Ansible Storage Automation Workshop

This content is a multi-purpose toolkit for effectively demonstrating Ansible's capabilities on storage equipment or providing informal workshop training in various forms -- instructor-led, hands-on or self-paced.

Presentation

Want the Presentation Deck? Its right here: Ansible storage Automation Workshop Deck (https://ansible.github.io/workshops/decks/ansible_storage.pdf)

Ansible storage Automation Exercises

- Exercise 1 - Exploring the lab environment (./1-explore/)
- Exercise 2 - Execute your first storage automation playbook (./2-first-playbook/)
- Exercise 3 - Use Ansible facts on storage devices (./3-facts/)
- Exercise 4 - Use variables in your playbooks (./4-variables/)
- Exercise 5 - Explore the Ansible Tower environment (./5-explore-tower/)
- Exercise 6 - Create an Ansible Tower Job Template (./6-tower-job-template/)
- Exercise 7 - Create an Ansible Tower Survey (./7-tower-survey/)
- Exercise 8 - Create an Ansible Tower Workflow (./9-tower-workflow)

storage Diagram

Additional information

- Storage Automation with Ansible Homepage (https://www.ansible.com/storage-automation)
- List of Storage Ansible Modules (http://docs.ansible.com/ansible/latest/list_of_storage_modules.html)
- Module Maintenance & Support (http://docs.ansible.com/ansible/latest/modules_support.html)

Exercise 1 - Exploring the lab environment

Table of Contents

Objective

Explore and understand the lab environment. This exercise will cover

- Determining the Ansible version running on the control node
- Locating and understanding the Ansible configuration file (`ansible.cfg`)
- Locating and understanding an `ini` formatted inventory file

Diagram

There is the Ansible Control Node, Netapp Cloud Ontap working environment and two RHEL 7 nodes. Most of this workshop will be spent configuring the storage device.

Guide

Step 1

Navigate to the `storage-workshop` directory.

```
[student1@ansible ~]$ cd ~/storage-workshop/
[student1@ansible storage-workshop]$
[student1@ansible storage-workshop]$ pwd
/home/student1/storage-workshop
```

- `~` - the tilde in this context is a shortcut for `/home/student1`

- pwd - Linux command for print working directory. This will show the full path to the current working directory.

Step 2

Run the `ansible` command with the `--version` command to look at what is configured:

```
ansible 2.8.1
  config file = /home/student1/.ansible.cfg
  configured module search path = [u'/home/student1/.ansible/plugins/modules', u'/u
sr/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, Jun 11 2019, 12:19:05) [GCC 4.8.5 20150623 (Re
d Hat 4.8.5-36)]
```

> *Note: The ansible version you see might differ from the above output*

This command gives you information about the version of Ansible, location of the executable, version of Python, search path for the modules and location of the `ansible configuration file`.

Step 3

Use the `cat` command to view the contents of the `ansible.cfg` file.

```
[student1@ansible ~]$ cat ~/.ansible.cfg
[defaults]
connection = smart
timeout = 60
forks = 10
inventory = ~student1/lightbulb/lessons/lab_inventory/
host_key_checking = False
```

Note the following parameters within the `ansible.cfg` file:

- `inventory`: shows the location of the ansible inventory being used

Step 4

The scope of a `play` within a `playbook` is limited to the groups of hosts declared within an Ansible **inventory**. Ansible supports multiple inventory (http://docs.ansible.com/ansible/latest/intro_inventory.html) types. An inventory could be a simple flat file with a collection of hosts defined within it or it could be a dynamic script (potentially querying a CMDB backend) that generates a list of devices to run the playbook against.

```
[student1@ansible ~]$ cat ~/lightbulb/lessons/lab_inventory/student1-instances
```

```
[web]
node-2 ansible_host=3.95.216.234
node-1 ansible_host=54.196.255.155

[ansible]
control ansible_host=3.80.127.176

[all:vars]
ansible_user=student1
ansible_ssh_pass=r3dh4t
ansible_port=22
vserver=citi_student1

[ontap]
ontap-mgr ansible_host=172.48.0.80 netapp_hostname=172.48.0.80 netapp_username=adm
in netapp_password=redhat123
```

Step 5

In the above output every `[ ]` defines a group. For example `[web]` is a group that contains the hosts `node-1` and `node-2`. Groups can also be ***nested***.

> *Parent groups are declared using the `children` directive. Having nested groups allows the flexibility of assigining more specific values to variables.*
>
> *Note: A group called **all** always exists and contains all groups and hosts defined within an inventory.*

We can associate variables to groups and hosts.

Host variables can be defined on the same line as the host themselves. For example for the host `rtr1`:

```
ontap-mgr ansible_host=172.48.0.80 ansible_user=admin ansible_password=redhat123
```

- `ontap-mgr` - The name that Ansible will use. This can but does not have to rely on DNS
- `ansible_host` - The IP address that ansible will use, if not configured it will default to DNS
- `private_ip` - This value is not reserved by ansible so it will default to a host variable (http://docs.ansible.com/ansible/latest/intro_inventory.html#host-variables) . This variable can be used by playbooks or ignored completely.

Group variables groups are declared using the `vars` directive. Having groups allows the flexibility of assigning common variables to multiple hosts. Multiple group variables can be

```
[[all:vars]
ansible_user=student1
ansible_ssh_pass=r3dh4t
ansible_port=22
netapp_hostname=172.48.0.80
netapp_username=admin
netapp_password=redhat123
```

- `ansible_user` - The user ansible will be used to login to this host, if not configured it will default to the user the playbook is run from
- `ansible_network_os` - This variable is necessary while using the `network_cli` connection type within a play definition, as we will see shortly.
- `ansible_connection` - This variable sets the connection plugin (https://docs.ansible.com/ansible/latest/plugins/connection.html) for this group. This can be set to values such as `netconf`, `httpapi` and `network_cli` depending on what this particular network platform supports.

Complete

You have completed lab exercise 1

Click Here to return to the Ansible Storage Automation Workshop (../README.md)

Exercise 2 - First Ansible Playbook

Table of Contents

Objective

Use Ansible to update the configuration of routers. This exercise will not create an Ansible Playbook, but use an existing provided one.

This exercise will cover

- examining an existing Ansible Playbook
- executing an Ansible Playbook on the command line using the `ansible-playbook` command
- check mode (the `--check` parameter)
- verbose mode (the `--verbose` or `-v` parameter)

Guide

Step 1

Navigate to the `storage-workshop` directory if you are not already there.

```
[student1@ansible ~]$ cd ~/storage-workshop/
[student1@ansible storage-workshop]$
[student1@ansible storage-workshop]$ pwd
/home/student1/storage-workshop
```

Examine the provided Ansible Playbook named `playbook.yml`. Feel free to use your text editor of choice to open the file. The sample below will use the Linux `cat` command.

```
[student1@ansible storage-workshop]$ cat playbook.yml
---
```

```
  gather_facts: no
  connection: local

  tasks:
  - name: ensure the the desired snmp strings are present
    na_ontap_snmp:
      state: present
      community_name: ansible_public
      access_control: 'ro'
      hostname: "{{ netapp_hostname }}"
      username: "{{ netapp_username }}"
      password: "{{ netapp_password }}"
      https: true
      validate_certs: false
```

- `cat` - Linux command allowing us to view file contents
- `playbook.yml` - provided Ansible Playbook

We will explore in detail the components of an Ansible Playbook in the next exercise. It is suffice for now to see that this playbook will set one Netapp snmp string

```
snmp-server community ansible-public RO
```

Step 3

Run the playbook using the `ansible-playbook` command:

```
[student1@ansible storage-workshop]$ ansible-playbook playbook.yml
```

```
PLAY [localhost] ******************************************************************
***********************************************************

TASK [ensure the the desired snmp strings are present] ***************************
***********************************************************
ok: [localhost]

---
PLAY RECAP ************************************************************************
**********************************************************
localhost                  : ok=1    changed=1    unreachable=0    failed=0    ski
pped=0    rescued=0    ignored=0
```

Step 4

Verify that the Ansible Playbook worked. Login to ontap cluster and check the running configuration on the Cisco IOS-XE device.

```
[student1@ansible storage-workshop]$ ssh ip_address_of_cluster_mgr
```

```
contact:

location:

authtrap:
        0
init:
        1
traphosts:
        -
community:
        ro  ansible_public
```

Step 5

The `na_ontap_snmp` module is idempotent. This means, a configuration change is pushed to the device if and only if that configuration does not exist on the end hosts.

> *Need help with Ansible Automation terminology? Check out the glossary here* (https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html) *for more information on terms like idempotency.*

To validate the concept of idempotency, re-run the playbook:

```
[student1@ansible storage-workshop]$  ansible-playbook playbook.yml
```

```
PLAY [localhost] *****************************************************************
***********************************************************

TASK [ensure the the desired snmp strings are present] ***************************
***********************************************************
ok: [localhost]

PLAY RECAP ***********************************************************************
**********************************************************
localhost                  : ok=1    changed=0    unreachable=0    failed=0    ski
pped=0    rescued=0    ignored=0

[student1@ansible storage-workshop]$
```

> *Note: See that the **changed** parameter in the **PLAY RECAP** indicates 0 changes.*

Re-running the Ansible Playbook multiple times will result in the same exact output, with **ok=1** and **change=0**. Unless another operator or process removes or modifies the existing configuration on rtr1, this Ansible Playbook will just keep reporting **ok=1** indicating that the configuration already exists and is configured correctly on the network device.

Step 6

```
snmp-server community ansible-test RO
```

Use the text editor of your choice to open the `playbook.yml` file to add the command:

```
[student1@ansible storage-workshop]$ nano playbook.yml
```

The Ansible Playbook will now look like this:

```
---
- hosts: ontap
  gather_facts: no
  connection: local

  tasks:
  - name: ensure the the desired snmp strings are present
    na_ontap_snmp:
      state: present
      community_name: ansible_public
      access_control: 'ro'
      hostname: "{{ netapp_hostname }}"
      username: "{{ netapp_username }}"
      password: "{{ netapp_password }}"
      https: true
      validate_certs: false

  - name: ensure the the desired snmp strings are present
    na_ontap_snmp:
      state: present
      community_name: ansible_test
      access_control: 'ro'
      hostname: "{{ netapp_hostname }}"
      username: "{{ netapp_username }}"
      password: "{{ netapp_password }}"
      https: true
      validate_certs: false
```

Step 7

Re-run this playbook to push the changes.

```
PLAY [localhost] *****************************************************************
***********************************************************

TASK [ensure the the desired snmp strings are present] ***************************
***********************************************************
ok: [localhost]

TASK [ensure the the desired snmp strings are present] ***************************
```

```
PLAY RECAP *************************************************************************
**********************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    ski
pped=0    rescued=0    ignored=0
```

Step 10

Verify that the Ansible Playbook applied `ansible-test` community. Login to ontap cluster manager and check the running configuration on the Ontap device.

```
[student1@ansible storage-workshop]$ ssh admin@<ip_address_of_ontap-mgr>
```

```
rtr1#sh run | i snmp
citi_student1::> system snmp show
contact:

location:

authtrap:
        0
init:
        1
traphosts:
        -
community:
        ro  ansible_public
        ro  ansible_test

citi_student1::>
```

Takeaways

- the **na_ontap** modules are idempotent, meaning they are stateful
- **verbose mode** allows us to see more output to the terminal window, including which commands would be applied
- This Ansible Playbook could be scheduled in **Red Hat Ansible Tower** to enforce the configuration. For example this could mean the Ansible Playbook could be run once a day for a particular device.

Solution

The finished Ansible Playbook is provided here for an answer key: playbook.yml (../playbook.yml)

Complete

You have completed lab exercise 2

Click here to return to the Ansible Storage Automation Workshop (../README.md)

Exercise 3: Ansible Facts

Table of Contents

Objective

Demonstration use of Ansible facts on network infrastructure.

Ansible facts are information derived from speaking to the remote storage elements. Ansible facts are returned in structured data (JSON) that makes it easy manipulate or modify. For example a network engineer could create an audit report very quickly using Ansible facts and templating them into a markdown or HTML file.

This exercise will cover:

- Building an Ansible Playbook from scratch.
- Using ansible-doc (https://docs.ansible.com/ansible/latest/cli/ansible-doc.html) .
- Using the na_ontap_gather_facts module (https://docs.ansible.com/ansible/latest/modules/na_ontap_gather_facts_module.html#na-ontap-gather-facts-module) .
- Using the debug module (https://docs.ansible.com/ansible/latest/modules/debug_module.html) .

Step 1

On the control host read the documentation about the `na_ontap_gather_facts` module and the `debug` module.

```
[student1@ansible storage-workshop]$ ansible-doc debug
```

What happens when you use `debug` without specifying any parameter?

```
[student1@ansible storage-workshop]$ ansible-doc na_ontap_gather_facts
```

How can you limit the facts collected ?

Ansible Playbooks are **YAML** files (https://yaml.org/) . YAML is a structured encoding format that is also extremely human readable (unlike it's subset - the JSON format)

Using your favorite text editor (`vim` and `nano` are available on the control host) create a new file called `facts.yml`:

```
[student1@ansible storage-workshop]$ vim facts.yml
```

Enter the following play definition into `facts.yml`:

```
---
- name: gather information from Netapp
  hosts: ontap
  gather_facts: no
  connection: local
```

Here is an explanation of each line:

- The first line, `---` indicates that this is a YAML file.
- The `- name:` keyword is an optional description for this particular Ansible Playbook.
- The `hosts:` keyword means this playbook against the `localhost` defined in the inventory file.
- The `gather_facts: no` is required since as of Ansible 2.8 and earlier, this only works on Linux hosts, and not storage infrastructure. We will use a specific module to gather facts for network equipment.

Step 3

Next, add the first `task`. This task will use the `na_ontap_gather_facts` module to gather facts about each device in the group `cisco`.

```
---
- hosts: ontap
  name: Gather Netapp Ontap Facts
  gather_facts: no
  connection: local

  tasks:
  - name: Gather Netapp Facts
    na_ontap_gather_facts:
      state: info
      hostname: "{{ netapp_hostname }}"
      username: "{{ netapp_username }}"
      password: "{{ netapp_password }}"
      https: true
      validate_certs: false
    register: result
```

A play is a list of tasks. Modules are pre-written code that perform the task.

Step 4

Execute the Ansible Playbook:

```
[student1@ansible storage-workshop]$ ansible-playbook facts.yml
```

The output should look as follows.

```
[student1@ansible storage-workshop]$ ansible-playbook facts.yml

PLAY [Gather Netapp Ontap Facts] ************************************************
***********************************************************

TASK [Gather Netapp Facts] *******************************************************
***********************************************************
ok: [localhost]

PLAY RECAP *************************************************************************
**********************************************************
localhost                  : ok=1    changed=0    unreachable=0    failed=0    ski
pped=0    rescued=0    ignored=0
```

Step 5

The play ran successfully and executed against the Cisco router(s). But where is the output? Re-run the playbook using the verbose -v flag.

```
[student1@ansible storage-workshop]$ ansible-playbook facts.yml -v
Using /home/student1/.ansible.cfg as config file

PLAY [Gather Netapp Ontap Facts] ************************************************
***************************************************************************

TASK [Gather Netapp Facts] *******************************************************
***************************************************************************
ok: [localhost]

TASK [Show output] ***************************************************************
****************************************************************************
ok: [localhost] => {
    "result": {
        "ansible_facts": {
            "discovered_interpreter_python": "/usr/bin/python",
            "ontap_facts": {
                "aggregate_info": {
                    "aggr0_citi_student1_01": {
```

```
                              "fsid": "598399611",
                              "type": "aggr"
.
.
 <output truncated for readability>
.
.
"vserver_name": "svm_citi_student1",
"vserver_subtype": "default",
"vserver_type": "data"
}
},
"vserver_login_banner_info": null,
---
"vserver_motd_info": null
}
},
"changed": false,
"failed": false,
"state": "info"
}
}

PLAY RECAP *********************************************************************
*****************************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0    ski
pped=0    rescued=0    ignored=0
```

> *Note: The output returns key-value pairs that can then be used within the playbook for subsequent tasks. Also note that all variables that start with **ansible_** are automatically available for subsequent tasks within the play.*

Step 6

Running a playbook in verbose mode is a good option to validate the output from a task. To work with the variables within a playbook you can use the `debug` module.

Write two additional tasks that display the routers' OS version and serial number.

`yaml --- - hosts: ontap name: Gather Netapp Ontap Facts gather_facts: no connection: local tasks: - name: Gather Netapp Facts na_ontap_gather_facts: state: info hostname: "{{ netapp_hostname }}" username: "{{ netapp_username }}" password: "{{ netapp_password }}" https: true validate_certs: false - name: Show output debug: msg: "The ontap version number is: {{ ontap_facts.ontap_version }}" - name: Show node serial number debug: msg: "The serial number is: {{ ontap_facts.system_node_info['citi_student1-01'].node_serial_number }}"`

Step 8

Now re-run the playbook but this time do not use the `verbose` flag:

```
PLAY [Gather Netapp Ontap Facts] ********************************************
*******************************************************************

TASK [Gather Netapp Facts] **************************************************
*******************************************************************
ok: [localhost]

TASK [Show output] **********************************************************
********************************************************************
ok: [localhost] => {
    "msg": "The ontap version number is: 160"
}

TASK [Show node serial number] **********************************************
*******************************************************************
ok: [localhost] => {
    "msg": "The serial number is: 902324218975267479 27"
}

PLAY RECAP ******************************************************************
*********************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0    ski
pped=0    rescued=0    ignored=0
```

Using less than 20 lines of "code" you have just automated version and serial number collection. Imagine if you were running this against your production network! You have actionable data in hand that does not go out of date.

Takeaways

- The ansible-doc (https://docs.ansible.com/ansible/latest/cli/ansible-doc.html) command will allow you access to documentation without an internet connection. This documentation also matches the version of Ansible on the control node.
- The na_ontap_gather_facts (https://docs.ansible.com/ansible/latest/modules/na_ontap_gather_facts_module.html#na-ontap-gather-facts-module) gathers structured data specific for Netapp Ontap.
- The debug module (https://docs.ansible.com/ansible/latest/modules/debug_module.html) allows an Ansible Playbook to print values to the terminal window.

Solution

The finished Ansible Playbook is provided here for an answer key: facts.yml (facts.yml) .

▪

You have completed lab exercise 3

Click here to return to the Ansible Storage Automation Workshop (../README.md)

Exercise 4 - Using Variables and Loops

Previous exercises showed you the basics of Ansible Core. In the next few exercises, we are going to teach some more advanced ansible skills that will add flexibility and power to your playbooks.

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

Variables are how we deal with differences between your systems, allowing you to account for a change in port, IP address or directory.

Loops enable us to repeat the same task over and over again. For example, lets say you want to install 10 packages. By using an ansible loop, you can do that in a single task.

For a full understanding of variables and loops; check out our Ansible documentation on these subjects.

- Ansible Variables (http://docs.ansible.com/ansible/latest/playbooks_variables.html)
- Ansible Loops (http://docs.ansible.com/ansible/latest/playbooks_loops.html)
- Ansible Handlers (http://docs.ansible.com/ansible/latest/playbooks_intro.html#handlers-running-operations-on-change)

Section 1: Using variables in a Playbook

To begin, we are going to create a new playbook, but it should look very familiar to the one you created previously

Step 1:

Navigate to your home directory and create a new playbook.

```
vim motd.yml
```

Step 2:

Add a play definition and some variables to your playbook.

%u2014--
%0A%u2013%20hosts%3A%20ontap%0A%20%20gather_facts%3A%20false%0A%20%20name%3A%20Setup%20ONTAP%0A%20%20connection%3A%20local%0A%2

Step 3:

Add a new task called ***set motd***.

%20%20tasks%3A%0A%20%20-
%20name%3A%20set%20motd%0A%20%20%20%20na_ontap_motd%3A%0A%20%20%20%20%20%20state%3A%20%22%7B%7B%20state%20%7D%7D%22%0A%2

NOTE

- `vars:` You've told Ansible the next thing it sees will be a variable name +
- `motd_message` You are defining a list-type variable called motd_message. What follows is the message you want to set +

Step 4

Run your playbook.

```
$ ansible-playbook motd.yml
```

```
PLAY [ontap] *******************************************************************************************************************
*

TASK [set motd] ****************************************************************************************************************
*
changed: [ontap-mgr]

PLAY RECAP *********************************************************************************************************************
ontap-mgr                  : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Verify that the Ansible Playbook applied set your `motd_message` string. Login to ontap cluster manager and check the running configuration on the Ontap device.

```
[student1@ansible storage-workshop]$ ssh admin@<ip_address_of_ontap-mgr>
```

```
citi_student1::> security login motd show
Vserver: citi_student1
Is the Cluster MOTD Displayed?: true
Message
```

```
Vserver: svm_citi_student1
Is the Cluster MOTD Displayed?: true
Message
-----------------------------------------------------------------------------
I am the MOTD.  I am managed by student1

2 entries were displayed.

citi_student1::>
```

Section 2: Creating Volumes and enabling file shares

Step 1:

Let's start by creating some files just for variables. In your `storage_workshop` directory, create two new directories `host_vars` and `group_vars`.

```
$ cd ~/storage_workshop
$ mkdir -p host_vars group_vars
```

In the `group_vars` directory create a file called `ontap` and edit it so that it looks like the following:

```
aggrs:
  - { name: "aggr1", vol_name: "ansibleVol1", vol_size: "10", vol_size_unit: "gb"  }
  - { name: "aggr1", vol_name: "ansibleVol2", vol_size: "10", vol_size_unit: "gb"  }
```

After that, create a playbook called `create-volume.yml` to create your volumes.

```
- hosts: ontap
  name: Volume Action
  gather_facts: false
  connection: local

  tasks:
  - name: Volume Create
    na_ontap_volume:
      state: present
      name: "{{ item.vol_name }}"
      vserver: "svm_{{ vserver_name }}"
      aggregate_name: "{{ item.name }}"
      size: 10
      size_unit: gb
      policy: default
      junction_path: "/{{ item.vol_name }}"
      hostname: "{{ netapp_hostname }}"
      username: "{{ netapp_username }}"
      password: "{{ netapp_password }}"
      https: true
      validate_certs: false
    with_items: "{{ aggrs }}"
```

NOTE

`aggrs` In your variables file you defined a list-type variable call `aggrs`. What follows is a list of aggregates and volumes you will create on your storage device. `item` You are telling Ansible that this will expand into a list. You can choose members of that list by adding `item.subelement` to the variable.

`with_items: "{{ aggrs }}"` This is your loop which is instructing Ansible to perform this task on every `item` in `aggrs`

Step 2:

Run your playbook:

```
$ ansible-playbook create_volume.yml -v
Using /home/student1/.ansible.cfg as config file

PLAY [Volume Action] ***********************************************************************************************************************
*

TASK [Volume Create] ***********************************************************************************************************************
*
ok: [ontap-mgr] => (item={u'vol_size': u'10', u'vol_size_unit': u'gb', u'vol_name': u'ansibleVol1', u'name': u'aggr1'}) => {"ansible_facts"
: {"discovered_interpreter_python": "/usr/bin/python"}, "ansible_loop_var": "item", "changed": false, "item": {"name": "aggr1", "vol_name":
"ansibleVol1", "vol_size": "10", "vol_size_unit": "gb"}}
changed: [ontap-mgr] => (item={u'vol_size': u'10', u'vol_size_unit': u'gb', u'vol_name': u'ansibleVol2', u'name': u'aggr1'}) => {"ansible_l
oop_var": "item", "changed": true, "item": {"name": "aggr1", "vol_name": "ansibleVol2", "vol_size": "10", "vol_size_unit": "gb"}}

PLAY RECAP ************************************************************************************************************************************
```

Step 3:

Verify that the Ansible Playbook created your two volumes. Login to ontap cluster manager and check that the volumes were created using the `volume show` command:

```
citi_student1::> volume show
Vserver   Volume       Aggregate    State      Type       Size  Available Used%
--------- ------------ ------------ ---------- ---- ---------- ---------- -----
citi_student1-01
          vol0         aggr0_citi_student1_01
                                    online     RW      72.71GB    64.49GB    6%
svm_citi_student1
          ansibleVol1  aggr1        online     RW         10GB     9.50GB    0%
svm_citi_student1
          ansibleVol2  aggr1        online     RW         10GB     9.50GB    0%
svm_citi_student1
          svm_citi_student1_root
                       aggr1        online     RW          1GB    971.6MB    0%
4 entries were displayed.
```

Section 3: YAML Aliases

The previous playbook can also be written this way:

```
- hosts: ontap
  name: Volume Action
  gather_facts: false
  connection: local
  vars:
    login: &login
      hostname: "{{ netapp_hostname }}"
      username: "{{ netapp_username }}"
      password: "{{ netapp_password }}"
      https: true
      validate_certs: false

  tasks:
  - name: Volume Create
    na_ontap_volume:
      state: present
      name: "{{ item.vol_name }}"
      vserver: "svm_{{ vserver_name }}"
      aggregate_name: "{{ item.name }}"
      size: 10
      size_unit: gb
      policy: default
      junction_path: "/{{ item.vol_name }}"
      <<: *login
    with_items: "{{ aggrs }}"
```

Look at the variables on lines 5-11. This is the setup and use of a YAML alias. This is not specific to Ansible, this is a YAML feature. It is created in a variable section and its format is as follows.

```
vars:
  alias_name: &alias_name
  variable: value
  variable: value
```

This makes 'alias_name' represent all the variable:value pairs that are setup in it. It is referenced with the following line.

```
<<: *alias_name
```

I am using an alias called 'login' and you can see how this will save you a lot of typing in the future.

Exercise 5: Explore Red Hat Ansible Tower

Table of Contents

Objective

Explore and understand the lab environment. This exercise will cover

- Determining the Ansible version running on the control node
- Locating and understanding:
    - Ansible Tower **Inventory**
    - Ansible Tower **Credentials**
    - Ansible Tower **Projects**

Guide

Step 1: Login to Ansible Tower

Open up your web browser and type in the Ansible control node's DNS name

> *For example if the student was assigned the student1 workbench and the workshop name was `citi` the link would be:*

```
**https://citi-student1.rhdemo.io**
```

> *This login information has been provided by the instructor at the beginning of class.*

- password provided by instructor

After logging in the Job Dashboard will be the default view as shown below.

1. Click on the **i** information button on the top right of the user interface.

2. A window will pop up similar to the following:

   Take note that both the Ansible Tower version and the Ansible Engine version are provided here.

Step 2: Examine the Ansible Tower Inventory

An inventory is required for Red Hat Ansible Tower to be able to run jobs. An inventory is a collection of hosts against which jobs may be launched, the same as an Ansible inventory file. In addition, Red Hat Ansible Tower can make use of an existing configuration management data base (cmdb) such as ServiceNow or Infoblox DDI.

> *More info on Inventories in respect to Ansible Tower can be found in the documentation here (https://docs.ansible.com/ansible-tower/latest/html/userguide/inventories.html)*

1. Click on the **Inventories** button under **RESOURCES** on the left menu bar.

2. Under Inventories there will be two inventories, the `Demo Inventory` and the `Workshop Inventory`. Click on the `Workshop Inventory`.

3. Under the `Workshop Inventory` click the **HOSTS** button at the top. There will be four hosts here, rtr1 through rtr4 as well as the ansible control node. Click on one of the devices.

   Take note of the **VARIABLES** field. The `host_vars` are set here including the `ansible_host` variable.

4. Click on the `Workshop Inventory` link at the top of the page to return the top level menu.

5. Click on **GROUPS**. There will be multiple groups here including `routers` and `cisco`. Click on one of the groups.

   Take note of the **VARIABLES** field. The `group_vars` are set here including the

Here is a walkthrough:

Prefer Youtube? Click Here (https://youtu.be/4JNbFNSUS9g)

Step 3: Examine the Ansible Tower Workshop Project

A project is how Ansible Playbooks are imported into Red Hat Ansible Tower. You can manage playbooks and playbook directories by either placing them manually under the Project Base Path on your Ansible Tower server, or by placing your playbooks into a source code management (SCM) system supported by Tower, including Git, Subversion, and Mercurial.

> *For more information on Projects in Tower, please refer to the documentation (https://docs.ansible.com/ansible-tower/latest/html/userguide/projects.html)*

1. Click on the **Projects** button under **RESOURCES** on the left menu bar.

2. Under **PROJECTS** there will be two pre-configured projects, `Demo Project` and the `Workshop Project`. Click on the `Workshop Project`.

   Note that `GIT` is listed for this project. This means this project is using Git for SCM.

3. Under the `Workshop Project` click the **SCM TYPE** drop down menu

   Note that Git, Mercurial and Subversion are choices. Return the choice to Git so that the Project continues to function correctly.

Prefer Youtube? Click Here (https://youtu.be/xRA97XTxMjA)

Step 4: Examine the Ansible Tower Workshop Credential

Credentials are utilized by Tower for authentication when launching **Jobs** against machines, synchronizing with inventory sources, and importing project content from a version control system. For the workshop we need a credential to authenticate to the network devices.

> *For more information on Credentials in Tower please refer to the documentation (https://docs.ansible.com/ansible-tower/latest/html/userguide/credentials.html) .*

1. Click on the **Credentials** button under **RESOURCES** on the left menu bar.

2. Under **CREDENTIALS** there will be two pre-configured credentials, `Demo Credential` and the `Workshop Credentials`. Click on the `Workshop Credential`.

3. Under the `Workshop Credential` examine the following:
   - The **CREDENTIAL TYPE** is a **Machine** credential.
   - The **USERNAME** is set to `ec2-user`.
   - The **PASSWORD** is blank.
   - The **SSH PRIVATE KEY** is already configured, and is **ENCRYPTED**.

Prefer Youtube? Click Here (https://youtu.be/UT0t_hlNw-c)

Takeaways

- Ansible Tower needs an inventory to execute Ansible Playbooks again. This inventory is identical to what users would use with the command line only Ansible project.
- Although this workshop already setup the inventory, importing an existing Ansible Automation inventory is easy. Check out this blog post (https://www.ansible.com/blog/three-quick-ways-to-move-your-ansible-inventory-into-red-hat-ansible-tower) for more ways to easily get an existing inventory into Ansible Tower.
- Ansible Tower can sync to existing SCM (source control management) including Github.
- Ansible Tower can store and encrypt credentials including SSH private keys and plain-text passwords. Ansible Tower can also sync to existing credential storage systems such as CyberArk and Vault by HashiCorp

---

Complete

You have completed lab exercise 5

You have now examined all three components required to get started with Ansible Tower. A credential, an inventory and a project. In the next exercise we will create a job template.

Click here to return to the Ansible Network Automation Workshop (../README.md)

M# Exercise 6: Creating a Tower Job Template

Table of Contents

Objective

Demonstrate applying an export policy job template for Red Hat Ansible Tower. This job template will create an export policy on your Ontap device so that the volumes that you previously created can be shared.

To run an Ansible Playbook in Ansible Tower we need to create a **Job Template**. A **Job Template** requires:

- An **Inventory** to run the job against
- A **Credential** to login to devices.
- A **Project** which contains Ansible Playbooks

Guide

Step 1: Create a Job Template

1. Open the web UI and click on the `Templates` link on the left menu.

   ![templates link](images/templates.png

2. Click on the green button to create a new job template

   > *Make sure to select `Job Template` and not `Workflow Template`)*

3. Fill out the job template parameters as follows:

policy.yml | | Credential | Workshop Credential |

1. Scroll down and click the green `save` button.

Here is a walkthrough:

Prefer Youtube? Click Here (https://youtu.be/EQVkFaQYRiE)

Step 2: Launch the Job Template

1. Navigate back to the `Templates` window, where all Job Templates are listed.
2. Launch the `Create Export Policy` Job Template by clicking the Rocket button.

   When the rocket button is clicked this will launch the job. The job will open in a new window called the **Job Details View**. More info about Tower Jobs (https://docs.ansible.com/ansible-tower/latest/html/userguide/jobs.html) can be found in the documentation.

Step 3: Examine the Job Details View

On the left side there is a **Details pane** on the right side there is the **Standard Out pane**.

1. Examine the **Details pane**

   The **Details pane** will information such as the timestamp for when the job started and finished, the job type (Check or Run), the user that launched the job, which Project and Ansible Playbook were used and more.

   If the Job has not finished yet, the **Details Pane** will have a cancel button that can be used to stop the Job.

2. Examine the **Standard Out pane**

   The **Standard Out pane** will display the output from the Ansible Playbook. Every task output will match exactly what would be seen on the command line.

3. Click on the **Expand Output** button

   This will expand the **Standard Out pane** to take the entirety of the window.

4. Click on a task in the **Standard Out pane** to open up structured output from that particular task.

Step 4: Examine the Jobs window

Any **Job Template** that has been run or is currently running will show up under the **Jobs** window.

1. Click the Jobs button the left menu.

   The Jobs link displays a list of jobs and their status–shown as completed successfully or failed, or as an active (running) job. Actions you can take from this screen include viewing the details and standard output of a particular job, relaunch jobs, or remove jobs.

2. Click on the **Create Export Policy** Job

   The **Create Export Policy** job was the most recent (unless you have been launching more jobs). Click on this job to return to the **Job Details View**. Ansible Tower will save the history of every job launched.

Step 5: Verify the export policy was created

1. Login to ontap cluster manager and check that the volumes were created using the `vserver export-policy rule show` command:

```
citi_student1::> vserver export-policy rule show
             Policy          Rule    Access   Client                RO
Vserver      Name            Index   Protocol Match                 Rule
------------ --------------- ------  -------- --------------------- -----
----
svm_citi_student1
             export-svm_citi_student1
                             1       any      172.48.0.0/20         any
svm_citi_student1
             export-svm_citi_student1
                             2       any      172.48.0.0/16         any
svm_citi_student1
             export-svm_citi_student1-ansibleVol1
                             1       any      172.48.0.0/16         any
svm_citi_student1
             export-svm_citi_student1-ansibleVol2
                             1       any      172.48.0.0/16         any
4 entries were displayed.
```

You have successfully demonstrated

- Creating a Job Template for creating export policies
- Launching a Job Template from the Ansible Tower UI
- Verifying the policies were correctly created

Complete

You have completed lab exercise 6

Click here to return to the Ansible Network Automation Workshop (../README.md)

Exercise 7: Creating a Survey

Table of Contents

Objective

Demonstrate the use of Ansible Tower survey feature (https://docs.ansible.com/ansible-tower/latest/html/userguide/job_templates.html#surveys) . Surveys set extra variables for the playbook similar to 'Prompt for Extra Variables' does, but in a user-friendly question and answer way. Surveys also allow for validation of user input.

Guide

Step 1: Create a Job Template

1. Open the web UI and click on the `Templates` link on the left menu.

2. Click on the green `+` button to create a new job template (make sure to select `Job Template` and not `Workflow Template`)

   | Parameter | Value | |---|---| | Name | Mount NFS Shares | | Job Type | Run | | Inventory | Workshop Inventory | | Project | Workshop Project | | Playbook | `nfs-mount.yml` | | Credential | Workshop Credential |

3. Scroll down and click the green `SAVE` button.

Here is what the `nfs-mount.yml` Ansible Playbook looks like:

```
yml --- - hosts: web gather_facts: no vars: volname: - "ansibleVol1" - "ansibleVol2"
#Variables to passed in: #state: #mount_state: tasks: - name: Update fstab file
lineinfile: state: "{{ exists }}" path: /etc/fstab line: "{{data_ips}}:/{{ item }}
/mnt/{{ item }} nfs defaults 0 0" with_items: "{{ volname }}" - name: Verify mount
directory exists file: state: "{{ exists }}" path: "/mnt/{{ item }}" state: directory
with_items: "{{ volname }}" - name: Mount nfs export mount: state: "{{ mount_state
}}" path: "/mnt/{{ item }}" src: "{{data_ips}}:/{{ item }}" fstype: nfs with_items: "
{{ volname }}" - name: Show volume is mounted shell: "df -h" register: output -
debug: msg: "{{ output.stdout.split('\n') }}"
```

> *Note: You can also view the Ansible Playbook here:*
> ***https://github.com/leerich/ansible-storage*** *(https://github.com/leerich/ansible-storage)*

The first task is editing the `/etc/fstab` file and adding mount points. This task is using the `lineinfile` module.

- The `lineinfile` module ensures a particular line is in a file, or replace an existing line using a back-referenced regular expression.
- This is primarily useful when you want to change a single line in a file only.
- See the replace module if you want to change multiple, similar lines or check blockinfile if you want to insert/update/remove a block of lines in a file. For other cases, see the copy or template modules.

The second task checks to see if a directory for the mount point exists and if it doesn't it either creates it or removes it depending on the value of the `exists` variable that is passed in.

The third task uses the `mount` module to actually perform the mount or unmount on the intended hosts.

Also note that we are passing in 2 variables to the task file.

1. `state`: This variable is populated using the `exists` variable This variable can be `present` or `absent`
2. `mount_state`: This variable is populated by a variable named `mount_state` The mount state can either be `mounted` or `unmounted`

Step 3: Create a survey

In this step you will create a *"survey"* of user input form to collect input from the user and populate the values for the variables `exists` and `mount_state`

1. Click on the blue add survey button

| Parameter | Value | |---|---| | Prompt | Please select the state of the mount in fstab | | Description | Select one of the following | | Answer Variable Name | `exists` | | Answer type | Multiple Choice (single select) | | Multiple Choice Options | present
absent | | Required | Checkmark |

For example:

3. Click the green `+Add` button

4. Next we will create a survey prompt to whether to mount the nfs share on your servers.

   | Parameter | Value | |--------------------------|--------------------------------| | Prompt | Please enter the mount state | | Description | Please select whether you want to actually mount the share | | Answer Variable Name | `mount_state` | | Answer type | Multiple Choice(single select) | | Multiple Choice Options | mounted
unmounted | | default answer | mounted | | Required | Checkmark |

   For example:

5. Click the green `+Add` button

6. Click the green **SAVE** button to save the survey. This will exit back to the main job template window. Scroll down and click the second green **SAVE** button to exit to the job templates window.

Step 4: Launch the Job Template

1. Click on the rocket ship to launch the job template.

   The job will immediately prompt the user to set the two variables defined in the survey.

2. Choose between `absent` and `present`.

3. Choose between `mounted` and `unmounted`.

4. Click next to see how the survey rendered the input as extra vars for the Ansible Playbook. For this example screen shot the word ANSIBLE rendered into ASCII art.

Let the job run to completion. Let the instructor know if anything fails.

Step 5: Verify the changes

Takeaways

You have successfully demonstrated

- Creation of a Job Template for configuring a banner on multiple network operating systems including Arista EOS, Cisco IOS and Juniper Junos.
- Creation of a self service survey for the Job Template to fill out the `network_banner` and `banner_type` variables
- Executing a Job Template on all four routers, loading a banner on them simultaneously

---

Complete

You have completed lab exercise 7

Click here to return to the Ansible Network Automation Workshop (../README.md)

Exercise 9: Creating a Workflow

Table of Contents

Objective

Demonstrate the use of Ansible Tower workflow (https://docs.ansible.com/ansible-tower/latest/html/userguide/workflows.html) . Workflows allow you to configure a sequence of disparate job templates (or workflow templates) that may or may not share inventory, playbooks, or permissions.

For this exercise we will create an svm server and interface, if the job successfully completes the workflow will simultaneously configure enable the nfs service and modify the policy.

Guide

Step 1: Create a Job Template

1. Make sure you are logged in as the **admin** user.
2. Click on the **Templates** link on the left menu.
3. Click on the green **+** button. Select the **Workflow Template**.
4. Fill out the the form as follows:

Parameter	Value
Name	Workshop Workflow
Organization	Default
Inventory	Workshop Inventory

1. Click on the **Save** button

Step 2: The Workflow Visualizer

1. When you click the **SAVE** the **WORKFLOW VISUALIZER** should automatically open. If not click on the blue **WORKFLOW VISUALIZER** button.

2. By default only a green **START** button will appear. Click on the **START** button.

3. The **ADD A TEMPLATE** window will appear on the right. Select the *Create SVM and Interface* Job Template that was created for you. Use the drop down box to select run. Click the green **SELECT** button.

   The `Create SVM and Interface` job template is now a node. Job or workflow templates are linked together using a graph-like structure called nodes. These nodes can be jobs, project syncs, or inventory syncs. A template can be part of different workflows or used multiple times in the same workflow. A copy of the graph structure is saved to a workflow job when you launch the workflow.

Step 3: Add the Enable NFS Job Template

1. Hover over the *Create SVM and Interface* node and click the green **+** symbol. The **ADD A TEMPLATE** window will appear again.

2. Select the **Enable NFS** Job Template. For the **Run** parameter select **On Success** from the drop down menu.

3. Click **SELECT**

4. A green line should exist between **Create SVM and Interface** and **Enable NFS**

1. Hover over the *Create SVM and Interface* node (not the **Enable NFS** node) and click the green **+** symbol. The **ADD A TEMPLATE** will appear again.
2. Select the **Create Policy** Job Template. For the **Run** parameter, select **On Success** from the drop down menu. Once the **SELECT** button appears green click it.

Step 5: Add the Create NFS Mounts Job Template

1. Hover over the **Enable NFS** node and click the green **+** symbol. The **ADD A TEMPLATE** will appear again.
2. Select the **Create NFS Mounts** job template. For the **Run** parameter, select **On Success** from the drop down menu.

Step 6: Create a converged link

1. Hover over the **Network-User** node and click the blue **chain** symbol.
2. Now, click on the existing **Network-Restore**. A **ADD LINK** window will appear. For the **RUN** parameter choose **On Failure**.
3. Click the green **SAVE** button

Step 7: Run the Workflow - NOT GOING TO WORK. HASN'T BEEN TESTED YET

1. Return to the **Templates** window
2. Click the rocket ship to launch the **Workshop Workflow** workflow template.

   At any time during the workflow job you can select an individual job template by clicking on the node to see the status.

Takeaways

- created a workflow template that creates a backup, attempts to create a user and banner for all network nodes
- made the workflow robust, if either job template fails it will restore to the specified backup
- launched the workflow template and explored the **VISUALIZER**

Complete

You have completed lab exercise 9

Click here to return to the Ansible Network Automation Workshop (../README.md)