

Lab Guide

Overview

This guide presents the instructions and other information concerning the lab activities for this course. You will be using Ansible, CentOS, ONTAP, and ElementOS in this course and this guide will present the instructions for accessing and working through the prescribed exercises.

Outline

This guide includes these activities:

- Exercise 1: Configuring the lab environment
- Exercise 2: Configuring Ansible and Ad-Hoc commands
- Exercise 3: Intro to YAML and Playbooks
- Exercise 4: Intro to variables, conditions, loops, inclusions, tags, handlers, and facts
- Exercise 5: Intro to Roles
- Exercise 6: Open Lab. Real life tasks, with no hand holding.

Exercise 1: Configure the Lab Environment

In this exercise, you will learn to use Ansible. You will Configure the Ansible Control host and download the files necessary to manage NetApp ONTAP storage and NetApp ElementOS storage. You will also download classfiles necessary to complete the course.

Objectives


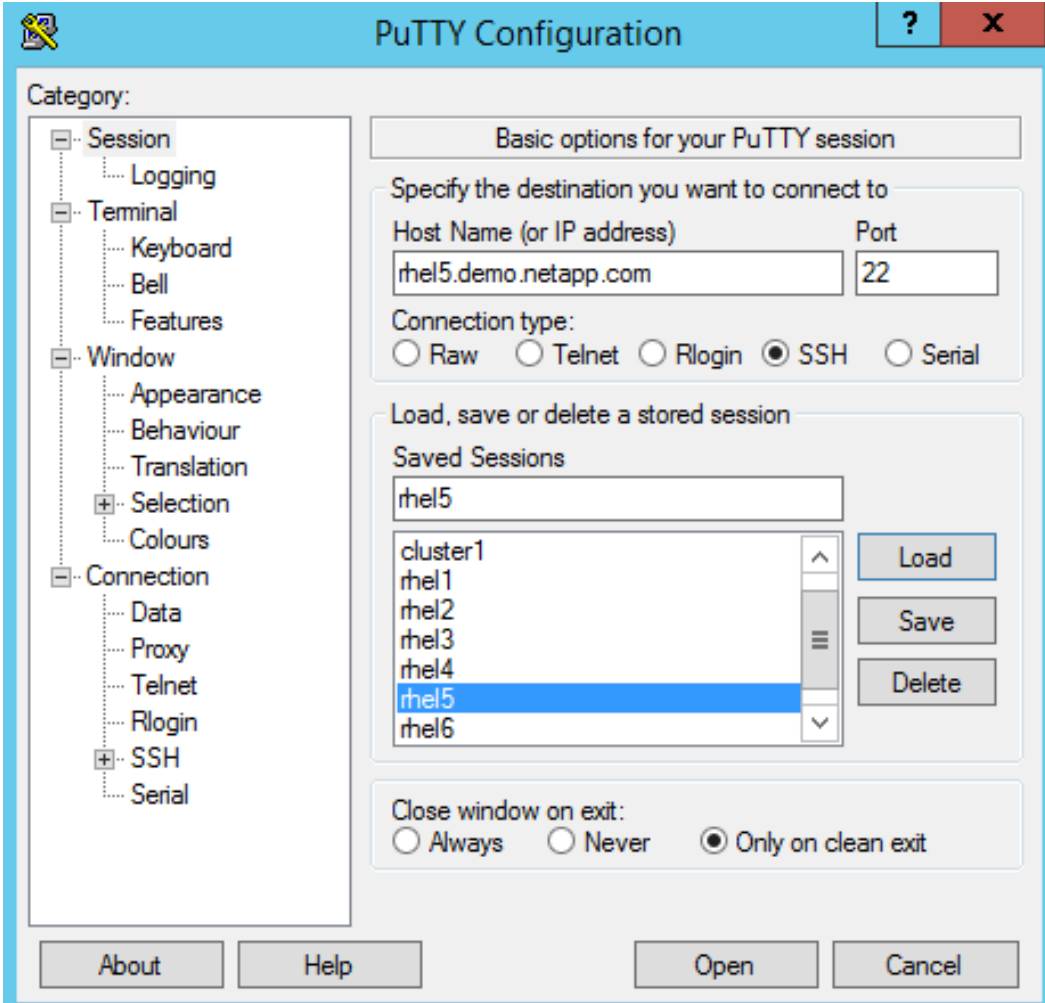
This exercise focuses on enabling you to do the following:

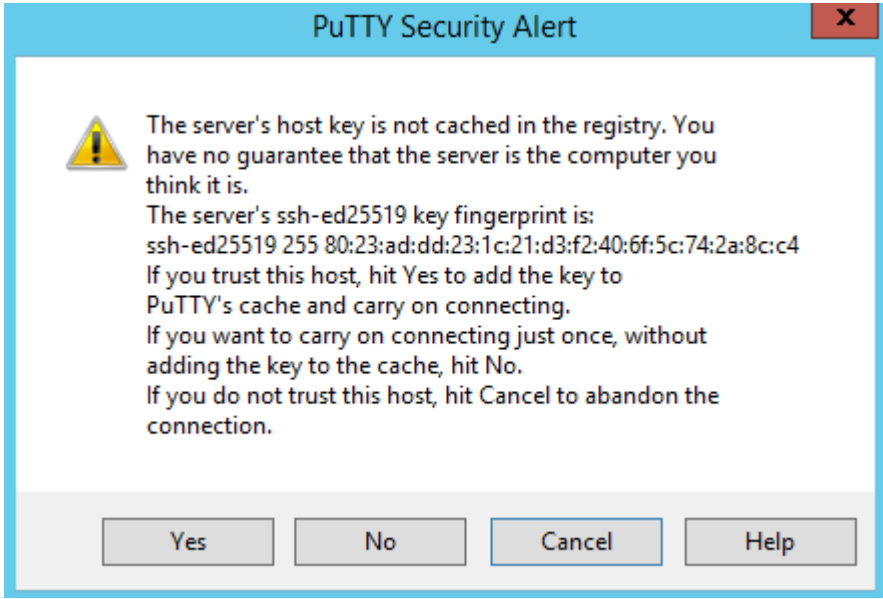
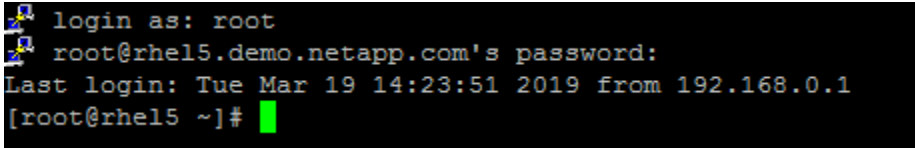
- Install Ansible on CentOS

Task 1: Intall Ansible on the Control Host

In this lab you work in a pre-configured lab environment. You will have access to the following hosts:

Role	Inventory name	IP	Username	Password
Windows DC	dc1	192.168.0.253	DEMOAdministrator	Netapp1!
Ansible Control Host	rhel5	192.168.0.66	root	Netapp1!
Managed Host 1	rhel1	192.168.0.61	root	Netapp1!
Managed Host 2	rhel2	192.168.0.62	root	Netapp1!
Managed Host 3	rhel3	192.168.0.63	root	Netapp1!
Managed Host 4	rhel4	192.168.0.64	root	Netapp1!
Managed Host 5	rhel6	192.168.0.69	root	Netapp1!
ONTAP Cluster 1	cluster1	192.168.0.101	admin	Netapp1!
ONTAP SVM 1	svm1	192.168.0.135		
ONTAP SVM 2	ansible_vserver	192.168.0.121		
ElementOS node	sf1	192.168.0.102	admin	Netapp1!

Step	Action
1-1	<p>On the taskbar of DC1, launch PuTTY.</p> 
1-2	<p>In the “Category:” window select Session.</p> <p>In the right pane, in the “Saved Sessions” list, scroll down and double-click rhel5.</p> 

Step	Action
1-3	<p>If Prompted with a PuTTY Security Alert click Yes.</p> 
1-4	<p>Log in using the username root, and the password Netapp1!.</p> 
1-5	<p>Ensure the Control Node is completely up to date by running the following command:</p> <pre>yum -y update</pre>
1-6	<p>Enable the EPEL repository to install the PIP Python Package Manager by using the following command:</p> <pre>yum install -y epel-release</pre>
1-7	<p>Install Python Pip</p> <pre>yum install -y python-pip</pre>

Step	Action
1-8	<p>Install git by using the following command</p> <pre>yum install -y git</pre>
1-9	<p>Update Pip</p> <pre>pip install -U pip</pre>
1-10	<p>Install Ansible via PIP</p> <pre>pip install ansible</pre>
1-11	<p>Install the netapp-lib python module for api connections to ONTAP and Solidfire using the following command:</p> <pre>pip install netapp-lib solidfire-sdk-python</pre>
1-12	<p>We will be passing credentials using SSH so we will need to install the sshpass module:</p> <pre>yum install -y sshpass</pre>
1-13	<p>Check Ansible has been configured correctly</p> <pre>ansible --version</pre> <pre> ansible 2.9.6 config file = None configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules'] ansible python module location = /usr/lib/python2.7/site-packages/ansible executable location = /usr/bin/ansible python version = 2.7.5 (default, Aug 7 2019, 00:51:29) [GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] </pre>

Step	Action
1-14	<p>Add the official Netapp ONTAP Collections for ONTAP and ElementSW (Solidfire) Collections became available in Ansible 2.9</p> <pre>ansible-galaxy collection install netapp.ontap -p /usr/share/ansible/collections</pre> <pre>ansible-galaxy collection install netapp.elementsw -p /usr/share/ansible/collections</pre> <p>Using collections will allow you to keep the NetApp, and other vendors, Ansible modules updated.</p> <p>To view all of the NetApp collections browse to https://galaxy.ansible.com/netapp.</p>
1-15	<p>Change the permissions to make the readable and executable by all users</p> <pre>chmod -R a+rx /usr/share/ansible/collections</pre>
1-16	<p>Now that Ansible is installed, clone the course Git repository. The repository contains the files used during this course.</p> <pre>git clone https://www.github.com/aenigmainc/ansible_workshop.git</pre>
1-17	<p>Prepare the lab environment by running the following commands:</p> <pre>cd ansible_workshop</pre> <pre>ansible-playbook setup.yml</pre>

End of Exercise

Exercise 2: Configuring Ansible and Ad-Hoc Commands

In this exercise, you will learn how to configure Ansible. You will also run configure a Windows computer as an Ansible target. Finally, you will get practice running ad-hoc commands.

Objectives

This exercise focuses on enabling you to do the following:

- Work with your inventory
- Understand the Ansible configuration file
- Ping a host using Ansible ping
- Configure a Windows client
- List modules and get module help
- Use the command and copy modules and set permissions
- Use the na_ontap_command ad-hoc command

Task 1: Work with your Inventory

In this task, you will examine an inventory file.

Step	Action
1-1	<p>To use the ansible command for host management, you need to provide an inventory file which defines a list of hosts to be managed from the control node. In this lab the inventory is provided for you. The inventory is an ini formatted file listing your hosts, sorted in groups, additionally providing some variables.</p> <p>To view the lab inventory, type the following:</p> <pre>cat /root/ansible_workshop/inventory</pre> <p>Review the structure of the inventory file. You will notice a few variables have been defined. A list of variables that can be defined in the inventory file is available here: https://docs.ansible.com/ansible/2.3/intro_inventory.html</p>
1-2	<p>To reference inventory hosts, you supply a host pattern to the ansible command. Ansible has a <code>--list-hosts</code> option which can be useful for clarifying which managed hosts are referenced by the host pattern in an ansible command.</p> <p>The most basic host pattern is the name for a single managed host listed in the inventory file. This specifies that the host will be the only one in the inventory file that will be acted upon by the ansible command. From the <code>ansible_workshop</code> directory, run:</p> <pre>ansible rhel4 --list-hosts</pre>

Step	Action
1-3	<p>An inventory file can contain a lot more information, it can organize your hosts in groups or define variables. In our example, the current inventory has the groups <code>linux_nodes</code> and <code>windows_nodes</code>. Run Ansible with these host patterns and observe the output:</p> <pre>ansible linux_nodes --list-hosts</pre> <pre>ansible linux_nodes,windows_nodes --list-hosts</pre> <pre>ansible 'lin*' --list-hosts</pre> <pre>ansible all --list-hosts</pre> <p>It is OK to put systems in more than one group. For instance, a server could be both a web server and a database server. Note that in Ansible the groups are not necessarily hierarchical.</p> <p>The inventory can contain more data. E.g. if you have hosts that run on non-standard SSH ports you can put the port number after the hostname with a colon. Or you could define names specific to Ansible and have them point to the “real” IP or hostname.</p>
1-4	Using vim create a new inventory file called <code>my_inventory</code> in the <code>ansible_workshop</code> directory
1-5	<p>In the <code>my_inventory</code> file create the myservers host group and add rhel2, rhel4, and rhel6 to the group. Create another host group called otherservers and add rhel1, rhel3, and rhel5 to the group.</p> <pre>[myservers] rhel2 rhel4 rhel6</pre> <pre>[otherservers] rhel1 rhel3 rhel5</pre>

Step	Action
1-6	<p>To use a non-standard inventory file you will use the -i option. Type the following command:</p> <pre data-bbox="261 281 1065 315">ansible -i my_inventory myservers --list-hosts</pre> <p>This will use the specified inventory file, my_inventory, and list the contents of the myservers host group.</p>
1-7	<p>Have Ansible list the hosts in the otherservers group of the my_inventory inventory file.</p>

Task 2: The Ansible Configuration Files

The behavior of Ansible can be customized by modifying settings in Ansible's ini-style configuration file. Ansible will select its configuration file from one of several possible locations on the control node, please refer to the Ansible documentation.

The recommended practice is to create an `ansible.cfg` file in the directory from which you run Ansible commands. This directory would also contain any files used by your Ansible project, such as the inventory and playbooks. Another recommended practice is to create a file `.ansible.cfg` in your home directory.

Step	Action
2-1	<p>In the lab environment provided to you an <code>.ansible.cfg</code> file has already been created and filled with the necessary details in the home directory of your user on the control node:</p> <pre>ls -la ~/.ansible.cfg</pre>
2-2	<p>View the contents of the Ansible configuration file:</p> <pre>cat ~/.ansible.cfg</pre> <p>There are multiple configuration flags provided. Most of them are not of interest here, but make sure to note the last line: there the location of the inventory is provided. That is the way Ansible knew in the previous tasks what machines to connect to when you didn't specify the inventory file.</p>
2-3	<p>Activate and configure privilege escalation by creating a file called <code>ansible.cfg</code> in the <code>ansible_workshop</code> directory and adding the following information.</p> <pre>[privilege_escalation] become=True become_method=sudo become_user=root become_ask_pass=true</pre> <p><code>become=True</code>: Enables privilege escalation <code>become_method=sudo</code>: Sets privilege escalation to use sudo <code>become_user=root</code>: Sets the user to become during privilege escalation <code>become_ask_pass=true</code>: Enables password prompting during privilege escalation</p>


Step	Action
2-4	<p>If you have multiple Ansible configuration files they are processed in the following order:</p> <ol style="list-style-type: none">1: <code>ANSIBLE_CONFIG</code> (an environment variable)2: <code>ansible.cfg</code> (in the current directory)3: <code>.ansible.cfg</code> (in the home directory)4: <code>/etc/ansible/ansible.cfg</code> <p>The impact of having multiple Ansible configuration files is addressed in the next task.</p>

Task 3: Ping a Host

Let's start with something really basic - pinging a host. To do that we use the Ansible `ping` module. The `ping` module makes sure our target hosts are responsive. Basically, it connects to the managed host, executes a small script there and collects the results. This ensures that the managed host is reachable and that Ansible is able to execute commands properly on it.

Think of a module as a tool which is designed to accomplish a specific task.

Step	Action
3-1	<p>Ansible needs to know that it should use the <code>ping</code> module: The <code>-m</code> option defines which Ansible module to use. Options can be passed to the specified module using the <code>-a</code> option. Type the following command.</p> <pre>ansible linux_nodes -m ping</pre> <p>Use Netapp1! As the BECOME password.</p> <p>As you see each node reports the successful execution and the actual result - here "pong".</p>
3-2	<p>The error message states no inventory was parsed. This means it couldn't find an inventory file. This is happening because Ansible is using the <code>ansible.cfg</code> file in the current working directory, not the <code>.ansible.cfg</code> from the home directory, and there isn't a path to the inventory defined in the working directory configuration file. When troubleshooting it is important to understand which files are being used during execution.</p>
3-3	<p>There are two ways we could fix this, we could use the <code>-i</code> option when running the ansible command, or we could just rename the <code>ansible.cfg</code> file so it won't be used during execution. Type the following command while in the <code>ansible_workshop</code> directory.</p> <pre>mv ansible.cfg to myansible.cfg</pre> <p>Renaming the file will prevent it from being used for the rest of the class.</p>
3-4	<p>Re-run the following command.</p> <pre>ansible linux_nodes -m ping</pre> <p>As you see each node reports the successful execution and the actual result - here "pong".</p> <p>You will notice it didn't prompt us for a BECOME password as we don't have privilege escalation defined in the <code>.ansible.cfg</code> file.</p>

Step	Action
3-5	<p>Try the same thing with a windows server:</p> <pre>ansible windows_nodes -m ping</pre> <p>Notice the error message. Ansible requires additional configuration to work with Windows.</p>
3-6	<p>On the control host install WinRM to allow Ansible to communicate with Windows.</p> <pre>pip install pywinrm</pre>
3-7	<p>The windows server needs to be configured to allow Ansible to talk to it. To do this we will run a PowerShell script that downloads the latest configuration from github, configures the execution policy and runs the configuration.</p> <p>Perform the following actions on DC1.</p> <p>Open PowerShell by clicking the PowerShell icon in the taskbar.</p>  <p>Run the following commands at the PowerShell prompt.</p> <pre>cd C:\Course_Files</pre> <pre>unblock-file .\ConfigureRemotingForAnsible.ps1</pre> <pre>Set-ExecutionPolicy Unrestricted (answer Y if prompted)</pre> <pre>.\ConfigureRemotingForAnsible.ps1 (answer R if prompted)</pre> <p>Minimize the PowerShell prompt.</p> <p>The directions for configuring a windows host for ansible use are here: https://docs.ansible.com/ansible/latest/user_guide/windows_setup.html</p>

Step	Action
3-8	<p>Try the windows server again:</p> <pre data-bbox="331 285 841 315">ansible windows_nodes -m ping</pre> <p>Notice the error message. The ping module doesn't work with Windows. For Windows you need to use <code>win_ping</code>, for network targets you need to use <code>net_ping</code>.</p>
3-9	<p>Try to ping the Windows server one more time using the correct module:</p> <pre data-bbox="331 590 911 619">ansible windows_nodes -m win_ping</pre> <p>We final get the successful execution and the actual result - here "pong".</p>

Task 4: Listing Modules and Getting Help

In this task, you will practice creating playbooks.

Host variables take precedence over group variables (more about precedence can be found in the docs).

Step	Action
4-1	<p>Ansible comes with a lot of modules by default. To list all modules run:</p> <pre>ansible-doc -l</pre> <p>In <code>ansible-doc</code> leave by pressing the button <code>q</code>. Use the up/down arrows to scroll through the content.</p>
4-2	<p>To find a particular set of modules, for example ONTAP, use the following command:</p> <pre>ansible-doc -l grep -i ontap</pre>
4-3	<p>Get help for a specific module including usage examples:</p> <pre>ansible-doc na_ontap_volume</pre> <p>Mandatory options are marked by a “=” in <code>ansible-doc</code>.</p>
4-4	<p>Many times the help in a module will tell you if there is an alternative. Type:</p> <pre>ansible-doc ping</pre> <p>In the description it tells you to use <code>win_ping</code> for Windows, and <code>net_ping</code> for network targets.</p>
4-5	<p>Look for the <code>-u</code> option in the output generated by the <code>ansible --help</code> command to verify the remote user configuration is set to the root remote user.</p> <pre>ansible --help grep -A1 -w -- -u</pre> <p>Examine the output and verify the default <code>remote_user</code> is set to root</p>
4-6	<p>Another place to get help is to browse the online Module Help</p> <p>https://docs.ansible.com/ansible/latest/modules/modules_by_category.html</p>

Step	Action
4-7	Examine the ~/.ansible.cfg file to verify the remote_user configuration.
4-8	<p>Using the yum module, install the latest available version of the httpd package by executing the following ad hoc command. Pass the package name and the desired state as arguments to the module in the form of a key/value pair list.</p> <pre>ansible rhel6 -m yum -a "name=httpd state=latest"</pre> <p>Examine the output. The status is CHANGED, the color is yellow, showing that httpd was installed. You can also read the output and see all of the supporting packages that were also installed.</p>
4-9	<p>Run the command again.</p> <pre>ansible rhel6 -m yum -a "name=httpd state=latest"</pre> <p>Examine the output. The status is now SUCCESS, and the color is now green. This shows there was no action taken because the result was as desired.</p>

Task 5: Use the Command Module

In this task, you will learn how to use modules. Modules are the executable plugins that get the real job done.

Usually modules can take “key=value” arguments and run in customized way depending up on the arguments themselves.

A module can be invoked from the command line or can be included in an Ansible playbook. You will practice with both methods in this task.

Step	Action
5-1	<p>Now let’s see how we can run a Linux command and format the output using the command module. It simply executes the specified command on a managed host:</p> <pre>ansible linux_nodes -m command -a "id"</pre> <p>In this case the module is called <code>command</code> and the option passed with <code>-a</code> is the actual command to run. Try to run this ad hoc command on all managed hosts using the <code>all</code> host pattern. *It will fail on ONTAP systems, windows systems, and ElementSW (SolidFire) systems.</p> <p>Notice we are authenticating as <code>root</code> as that is what is defined in the inventory file with the <code>ansible_user</code> variable.</p>
5-2	<p>Have a quick look at the kernel versions on the Linux hosts:</p> <pre>ansible linux_nodes -m command -a 'uname -r'</pre>
5-3	<p>Sometimes it’s desirable to have the output for a host on one line:</p> <pre>ansible linux_nodes -m command -a 'uname -r' -o</pre> <p>Like many Linux commands, <code>ansible</code> allows for long-form options as well as short-form. For example <code>ansible linux_nodes --module-name ping</code> is the same as running <code>ansible linux_nodes -m ping</code>. We are going to be using the short-form options throughout this workshop.</p>

Task 6: The Copy Module and Permissions

In this task, you use the copy module to copy information into a file. Normally copy is used to copy a file from one location to another, but you can also pass a string of information into a file.

Step	Action
6-1	<p>Using the <code>copy</code> module, execute an ad hoc command on <code>rhel4</code> to change the contents of the <code>/etc/motd</code> file. The content is handed to the module through an option in this case.</p> <p>Run the following, but expect an error if you can't write to the destination:</p> <pre>ansible rhel4 -m copy -a 'content="Managed by Ansible\n" dest=/etc/motd'</pre> <p>Notice the first line is “CHANGED”, and the font is yellow.</p> <p>The “changed” status is “true”.</p>
6-2	<p>If you do not have permission to write to the destination location it is a case for privilege escalation and the reason to have <code>sudo</code> properly configured. We need to instruct Ansible to use <code>sudo</code> to run the command as root by using the parameter <code>-b</code> (think “become”).</p> <p>Ansible will connect to the machines using your current user name (root in this case), just like SSH would. To override the remote user name, you could use the <code>-u</code> parameter.</p> <p>Change the command to use the <code>-b</code> parameter and run again:</p> <pre>ansible rhel4 -m copy -a 'content="Managed by Ansible\n" dest=/etc/motd' -b</pre> <p>Its ok to connect as different user because <code>sudo</code> is set up. Not just for the user in the <code>sudoers</code> file. Since we didn't define the privilege escalation globally in our Ansible Configuration file we can use it on a case by case basis by using the <code>-b</code> parameter.</p> <p>Notice the first line is “SUCCESS”, and the font is green.</p> <p>The “changed” status is “false”.</p> <p>This status change makes it a lot easier to spot changes and what ansible actually did</p>
6-3	<p>Use Ansible with the generic command module to check the content of the <code>motd</code> file:</p> <pre>ansible rhel4 -m command -a 'cat /etc/motd'</pre>

Task 7: The na_ontap_command ad-hoc

In this task, you will explore the NetApp na_ontap_command Ansible Modules.

Step	Action
7-1	<p>Using the na_ontap_command module, execute an ad hoc command against cluster1 to check the version of ONTAP.</p> <pre>ansible localhost -m na_ontap_command -a 'hostname=cluster1 username=admin password=Netapp1! https=true validate_certs=false command="version" '</pre> <p>Pay attention to the fact that the host is defined as localhost. This is because all communication to ONTAP happens from the host that Ansible is running on. Also, the na_ontap_command module is not idempotent and will always return a changed result.</p> <p>This module lets you pass any standard ONTAP command</p>

End of Exercise

Exercise 3: Playbooks

While Ansible ad hoc commands are useful for simple operations, they are not suited for complex configuration management or orchestration scenarios. For such use cases playbooks are the way to go.

Playbooks are files which describe the desired configurations or steps to implement on managed hosts. Playbooks can change lengthy, complex administrative tasks into easily repeatable routines with predictable and successful outcomes.

A playbook is where you can take some of those ad-hoc commands you just ran and put them into a repeatable set of plays and tasks.

A playbook can have multiple plays and a play can have one or multiple tasks. In a task a module is called, like the modules in the previous chapter. The goal of a play is to map a group of hosts. The goal of a task is to implement modules against those hosts.

Objectives

This exercise focuses on enabling you to do the following:

- Intro to YAML
- Create a playbook
- Execute a playbook
- Modify a playbook
- Run a playbook against multiple hosts

Task 1: Playbook Basics: Intro to YAML

Playbooks are text files written in YAML format and therefore need:

- to start with three dashes (---)
- proper indentation using spaces and not tabs!
- There are some important concepts:
- hosts: the managed hosts to perform the tasks on
- collections: List of collections being used as modules or roles.
- tasks: the operations to be performed by invoking Ansible modules and passing them the necessary options.
- become: privilege escalation in Playbooks, same as using -b in the ad hoc command. *Not used with NetApp modules

The ordering of the contents within a Playbook is important, because Ansible executes plays and tasks in the order they are presented.

A Playbook should be idempotent, so if a Playbook is run once to put the hosts in the correct state, it should be safe to run it a second time and it should make no further changes to the hosts.

Most Ansible modules are idempotent, so it is relatively easy to ensure this is true.

Step	Action																
1-1	<p>In the yaml directory perform the following action(s).</p> <p>Update the task_1.1.yml file to add a Vegetable - Carrot.</p>																
1-2	<p>In the yaml directory perform the following action(s).</p> <p>Update the task_1.2.yml file to add a list of Vegetables - Carrot, Tomato, Cucumber.</p>																
1-3	<p>In the yaml directory perform the following action(s).</p> <p>The task_1.3.yml file has been updated with nutrition information for Fruits.</p> <p>Similarly update the nutrition information for Vegetables. Use the below table for information.</p> <table><tr><th>Vegetable</th><th>Calories</th><th>Fat</th><th>Carbs</th></tr><tr><td>Carrot</td><td>25</td><td>0.1 g</td><td>6 g</td></tr><tr><td>Tomato</td><td>22</td><td>0.2 g</td><td>4.8 g</td></tr><tr><td>Cucumber</td><td>8</td><td>0.1 g</td><td>1.9 g</td></tr></table>	Vegetable	Calories	Fat	Carbs	Carrot	25	0.1 g	6 g	Tomato	22	0.2 g	4.8 g	Cucumber	8	0.1 g	1.9 g
Vegetable	Calories	Fat	Carbs														
Carrot	25	0.1 g	6 g														
Tomato	22	0.2 g	4.8 g														
Cucumber	8	0.1 g	1.9 g														

Step	Action								
1-4	<p>In the yaml directory perform the following action(s).</p> <p>Update the <code>task_1.4.yml</code> file using the information below.</p> <p>Jacob is a 30 year old Male working as a Systems Engineer at a firm.</p> <p>Represent Jacob's information (Name, Sex, Age, Title) in YAML format.</p> <p>Create a dictionary named Employee and define necessary properties.</p>								
1-5	<p>In the yaml directory perform the following action(s).</p> <p>Update the <code>task_1.5.yml</code> file using the information below.</p> <p>Update the YAML file to represent the Projects assigned to Jacob.</p> <p>Remember Jacob works on Multiple projects - Automation and Support. So remember to use a list.</p>								
1-6	<p>In the yaml directory perform the following action(s).</p> <p>Update the <code>task_1.6.yml</code> file to include Jacob's pay slips. Add a new property - Payslips and create a list of pay slip details. Each payslip detail contains Month and Wage. Use a list of dictionaries. Each item in the list must be a dictionary containing two properties - Month and Wage. Use values from the table below.</p> <table border="1"> <thead> <tr> <th>Month</th><th>Wage</th></tr> </thead> <tbody> <tr> <td>June</td><td>4000</td></tr> <tr> <td>July</td><td>4500</td></tr> <tr> <td>August</td><td>4000</td></tr> </tbody> </table>	Month	Wage	June	4000	July	4500	August	4000
Month	Wage								
June	4000								
July	4500								
August	4000								
1-7	<p>In the yaml directory perform the following action(s).</p> <p>In this task you will play with Anchors and Aliases.</p> <p>Open the <code>task_1.7.yml</code> file and answer the following questions:</p> <ol style="list-style-type: none"> 1. In the default how many stooges do we have, and what is/are their name(s). 2. In the default what is the URL for the stooges. 3. In development how many stooges are there, and what is/are their name(s). 4. In development what is the URL for the stooges. 5. In test how many stooges are there, and what is/are their name(s). 6. In test what is the URL for the stooges. 7. Modify the <code>task_1.7.yml</code> file so no pies get thrown in test. 								

Task 2: Creating a Directory Structure and File for your Playbook

In this lab you create a Playbook to set up a Volume for NFS export:

- First step: Create an Export Policy for the export
- Second step: Create an Export Policy Rule for the new Export Policy
- Third step: Create a Volume

This Playbook makes sure all the steps for getting a new NFS export created are completed.

There is a best practice on the preferred directory structures for playbooks. It is strongly encouraged to read and understand these practices as you develop your Ansible skills. That said, this playbook is very basic and creating a complex structure will just confuse things.

Instead, you are going to create a very simple directory structure for your playbook and add just a couple of files to it.

Step	Action
2-1	<p>On your control host, rhel5, use putty and create a directory called workshop in your home directory and change directories into it:</p> <pre>mkdir -p /root/ansible_workshop/workshop</pre> <pre>cd ~/ansible_workshop/workshop/</pre>
2-2	<p>Add a file called nfsexport.yml with the following content. Use vi/vim to create the file and add the following content.</p> <pre>--- - hosts: localhost collections: - netapp.ontap gather_facts: false name: Setup NFS Export</pre> <p>This shows one of Ansible's strengths: The Playbook syntax is easy to read and understand. In this Playbook:</p> <ul style="list-style-type: none">• The host to run the playbook against is defined via <code>hosts:</code>.• The NetApp ONTAP collection is defined as being used.• Ansible's default host fact gathering is disabled as we don't need it for ONTAP and this saves time.• A name is given for the play via <code>name:</code>.

Step	Action
2-3	<p>Now that you have defined the play, add a task to get something done. You will add a task in which ONTAP will ensure that an Export Policy called 'ansible_volume_policy'.</p> <p>Read the next step before editing the file</p> <p>Modify the file so that it looks like the following listing:</p> <pre> --- - hosts: localhost collections: - netapp.ontap gather_facts: false name: Setup NFS Export tasks: - name: Create Policy na_ontap_export_policy: state: present name: ansible_volume_policy vservers: ansible_vserver hostname: 192.168.0.101 username: admin password: Netapp1! https: true validate_certs: false </pre> <p>Save your playbook and exit your editor.</p>

Step	Action
2-4	<p>Since playbooks are written in YAML, alignment of the lines and keywords is crucial. Each section is two spaces in from the section it is a part of. Make sure to vertically align the <code>-</code> in the first run with the <code>task</code> in task. Once you are more familiar with Ansible, make sure to take some time and study a bit the YAML Syntax.</p> <p>In the added lines:</p> <ul style="list-style-type: none"> • We prepare the area where our tasks will go:. • A task is named and the module for the task is referenced. Here it uses the <code>na_ontap_export_policy</code> module. • Parameters for the module are added: • <code>state</code>: to define the wanted state of the policy • <code>name</code>: to identify the policy name • <code>vserver</code>: to identify the vserver this policy is part of • <code>hostname</code>: to identify ONTAP cluster to run this against • <code>username</code>: define the username to run this operation as • <code>password</code>: define the password for that username • <code>https</code>: configure the task to run over https instead of http • <code>validate_certs</code>: disable certification validation since we are using a self-signed certificate. <p>The module parameters are individual to each module. If in doubt, look them up again with <code>ansible-doc</code>.</p>

Task 3: Running the Playbook

In this task, you will edit and run playbooks you create.

Step	Action
3-1	<p>Playbooks are executed using the <code>ansible-playbook</code> command on the control node. Before you run a new Playbook it's a good idea to check for syntax errors:</p> <pre>ansible-playbook --syntax-check nfsexport.yml</pre> <p>If there are no returned errors the playbook has no syntax errors.</p>
3-2	<p>Now you should be ready to run your Playbook:</p> <pre>ansible-playbook nfsexport.yml</pre>
3-3	<p>The output should not report any errors but provide an overview of the tasks executed and a play recap summarizing what has been done. Use SSH to make sure the policy has been installed on cluster1.</p> <pre>ssh -l admin 192.168.0.101 export-policy show</pre> <p>The result should look similar to the example below</p> <pre> Vserver Policy Name ----- ansible_vserver ansible_volume_policy ansible_vserver default svml default 3 entries were displayed.</pre>
3-4	<p>Run the Playbook a second time and compare the output: The output changed from “changed” to “ok”, and the color changed from yellow to green. Also, the “PLAY RECAP” is different now. This make it easy to spot what Ansible actually did.</p> <pre>ansible-playbook nfsexport.yml</pre>

Task 4: Extend your Playbook: Create an Export Rule

In this task, you will practice editing and extending the playbook you created in the previous task.

Step	Action
4-1	The next part of the Playbook makes sure that a rule allowing access for the lab subnet is created within the Export Policy from the first task.

Step	Action
4-2	<p>Add the following information to the nfsexport.yml play.</p> <pre> - name: Setup rules na_ontap_export_policy_rule: state: present policy_name: ansible_volume_policy client_match: 192.168.0.0/24 ro_rule: sys rw_rule: sys super_user_security: sys vservers: ansible_vserver hostname: 192.168.0.20 username: admin password: Netapp1! https: true validate_certs: false - name: Create volume na_ontap_volume: state: present name: ansible_volume aggregate_name: aggr1 size: 10 size_unit: gb policy: ansible_volume_policy junction_path: /ansible_volume space_guarantee: "none" vservers: ansible_vserver hostname: 192.168.0.101 username: admin password: Netapp1! https: true validate_certs: false volume_security_style: unix </pre>

Step	Action
4-3	<p>You are getting used to the Playbook syntax, so what happens? The new task uses the <code>na_ontap_export_policy_rule</code> and <code>na_ontap_volume</code> modules.</p>
4-4	<p>Before you run the modified Playbook it's a good idea to check for syntax errors:</p> <pre>ansible-playbook --syntax-check nfsexport.yml</pre> <p>If there are no returned errors the playbook has no syntax errors.</p>
4-5	<p>Now you should be ready to run your Playbook:</p> <pre>ansible-playbook nfsexport.yml</pre> <p>Have a good look at the output</p> <p>Using ssh or an ad-hoc command, do a 'volume show' against the ONTAP system</p>
4-6	<p>In the <code>/root/ansible_workshop/workshop</code> directory create a new playbook called <code>rhel4_httpd.yml</code></p>
4-7	<p>Add the following:</p> <pre> --- - hosts: rhel4 name: install httpd tasks: - name: latest httpd version installed yum: state: latest name: httpd </pre> <p>This is the playbook equivalent of the ad-hoc command you ran in Exercise 2, task 4.</p>
4-8	<p>Run the <code>rhel4_httpd.yml</code> playbook again.</p> <p>Examine the output.</p> <p>Compare it to the previous execution.</p> <p>Notice the first time you ran it, the output for the task was yellow with a status of changed.</p> <p>The second time you ran the playbook, the task was green with a status of ok.</p>

Task 5: Practice: Apply to Multiple Host

In this task, you will learn how to apply a playbook to multiple hosts.

Step	Action
5-1	<p>The real power of Ansible is its ability to apply the same set of tasks reliably to many hosts.</p> <ul style="list-style-type: none">Doing this with ONTAP is possible but takes a little bit of thought and the use of a variable. <p>*Variables will be covered more in the next exercise</p> <p>Create a file in the <code>/root/ansible_workshop/workshop</code> called <code>motd.yml</code> and add the following:</p> <pre>--- - name: Set the Message of the Day hosts: svm collections: - netapp.ontap gather_facts: false tasks: - name: Verifying the MOTD na_ontap_motd: state: present motd_message: Set by Ansible vserver: "{{ shortname }}" hostname: "{{ inventory_hostname }}" username: admin password: Netapp1! https: true validate_certs: false connection: local</pre> <p>The vserver <code>{{ shortname }}</code> is coming from the inventory file and the <code>{{ inventory_hostname }}</code> is a special variable that is the name of the hostname as configured in Ansible's inventory host file.</p>
5-2	<p>Before you run the new Playbook it's a good idea to check for syntax errors:</p> <pre>ansible-playbook --syntax-check motd.yml</pre> <p>If there are no returned errors the playbook has no syntax errors.</p>

Step	Action
5-3	<p>Now you should be ready to run your Playbook:</p> <pre>ansible-playbook motd.yml</pre>
5-4	Setting your own variables will be covered in the next exercise
5-5	In the <code>/root/ansible_workshop/workshop</code> directory create a playbook called <code>install_httpd.yml</code>
5-6	<p>Add the following:</p> <pre> --- - hosts: rhel5,rhel6 name: install httpd tasks: - name: latest httpd version installed yum: state: latest name: httpd - pause: minutes=5 - name: start httpd service: name: httpd enabled: true state: started - name: copy test html page copy: content: "welcome to our group, leaving is not an option \n" dest: /var/www/html/index.html </pre> <p>The <code>content:</code> section word wraps in the example. Make sure its all one line.</p> <p>This playbook adds a pause between the installation of a service, and the starting of a service. Sometimes if you try to start something too soon after installing it, it fails, as the installation hasn't completed.</p>
5-7	Open a web browser and browse to http://rhel5 to verify the welcome page. Test rhel6 as well.

End of Exercise

Exercise 4: Working with Playbooks

Previous exercises showed you the basics of Ansible Engine. In the next few exercises, you are going to learn some more advanced Ansible skills that will add flexibility and power to playbooks.

Ansible exists to make tasks simple and repeatable. However, not all systems are exactly alike and often require some slight change to the way an Ansible playbook is run. Enter variables.

Ansible supports variables to store values that can be used in Playbooks. Variables can be defined in a variety of places and have a clear precedence. Ansible substitutes the variable with its value when a task is executed.

Variables are referenced in Playbooks by placing the variable name in quoted double curly braces:

Here comes a variable "{{ variable1 }}"

Variables and their values can be defined in various places: the inventory, additional files, on the command line, etc.

The recommended practice to provide variables in the inventory is to define them in files located in two directories named `host_vars` and `group_vars`:

- To define variables for a group “servers”, a YAML file named `group_vars/servers` with the variable definitions is created.
- To define variables specifically for a host `rhel5`, the file `host_vars/rhel5` with the variable definitions is created.

Host variables take precedence over group variables (more about precedence can be found in the docs).

Objectives

This exercise focuses on enabling you to do the following:

- Create variables in playbooks
- Create variables in external files
- Define variables at the command line
- Use YAML anchors and aliases in a playbook
- Conditional execution
- Simple loops
- Loops over hashes
- Using templates
- Managing facts
- Managing inclusions
- Implementing tags
- Implementing handlers

Task 1: Create Variables in Playbook

In the last exercise a playbook to create an NFS export and new Volume was used. The way the playbook was written makes it more difficult to update than needed, so you will be modifying it to contain variables.

Step	Action
1-1	<p>On the ansible control host, change into the 'workshop' directory</p> <pre>cd /root/ansible_workshop/workshop/</pre>

Step	Action
1-2	<p>open the nfsexport.yml file for editing</p> <p>Now add a 'vars' section before the tasks section so that it looks like this.</p> <pre> --- - hosts: localhost collections: - netapp.ontap gather_facts: false name: Setup NFS Export vars: hostname: 192.168.0.101 username: admin password: Netapp1! vserver: ansible_vserver volname: ansible_volume size: 10 client_match: 192.168.0.0/24 tasks: [...] </pre> <p>* The [...] just represents that there is more to this file that is not being shown.</p> <p>This allows you to use as variables, 'hostname', 'username', 'password', 'vserver', 'volname', 'size', 'client_match'.</p>

Step	Action
1-3	<p>Next step is to use those variables in the rest of the playbook.</p> <p>Replace the hard-coded entries for these options in the respective task sections. Here is the <code>na_export_policy</code> section.</p> <pre> - name: Create Policy na_ontap_export_policy: state: present name: "{{ volname }}_policy" vservers: "{{ vservers }}" hostname: "{{ hostname }}" username: "{{ username }}" password: "{{ password }}" https: true validate_certs: false </pre> <p>Pay attention to the 'name:' section replacement. You can use one or more variables and plain text to create naming conventions. Each variable has to be in its own set of <code>{{ }}</code>. For example <code>"{{ username }}:{{ password }}"</code>.</p> <p>Go ahead and replace the rest of the hard coded values with the appropriate variables.</p>
1-4	<p>Now you can save and exit, then run the playbook again and see that it doesn't change anything because you are using the same variables.</p> <pre> ansible-playbook nfsexport.yml </pre> <p>To see how easy changing variables are, change the volume name variable to something else and rerun the playbook. You may want to do a syntax check before you run it.</p>
1-5	<p>To see how easy changing variables are, change the volume name variable to something else (pick a name that is unique) and rerun the playbook.</p>

Task 2: Create Variable Files

In this task you will create an external file that contains the variables, then use those variables in a playbook.

Step	Action
2-1	<p>In addition to declaring variables directly in a playbook, you can also import variable files to a playbook.</p> <p>Now create a new file called <code>variables.yml</code> in <code>~/ansible_workshop/workshop/</code>:</p> <p>Variable files are simple YAML pair lists. Add your variables to the file so it looks like this.</p> <pre>hostname: 192.168.0.101 username: admin password: Netapp1! vserver: ansible_vserver volname: ansible_volume size: 10 client_match: 192.168.0.0/24</pre> <p>Save the file</p>
2-2	<p>With this file saved, you can replace the <code>'vars'</code> section of your <code>nfsexport_vars.yml</code> playbook with a <code>vars_files</code> section so it looks like this.</p> <pre>--- - hosts: localhost collections: - netapp.ontap name: Setup NFS Export vars_files: - variables.yml tasks: [...]</pre> <p>Now run the playbook again to verify those variables work.</p>
2-3	<p>Run the playbook again to see what happens if you try to create a volume that has the same name/configuration.</p>
2-4	<p>Modify the <code>variables.yml</code> file to create another uniquely named volume.</p> <p>Run the playbook again</p>

Task 3: Defining Variables and Variable Files at the Command Line

In this task, you will pass variables from the command line into a playbook, at run time..

Step	Action
3-1	<p>Variables can also be passed into a playbook from the command line, allowing for dynamic changing of variable values which can be used with automation setups.</p> <p>This is accomplished using the <code>--extra-vars</code> switch for <code>ansible-playbook</code>. The format is the following</p> <pre>ansible-playbook nfsexport.yml --extra-vars "volname=clivoll size=11 @variables.yml"</pre> <p>Notice the <code>volname</code> variable in the command line is the same name as the corresponding variable in the <code>variables.yml</code> file.</p>
3-2	<p>The defined variables at the command line override even the variable file listed in the command line. The important part to see is that <code><variable>=<value></code> and <code>@<path_to_file></code> is how you define variables or variable files at the command line.</p> <p>Try resizing your volume by passing in a larger 'size' at the command line with <code>extra-vars</code></p> <pre>ansible-playbook nfsexport.yml --extra-vars "size=15"</pre>

Task 4: YAML Anchors and Aliases: Applied

YAML anchors and aliases are a component of YAML not Ansible, but are very useful, especially with NetApp tasks. A YAML aliases allows any number of lines to be represented by a single line as many times as you wish.

There are 2 parts to this:

- The anchor '&' which defines a chunk of configuration

- The alias '*' used to refer to that chunk elsewhere

But what if you want essentially the same block of code with one small change?

You can use overrides with the characters '<<:' to add more values, or override existing ones.

Since the playbook that was created for NFSexport uses vserver, hostname, username, password, https, and validate_certs in each task, we can define this once at the top of the page and use it where it's needed. We covered a little theory behind Anchors and Aliases in Exercise 3. In this task you will apply them in a playbook.

Step	Action
------	--------

Step	Action
<ul style="list-style-type: none"> 	<p>This exercise will use 'ontap' as the alias name. A YAML alias section is filled out like any other YAML section with a two space indentation. Edit the vars section of <code>nfsexport.yml</code> to look like this.</p> <pre> --- - hosts: localhost collections: - netapp.ontap name: Setup NFS Export vars_files: variables.yml vars: ontap: &ontap vservers: "{{ vservers }}" hostname: "{{ hostname }}" username: "{{ username }}" password: "{{ password }}" https: true validate_certs: false tasks: [...] </pre> <p>Since the variable file is still being loaded, all those variables will exist for creating the alias.</p>
<ul style="list-style-type: none"> 	<pre> - name: Create Policy na_ontap_export_policy: state: present name: "{{ volname }}_policy" <<: *ontap </pre> <p>Rerun the playbook to see that everything still works. *If you resized your volume but have the old size in the playbook, it will show as changed for the volume as it is resized again.</p>

Task 5: Conditionals

Ansible can use conditionals to execute tasks or plays when certain conditions are met.

To implement a conditional, the `when` statement must be used, followed by the condition to test. The condition is expressed using one of the available operators like e.g. for comparison:

- `==` Compares two objects for equality.
- `!=` Compares two objects for inequality.
- `>` true if the left hand side is greater than the right hand side.
- `>=` true if the left hand side is greater or equal to the right hand side.
- `<` true if the left hand side is lower than the right hand side.
- `<=` true if the left hand side is lower or equal to the right hand side.

For more on this, please refer to the documentation: <http://jinja.pocoo.org/docs/2.10/templates/>

For example, creating a single playbook that can do a new NFS export or a new CIFS share.

Step	Action
5-1	<p>Now we edit <code>nfsexport.yml</code> and add a <code>when</code> statement for 'protocol' to the NFS specific sections.</p> <pre>[...] - name: Create Policy na_ontap_export_policy: state: present name: "{{ volname }}_policy" <<: *ontap when: protocol.lower() == 'nfs' - name: Setup rules na_ontap_export_policy_rule: state: present policy_name: "{{ volname }}_policy" client_match: "{{ client_match }}" ro_rule: sys rw_rule: sys super_user_security: sys <<: *ontap when: protocol.lower() == 'nfs'</pre> <p>The '<code>.lower()</code>' converts the protocol variable into all lowercase to make sure matching is easier.</p>

Step	Action
5-2	<p>Run the <code>nfsexport.yml</code> playbook</p> <p>It generates an error because the <code>protocol</code> variable hasn't been defined.</p>
5-3	<p>Add the variable to the <code>variables.yml</code> file and set it to 'nfs' for now.</p> <pre> hostname: 192.168.0.20 username: admin password: Netapp1! vserver: ansible_vserver volname: ansible_volume size: 10 client_match: 192.168.0.0/24 protocol: nfs </pre>
5-4	<p>Run the <code>nfsexport.yml</code> playbook. It should show all is OK as it will just verify what was created in the previous exercise.</p>
5-5	<p>Now a CIFS share section can be added to <code>nfsexport.yml</code>. This will use the 'na_ontap_cifs' module.</p> <pre> - name: Create CIFS share na_ontap_cifs: state: present share_name: "{{ volname }}" path: "/" + "{{ volname }}" <<: *ontap when: protocol.lower() == 'cifs' </pre> <p>Because of procedural playbook ordering, the CIFS share needs to be placed after the volume has been created.</p> <p>Change the <code>variables.yml</code> protocol entrance and set it to 'CIFS' and run the <code>nfsexport.yml</code> playbook again.</p> <p>You will get a failure because there isn't actually a CIFS server running by default in this lab, but you can see that the export policy steps are skipped and the CIFS share task was run.</p> <p>The <code>nfsexport.yml</code> playbook will not be negatively affected by the extra variable 'protocol'. Errors are only caused if you are calling a variable that is not defined somewhere, not if you define more variables than you use.</p>

Task 6: Simple Loops

Loops enable us to repeat the same task over and over again without having to call the same module more than once in a playbook. For example, let's say you want to create multiple volumes. By using an Ansible loop, you can do that in a single task. Loops can also iterate over more than just basic lists. For example, if you have a list of volumes with their corresponding group, loop can iterate over them as well. Find out more about loops in the Ansible Loops documentation.

Step	Action
------	--------

Step	Action
6-1	<p>To show the loops feature we will generate three new volumes. Put together all you have done so far and make a playbook from scratch. Create the file <code>loop_volumes.yml</code> in <code>~/ansible_workshop/workshop</code> on your ansible control node. We will use the <code>na_ontap_volume</code> module to generate the volumes.</p> <pre> --- - hosts: localhost collections: - netapp.ontap gather_facts: false name: Setup NFS Export vars_files: - variables.yml vars: ontap: &ontap vservers: "{{{ vservers }}}" hostname: "{{{ hostname }}}" username: "{{{ username }}}" password: "{{{ password }}}" https: true validate_certs: false tasks: - name: Create volume na_ontap_volume: state: present name: "{{{ item }}}" aggregate_name: aggr1 size: "{{{ size }}}" size_unit: gb junction_path: "/{{{ item }}}" space_guarantee: "none" <<: *ontap loop: - vol1 - vol2 - vol3 </pre>

Step	Action
6-2	<p>Save and run the playbook.</p> <p>Understand the playbook and the output:</p> <ul style="list-style-type: none"> • The names are not provided to the <code>na_ontap_volume</code> module directly. Instead, there is only a variable called <code>{{ item }}</code> for the parameter <code>name</code>. • The <code>loop</code> keyword lists the actual volume names. Those replace the <code>{{ item }}</code> during the actual execution of the playbook. • During execution the task is only listed once, but there are three changes listed underneath it.
6-3	Run the playbook again to see what happens if you try to create a volume that already exists.
6-4	Modify the <code>loop_volumes.yml</code> file to create 3 uniquely named volumes and run the playbook again.

Task 7: Loops over Hashes

In this task, you will learn how to perform a loop with a list of hashes.

Step	Action
7-1	<p>Loops can also be over lists of hashes. We can use this process to create the different volumes each with a different size. Modify the file <code>loop_volumes.yml</code></p> <pre>tasks: - name: Create volume na_ontap_volume: state: present name: "{{ item.name }}" aggregate_name: aggr1 size: "{{ item.size }}" size_unit: gb junction_path: "/" + item.name space_guarantee: "none" <<: *ontap loop: - { name: 'vol4', size: '10' } - { name: 'vol5', size: '20' } - { name: 'vol6', size: '15' }</pre> <p>This time the 'item' variable also has the additional variable defined in the hash of the loop. This loop has 'name' and 'size' as variables in it.</p> <p>Ansible uses Jinja2 templating to modify files before they are distributed to managed hosts. Jinja2 is one of the most used template engines for Python (http://jinja.pocoo.org/).</p>
7-2	<p>Save and run the <code>loop_volumes.yml</code>.</p>

Task 8: Using Templates in Playbooks

When a template for a file has been created, it can be deployed to the managed hosts using the `template` module, which supports the transfer of a local file from the control node to the managed hosts.

As an example of using templates you will change the `motd` file to contain host-specific data.

Step	Action
8-1	<p>First in the <code>~/ansible_workshop/workshop/</code> directory create the template file <code>motd-facts.j2</code>:</p> <pre>Welcome to {{ ansible_hostname }}. {{ ansible_distribution }} {{ ansible_distribution_version}} deployed on {{ ansible_architecture }} architecture.</pre> <p>The template file contains the basic text that will later be copied over. It also contains variables which will be replaced on the target machines individually.</p>
8-2	<p>Next we need a playbook to use this template. In the <code>~/ansible_workshop/workshop/</code> directory create the Playbook <code>motd-facts.yml</code>:</p> <pre>--- - name: Fill motd file with host data hosts: rhel4 become: yes tasks: - template: src: motd-facts.j2 dest: /etc/motd owner: root group: root mode: 0644</pre>
8-3	<p>Execute the Playbook <code>motd-facts.yml</code>.</p> <p>Login to <code>rhel4</code> via SSH or Putty and check the message of the day content.</p> <p>Log out of <code>rhel4</code>.</p> <p>You should see how Ansible replaces the variables with the facts it discovered from the system.</p>

Task 9: Managing Facts

In this task you will gather and view facts.

Step	Action
9-1	<p>The Ansible setup module retrieves facts from a system. Run the following command to retrieve facts from rhel4</p> <pre>ansible rhel4 -m setup</pre>
9-2	<p>Review the variables returned</p> <p>Look for the iSCSI IQN.</p>
9-3	<p>Run the following command</p> <pre>ansible rhel4 -m setup grep iscsi</pre>
9-4	<p>Now lets use the filter capability of Ansible</p> <pre>ansible rhel4 -m setup -a 'filter=ansible_iscsi*'</pre>
9-5	<p>We can also gather information from ONTAP systems.</p> <p>Type the following command:</p> <pre>ansible localhost -m na_ontap_info -a 'hostname=cluster1 username=admin password=Netappl! https=true validate_certs=false'</pre>
9-6	<p>Review the variables returned</p> <p>Look for the aggregate_info.</p> <p>It may have scrolled off the screen as the buffer may not be large enough</p>
9-7	<p>The na_ontap_info module doesn't have a filter option, but it does have an option to filter out what information is gathered. Run the following command:</p> <pre>ansible localhost -m na_ontap_info -a 'hostname=cluster1 username=admin password=Netappl! https=true validate_certs=false gather_subset=aggregate_info'</pre>

Task 10: Managing Inclusions

In this task you will create variables and tasks in separate files and include them in a playbook.

Step	Action
10-1	<p>In the ~/ansible_workshop/workshop directory run the following commands:</p> <pre>mkdir -p ./inclusions/tasks mkdir -p ./inclusions/vars</pre>
10-2	<p>Change directory into the ~/ansible_workshop/workshop/inclusions/tasks directory</p> <pre>cd ~/ansible_workshop/workshop/inclusions/tasks</pre>
10-3	<p>Create a file called env.yml and add the following information:</p> <pre>--- - name: install the {{ package }} package yum: name: "{{ package }}" state: latest - pause: minutes=5 - name: start the {{ service }} service service: name: "{{ service }}" state: "{{ svc_state }}" - name: create default web page copy: content: "{{ www_content }}" dest: /var/www/html/index.html</pre> <p>Save and close the file</p>
10-4	<p>Change directory into the ~/ansible_workshop/workshop/inclusions/vars directory</p> <pre>cd ~/ansible_workshop/workshop/inclusions/vars</pre>

Step	Action
10-5	<p>Create a file called vars.yml and add the following information:</p> <pre> package: httpd service: httpd svc_state: started www_content: "{{ ansible_fqdn }}" has been customized using Ansible on the {{ ansible_date_time.date }}\n" </pre> <p>Save and close the file</p>
10-6	<p>Change directory into the ~/ansible_workshop/workshop/inclusions/ directory</p> <pre>cd ~/ansible_workshop/workshop/inclusions/</pre>
10-7	<p>Create a file called playbook.yml and add the following information</p> <pre> --- - hosts: rhel1,rhel2 tasks: - name: include variables include_vars: vars/vars.yml - name: include environment file include: tasks/env.yml </pre> <p>Save and close the file</p>
10-8	<p>Run the playbook by using the following command</p> <pre>ansible-playbook playbook.yml</pre>
10-9	<p>On DC1 open a web browser and browse to http://rhel1. Verify the default web page contains the content we configured. Verify the default web page on rhel2</p>
10-10	<p>Where did we define the variable ansible_fqdn? Where did it come from?</p> <p>Where did we define the variable ansible_date_time? What did the .date index do?</p>

Task 11: Implementing Tags

In this task, you will create tag for part of a play and only run that part.

Step	Action
11-1	<p>Modify the file <code>~/ansible_workshop/workshop/inclusions/tasks/vars.yml</code> and add the following information:</p> <pre>package: httpd service: httpd svc_state: started www_content: "{{ ansible_default_ipv4.address }}" has been customized using Ansible on the {{ ansible_date_time.date }}\n"</pre> <p>Save and close the file</p>
11-2	<p>Modify the file <code>~/ansible_workshop/workshop/inclusions/tasks/env.yml</code> and add the tags option as shown below:</p> <pre>--- - name: install the {{ package }} package yum: name: "{{ package }}" state: latest - pause: minutes=5 - name: start the {{ service }} service service: name: "{{ service }}" state: "{{ svc_state }}" - name: create default web page copy: content: "{{ www_content }}" dest: /var/www/html/index.html tags: - html</pre> <p>Save and close the file</p>

Step	Action
11-3	<p>Modify the file <code>~/ansible_workshop/workshop/inclusions/playbook.yml</code> and add the following information</p> <pre> --- - hosts: rhel1,rhel2 var_files: vars/vars.yml tasks: - name: include environment file include: tasks/env.yml </pre> <p>Save and close the file</p>
11-4	<p>Run the playbook by using the following command</p> <pre> ansible-playbook playbook.yml --tags 'html' </pre> <p>Notice it doesn't run all of the tasks, it only copies the file.</p>
11-5	<p>On DC1 open a web browser and browse to http://rhel1. Verify the default web page contains the content we configured. Verify the default web page on rhel2.</p> <p>Notice it no longer uses the FQDN, instead its using the IP address.</p>

Task 12: Implementing Handlers

Sometimes when a task does make a change to the system, an additional task or tasks may need to be run. For example, a change to a service's configuration file may then require that the service be restarted so that the changed configuration takes effect.

Here Ansible's handlers come into play. Handlers can be seen as inactive tasks that only get triggered when explicitly invoked using the "notify" statement. Read more about them in the Ansible Handlers documentation.

None of the NetApp modules really have a point for handlers, but it's important to understand how they can be used.

Step	Action
12-1	<p>As an example, here is a playbook that:</p> <ul style="list-style-type: none">manages Apache's configuration file <code>httpd.conf</code> on all hosts in the web grouprestarts Apache when the file has changed <p>Create a playbook called <code>change_http_port.yml</code> in the <code>~/root/ansible_workshop</code> directory. Add the following to the playbook.</p> <pre>--- - name: manage httpd.conf hosts: rhell tasks: - name: Copy Apache configuration file copy: src: ansible_files/httpd.conf dest: /etc/httpd/conf/ notify: - restart_apache handlers: - name: restart_apache service: name: httpd state: restarted</pre> <ul style="list-style-type: none">The "notify" section calls the handler only when the copy task actually changes the file. That way the service is only restarted if needed - and not each time the playbook is run.The "handlers" section defines a task that is only run on notification.
12-2	Run the <code>change_http_port.yml</code> playbook.

Step	Action
12-3	Notice there is a new section called RUNNING HANDLER. This shows that handlers are being run.
12-4	Run the <code>change_http_port.yml</code> playbook again.
12-5	Notice the handlers didn't run this time. Since there were no changes being done, there is no reason to run the handler.
12-6	Verify the port changed for HTTP. Open a web browser on DC1 and connect to http://rhel1 . It should fail. Now connect to http://rhel1:8080 .
12-7	Modify <code>~/ansible_workshop/ansible_files/httpd.conf</code> so http will use port 80 again. you should see a section that says "Listen 8080" change it to be "Listen 80" Close and save the file.
12-8	Run the <code>change_http_port.yml</code> playbook again. You should see the handler run again since we are introducing a change.
12-9	Verify the port changed for HTTP. Open a web browser on DC1 and connect to http://rhel1 .

End of Exercise

Exercise 5: Roles

While it is possible to write a playbook in one file as we've done throughout this workshop, eventually you'll want to reuse files and start to organize things.

Ansible Roles are the way we do this. When you create a role, you deconstruct your playbook into parts and those parts sit in a directory structure.

Roles follow a defined directory structure; a role is named by the top-level directory. Some of the subdirectories contain YAML files, named `main.yml`. The files and templates subdirectories can contain objects referenced by the YAML files.

An example project structure could look like this, the name of the role would be "nas":

```
nas/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

The various `main.yml` files contain content depending on their location in the directory structure shown above. For instance, `vars/main.yml` references variables, `handlers/main.yml` describes handlers, and so on. Note that in contrast to playbooks, the `main.yml` files only contain the specific content and not additional playbook information like `hosts`, `become` or other keywords.

There are actually two directories for variables: `vars` and `default`: Default variables have the lowest precedence and usually contain default values set by the role authors and are often used when it is intended that their values will be overridden. Variables can be set in either `vars/main.yml` or `defaults/main.yml`, but not in both places.

Using roles in a Playbook is straight forward. The roles would be called as shown in the example shown.

```
---  
- name: launch roles  
  hosts: web  
  roles:  
    - role1  
    - role2
```

For each role, the tasks, handlers and variables of that role will be included in the Playbook, in that order. Any copy, script, template, or include tasks in the role can reference the relevant files, templates, or tasks without absolute or relative path names. Ansible will look for them in the role's files, templates, or tasks respectively, based on their use.

Objectives

This exercise focuses on enabling you to do the following:

- Create a basic role directory structure
- Create task files
- Test roles

Task 1: Create a Basic Role Directory Structure

Ansible looks for roles in a subdirectory called `roles` in the project directory. This can be overridden in the Ansible configuration. Each role has its own directory. To ease creation of a new role the tool `ansible-galaxy` can be used.

Step	Action
1-1	<p>Here is how to a role template. Run these commands in your <code>~/ansible_workshop/workshop</code> directory:</p> <pre>ansible-galaxy init --offline roles/roleexample</pre>
1-2	<p>To look at the role directories and their content we will use a command called <code>tree</code>. <code>Tree</code> is a recursive directory listing command that produces a depth indented listing of files. Install <code>tree</code> using the following command:</p> <pre>yum install -y tree</pre>
1-3	<p>view a recursive listing of the roles directory using the following command:</p> <pre>tree roles</pre>

Task 2: Create the Tasks File

In this task, you will create a task file consisting of the information in the `nfscreeate.yml` playbook.

Step	Action
2-1	<p>Create the <code>nfscreeate</code> role directory structure using the following command:</p> <pre>ansible-galaxy init --offline roles/nfscreeate</pre>
2-2	<p>The <code>main.yml</code> file in the <code>tasks</code> subdirectory of the role should do the following:</p> <ul style="list-style-type: none">• Create an export policy and export policy rule if <code>nfs</code> is specified• Create a volume• Create a CIFS share if <code>cifs</code> is specified <p>The <code>main.yml</code> (and other files possibly included by <code>main.yml</code>) can only contain tasks, not complete Playbooks! Also YAML alias don't work in roles files.</p>
2-3	<p>Copy the <code>nfsexport.yml</code> file into <code>roles/nfscreeate/tasks/main.yml</code> and edit it for the role.</p> <pre>cp nfsexport.yml roles/nfscreeate/tasks/main.yml</pre>

Step	Action
2-4	<p>Remove the hosts, gather_facts, vars_files, vars, tasks, headers and sections, and change the task calls to fully filled out, not using aliases (you can still use variables). Finally don't forget to delete the two extra spaces that are now in front of each line.</p> <pre> --- - name: Create Policy na_ontap_export_policy: state: present name: "{{ volname }}_policy" vservers: "{{ vservers }}" hostname: "{{ hostname }}" username: "{{ username }}" password: "{{ password }}" https: true validate_certs: false when: protocol.lower() == 'nfs' - name: Setup rules na_ontap_export_policy_rule: state: present policy_name: "{{ volname }}_policy" client_match: "{{ client_match }}" ro_rule: sys rw_rule: sys super_user_security: sys vservers: "{{ vservers }}" hostname: "{{ hostname }}" username: "{{ username }}" password: "{{ password }}" https: true validate_certs: false when: protocol.lower() == 'nfs' [...]</pre> <p>There isn't enough room in this step to show all of the tasks. Ensure the following tasks are present, Create Policy, Setup Rules, Create Volume, and Create CIFS share.</p> <p>Note that here just tasks were added. The details of a playbook are not present.</p>

Step	Action
2-5	<p>Since we will still be using variables, we need to have the role know what variables it will accept. Copy the <code>variable.yml</code> file to <code>roles/nfscreate/defaults/main.yml</code>.</p> <pre>cp ~/ansible-files/variables.yml roles/nfscreate/defaults/main.yml</pre> <p>That is all one command, it just word wrapped.</p> <p>This makes sure the variables exist in the role. You can remove the variable values from <code>main.yml</code> or leave them. You will be overriding them from the <code>variables.yml</code> file.</p>

Task 3: Test the Role

In this task, you will test the role you created in the previous task.

Step	Action
3-1	<p>You are ready to test the role. A playbook still needs to be created to call the role. Create the file <code>nfs_role.yml</code> in the directory <code>~/ansible_workshop/workshop</code> and add the following content.</p> <pre>--- - hosts: localhost collections: - netapp.ontap gather_facts: false vars_files: - variables.yml roles: - nfscreate</pre> <p>The role we are using isn't in a collection, but its modules are, so we still need to define a collection to use in the playbook.</p> <p>That's the whole playbook to call the role.</p>
3-2	<p>Now you are ready to run your playbook:</p> <pre>ansible-playbook nfs_role.yml</pre> <p>Experiment by changing some of the values <code>variable.yml</code> and run the <code>nfs_role.yml</code> playbook again. Remember that with 'protocol: cifs' it will always error on that step.</p>

End of Exercise

Exercise 6: Open Lab

In this exercise, you will practice creating playbooks and roles to manage NetApp storage arrays. These exercises contain almost no assistance, just directions. The goal is to take what you learned in previous exercises and use that information to complete the tasks. This exercise will be performed in the `~/ansible_workshop/ontap` directory. There are files to “help” you in the `~/ansible_workshop/ontap/help` directory.

Objectives

This exercise focuses on enabling you to do the following:

- Create a volume using a playbook
- Create a LIF using a playbook
- Create a playbook with multiple tasks
- Create a playbook with multiple tasks and anchors and aliases
- Create a complete SVM
- Mount an NFS export to a unix host
- Turn a playbook into a role
- Create an account and volume for Solidfire

Task 1: Create a Volume

In this task, you will create a playbook that creates a volume. This task will be performed in the `~/ansible_files/ontap` directory

Step	Action
1-1	Open the <code>create_volume.yml</code> playbook. Fix the playbook to create a 10GB volume that is mounted as <code>/<volume name></code> . Export the volume using the default export policy. Use variables where appropriate for the module parameters.
1-2	Use ssh, putty or system manager, to verify the volume was created and mounted correctly
1-3	Modify the <code>create_volume.yml</code> playbook so as to change the volume size to 20GB run the playbook and verify it worked correctly.
1-4	Modify the <code>create_volume.yml</code> playbook so as to delete the volume. run the playbook and verify it worked correctly.
1-5	
1-6	

Task 2: Create a LIF

In this task you will create a playbook that creates a LIF.

Step	Action
2-1	<p>Create a playbook, called <code>create_lif.yml</code>, that creates a new LIF for svm1: Use the following information to configure the LIF:</p> <ul style="list-style-type: none">■ Interface Name: <code><vserver name>_ansible_data_mgmt_lif1</code>■ Home port: <code>e0d</code>■ Home Node: <code>cluster1-01</code>■ Role: <code>Data</code>■ Protocols: <code>NFS</code>■ Failover policy: <code>local only</code>■ Firewall_policy: <code>data</code>■ Auto revert: <code>True</code> <p>IP Address: <code>192.168.0.142/24</code></p>
2-2	<p>Run the playbook, and verify it worked as intended.</p>
2-3	<p>Modify the <code>create_lif.yml</code> playbook to delete the <code><vserver>_ansible_data_mgmt_lif1</code> you just created.</p> <p>Run the playbook.</p> <p>On cluster1 validate the LIF was deleted.</p>

Task 3: Create a playbook with multiple tasks

In this task, you will create a playbook that creates both a volume and a LIF.

Step	Action
3-1	<p>Create a single playbook, called “new_vol_lif.yml” that creates a volume called ansiblevol1 and creates a LIF called <vserver>_ansible_data_mgmt_lif1. Use the information from the previous playbooks to assist you in the creation of this playbook.</p> <p>Store the variables for the hostname, username, password, vserver, aggr, volume name, and the IP Address for the new LIF in a variables file called “var_file.yml”.</p> <p>When creating multiple steps in a task order may matter. You can’t modify something that doesn’t exist. The playbook is a serialized operation.</p>
3-2	<p>Run the playbook.</p> <p>On cluster1 validate the volume and LIF were created.</p>
3-3	<p>Modify the “new_vol_lif.yml” playbook to remove newly created volume and LIF.</p> <p>Run the playbook.</p> <p>On cluster1 validate the volume and LIF were removed.</p>

Task 4: Create a playbook with multiple tasks and anchors and aliases

In the `new_vol_lif.yml` file you have some information that repeats from one step to the next. Specifically we had to reenter the hostname, username, password, https, and certificate information for each step. Even though we stored the data in a variable, we still have to specify the variable to use. In a playbook with only a couple steps this isn't a lot of work, but imagine a playbook with dozens or hundreds of steps, we would be repeating a lot of effort.

Luckily you learned a solution for this problem.

Step	Action
<ul style="list-style-type: none">•	<p>Modify the “<code>new_vol_lif.yml</code>” file to use an anchor and alias. Anchor the following information to <code>&login</code>:</p> <p>hostname, username, password, https, and validate_certs.</p> <p>The format should look similar to this:</p> <pre>vars: login: &login xxx: "{{ var1 }}" yyy: "{{ var2 }}" zzz: var3 vars_files: - /path/to/var_file.yml</pre> <p>Remember, if you are passing a variable you are going to use the curly braces, To call the anchor use the following syntax:</p> <pre><<: *login</pre>
<ul style="list-style-type: none">•	<p>Run the playbook.</p> <p>On cluster1 validate the volume and LIF were created.</p>
<ul style="list-style-type: none">•	<p>Modify the playbook to remove the volume and LIF.</p> <p>Run the playbook.</p> <p>On cluster1 validate the volume and LIF were deleted.</p>

Task 5: Create a complete SVM

Now that you have created a playbook with multiple steps that modifies an existing SVM, let's make a playbook that creates a whole NEW SVM, creates a single LIF that does both Data and Management, and creates both the SVM root volume and data volume in aggr1. We will also configure the root volume to be exported via NFS read only to everyone and the data volume exported via NFS ReadWrite to everyone.

Step	Action
5-1	<p>Use the following criteria for the new SVM:</p> <pre>hostname: 192.168.0.101 username: admin password: Netappl! vserver: ansibleSVM aggr: aggr1 vol_name: ansiblevol1 address: 192.168.0.142 data_policy: data_export_policy</pre> <p>Store the variables in a file called <code>svm_var_file.yml</code>.</p> <p>You will need to use the following modules to accomplish this, and remember, order matters.</p> <pre>na_ontap_svm na_ontap_interface na_ontap_nfs na_ontap_export_policy na_ontap_export_policy_rule (you will need to create two rules, one in the default policy and one in the data_export_policy)</pre> <p><code>na_ontap_volume</code> (hint: you only need to use this to create the data volumes. The SVM root volume creation is part of the SVM creation module).</p>
5-2	<p>Run the playbook.</p> <p>Verify the SVM was created.</p> <p>Verify the volume was created in the SVM.</p> <p>Verify the data volume is exported using the newly created policy.</p> <p>Verify the data LIF can be reached from DC1.</p>

Task 6: Mount an NFS export to a Unix Host

In this task, you will mount an NFS export to a Unix Host.

Step	Action
6-1	<p>Create a playbook called <code>rhel5_mnt.yml</code>.</p> <p>Use the Ansible <code>mount</code> module to mount the volume you exported in the previous task.</p> <p>Mount the <code>192.168.0.142:/ansiblevol1</code> to <code>/mnt/ansiblevol1</code> on <code>rhel5</code>.</p>

Task 7: Turn a playbook into a Role.

In this task, you will explore the NetApp Ansible Modules.

Step	Action
7-1	Turn the playbook you created in Task 5 into a Role Modify the SVM name to be ansibleSVM2 Modify the IP address to be 192.168.0.172

Task 8: Create an account and volume for Solidfire

In this task, you will use the skills learned in earlier tasks to create an account and volume on a SolidFire node.

Step	Action
8-1	<p>You are tasked to create a new volume on a solidfire node. The information you need to create the volume is here:</p> <ul style="list-style-type: none">• <code>hostname: sf1</code>• <code>username: admin</code>• <code>password: Netapp1!</code>• <code>volname: AnsibleVol</code>• <code>account: ansibleAccount</code>• <code>size_gb: 10</code>• <code>min: 1000</code>• <code>max: 2000</code>• <code>burst: 3000</code> <p>You will need to create an account first, then create the volume. The two modules you will need to use are <code>na_elementsw_account</code> and <code>na_elementsw_volume</code>.</p>

End of Exercise