Dell EMC Isilon with Ansible and AWX (Red Hat Ansible Tower)

Description

This whitepaper demonstrates how to use Ansible and AWX (Red Hat Ansible Tower) to interact with Dell EMC Isilon. It provides a step by step guide for configuring Ansible playbooks to interact with Isilon. It is not intended to be an Ansible best practices guide.

2nd of February 2020

Author: Evan Koutsandreou
Senior Systems Engineer
Unstructured Data Solutions
Dell Technologies

Document Version 1.5

Table of Contents

Document revisions

Date	Version	updates	Author
2019/01/08	Initial Draft		Evan Koutsandreou
2019/02/01	0.2	Added AWX section	Evan Koutsandreou
2019/05/23	1.0	Fixed issue with create SMB share playbook	Evan Koutsandreou
2019/06/12	1.1	- Updated AWX playbook, replaced Isilon URL with ansible variable - Added an appendix section, new ansible playbook that utilizes “roles” to simplify Isilon tasks and improve playbook structure and readability - Added SMB role for Onefs 8.2.0 - Added document conclusion	Evan Koutsandreou
2019/06/24	1.2	- Additional role required for Onefs 8.2.0 to support snapshots - Added “The Unofficial Guide” to the document title	Evan Koutsandreou
2019/08/06	1.3	- Added Dell copyright and legal notification - Removed “The Unofficial Guide” from the document title page	Evan Koutsandreou
2019/08/21	1.4	- Minor corrections to contents menu and heading descriptions	Evan Koutsandreou
2020/02/10	1.5	- gramma and general corrections	Evan Koutsandreou

Purpose

Detail the configuration required to enable Dell EMC Isilon interoperability with Ansible core and AWX. AWX is the upstream project from which the Red Hat Ansible Tower offering is derived. Isilon is the industry benchmark for scale-out file and is widely recognized as a leader by Gartner and IDC.

The document will explore using the native Ansible modules to access and drive Isilon functionality. I will demonstrate how to access and use the Isilon RESTful Application Programable Interface (API) with curl, the Ansible "URI" module and the traditional Ansible Command Line Interface (CLI) module.

I will highlight the typical challenges that I encountered using the Isilon API and other variable substitution issues in YAML files used with ansible play books.

I describe the core configuration and troubleshooting of the environment as it relates to the overall purpose of this document. Please note, this document is not intended to be an Ansible best practice or detailed how-to guide.

Ultimately the information provided is focused on enabling system engineers and developers alike to include Isilon in their devOps and infrastructure as code journey.

Context

Ansible is one of several devOps tools gaining widespread popularity within the I.T. industry.

I am a Senior Systems Engineer at Dell EMC and many of my clients (past and present) use Ansible for configuration management, application, cloud, and storage deployments. This guide has been specifically written to empower Dell EMC Isilon clients that have already invested in Ansible.

The examples within this document will be kept simple as the key focus is on interoperability, and enablement;

Intended audience

The document has been written for System administrators, DevOps engineers, developers that wish to incorporate Isilon into their Ansible environment. Concepts and terminology used within this document will be of a technical nature. I will provide links to the relevant how-to documentation to ensure you have all the required knowledge to get started with Isilon and/or duplicate this lab for your own learning.

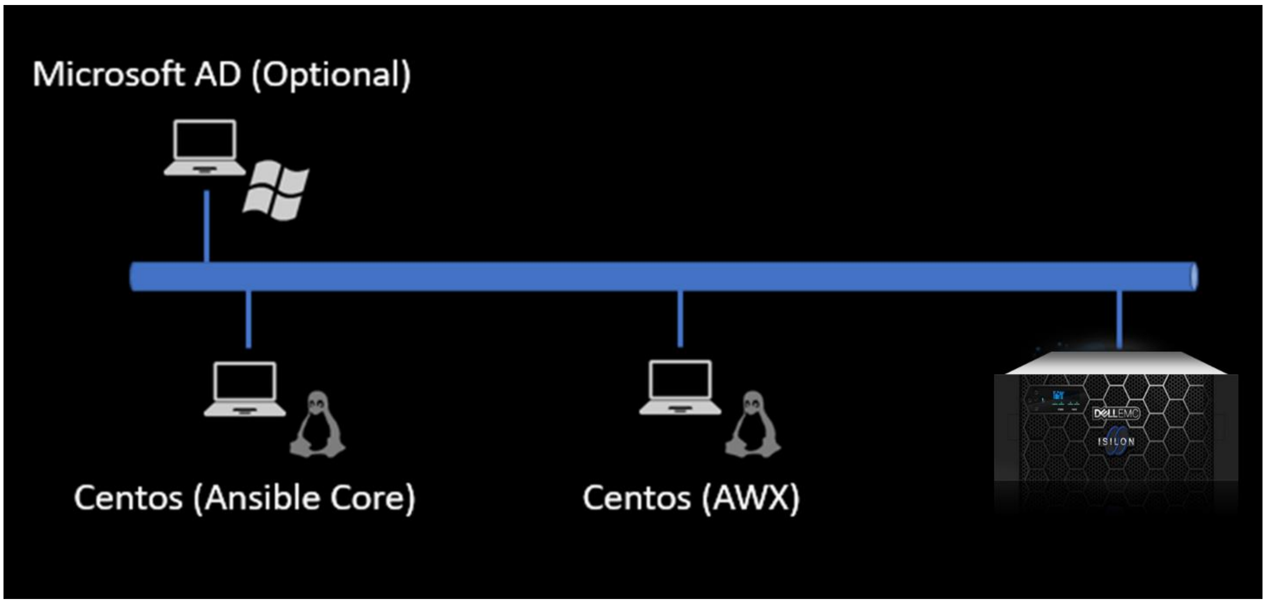
Are you an experienced Ansible user?

If you are an experienced ansible user and are only interested in executing ansible playbooks / templates on Isilon, then continue to one of the following sections:

- Ansible – running ansible play books against Isilon
- AWX (Red Hat Tower) – running Ansible play books against Isilon

Lab environment overview

The lab environment is specially designed to demonstrate using ansible core (CLI version) and AWX (Red Hat Ansible Tower) with Dell EMC Isilon.



A high-level description of the Virtual Machines used in the lab:

- (Optional) Microsoft Active Directory Server
  - Microsoft Windows 2016
- (Required) Isilon cluster - simulator
  - 3x nodes (virtual machines)
  - Onefs version (8.1.x or 8.2.x)
- (Required) Ansible core
  - Centos 7.5 or higher
  - Our Ansible control node
- (Required) AWX
  - Centos 7.5 or higher
  - Ansible
  - Docker community edition
    - AWX components deployed with official docker image repository

Replicating my lab environment:

Depending on your actual environment, the below installation and configuration sections may or may not be relevant.

Where to find, download and configure the software:

(Required) Isilon Cluster

My Isilon lab environment consists of 3x Isilon simulator VMs configured as a single cluster. I happened to use an earlier version-based on 8.1.0.4 however I recommend downloading and using the latest available version (at the time of writing this is version 8.1.2).

Alternatively, If you are lucky enough to have access to a physical (or IsilonSD Edge) test / dev environment, then that would work equally as well.

The Isilon simulator requires VMware virtualization technology:

Isilon simulator software:

https://www.dellemc.com/en-us/products-solutions/trial-software-download/isilon.htm

Follow the installation guide to install your Isilon simulator.

https://support.emc.com/docu90413_Isilon-OneFS-8.1.2-Simulator-Installation-Guide.pdf

Don’t forget to license your cluster using the below command:

```
isi license add —evaluation
ONEFS,SMARTQUOTAS,SNAPSHOTIQ,SMARTCONNECT_ADVANCED,SYNCIQ,SMAR
TPOOLS,SMARTLOCK,HDFS,SMARTDEDUPE,CLOUDPOOLS,SWIFT,HARDENING
```

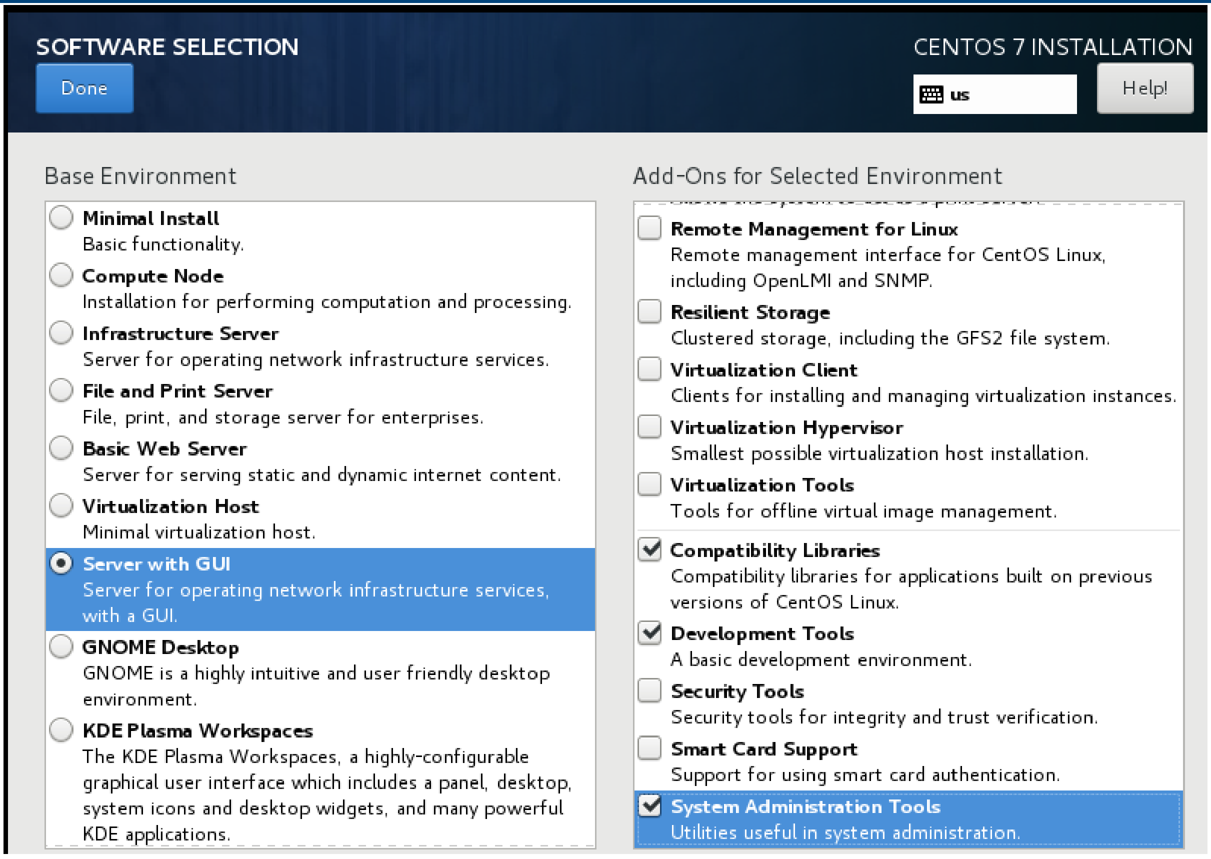
(Required) Install ansible core (control node)

Ansible is installed on a Linux operating system. I have chosen to use Centos 7 for this environment.

Obtain a copy of Centos 7 from the official download site:

Install CentOS 7 and configure the basics during the initial GUI installation:

1. I typically install my Linux lab server instances with the following software to ensure configuration flexibility:
   a. Server with GUI option the best choice if you are not familiar with Linux
   b. Include the optional software packages:
      i. Compatibility Libraries
      ii. Development tools
      iii. System administration tools
      iv. File and Storage Server (optional)
      v. Java platform (optional)
      vi. Network file system client (optional)



SOFTWARE SELECTION
Done
CENTOS 7 INSTALLATION
us
Help!

Base Environment

- ○ **Minimal Install** Basic functionality.
- ○ **Compute Node** Installation for performing computation and processing.
- ○ **Infrastructure Server** Server for operating network infrastructure services.
- ○ **File and Print Server** File, print, and storage server for enterprises.
- ○ **Basic Web Server** Server for serving static and dynamic internet content.
- ○ **Virtualization Host** Minimal virtualization host.
- ● **Server with GUI** Server for operating network infrastructure services, with a GUI.
- ○ **GNOME Desktop** GNOME is a highly intuitive and user friendly desktop environment.
- ○ **KDE Plasma Workspaces** The KDE Plasma Workspaces, a highly-configurable graphical user interface which includes a panel, desktop, system icons and desktop widgets, and many powerful KDE applications.

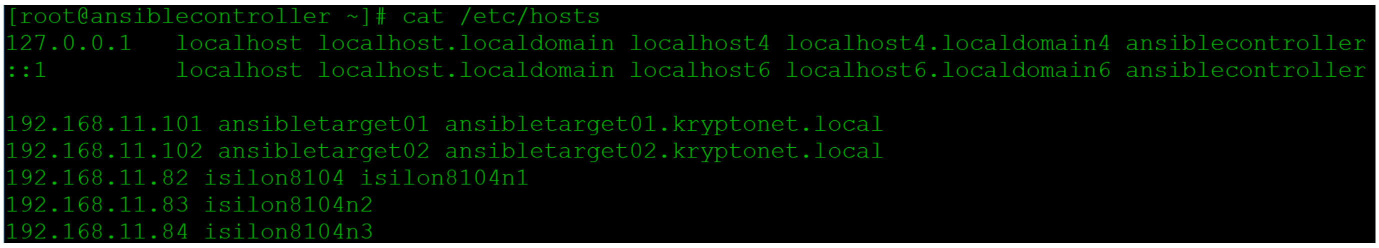
Add-Ons for Selected Environment

- ☐ **Remote Management for Linux** Remote management interface for CentOS Linux, including OpenLMI and SNMP.
- ☐ **Resilient Storage** Clustered storage, including the GFS2 file system.
- ☐ **Virtualization Client** Clients for installing and managing virtualization instances.
- ☐ **Virtualization Hypervisor** Smallest possible virtualization host installation.
- ☐ **Virtualization Tools** Tools for offline virtual image management.
- ☑ **Compatibility Libraries** Compatibility libraries for applications built on previous versions of CentOS Linux.
- ☑ **Development Tools** A basic development environment.
- ☐ **Security Tools** Security tools for integrity and trust verification.
- ☐ **Smart Card Support** Support for using smart card authentication.
- ☑ **System Administration Tools** Utilities useful in system administration.

2. Setup your root user password
3. Local user account (optional)

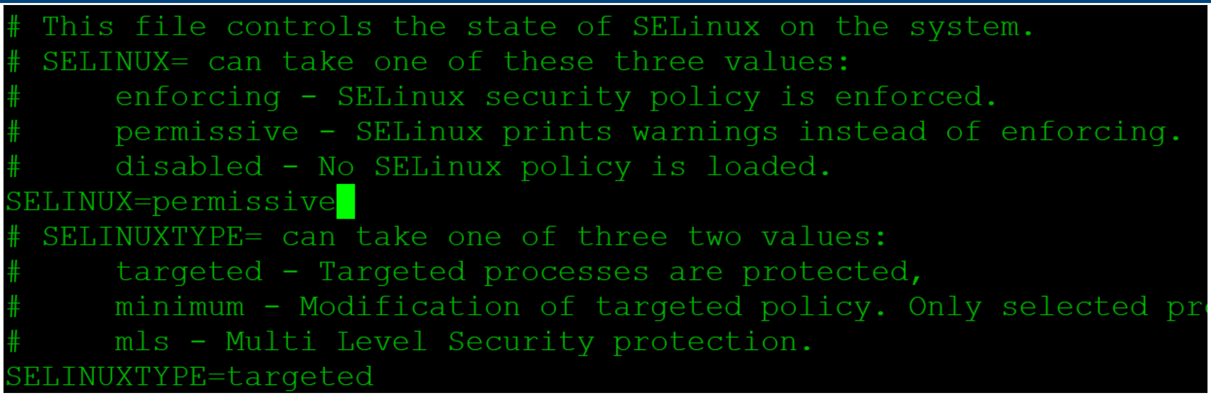
Post initial CentOS install:

4. Set the hostname in /etc/hostname
5. Set the hostname and IP in /etc/hosts (don't forget to include other lab server hostnames and IP addresses in local /etc/host file). Example /etc/host file:



```
[root@ansiblecontroller ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4 ansiblecontroller
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6 ansiblecontroller

192.168.11.101 ansibletarget01 ansibletarget01.kryptonet.local
192.168.11.102 ansibletarget02 ansibletarget02.kryptonet.local
192.168.11.82 isilon8104 isilon8104n1
192.168.11.83 isilon8104n2
192.168.11.84 isilon8104n3
```

6. Set the IP address of the server and ensure it is static IP to make troubleshooting easier (there are many ways to set a static IP address on CentOS, I typically use network manager GUI or nmtui / nmcli (depending on runlevel) or the more traditional method by editing the /etc/sysconfig/network-scripts/< *interface file* >.
7. Disable SELinux, or set to permissive mode /etc/selinux/config, example:



```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=permissive
# SELINUXTYPE= can take one of three two values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected pr
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

8. Disable linux firewall daemon
   a. Systemctl disable firewalld
   b. Systemctl stop firewalld
   c. Reboot your server and check the following information is correct:
      i. Hostname – use "hostname"
      ii. IP address – use "ip addr show" or "ifconfig"
      iii. Check SE Linux status – use "sestatus"
      iv. Check firewall daemon is not running – use "systemctl status firewalld"

Please Note: I do not recommend disabling Linux firewalld on production servers, however for the purposes of this lab, it'll make troubleshooting any connectivity issues a lot easier. **The same is true for SELinux.**

Install ansible on the Linux server (official site with ansible installation instructions:

9. Add the EPEL Repository:
   a. sudo yum -y update
   b. sudo yum -y install ansible

   check ansible version:

   c. ansible –version

   At the time of writing the version of ansible is: 2.7.5 (latest available version is 2.9.x). If your version is displaying an earlier version, please try installing/adding the epel repository using "sudo yum -y install epel-release" and repeat the above steps to update the installed ansible packages.

for additional information review the following site:
https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-the-control-machine

(Required) Install AWX server

AWX can be installed using the traditional method of package installation or installed using readily available docker images. For this guide I have chosen to install AWX using docker as it is by far the easiest way to install and all dependencies are pre-configured. It's also beneficial to cut your teeth on docker if you haven't already done so. After all, majority of workloads are moving to containerized platforms and micro services. (E.g. Docker, Kubernetes, Mesos, Cloud foundry / PCF, etc..).

Official AWX installation material is located on github:
https://github.com/ansible/awx/blob/devel/INSTALL.md

Installation process:

1. Install your Linux server as per "Ansible core (control node)"
2. Follow this above guide to setup your AWX software and environment.

Alternatively use google and search for installing AWX with docker (there are many guides available online). I found www.howtoforge.com included some very easy to follow ansible AWX installation guides.

(Optional) Microsoft Active Directory Server

Microsoft Active Directory and DNS is included in my lab environment as I have joined the Isilon cluster to my domain. However, it is not an essential component in our environment as local Isilon user accounts can be used. Nevertheless, if you have the time and resources available, I would encourage you to install AD 2016 and configure AD as an authentication provider for Isilon. This will provide a perfect foundation for future testing.

Please Note: The SMB share create playbook detailed in later sections requires MS AD authentication to successfully run.

Download evaluation version from:

https://www.microsoft.com/en-us/evalcenter/evaluate-windows-server-2016?filetype=ISO

Alternatively perform a web search for Microsoft Evaluation Center Windows Server 2016 (windows 2019 also supported)

Basic instructions

- Create a windows 2016 server
- Prompt the server to a domain controller
    - Active directory
    - DNS
- Add the DNS records for all hosts within the lab environment

Isilon – Configure Isilon for Ansible connectivity

Ansible requires a user account on each target node to execute tasks.

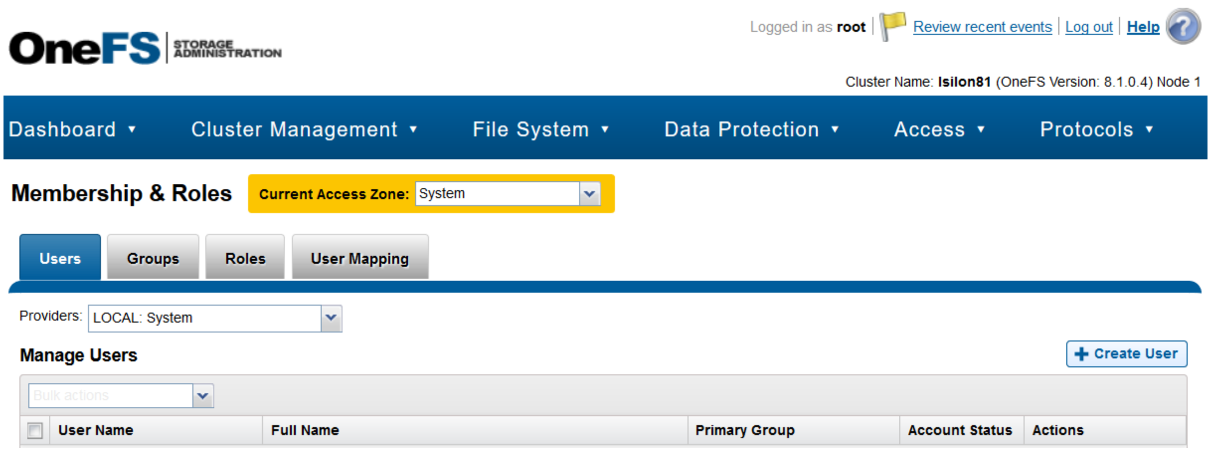
In this section we are going to:

1. setup a dedicated ansible user account on Isilon and add the API role to the new ansible user account to ensure we can access the Isilon API.
2. demonstrate how to setup SSH with password-less access using private and public keys.

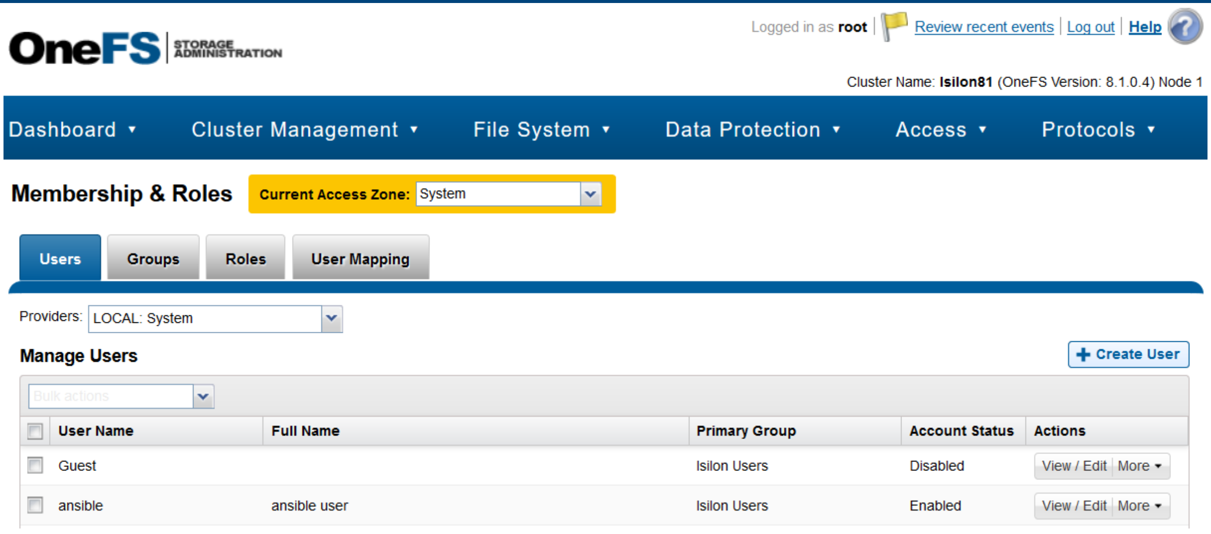
Please note: User account and password information for each ansible target node can be stored in the ansible inventory (we will show an example of this during the ansible control and AWX setup) however better security practices recommend using ssh private and public keys for target node access.

Use an internet browser (Firefox/Chrome) to access the Isilon GUI.

1. Login with "root" and select "access" on the OneFS menu bar.
2. Ensure Current Access Zone is set to "System"
3. In the Users sub tab change the provider in the drop down to: "LOCAL:System"



4. Click create user
   a. Username "ansible"
   b. Full Name "ansible user"
   c. Add primary group "Isilon Users"
   d. Set preferred shell "/bin/sh"


Logged in as **root** | Review recent events | Log out | Help

Cluster Name: **Isilon81** (OneFS Version: 8.1.0.4) Node 1

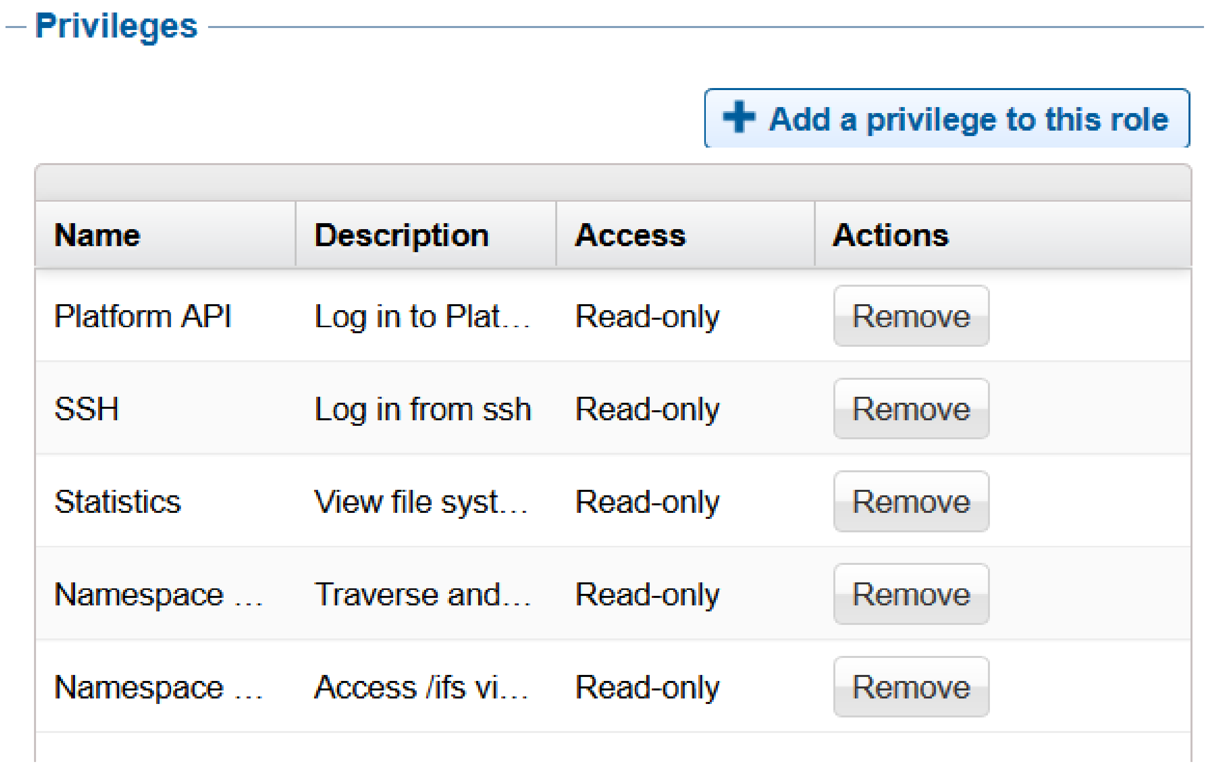
Membership & Roles Current Access Zone: System

Providers: LOCAL: System

Manage Users

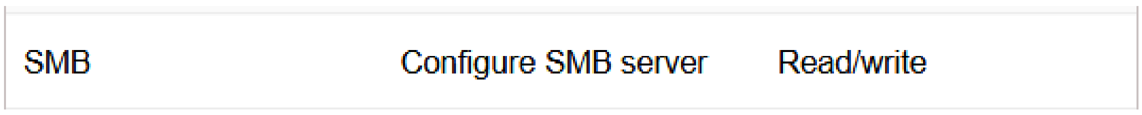
User Name	Full Name	Primary Group	Account Status	Actions
Guest		Isilon Users	Disabled	View / Edit More
ansible	ansible user	Isilon Users	Enabled	View / Edit More

5. Create Isilon API privileges role and add the newly created "ansible user" to the role
   a. Select create a Role
   b. Give the role a new name "API_PRIV"
   c. Add the new local ansible user to the role
   d. Add the "log in to platform API and WebUI" role


Privileges

Name	Description	Access	Actions
Platform API	Log in to Plat…	Read-only	Remove
SSH	Log in from ssh	Read-only	Remove
Statistics	View file syst…	Read-only	Remove
Namespace …	Traverse and…	Read-only	Remove
Namespace …	Access /ifs vi…	Read-only	Remove

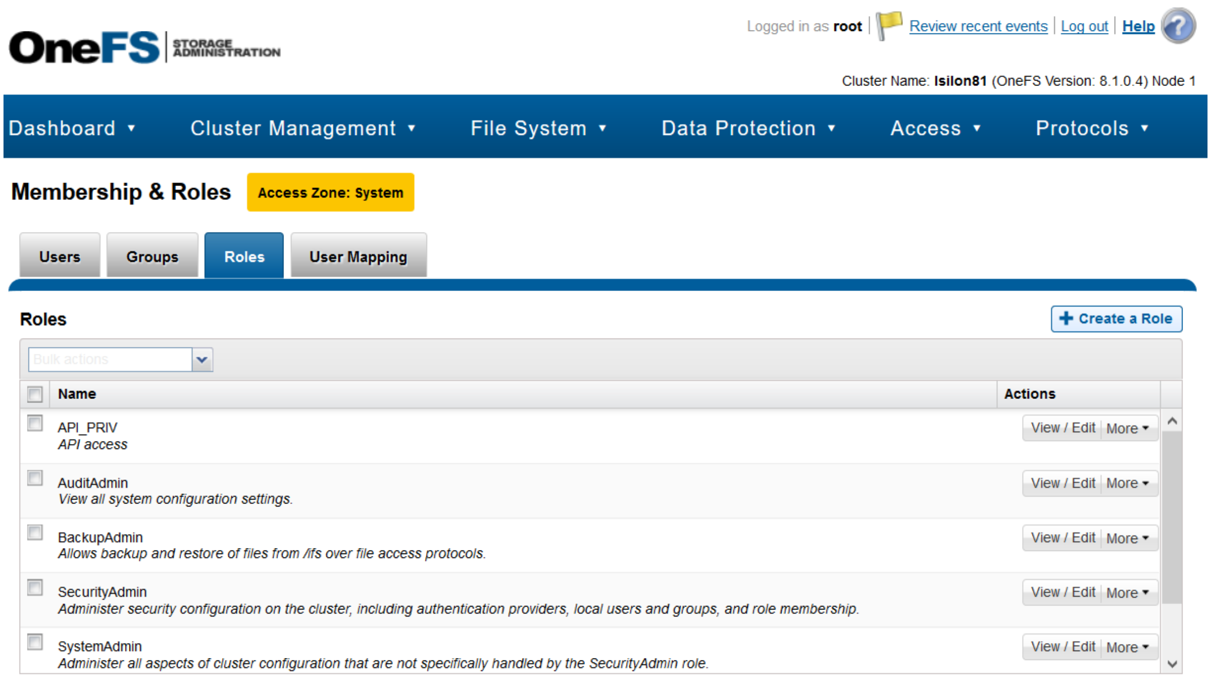
   e. PLEASE NOTE: Onefs 8.2.0 requires an additional "read/write" privilege to perform SMB action:



SMB	Configure SMB server	Read/write

In addition, SNAPSHOT role is also required in Onefs 8.2.0

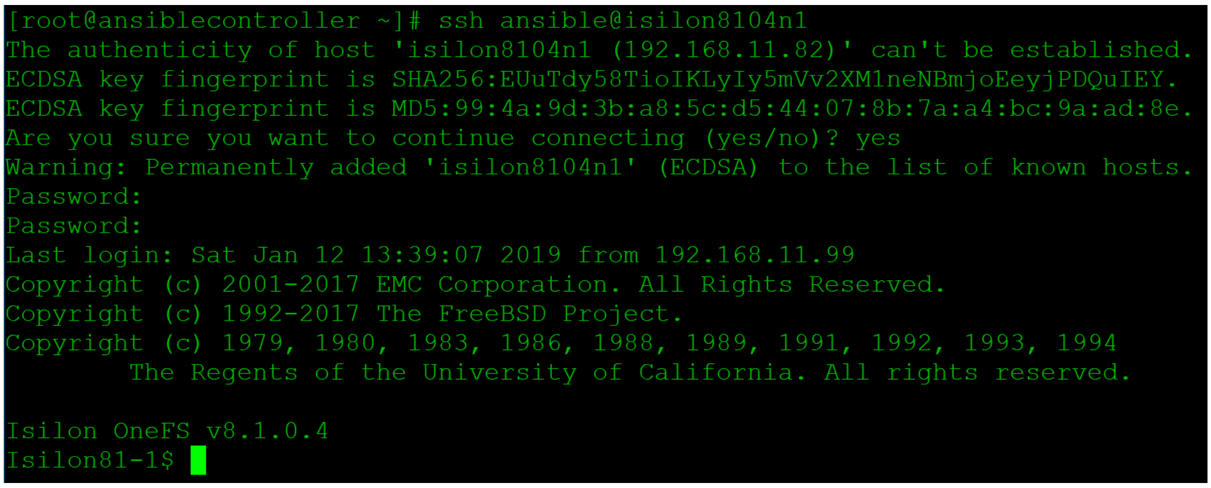
f. (Optional) add Statistics and SSH privileges to allow querying Isilon performance metrics from the API and to ensure SSH access is available to members of the role.
g. (Recommended) add RESTful API NameSpace (RAN) roles to ensure the "ansible" users has sufficient permission to interact with directory (containers) and files (objects) on the OneFS file system.
h. Create new role



	Name	Actions
☐	API_PRIV *API access*	View / Edit | More ▾
☐	AuditAdmin *View all system configuration settings.*	View / Edit | More ▾
☐	BackupAdmin *Allows backup and restore of files from /ifs over file access protocols.*	View / Edit | More ▾
☐	SecurityAdmin *Administer security configuration on the cluster, including authentication providers, local users and groups, and role membership.*	View / Edit | More ▾
☐	SystemAdmin *Administer all aspects of cluster configuration that are not specifically handled by the SecurityAdmin role.*	View / Edit | More ▾

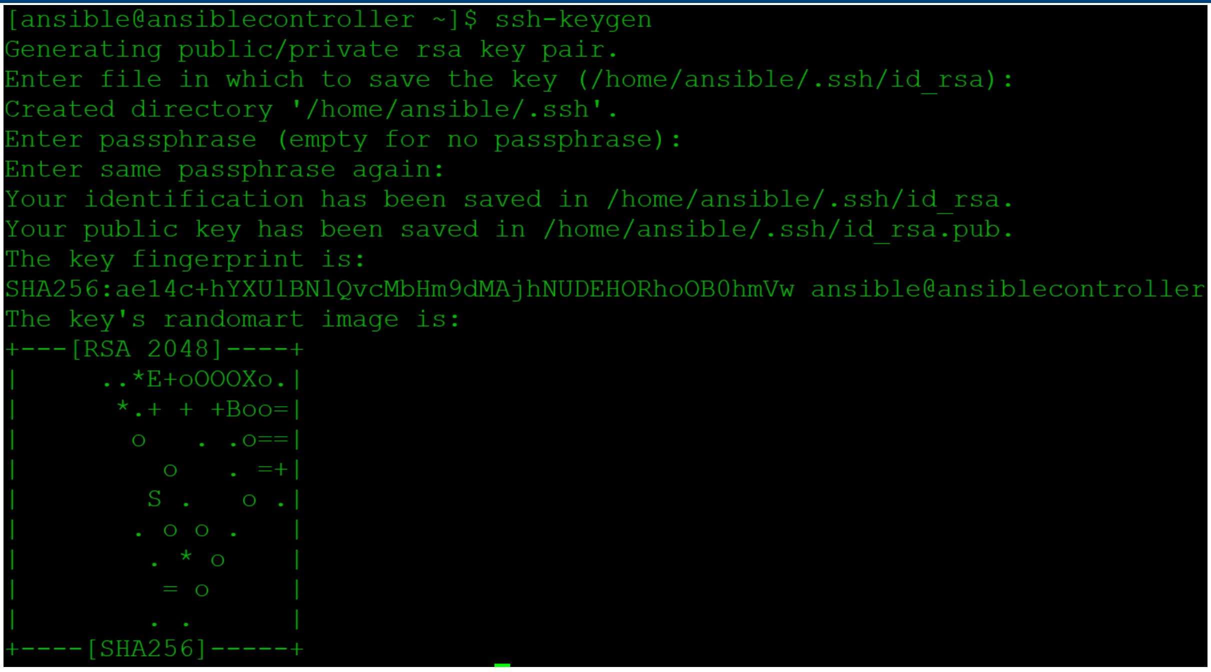
6. (Lab only, optional step) To ensure the ansible user has all required Isilon privileges for this lab, please modify the SecurityAdmin and SystemAdmin by adding the newly created ansible user to these roles.

Logout and ensure the new ansible user can log in via the Isilon GUI and CLI (SSH). for CLI connectivity use your favourite terminal app on windows, or command terminal from linux. For example:



```
[root@ansiblecontroller ~]# ssh ansible@isilon8104n1
The authenticity of host 'isilon8104n1 (192.168.11.82)' can't be established.
ECDSA key fingerprint is SHA256:EUuTdy58TioIKLyIy5mVv2XM1neNBmjoEeyjPDQuIEY.
ECDSA key fingerprint is MD5:99:4a:9d:3b:a8:5c:d5:44:07:8b:7a:a4:bc:9a:ad:8e.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'isilon8104n1' (ECDSA) to the list of known hosts.
Password:
Password:
Last login: Sat Jan 12 13:39:07 2019 from 192.168.11.99
Copyright (c) 2001-2017 EMC Corporation. All Rights Reserved.
Copyright (c) 1992-2017 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.

Isilon OneFS v8.1.0.4
Isilon81-1$
```

7. Create SSH private and public key pair
8. I'm using the Ansible Controller host to setup the ssh keys.
   a. ssh-keygen (do not enter a passphrase)



```
[ansible@ansiblecontroller ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa):
Created directory '/home/ansible/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ansible/.ssh/id_rsa.
Your public key has been saved in /home/ansible/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:ae14c+hYXUlBNlQvcMbHm9dMAjhNUDEHORhoOB0hmVw ansible@ansiblecontroller
The key's randomart image is:
+---[RSA 2048]----+
|      ..*E+oOOOXo.|
|       *.+ + +Boo=|
|        o    . .o==|
|          o    . =+|
|         S .    o .|
|        . o o .    |
|         . * o     |
|          = o      |
|         . .       |
+----[SHA256]-----+
```

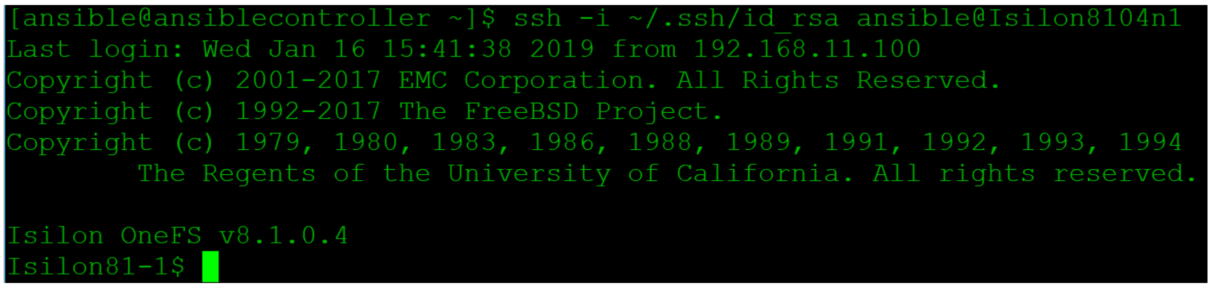
b. Copy the contents of ~/.ssh/id_rsa.pub to Isilon “ansible” home drive authorized_keys. For example:

c. Make sure the “.ssh” directory exists on Isilon before continuing.

```
scp ~/.ssh/id_rsa.pub ansible@Isilon8104n1:~/.ssh/authorized_keys
```

As an alternative copying and pasting the key using cat ~/.ssh/id_rsa.pub and coping directly into the authorized_keys on Isilon would work equally as well.

Test ssh key login:



```
[ansible@ansiblecontroller ~]$ ssh -i ~/.ssh/id_rsa ansible@Isilon8104n1
Last login: Wed Jan 16 15:41:38 2019 from 192.168.11.100
Copyright (c) 2001-2017 EMC Corporation. All Rights Reserved.
Copyright (c) 1992-2017 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.

Isilon OneFS v8.1.0.4
Isilon81-1$
```

If successful, your Isilon is all setup for ansible 😊

Ansible – Configure ansible controller

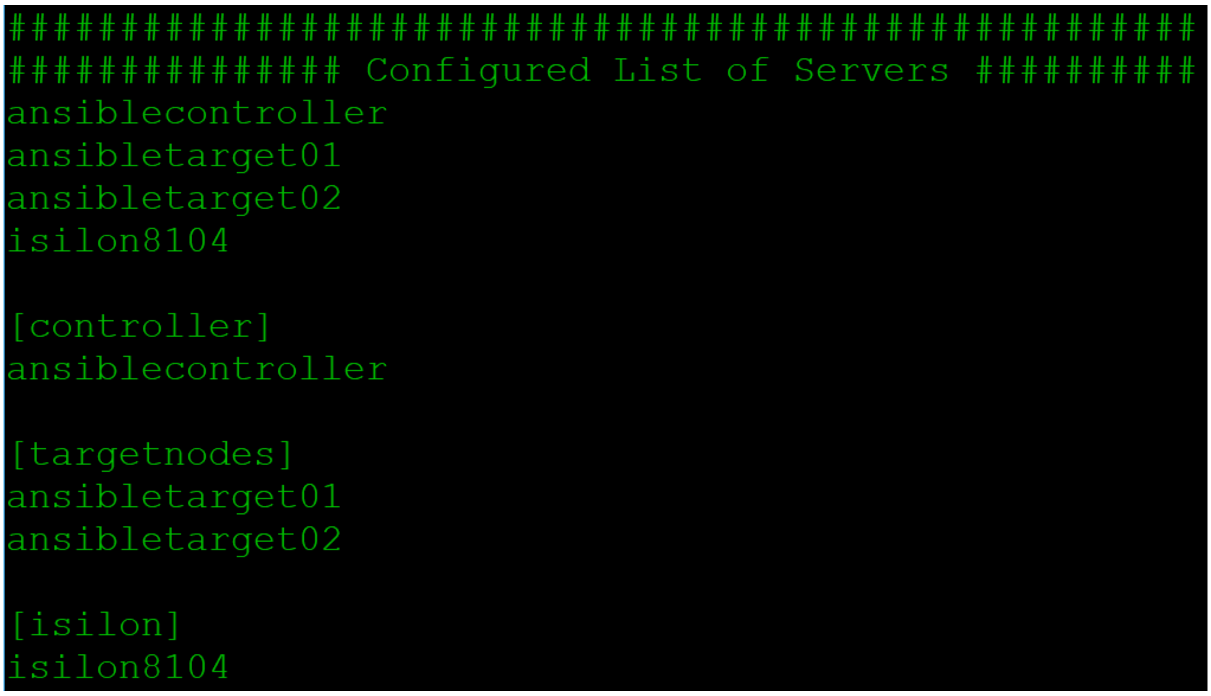
We previous installed the ansible software on the ansible controller host. Since we are creating a very simple ansible environment for demonstration purposes there are only a couple short tasks to complete.

In this section we will complete the following tasks:

1. Setup your ansible inventory file
2. Ensure DNS is working correctly and test ansible connectivity to defined target nodes

Add your ansible hosts to the ansible inventory file. (default location: /etc/ansible/hosts)

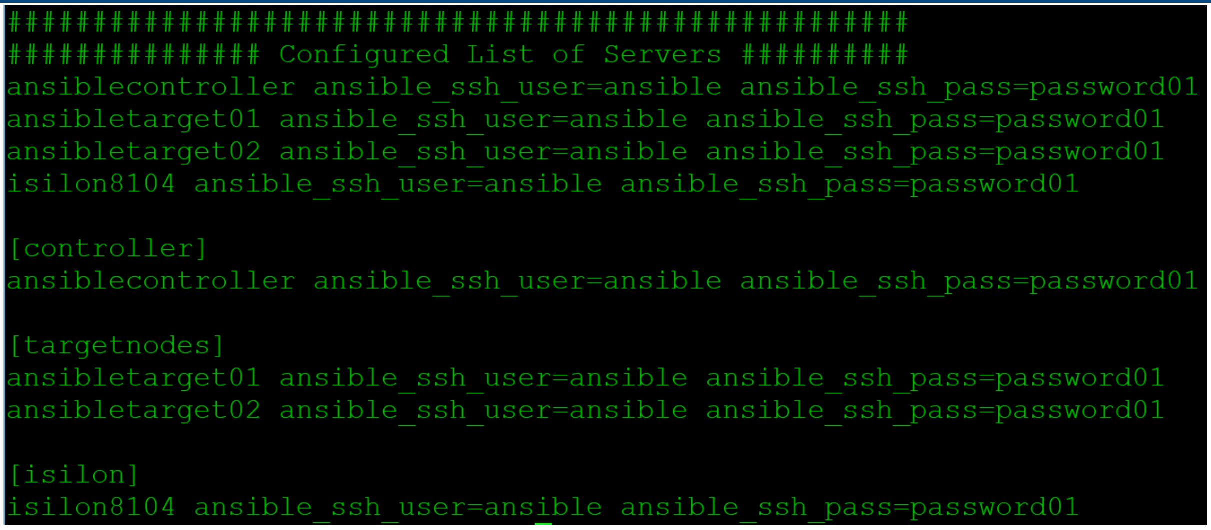
Ansible inventory file uses DNS services to contact each of the ansible target nodes; hence please ensure it is correct and up to date. Below is a copy of my ansible file.



```
#####################################################
############### Configured List of Servers ##########
ansiblecontroller
ansibletarget01
ansibletarget02
isilon8104

[controller]
ansiblecontroller

[targetnodes]
ansibletarget01
ansibletarget02

[isilon]
isilon8104
```

Notice some entries have square brackets, this how multiple ansible target nodes are grouped together. For example, running an ansible command against “ansibletarget01” will run that action against a single target node, while running that action against the group name of “targetnodes” will run the action against each node list under “[targetnodes]” in the inventory file.

Also, usernames and passwords for each ansible targetnode can be included in the inventory file as follows:



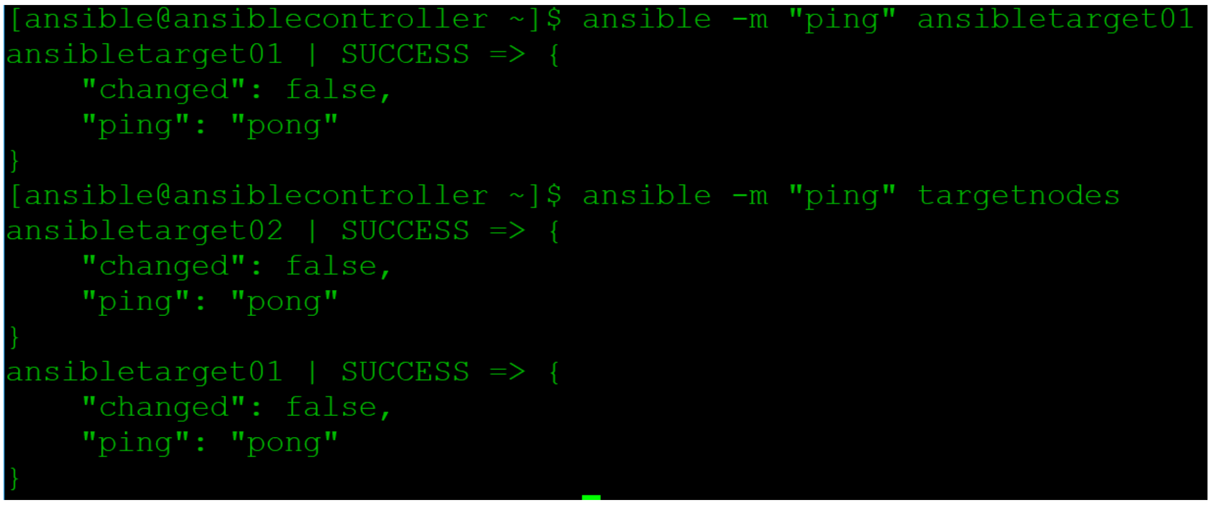
```
#####################################################
############### Configured List of Servers ##########
ansiblecontroller ansible_ssh_user=ansible ansible_ssh_pass=password01
ansibletarget01 ansible_ssh_user=ansible ansible_ssh_pass=password01
ansibletarget02 ansible_ssh_user=ansible ansible_ssh_pass=password01
isilon8104 ansible_ssh_user=ansible ansible_ssh_pass=password01

[controller]
ansiblecontroller ansible_ssh_user=ansible ansible_ssh_pass=password01

[targetnodes]
ansibletarget01 ansible_ssh_user=ansible ansible_ssh_pass=password01
ansibletarget02 ansible_ssh_user=ansible ansible_ssh_pass=password01

[isilon]
isilon8104 ansible_ssh_user=ansible ansible_ssh_pass=password01
```

Please Note: clear text passwords is not recommended, however if you have not setup ssh keys or wish to use the inventory options for username and passwords in production then I recommend looking into “ansible vault”.

Ansible connectivity checks

Ansible provide a native module called “ping” which checks ansible connectivity to the desired target nodes. Please note it is not running an operating system / network ping (ICMP) connection. It essentially checks that the Ansible controller can communicate to the targets via the configured protocol, default is SSH for Linux / Unix and WinRM for windows hosts.

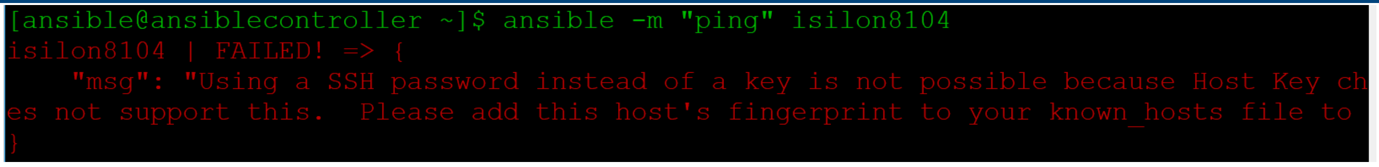
Please note: I have deployed two additional linux nodes to act as dummy ansible targetnodes. I also added a local “ansible” user to ansibletarget01 and ansibletarget02, including editing the sudoers file with “visudo” and appending “ansible ALL=(ALL) NOPASSWD: ALL” to the last line of the file.



```
[ansible@ansiblecontroller ~]$ ansible -m "ping" ansibletarget01
ansibletarget01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
[ansible@ansiblecontroller ~]$ ansible -m "ping" targetnodes
ansibletarget02 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
ansibletarget01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

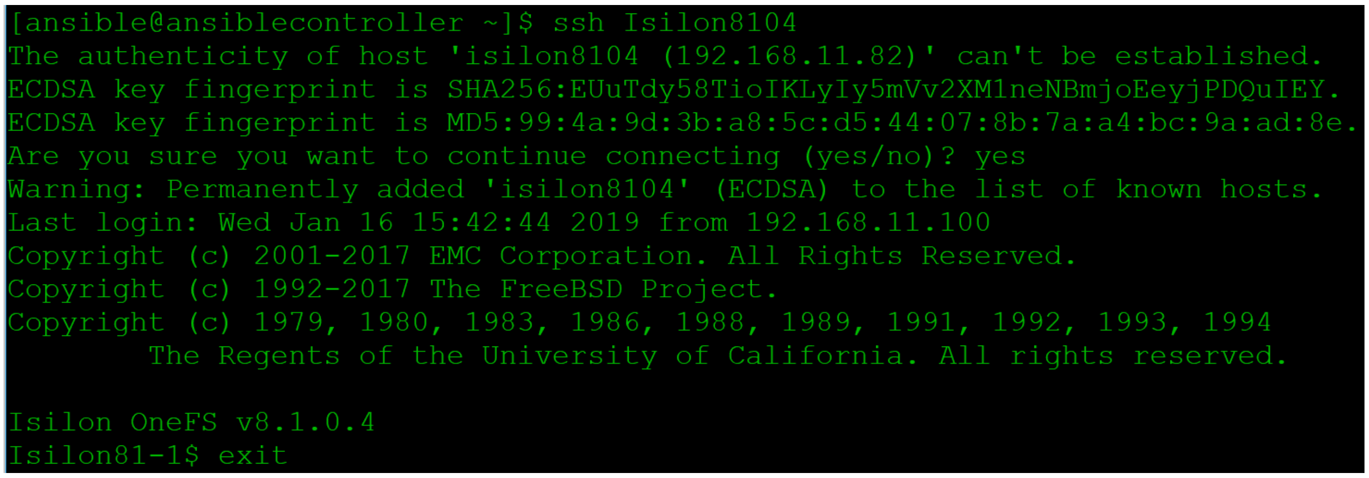
If you see a response of “SUCCESS” and a “ping”: “pong” response form the ping module then we are all good to go.

If you encounter a problem, make sure to check your Isilon connectivity as per the below example:



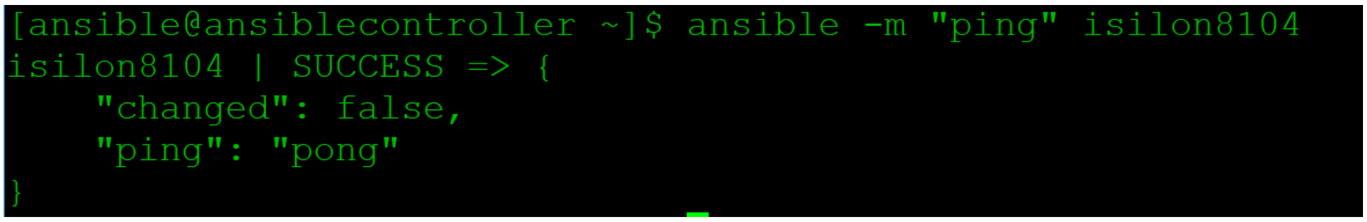
```
[ansible@ansiblecontroller ~]$ ansible -m "ping" isilon8104
isilon8104 | FAILED! => {
    "msg": "Using a SSH password instead of a key is not possible because Host Key ch
es not support this.  Please add this host's fingerprint to your known_hosts file to
}
```

If you see an error talking about fingerprints that you need to log in and add the target machines SSH fingerprint to the Ansible controller. Resolve by logging in with ssh directly and save/update the host key.



```
[ansible@ansiblecontroller ~]$ ssh Isilon8104
The authenticity of host 'isilon8104 (192.168.11.82)' can't be established.
ECDSA key fingerprint is SHA256:EUuTdy58TioIKLyIy5mVv2XM1neNBmjoEeyjPDQuIEY.
ECDSA key fingerprint is MD5:99:4a:9d:3b:a8:5c:d5:44:07:8b:7a:a4:bc:9a:ad:8e.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'isilon8104' (ECDSA) to the list of known hosts.
Last login: Wed Jan 16 15:42:44 2019 from 192.168.11.100
Copyright (c) 2001-2017 EMC Corporation. All Rights Reserved.
Copyright (c) 1992-2017 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.

Isilon OneFS v8.1.0.4
Isilon81-1$ exit
```

Then retry the ansible ping test



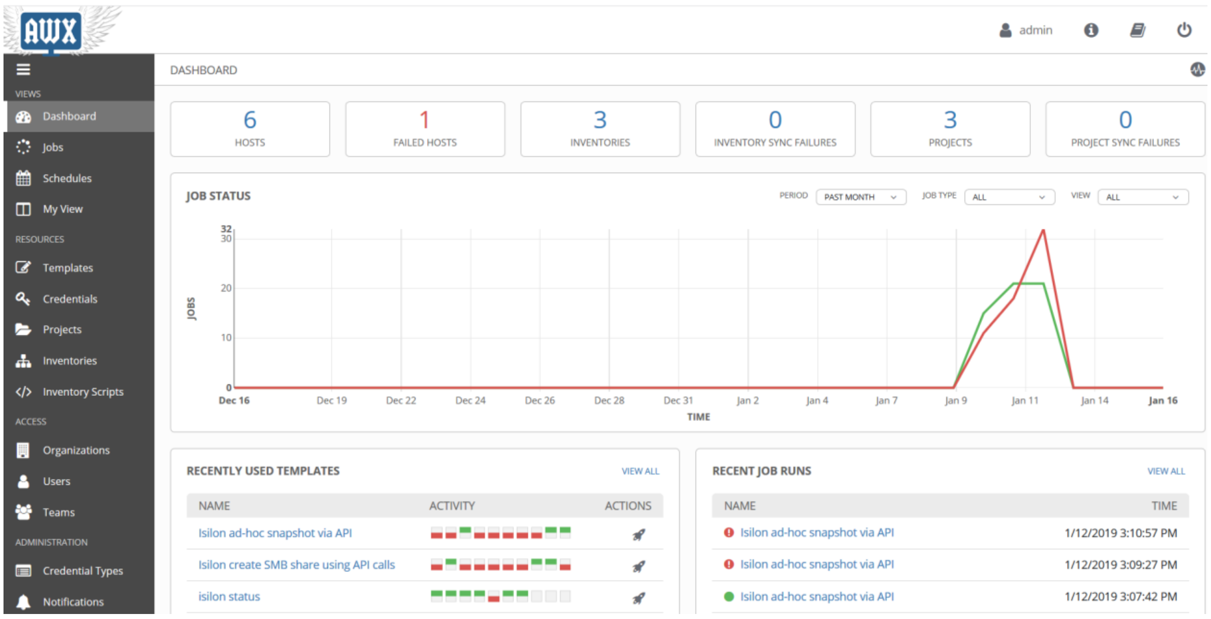
```
[ansible@ansiblecontroller ~]$ ansible -m "ping" isilon8104
isilon8104 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Success 😊

AWX – Configure via webUI

At this stage your AWX environment should be installed, running and you should be able to login via the URL. In my environment I login to AWX using the http://192.168.11.99 (This is the IP address for my AWX linux host).

Login with the admin user and you should see dashboard like the following.



6	1	3	0	3	0
HOSTS	FAILED HOSTS	INVENTORIES	INVENTORY SYNC FAILURES	PROJECTS	PROJECT SYNC FAILURES

RECENTLY USED TEMPLATES — VIEW ALL

NAME	ACTIVITY	ACTIONS
Isilon ad-hoc snapshot via API		
Isilon create SMB share using API calls		
isilon status		

RECENT JOB RUNS — VIEW ALL

NAME	TIME
Isilon ad-hoc snapshot via API	1/12/2019 3:10:57 PM
Isilon ad-hoc snapshot via API	1/12/2019 3:09:27 PM
Isilon ad-hoc snapshot via API	1/12/2019 3:07:42 PM

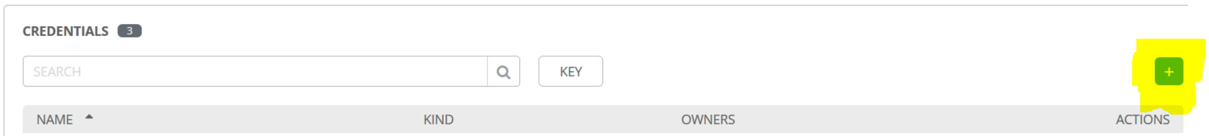
You'll notice my dashboard has 3x projects, 6x hosts and 3x inventories configured. We'll configure your AWX environment in a similar fashion.

In this section we will complete the following tasks:

1. Add a new credential to the AWX service based on the local Isilon and linux ansible user setup previously.
2. Add a new inventory and create an Isilon host
3. (optional) create a new inventory for other AWX target nodes

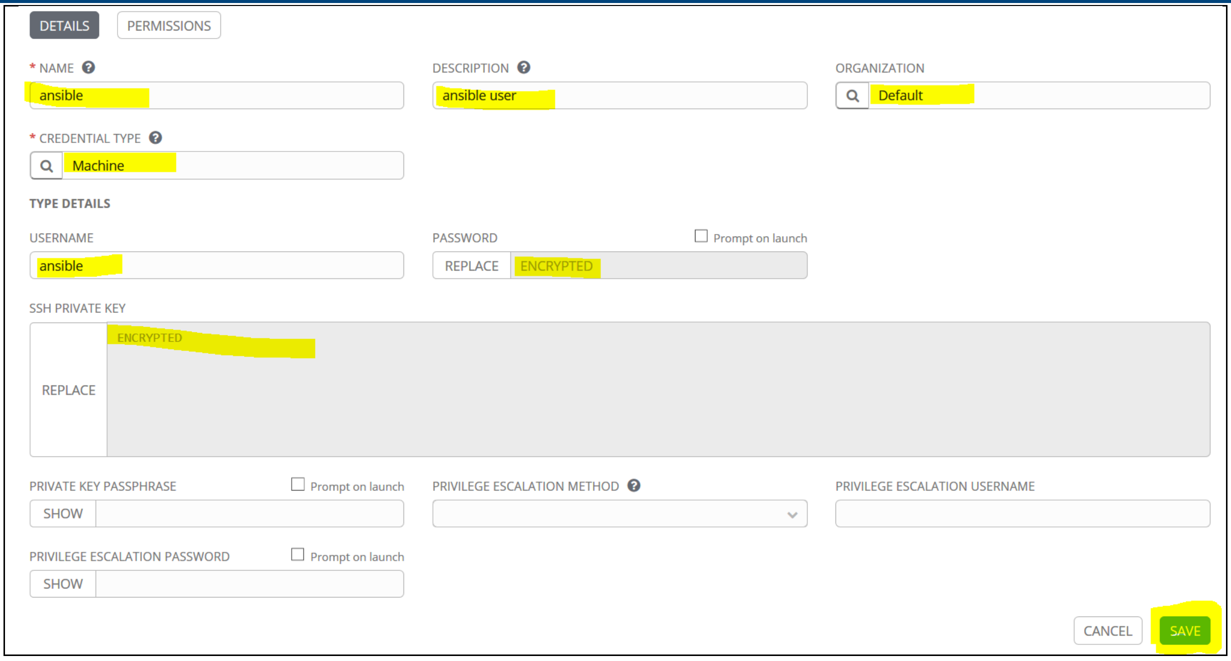
Create new credential for connecting to Isilon

Use the side menu and click on "credentials" then client the "+" sign to add a new user.



There are several items to fill in when creating new AWX users.

1. Name
2. Description
3. Select AWX Organisation (for this lab, I plan to use the default organization)
4. Credential Type (we are using a local user on each target node; hence we select the "Machine" type)
5. Username (this is the Linux / Isilon username we configured in previous sections)
6. Password or SSH **private** key (this is the ssh that does not include the ".pub" extension)
7. Save the user configuration



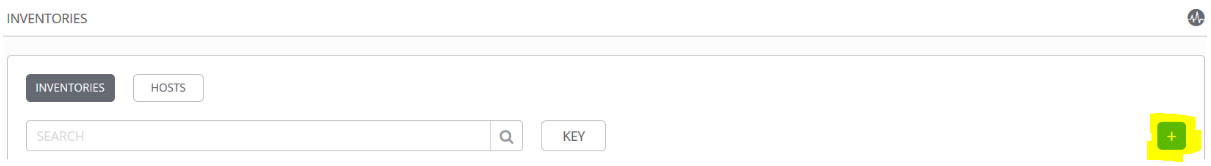
* NAME: ansible
DESCRIPTION: ansible user
ORGANIZATION: Default
* CREDENTIAL TYPE: Machine

TYPE DETAILS

USERNAME: ansible
PASSWORD: ENCRYPTED ☐ Prompt on launch
SSH PRIVATE KEY: ENCRYPTED
PRIVATE KEY PASSPHRASE: ☐ Prompt on launch
PRIVILEGE ESCALATION METHOD:
PRIVILEGE ESCALATION USERNAME:
PRIVILEGE ESCALATION PASSWORD: ☐ Prompt on launch

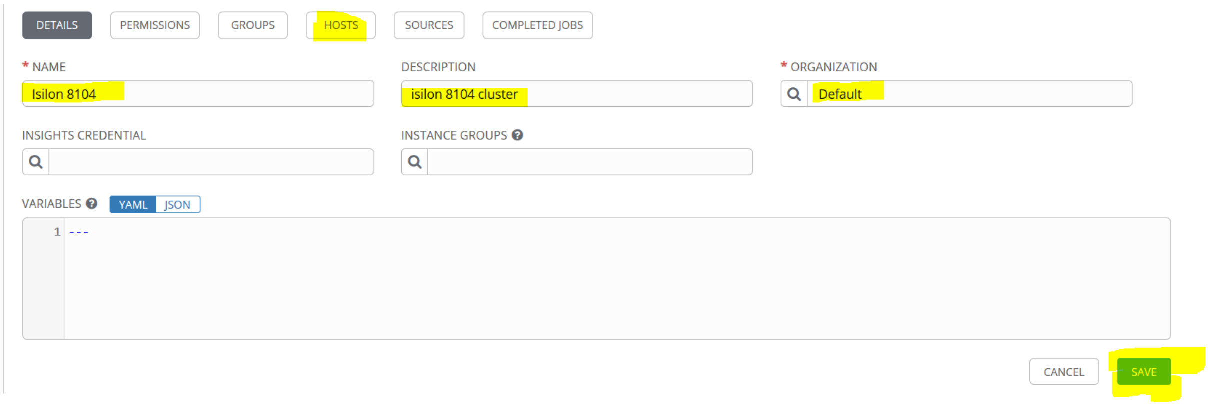
Create new AWX inventory and host entry for Isilon

Use the side menu and click on “inventories” then client the “+” sign and select inventory to add a new Isilon inventory.



Fill in new Isilon inventory details:

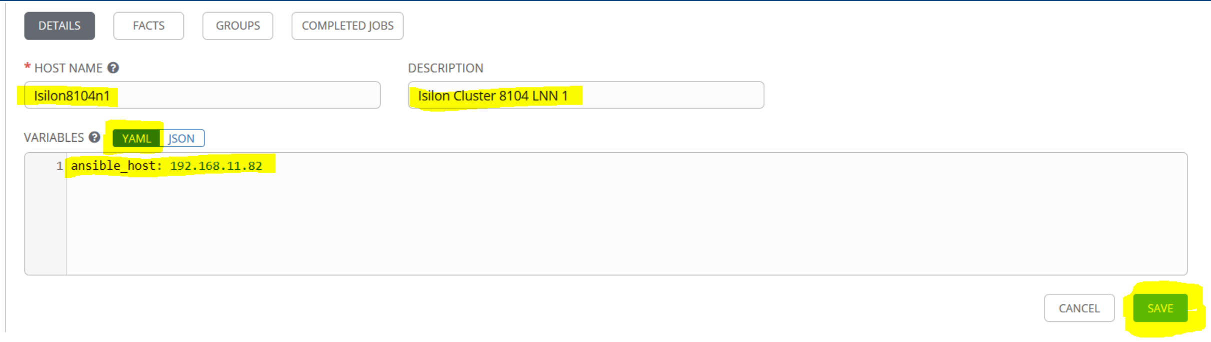
1. Name
2. Description
3. Select AWX organization
4. Click save



* NAME: Isilon 8104
DESCRIPTION: isilon 8104 cluster
* ORGANIZATION: Default
INSIGHTS CREDENTIAL:
INSTANCE GROUPS:
VARIABLES: YAML JSON

```
---
```

5. Click on “Hosts”
6. Add host details and save



* HOST NAME: Isilon8104n1

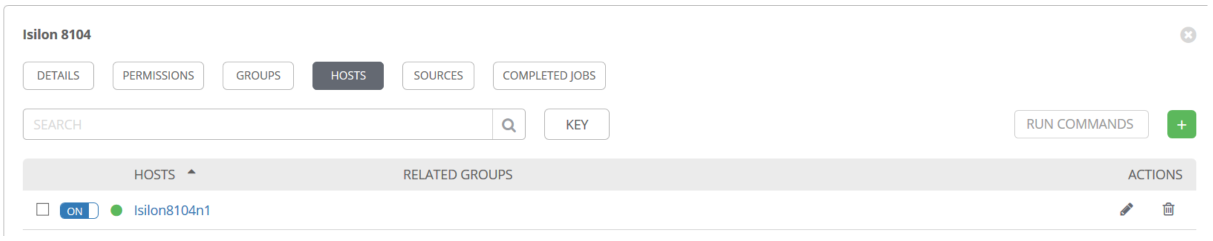
DESCRIPTION: Isilon Cluster 8104 LNN 1

VARIABLES: YAML JSON

```
ansible_host: 192.168.11.82
```

(Optional) - Repeat the above steps for each Isilon node in your cluster.

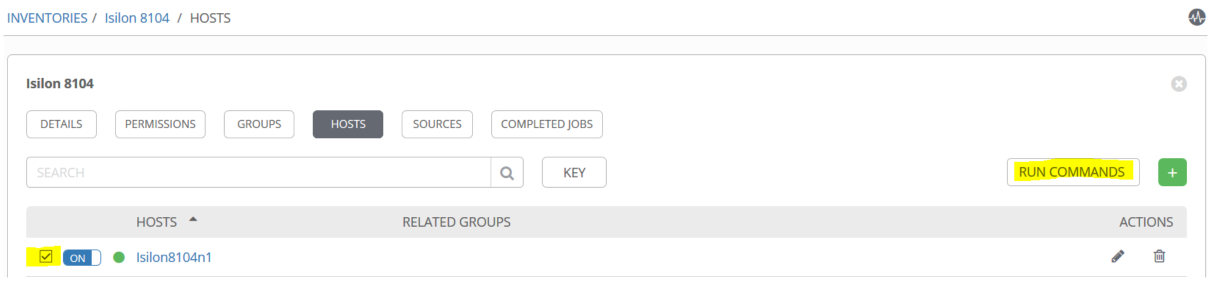
Please note: When running ansible jobs against an Isilon cluster, ensure all commands run against a single node in the cluster to prevent the command running multiple times on the same cluster.



HOSTS	RELATED GROUPS	ACTIONS
☐ ON ● Isilon8104n1		

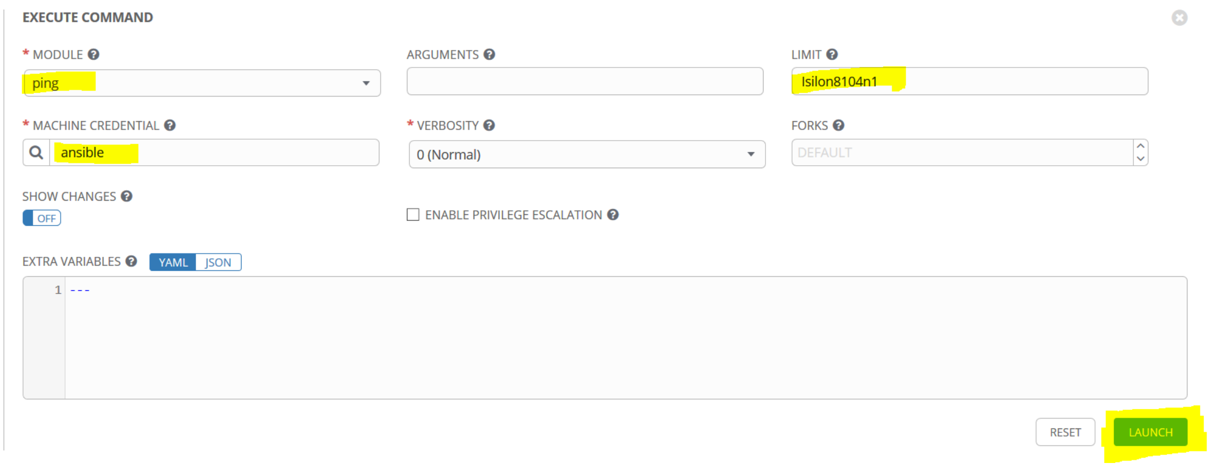
Check our configuration is working

Navigate back to "inventories" -> select "Isilon 8104" -> select "hosts" and select the checkbox next to the desired an Isilon host name. click "run commands"



HOSTS	RELATED GROUPS	ACTIONS
☑ ON ● Isilon8104n1		

1. Select module name "ping"
2. Ensure limit is set to only one Isilon node
3. Set Machine Credential to "ansible"



* MODULE: ping

ARGUMENTS:

LIMIT: Isilon8104n1

* MACHINE CREDENTIAL: ansible

* VERBOSITY: 0 (Normal)

FORKS: DEFAULT

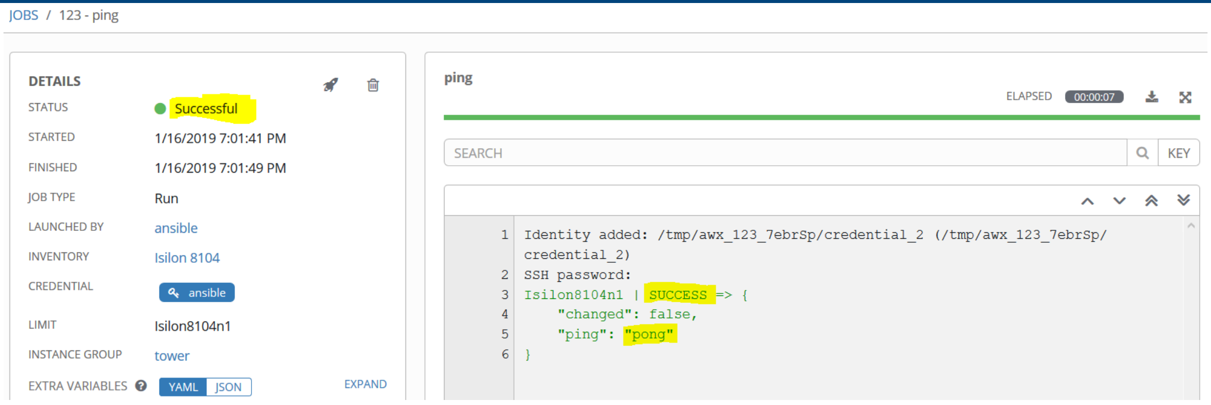
SHOW CHANGES: OFF

☐ ENABLE PRIVILEGE ESCALATION

EXTRA VARIABLES: YAML JSON

```
---
```

4. Click Launch



DETAILS	
STATUS	Successful
STARTED	1/16/2019 7:01:41 PM
FINISHED	1/16/2019 7:01:49 PM
JOB TYPE	Run
LAUNCHED BY	ansible
INVENTORY	Isilon 8104
CREDENTIAL	ansible
LIMIT	Isilon8104n1
INSTANCE GROUP	tower
EXTRA VARIABLES	YAML JSON

```
Identity added: /tmp/awx_123_7ebrSp/credential_2 (/tmp/awx_123_7ebrSp/
credential_2)
SSH password:
Isilon8104n1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

If the results show “successful” and you also see the “SUCCESS and PONG” response from the modules console screen, then AWX is configured to run tasks on Isilon.

(Optional) – Add a new inventory for ansible Linux hosts

Repeat the steps above to add additional linux inventory and linux target nodes to the AWX environment as required.

Ansible terminology and playbook YAML file structure

Ansible terminology:

- Control node – where ansible or AWX is installed
- Target node – The recipients of Ansible / AWX module or playbook executions
- Module – the units of code ansible executes, each module has a particular use
- Playbook – ordered list of tasks, saved so you can run those tasks in that order repeatably
- YAML – Ansible playbooks are written in YAML
- Ansible inventory – by default a file that has a list of target nodes and groups

Ansible Modules:

- Command – run a simple command on the target node
- Shell – same as "command" module but runs the command via a shell (/bin/sh) on the target node
- Script – run a user defined script on the targeted node
- Fetch – copy files from the target node to the ansible controller
- URI – intelligently communicate with HTTP/S services (REST API)

Playbook YAML file structure

Ansible playbooks are written in YAML.

Whitespace indentation is used to denote structure however tab characters are never allowed.

A YAML file typically starts with three --- and ends with three … (the three periods at the end of a YAML file are not enforced).

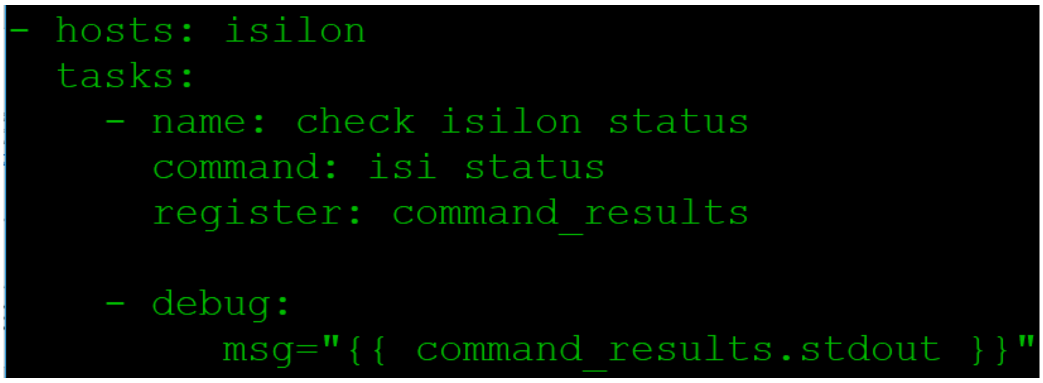
Also, Ansible does not strictly enforce starting a file with ---. As an example, the below playbook which calls the "command" module executes just fine on Ansible. An ansible file usually has an extension of yml or yaml.

Syntax reference

"-"	the dash next to host represents a play
"hosts:"	refers to the hostname in the ansible inventory.
"tasks:"	Each play contains a list of tasks. Tasks are executed in order, one at a time, against all machines matched by the host pattern
"- name:"	assigns a name to the task, the "-" defines an individual task
"Command:"	is the name of the module being called – (See ansible modules documentation)
"register:"	assigns a variable to the output of the task being executed

"- debug:"	Debug is an Ansible module that prints statements after execution. It is useful for debugging variables and other expressions. (See ansible modules documentation)
"msg="	subset of debug module and prints the output of the command_result variable
"{{ }}"	variable substitution in YAML are enclosed in "{{ }}" including the double quotes.
" "	Pay special attention to the whitespaces used for indentation, ensure equal whitespace indentation (typically 2 spaces per indentation)

Example playbook with 2 tasks:



```
- hosts: isilon
  tasks:
    - name: check isilon status
      command: isi status
      register: command_results

    - debug:
         msg="{{ command_results.stdout }}"
```

Please note: This guide is not intended to be a YAML tutorial or a detailed Ansible best practices guide. The intent is to demonstrate Isilon and ansible interoperability; Hence the guide will focus on the basics necessary to demonstrate Isilon and ansible interoperability.

Getting started with the Isilon CLI and API

The Ansible command and script modules within this guide use the Isilon command line; hence this section provides a basic overview of the Isilon CLI.

Requirement:

- Access to an Isilon cluster (or simulator)
- A validate Isilon user account with SSH privileges

I previously created a local Isilon system access zone user "ansible" with SSH, Platform API and System / Security admin rights to Isilon for demonstration purposes

Terminal Application:

- Windows users – use putty or your favourite alternative
- Linux / OSX – use built in terminal

Test various CLI commands on Isilon:

1. Use terminal app
2. SSH to Isilon
3. Execute test commands:
    - isi status
    - isi smb shares list
    - isi snapshot snapshots list
    - isi auth users view ansible
    - isi auth roles members list API_PRIV

Access the Isilon API

The Ansible URI module uses the Isilon API; hence this guide provides a basic overview of the Isilon API. Also for more information please visit the Isilon documentation page: https://community.emc.com/docs/DOC-75133

The Dell EMC Isilon system configuration API is a RESTful API and often referred to as the Platform API (or PAPI). There is also an Isilon data management API often referred to as RESTful Access to\ NameSpace or RAN for short.

Requirement:

- Access to the Isilon cluster (or simulator)
- A validate Isilon user account with API privileges

Please note: I created an Isilon local user called "ansible" in the system access zone with SSH, Platform API and SystemAdmin / Security admin rights to Isilon for demonstration purposes.

HTTP Command line tool:

- Windows users – download and install curl from the official site

- Linux / OSX – download and install curl

The Isilon API Basics:

The Isilon API is divided up into two primary functions

1. System configuration API: http://doc.isilon.com/onefs/8.1.0/api/03-ifs-br-system-config-api.htm
2. File System Access API: http://doc.isilon.com/onefs/8.1.0/api/04-ifs-br-file-system-access-apis.htm

The guide will primarily focus on the "System configuration API" referred to as the **platform** API (PAPI). In addition to the PAPI interface, we also authenticate with the Isilon RESTful API **namespace** (RAN) so that we can read, write and modify directory and files on the OneFS filesystem.

API authentication methods:

- When you send a HTTP request to the Isilon API, it verifies your username and password, either through HTTP Basic Authentication for single requests or through an established session cookie to a single node for multiple requests over a period of time. Our connections will be secured over SSL (HTTPS)

Explore the Isilon API using your favourite web browser:

- https://<Isilon_Node_IP>:8080/platform/
- Append "?describe" to the end of a particular object for a UNIX/Linux like man page description
    - For example: https://<Isilon_Node_IP>:8080/platform/5/protocols/smb/shares?describe

HTTP basic auth

HTTP basic auth to Isilon (examples taken from Linux bash shell)

The below command accesses the Isilon API to list our "ansible" user information

curl --insecure -X GET -H "Authorization: basic `echo -n ansible:password01 | base64`" https://<Isilon_Node_IP>:8080/platform/1/auth/users/ansible

Breaking down the above "curl" command

--insecure (or **-k**): make curl skip certificate validation, data is still sent securely

-X : Specifies the custom request method

-H : HTTP Extra header to include in the request when sending HTTP to a server

- Authorization: basic (This is used to send base64 encoded username and password)
- `echo -n ansible:password01 | base64` (linux CLI command pipe which prints out the "username:password" and encodes the output in base64)

URL Syntax : The http server target and corresponding platform/API-version/collection/objected….(see Isilon API docs)

- https://<Isilon_Node_IP>:8080/platform/1/auth/users

Please note: The following is true for HTTP basic auth and may represent a security risk in your environment:

1. The password is sent over the wire in base64 encoding (which can be easily converted to plaintext).
2. The password is sent repeatedly, for each request. (Larger attack window)
3. The password is cached by the web browser, at a minimum for the length of the window / process. (Can be silently reused by any other request to the server, e.g. CSRF attack).
4. The password may be stored permanently in the browser, if the user requests. (Same as previous point, in addition might be stolen by another user on a shared machine).

Using SSL (HTTPS) in the above commands ensures the username and password are NOT sent in plain text; it does not prevent security risks associated with points 2,3 & 4.

HTTP session cookie authentication

As an alternative to HTTP basic auth and a more secure authentication method, Isilon supports using X-CSRF session cookies (tokens) for authentication. Depending on the version of Isilon, you may have to update the http.conf file on your Isilon cluster to enable CSRF authentication. This support knowledge base article describes the process: https://support.emc.com/kb/517421

The below commands use a session cookie for authentication provided by the apache server on Isilon to list our "ansible" user information.

The first command requests a session cookie and the subsequent command performs an action using the session cookie.

1. curl --insecure -X POST -H "Content-Type: application/json" -c @/tmp/cookiefile -d @/ansible/auth.json https://192.168.11.82:8080/session/1/session
2. curl --insecure -X GET -H "X-CSRF-Token: `grep isicsrf /tmp/cookiefile | awk '{print $(NF)}'`" --referer https://192.168.11.82:8080 -b @/tmp/cookiefile https://192.168.11.82:8080/platform/1/auth/users/ansible

Let's break each of the above commands:

- --insecure : make curl skip certificate validation, data is still sent securely
- -X : Specifies the custom request method
- -H : HTTP Extra header to include in the request when sending HTTP to a server
    - Command 1) we are setting the application content type to json, this affects the "-d" provided data
    - Command 2) sets a custom header required for X-CSRF token and used linux shell commands to print out the csrf session id
- -c : cookie jar where to store the received session ids, the "@" character tells curl the following data is a file
- -d : file that stores initial authentication information, contents are in json:

```
{
    "username": "ansible",
    "password": "password01",
    "services": ["platform", "namespace"]
}
```

- --referrer : sends the referrer page information to the HTTP server , required by X-CSRF session token authentication
- -b : the session cookie, the “@” character tells curl the following data is a file
- URL Syntax - https://192.168.11.82:8080/platform/1/auth/users : The http server target and corresponding platform/API-version/collection/objected….(see Isilon API docs)

Please note: Its good practice is to clean up after the completed operation by deleting the session cookie

- curl --insecure -X DELETE -b @/tmp/cookiefile -H "X-CSRF-Token: `grep isicsrf /tmp/cookiefile | awk '{print $(NF)}'`" --referer https://192.168.11.82:8080 https://192.168.11.82:8080/session/1/session

Recommendations and notes when using API session tokens / cookies

- You can re-use the cookie for subsequent requests
- The cookie does have an expiration time and you may be required to re-authenticate if you have a very long running set of commands
- Regarding deleting session cookies, only clean up after everything is completed to avoid creating and destroying sessions too often

Ansible – running ansible play books against Isilon

Now that we have a working Isilon cluster, Ansible core controller, and Ansible AWX (Red Hat Tower) servers, we can finally get into demonstrating how integrate Dell EMC Isilon into your ansible environment.

We will cover executing ansible play books against an Isilon node and demonstrate using the ansible native "command", "script", "shell", "fetch" and "URI" modules.

Please note: I will use hard coded values in this section to keep the examples as straight forward as possible, however when we repeat the same tasks in AWX (Red Hat Ansible Tower) I will look at introducing variables into our playbooks.

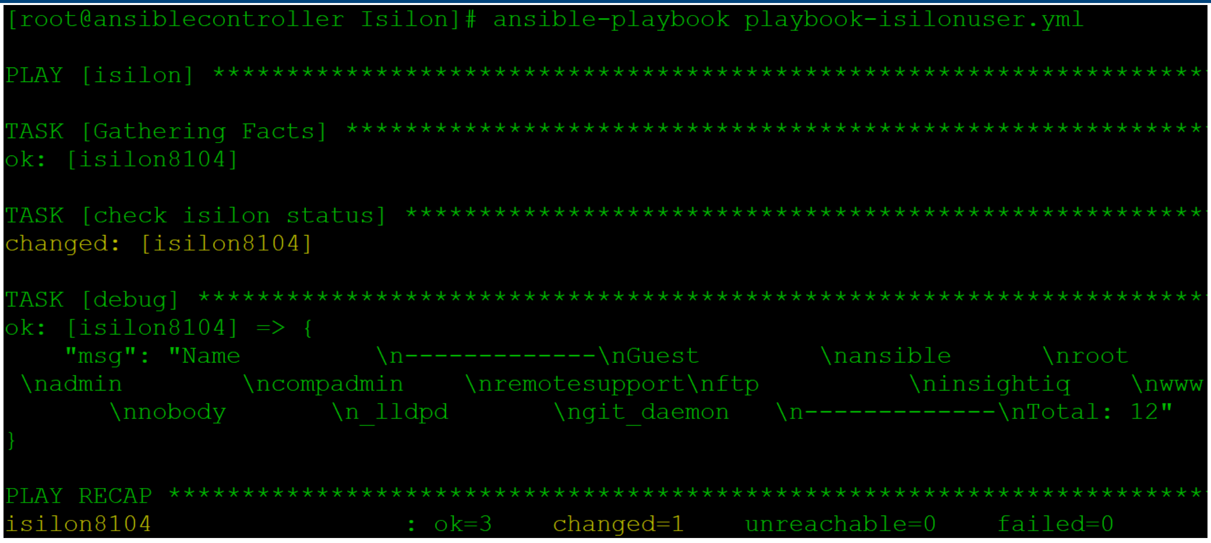
Playbook examples covered in this section:

1. Simple command module playbook that lists Isilon users
2. Simple command module playbook that executes "isi status" (Isilon CLI health status).
3. A more complicated playbook that uses the script module to execute a series of commands on an Isilon node
4. Create ansible playbook with URI module to create a new Isilon SMB share
5. Create ansible playbook with URI module to create a new Isilon Snapshot with an expire time

Example playbook using ansible native command modules to list Isilon users

Contents of playbook-isilonuser.yml

```
--------------------------------------------------------------------------
- hosts: isilon
  tasks:
    - name: check isilon status
      command: isi auth users list
      register: command_results
    - debug:
        msg="{{ command_results.stdout }}“
--------------------------------------------------------------------------
```



```
[root@ansiblecontroller Isilon]# ansible-playbook playbook-isilonuser.yml

PLAY [isilon] *******************************************************************

TASK [Gathering Facts] **********************************************************
ok: [isilon8104]

TASK [check isilon status] ******************************************************
changed: [isilon8104]

TASK [debug] ********************************************************************
ok: [isilon8104] => {
    "msg": "Name          \n-------------\nGuest        \nansible      \nroot
 \nadmin        \ncompadmin    \nremotesupport\nftp          \ninsightiq    \nwww
       \nnobody       \n_lldpd       \ngit_daemon   \n-------------\nTotal: 12"
}

PLAY RECAP **********************************************************************
isilon8104                 : ok=3    changed=1    unreachable=0    failed=0
```

Default ansible output is in JSON, Also, notice the “changed” state, the command module requires additional logic in the playbook to correctly identify the state

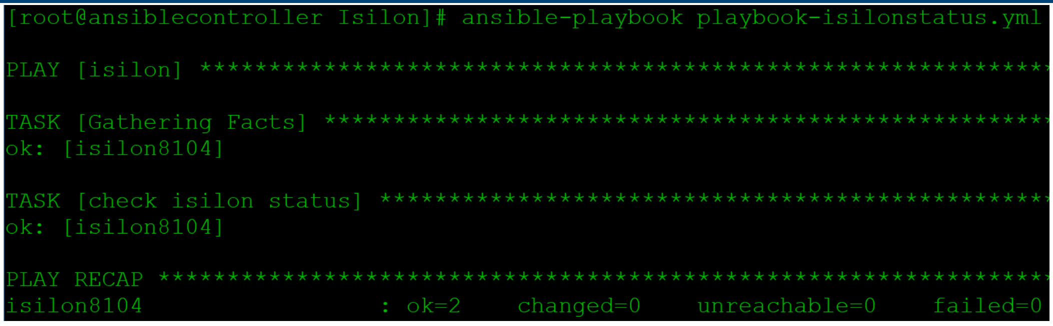
Example playbook using ansible native command modules to check Isilon status

This playbook demonstrates using the “when_changed” parameter to set conditions for changed events in Ansible tasks.

Contents of playbook-isilonstatus.yml

```
--------------------------------------------------------------------------------

- hosts: isilon

  tasks:

    - name: check isilon status

      command: isi status

      register: command_results

      changed_when: "'[  OK ]' not in command_results.stdout"

#   - debug:

#     msg="{{ command_results.stdout }}“

--------------------------------------------------------------------------------
```

Debug output has been commented out in the above playbook task, also please note the “when_changed” condition.



```
[root@ansiblecontroller Isilon]# ansible-playbook playbook-isilonstatus.yml

PLAY [isilon] ***************************************************************

TASK [Gathering Facts] ******************************************************
ok: [isilon8104]

TASK [check isilon status] **************************************************
ok: [isilon8104]

PLAY RECAP ******************************************************************
isilon8104                 : ok=2    changed=0    unreachable=0    failed=0
```

Example playbook using ansible native script and shell modules to check Isilon status

This play book demonstrates a play consisting of 3 tasks which transfers and executes the script on Isilon, identifies new files and copies files locally.

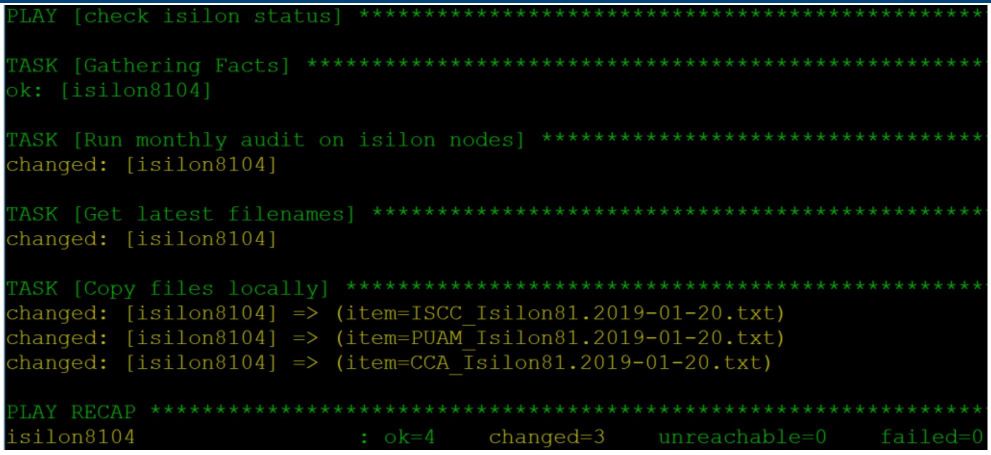
Contents of playbook-isilonENVaudit.yml

```
-----------------------------------------------------------------------------

- name: check isilon status

  hosts: isilon

  tasks:

    - name: Run monthly audit on isilon nodes

      script: /usr/local/playbooks/Isilon/playbook-isilonENVaudit.sh

      register: script_results

    - name: Get latest filenames

      shell: |

          (cd /ifs/data/Isilon_Support; find . –maxdepth 1 –type f) |

          grep `date +”%Y-%m-%d”` | cut –d’/’ –f2

      register: files_to_copy

    - name: Copy files locally

      fetch:

        src: /ifs/data/Isilon_Support/{{ item }}

        dest: /tmp/Isilon_Scripts/

        flat: yes

      with_items: “{{ files_to_copy.stdout_lines }}”

-----------------------------------------------------------------------------
```

Contents of playbook-isilonENVaudit.sh

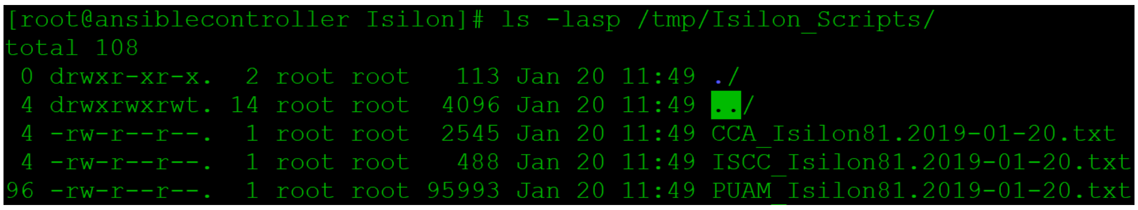
```
--------------------------------------------------------------------------
#!/bin/bash
### Variables ###
TOdate=$(date +"%Y-%m-%d")
CLUSTname=$(echo `hostname` | cut -d'-' -f1)
SCRIPThome=/ifs/data/Isilon_Support
cd /ifs/data/Isilon_Support
(
last
) | perl -p -e 's/\n/\r\n/' | tee ${SCRIPThome}/PUAM_${CLUSTname}.${TOdate}.txt
(
isi auth ldap list --verbose
isi auth local list --verbose
) | perl -p -e 's/\n/\r\n/' | tee ${SCRIPThome}/ISCC_${CLUSTname}.${TOdate}.txt
(
hostname
cat /etc/resolv.conf
isi snmp settings view
isi email settings view
isi network interfaces list
isi network groupnets list --verbose
isi network subnets list
isi auth ads list
isi version --verbose
isi auth settings krb5 view
### Not necessary ### isi zone zones list --verbose
) | perl -p -e 's/\n/\r\n/' | tee ${SCRIPThome}/CCA_${CLUSTname}.${TOdate}.txt
--------------------------------------------------------------------------
```

The play book run:



```
PLAY [check isilon status] ***************************************************

TASK [Gathering Facts] *******************************************************
ok: [isilon8104]

TASK [Run monthly audit on isilon nodes] *************************************
changed: [isilon8104]

TASK [Get latest filenames] **************************************************
changed: [isilon8104]

TASK [Copy files locally] ****************************************************
changed: [isilon8104] => (item=ISCC_Isilon81.2019-01-20.txt)
changed: [isilon8104] => (item=PUAM_Isilon81.2019-01-20.txt)
changed: [isilon8104] => (item=CCA_Isilon81.2019-01-20.txt)

PLAY RECAP *******************************************************************
isilon8104                 : ok=4    changed=3    unreachable=0    failed=0
```

Output files produced:



```
[root@ansiblecontroller Isilon]# ls -lasp /tmp/Isilon_Scripts/
total 108
 0 drwxr-xr-x.  2 root root   113 Jan 20 11:49 ./
 4 drwxrwxrwt. 14 root root  4096 Jan 20 11:49 ../
 4 -rw-r--r--.  1 root root  2545 Jan 20 11:49 CCA_Isilon81.2019-01-20.txt
 4 -rw-r--r--.  1 root root   488 Jan 20 11:49 ISCC_Isilon81.2019-01-20.txt
96 -rw-r--r--.  1 root root 95993 Jan 20 11:49 PUAM_Isilon81.2019-01-20.txt
```

Let’s break down the new components of the script:

- name: check isilon status

  hosts: isilon

  tasks:

    - name: Run monthly audit on isilon nodes

      script: /usr/local/playbooks/Isilon/playbook-isilonENVaudit.sh

### Executes the script modules which temporarily copies the script to the target node and executes. The script is removed after execution

      register: script_results

    - name: Get latest filenames

      shell: |

        (cd /ifs/data/Isilon_Support; find . –maxdepth 1 –type f) |

        grep `date +”%Y-%m-%d”` | cut –d’/’ –f2

### Shell module, executes the specified commands via a shell on the target node. The “|” character at the beginning (“Shell: |”), enables the commands to be spread across multiple lines for readability. The shell could have been written as: < shell: (cd /ifs/data/Isilon_Support; find . –maxdepth 1 –type f) | grep `date +”%Y-%m-%d”` | cut –d’/’ –f2 >

      register: files_to_copy

    - name: Copy files locally

fetch:

src: /ifs/data/Isilon_Support/{{ item }}

dest: /tmp/Isilon_Scripts/

flat: yes

with_items: “{{ files_to_copy.stdout_lines }}”

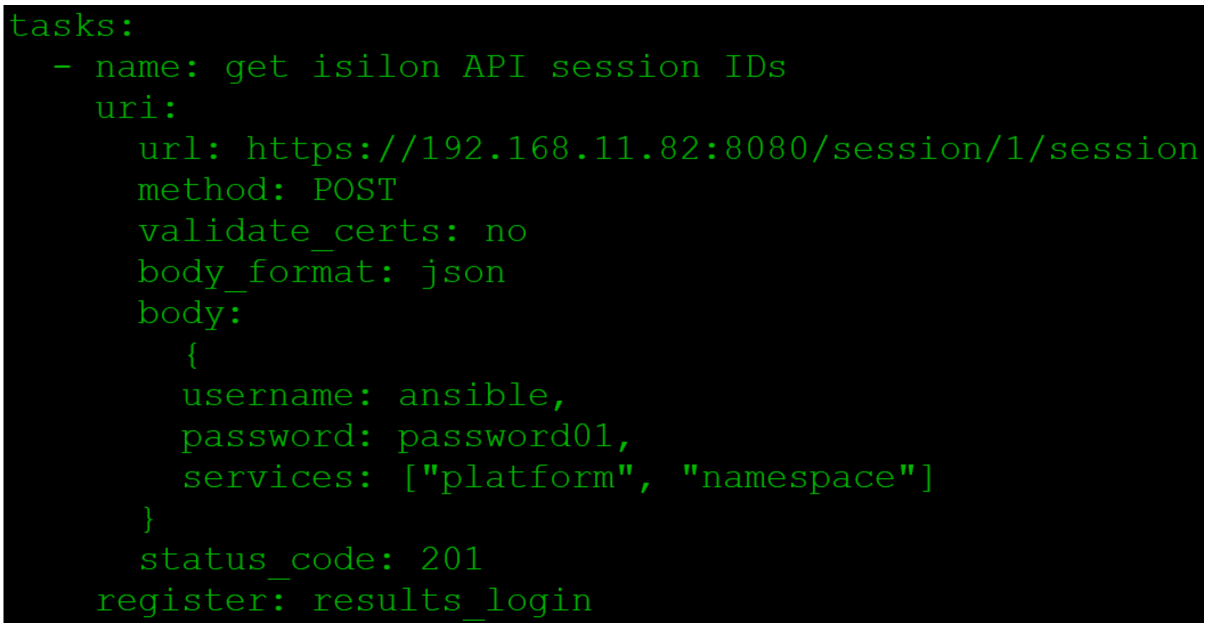
### Fetch module, copies the files from source (target node) to the destination on the local node (control node). The “with_items:” is similar to a loop function which uses the output from the previous “shell” module (stored in the variable “files_to_copy”) and stores it in a list variable of “item”. The {{ item }} variable is then called by the fetch module src value.

Isilon API and URI module

This section builds on what we learnt in the previous examples and from the earlier section “Using the Isilon API”. We will use session cookie authentication.

The typical Isilon play consists of three tasks:

1. Requesting a session cookie from Isilon API



```
tasks:
  - name: get isilon API session IDs
    uri:
      url: https://192.168.11.82:8080/session/1/session
      method: POST
      validate_certs: no
      body_format: json
      body:
        {
        username: ansible,
        password: password01,
        services: ["platform", "namespace"]
      }
      status_code: 201
    register: results_login
```

2. Perform the URI module action (e.g. create SMB share)

```
- name: make SMB share
  uri:
    url: https://192.168.11.82:8080/platform/4/protocols/smb/shares
    method: POST
    return_content: no
    validate_certs: no
    headers:
      Cookie: "isisessid={{ results_login.cookies.isisessid }}"
      X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
      referer: "https://192.168.11.82:8080"
    body_format: json
    body:
      {
        ntfs_acl_support: true,
        create_permissions: "default acl",
        name: "API SMB share create",
        path: "/ifs/Scripts/RAN",
        description: "TEST API SMB CREATE",
        browsable: true,
        zone: system,
        create_path: true,
          permissions: [
           {
            permission: "full",
            permission_type: "allow",
            trustee:
              {
               name: "KRYPTOLULA\\domain users",
               type: "group"
              }
          } ]
      }
    status_code: 201
  register: results_cookie
```

3. Optional – apply Microsoft AD "domain users" group permission / ACLs to file share

   Please Note: Performing namespace operations on Isilon requires recursive read access to the OneFS access point, directory and/or file.

```
- name: Apply Directory permissions via namespace API
  uri:
    url: https://192.168.11.82:8080/namespace/ifs/Scripts?acl
    method: PUT
    return_content: no
    validate_certs: no
    headers:
      Cookie: "isisessid={{ results_login.cookies.isisessid }}"
      X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
      x-isi-ifs-target-type: "container"
      referer: "https://192.168.11.82:8080"
    body_format: json
    body:
      {
        group:
        {
          name: "KRYPTOLULA\\domain users",
          type: "group"
        },
        authoritative: acl,
        action: update,
        acl: [
         {
          trustee:
           {
            name: "KRYPTOLULA\\domain users",
            type: "group"
           },
          accesstype: "allow",
          accessrights: [
           "dir_gen_read",
           "dir_gen_write",
           "dir_gen_execute",
           "std_write_dac",
           "delete_child"
          ],
          inherit_flags: [
           "object_inherit",
           "container_inherit"
          ],
          op: "add"
         } ]
      }
    status_code: 200
  register: results_permission
```

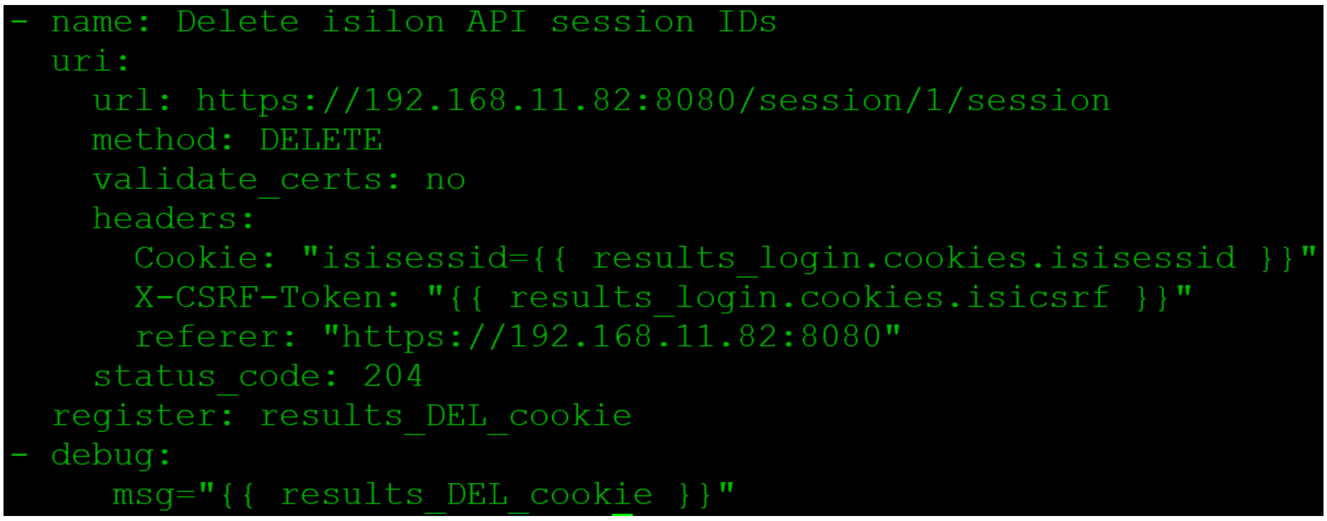
Please note: If you encounter an AEC_FORBIDDEN error or an error with “set ACL” use the Isilon CLI to apply read permissions to the OneFS access point and directory structure. See the section “File System Access API” in OneFS API Guide: https://support.emc.com/docu90414_Isilon-OneFS-8.1.2-API-Reference-Guide.pdf

WARNING: Please evaluate what permissions are required and suitable for your Isilon cluster. Dell EMC is not responsible for any damages related to applying incorrect permissions on your Isilon environments. The permission change below is for demonstration purposes ONLY, this example chmod command grants full permissions to the Isilon local “ansible” user.

For example, the following commands will ensure the “ansible” user account on Isilon has full inheritable permissions to the /ifs and /ifs/Scripts directory:

- chmod +a user ansible allow dir_gen_all,delete_child,object_inherit,container_inherit,file_gen_all,add_file,add_subdir /ifs
- chmod +a user ansible allow dir_gen_all,delete_child,object_inherit,container_inherit,file_gen_all,add_file,add_subdir /ifs/Scripts

4. Delete session id at the end of play book



```
- name: Delete isilon API session IDs
  uri:
    url: https://192.168.11.82:8080/session/1/session
    method: DELETE
    validate_certs: no
    headers:
      Cookie: "isisessid={{ results_login.cookies.isisessid }}"
      X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
      referer: "https://192.168.11.82:8080"
    status_code: 204
  register: results_DEL_cookie
- debug:
     msg="{{ results_DEL_cookie }}"
```

Ansible URI module example – create SMB share

A single play used to create an Isilon SMB share. Let’s describe each task and new module function:

Known Issue with SMB API playbook:

PLEASE NOTE: I have discovered an issue with the SMB share create playbook. The below URI API call does not properly assign the required Isilon file system “windows” ACLS to the SMB share. This should be simple to fix by including the required API calls in the below play. I’ll include the updated playbook in the next update. In the meantime, please feel free to explore the Isilon API and as a subsequent challenge, please attempt to fix the solution yourself. I’d love to hear from you regarding the attributes required to apply the required ACLs.

```
================================ PLAYBOOK ======================================

------------------------------------ PLAY ------------------------------------------------

- name: Isilon New SMB share with URI module

  hosts: isilon

  tasks:

------------------------------------ TASK 1 ------------------------------------------------

    - name: get isilon API session IDs

      uri:

        url: https://192.168.11.82:8080/session/1/session

        method: POST

        validate_certs: no

        body_format: json

        body:

          {

          username: ansible,

          password: password01,

          services: ["platform", "namespace"]

        }

        status_code: 201

      register: results_login

    - debug:

         msg="{{ results_login }}"

-------------------------------------------------------------------------------
```

The first task simply requests a session id from the Isilon API (specifically the built-in apache server). The important URI module attributes required are:

The URL to the API resource, HTTP request method “POST”, validate_cert which is equivalent to “curl –insecure”, body_format set to JSON, the body has authentication info for our ansible user and the Isilon API services (this is in plain text for tutorial purposes, not best practice), status_code tells ansible the successful return code and finally we register the results as a variable called “results_login” (we’ll need this for the next two tasks). Debug msg used to capture authentication information

```
------------------------------------ TASK 2 ------------------------------------------------

    - name: make SMB share
```

```
      uri:
        url: https://192.168.11.82:8080/platform/4/protocols/smb/shares
        method: POST
        validate_certs: no
        headers:
          Cookie: "isisessid={{ results_login.cookies.isisessid }}"
          X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
          referer: "https://192.168.11.82:8080"
        body_format: json
        body:
          {
            name: "API SMB share create",
            path: "/ifs/Scripts",
            description: "TEST API SMB CREATE",
            browsable: true,
            zone: system,
              permissions: [
               {
                permission: "full",
                permission_type: "allow",
                trustee:
                  {
                   name: "KRYPTOLULA\\domain users",
                   type: "group"
                  }
              } ]
          }
        status_code: 201
------------------------------------------------------------------------------
```

The second task users our session ids obtained from the first task which was stored in the {{ results_login }} variable. The new parameters used in the creation of the SMB share are:

url: <points to the Isilon API SMB shares object>

headers: Custom headers required for Isilon X-CSRF session authentication, variable values used from previous task (E.g. {{ results_login.isisessid }} and {{ results_login.cookies.isicsrf }})

body: Specifies the attributes to create a new Isilon SMB share. The share is created as browsable, it belongs to the system zone, and the permissions have been set to allow domain users. Please note permissions are of array type, determined by the “[…]”.

Review the Isilon API documentation for more information regarding SMB share creation and required attributes https://<Isilon_Node_IP>:8080/platform/5/protocols/smb/shares?describe

```
------------------------------------ TASK 3 ----------------------------------------------

  - name: Apply Directory permissions via namespace API
    uri:
      url: https://192.168.11.82:8080/namespace/ifs/Scripts?acl
      method: PUT
      return_content: no
      validate_certs: no
      headers:
        Cookie: "isisessid={{ results_login.cookies.isisessid }}"
        X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
        x-isi-ifs-target-type: "container"
        referer: "https://192.168.11.82:8080"
      body_format: json
      body:
        {
          group:
          {
            name: "KRYPTOLULA\\domain users",
            type: "group"
          },
          authoritative: acl,
          action: update,
          acl: [
           {
            trustee:
             {
              name: "KRYPTOLULA\\domain users",
              type: "group"
             },
            accesstype: "allow",
            accessrights: [
             "dir_gen_read",
             "dir_gen_write",
             "dir_gen_execute",
             "std_write_dac",
             "delete_child"
            ],
            inherit_flags: [
             "object_inherit",
             "container_inherit"
            ],
            op: "add"
           } ]
         }
       status_code: 200
     register: results_permission
```

The third task users our session ids obtained from the first task which was stored in the

{{ results_login }} variable. The new parameters used in the creation of the SMB share are:

url: <points to the Isilon namespace API, access point “ifs” and the container “Scripts”> The “?acl” is used to set ACLs on the container (also known as a directory / folder on the OneFS file system).

headers: Custom headers required for Isilon X-CSRF session authentication, variable values used from previous task (E.g. {{ results_login.isisessid }} and {{ results_login.cookies.isicsrf }}) We also need to add the “x-isi-ifs-target-type: "container"” header as we are modifying a directory on OneFS via the namespace API (RAN).

body: Specifies the attributes to assign to the directory, group owner, ACLS and inheritable parameters

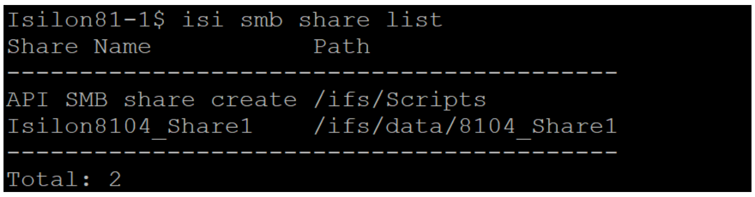
Review the Isilon API documentation for more information regarding file system access API.

```
------------------------------------ TASK 4 -----------------------------------------------

  - name: Delete isilon API session IDs

    uri:

      url: https://192.168.11.82:8080/session/1/session

      method: DELETE

      validate_certs: no

      headers:

        Cookie: "isisessid={{ results_login.cookies.isisessid }}"

        X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"

        referer: "https://192.168.11.82:8080"

      status_code: 204

    register: results_DEL_cookie

------------------------------------------------------------------------------
```

The final task cleans up our authentication by deleting our session id from the Isilon API (specifically the built-in apache server). Once again the {{ results_login }} variable is required to pass the session id cookies to the Isilon apache / API service. The HTTP method is DELETE.

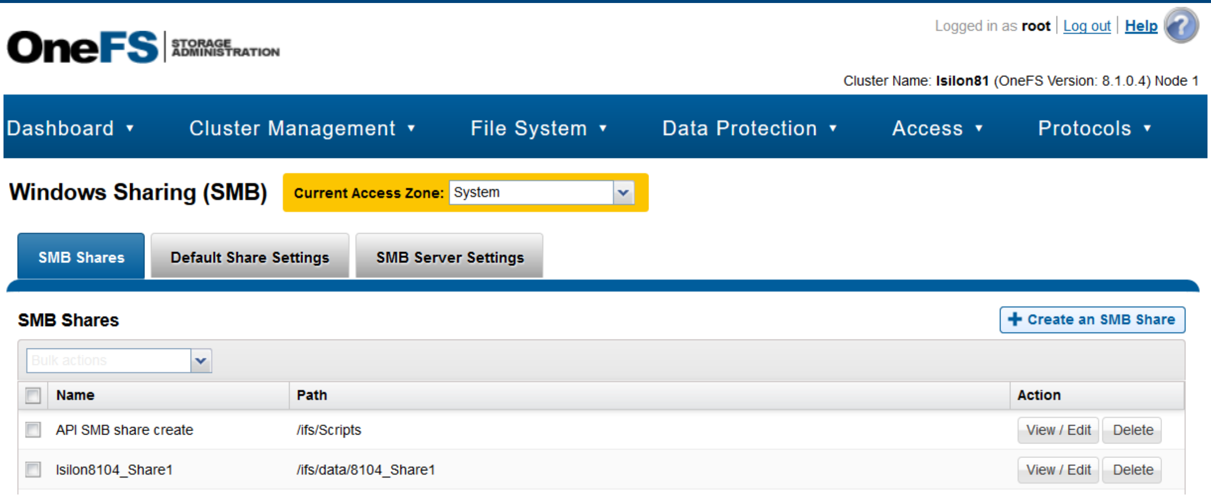
For more information, refer to the URI module documentation on ansible web site

Results of play book run on CLI:



```
Isilon81-1$ isi smb share list
Share Name           Path
------------------------------------------
API SMB share create /ifs/Scripts
Isilon8104_Share1    /ifs/data/8104_Share1
------------------------------------------
Total: 2
```

Results of play book run on GUI:



Name	Path	Action
API SMB share create	/ifs/Scripts	View / Edit Delete
Isilon8104_Share1	/ifs/data/8104_Share1	View / Edit Delete

Ansible URI modules example – create Isilon Snapshot

The first and last tasks are identical to the SMB playbook in the previous slides, so I will only cover the task used to create the snapshot (task 2) in this example. The goals of this play book are to create an ad-hoc snapshot of an Isilon directory path and set it to automatically expire in 3 days.

YAML (at least in Ansible) requires boolean, integer, float types to NOT be enclosed in double quotes. Using the YAML variable substitution directly in the playbook results in these values being evaluated as a type “string”. Hence, we need a Jinja2 file to store our variables.

Play book:

```
--------------------------------------------------------------------------------------------------------

- name: Isilon create Snapshot with URI module

  hosts: isilon

  tasks:



    <get Isilon session ID task not shown>

   - name: Create snapshot on isilon

     uri:

       url: https://192.168.11.82:8080/platform/1/snapshot/snapshots

       method: POST

       return_content: no

       validate_certs: no

       headers:
         Cookie: "isisessid={{ results_login.cookies.isisessid }}"
         X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
         referer: "https://192.168.11.82:8080"
       body_format: json
       body: '{{ lookup("template", "playbook-isilonSNAP.j2") }}'
       status_code: 201
     register: results_snapshot
   <delete Isilon session ID task not shown>
```

Contents of Jinj2 file used for variable assignment

```
{
  "name": "API snapshot from ansible",
  "path": "/ifs/data",
  "expires": {{ (ansible_date_time.epoch | int)  + ( 86400 * 3 ) }}
}
```

Description:

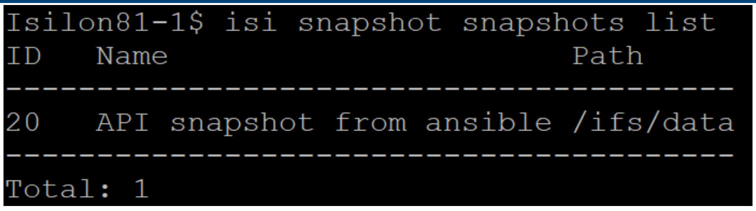
The ansible lookup “template” function is used to process the variables within the Jinj2 file. The lookup is a quick method to create a template, however a fully featured Jinj2 templating module is included with Ansible. https://docs.ansible.com/ansible/latest/modules/template_module.html

Key variable assignments are:

- Name: Name of the snapshot to create
- Path: snapshot target directory
- Expire: When should the snapshot be deleted (Please note this value does not include double quotes and is using the YAML “int” function and arithmetic. It is of type integer).

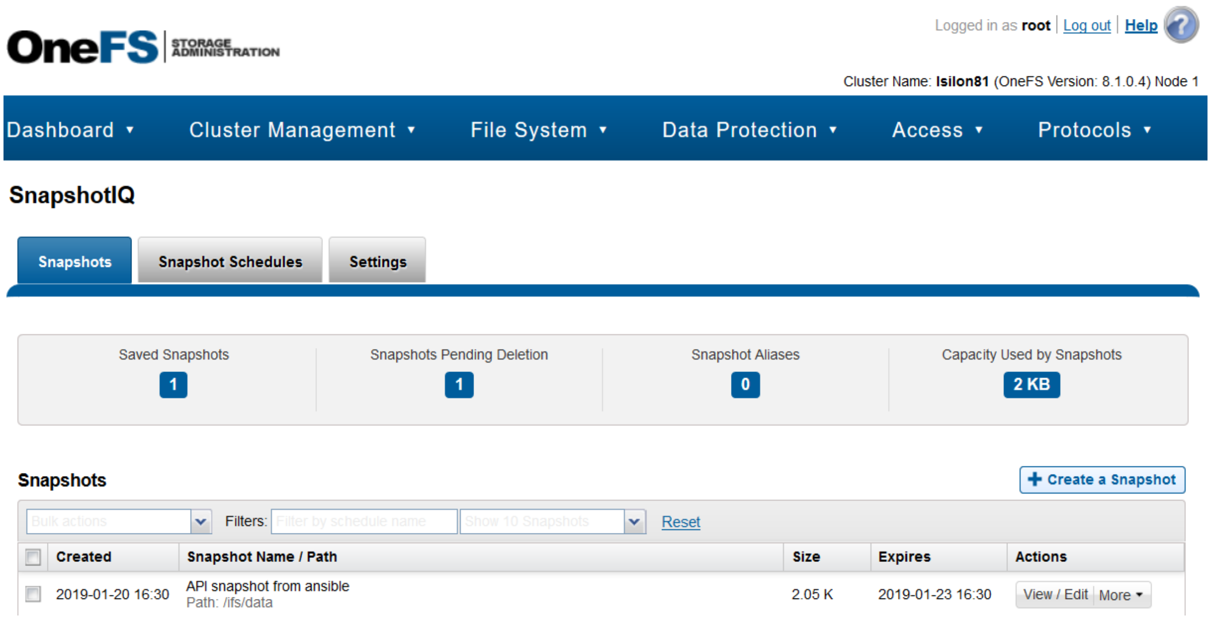
  – Ansible current unix time in seconds convert to an integer, add 3 days

Results of play book run on CLI:



```
Isilon81-1$ isi snapshot snapshots list
ID   Name                          Path
---------------------------------------
20   API snapshot from ansible /ifs/data
---------------------------------------
Total: 1
```

Results of play book run on GUI:


OneFS STORAGE ADMINISTRATION
Logged in as root | Log out | Help
Dashboard ▾ Cluster Management ▾ File System ▾ Data Protection ▾ Access ▾ Protocols ▾
Snapshots
Snapshot Schedules
Settings
Create a Snapshot
Bulk actions
Filters:
Filter by schedule name
Show 10 Snapshots
Reset
View / Edit
More ▾

Cluster Name: Isilon81 (OneFS Version: 8.1.0.4) Node 1

SnapshotIQ

Saved Snapshots	Snapshots Pending Deletion	Snapshot Aliases	Capacity Used by Snapshots
1	1	0	2 KB

Snapshots

Created	Snapshot Name / Path	Size	Expires	Actions
2019-01-20 16:30	API snapshot from ansible Path: /ifs/data	2.05 K	2019-01-23 16:30	View / Edit More

AWX (Red Hat Tower) – running Ansible play books against Isilon

In this section we will reuse our existing SMB and Snapshot ansible play books created in the previous section and migrate them to AWX. We will modify the playbooks by introducing variables and explore some differences between Ansible and AWX.

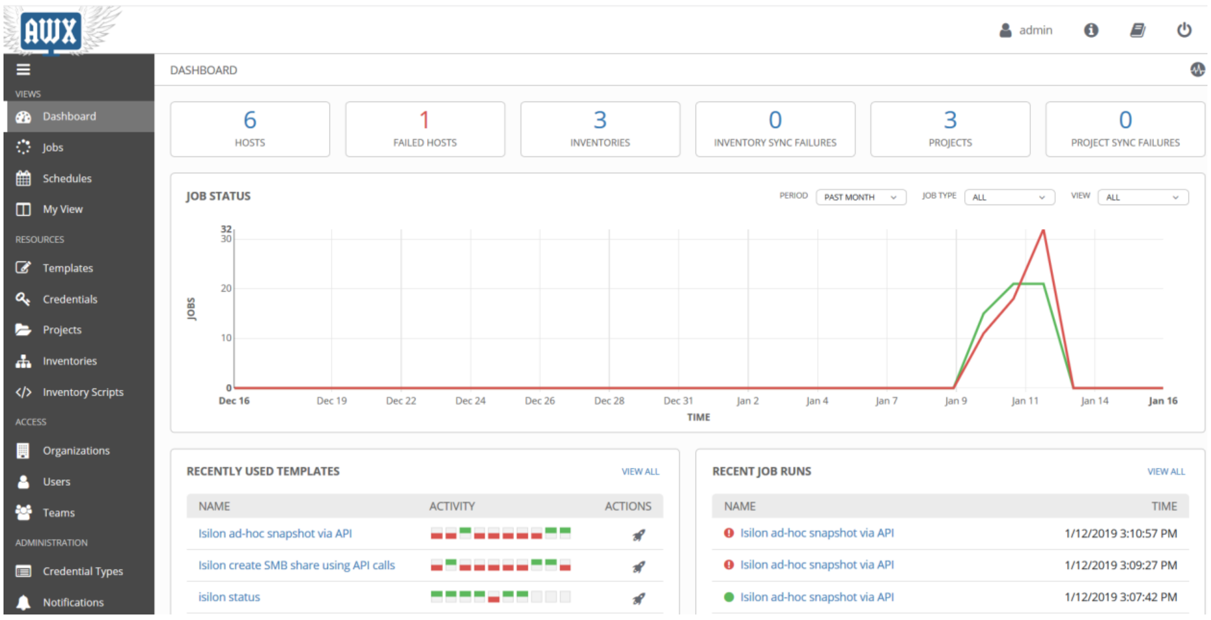
Playbook examples covered in this section:

1. Create ansible playbook with URI module to create a new Isilon SMB share
2. Create ansible playbook with URI module to create a new Isilon Snapshot with an expire time

Firstly, we need to learn some new terminology for AWX (Red Hat Ansible Tower)

- Credentials – a user account used by AWX to execute commands and run templates (play books) on hosts (target nodes)
- Templates – job templates are a set of definitions and set of parameters for running Ansible jobs (Playbooks)
- Inventories – a collection of hosts (target nodes) from which job templates and other ansible jobs can be executed against
- Projects – a logical collection of ansible playbooks
- Jobs – active and previously executed job templates
- Host – the AWX target node

Getting started with AWX



6 HOSTS | 1 FAILED HOSTS | 3 INVENTORIES | 0 INVENTORY SYNC FAILURES | 3 PROJECTS | 0 PROJECT SYNC FAILURES

PERIOD: PAST MONTH — JOB TYPE: ALL — VIEW: ALL

RECENTLY USED TEMPLATES

NAME	ACTIVITY	ACTIONS
Isilon ad-hoc snapshot via API	🟥🟥🟩🟥🟥🟥🟥🟥🟩🟩	
Isilon create SMB share using API calls	🟥🟩🟥🟥🟥🟥🟥🟩🟩🟥	
isilon status	🟩🟩🟩🟩🟥🟩🟩	

RECENT JOB RUNS

NAME	TIME
● Isilon ad-hoc snapshot via API	1/12/2019 3:10:57 PM
● Isilon ad-hoc snapshot via API	1/12/2019 3:09:27 PM
● Isilon ad-hoc snapshot via API	1/12/2019 3:07:42 PM

Logging into AWX you'll be presented with a similar dashboard (hopefully not so much red). Let's revise the steps necessary to ensure AWX is configured and ready to run Isilon playbooks on AWX:

1. Setup your Isilon user credentials
2. Setup your Isilon inventories (hosts)

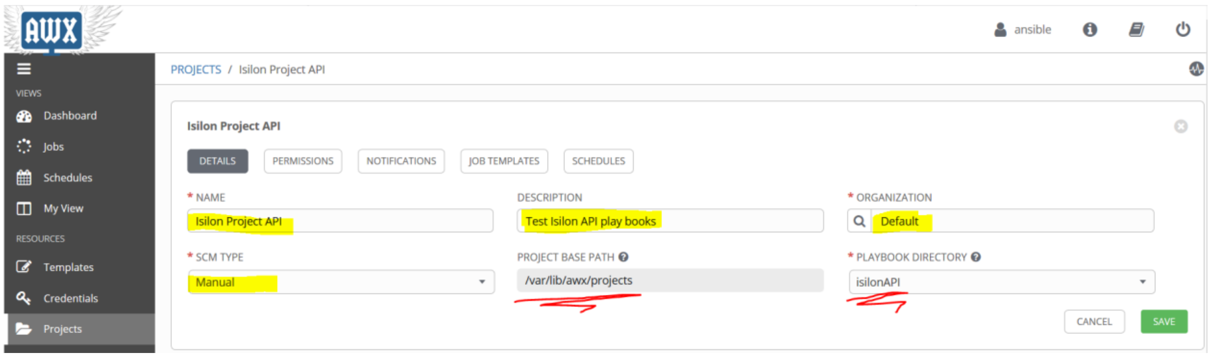
3. Test you can run the ansible "ping" module
    - Test against Isilon hosts (target nodes)
    - Make sure to use your Isilon user credentials
    - Review results on the jobs page
4. Setup your Isilon projects
5. Modify "create SMB share" play book for AWX
6. Modify "ad-hoc Snapshot" play book for AWX

If you have been following along, all the steps except steps 4 to 6 should be complete. I will start with step 4, so if you haven't yet completed setting up your AWX environment now would be a good time.

Setup your AWX Isilon Project

Use the side menu and click on "Projects" then click the "+" sign to add a new Isilon Project

Pre-requisite: If you do not have admin access to the AWX server, you will need the AWX systems administrator to create the playbook directory for you. Ask your AWX (Red Hat Ansible Tower) administrator to copy your ansible playbooks and supporting files into the "playbook directory"



* NAME: Isilon Project API
DESCRIPTION: Test Isilon API play books
* ORGANIZATION: Default
* SCM TYPE: Manual
PROJECT BASE PATH: /var/lib/awx/projects
* PLAYBOOK DIRECTORY: isilonAPI

The red underlined parameters are the locations where AWX access your ansible play books

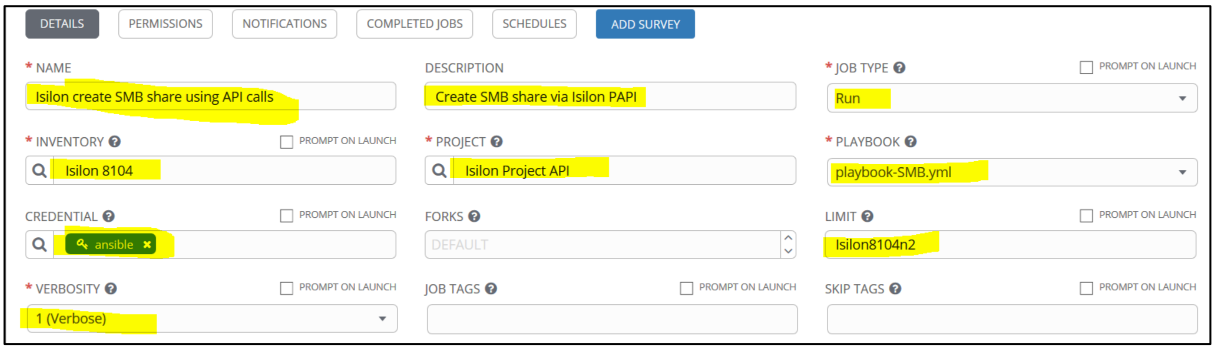
- Project Base Path is setup at the AWX time of install and can not be modified
- Playbook Directory is a child directory under the "Project Base Path" where your ansible play books reside

Fill in new Projects details:

1. Name
2. Description
3. AWX Organization
4. Select AWX organization
5. Click save

Template for creating SMB share

Use the side menu and click on "Template" then click the "+" sign to create the new job template



* NAME: Isilon create SMB share using API calls
DESCRIPTION: Create SMB share via Isilon PAPI
* JOB TYPE: Run
* INVENTORY: Isilon 8104
* PROJECT: Isilon Project API
* PLAYBOOK: playbook-SMB.yml
CREDENTIAL: ansible
FORKS: DEFAULT
LIMIT: Isilon8104n2
* VERBOSITY: 1 (Verbose)
JOB TAGS:
SKIP TAGS:

Fill in new Isilon inventory details:

1. Name
2. Description
3. Job type
4. Inventory
5. Project
6. Playbook
7. Credentials
8. Limit (This is the target node / hosts which this play templated it limited to. It can not execute on any other nodes)
9. Verbosity (I have set this to one for more output)
10. In the variable section, tick the box to prompt on launch
11. Enter variables on the right into the text window (see variables below)

Add the following variables to the "EXTRA VARIABLES" section and click prompt on launch (top right hand corner)

```
---
 sharename: "API SMB share create"
 sharepath: /ifs/Scripts
 descr: "Share created via isilon API"
 adgroup: "KRYPTOLULA\\domain users"
 sessionID: "https://192.168.11.82:8080/session/1/session"
 createSMBshare: "https://192.168.11.82:8080/platform/4/protocols/smb/shares"
 isilon_url: "https://192.168.11.82:8080"
```

The AWX playbook – create SMB share

The AWX template "Isilon create SMB share using API calls" consists of a single playbook "playbook-SMB.yml"

(This file is located in the projects directory /var/lib/awx/projects/IsilonAPI)

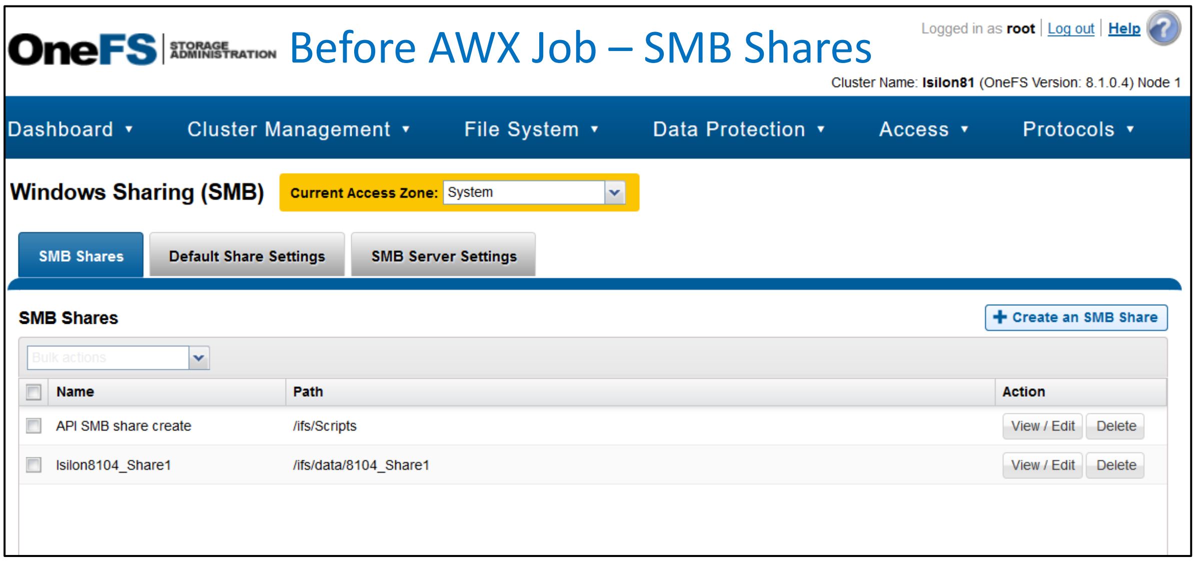
```
-
 name: Isilon URI session id
 hosts: all
 tasks:
  - name: get isilon API session IDs
    uri:
      url: "{{ sessionID }}"
      method: POST
      validate_certs: no
      body_format: json
      body:
        {
        username: ansible,
        password: password01,
        services: ["platform", "namespace"]
      }
      status_code: 201
    register: results_login
  - debug:
       msg="{{ results_login }}"

  - name: make SMB share
    uri:
      url: "{{ createSMBshare }}"
      method: POST
      return_content: no
      validate_certs: no
      headers:
        Cookie: "isisessid={{ results_login.cookies.isisessid }}"
        X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
        referer: "{{ isilon_url }}"
      body_format: json
      body:
        {
          ntfs_acl_support: true,
          create_permissions: "default acl",
          name: "{{ sharename }}",
          path: "{{ sharepath }}",
          description: "{{ descr }}",
          browsable: true,
          zone: system,
          create_path: true,
          permissions: [
           {
            permission_type: "allow",
          trustee:
           {
            name: "{{ adgroup }}",
            type: "group"
           },
          permission: "full"
         } ]
       }
     status_code: 201
    register: results_cookie
  - debug:
      msg="{{ results_cookie }}"

  - name: Apply Directory permissions via namespace API
    uri:
     url: https://192.168.11.82:8080/namespace{{ sharepath }}?acl
     method: PUT
     return_content: no
     validate_certs: no
     headers:
       Cookie: "isisessid={{ results_login.cookies.isisessid }}"
       X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
       x-isi-ifs-target-type: "container"
       referer: "{{ isilon_url }}"
     body_format: json
     body:
       {
         group:
         {
           name: "{{ adgroup }}",
           type: "group"
         },
         authoritative: acl,
         action: update,
         acl: [
          {
           trustee:
            {
             name: "{{ adgroup }}",
             type: "group"
            },
           accesstype: "allow",
           accessrights: [
            "dir_gen_read",
            "dir_gen_write",
            "dir_gen_execute",
            "std_write_dac",
            "delete_child"
            ],
            inherit_flags: [
             "object_inherit",
             "container_inherit"
            ],
            op: "add"
           } ]
         }
       status_code: 200
     register: results_permission
   - debug:
        msg="{{ results_permission }}"

   - name: Delete isilon API session IDs
     uri:
       url: "{{ sessionID }}"
       method: DELETE
       validate_certs: no
       headers:
         Cookie: "isisessid={{ results_login.cookies.isisessid }}"
         X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
         referer: "{{ isilon_url }}"
       status_code: 204
     register: results_DEL_cookie
   - debug:
        msg="{{ results_DEL_cookie }}"
-------------------------------------------------------------------------------------
```

To better illustrate playbook reusability with AWX, I have defined several AWX template variables that can be changed on launch. See variables in the template section, also you'll notice I have used blue text to highlight the variables within the playbook.

When the template is launched, the user will be asked if they would like to change any variables, so that additional shares can be created using the same job template.

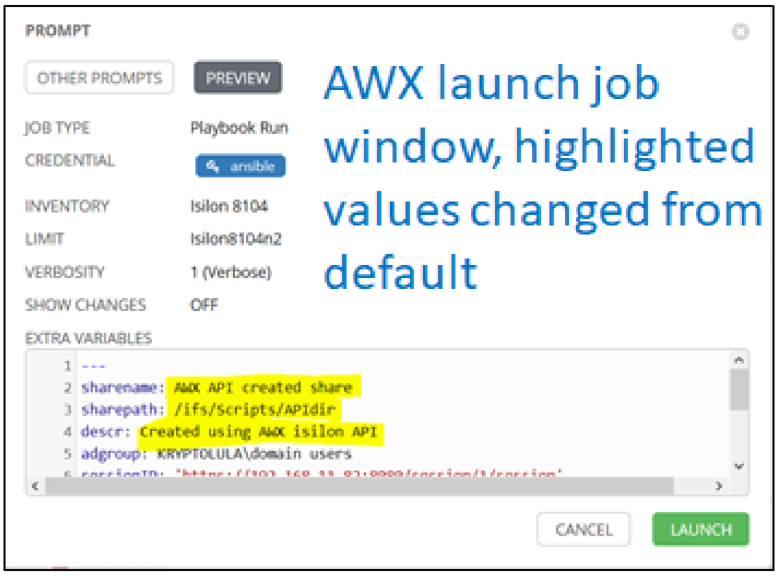
Please note: these job templates can also be called using the AWX (Red Hat Ansible Tower) API. If you are already exposing the AWX API to your developers, then your Isilon playbooks can also be incorporated into their development projects.

Isilon before AWX job template launch



Name	Path	Action
API SMB share create	/ifs/Scripts	View / Edit Delete
Isilon8104_Share1	/ifs/data/8104_Share1	View / Edit Delete

AWX job template on launch variable change

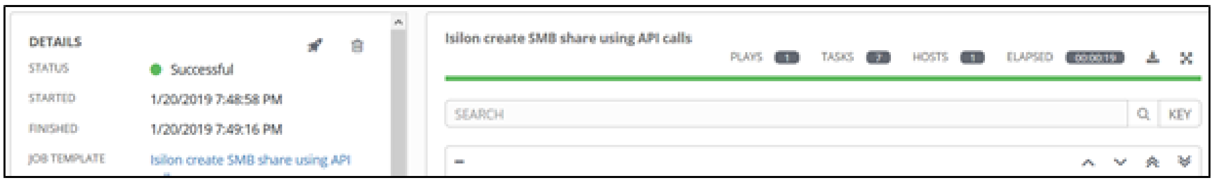

PROMPT
OTHER PROMPTS
PREVIEW
AWX launch job window, highlighted values changed from default
CANCEL
LAUNCH

JOB TYPE	Playbook Run
CREDENTIAL	ansible
INVENTORY	Isilon 8104
LIMIT	Isilon8104n2
VERBOSITY	1 (Verbose)
SHOW CHANGES	OFF

EXTRA VARIABLES

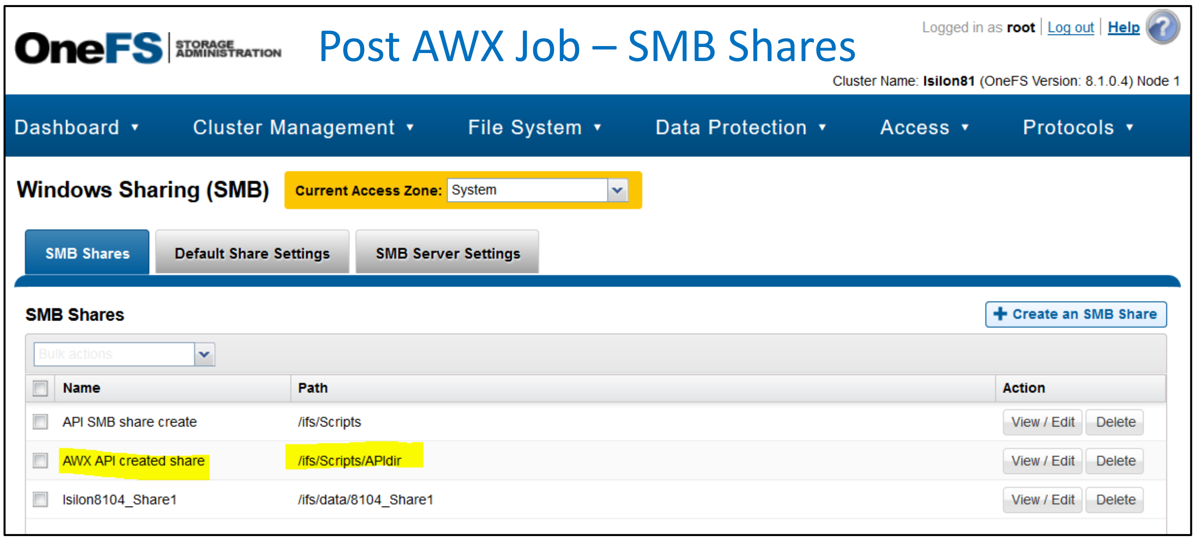
```
---
sharename: AWX API created share
sharepath: /ifs/Scripts/APIdir
descr: Created using AWX isilon API
adgroup: KRYPTOLULA\domain users
```

Highlighted values were changed from default



STATUS	● Successful
STARTED	1/20/2019 7:48:58 PM
FINISHED	1/20/2019 7:49:16 PM
JOB TEMPLATE	Isilon create SMB share using API

Isilon post AWX job template execution



Cluster Name: Isilon81 (OneFS Version: 8.1.0.4) Node 1

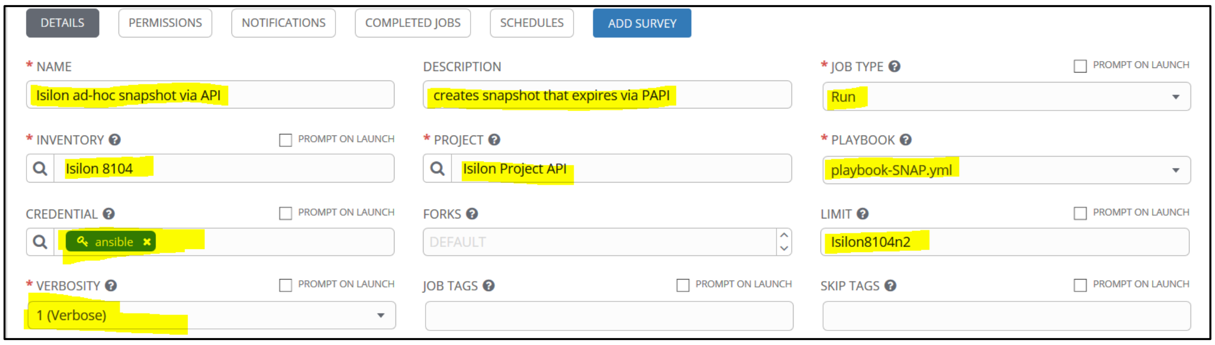
Windows Sharing (SMB) — Current Access Zone: System

SMB Shares

Name	Path	Action
API SMB share create	/ifs/Scripts	View / Edit Delete
AWX API created share	/ifs/Scripts/APIdir	View / Edit Delete
Isilon8104_Share1	/ifs/data/8104_Share1	View / Edit Delete

Template for creating ad-hoc Snapshot

Use the side menu and click on “Template” then click the “+” sign to create the new job template



* NAME: Isilon ad-hoc snapshot via API
DESCRIPTION: creates snapshot that expires via PAPI
* JOB TYPE: Run
* INVENTORY: Isilon 8104
* PROJECT: Isilon Project API
* PLAYBOOK: playbook-SNAP.yml
CREDENTIAL: ansible
FORKS: DEFAULT
LIMIT: Isilon8104n2
* VERBOSITY: 1 (Verbose)
JOB TAGS:
SKIP TAGS:

Fill in new Isilon inventory details:

1. Name
2. Description
3. Job type
4. Inventory
5. Project
6. Playbook
7. Credentials
8. Limit (This is the target node / hosts which this play templated it limited to. It can not execute on any other nodes)
9. Verbosity (I have set this to one for more output)

10. In the variable section, tick the box to prompt on launch

11. Enter variables on the right into the text window (see below default variables)

Add the following variables to the "EXTRA VARIABLES" section and click prompt on launch (top right hand corner)

```
---
  snapname: "API ad-hoc snapshot"
  snappath: /ifs
  keepdays: 2
  descr: "Snapshot created via isilon API"
  sessionID: "https://192.168.11.82:8080/session/1/session"
  createSNAP: "https://192.168.11.82:8080/platform/1/snapshot/snapshots"
  isilon_url: "https://192.168.11.82:8080"
```

The AWX playbook – ad-hoc Snapshot

The AWX template "Isilon ad-hoc snapshot via API" consists of a playbook file "playbook-SNAP.yml", a Jinj2 template file for integer variable substitution "playbook-SNAP.j2" and a JSON file consisting of user, password, services API services information "playbook-SNAP.auth.json"

(This files is located in the projects directory /var/lib/awx/projects/IsilonAPI)

Additional files explained:

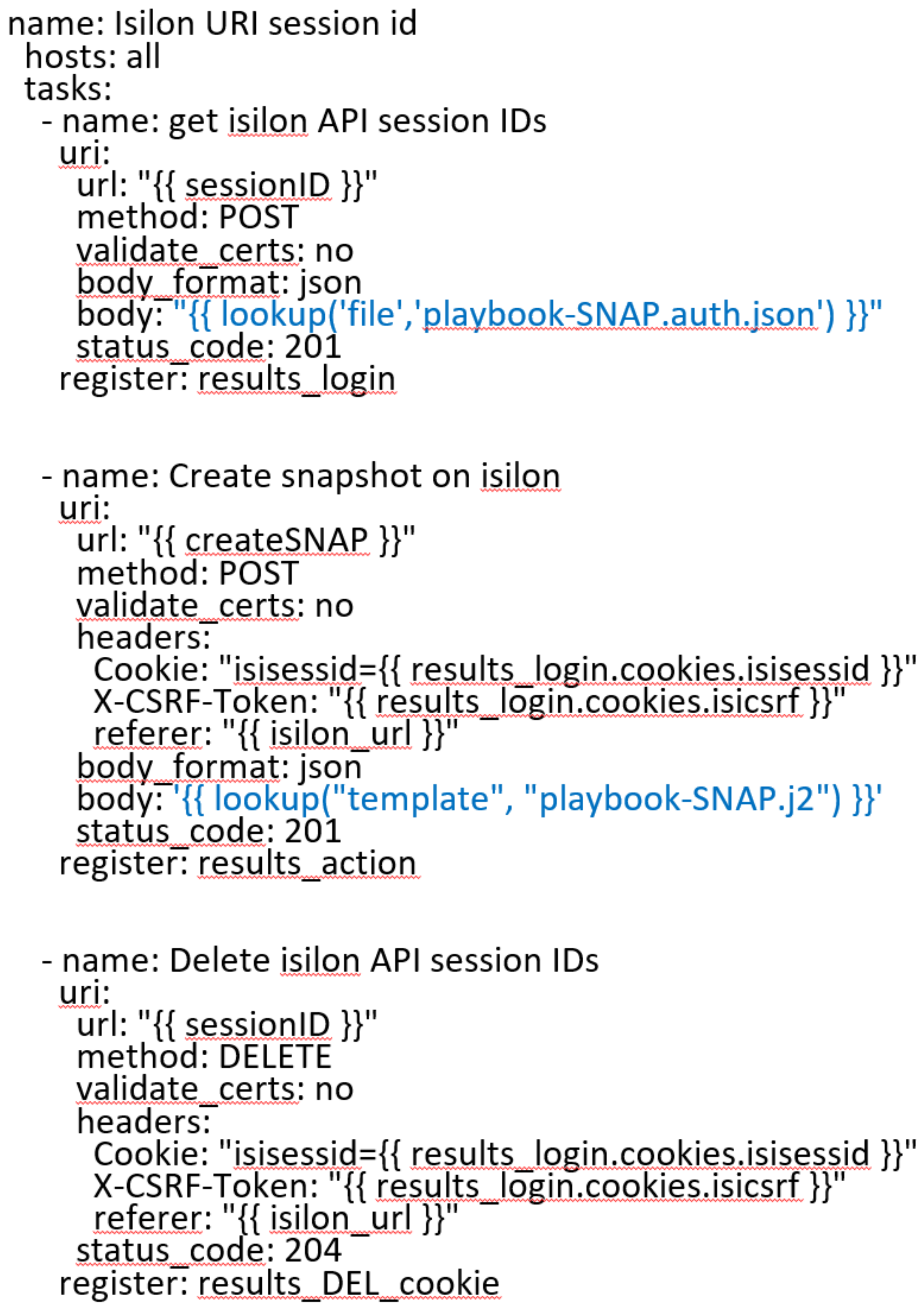
This example has an external json formatted file to store the user authentication information and a Jinja2 formatted file for the Snapshot parameters. The Jinja2 template file is required because we are setting a variable as type integer.

Contents of Auth file (playbook-SNAP.auth.json):

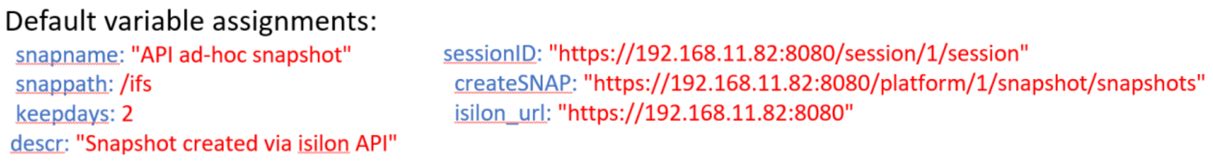
```
{
      "username": "root",
      "password": "password01",
      "services": ["platform", "namespace"]
}
```

Contents of Jinja2 file (playbook-SNAP.j2):

```
{
  "name": "{{ snapname }}",
  "path": "{{ snappath }}",
  "expires": {{ (ansible_date_time.epoch | int)  + ( 86400 * (keepdays | int) ) }}
}
```



```
name: Isilon URI session id
  hosts: all
  tasks:
    - name: get isilon API session IDs
      uri:
        url: "{{ sessionID }}"
        method: POST
        validate_certs: no
        body_format: json
        body: "{{ lookup('file','playbook-SNAP.auth.json') }}"
        status_code: 201
      register: results_login


    - name: Create snapshot on isilon
      uri:
        url: "{{ createSNAP }}"
        method: POST
        validate_certs: no
        headers:
          Cookie: "isisessid={{ results_login.cookies.isisessid }}"
          X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
          referer: "{{ isilon_url }}"
        body_format: json
        body: '{{ lookup("template", "playbook-SNAP.j2") }}'
        status_code: 201
      register: results_action


    - name: Delete isilon API session IDs
      uri:
        url: "{{ sessionID }}"
        method: DELETE
        validate_certs: no
        headers:
          Cookie: "isisessid={{ results_login.cookies.isisessid }}"
          X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
          referer: "{{ isilon_url }}"
        status_code: 204
      register: results_DEL_cookie
```

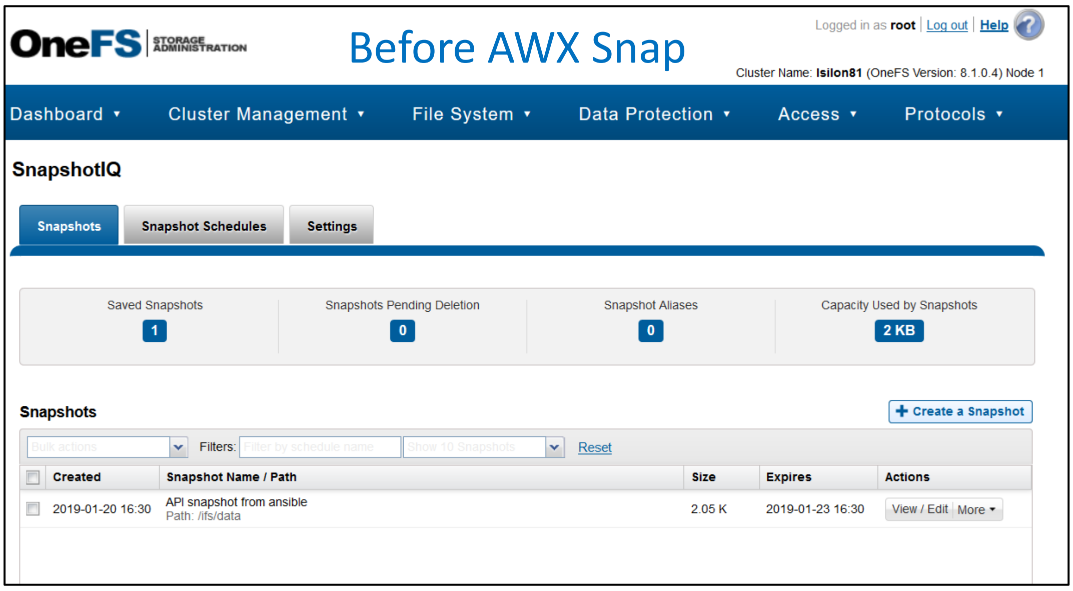


Default variable assignments:

```
 snapname: "API ad-hoc snapshot"            sessionID: "https://192.168.11.82:8080/session/1/session"
 snappath: /ifs                              createSNAP: "https://192.168.11.82:8080/platform/1/snapshot/snapshots"
 keepdays: 2                                 isilon_url: "https://192.168.11.82:8080"
descr: "Snapshot created via isilon API"
```

When the template is launched, the user will be asked if they would like to change any variables, so that additional shares can be created using the same job template.

Isilon before AWX job template launch - snapshots

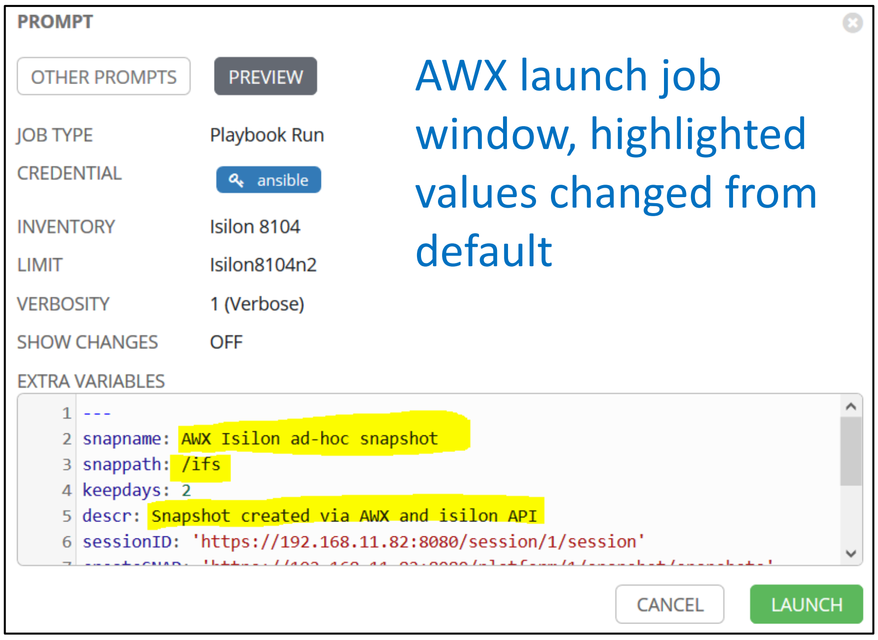


Saved Snapshots	Snapshots Pending Deletion	Snapshot Aliases	Capacity Used by Snapshots
1	0	0	2 KB

Snapshots

Created	Snapshot Name / Path	Size	Expires	Actions
2019-01-20 16:30	API snapshot from ansible Path: /ifs/data	2.05 K	2019-01-23 16:30	View / Edit More

AWX job template on launch variable change

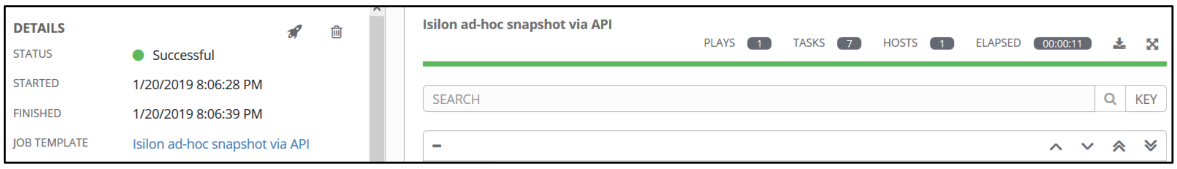

PROMPT
OTHER PROMPTS
PREVIEW
AWX launch job window, highlighted values changed from default
CANCEL
LAUNCH

JOB TYPE	Playbook Run
CREDENTIAL	ansible
INVENTORY	Isilon 8104
LIMIT	Isilon8104n2
VERBOSITY	1 (Verbose)
SHOW CHANGES	OFF

EXTRA VARIABLES

```
---
snapname: AWX Isilon ad-hoc snapshot
snappath: /ifs
keepdays: 2
descr: Snapshot created via AWX and isilon API
sessionID: 'https://192.168.11.82:8080/session/1/session'
```

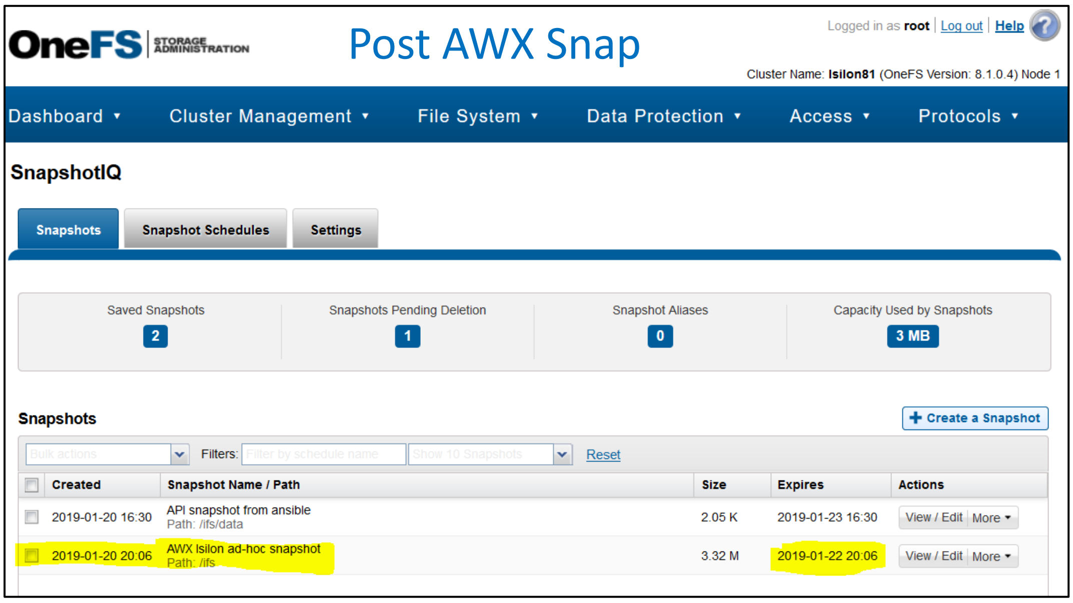
Highlighted values were changed from default



DETAILS

STATUS	Successful
STARTED	1/20/2019 8:06:28 PM
FINISHED	1/20/2019 8:06:39 PM
JOB TEMPLATE	Isilon ad-hoc snapshot via API

Isilon post AWX job template execution



SnapshotIQ

Saved Snapshots	Snapshots Pending Deletion	Snapshot Aliases	Capacity Used by Snapshots
2	1	0	3 MB

Snapshots

Created	Snapshot Name / Path	Size	Expires	Actions
2019-01-20 16:30	API snapshot from ansible Path: /ifs/data	2.05 K	2019-01-23 16:30	View / Edit More
2019-01-20 20:06	AWX Isilon ad-hoc snapshot Path: /ifs	3.32 M	2019-01-22 20:06	View / Edit More

Conclusion

Dell EMC Isilon is the leader in the industry and the definitive benchmark in scale out file.

Delivering simplicity at scale, and an intuitive easy to use management interface. Isilon makes managing 50PB of data in a single storage array as easily as managing 10TB of data. This capability is unmatched.

Leveraging our RESTAPI functionality with a DevOps tool such as ansible or integrating Isilon RESTAPI calls directly into your application development initiatives could not be easier.

Many of our clients have extended this capability into ServiceNow and other operational platforms

I hope this document helps you on your devOps journey with Isilon.

Appendix A – Isilon playbook example utilizing ansible roles in AWX

The ansible playbook examples within the core of the document have generally been developed for demonstration purposes; hence the playbook examples within the core of the document have not following followed ansible best practices.

To improve the readability and inject some best practice into our Isilon playbooks, I have modified the "SMB create playbook" on AWX with the following better practices:

- Split up the playbook into "roles" for each Isilon task
  - get-isilon-auth
  - create-isilon-share
  - apply-isilon-acl
  - del-isilon-auth
- Replaced hardcoded Isilon URL with ansible host variable "anisble_host" from AWX inventory.

Step 1 – create role directory structure for SMB create playbook

Create "role" directory structure where the playbooks reside.

1. cd <playbook location>
2. mkdir roles
3. cd roles
4. for i in "get-isilon-auth create-isilon-share apply-isilon-acl del-isilon-auth"; do

   mkdir ${i}/files ${i}/handlers ${i}/meta ${i}/tasks ${i}/template ${i}/vars

   done

Step 2 – Split original create SMB share playbook into "roles" task files

Create the new playbook core file and role task files as follows:

Check working directory, ensure core playbook file is in the directory

```
[root@ansibleAWX isilonAPI]# pwd
/var/lib/awx/projects/isilonAPI
```

Contents of core playbook file

```
[root@ansibleAWX isilonAPI]# cat playbook-SMB-permission-roles.yml
---
- hosts: all
  roles:
    - get-isilon-auth
    - create-isilon-share
    - apply-isilon-acl
    - del-isilon-auth
...
```

Contents of "get-isilon-auth" role task file

```
[root@ansibleAWX isilonAPI]# cat roles/get-isilon-auth/tasks/main.yml
---
```

```
- name: get isilon API session IDs
  uri:
    url: "{{ sessionID }}"
    method: POST
    validate_certs: no
    body_format: json
    body:
      {
      username: ansible,
      password: password01,
      services: ["platform", "namespace"]
        }
    status_code: 201
  register: results_login
- debug:
     msg="{{ results_login }}"
...
```

Contents of "create-isilon-share" role task file

```
[root@ansibleAWX isilonAPI]# cat roles/create-isilon-share/tasks/main.yml
---
- name: make SMB share
  uri:
    url: "{{ createSMBshare }}"
    method: POST
    return_content: no
    validate_certs: no
    headers:
      Cookie: "isisessid={{ results_login.cookies.isisessid }}"
      X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
      referer: "{{ isilon_url }}"
    body_format: json
    body:
      {
        ntfs_acl_support: true,
        create_permissions: "default acl",
        name: "{{ sharename }}",
        path: "{{ sharepath }}",
        description: "{{ descr }}",
        browsable: true,
        zone: system,
        create_path: true,
        permissions: [
         {
          permission_type: "allow",
          trustee:
            {
             name: "{{ adgroup }}",
          type: "group"
         },
       permission: "full"
      } ]
    }
   status_code: 201
  register: results_cookie
- debug:
    msg="{{ results_cookie }}"
```

Contents of “apply-isilon-acl” role task file

```
[root@ansibleAWX isilonAPI]# cat roles/apply-isilon-acl/tasks/main.yml
---
- name: Apply Directory permissions via namespace API
  uri:
    url: "{{ isilon_url }}/namespace{{ sharepath }}?acl"
    method: PUT
    return_content: no
    validate_certs: no
    headers:
      Cookie: "isisessid={{ results_login.cookies.isisessid }}"
      X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
      x-isi-ifs-target-type: "container"
      referer: "{{ isilon_url }}"
    body_format: json
    body:
      {
        group:
        {
          name: "{{ adgroup }}",
          type: "group"
        },
        authoritative: acl,
        action: update,
        acl: [
         {
          trustee:
           {
            name: "{{ adgroup }}",
            type: "group"
           },
          accesstype: "allow",
          accessrights: [
           "dir_gen_read",
           "dir_gen_write",
           "dir_gen_execute",
           "std_write_dac",
           "delete_child"
          ],
          inherit_flags: [
           "object_inherit",
           "container_inherit"
          ],
          op: "add"
         } ]
       }
    status_code: 200
  register: results_permission
- debug:
     msg="{{ results_permission }}"
...
```

Contents of “del-isilon-auth” role task file

```
[root@ansibleAWX isilonAPI]# cat roles/del-isilon-auth/tasks/main.yml
---
- name: Delete isilon API session IDs
  uri:
    url: "{{ sessionID }}"
    method: DELETE
    validate_certs: no
    headers:
      Cookie: "isisessid={{ results_login.cookies.isisessid }}"
      X-CSRF-Token: "{{ results_login.cookies.isicsrf }}"
      referer: "{{ isilon_url }}"
    status_code: 204
  register: results_DEL_cookie
- debug:
     msg="{{ results_DEL_cookie }}"
...
```

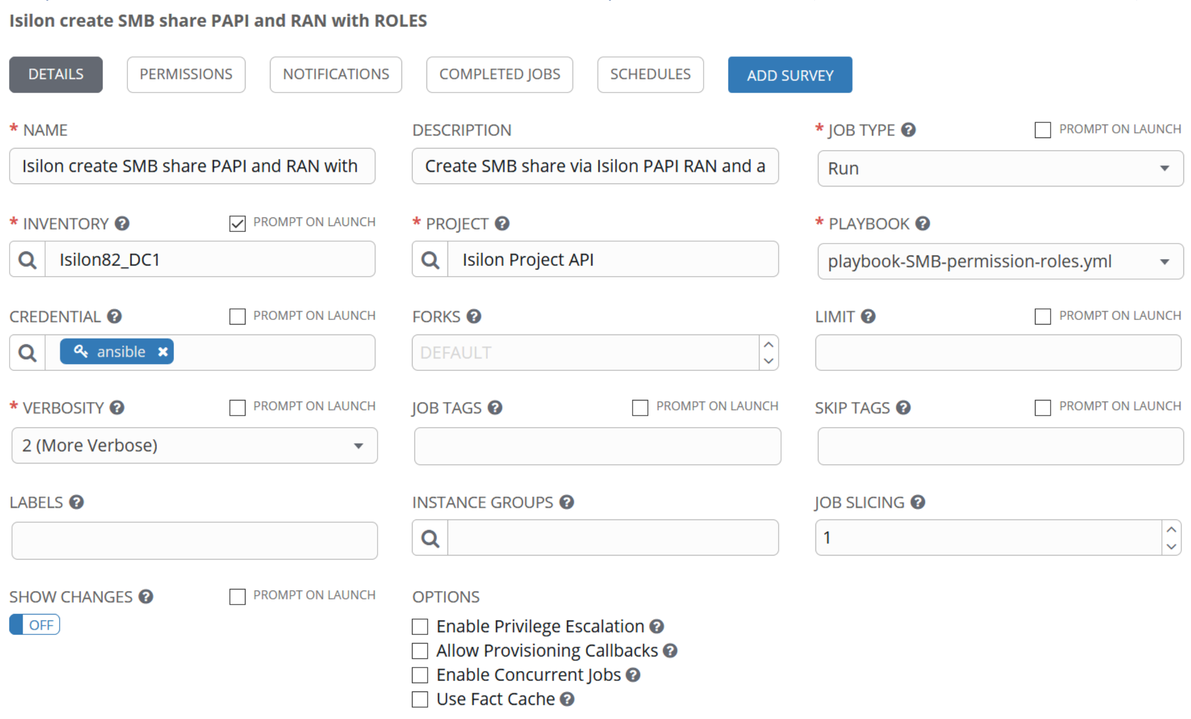
Ansible playbook and roles directory and file structure:

The final ansible playbook and role directory and file structure should be as follows:

```
Parent Directory
        |
        ---> playbook-SMB-permission-roles.yml        ### THIS IS THE CORE PLAYBOOK FILE ###
        |
        ---> Roles
                |
                ---> get-isilon-auth
                        |
                        ---> file
                        ---> handlers
                        ---> meta
                        ---> tasks
                                |
                                ---> main.yml          ### GET ISILON AUTH TASK FILE ###
                        ---> templates
                        ---> vars
                ---> create-isilon-share
                        |
                        ---> file
                        ---> handlers
                        ---> meta
                        ---> tasks
                                |
                                ---> main.yml          ### CREATE ISILON SHARE TASK FILE ###
                        ---> templates
                        ---> vars
                ---> apply-isilon-acl
                        |
                        ---> file
                        ---> handlers
                        ---> meta
                        ---> tasks
                                |
                                ---> main.yml          ### APPLY ISILON ACL TASK FILE ###
                        ---> templates
                        ---> vars
                ---> del-isilon-auth
                        |
                        ---> file
                        ---> handlers
                        ---> meta
                        ---> tasks
                                |
                                ---> main.yml          ### DELETE ISILON AUTH TASK FILE ###
```

---> templates
---> vars

Step 3 – create new "create SMB share template" in AWX (Red Hat Ansible Tower)



Isilon create SMB share PAPI and RAN with ROLES

* NAME: Isilon create SMB share PAPI and RAN with	DESCRIPTION: Create SMB share via Isilon PAPI RAN and a	* JOB TYPE (☐ PROMPT ON LAUNCH): Run
* INVENTORY (☑ PROMPT ON LAUNCH): Isilon82_DC1	* PROJECT: Isilon Project API	* PLAYBOOK: playbook-SMB-permission-roles.yml
CREDENTIAL (☐ PROMPT ON LAUNCH): ansible	FORKS: DEFAULT	LIMIT (☐ PROMPT ON LAUNCH):
* VERBOSITY (☐ PROMPT ON LAUNCH): 2 (More Verbose)	JOB TAGS (☐ PROMPT ON LAUNCH):	SKIP TAGS (☐ PROMPT ON LAUNCH):
LABELS:	INSTANCE GROUPS:	JOB SLICING: 1
SHOW CHANGES (☐ PROMPT ON LAUNCH): OFF	OPTIONS: ☐ Enable Privilege Escalation ☐ Allow Provisioning Callbacks ☐ Enable Concurrent Jobs ☐ Use Fact Cache	

Add the following variables to the playbook:

sharename: "Ansible Share {{ansible_host}}"

sharepath: /ifs/Shared

descr: "Share created via isilon API"

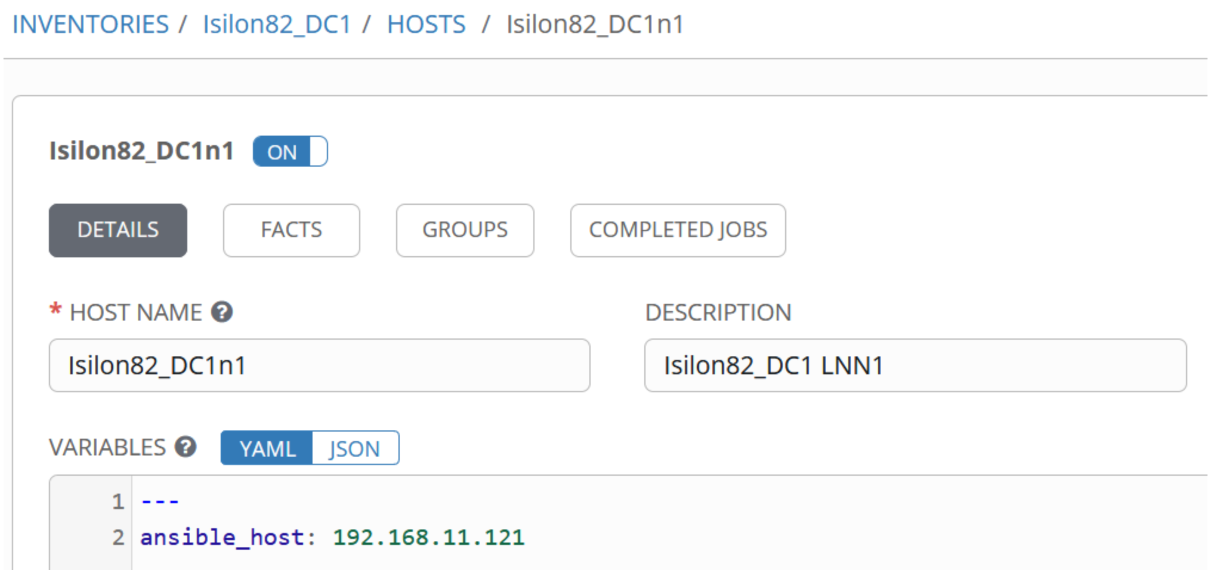
adgroup: "KRYPTOLULA\\domain users"

sessionID: "https://{{ansible_host}}:8080/session/1/session"

createSMBshare: "https://{{ansible_host}}:8080/platform/4/protocols/smb/shares"

isilon_url: "https://{{ansible_host}}:8080"

Please note: I have created this playbook to run on a new inventory which contains a single Isilon node "Isilon82_DC1", running OneFS version 8.2.0. This inventory / host definition includes a variable "ansible_host" which is set to the I.P. address of the isilon82_DC1n1. See following pic:



* HOST NAME: Isilon82_DC1n1

DESCRIPTION: Isilon82_DC1 LNN1

VARIABLES: YAML

```
---
ansible_host: 192.168.11.121
```