

Technical Report

Demo Ansible in Lab On Demand

Louis Pauls and Yaron Regev, NetApp  
September 2019

TABLE OF CONTENTS

[1 Introduction 3](#_Toc18486968)

[2 Setting Up the Demo Environment 4](#_Toc18486969)

[3 Ansible Ad-hoc Commands 5](#_Toc18486970)

[4 Ansible Playbooks 7](#_Toc18486971)

[4.1 Configuring *cluster1* using the Ansible Playbook 8](#_Toc18486972)

[4.2 The Idempotent Nature of Ansible 9](#_Toc18486973)

[4.3 Using Ansible to Deploy Trident 10](#_Toc18486974)

[5 Ansible Roles 16](#_Toc18486975)

[5.1 Configuring *cluster1* using the Ansible Playbook and NetApp Roles 17](#_Toc18486976)

[Appendix A: Alternative Method to Installing Ansible 20](#_Toc18486977)

[Appendix B: Configuration Files Used in the Demo 21](#_Toc18486978)

# Introduction

Ansible is an open-source automation engine which can greatly improve consistency, scalability and reliability of an hybrid-cloud environment. Ansible is primarily used for three types of tasks: configuration management (modify the configuration files across many hosts, switches, cloud providers, etc.), application deployment, especially in the Devops world (automating the deployment and update of applications), and provisioning (i.e. setting up servers/compute instances/storage arrays/switches etc. as part of initial deployment). Ansible can automate many tasks across many environment – anywhere from traditional bare metal servers to virtualization platforms or the cloud.

Ansible was acquired by Red Hat in 2015. In additional to the open-source free version of Ansible (which is what will be used in the exercises in the lab), there is also a commercial version available called Ansible Tower. Red Hat Ansible Tower enhances the Ansible engine by providing a visual dashboard as well as other enterprise features like RBAC, job scheduling, a powerful notification engine, as well as an inventory management framework. In addition, an upstream version of Tower is available as well. This product is known as AWX. It is open-source, community supported and free to download and use, but it is not officially supported by Red Hat. It should therefore not be used in production environments. Features that are added to AWX through community contributions typically makes itself into Red Hat Tower at some later point.

The exercises in this lab document makes use of the command line open-source version of Ansible. It is very easy to install using native Linux/UNIX commands as will be evident through the exercises.

There are many other competing solutions to Ansible in the Configuration Management (CM) space, most notable are Puppet and Chef. One of the major differentiators between Ansible and the other CM products is that Ansible is a based on a push architecture, and does therefore not require client software or agents to be installed on the infrastructure nodes it manages. It simply uses sh and Ansible modules to access the assets it is tasked to manage. And just as important, it uses a very simple language (YAML, in the form of Ansible playbooks) that allows one to very easily describe automation in a way that approaches plain English.

Ansible modules, which will be discussed in detail later, are key to the architecture of Ansible. Ansible modules essentially do all the heavy lifting – it executes the code across the environment by utilizing parameters passed to the modules from within an Ansible playbook. In doing so it abstracts the user from the underlying complexities of configuration management.

In 2018 Red Hat introduced the Ansible Certification and NetApp is one of six companies that participated in this certification program since its initiation. NetApp was the only data management company on that initial list. Vendors that are part of this program submits their modules to Red Hat which in turn then test, certify and supports the modules. It is very important to bring this up with customers. The NetApp modules are tested and supported by Red Hat. This speaks to the tight integration between NetApp and Red Hat when it comes to Ansible. For a list of all vendors that are members of this certification program please see this [link](https://www.ansible.com/certified-partners).

In the next section we will install Ansible in the Lab on demand environment.

# Setting Up the Demo Environment

The instructions provided in this section will work with the Lab on Demand (LOD) lab named “**Exploring the ONTAP REST API v1.0**”. If you are trying to use Ansible with a different LOD lab and the instructions do not work, please see Appendix A for an alternative way to deploy Ansible.

After reserving and connecting to the lab, log into the Linux server *rhel1* using PuTTY, with username *root* and password *Netapp1!*

1. Using git, download the lab files we will be using throughout the demo. Do this by typing the following commands:

[root@rhel1 ~]# **cd ~/**  
[root@rhel1 ~]# **git clone https://github.com/louispauls/ansible**

1. Install Ansible on *rhel1* (the Ansible Control Node) by typing the following command:

[root@rhel1 ~]# **yum install –y ansible**

Since the Ansible architecture is a push architecture, Ansible does not need to be installed on any of the nodes or other infrastructure it needs to manage. In only runs on the one control node and pushes commands from this control node to the rest of the environment.

1. Since Ansible modules are written in Python, in order to use the NetApp ONTAP modules we need to install the ONTAP python library. Do this by typing the following command:

[root@rhel1 ~]# **pip install netapp-lib**

1. The final step in configuring Ansible is to copy the hosts file (the file that contains the host names of the systems we want to access using Ansible) into the Ansible directory (when prompted, respond “y” to overwrite the existing (default) host file):

[root@rhel1 ~]# **mv ansible/lod/hosts /etc/ansible/**

This concludes the installation of Ansible on the Ansible Control Node (*rhel1* in our example). In the next section we will focus on some basic Ansible functionality using *Ansible Ad-hoc Commands* before moving on to explaining the concepts of *Ansible Playbooks* and *Ansible Roles*.

# Ansible Ad-hoc Commands

We can (and will, later in this exercise) use Ansible to configure the ONTAP cluster using Ansible Playbooks and Roles, but before we do that, let’s explore some very basic functionality of Ansible by executing some commands in an ad-hoc fashion on the two Red Hat hosts *rhel1* and *rhel2*.

As an example, let’s use *ansible* to ask the question what version of *docker* is installed on the host *rhel1*:

[root@rhel1 ~]# **ansible rhel1 –a "docker –-version"**

**Note:** *ansible* uses *sh* to execute commands on hosts, and the first time one uses a remote sh command it will prompt for authentication (assuming public/private ssh key authentication is configured). Type *yes* when prompted. You will get a message that the host has been added to the lists of known hosts. You should then see the response

**rhel1 | CHANGED | rc=0 >>**

**Docker version 17.05.0-ce, build 89658be**

This simply means that Ansible used a module to execute the command docker –version on the host *rhel1.*

Let’s now execute the same command on *rhel2* but use Ansible to do so from the Ansible Control Node *rhel1* **so from within the *rhel1* PuTTY session** type the command:

[root@rhel1 ~]# **ansible rhel2 –a "docker –-version"**

and type “yes” (if prompted). This will now show the version of *docker* running on *rhel2* by using Ansible installed on *rhel1.*

These simple commands demonstrate the power of Ansible. One can run very basic to very sophisticated commands on any host in the environment by using Ansible installed on the Ansible Control Node. The question ow becomes how does one issue the same command to many hosts at once, or a subset of the hosts in the averment (all the web servers for instance).

That is where the concept of the Ansible Inventory file comes in. As part of setting up the demo environment a file named *hosts* was copied to */etc/ansible*. This file allows one to define groups of hosts, which one can then pass to Ansible in a similar fashion as above (i.e. instead of passing the host name, one can pass a group names as defined in the hosts file, and Ansible will execute the command on all the hosts in that group). The */etc/ansible/hosts* file in our example is very simple, defining a single group named *prod* with members *rhel1* and *rhel2* as seen below:

[prod]

rhel1

rhel2

One can now use the group to check the versions of docker running on these hosts using a single Ansible command. Do so by running the following command:

[root@rhel1 ~]# **ansible prod –a "docker ––version"**

The result now shows the version of docker installed on both nodes *rhel1* and *rhel2*. It is easy to see how this can save a tremendous amount of time when configuring (in this case) hosts. To scale, one can simply add more members to the *prod* group in */etc/ansible/hosts*, or add more groups (e.g. *web-servers*, *dev, oracle*) and execute commands on these *hosts* individually or on the *groups* or on *all* which will execute the command on all hosts in the hosts file.

As an exercise, go ahead and execute the following commands on the group *prod*:

df –h to see the file systems on each of the hosts

uname –a returns some system information on each host

Hint: execute the command ansible prod –a “<command>” in the PuTTY session (replace <command> with the commands above, one at a time).

One can also use *ansible* to install software on hosts. Let’s assume we want to pull down the *docker* image *ubuntu* from *docker hub*. To do so, simply execute the command:

[root@rhel1 ~]# **ansible prod –a "docker image pull ubuntu"**

To verify the image is pulled down, execute:

[root@rhel1 ~]# **ansible prod –a "docker image ls"**

The *ubuntu* docker image should now be residing on both hosts (sample output below):

**rhel2 | CHANGED | rc=0 >>**

**REPOSITORY TAG IMAGE ID CREATED SIZE**

**ubuntu latest a2a15febcdf3 2 weeks ago 64.2MB**

**rhel1 | CHANGED | rc=0 >>**

**REPOSITORY TAG IMAGE ID CREATED SIZE**

**ubuntu latest a2a15febcdf3 2 weeks ago 64.2MB**

# Ansible Playbooks

From the previous section the power of Ansible is evident. But how does one now string a set of commands together to form a list of repeatable actions, and execute these actions on many hosts (and/or switches, cloud objects, and/or NetApp ONTAP, Santricity or Element instances) without typing the commands in one-by-one in an ad-hoc fashion as above? This is where the concept of Ansible Playbooks come in. An Ansible Playbook is simply a combination of Tasks, where each task calls an Ansible module. As an example, to configure the NetApp cluster *cluster1* in the lab environment, one can execute an Ansible playbook that will consist of several tasks (create aggregates, create SVMs, create LIFs, configure NFS etc.), each task calling on an Ansible module to perform the appropriate action.

The file *configure\_ontap\_playbook.yml* (in the *ansible/lod* folder on *rhel1*)is an Ansible playbook that does exactly that, and more. Let’s look at this file in some detail.

(Use the command: more ansible/lod/configure\_ontap\_playbook.yml to look at the file if you like).



Figure 1 Playbook to configure the ONTAP Cluster using calls to Ansible modules

This playbook consists of two plays.

* The first play configures the ONTAP Cluster. It consists of nine tasks (Create Aggregates, Create SVM, etc.) These tasks will be executed in order, starting at the top.
* The second play in this playbook configures the hosts in the ansible group *prod* (which in our example consists of the hosts *rhel1* and *rhel2*). It will install NFS on the hosts in *prod* (first task), and then mount a NFS export (second task) which was created as [art of the first play on the NetApp cluster.

Before we execute the playbook, let’s look at the task that creates the SVM in more detail:



Figure 2 The Task from this playbook that creates a SVM

This task calls the module [*na\_ontap\_svm*](https://docs.ansible.com/ansible/latest/modules/na_ontap_svm_module.html#na-ontap-svm-module) and passes variables (*state, name, root\_volume*, etc.) to it. Everything in the curly brackets {{…}} are variables defined in the file \_*var\_playbook.yml* in the *ansible/lod* folder on *rhel1*. A full list of all supported NetApp modules can be found [here](https://docs.ansible.com/ansible/latest/modules/list_of_storage_modules.html#netapp). Scroll on this page to see modules published by Pure, EMC and others (note that NetApp has over 100 supported modules, many more than any of our competitors). The [Ansible Module Index](https://docs.ansible.com/ansible/latest/modules/modules_by_category.html) lists all the modules published to Red Hat (click on *Cloud modules* for instance to see a list of all the cloud provider modules - AWS, Azure, GCP, others). Remember that only a handful of vendor modules are tested and supported by Red Hat. Most vendors’ modules are not supported by Red Hat, as discussed in the Introduction.

## Configuring *cluster1* using the Ansible Playbook

In order to now execute the playbook that will configure and setup *cluster1* and mount an export to the hosts *rhel1* and *rhel2*, simply execute the following command in the PuTTY session on *rhel1* (from within the *ansible/lod* folder)

[root@rhel1 ~]# **cd ~/ansible/lod**  
[root@rhel1 ~]# **ansible-playbook configure\_ontap\_playbook.yml**

As soon as the command completes, log into *cluster1* using the NetApp System Manager (through Google Chrome with **username: admin** and **password: Netapp1!**)and verify that a SVM (SVM\_NFS) is created, aggregates are created, NFS is configured etc. Also verify that the NFS export is mounted to *rhel1* and *rhel2* (the playbook mounted the export to /mnt/tmp on both hosts in the group *prod*). Easiest way to do that is to execute the command *df –h* on both hosts (by using Ansible) so:

[root@rhel1 ~]# **ansible prod –a "df –h /mnt/tmp"**

You should see an entry for each of the hosts pointing to the export on the *LIF 192.168.0.145* on *SVM\_NFS* on *cluster1*:

**rhel2 | CHANGED | rc=0 >>**

**Filesystem Size Used Avail Use% Mounted on**

**192.168.0.145:/NfsDataVolume 9.5G 256K 9.5G 1% /mnt/tmp**

**rhel1 | CHANGED | rc=0 >>**

**Filesystem Size Used Avail Use% Mounted on**

**192.168.0.145:/NfsDataVolume 9.5G 256K 9.5G 1% /mnt/tmp**

*Note that this playbook did not apply any of the protocol and feature licenses to the cluster (since the LOD instance already had these license codes installed). However, one can simply add a task to the playbook to call the module* [*na\_ontap\_license*](https://docs.ansible.com/ansible/latest/modules/na_ontap_license_module.html#na-ontap-license-module) *to accomplish this. (The same holds for the assignment of the disks (*[*na\_ontap\_disks*](https://docs.ansible.com/ansible/latest/modules/na_ontap_disks_module.html#na-ontap-disks-module)*)).*

## The Idempotent Nature of Ansible

Idempotency, simply put is a feature of Ansible that allows one to make the same call repeatedly to an object (host, NetApp cluster, etc.) while producing the same result. In other words, making multiple identical requests has the same effect as making a single request. **This is a very powerful feature!** One can run the playbook *configure\_ontap\_playbook.yml* again and again, and it will not fail second time around. Executing a script for instance to create an aggregate that already exists will fail with a message saying that the aggregate by that name already exists. But Ansible being idempotent will not fail. It will simply skip that step, and move to the next step.

To demonstrate this feature, go ahead and execute the playbook again running the same command as above and look at the output:

[root@rhel1 ~]# **ansible-playbook configure\_ontap\_playbook.yml**

All output is green which verifies that everything is exactly as it was configured when the playbook was executed the first time:

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

localhost : ok=9 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

rhel1 : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

rhel2 : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

This is an extremely powerful feature of Ansible (one of the most important ones). One can identify and correct problems very fast using Ansible*.*

To demonstrate this feature, log into *cluster1* through the NetApp System Manager and **delete** the two data-serving LIFs. One cannot simply delete the LIF – the LIF needs to be *Disabled* first. In order to do that, follow these two steps:

* Go to (Network/Network Interfaces), highlight *nfs\_data\_lif1*, click on *Status* and select *Disable*. Do the same for *nfs\_data\_lif2.* With both NFS data LIFs disabled, click on *Refresh* and verify that the interfaces are disabled.
* Select the two disabled interfaces and click on Delete and confirm that both are now deleted.

This will immediately cause the hosts rhel1 and rhel2 not to have access to the NFS export anymore. This can be verified by executing the command df on rhel1. The session will hang. Hit <Ctrl> C several times to break out of the hang condition. If you like, try to access /mnt/tmp (the mount point for the NFS export) to prove it’s gone. If this should happen in a production environment, cases are opened with IT and NetApp to troubleshoot the problem.

A simple way to fix this is to simply run the Ansible playbook that holds the original and tested configuration again, in our case *configure\_ontap\_playbook.yml*. So go ahead and execute:

[root@rhel1 ~]# **cd ~/ansible/lod**  
[root@rhel1 ~]# **ansible-playbook configure\_ontap\_playbook.yml**

One can see that ansible will execute all the task, skipping most (**green**) with the exception of the two tasks that create the two deleted interfaces (**yellow**).

TASK [Create Aggregates (na\_ontap\_aggregate)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ok: [localhost] => (item={u'node': u'cluster1-01', u'name': u'n1\_aggr1'})**

**ok: [localhost] => (item={u'node': u'cluster1-02', u'name': u'n2\_aggr1'})**

TASK [Create a SVM (na\_ontap\_svm)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ok: [localhost]**

TASK [Create an Interface (na\_ontap\_interface)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**changed: [localhost]**

TASK [Create an Interface (na\_ontap\_interface)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**changed: [localhost]**

TASK [Configure NFS (na\_ontap\_nfs)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ok: [localhost]**

TASK [Create default Export Policy Rule (na\_ontap\_export\_policy\_rule)] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ok: [localhost]**

After the playbook completes, the cluster should now be back in an operational state. Look at the list of interfaces on *cluster1* to verify that the interfaces are operational again. Also execute df on *rhel1* to verify normal operations.

## Using Ansible to Deploy Trident

As we discussed earlier, Ansible can be used to install software packages across multiple hosts. We have already verified that Docker is installed on *rhel1* and *rhel2* (by issuing the command: docker –-version earlier through Ansible). In this section, we will deploy the NetApp Trident plugin for Docker on these hosts by using an Ansible playbook. Before we run this playbook though, we need to install the Docker SDK for Python. Do so by issuing the following two commands:

[root@rhel1 ~]# **ansible prod –a "yum install –y python-pip”**

followed by:

[root@rhel1 ~]# **ansible prod –a "pip install docker”**

As soon as these two packages are installed/updated, we can now proceed with running Docker containers on *rhel1* and *rhel2* which is deployed by running the playbook deploy\_trident.yml.

This playbook consists of six tasks:

1. Configure Trident: copies the Trident configuration file (\_*trident\_config\_file.json*) to the */etc/netappdvp/* folder on both hosts defined in the Ansible group *prod*
2. Install Trident: installs the Trident plugin to Docker on both hosts defined in the Ansible group *prod*
3. Create Persistent Volume: uses Trident to create a Docker Persistent Volume (PV) called *pvol1* with the PV residing in *SVM\_NFS* on *cluster1*
4. Create Apache Container: deploys a web container using the persistent volume as the data repository on both hosts defined in the Ansible group *prod*
5. Customize the Web Server: copy the web server content (the landing page *index.html*) to the web servers’ persistent volume
6. Create a Snapshot backup of the Web Server Content: takes a Snapshot of the ONTAP volume backing the Docker persistent volume (i.e. backing up the web server content created in task 5)

To run this playbook, issue the following commands:

[root@rhel1 ~]# **cd ~/ansible/lod**  
[root@rhel1 ~]# **ansible-playbook deploy\_trident.yml**

After completion of the playbook, validate the deployment of the NetApp Trident plugin, the new persistent volume and deployed containers by executing the following Ansible ad-hoc commands:

[root@rhel1 ~]# **ansible prod –a "docker plugin ls"**[root@rhel1 ~]# **ansible prod -a "docker volume ls"**  
[root@rhel1 ~]# **ansible prod -a "docker container ls"**

with output verifying that both hosts have the Trident plugin installed, have access to the persistent volume pvol1, and have the Web Server container named *myweb1* running:

**Output for docker plugin ls  
rhel2 | CHANGED | rc=0 >>**

**ID NAME DESCRIPTION ENABLED**

**724b93843897 netapp:latest Trident - NetApp Docker Volume Plugin true**

**rhel1 | CHANGED | rc=0 >>**

**ID NAME DESCRIPTION ENABLED**

**40fc16185753 netapp:latest Trident - NetApp Docker Volume Plugin true  
  
Output for docker volume ls  
rhel2 | CHANGED | rc=0 >>**

**DRIVER VOLUME NAME**

**netapp:latest pvol1**

**rhel1 | CHANGED | rc=0 >>**

**DRIVER VOLUME NAME**

**netapp:latest pvol1  
  
Output for docker container ls  
rhel2 | CHANGED | rc=0 >>**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**52cbff4ab9fe httpd "httpd-foreground" About a minute ago Up About a minute 0.0.0.0:80->80/tcp myweb1**

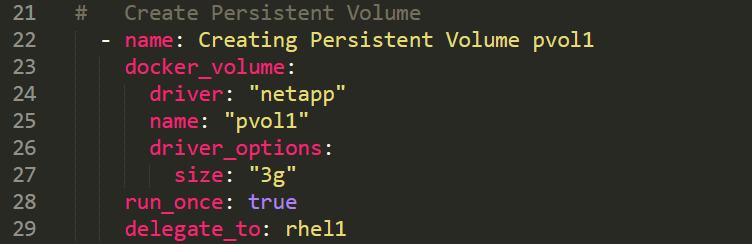
**rhel1 | CHANGED | rc=0 >>**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**516486c39266 httpd "httpd-foreground" About a minute ago Up About a minute 0.0.0.0:80->80/tcp myweb1**

**Note:** *the persistent volume pvol1 which appears as a Docker volume under rhel1 and rhel2 is the exact same persistent volume backed by the exact same FlexVol volume being mounted on both containers using NFS. The two containers (myweb1) on rhel1 and rhel2 are* ***not*** *the same container as is evident from the Container ID’s being different. They are two separate containers running on two different hosts (rhel1 and rhel2). However,* ***they share the same persistent volume, which means they both show the exact same web content.***

Let’s look at Task 3 of this playbook in more detail (*Task: Create Persistent Volume*). This task calls the module docker\_volume to create a persistent volume of size 3 GB using the NetApp (Trident) driver and executes only on rhel1 (as indicated by the run\_once: and delegate\_to: commands below).



But looking at the output of docker volume ls (above) shows the persistent volume local to Docker running on both rhel1 and rhel2! The command was only issues on rhel1. To verify this, create another Docker volume pvol\_test by issuing the following Docker command on *rhel1*:

[root@rhel1 ~]# **docker volume create -d netapp --name pvol\_test**

Looking at the Docker volumes on both hosts:

[root@rhel1 ~]# **ansible prod -a "docker volume ls"**

shows this volume present to Docker running on both rhel1 and rhel2 – the same behavior we saw earlier by running the playbook.

**rhel1 | CHANGED | rc=0 >>**

**DRIVER VOLUME NAME**

**netapp:latest pvol1**

**netapp:latest pvol\_test**

**rhel2 | CHANGED | rc=0 >>**

**DRIVER VOLUME NAME**

**netapp:latest pvol1**

**netapp:latest pvol\_test**

This behavior is expected and a **significant differentiator** that NetApp provides to our customers. Since Trident was installed on both hosts using the same config file *\_trident\_config\_file.json* (see below), Docker on both hosts will use any NetApp volume with prefix “docker\_” in SVM\_NFS as a persistent volume from the SVM serving NFS volumes.

{

"version": 1,

"storageDriverName": "ontap-nas",

"storagePrefix": "docker\_",

"managementLIF": "192.168.0.101",

"dataLIF": "192.168.0.145",

"svm": "SVM\_NFS",

"username": "admin",

"password": "Netapp1!",

"defaults": {

"size": "10G",

"spaceReserve": "none",

"exportPolicy": "DataPolicy",

"snapshotReserve": "0",

"snapshotDir": "true"

}

}

It is therefore best practices to have a dedicated SVM for all Trident volumes. And if one wants to differentiate between different volumes and/or SVM for different hosts (larger environments with say WEB and DATABASE servers mapping to different SVMs) one can have different Trident configuration files per host.

To recap, the playbook created two identical web containers running on *rel1* and *rhel2* respectively, both storing their web data on a persistent volume that resides on NetApp. In a real-world example, one would now have a load-balancer in front of the two web containers which will balance web access between the containers. Typically, this is accomplished by using a container orchestrator like Kubernetes. Kubernetes will deploy the web application across several pods running on different hosts, and expose the public IP as a service through a load balancer.

Open a web browser and go to the following URLs: <http://rhel1> and <http://rhel2> and observe the new web container running on both nodes, pointing to the same persistent volume residing on *cluster1*. It should look like this:

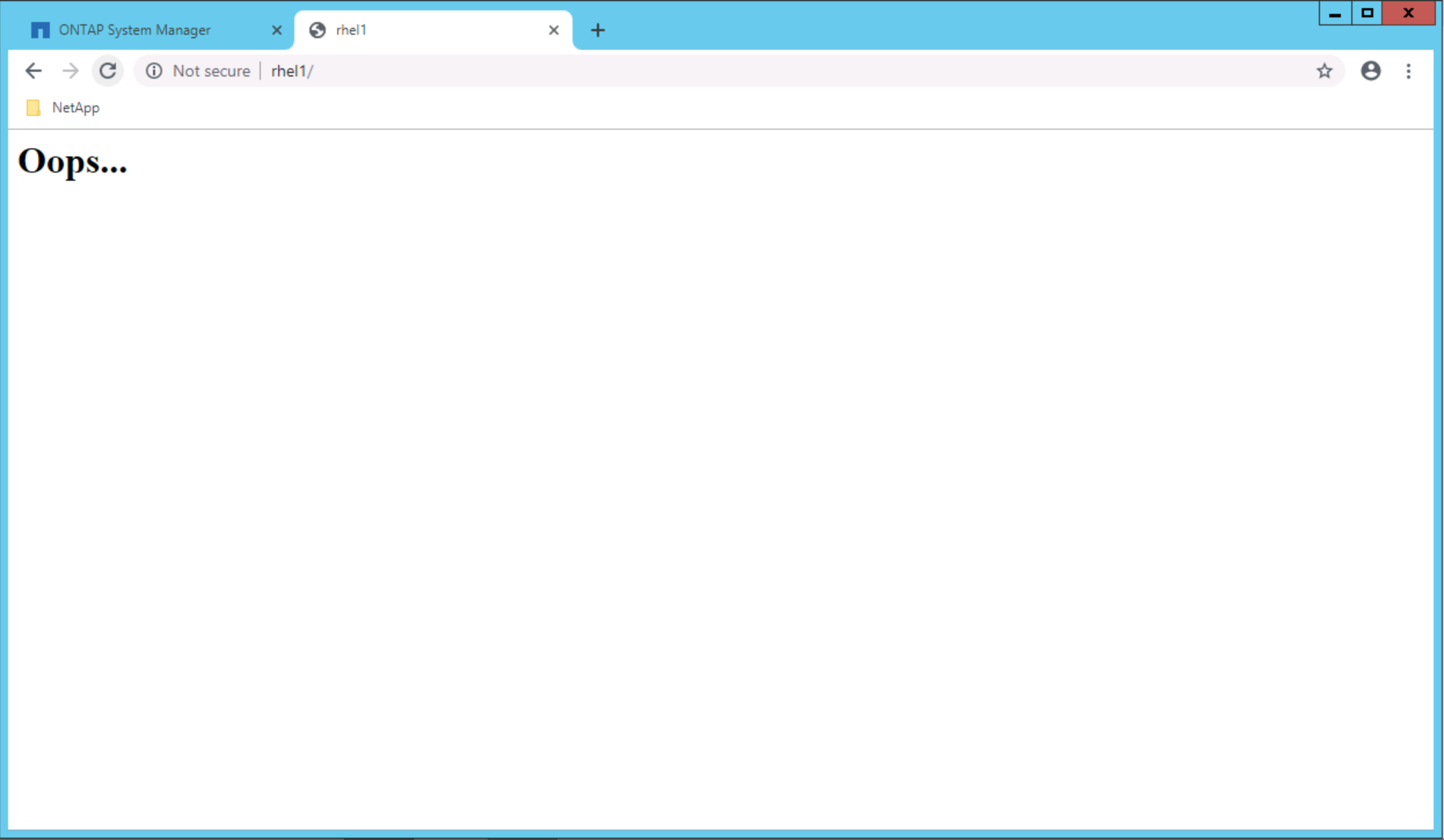


Next, we will quickly demonstrate the power of NetApp Trident by introducing data corruption to the web container and perform a simple recovery. Follow the steps below:

1. Type the following command to introduce data corruption into the container:

[root@rhel1 ~]# **docker cp ~/ansible/lod/oops.html myweb1:/usr/local/apache2/htdocs/index.html**

1. Refresh the browser display and show the content of our web server changed. It should look like this:



1. Next, we will recover the data by going into the container (by executing a shell command against the container):

[root@rhel1 ~]# **docker exec -it myweb1 /bin/bash**

Note that you are now in a shell on the webserver *myweb1*

1. From within the shell, and demonstrating the snapshot integration with containers for NFS-based persistent volumes. Recover the data by typing the following commands:

root@bca91d75d905:/usr/local/apache2# **cd htdocs**root@bca91d75d905:/usr/local/apache2# **ls -la**root@bca91d75d905:/usr/local/apache2# **cat index.html**

This shows that the index.html file on the life filesystem is corrupt.

1. As a next step let’s look at the contents of the .snapshot folder. This folder contains the single snapshot that was taken as part of the playbook and resides on the persistent volume in the NetApp SVM *SVM\_NFS*. Looking at the contents of this snapshot reveals the index.html file that’s not corrupt:

root@bca91d75d905:/usr/local/apache2# **cd .snapshot**  
root@bca91d75d905:/usr/local/apache2# **ls -la**  
root@bca91d75d905:/usr/local/apache2# **cd backup1**  
root@bca91d75d905:/usr/local/apache2# **ls -la**  
root@bca91d75d905:/usr/local/apache2# **cat index.html**

1. Recover this snapshot now by copying it over the corrupt one, and exit the shell:

root@bca91d75d905:/usr/local/apache2# **cp index.html ../../**  
root@bca91d75d905:/usr/local/apache2# **exit**

1. Refresh the browser display and show that the data was successfully recovered and looks again like this:



# Ansible Roles

Roles in Ansible are the next level of abstraction of Ansible playbooks. It allows for a much greater simplification and efficiency in documenting infrastructure as code by aggregating multiple steps and modules into a few steps, allowing for much greater quality, less likelihood for errors, and the opportunity to adhere to best practices. In enterprise environments that consists of 1,000 of compute, network and storage objects, using Ansible Roles makes sense.

Roles for NetApp Ansible modules are maintained by NetApp and needs to be downloaded from the NetApp GitHub repository. Simply put, a role combines multiple Ansible modules, playbooks and code into a single call, to which a set of variables is passed. As of this writing there are five ONTAP roles, two ONTAP Select roles and one Santricity role defined. It is therefore important to do a git pull frequently on the repository to ensure that the latest version of the existing roles, as well as any newly added roles are installed on the Ansible Control Node.

To download the NetApp specific Ansible roles onto the Ansible Control Node (*rhel1* in our example), do the following:

[root@rhel1 ~]# **cd ~/**

[root@rhel1 ~]# **git clone https://github.com/netapp/ansible /etc/ansible/roles**

This will clone the eight roles into the **/etc/ansible/roles** folder. Looking at this folder will lists all the roles (folders highlighted in green below):

[root@rhel1 ~]# **ls -la /etc/ansible/roles**

**total 8**

**drwxr-xr-x 14 root root 4096 Aug 30 13:20 .**

**drwxr-xr-x 3 root root 48 Aug 30 13:19 ..**

**drwxr-xr-x 2 root root 21 Aug 30 13:20 defaults**

**drwxr-xr-x 8 root root 152 Aug 30 13:20 .git**

**drwxr-xr-x 2 root root 21 Aug 30 13:20 meta**

**drwxr-xr-x 8 root root 112 Aug 30 13:20 na\_ontap\_cluster\_config**

**drwxr-xr-x 8 root root 112 Aug 30 13:20 na\_ontap\_nas\_create**

**drwxr-xr-x 8 root root 112 Aug 30 13:20 na\_ontap\_san\_create**

**drwxr-xr-x 8 root root 98 Aug 30 13:20 na\_ontap\_snapmirror\_create**

**drwxr-xr-x 8 root root 112 Aug 30 13:20 na\_ontap\_vserver\_create**

**drwxr-xr-x 6 root root 92 Aug 30 13:20 na\_ots\_cluster**

**drwxr-xr-x 5 root root 60 Aug 30 13:20 na\_ots\_deploy**

**drwxr-xr-x 8 root root 103 Aug 30 13:20 nar\_santricity\_host**

**-rw-r--r-- 1 root root 84 Aug 30 13:20 README.md**

**drwxr-xr-x 2 root root 21 Aug 30 13:20 tasks**

Each of the folders defining a role contains all the code and calls to Ansible modules will be executed when a role is called from a playbook. As an example, the tasks that the role *na\_ontap\_cluster\_config* will execute when called upon is defined by the information contained in the yml file */etc/ansible/roles/na\_ontap\_cluster\_config/tasks/main.yml.* Viewing this file (use the command more /etc/ansible/roles/na\_ontap\_cluster\_config/tasks/main.yml) looks very similar to the playbook discussed in the previous section.

The table below lists the five ONTAP roles, as well as what they do, and the Ansible modules they rely on. Explore on your own the Tasks contained in the ONTAP Select roles as well as the Santricity roles.

Table 1 NetApp defined ONTAP Roles and the Ansible Modules they call

|  |  |  |  |
| --- | --- | --- | --- |
| **ONTAP Roles** | **Role Description** | **Tasks Executed as part of Role** | **Ansible Modules Executed by the Tasks** |
| na\_ontap\_cluster\_config | Configuring an ONTAP Cluster | Assign Disks  Set Login Message  Setup DNS  Set NTP Server  Create SNMP community  Create Aggregates  Remove ports from Default broadcast domain  Modify Net Port  Create Interface Group  Modify Net Port  Create VLAN  Create Broadcast Domain  Create Intercluster LIF | na\_ontap\_license  na\_ontap\_disks  na\_ontap\_motd  na\_ontap\_dns  na\_ontap\_ntp  na\_ontap\_snmp  na\_ontap\_aggregate  na\_ontap\_broadcast\_domain\_ports  na\_ontap\_net\_port  na\_ontap\_net\_ifgrp  na\_ontap\_net\_port  na\_ontap\_net\_vlan  na\_ontap\_broadcast\_domain  na\_ontap\_interface |
| na\_ontap\_nas\_create | Role for Creating NFS and CIFS Shares | Create Policy  Setup Rules  Create Volume  Create Share | na\_ontap\_export\_policy  na\_ontap\_export\_policy\_rule  na\_ontap\_volume  na\_ontap\_cifs |
| na\_ontap\_san\_create | Role for Creating LUNs | Create iGroup  Create Volume  Create LUN  Create LUN Mapping | na\_ontap\_igroup  na\_ontap\_volume  na\_ontap\_lun  na\_ontap\_lun\_map |
| na\_ontap\_snapmirror\_create | Create SnapMirror Relationship | Create Cluster Peer  Create vserver Peer  Validate FlexVol  Create SnapMirror | na\_ontap\_cluster\_peer  na\_ontap\_vserver\_peer  na\_ontap\_volume  na\_ontap\_snapmirror |
| na\_ontap\_vserver\_create | Create one or more SVMs | Create vserver  Setup FCP  Setup iSCSI  Modify Adapter  Create interface  Add Default Route  Create DNS  Create CIFS Server  Create NFS Server  Setup NFS Rule | na\_ontap\_svm  na\_ontap\_fcp  na\_ontap\_iscsi  na\_ontap\_ucadapter  na\_ontap\_interface  na\_ontap\_net\_routes  na\_ontap\_dns  na\_ontap\_cifs\_server  na\_ontap\_nfs  na\_ontap\_export\_policy\_rule |

## Configuring *cluster1* using the Ansible Playbook and NetApp Roles

The file *configure\_ontap\_roles.yml* (in the *ansible/lod* folder on *rhel1*)is an ansible playbook that consists of two plays: the first configures the ONTAP cluster *cluster1,* and the second to configure the two hosts in the group prod. The play to configure the cluster contains several tasks, but unlike in the previous example where each task that configured the ONTAP cluster called an Ansible module, in this case, some tasks calls Ansible modules, and some tasks import NetApp defined roles (which in turn calls Ansible modules). Looking at the file in more detail:



Figure 3 Playbook to configure the ONTAP Cluster using calls to NetApp defined roles

Here one can see the two plays - the first to configure the ONTAP Cluster, and the second to configure the NFS hosts and mount the NFS export. If one looks at the one task named Create SVMs, unlike the case in the earlier example where this task simply executed a single Ansible module, in this case this task imports a role, as seen below:



Figure 4 The Task from this playbook that creates a SVM

The role (*na\_ontap\_vserver\_create*) in turn calls on 10 Ansible modules, as seen in the Table 1 above. So one may look at this and think this is much more complicated than the earlier example where we simply executed a playbook that called the Ansible modules directly (shown in Figure 1 and Figure 2). The simplicity comes from the YAML file that contains the variables to be passed to this playbook. As soon as a playbook that uses rules is constructed, one can simply add entries to the control file *(\_var\_roles.yml)* that the playbook uses to create additional entities on the ONTAP cluster for instance.

This control file *(\_var\_roles.yml)* is the repository of all the non-default variables used during the configuration of the cluster. This file is easily readable and acts as a documentation of the infrastructure to deploy (also can be referred to as Infrastructure-as-code). Use the command more \_var\_roles.yml to look at its content. Below shows the subsection of this file that’s responsible for creating SVM’s:

vservers:

- { name: SVM\_NFS , aggr: n1\_aggr1, protocol: nfs }

- { name: SVM\_CIFS, aggr: n1\_aggr1, protocol: cifs }

- { name: SVM\_SAN , aggr: n2\_aggr1, protocol: iscsi }

So adding another SVM to the cluster is as easy as adding another entry under :vservers using the syntax above. So adding a single line to this file replaces calling the module in a playbook as was discussed and shown in Figure 2.

The playbook that uses roles will do exactly what the playbook in the previous example did, but more. It will also create additional CIFS and iSCSI SVMs as well as volumes and LIFs associated with these SVMs. Because of the idempotent nature of Ansible, it will know that the NFS SVM (and aggregates, LIFs, etc.) is already present and will skip those steps.

To now fully build out the cluster, execute this playbook by typing the following commands:

[root@rhel1 ~]# **cd ~/ansible/lod**  
[root@rhel1 ~]# **ansible-playbook configure\_ontap\_roles.yml**

***Note:*** *when prompt for domain admin, type* ***administrator****. When prompt for domain admin password, type* ***Netapp1!***

Use ONTAP System Manager to view the configured cluster. As you can see, using roles and a global variables file is a great way to document the roll out of your infrastructure.

In closing, the recommendation is to use the examples and playbooks defined in this document, modify those and execute them on the lab on demand instance to familiarize oneself with the power of Ansible. Many of our customers are using Ansible on a day to day, and are unaware of the fact that NetApp and Red Hat has such a tight relationship when it comes to Ansible and the modules NetApp writes for Ansible. It is our responsibility to educate customers on the integration that we have with Ansible.

Appendix A: Alternative Method to Installing Ansible

Since using the yum install command on a RHEL system requires an active registered subscription manager, it is possible that this method will not work for some of the LOD labs that do not have an active subscription. It is fairly easy to identify if that is the cause for failed installation. Simply run the yum command and observe the output. If it is similar to the one below, you will need to use an alternative method for the installation.

[root@rhel1 ~]# **yum**

Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager

This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.

To install Ansible in this case, follow the following steps:

1. Log into the Linux server you want to install Ansible on using PuTTY. In most LoD environments that will be *rhel1* with username *root* and password *Netapp1!*
2. Install pip by typing the following command:

[root@rhel1 ~]# **cd ~/**

[root@rhel1 ~]# **curl** [**https://bootstrap.pypa.io/get-pip.py -o get-pip.py**](https://bootstrap.pypa.io/get-pip.py%20-o%20get-pip.py)[root@rhel1 ~]# **python get-pip.py**  
[root@rhel1 ~]# **pip install ansible**

[root@rhel1 ~]# **mkdir /etc/ansible**

Appendix B: Configuration Files Used in the Demo

This section contains all the files provided as part of the Git repository.

* hosts

[prod]

rhel1

rhel2

* configure\_ontap\_playbook.yml

---

##[PLAY] Configure ONTAP Cluster

- hosts: localhost

name: Play [Configure ONTAP Cluster]

vars\_files:

- \_var\_playbook.yml

tasks:

# Create Aggregates

- name: Create Aggregates (na\_ontap\_aggregate)

na\_ontap\_aggregate:

state: "{{ state }}"

name: "{{ item.name }}"

nodes: "{{ item.node }}"

disk\_count: 13

raid\_size: 13

https: true

validate\_certs: false

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

with\_items:

- { name: "n1\_aggr1", node: "cluster1-01" }

- { name: "n2\_aggr1", node: "cluster1-02" }

- name: Waiting for Aggregates to come Online

wait\_for:

timeout: 5

# Create a SVM

- name: Create a SVM (na\_ontap\_svm)

na\_ontap\_svm:

state: "{{ state }}"

name: "{{ vserver }}"

root\_volume: "{{ vserver}}\_root"

root\_volume\_aggregate: "{{ aggr }}"

root\_volume\_security\_style: unix

aggr\_list: n1\_aggr1,n2\_aggr1

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

https: true

validate\_certs: false

# Create an Interface

- name: Create an Interface (na\_ontap\_interface)

na\_ontap\_interface:

state: "{{ state }}"

interface\_name: nfs\_data\_lif1

home\_port: e0d

home\_node: cluster1-01

role: data

protocols: nfs

address: "{{ data\_lif1 }}"

netmask: 255.255.255.0

vserver: "{{ vserver }}"

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

https: true

validate\_certs: false

# Create an Interface

- name: Create an Interface (na\_ontap\_interface)

na\_ontap\_interface:

state: "{{ state }}"

interface\_name: nfs\_data\_lif2

home\_port: e0d

home\_node: cluster1-02

role: data

protocols: nfs

address: "{{ data\_lif2 }}"

netmask: 255.255.255.0

vserver: "{{ vserver }}"

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

https: true

validate\_certs: false

# Configure NFS

- name: Configure NFS (na\_ontap\_nfs)

na\_ontap\_nfs:

state: "{{ state }}"

service\_state: started

vserver: "{{ vserver }}"

nfsv3: enabled

nfsv4: disabled

nfsv41: disabled

tcp: enabled

udp: enabled

vstorage\_state: disabled

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

https: true

validate\_certs: false

# Setup Default Rules

- name: Create default Export Policy Rule (na\_ontap\_export\_policy\_rule)

na\_ontap\_export\_policy\_rule:

state: "{{ state }}"

policy\_name: default

vserver: "{{ vserver }}"

client\_match: 0.0.0.0/0

ro\_rule: any

rw\_rule: none

super\_user\_security: none

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

https: true

validate\_certs: false

# Create Policy

- name: Create Export Policy (na\_ontap\_export\_policy)

na\_ontap\_export\_policy:

state: "{{ state }}"

name: "{{ policy }}"

vserver: "{{ vserver }}"

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

https: true

validate\_certs: false

# Setup Rules

- name: Create Export Policy Rule (na\_ontap\_export\_policy\_rule)

na\_ontap\_export\_policy\_rule:

state: "{{ state }}"

policy\_name: "{{ policy }}"

vserver: "{{ vserver }}"

client\_match: "{{ client }}"

ro\_rule: any

rw\_rule: any

super\_user\_security: any

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

https: true

validate\_certs: false

# Create Volume

- name: Create Volume (na\_ontap\_volume)

na\_ontap\_volume:

state: "{{ state }}"

name: "{{ volname }}"

aggregate\_name: "{{ aggr }}"

size: 10

size\_unit: gb

policy: "{{ policy }}"

junction\_path: "/{{ volname }}"

space\_guarantee: "none"

vserver: "{{ vserver }}"

hostname: "{{ hostname }}"

username: "{{ username }}"

password: "{{ password }}"

volume\_security\_style: unix

https: true

validate\_certs: false

##[PLAY] Mount NFS export to hosts in /etc/ansible/hosts

- hosts: prod

name: Play [Configure nfs on Linux Instances and Mount the nfs export]

vars\_files:

- \_var\_playbook.yml

tasks:

# Install NFS on Linux Instances

- name: Install nfs on all systems in /etc/ansible/hosts

package:

name: nfs-utils

state: present

# Mount ONTAP NFS export on all Linux Instances

- name: Mount nfs export on all systems in /etc/ansible/hosts

mount:

state: mounted

path: /mnt/tmp

src: "{{ data\_lif2 }}:/{{ volname}}"

fstype: nfs

* \_var\_playbook.yml

hostname: "192.168.0.101"

username: "admin"

volname: NfsDataVolume

vserver: SVM\_NFS

policy: DataPolicy

aggr: n1\_aggr1

client: 0.0.0.0/0

data\_lif1: 192.168.0.144

data\_lif2: 192.168.0.145

state: present

password: Netapp1!

* configure\_ontap\_roles.yml

---

##[PLAY] Configure ONTAP Cluster

- hosts: localhost

gather\_facts: no

vars\_prompt:

- name: admin\_user\_name

prompt: Domain Admin Username (used for CIFS configuration)

- name: admin\_password

prompt: Domain Admin Password (used for CIFS configuration)

vars:

input: &input

hostname: "{{ netapp\_hostname }}"

username: "{{ netapp\_username }}"

password: "{{ netapp\_password }}"

file: \_var\_roles.yml

vars\_files:

- "{{ file }}"

tasks:

# Gather facts about the ONTAP cluster

- name: Get ONTAP version

na\_ontap\_gather\_facts:

state: info

<<: \*input

https: true

validate\_certs: false

ontapi: 32

# Configure Cluster

- import\_role:

name: na\_ontap\_cluster\_config

vars:

<<: \*input

# Create SVMs

- import\_role:

name: na\_ontap\_vserver\_create

vars:

<<: \*input

when: vservers != None

# Create NAS volumes

- import\_role:

name: na\_ontap\_nas\_create

vars:

<<: \*input

when: nas != None

# Create SAN volumes and LUNs

- import\_role:

name: na\_ontap\_san\_create

vars:

<<: \*input

when: luns != None

##[PLAY] Mount NFS export to hosts in /etc/ansible/hosts

- hosts: prod

name: Play [Configure nfs on Linux Instances and Mount the nfs export]

vars\_files:

- \_var\_playbook.yml

tasks:

# Install NFS on Linux Instances

- name: Install nfs on all systems in /etc/ansible/hosts

package:

name: nfs-utils

state: present

# Mount ONTAP NFS export on all Linux Instances

- name: Mount nfs export on all systems in /etc/ansible/hosts

mount:

state: mounted

path: /mnt/tmp

src: "{{ data\_lif2 }}:/{{ volname}}"

fstype: nfs

* \_var\_roles.yml

cluster: cluster1

netapp\_hostname: 192.168.0.101

netapp\_username: admin

netapp\_password: Netapp1!

#license\_codes: XXX

disks:

- cluster1-01

- cluster1-02

#motd: "This cluster was configured using Ansible"

dns:

- { dns\_domains: demo.netapp.com, dns\_nameservers: 192.168.0.253 }

ntp:

- { server\_name: dc1.demo.netapp.com, version: auto }

snmp:

- { community\_name: public, access\_control: ro }

aggrs:

- { name: n1\_aggr1, node: cluster1-01, disk\_count: 13, max\_raid: 13 }

- { name: n2\_aggr1, node: cluster1-02, disk\_count: 13, max\_raid: 13 }

ports:

- { node: cluster1-01, port: }

vservers:

- { name: SVM\_NFS , aggr: n1\_aggr1, protocol: nfs }

- { name: SVM\_CIFS, aggr: n1\_aggr1, protocol: cifs }

- { name: SVM\_SAN , aggr: n2\_aggr1, protocol: iscsi }

vserver\_dns:

- { vserver: SVM\_CIFS, dns\_domains: demo.netapp.com, dns\_nameservers: 192.168.0.253 }

- { vserver: SVM\_NFS , dns\_domains: demo.netapp.com, dns\_nameservers: 192.168.0.253 }

lifs:

- { name: nfs\_data\_lif1 , vserver: SVM\_NFS , node: cluster1-01, port: e0d, protocol: nfs , address: 192.168.0.144, netmask: 255.255.255.0 }

- { name: nfs\_data\_lif2 , vserver: SVM\_NFS , node: cluster1-02, port: e0d, protocol: nfs , address: 192.168.0.145, netmask: 255.255.255.0 }

- { name: cifs\_data\_lif1, vserver: SVM\_CIFS, node: cluster1-01, port: e0d, protocol: cifs , address: 192.168.0.146, netmask: 255.255.255.0 }

- { name: cifs\_data\_lif2, vserver: SVM\_CIFS, node: cluster1-02, port: e0d, protocol: cifs , address: 192.168.0.147, netmask: 255.255.255.0 }

- { name: san\_data\_lif1 , vserver: SVM\_SAN , node: cluster1-01, port: e0d, protocol: iscsi, address: 192.168.0.148, netmask: 255.255.255.0 }

- { name: san\_data\_lif2 , vserver: SVM\_SAN , node: cluster1-02, port: e0d, protocol: iscsi, address: 192.168.0.149, netmask: 255.255.255.0 }

cifs:

- { vserver: SVM\_CIFS, cifs\_server\_name: netapp1, domain: demo.netapp.com, force: true }

nas:

- { name: CifsDataVolume, protocol: cifs, vserver: SVM\_CIFS, share: smbdata , aggr: n1\_aggr1, size: 10 }

igroups:

- { name: rhel1\_igroup, vserver: SVM\_SAN, group\_type: iscsi, ostype: linux, initiator: "iqn.1994-05.com.redhat:rhel1.demo.netapp.com" }

luns:

- { name: lun1, vol\_name: SanDataVolume, vserver: SVM\_SAN, size: 10, aggr: n2\_aggr1, ostype: linux, space\_reserve: false, igroup: rhel1\_igroup }

* deploy\_trident.yml

---

##[PLAY] Deploy Trident, Install and Customize Apache Container and Create a Snapshot Backup

- hosts: prod

name: Play [Deploy Trident, Install and Customize Apache Container and Create a Snapshot Backup]

tasks:

# Configure Trident

- name: Copy config.json file to both rhel1 and rhel2

copy:

src: \_trident\_config\_file.json

dest: /etc/netappdvp/config.json

backup: yes

# Install Trident

- name: Install Trident on hosts rhel1 and rhel2

shell: docker plugin install --grant-all-permissions --alias netapp netapp/trident-plugin:latest

ignore\_errors: yes

# Create Persistent Volume

- name: Creating Persistent Volume pvol1

docker\_volume:

driver: "netapp"

name: "pvol1"

driver\_options:

size: "3g"

run\_once: true

delegate\_to: rhel1

# Create Apache Container

- name: Create the Apache Container myweb1 on rhel1 and rhel2 using the Persistent Volume pvol1 for its data repository

docker\_container:

name: myweb1

image: httpd

ports:

- "80:80"

volumes:

- "pvol1:/usr/local/apache2/htdocs"

# Customize the Web Server

- name: Customize the Web Server content

shell: docker cp ~/ansible/lod/htdocs/ myweb1:/usr/local/apache2/

run\_once: true

delegate\_to: rhel1

# Create a Snapshot backup of the Web Server Content

- name: Create the Snapshot backup1 on the NetApp volume

na\_ontap\_snapshot:

snapshot: backup1

volume: docker\_pvol1

vserver: SVM\_NFS

username: admin

password: Netapp1!

hostname: 192.168.0.101

https: true

validate\_certs: false

run\_once: true

delegate\_to: localhost

* \_trident\_config\_file.json

{

"version": 1,

"storageDriverName": "ontap-nas",

"storagePrefix": "docker\_",

"managementLIF": "192.168.0.101",

"dataLIF": "192.168.0.145",

"svm": "SVM\_NFS",

"username": "admin",

"password": "Netapp1!",

"defaults": {

"size": "10G",

"spaceReserve": "none",

"exportPolicy": "DataPolicy",

"snapshotReserve": "0",

"snapshotDir": "true"

}

}

* index.html

<html>

<head>

<title>Demo container using Ansible and NetApp Trident managed persistent volumes</title>

</head>

<body>

<p><img src="trident.png" width="100" height="100"><img src="netapp-logo.png" width="150" height="100"><img src="docker\_logo.png" width="100" height="100"></p>

<br>

<p>This Apache web server container was created by an <strong>Ansible playbook</strong>, and uses a persistent volume backed by <strong>NetApp</strong> using <strong>Trident</strong>.</p>

<p>Trident is a fully supported open source project maintained by <a href="http://www.netapp.com"">NetApp</a>. It has been designed from the ground up to help you meet the sophisticated persistence demands of your containerized applications.</p>

<p>Through its support for popular container platforms like <a href="https://kubernetes.io">Kubernetes</a> and <a href="http://docker.com">Docker</a>, Trident understands the natural and evolving languages of those platforms, and translates requirements expressed or implied through them into an automated and orchestrated response from the infrastructure.</p>

<p>Today, that infrastructure includes our <a href="https://www.netapp.com/us/products/data-management-software/ontap.aspx">ONTAP</a> (AFF/FAS/Select/Cloud), <a href="https://www.netapp.com/us/products/data-management-software/element-os.aspx">Element</a> (HCI/SolidFire), and <a href="https://www.netapp.com/us/products/data-management-software/santricity-os.aspx">SANtricity</a> (E/EF-Series) data management software, as well as the <a href="https://azure.microsoft.com/en-us/services/netapp/">Azure NetApp Files</a> service and the <a href="https://cloud.netapp.com/cloud-volumes-service-for-aws?utm\_source=GitHub&utm\_campaign=Trident">Cloud Volumes Service on AWS</a>.</p>

<p>That list continues to grow.</p>

<p>Detailed documentation for Trident can be found on <a href="https://netapp-trident.readthedocs.io">Read the Docs</a>.</p>

</body>

</html>

* oops.html

<html>

<body>

<h1>Oops...</h1>

</body>

</html>

Refer to the [Interoperability Matrix Tool (IMT)](http://support.netapp.com/matrix/mtx/login.do) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.



NetApp provides no representations or warranties regarding the accuracy, reliability, or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information in this document is distributed AS IS, and the use of this information or the implementation of any recommendations or techniques herein is a customer’s responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. This document and   
the information contained herein may be used solely in connection with the NetApp products discussed   
in this document.

© 2019 NetApp, Inc. All rights reserved. No portions of this document may be reproduced without prior written consent of NetApp, Inc. Specifications are subject to change without notice. NetApp, the NetApp logo, xxx, and xxx are trademarks or registered trademarks of NetApp, Inc. in the United States and/or other countries. <<Insert third-party trademark notices here.>> All other brands or products are trademarks or registered trademarks of their respective holders and should be treated as such. TR-XXXXi-MMYR