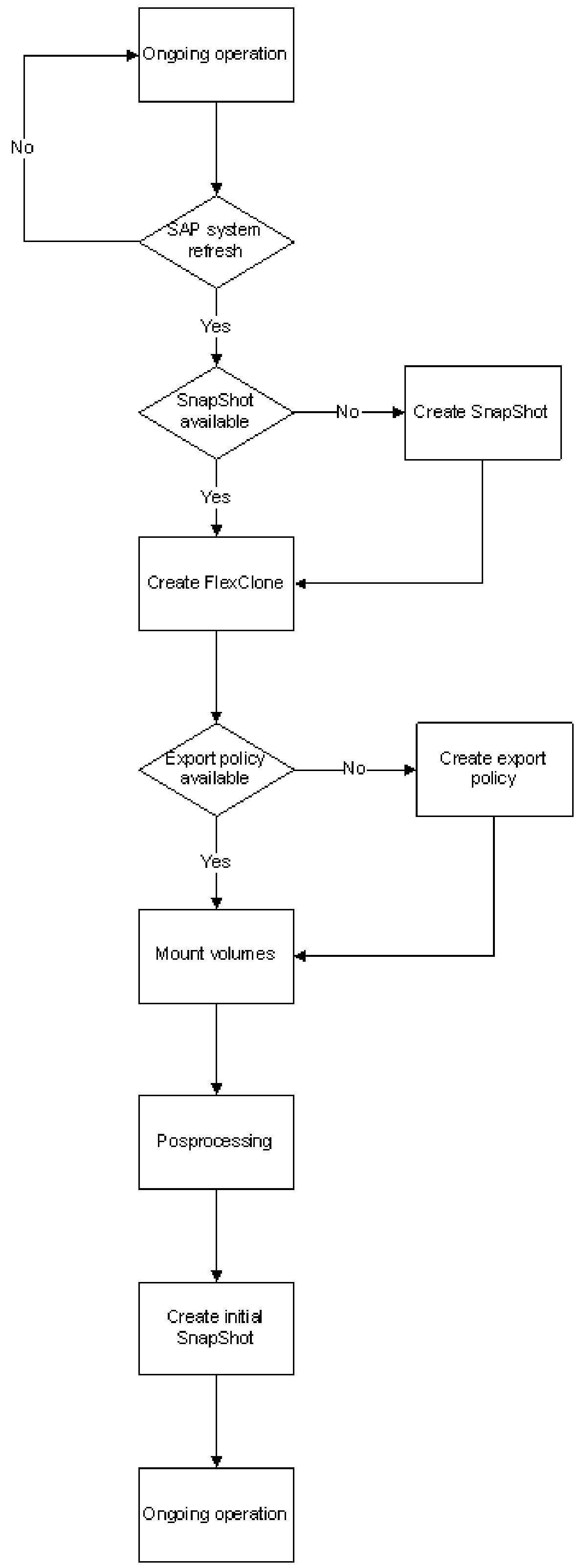

Figure 4: Workflow Day 2 automation

Cleanup

To clean up everything i.e., deleting a FlexClone before creating a new FlexClone the following Playbooks are needed:

- delete_clone.yml
  Delete existing FlexClone
- delete_snapshot.yml
  Delete existing SnapShot
- delete_volume.yml
  Delete existing Volume
- delete_svm.yml
  Delete Storage Virtual Machine

Querying ONTAP information using Playbooks

As mentioned before, there are two ways of querying ONTAP details.

- Running native CLI commands using ZAPI API
  system_details.yml (ZAPI API and native CLI command)
- Querying predefined information using REST API.
  get_svms.yml (REST API)

If information is needed which is not accessible via REST AP, ZAPI API must be used. Examples are documented in the two YAML files mentioned above.

Running Playbooks

There are three ways when running Playbooks

1. Code all parameters inside the Playbook
2. Use an inventory defined in an inventory file. To run a Playbook using an inventory defined in `inventory.yml` execute
   `ansible-playbook -i inventory.yml create_svm.yml`
3. Use variables defined in a variable file. To run the Playbook Playbook using an inventory defined in `ontap_vars.yml` execute
   `ansible-playbook create_svm.yml --extra-vars "@ontap_vars.yml"`

An example inventory file is shown in chapter inventory.yml.

An example variable file is shown in chapter ontap_vars.yml.

The key value pairs are self-explaining and fit parameters which are described for each `netapp.ontap` Ansible module. The subsequent Playbooks make use of the defined variables either from inventory.yml or from ontap_vars.yml

YAML Files

Here are all needed YAML files.

inventory.yml

```
ontapservers:

  hosts:

    testcl1-01:

      hostname: 192.168.71.25 or ansible_host (use inventory_hostname
then in playbook)

      ansible_host: 192.168.71.25

      username: "holger"

      password: "your password"

      keyfile: "/root/ansible/certs/ontap.key"

      certfile: "/root/ansible/certs/ontap.pem"

      svmname: "svm-sap03"

      aggrlist: "data_aggr_0"

      exportpolicyname: "192er_LAN_SAP"

      sizeunit: "gb"

      datavolumesize: "100"

      datavolumename: "L01_data"

      logvolumename: "L01_log"

      logvolumesize: "256"

      sharedvolumename: "L01_shared"

      sharedvolumesize: "256"

      dataaggrname: "data_aggr_0"

      protocols: "nfs,nfs3"

      networkrange: "192.168.71.0/24"

      ruleindex: "100"

      rorule: "none"

      rwrule: "any"

      snapshotpostfix: "_snap_1"

      clonepostfix: "_clone_1"

linuxservers:

   hosts:
     velociraptor:
       ansible_host: 192.168.71.229
       ansible_ssh_user: holger
       ansible_password: <your password>
```

ontap_vars.yml

```
hostname: "192.168.71.25"
username: "holger"
password: "your password"
keyfile: "/root/ansible/certs/ontap.key"
certfile: "/root/ansible/certs/ontap.pem"
svmname: "svm-sap03"
aggrlist: "data_aggr_0"
exportpolicyname: "192er_LAN_SAP"
sizeunit: "gb"
datavolumesize: "100"
datavolumename: "L01_data"
logvolumename: "L01_log"
logvolumesize: "256"
sharedvolumename: "L01_shared"
sharedvolumesize: "256"
dataaggrname: "data_aggr_0"
protocols: "nfs,nfs3"
networkrange: "192.168.71.0/24"
ruleindex: "100"
rorule: "none"
rwrule: "any"
snapshotpostfix: "_snap_1"
clonepostfix: "_clone_1"
```

system_details.yml (ZAPI API and native CLI command)

```
---
```

```
- name: Get System details 1
  connection: local
  collections:
    - netapp.ontap
  hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables


  tasks:
  - name: Get details of installed cluster
    na_ontap_command:
      use_rest: always
      hostname: "{{ (inventory_)hostname }}"
      username: "{{ username }}"
      password: "{{ password }}"
      https: true
      validate_certs: false
      command: ['system show -instance']
    register: ontap_return
  - debug: var=ontap_return
```

get_svms.yml (REST API)

```
---


- name: Get SVMs
  collections:
    - netapp.ontap
  hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables
  connection: local


  tasks:
```

```
  - name: Get details of configured SVMs
    na_ontap_rest_info:
      use_rest: always
      hostname: "{{ (inventory_)hostname }}"
      cert_filepath: "{{ certfile }}"
      key_filepath: "{{ keyfile }}"
      https: true
      validate_certs: false
      gather_subset:
      - svm/svms
    register: ontap_return
  - debug: var=ontap_return
```

create_svm.yml

```
---
- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables
  connection: local
  collections:
    - netapp.ontap
  gather_facts: false
  name: Onboard SVM
  tasks:
  - name: Create SVM
    na_ontap_svm:
      state: present
      name: "{{ svmname }}"
      use_rest: always
      services:
        cifs:
          allowed: false
        fcp:
          allowed: false
        nfs:
          allowed: true
          enabled: true
      aggr_list: "{{ aggrlist }}"
      hostname: "{{ (inventory_)hostname }}"
      cert_filepath: "{{ certfile }}"
      key_filepath: "{{ keyfile }}"
      https: true
      validate_certs: false
```

set_svm_options.yml

```
---
- hosts: ontapservers|localhost - depending if inventory.yml will be
  connection: local
  collections:
    - netapp.ontap
  gather_facts: false
  name: Set SVM Options
  tasks:
  - name: Set SVM Options via CLI
    na_ontap_command:
      use_rest: always
      hostname: "{{ (inventory_)hostname }}"
      username: "{{ username }}"
      password: "{{ password }}"
      https: true
      validate_certs: false
      command: ['set advanced -confirmations off; nfs modify -vserver
"{{ svmname }}" -tcp-max-xfer-size 1048576; vol modify -vserver "{{
svmname }}" -volume "{{ datavolumename }}"  -snapdir-access true; vol
modify -vserver "{{ svmname }}" -volume "{{ datavolumename }}" -
snapshot-policy none; vol modify -vserver "{{ svmname }}" -volume "{{
datavolumename }}" -atime-update false']
```

create_export_policy.yml

```
---

- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables

  connection: local

  collections:

    - netapp.ontap

  gather_facts: false

  name: Export Policy

  tasks:

  - name: Create Export Policy

    na_ontap_export_policy_rule:

      state: present

      name: "{{ exportpolicyname }}"

      vserver: "{{ svmname }}"

      rule_index: "{{ ruleindex }}"

      client_match: "{{ networkrange }}"

      protocol: "{{ protocols }}"

      hostname: "{{ (inventory_)hostname }}"

      ro_rule : "{{ rorule }}"

      rw_rule: "{{ rwrule }}"

      cert_filepath: "{{ certfile }}"

      key_filepath: "{{ keyfile }}"

      https: true

      validate_certs: false
```

create_volume.yml

```
---

- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables

  connection: local
  collections:
    - netapp.ontap
  gather_facts: false
  name: Onboard FlexVol
  tasks:
  - name: Create Volume
    na_ontap_volume:
      state: present
      name: "{{ datavolumename }}"
      aggregate_name: "{{ dataaggrname }}"
      use_rest: always
      size: "{{ datavolumesize }}"
      size_unit: "{{ sizeunit }}"
      tiering_policy: none
      export_policy: "{{ exportpolicyname }}"
      percent_snapshot_space: 80
      vserver: "{{ svmname }}"
      junction_path: '/{{ datavolumename }}'
      wait_for_completion: True
      hostname: "{{ (inventory_)hostname }}"
      cert_filepath: "{{ certfile }}"
      key_filepath: "{{ keyfile }}"
      https: true
      validate_certs: false
```

create_snapshot.yml

```
---
- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables
  connection: local
  collections:
    - netapp.ontap
  gather_facts: false

  name: SnapShot

  tasks:

  - name: Create SnapShot

    na_ontap_snapshot:

      state: present

      snapshot: "{{ datavolumename }}{{ snapshotpostfix }}"

      use_rest: always

      volume: "{{ datavolumename }}"

      vserver: "{{ svmname }}"

      hostname: "{{ (inventory_)hostname }}"

      cert_filepath: "{{ certfile }}"

      key_filepath: "{{ keyfile }}"

      https: true

      validate_certs: false
```

restore_snapshot.yml

```
---

- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables

  connection: local

  collections:

    - netapp.ontap

  gather_facts: false

  name: Restore FlexVol

  tasks:

  - name: Restore Volume

    na_ontap_volume:

      state: present

      name: "{{ datavolumename }}"

      use_rest: always

      snapshot_restore: "{{ datavolumename }}{{ snapshotpostfix }}"
      vserver: "{{ svmname }}"
      wait_for_completion: True
      hostname: "{{ (inventory_)hostname }}"
      cert_filepath: "{{ certfile }}"
      key_filepath: "{{ keyfile }}"
      https: true
      validate_certs: false
```

create_clone.yml

```
---
- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables
  connection: local
  collections:
    - netapp.ontap
  gather_facts: false
  name: Create FlexClone
  tasks:
  - name: Clone Volume
    na_ontap_volume_clone:
      state: present
      name: "{{ datavolumename }}{{ clonepostfix }}"
      use_rest: always
      vserver: "{{ svmname }}"
      junction_path: '/{{ datavolumename }}{{ clonepostfix }}'
      parent_volume: "{{ datavolumename }}"
      parent_snapshot: "{{ datavolumename }}{{ snapshotpostfix }}"
      hostname: "{{ (inventory_)hostname }}"
      cert_filepath: "{{ certfile }}"
      key_filepath: "{{ keyfile }}"
      https: true
      validate_certs: false
```

delete_clone.yml

```
---

- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables

  connection: local

  collections:

    - netapp.ontap

  gather_facts: false

  name: Delete FlexClone

  tasks:

  - name: Delete Clone

    na_ontap_volume:

      state: absent

      name: "{{ datavolumename }}{{ clonepostfix }}"

      aggregate_name: "{{ dataaggrname }}"

      use_rest: always

      vserver: "{{ svmname }}"

      wait_for_completion: True

      hostname: "{{ (inventory_)hostname }}"

      cert_filepath: "{{ certfile }}"

      key_filepath: "{{ keyfile }}"

      https: true

      validate_certs: false
```

delete_snapshot.yml

```
---

- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables

  connection: local

  collections:

    - netapp.ontap

  gather_facts: false
  name: SnapShot
  tasks:
  - name: Delete SnapShot
    na_ontap_snapshot:
      state: absent
      snapshot: "{{ datavolumename }}{{ snapshotpostfix }}"
      use_rest: always
      volume: "{{ datavolumename }}"
      vserver: "{{ svmname }}"
      hostname: "{{ (inventory_)hostname }}"
      cert_filepath: "{{ certfile }}"
      key_filepath: "{{ keyfile }}"
      https: true
      validate_certs: false
```

delete_volume.yml

```
---
- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables
  connection: local
  collections:
    - netapp.ontap
  gather_facts: false
  name: Delete FlexVol
  tasks:
  - name: Delete Volume
    na_ontap_volume:
      state: absent
      name: "{{ datavolumename }}"
      aggregate_name: "{{ dataaggrname }}"
      use_rest: always
      vserver: "{{ svmname }}"
      wait_for_completion: True
      hostname: "{{ (inventory_)hostname }}"
      cert_filepath: "{{ certfile }}"
      key_filepath: "{{ keyfile }}"
      https: true
      validate_certs: false
```

delete_svm.yml

```
---
- hosts: ontapservers|localhost - depending if inventory.yml will be
used or variables
  connection: local
  collections:
    - netapp.ontap
  gather_facts: false
  name: SVM
  tasks:
  - name: Delete SVM
    na_ontap_svm:
      state: absent
      name: "{{ svmname }}"
      use_rest: always
      aggr_list: "{{ aggrlist }}"
      hostname: "{{ (inventory_)hostname }}"
      cert_filepath: "{{ certfile }}"
      key_filepath: "{{ keyfile }}"
      https: true
      validate_certs: false
```

References

https://netapp.io/2016/11/08/certificate-based-authentication-netapp-manageability-sdk-ontap/

https://docs.netapp.com/us-en/ontap/authentication/install-server-certificate-cluster-svm-ssl-server-task.html

https://docs.ansible.com/ansible/latest/collections/netapp/ontap/index.html

https://netapp.io/2018/10/08/getting-started-with-netapp-and-ansible-install-ansible/

https://docs.ansible.com/ansible/2.9/installation_guide/intro_installation.html

https://galaxy.ansible.com/netapp/ontap

https://github.com/ansible-collections/netapp.ontap