Ansible AWX Training

Introduction

This document provides an overview on Ansible and AWX features and hands on tutorials.

Installation

Please refer to the following link: https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

CentOS Red Hat Installation

These are the steps to install AWX/Ansible on top of CentOS/RHEL:

#EPEL Repo

yum -y install epel-release

Install Additional Packages

yum -y install git gettext ansible docker nodejs npm gcc-c++ bzip2

yum -y install python-docker-py

Run Docker

systemctl start docker

systemctl enable docker

Clone AWX Repo and Deploy

git clone https://github.com/ansible/awx.git

cd awx/installer/

ansible-playbook -i inventory install.yml

Notes

You might need to modify git commands that access remote repositories if your internet access is through a proxy server. Once you have obtained the proxy settings (server URL, port, username and password); you need to configure your git as follows:

 # git config --global http.proxy http://<username>:<password>@<proxy-server-url>:<port>

You would need to replace <username>, <password>, <proxy-server-url>, <port> with the values specific to your proxy server credentials.

If you do not specify “project_data_dir=/var/lib/awx/projects” during the deployment, you can’t use Manual SCM. You might face issue while creating the new project.

If you are behind an HTTP or HTTPS proxy server, you need to add the proxy configuration in the Docker systemd service file.

Docker

AWX relies on five Docker containers provided by Red Hat Ansible in Docker Hub (https://hub.docker.com/u/ansible).

The Ansible installation creates a PostgreSQL database that will be in one container and will create the AWX one that contains the web tier, the engine, a cache and a queue:

- AWX Web. Server which provides HTTP GUI access.
- AWX Task. Engine responsible of providing API interworking for solution components.
- RabbitMq. Acting as a messaging broker between solution components.
- Memcached. Memcached is a general-purpose distributed memory caching system. It is used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read.
- Postgres database. AWX requires access to a PostgreSQL database, and by default, one will be created and deployed in a container, and data will be persisted to a host volume.

Docker Getting Started

Once Docker is installed, open a terminal and type the following commands:

#docker info

#docker -v

If the installation worked, you will see a bunch of information about your Docker installation and docker version running:

 # docker -v

```
# docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.13.1
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: journald
Cgroup Driver: systemd
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Authorization: rhel-push-plugin
Swarm: inactive
Runtimes: docker-runc runc
Default Runtime: docker-runc
```

You can list the containers that are running by issuing the following command:

```
# docker ps
CONTAINER ID        IMAGE
COMMAND                  CREATED             STATUS
PORTS                                        NAMES
28143772a881        ansible/awx_task:2.1.2              "/tini -- /bin/
sh ..."             19 minutes ago      Up 19 minutes       8052/
tcp                                          awx_task
ded05efa006d        ansible/awx_web:2.1.2               "/tini -- /bin/
sh ..."             19 minutes ago      Up 19 minutes       0.0.0.0:80->8052/
tcp                                          awx_web
a76b6a798a5b        memcached:alpine                    "docker-entrypoint..."
      43 hours ago        Up 6 hours              11211/
tcp                                          memcached
```

Use the command "docker exec -it <container ID> /bin/bash" to get a bash shell in the container:

```
[root@RedhatNewAnsible projects]# docker exec -it ded05efa006d  /bin/bash
[root@awxweb awx]# ls
awxfifo  favicon.ico  projects  public  supervisord.log  supervisord.pid  venv  wsgi.py
[root@awxweb awx]# exit
exit
[root@RedhatNewAnsible projects]#
```

Docker Useful Commands

Stops one or more containers.

```
docker stop my_container
```

Stop all running containers:

```
docker stop $(docker ps -a -q)
```

The following command does not attempt to shut down the process gracefully first:

```
docker kill my_container
```

Start one or more containers:

```
docker start my_container
```

This command displays the logs of a container:

```
docker logs --follow my_container
```

Remove one or more containers:

```
docker rm my_container
```

Remove one or more images:

docker rmi my_image

Running Ansible on your laptop

Vagrant is a tool for building and managing virtual machine environments. Vagrant provides easy to configure, reproducible, and portable work environments built on top of industry-standard technology.

Installing Vagrant

First of all, request local admin rights in advance to install the required software.

Download Virtualbox: https://www.virtualbox.org and install it.

Download Vagrant: https://www.vagrantup.com for your operating system.

Ensure that Vagrant is properly installed by issuing the "vagrant" command as shown below:

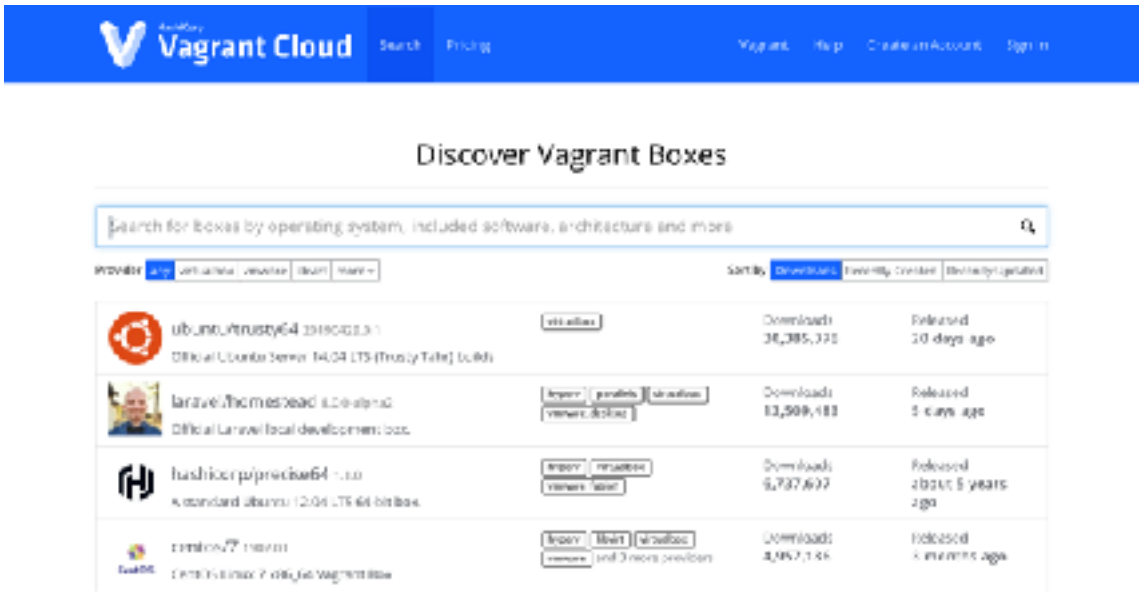
Create a local directory and then make a `shared` folder.

sudo mkdir centos7

cd centos7

sudo mkdir shared

Download the boxes you need to deploy from Vagrant Cloud: https://app.vagrantup.com/boxes/search (i.e. centos/7)



The command below downloads a centos 7 box

sudo vagrant box add centos/7

In this case, I am downloading a box called "jumperfly/ansible-2.8" which provides Ansible already installed.

NOTE: This is not an official box so be aware of the security risks.

Next step is to initialise the box.

sudo vagrant init centos/7

Launch your VM by issuing the command below:

sudo vagrant up

Access the VM initiated. In this case, Ansible is already installed ("ansible --version").

sudo vagrant ssh

Vagrant Basic Commands
Navigate to the folder which stores your Vagrantfile and pause the VM

sudo vagrant suspend

Switch off the VM

sudo vagrant halt

Destroy the VM

sudo vagrant destroy

Quick Start

Concepts

Ansible Engine

Inventory
The Ansible inventory file defines the hosts and groups of hosts upon which commands, modules, and tasks in a playbook operate. The file can be in one of many formats depending on your Ansible environment and plugins. The default location for the inventory file is /etc/ansible/hosts. If necessary, you can also create project-specific inventory files in alternate locations.

The inventory file can list individual hosts or user-defined groups of hosts. For example, if you are managing one or more data centers, you can create Ansible groups for those components that require the same set of operations.

YAML
YAML stands for "YAML Ain't Markup Language" (Please refer to: https://yaml.org).

it is basically a human-readable structured data format. It is less complex and ungainly than XML or JSON, but provides similar capabilities.

There are some rules that YAML has in place to avoid issues related to ambiguity. These rules make it possible for a single YAML file to be interpreted consistently, regardless of which library is being used to interpret it.

- YAML files should end in .yaml.
- YAML is case sensitive.
- YAML uses a fixed indentation scheme to represent relationships between data layers.
- Dictionary keys are represented in YAML as strings terminated by a trailing colon. Values are represented by either a string following the colon, separated by a space.
- To represent lists of items, a single dash followed by a space is used. Multiple items are a part of the same list as a function of their having the same level of indentation.

The following example represents a YAML playbook:

- name: configure interface settings ios config: lines: - description

Configuration File

Certain settings in Ansible are adjustable via a configuration file (ansible.cfg). The stock configuration should be sufficient for most users, but there may be reasons you would want to change them.

Configuration file which will be processed in the following order:

1. Environment variable: ANSIBLE_CONFIG
2. Actual directory "ansible.cfg" file.
3. Home directory "ansible.cfg" file.
4. Configuration file stored in "/etc/ansible/ansible.cfg".

Option	Default Value	Description
inventory	/etc/ansible/hosts	Inventory location
forks	5	Specify number of parallel processes to use
remote_port	22	Remote SSH port
host_key_checking	true	Check host key installed
timeout	10	SSH connection timeout in seconds
remote_user	root	Remote connection user
become	false	Run operations with become (does not imply password prompting)
become_method	sudo	Privilege escalation method to use
pipelining	false	Reduces the number of network operations required to execute a module on the remote server, by executing many Ansible modules without actual file transfer.This can result in a very significant performance improvement when enabled

Please refer to Ansible documentation for more details: https://docs.ansible.com/ansible/latest/reference_appendices/config.html

Playbooks

Playbooks are Ansible's configuration, deployment, and orchestration language. Playbooks are designed to be human-readable and are developed in a basic text language. There are multiple ways to organize playbooks and the files they include.

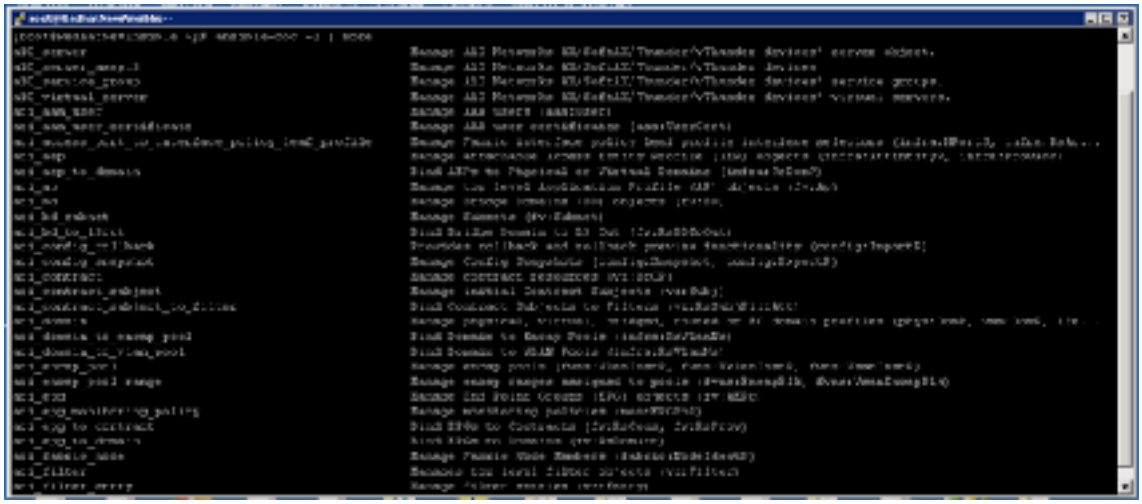
Modules

Ansible ships with a number of modules (called the 'module library') that can be executed directly on remote hosts or through playbooks. Users can also write their own modules. These modules can control system resources, like services, packages, or files , or handle executing system commands.

Ansible modules: https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html

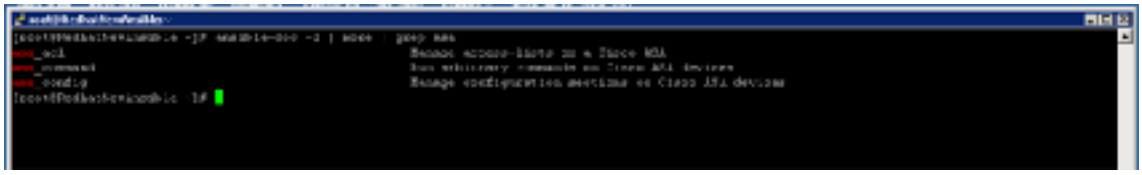
The following command displays the list of available modules:

#ansible-doc -l | more



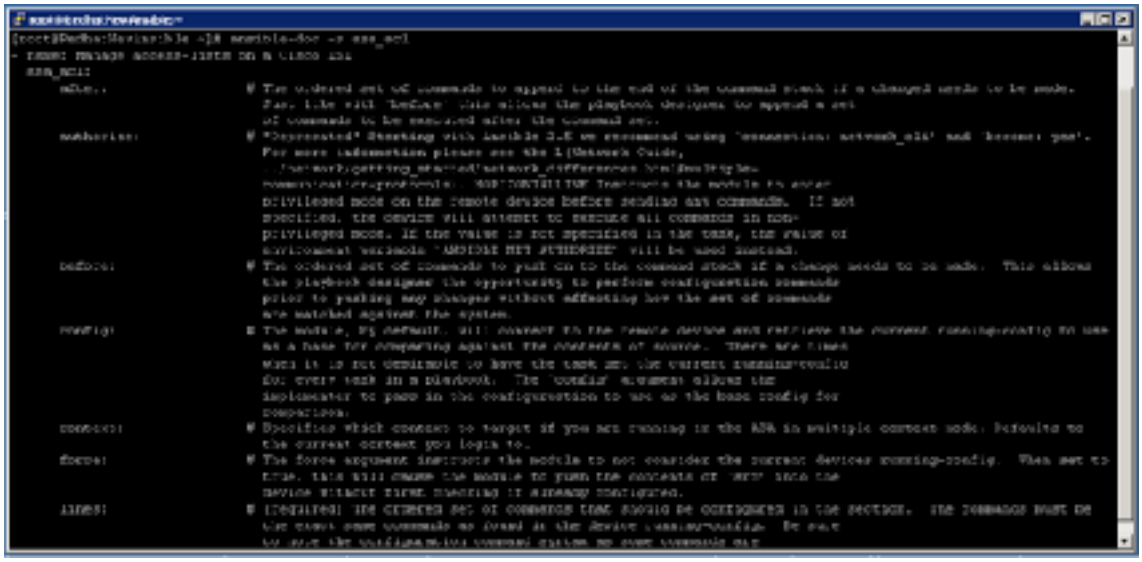
Narrow down modules related to ASA devices:

#ansible-doc -l | more | grep asa



Show actions that a module can perform:

ansible-doc -s module_name



Ansible Tower - AWX

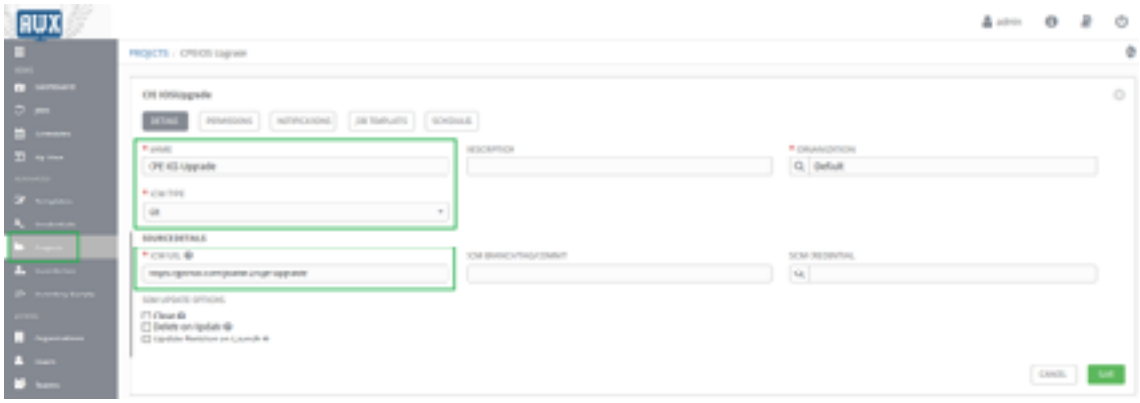
The concepts across Ansible Engine and AWX (Ansible Tower) are the same. However, the GUI provided by AWX makes workflows being configured in a different way.

For running an Ansible Playbook with AWX, you need to configure the following items:

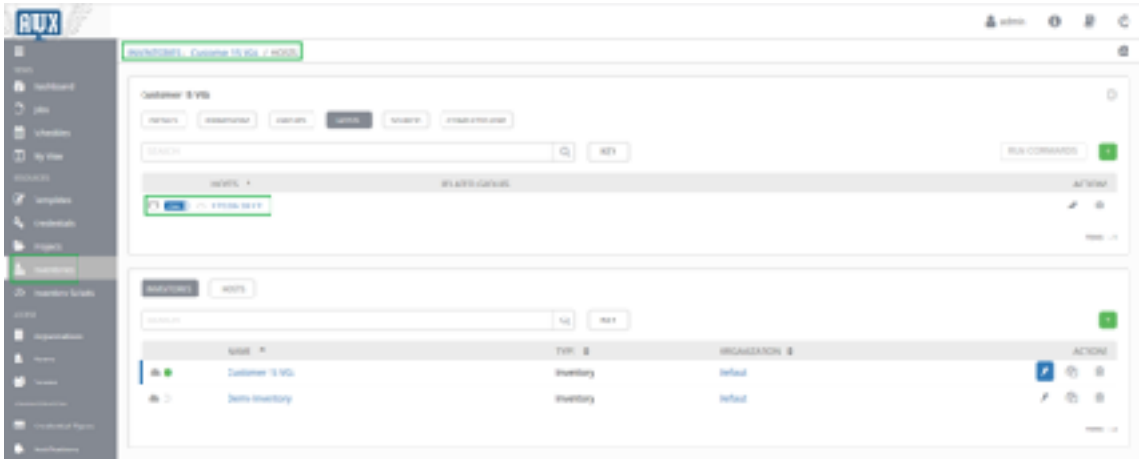
1. Credentials: User name/password or ssh key to connect to remote component.
2. Project: It contains the Ansible playbook, config, roles, templates etc.
3. Inventories: What servers the playbook will run against and connection specific configuration.
4. Templates: Job template to associate all of the above and run the playbook
5. Launch Templates: Launching current project.

For instance, credentials (1) for a CPE device (Credential Type = Network) are created by typing the username and password value pair.

Next step requires creating a new Project (2) which is using Git as source.

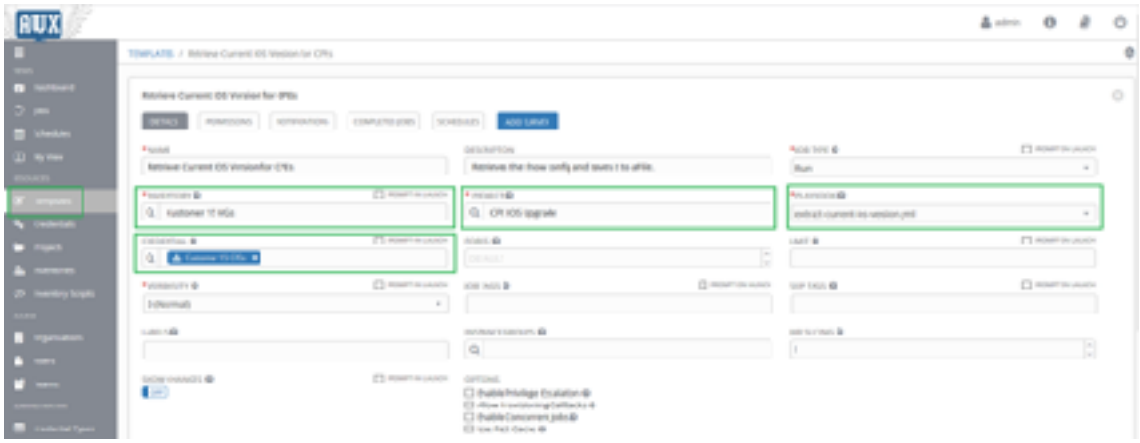


Once the Project is saved, click on Inventories (3), create the corresponding one and add Hosts to it as shown below:

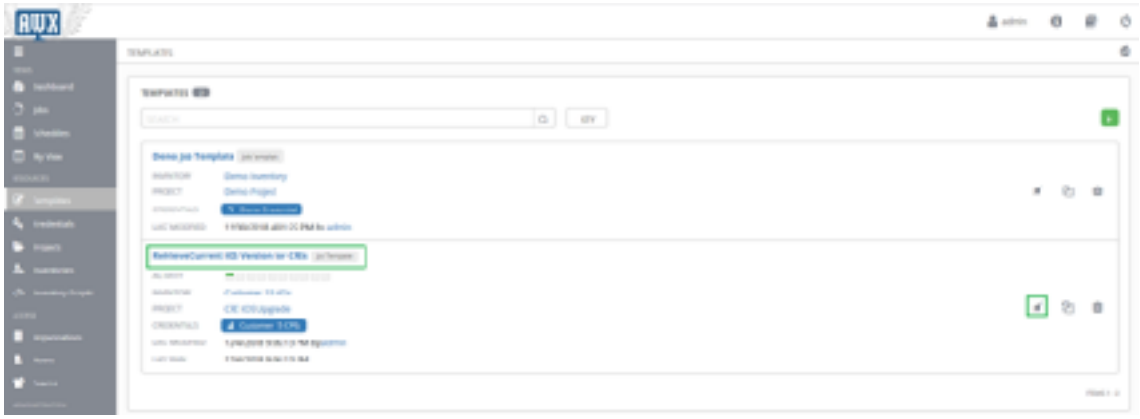


A new Hosts (i.e CPE IP address) is added to the "Customer 15 VGs" inventory.

Lets create a Job Template (4) which associated all the above: the inventory, credential and the Project which stores the list of playbooks.



Finally, click on the Templates tab and launch it.



Ad-hoc Commands

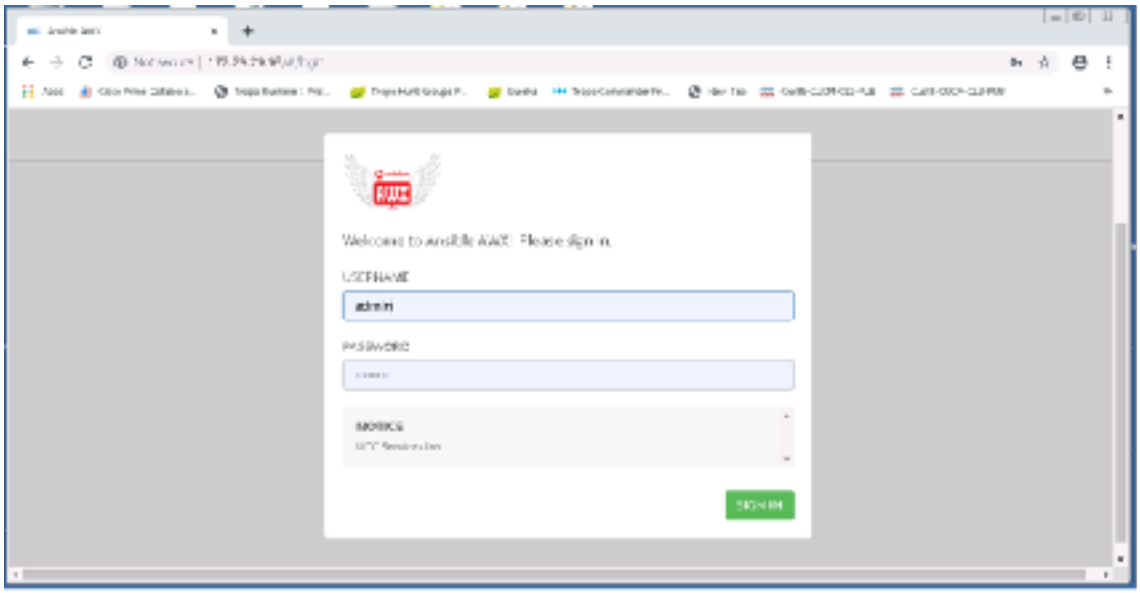
An Ansible Ad-hoc command is a one-liner Ansible command that performs a single task on the target host. It allows you to execute simple one-line task against one or group of hosts defined on the inventory file configuration.

An Ad-Hoc command will only have two parameters, the group of a host that you want to perform the task and the Ansible module to run.

Command	Description
#ansible all -m ping	The basic command of ansible ad-hoc against 'all' hosts on the inventory file and using the 'ping' module
#ansible localhost -m copy -a 'src=/home/myfile dest=/home/mydestinationfolder/myfilecopied'	The command copies a file to a destination folder
#ansible all -m yum -a "name=telnet state=present" -- become	The command installs the "telnet" package against 'all' hosts on the inventory file
#ansible all -m service -a "name=nginx state=started enabled=yes" –-become	The commands starts the nginx service
#ansible localhost -m setup | more	The command retrieves a bunch of system data from the remote host

AWX

Browse the corresponding URL as shown below.



Sign in with your credentials and you will be redirected to the AWX dashboard.

Click on Users tab, choose your username and click on Edit.

Edit your password, add you to specific organizations and teams, manage permissions and click on Save.

Ansible Engine

You can access Ansible Engine via SSH using any SSH client.

As a brief example, the /home/AnsibleTraining/ folder contains a inventory file called “hosts”.

The /home/AnsibleTraining/playbooks/ folder stores the playbook below:

```
[root@RedhatNewAnsible playbooks]# cat asa_show_version.yml
---
- hosts: asa-lab
  gather_facts: false
  connection: network_cli
tasks:
- name: SHOW ASA VERSION
   asa_command:
   commands:
      - show version
   register: version
- debug: var=version.stdout_lines
```

Let´s analize the anatomy of the playbook above:

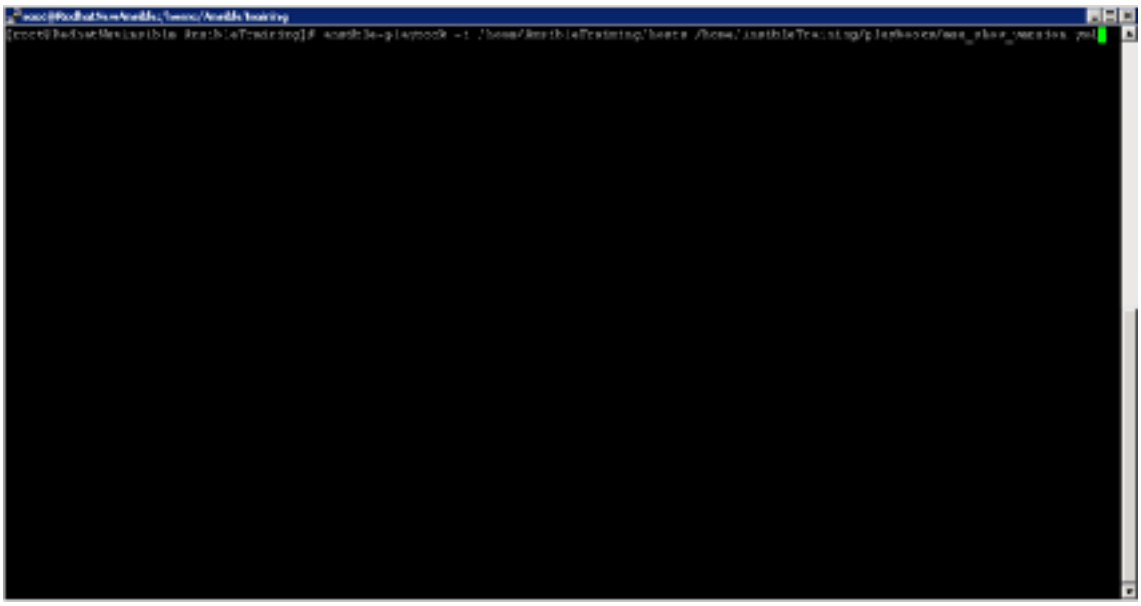
- hosts: Instructs the playbook to be run againsts the "asa-lab" group in the inventory file.
- gather_facts will connect to the managed host, run a script that collects a bunch of data (system, version, environment variables, etc.).
- Ansible uses the "connection" setting to determine how to connect to a remote device. When working with Ansible Networking, set this to network_cli so Ansible treats the remote

node as a network device with a limited execution environment. Without this setting, Ansible would attempt to use ssh to connect to the remote and execute the Python script on the network device, which would fail because Python generally isn't available on network devices.

- The tasks section includes two modules: "asa command" and "debug".
- The "asa_command" module is called in order to issue a "show version" command and store the fetched data in a register.
- "The "debug" commands prints the register output.

The following command runs the mentioned playbook:

#ansible-playbook -i /home/AnsibleTraining/hosts /home/AnsibleTraining/playbooks/asa_show_version.yml



Inventories

The inventory file defines the hosts and groups of hosts upon which commands, modules, and tasks in a playbook operate. The inventory file can be in one of many formats depending on your Ansible environment and plugins.

The default location for the inventory file is /etc/ansible/hosts. If necessary, you can also create project-specific inventory files in alternate locations.

The inventory file can list individual hosts or user-defined groups of hosts. This enables you to define groups of devices with similar roles upon which to perform the same operational and configuration tasks. For example, if you are managing one or more data centers, you can create Ansible groups for those elements that require the same set of operations.

Ansible Engine

The following INI-formatted sample inventory file defines an individual host, called "webserver", and two groups of devices, ios and nxos.

$ cat /etc/ansible/hosts

The inventory file includes variables also:

```
[cisco-ios-devices:vars]
ansible_port=22
ansible_user=netadmin
```

Groups of groups are also supported:

```
[cisco-ios-devices]
10.10.0.1
10.10.0.2

[cisco-nxos-devices]
10.10.20.1
10.10.20.2

[network-devices:children]
cisco-ios-devices
cisco-nxos-devices
```

Patterns can be used as shown below:

```
[cisco-nexus-7000]
nexus[01:04].companydomain.com
```

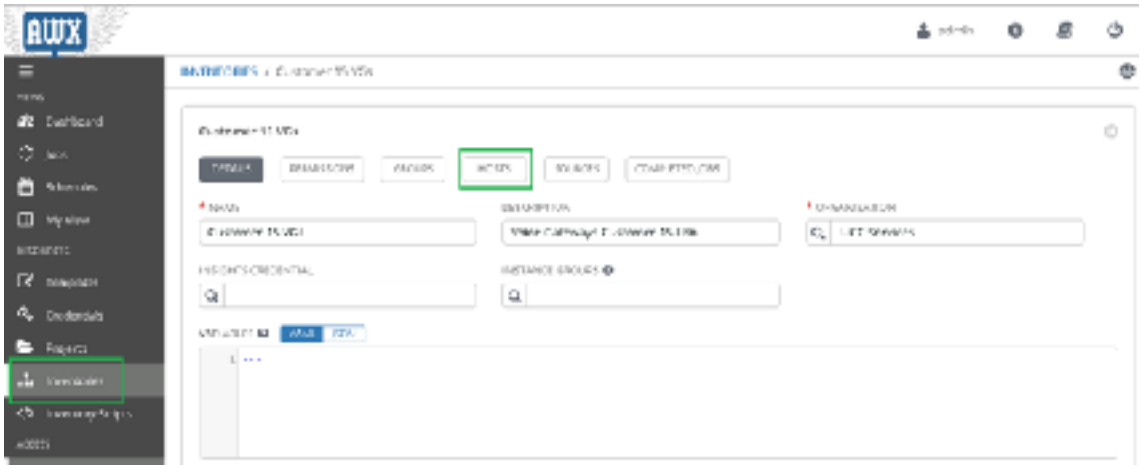
The pattern above corresponds with the following bunch of hosts:

```
nexus01.companydomain.com
nexus02.companydomain.com
nexus03.companydomain.com
nexus04.companydomain.com
```

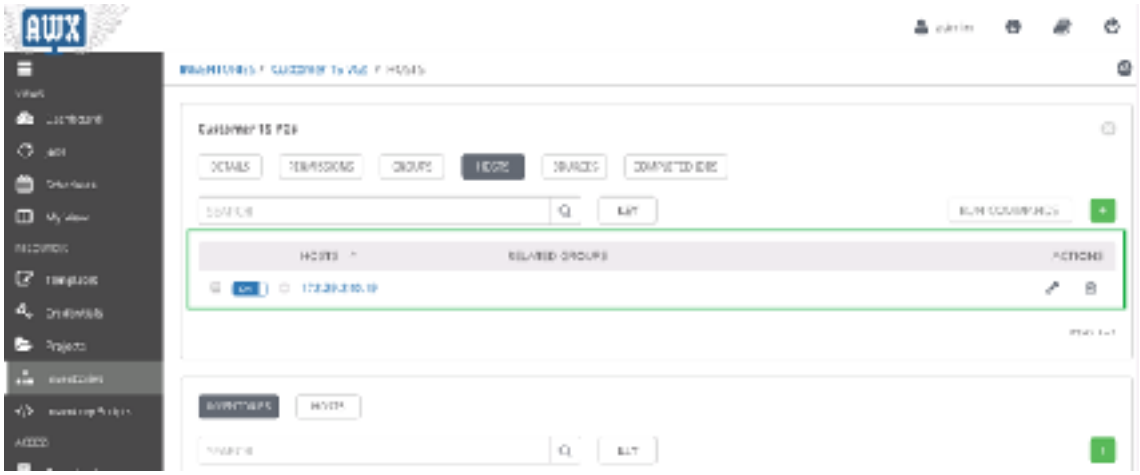
AWX

AWX stores the inventory in the backend database. To create an inventory list, use the option “Inventories” and click Add and name the Inventory list and save. Click the Add Host button to add a host, the host name can be a DNS resolvable name or an IP address.

Create a new inventory by navigating to Inventories tab and clicking on .



Once the new inventory is saved , click on the “Hosts” tab and click ⊕ to add new hosts.



Bulk Import

AWX offers a utility called “awx-manage” to perform several of operations via CLI . One of the most important features is bulk hosts import.

Login to the awx task container and execute the following command to import hosts to inventory:

You can list the containers that are running by issuing the following command:

```
# docker ps
CONTAINER ID        IMAGE
COMMAND                  CREATED             STATUS
PORTS                                              NAMES
28143772a881        ansible/awx_task:2.1.2             "/tini -- /bin/
sh ..."             19 minutes ago      Up 19 minutes       8052/
tcp                                                awx_task
ded05efa006d        ansible/awx_web:2.1.2              "/tini -- /bin/
sh ..."             19 minutes ago      Up 19 minutes       0.0.0.0:80->8052/
tcp                                                awx_web
```

Use the command "docker exec -it <container ID> /bin/bash" to get a bash shell in the container:

[root@RedhatNewAnsible projects]# docker exec -it ded05efa006d /bin/bash

[root@awxweb awx]# ls

awxfifo favicon.ico projects public supervisord.log supervisord.pid venv wsgi.py

[root@awxweb awx]# exit

exit

[root@RedhatNewAnsible projects]#

Create a file which contains the list of hosts to add:

[root@awx awx]# more customer_15_vgs

172.29.240.20

172.29.240.21

172.29.240.22

172.29.240.23

172.29.240.24

Once the list is ready in the file, identify the inventory name in which you need to add the hosts from AWX GUI. In the example, I am adding the hosts to “Customer 15 VGs” inventory.

[root@awx awx]# awx-manage inventory_import --inventory-name 'Customer 15 VGs' --source customer_15_vgs

2.326 INFO Updating inventory 2: Customer 15 VGs

2.496 INFO Reading Ansible inventory source: /var/lib/awx/customer_15_vgs

3.532 INFO Processing JSON output...

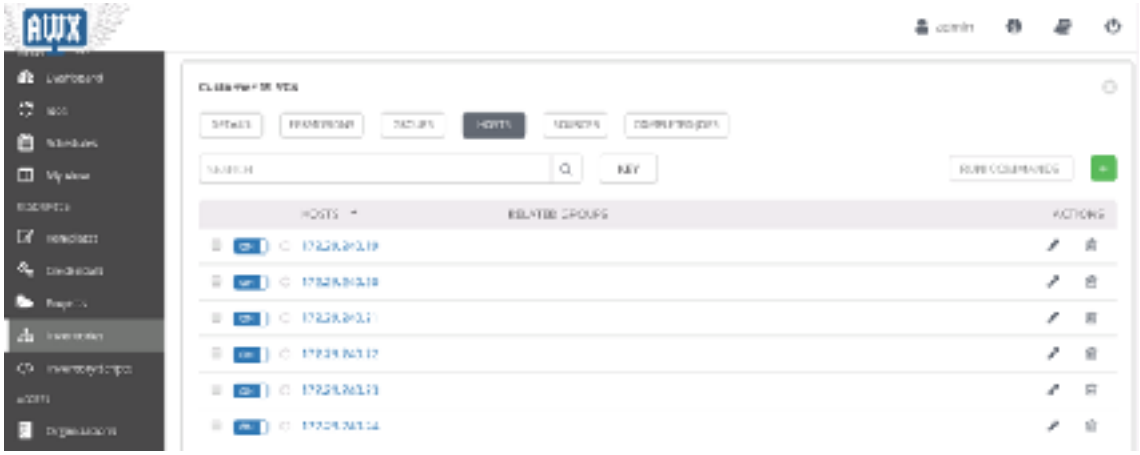
3.533 INFO Loaded 0 groups, 5 hosts

2018-12-17 21:57:24,015 DEBUG awx.main.models.inventory Going to update inventory computed fields

2018-12-17 21:57:24,114 DEBUG awx.main.models.inventory Finished updating inventory computed fields

3.702 INFO Inventory import completed for (Customer 15 VGs - 12) in 1.4s

The imported hosts are displayed now in AWX GUI.



Dynamic Inventory

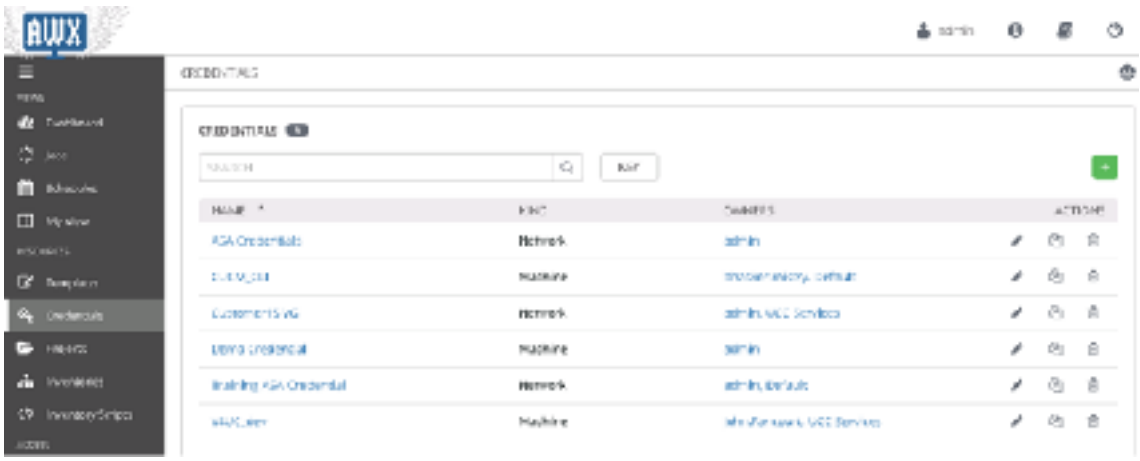
Please refer to the Ansible documentation: https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html

vmware

https://docs.ansible.com/ansible/latest/scenario_guides/vmware_scenarios/vmware_inventory.html

Credentials

For AWX, click on Credentials tab and add button .



A bunch of credentials type are supported such as AWS, Machine, Google Cloud, Azure, Network, VMware, Vault, etc.

Playbooks

Playbooks are the heart of Ansible. They can contain other playbooks, roles and/or tasks.

```
---
- hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: latest
      notify: restart apache

    - name: Create an index
      file:
        state: touch
        path: /var/www/html/index.html

    - name: Add line to index if not present
      lineinfile:
        state: present
        path: /var/www/html/index.html
        line: '<html>My hostname is {{ansible_hostname}}</html>'

  handlers:
```

Let’s review the code line by line.

```
---
```

This line starts every playbook. The three dashes tell the interpreter that this is a YAML document.

```
- hosts: webservers
```

At this point starts the first play in the playbook. Each play is defined in a YAML as a list entry and has to have at least two keys, hosts and tasks (or roles).

These entries can be in any order as long as they are present. The hosts entry can either be a individual host in the inventory or an inventory group.

```
become: yes
```

This line will have the playbook run as root.

```
tasks:
  - name: Install Apache
    apt:
      name: apache2
      state: latest
    notify: restart apache
```

The tasks section contains all of the tasks that will run on the hosts defined earlier. Each separate task is a list entry under this section and contain one or more modules.

```
- name: Add line to index if not present
  lineinfile:
    state: present
    path: /var/www/html/index.html
    line: '<html>My hostname is {{ansible_hostname}}</html>'
```

This is an example of a variable ({{ansible_hostname}}). Variables can be set by a variety of sources including the command line and the inventory file.

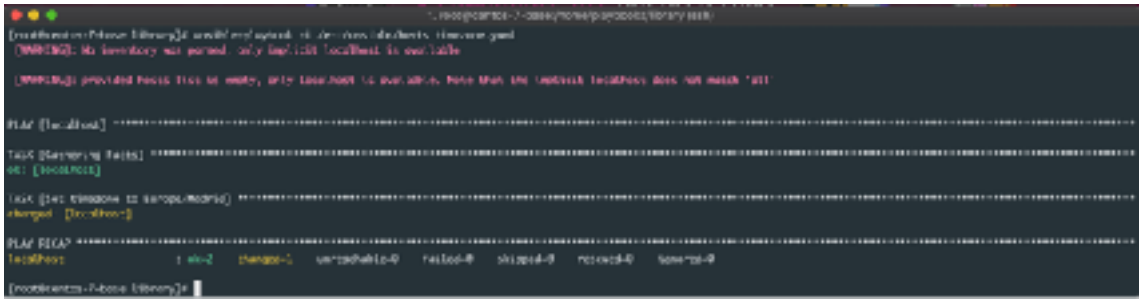
This particular variable will be set when Ansible runs through the fact gathering phase of the host, which provides the playbook run with a wide range of information that you can use in your playbooks.

Finally, the playbook uses handlers. The line "notify: restart apache" in the first task triggers this handler if there was a change that was required to be made by that task. This means if Apache wasn't installed or a new version was found the handler will be run and apache will be restarted.

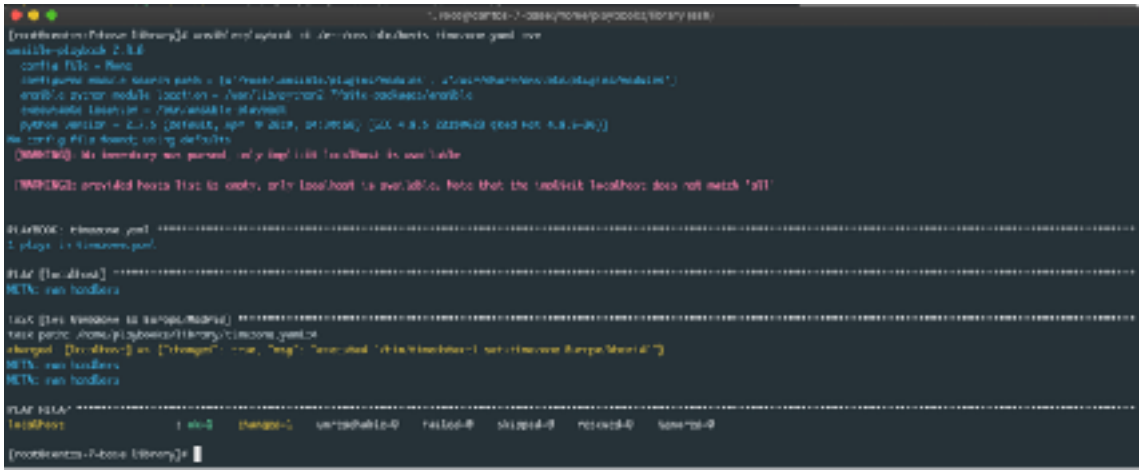
Running Playbooks Deep Dive

Ansible Engine

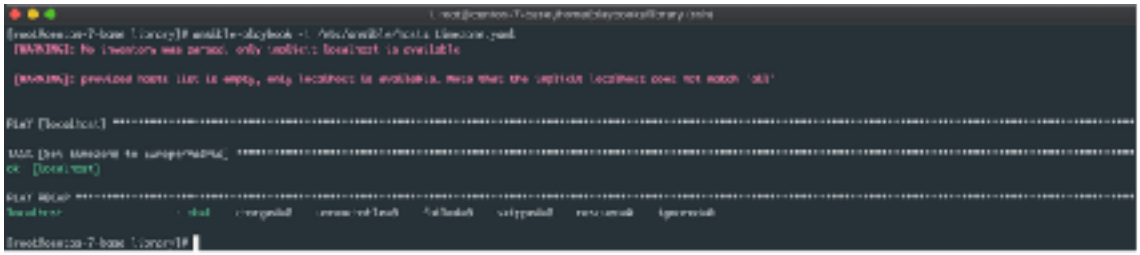
#ansible-playbook -i /path/inventory_file /path/playbook.yaml



One of the first options anyone picks up is the debug option. To understand what is happening when you run the playbook, you can run it with the verbose (-v) option. Every extra v will provide the end user with more debug output.



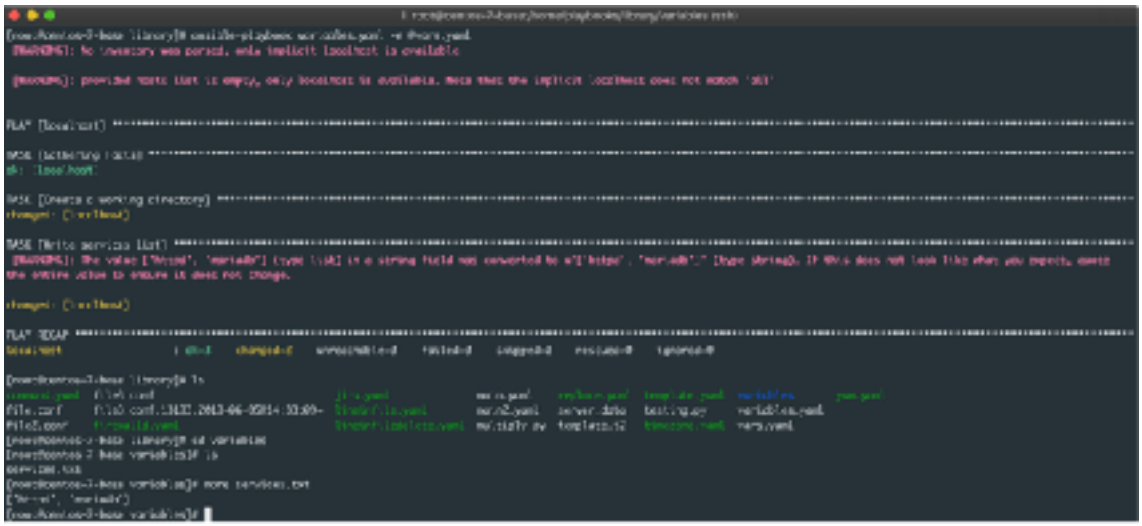
If you run the playbook again, the execution is successful but no changes are made (changed=0).



Variables

You define variable in files as mentioned above when using ansible-vault and in the vars section of a play, where you are defining the variable for the set of hosts in the play.

This playbooks just fetches the variables from the file and displays the output.

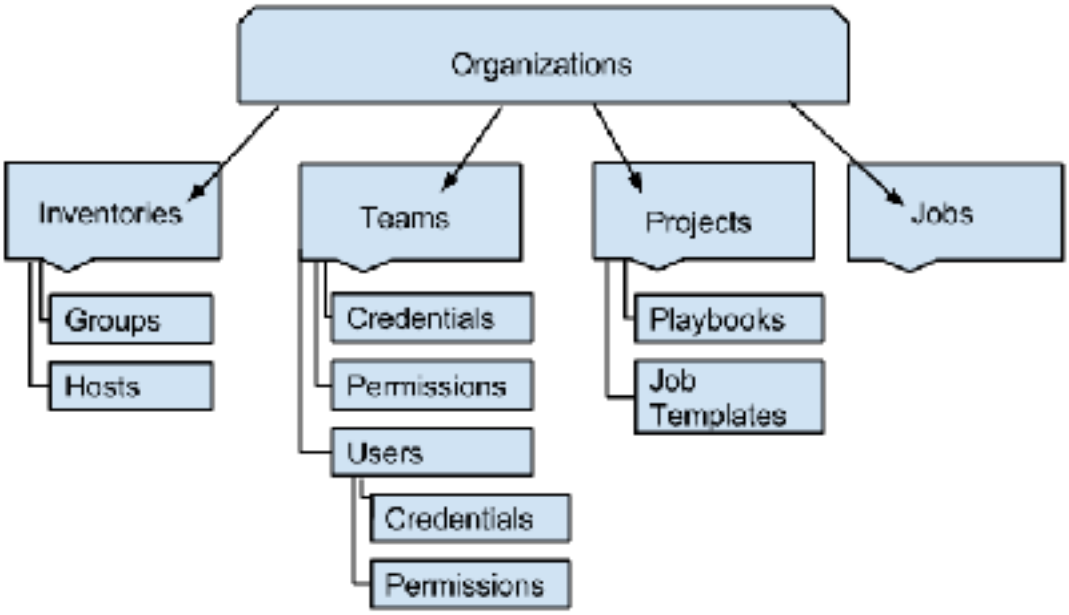


AWX

Let´s introduce some concepts before getting deeper on how to run playbooks with AWX.

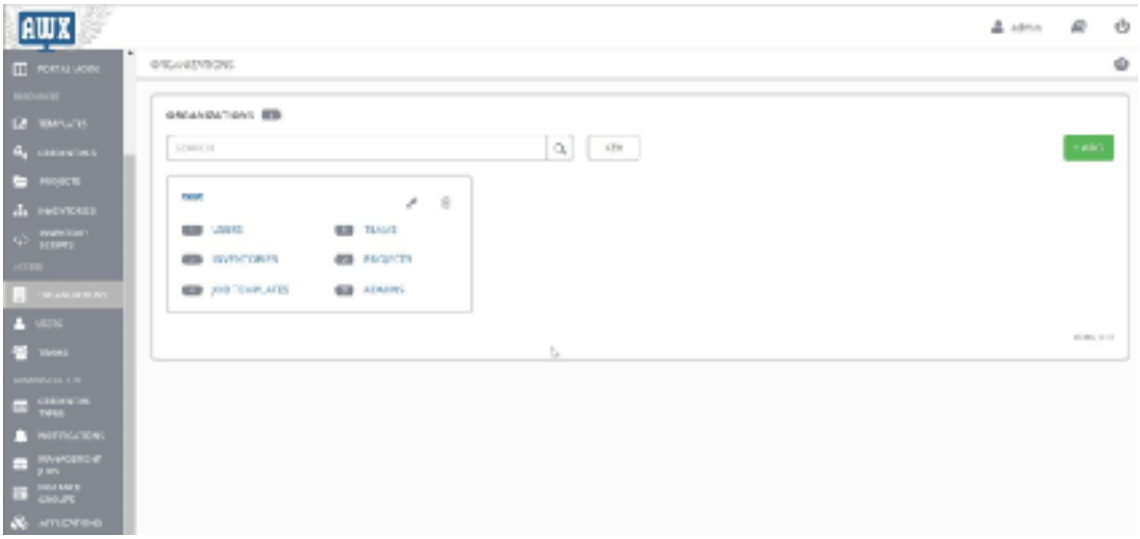
Organizations

Ansible AWX offers multilevel access delegation and role-based access control to the organization. An organization is a logical collection of Users, Teams, Projects, and Inventories. It is the highest level in the AWX object hierarchy. To support multiple clients from one umbrella, you could create an organization for each client and manage multiple teams under that. The picture below depicts the AWX organization hierarchy:

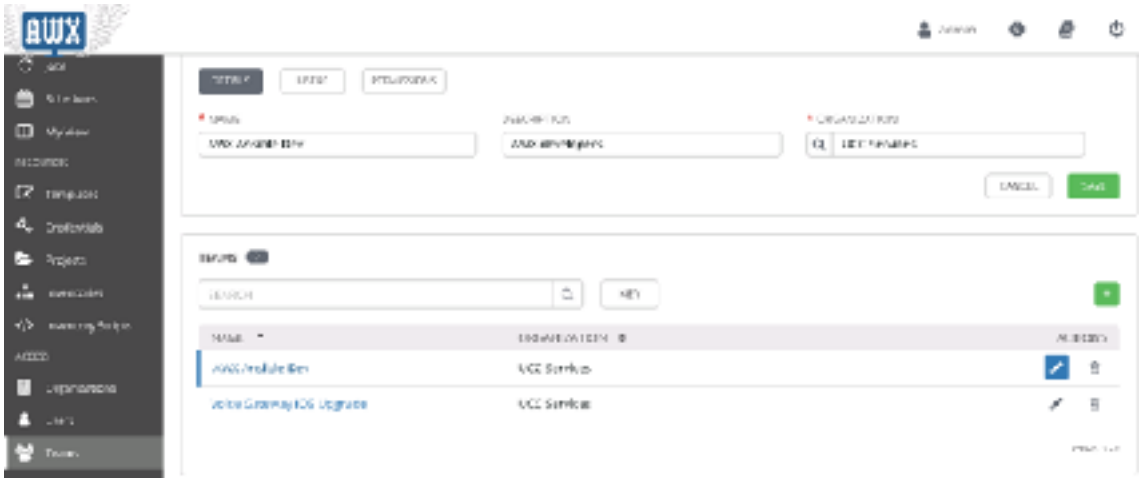


In order to create a new organization, login to AWX console with admin privileges.

Select “Organization” from the navigation and click on add . Enter the organization name and click on Save. Selecting the instance group (requires clustering) might be useful on large deployment.

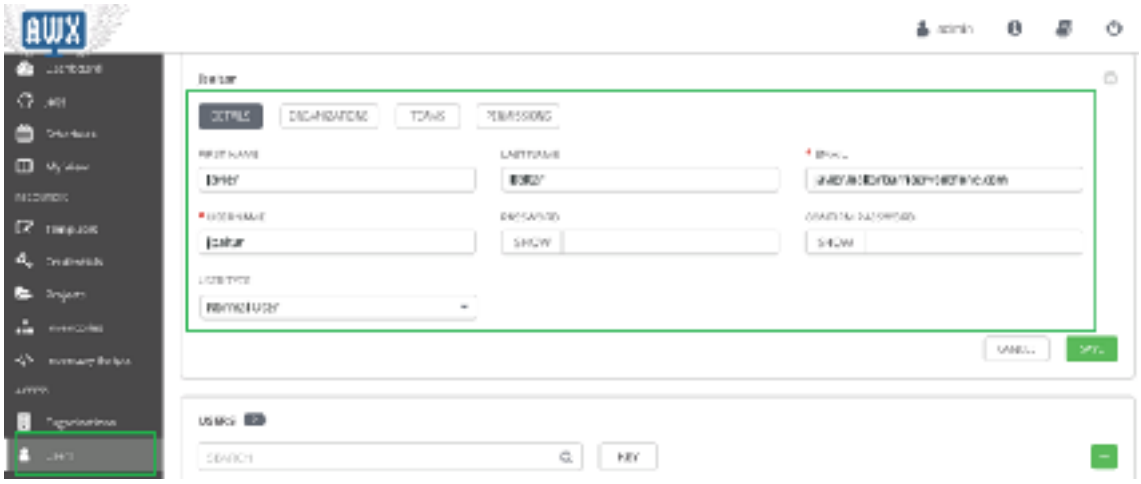


In order to create a new team, navigate to “Teams” from the menu. Click on to add a new team. Enter the team name and select the organization (you can add multiple teams based on the requirements).

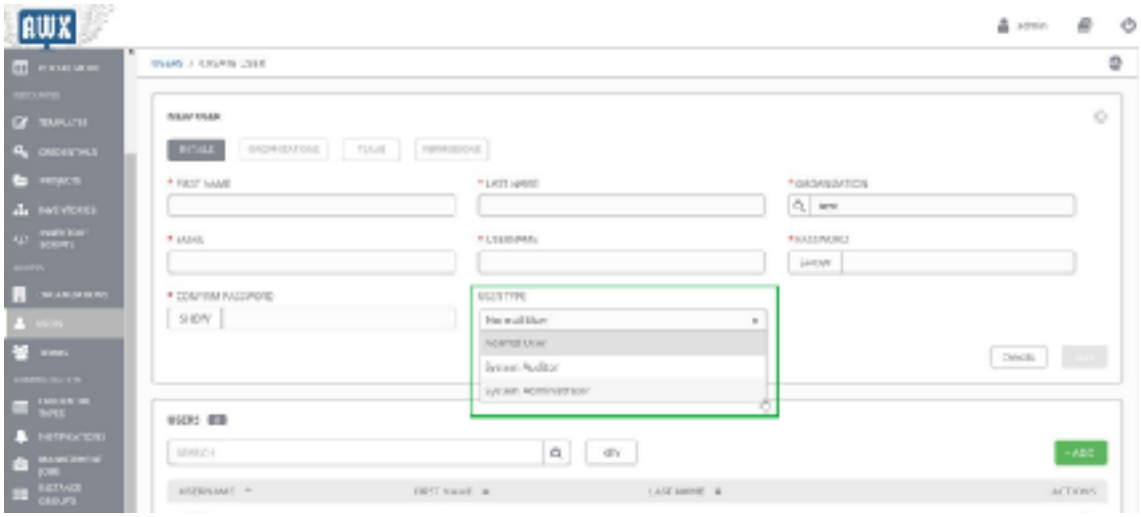


User Management

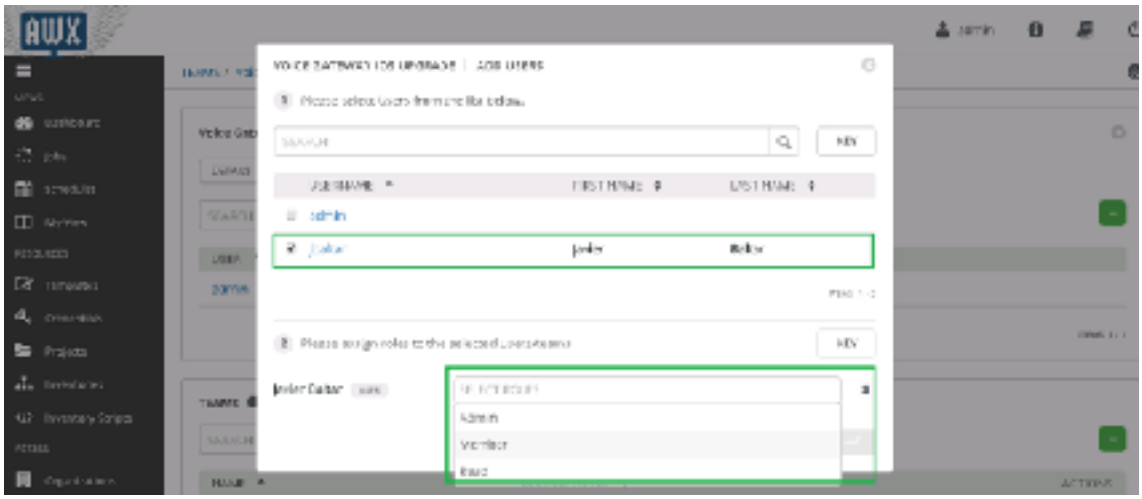
From the AWX main menu,you can aad users to current organization. Select “users” and click on . Enter the user details and select the destination organization.



There are three user types: Normal User, System Auditor and System Administrator as shown below:



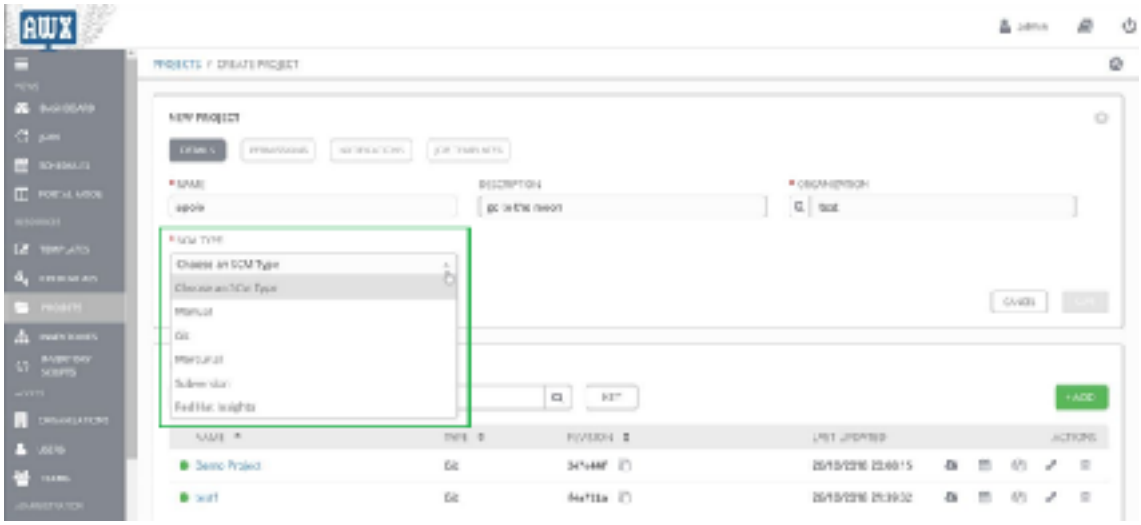
From the Teams menu, you can add users to teams assigning the proper role (admin, member or read only).



Project

Next, you configure a project, the project contains the SCM type, and playbook directory which you must select. The SCM type can be manual but other options such as Git are available for central management of scripts.

The playbook directory drop down should contain the sub directories in the root projects folder. Select the relevant folder for this project that contains all the playbooks required for jobs connected to this project.

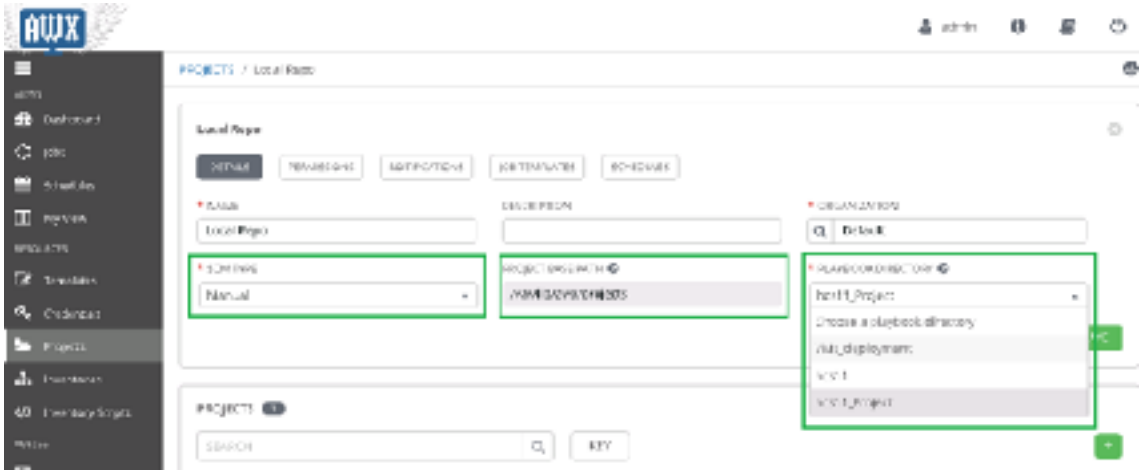


Manual Local Repo

You can create the project folder to store the playbooks in the following local path:

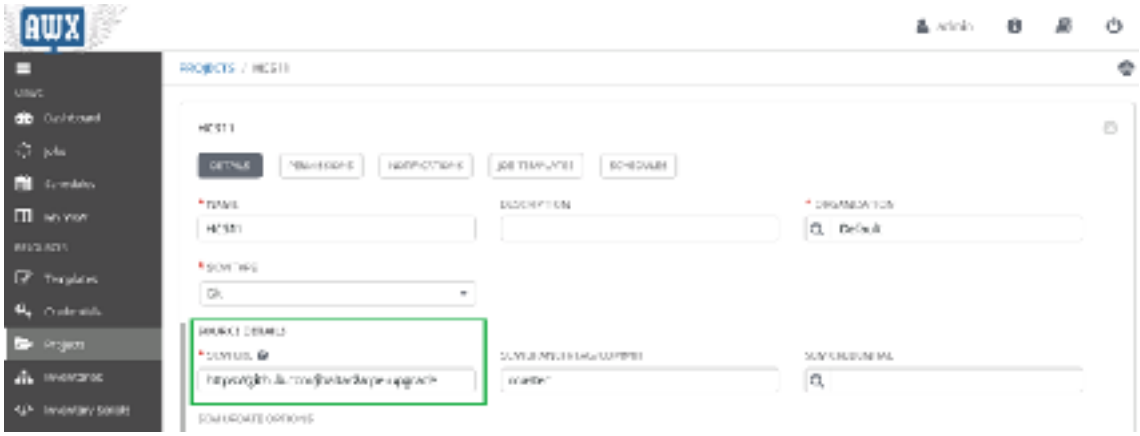
cd /var/lib/awx/projects/

mkdir my_project_name



Git Repo

A externat Git repository can be configured at Project level as shown below:



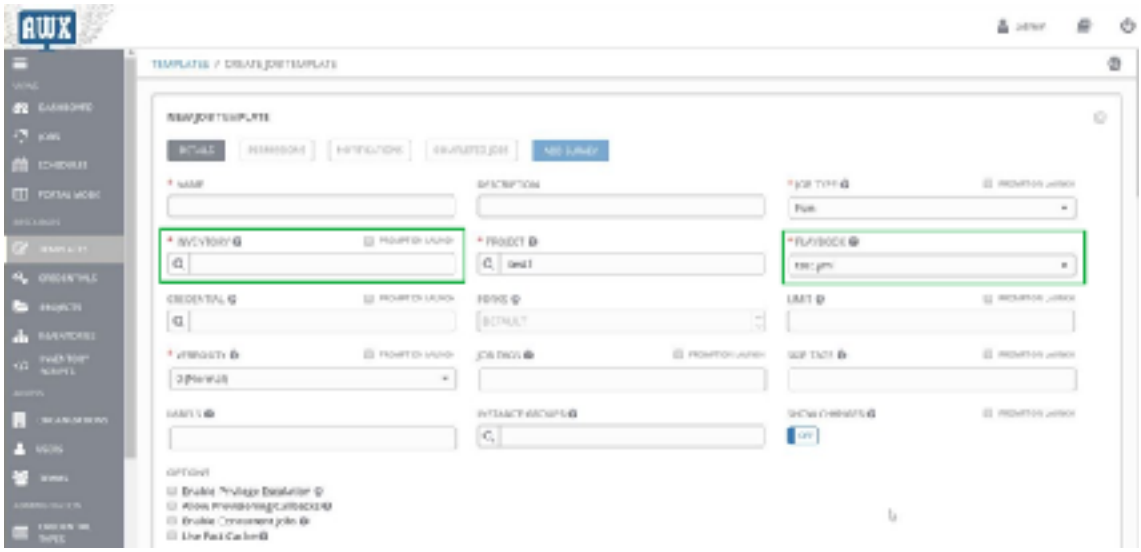
The Projects tab displays the status of the repo, last revision and the sync button.

Templates

Job Template

A job template is a definition and set of parameters for running an Ansible job. Job templates are useful to execute the same job many times and reuse of Ansible playbook content between teams.

To create a new job template, click the add button then select "Job Template" from the menu list.



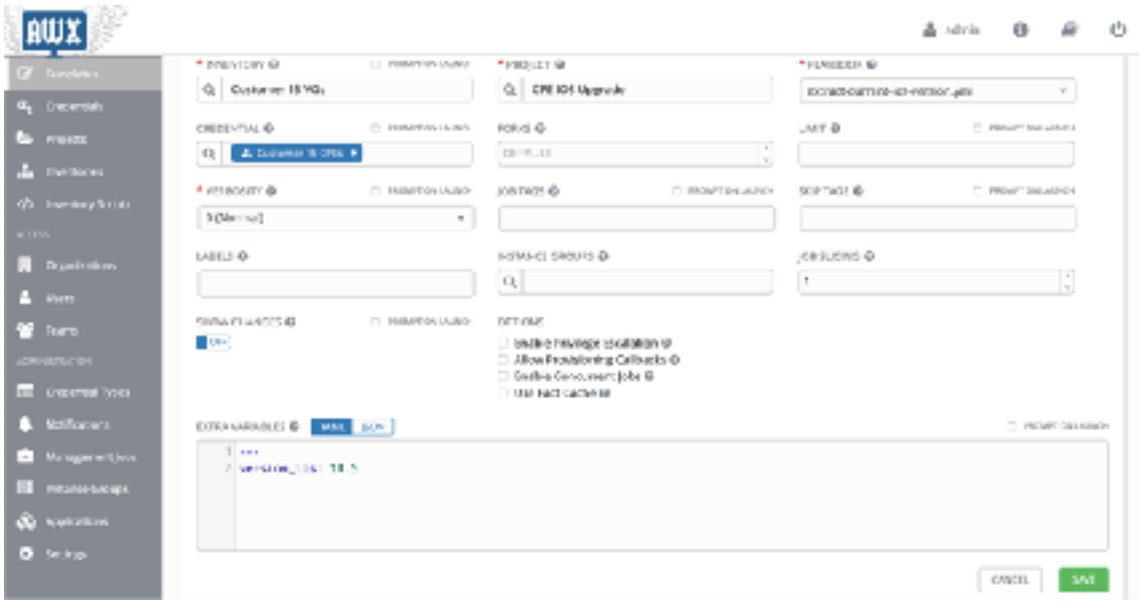
Enter the appropriate details into the following fields:

- Name: Enter a name for the job.
- Description: Enter an arbitrary description as appropriate (optional).
- Job Type:
    - Run: Execute the playbook when launched, running Ansible tasks on the selected hosts.
    - Check: Perform a “dry run” of the playbook and report changes that would be made without actually making them. Tasks that do not support check mode will be skipped and will not report potential changes.
        - Prompt on Launch – If selected, even if a default value is supplied, you will be prompted upon launch to choose a job type of run or check.
- Inventory: Choose the inventory to be used with this job template from the inventories available to the currently logged in AWX user.
    - Prompt on Launch – If selected, even if a default value is supplied, you will be prompted upon launch to choose an inventory to run this job template against.
- Project: Choose the project to be used with this job template from the projects available to the currently logged in AWX user.
- Playbook: Choose the playbook to be launched with this job template from the available playbooks. This menu is automatically populated with the names of the playbooks found in the project base path for the selected project.
- Credential: Click the search button to open a separate window. Choose the credential from the available options to be used with this job template. Use the drop-down menu list to filter by credential type if the list is extensive.
    - Prompt on Launch: If selected, upon launching a job template that has a default machine credential, you will not be able to remove the default machine credential in the Prompt dialog without replacing it with another machine credential before it can launch.
- Forks: The number of parallel or simultaneous processes to use while executing the playbook. A value of zero uses the Ansible default setting, which is 5 parallel processes unless overridden in /etc/ansible/ansible.cfg.
- Limit: A host pattern to further constrain the list of hosts managed or affected by the playbook. Multiple patterns can be separated by colons (”:”). As with Ansible engine, “a:b” means “in group a or b”, “a:b:&c” means “in a or b but must be in c”, and “a:!b” means “in a, and definitely not in b”.

    - Prompt on Launch: If selected, even if a default value is supplied, you will be prompted upon launch to choose a limit.

- Verbosity: Control the level of output Ansible produces as the playbook executes. Set the verbosity to any of Default, Verbose, or Debug. This only appears in the “details” report view. Verbose logging includes the output of all commands. Debug logging is exceedingly verbose and includes information on SSH operations that can be useful in certain support instances.
    - Prompt on Launch: If selected, even if a default value is supplied, you will be prompted upon launch to choose a verbosity.
- Job Tags: Provide a comma-separated list of playbook tags to specify what parts of the playbooks should be executed.
    - Prompt on Launch – If selected, even if a default value is supplied, you will be prompted upon launch to choose a job tag.
- Skip Tags: Provide a comma-separated list of playbook tags to skip certain tasks or parts of the playbooks to be executed.
    - Prompt on Launch – If selected, even if a default value is supplied, you will be prompted upon launch to choose tag(s) to skip.
- Labels: Supply optional labels that describe this job template, such as “dev” or “test”. Labels can be used to group and filter job templates and completed jobs in the AWX display.
    - Labels are created when they are added to the Job Template. Labels are associated to a single Organization using the Project that is provided in the Job Template. Members of the Organization can create labels on a Job Template if they have edit permissions (such as admin role).
    - Once the Job Template is saved, the labels appear in the Job Templates overview.
    - Click on the “x” beside a label to remove it. When a label is removed, and is no longer associated with a Job or Job Template, the label is permanently deleted from the list of Organization labels.
    - Jobs inherit labels from the Job Template at the time of launch. If a label is deleted from a Job Template, it is also deleted from the Job.
- Instance Groups: Click the search button to open a separate window. Choose the instance groups on which you want to run this job template. If the list is extensive, use the search to narrow the options.
- Show Changes: Allows you to see the changes made by Ansible tasks.
    - Prompt on Launch – If selected, even if a default value is supplied, you will be prompted upon launch to choose whether or not to show changes.
- Options: Supply optional labels that describe this job template, such as “dev” or “test”. Labels can be used to group and filter job templates and completed jobs in the AWX display.
    - Enable Privilege Escalation: If enabled, run this playbook as an administrator. This is the equivalent of passing the --become option to the ansible-playbook command.
    - Allow Provisioning Callbacks: Enable a host to call back to AWX via the AWX API and invoke the launch of a job from this job template.
    - Enable Concurrent Jobs: Allow jobs in the queue to run simultaneously if not dependent on one another.
    - Use Fact Cache: When enabled, AWX will activate an Ansible fact cache plugin for all hosts in an inventory related to the job running.
- Extra Variables:
    - Pass extra command line variables to the playbook. This is the “-e” or “–extra-vars” command line parameter for ansible-playbook that is documented in the Ansible documentation at Passing Variables on the Command Line.

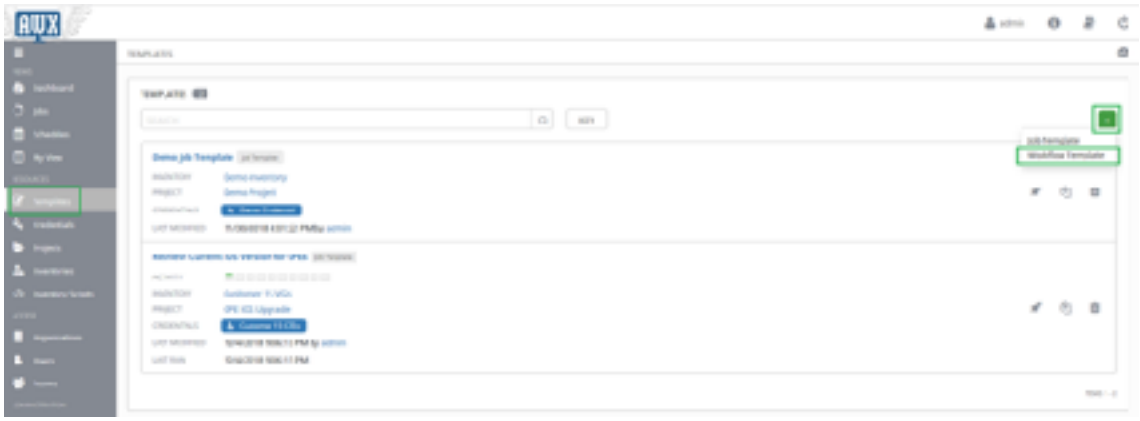
- Provide key/value pairs using either YAML or JSON. These variables have a maximum value of precedence and overrides other variables specified elsewhere.



Workflow Template

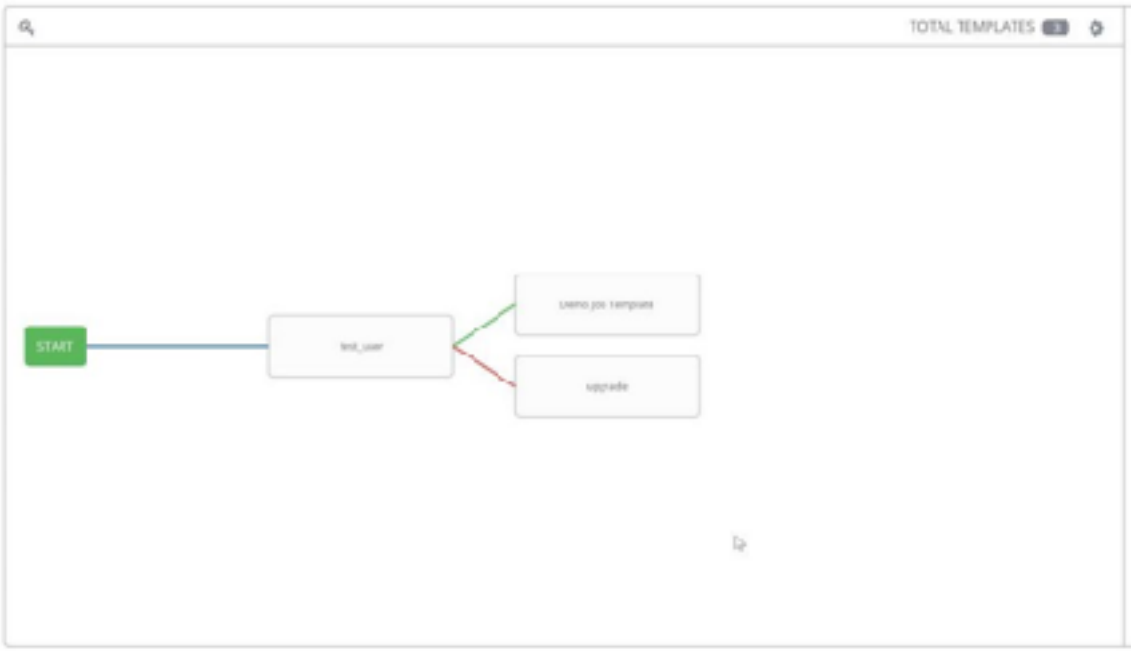
This feature in AWX enables users to create sequences consisting of any combination of job templates, project syncs, and inventory syncs that are linked together in order to execute them as a single unit. Another reason workflows are useful is because they allow the user to take any number of playbooks with the ability to make a decision tree depending on a job's success or failure and sending notifications also.

From the Templates meny, click on and make sure to select "Workflow Template".



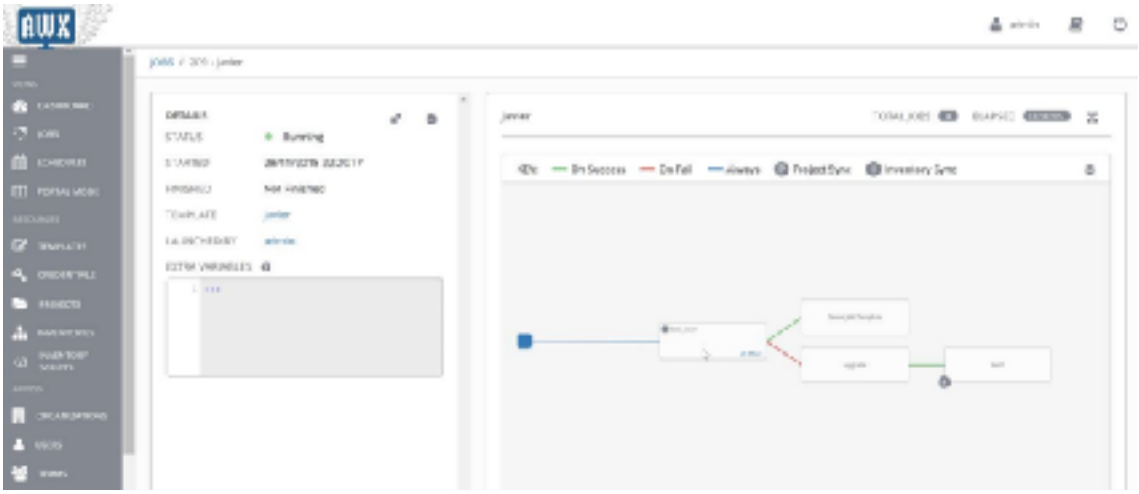
Once the mandatory fields are filled, click on Save and on Workflow Visualizer:

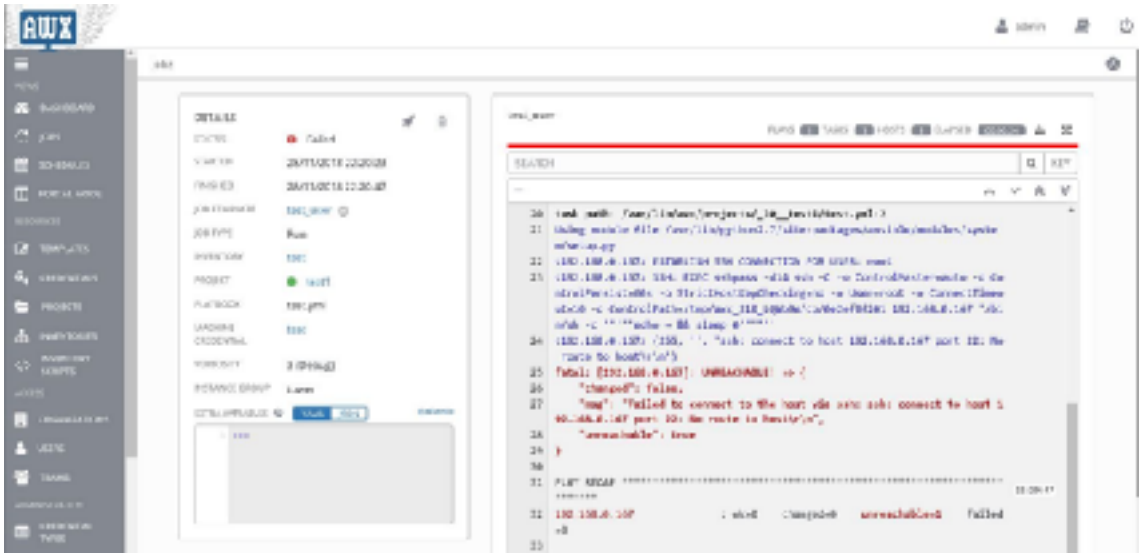
This screen will come up, where you can add different job templates and make sure they run on failure, success, or with either outcome.



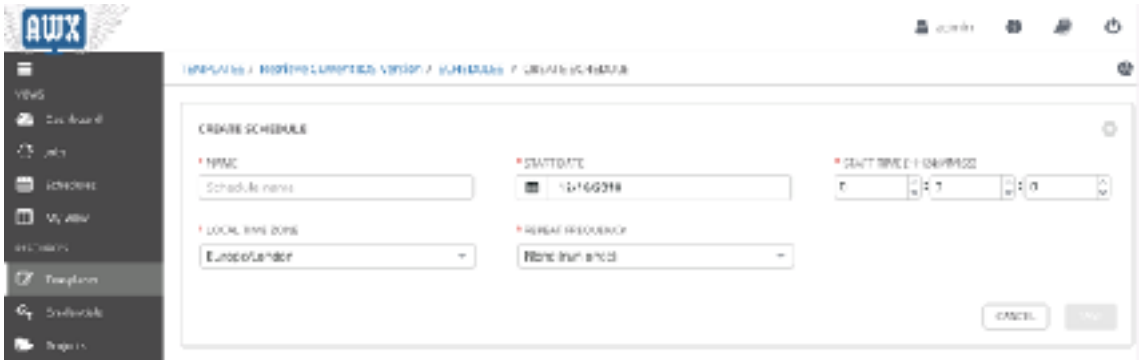
Note that you can decide if things run on success, on failure, or always.

After everything is set and saved, you are ready to launch your template, which you can do by clicking on the rocket icon next to the workflow you would like to run.





You can schedule your workflows to run when you need them to. Click on the calendar icon next to any workflow job template:

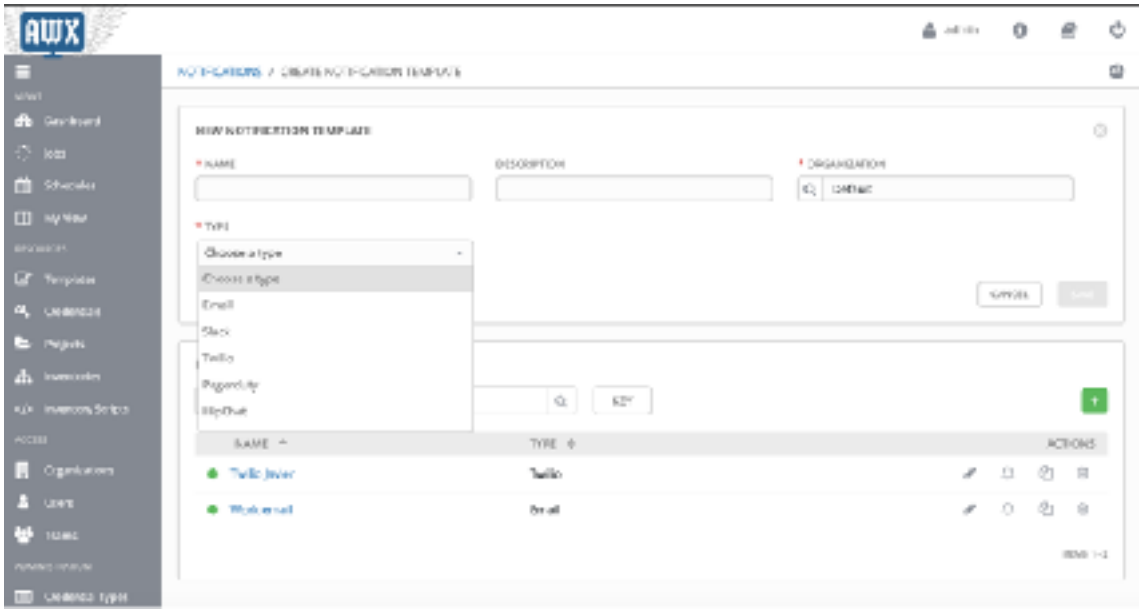


If you need to set extra variables for the playbooks involved in a workflow template and/or allow for authorization of user input, then setting up surveys is the way to go.

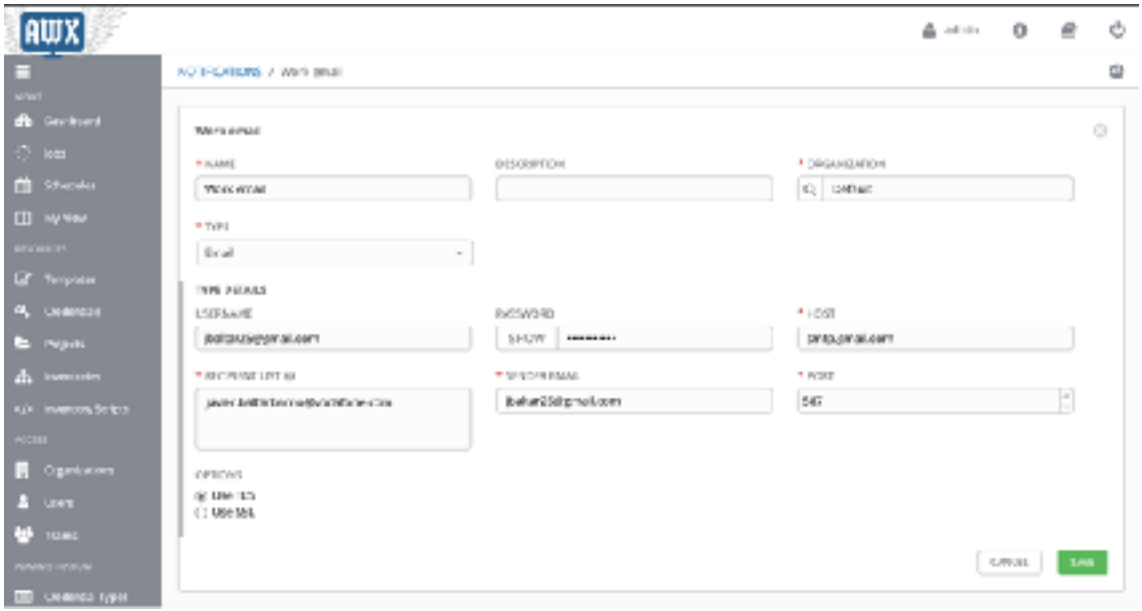
In order to set one up, select a workflow template and click on the “Add Survey” button:

Notifications

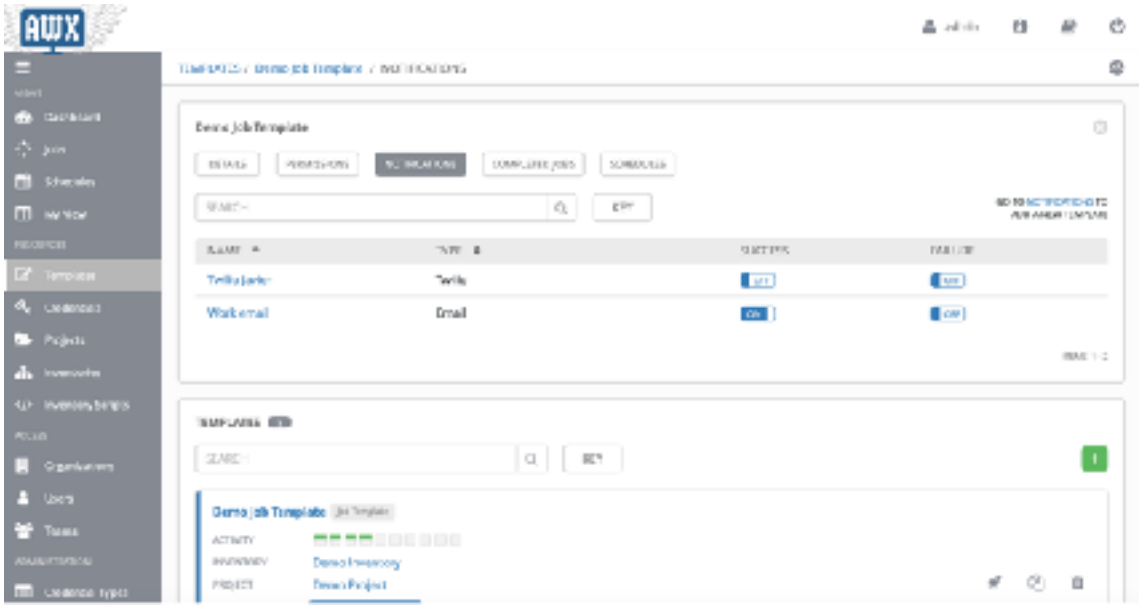
AWX provides a bunch of notification channels: email, Slack, Twilio, etc.



Email
Click on Notification and add a new one:



From the Templates tab, you can control the notifications that are triggered by an specific template when it is successfully completed or fails.

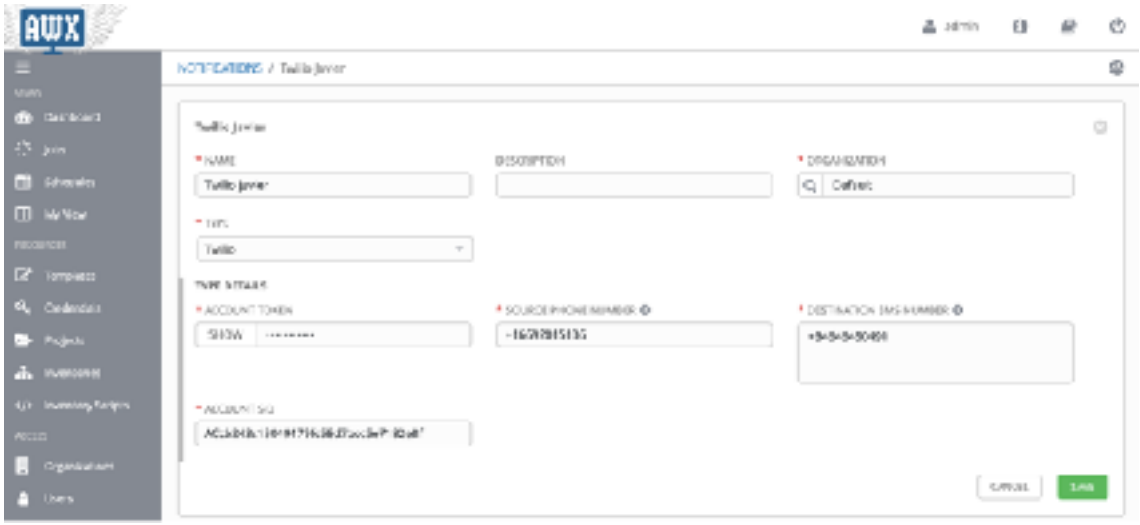


This is an example of email notification received to your mailbox.

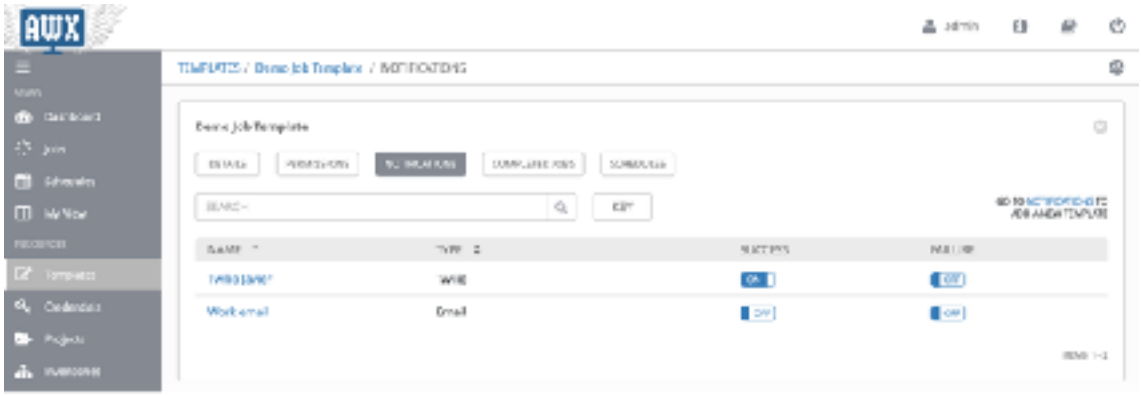
Twilio

To trigger a Twilio API call, you need to configure the following items in your account:
Click on dashboard > Twilio account details, where Account SID and AUTH TOKEN are listed.
Click on Twilio verified called IDs, which contains the allowed destination numbers.
Click on Geo Restrictions in order to ensure your destination number is not restricted:

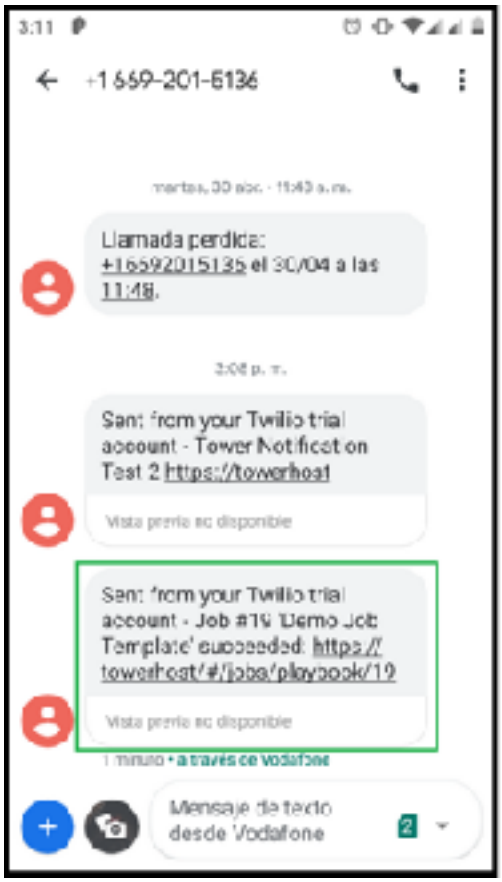
You have to create your Twilio notification in AWX:



Associate your template with the notification:



After running your job, the SMS notification below is dropped to my mobile:



Advanced Playbooks Features

Error Handling

Blocks allow you to group related tasks together and apply particular task parameters on the block level. They also allow you to handle errors inside the blocks in a way similar to most programming languages exception handling.

This example playbook that uses blocks to run group of tasks specific to one platform. If you want to perform a series of tasks with one set of task parameters (e.g. with_items, when, or sudo) applied, blocks are very handy.

Blocks are also useful if you want to be able to gracefully handle failures. There might be a non-critical task that is not essential for a deployment to succeed, so it would be better to gracefully handle a failure than to stop the entire deployment rollout.

This is an example of how to use a block to gracefully handle task failures:

```
tasks:
 - block:
    - name: Shell script to deploy a critical service.
      script: deploy_application.sh
   rescue:
    - name: This will only run in case of an error in the block.
      debug: msg="There was an error in the block."
   always:
    - name: This will always run
      debug: msg="This always executes."
```

Tasks inside the block will be run first. If there is a failure in any task in block, tasks inside rescue will be run. The tasks inside always will always be run, whether or not there were failures in either block or rescue.

Please refer to Ansible documentation: https://docs.ansible.com/ansible/latest/user_guide/playbooks_blocks.html#error-handling

Handlers

A Handler is exactly the same as a task, but it will run when called by another task. A Handler will take an action when called by an event it listens for.

This is useful for secondary actions that might be required after running a task, such as starting a new service after installation or reloading a service after a configuration change.

```
---
- hosts: webservers
  tasks:
   - name: Install Nginx
     apt: pkg=nginx state=installed update_cache=true
     notify:
       - start Nginx
  handlers:
   - name: start Nginx
     service: name=nginx state=started
```

Templates and Variables

A template in Ansible is a file which contains all your configuration parameters, but the dynamic values are given as variables. During the playbook execution, depending on the conditions like which cluster you are using, the variables will be replaced with the relevant values.

```
- hosts: webservers
  vars:
    variable_to_be_replaced: 'value1'
    inline_variable: 'value2'
  tasks:
   - name: Ansible using templates
     template:
      src: template_example.j2
      dest: /userA/docs/personal_data.txt
```

Template file

```
template_example.j2

{{ variable_to_be_replaced }}
Variable given as inline - {{ inline_variable }} -
```

This is the output of the playbook above:

```
value1
Variable given as inline - value2 -
```

Loops

With_items

```
- name: Remove users from the system.
  user:
    name: "{{ item }}"
    state: absent
    remove: yes
  with_items:
    - userA
    - userB
```

With_nested
Define variables

```
users_with_items:
  - name: "userA"
    personal_directories:
      - "old_files"
```

Playbook

```
- name: Create common users directories using
  file:
   dest: "/home/{{ item.0.name }}/{{ item.1 }}"
   owner: "{{ item.0.name }}"
   group: "{{ item.0.name }}"
   state: directory
 with_nested:
   - "{{ users_with_items }}"
   - "{{ common_directories }}"
```

File Manipulation

Useful ad-hoc commands for file manipulation:

Delete a File

```
#Delete the file /backups/tmp/nodelist.txt on all servers
ansible all -b -m file -a "state=absent path=/backups/tmp/nodelist.txt"
```

Update a Line in File

```
#Update the line of text "MY_SETTING" to "BLUE" in /opt/configuration.txt
on all servers
ansible all -b -m lineinfile -a "regexp=MY_SETTING line=BLUE path=/opt/
configuration.txt"
```

Extract /tmp/package.tgz to /opt/ on all hosts in webservers

```
ansible webservers -b -m unarchive -a "src=/tmp/package.tgz dest=/opt/
remote_src=yes"
```

Set the group ownership of a directory on each host in webservers

```
ansible webservers -b -m file -a "recurse=yes sate=directory path=/opt/
automators/secrets group=protected"
```

Modules

Please refer to https://docs.ansible.com/ansible/latest/modules/modules_by_category.html for Ansible module index.

Files

Archive

Some examples using the archive modules:

```
tasks:
- name: Backup Directory /var/log/application01/
- archive:
  path: /var/log/application01/
  dest: "/var/backups/application01-{{ ansible_date_time.date }}.tgz"
```

Copy

```
tasks:
- name: Copy File
- copy:
    src: /var/log/application01/filename
    dest: "/var/backups/filename
    owner: root
    group: root
    mode: u=r,g=r,o=
```

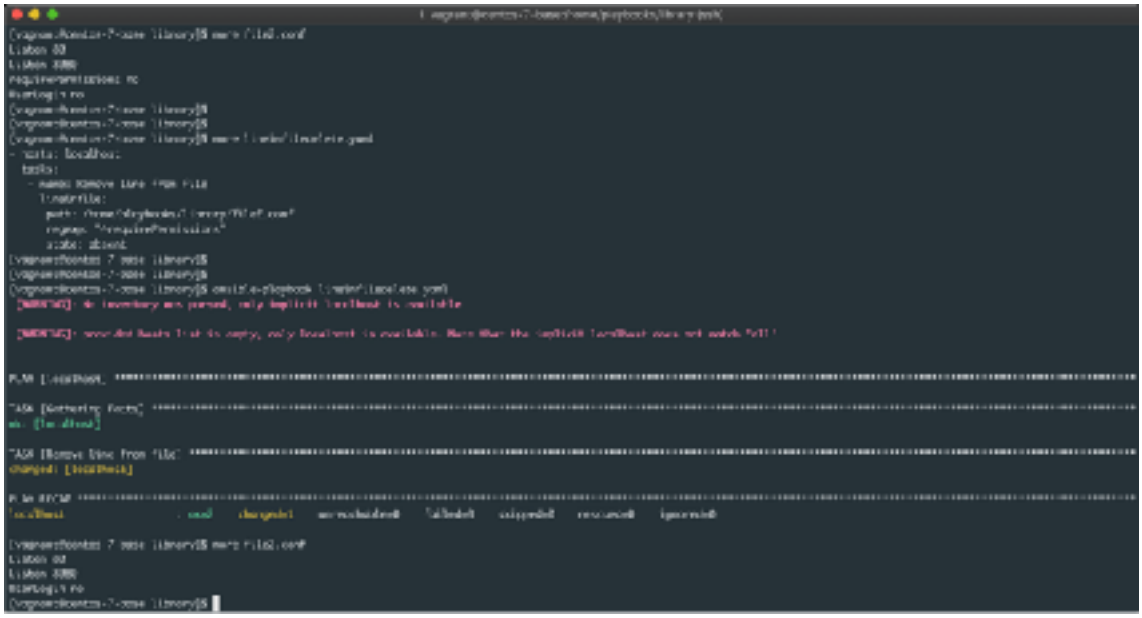
Fetch

```
tasks:
- name: Copy File from Remote Node

- fetch:
    src: /var/log/application01/filename
    dest: "/var/backups/
```

Lineinfile

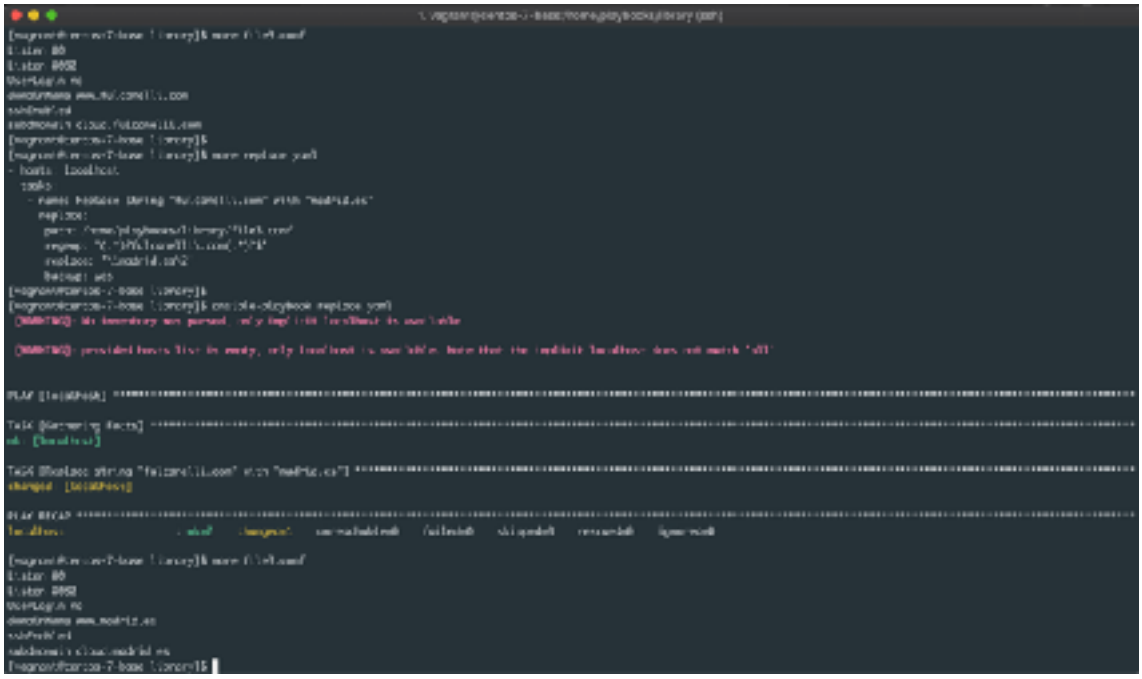
The following playbook add a line after a string:

The following playbook deletes a line from a file:



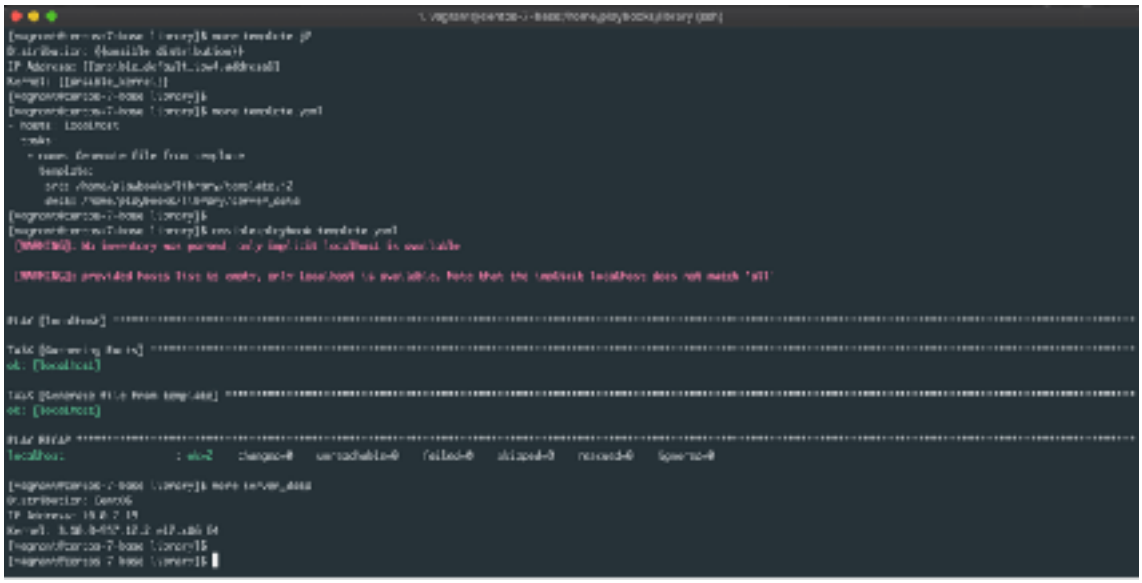
Replace

This is an example of playbook using the replace module.



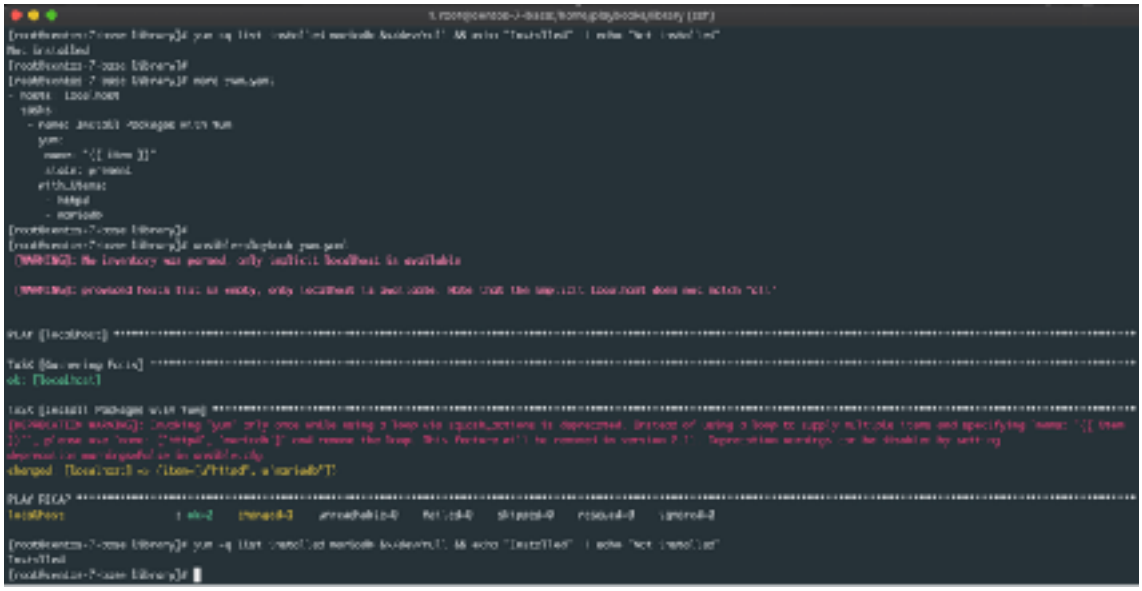
Template

Example playbook using the template module.



Yum

Example playbook using the yum module for installing https and mariadb packages.



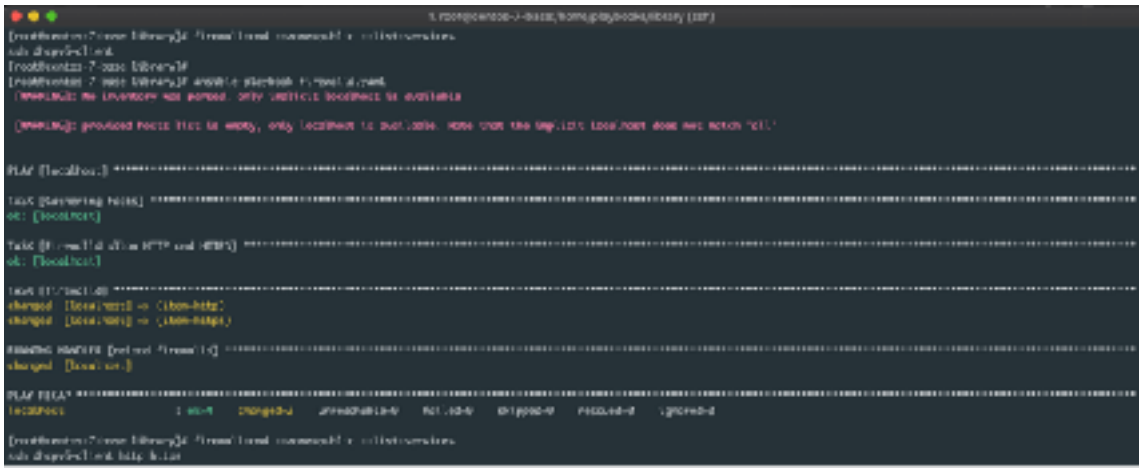
Command

Example playbook using the raw command module.

System

Firewalld

Example playbook using the firewalld module allowing ports 80 and 443.



Timezone

This playbook uses the timezone module for setting the timezone.

Jira

This playbook creates a task in Jira.

Notifications

Slack

```
- name: Sending message to Slack Channel
  slack:
    token: '{{ slack_token }}'
    channel: "#companynameAnsible"
    domain: "companyname.slack.com"
    parse: "full"
    color: "good"
    msg: 'The changes is completed on {{ inventory_hostname }}.'
```

Twilio

```
- name: Send an SMS to multiple phone numbers when the change is
completed
  twilio:
    msg: The configuration change is completed!
    account_sid: XXXXXXXXXXX
    auth_token: XXXXXXXXXXX
    from_number: +34XXXXXXXXX
    to_number:
      - +34XXXXXXXX1
      - +34XXXXXXXX2
  delegate_to: localhost
```

NOTE: The delegate_to: localhost is often required when interworking with APIs.

connection: local runs your entire playbook play locally.

delegate_to: localhost runs a specific task on the localhost.

Custom Modules: Writing your own module

Folder Structure

Ansible requires that the module is stored at /library/custom_module_name.py at the same level where the playbook is located.

In the following example, the playbook is called "main.yaml" and the customer module "multiply.py" is located in the /library/ folder.

Ansible recommends to include strings for DOCUMENTATION and EXAMPLES at the top of our module code.

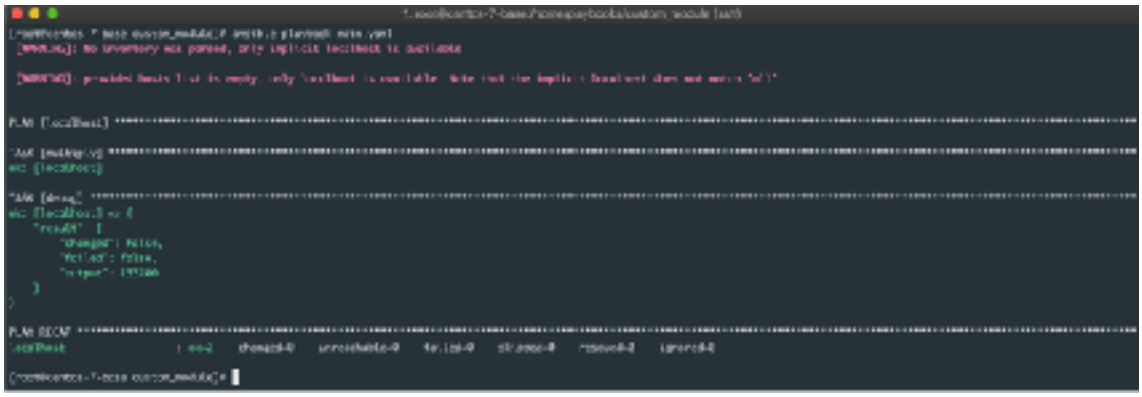
```
DOCUMENTATION = '''
---
module: multiply
short_description: Multiply  two given numbers
'''


EXAMPLES = '''
tasks:
- multiply:
    a: 200
    b: 10
  register: result
- debug: var=result
'''
```

The key part is to import the boilerplate code from ansible.module_utils.basic like this:

```
from ansible.module_utils.basic import AnsibleModule
if __name__ == '__main__':
    main()
```

The AnsibleModule provides lots of common code for handling returns, parses your arguments for you, and allows you to check inputs.

Let's run the playbook and check the result:



Roles

Roles are nothing but directories laid out in a specific manner. Roles follow predefined directory layout conventions and expect each component to be in the path meant for it. A role directory structure contains the directories below. Each directory must contain a main.yml which is the default file.

[root@centos-7-base unxnn.mysql]# ls -lrt
total 12
-rw-rw-r--. 1 root root 7657 May 8 11:03 README.md
-rw-rw-r--. 1 root root 1061 May 8 11:03 LICENSE
drwxr-xr-x. 2 root root 22 Jun 14 12:04 handlers
drwxr-xr-x. 2 root root 22 Jun 14 12:04 defaults
drwxr-xr-x. 3 root root 21 Jun 14 12:04 molecule
drwxr-xr-x. 2 root root 67 Jun 14 12:04 templates
drwxr-xr-x. 2 root root 143 Jun 14 12:04 tasks
drwxr-xr-x. 2 root root 64 Jun 14 12:04 vars
drwxr-xr-x. 2 root root 39 Jun 14 12:04 tests
drwxr-xr-x. 2 root root 50 Jun 14 12:04 meta

Defaults: default variables for the role
Vars: variables for the role
Tasks: the main list of steps to be executed by the role
Files: contains files which we want to be transferred to the host
Templates: file template which supports modifications from the role
Meta: contains metadata of role such as dependencies
Handlers: handlers which can be invoked by notify directives

```
-- unxnn.mysql
   |-- defaults
   |   `-- main.yml
   |-- handlers
   |   `-- main.yml
   |-- LICENSE
   |-- meta
   |   `-- main.yml
   |-- molecule
   |   `-- default
   |       |-- molecule.yml
   |       |-- playbook.yml
   |       |-- tests
   |       |   `-- test_default.py
   |       `-- yaml-lint.yml
   |-- README.md
   |-- tasks
   |   |-- configure.yml
   |   |-- main.yml
   |   |-- secure-installation.yml
   |   |-- setup-Debian.yml
   |   |-- setup-RedHat.yml
   |   `-- variables.yml
   |-- templates
   |   |-- my.cnf.j2
   |   |-- root-my.cnf.j2
   |   `-- user-my.cnf.j2
   |-- tests
   |   |-- inventory
   |   `-- test.yml
```

Let's take a glance of these files:

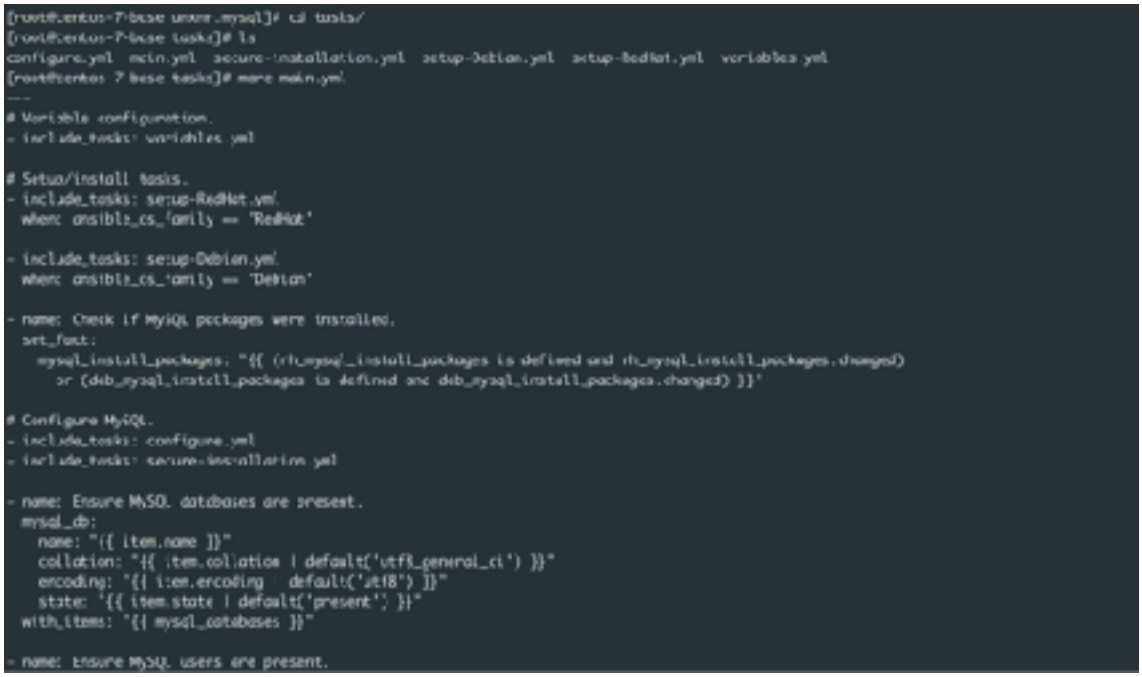
/defaults/main.yml

/handlers/mail.yml

/meta/main.yml

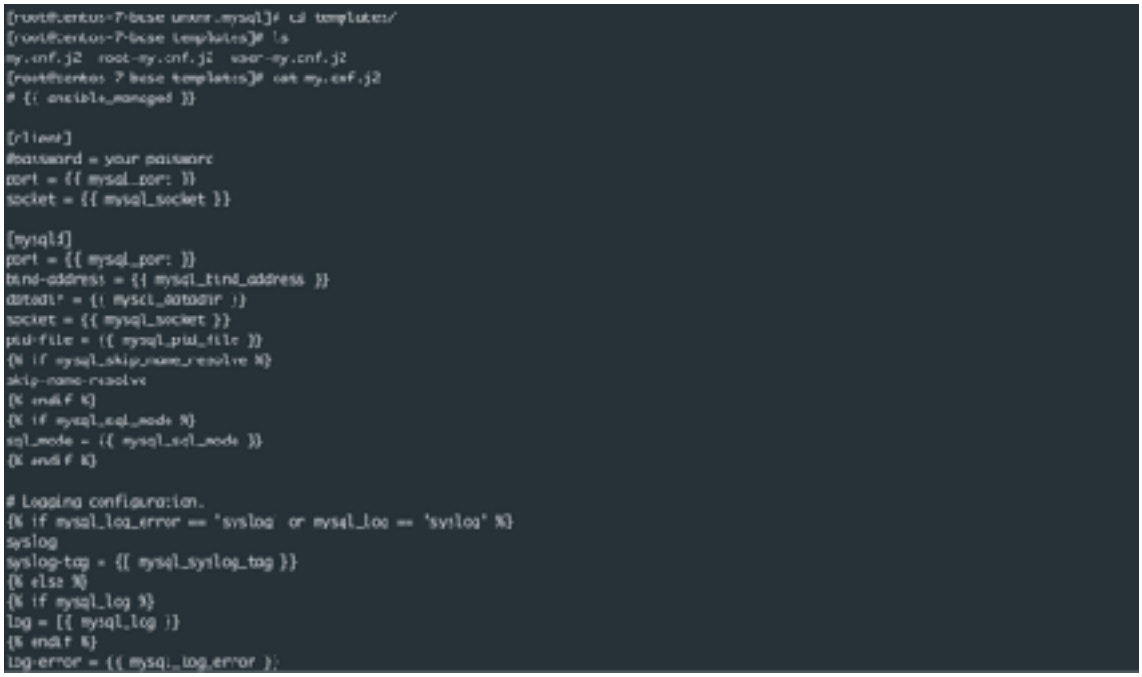
```
[root@centos-7-base meta]# pwd
/root/.ansible/roles/unam.mysql/meta
[root@centos-7-base meta]#
[root@centos 7 base meta]# ls
main.yml
[root@centos-7-base meta]# cat main.yml
---
dependencies: []

galaxy_info:
  author: unam
  role_name: mysql
  description: MySQL server for RHEL/CentOS and Debian/Ubuntu.
  license: "license (MIT)"
  min_ansible_version: 2.4
  platforms:
    - name: EL
      versions:
        - 6
        - 7
    - name: Ubuntu
      versions:
        - all
    - name: Debian
      versions:
        - all
  galaxy_tags:
    - database
    - mysql
    - mariadb
    - db
    - sql
[root@centos-7-base meta]#
```

/tasks/main.yml

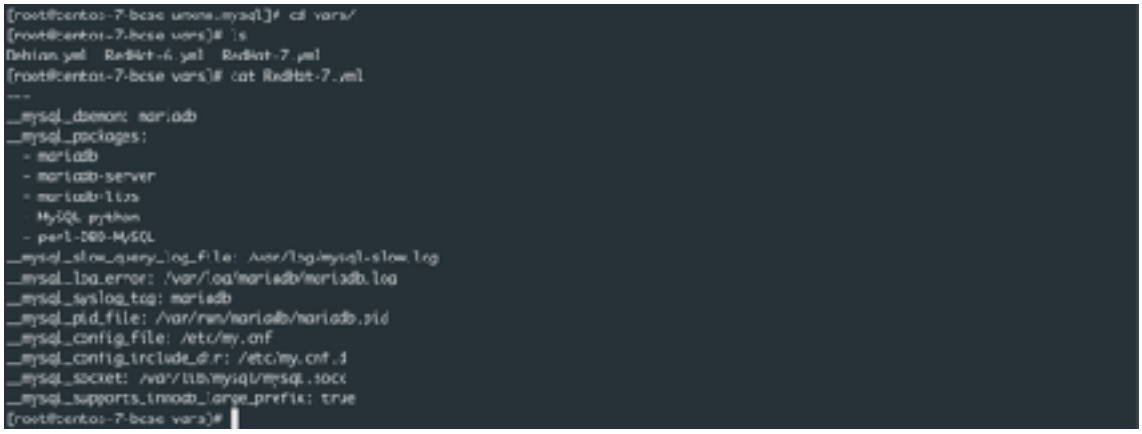


```
[root@centos-7-base unam.mysql]# cd tasks/
[root@centos-7-base tasks]# ls
configure.yml  main.yml  secure-installation.yml  setup-Debian.yml  setup-RedHat.yml  variables.yml
[root@centos 7 base tasks]# more main.yml
---
# Variable configuration.
- include_tasks: variables.yml

# Setup/install tasks.
- include_tasks: setup-RedHat.yml
  when: ansible_os_family == 'RedHat'

- include_tasks: setup-Debian.yml
  when: ansible_os_family == 'Debian'

- name: Check if MySQL packages were installed.
  set_fact:
    mysql_install_packages: "{{ (rh_mysql_install_packages is defined and rh_mysql_install_packages.changed)
      or (deb_mysql_install_packages is defined and deb_mysql_install_packages.changed) }}"

# Configure MySQL.
- include_tasks: configure.yml
- include_tasks: secure-installation.yml

- name: Ensure MySQL databases are present.
  mysql_db:
    name: "{{ item.name }}"
    collation: "{{ item.collation | default('utf8_general_ci') }}"
    encoding: "{{ item.encoding | default('utf8') }}"
    state: "{{ item.state | default('present') }}"
  with_items: "{{ mysql_databases }}"

- name: Ensure MySQL users are present.
```

/templates/



/vars/



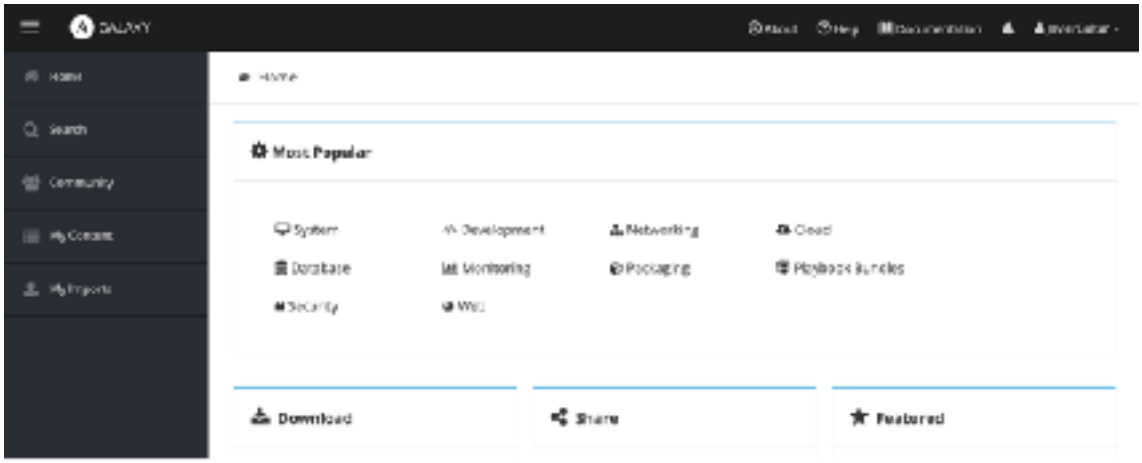
Molecule

Please refer to the following link: https://molecule.readthedocs.io/en/stable/

Ansible Galaxy

Ansible Galaxy refers to the Ansible website where users can share roles, and to a command line tool for installing, creating and managing roles.

https://galaxy.ansible.com/home



The ansible-galaxy command comes bundled with Ansible, and you can use it to install roles from Galaxy or directly from a git based SCM. You can also use it to create a new role, remove roles, or perform tasks on the Galaxy website.

The command line tool by default communicates with the Galaxy website API using the server address https://galaxy.ansible.com. Since the Galaxy project is an open source project, you may be running your own internal Galaxy server and wish to override the default server address. You can do this using the –server option or by setting the Galaxy server value in your ansible.cfg file. For information on setting the value in ansible.cfg visit Galaxy Settings.

Search

Search the Galaxy database by tags, platforms, author and multiple keywords. For example:

$ ansible-galaxy search elasticsearch

List installed roles

Use list to show the name and version of each role installed in the *roles_path*.

$ ansible-galaxy list

- chouseknecht.role-install_mongod, master
- chouseknecht.test-role-1, v1.0.2
- chrismeyersfsu.role-iptables, master
- chrismeyersfsu.role-required_vars, master

Remove installed roles

Use remove to delete a role from *roles_path*:

Import a role

The import command requires that you first authenticate using the login command. Once authenticated you can import any GitHub repository that you own or have been granted access.

Use the following to import to role:

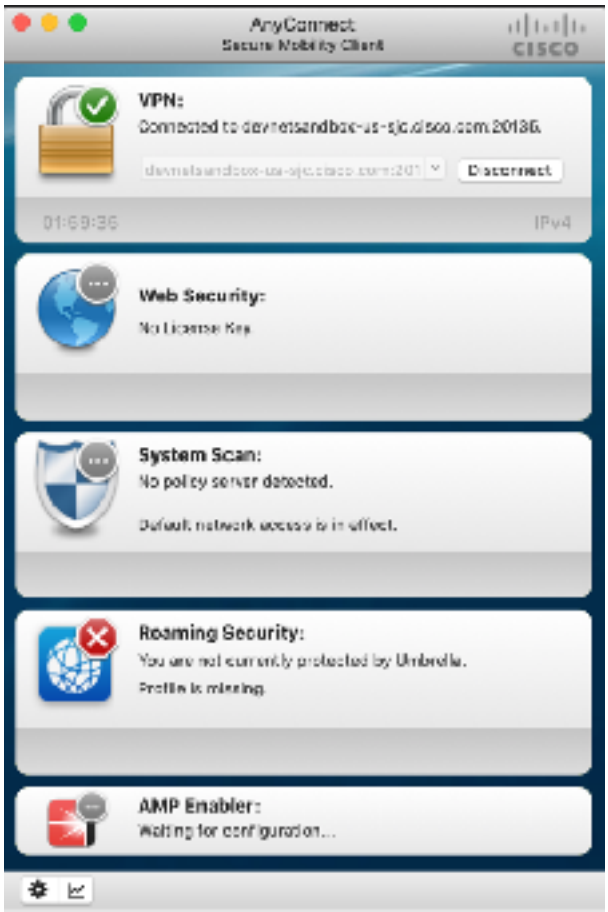
$ ansible-galaxy import github_user github_repo

UCS Manager Management

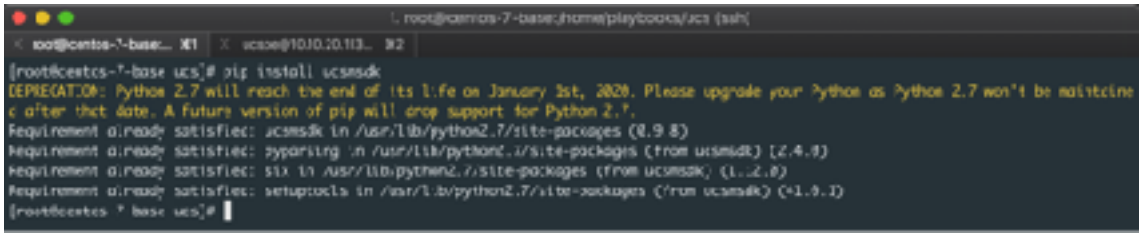
Some of the examples below are executed upon Cisco Devnet sandboxes: https://developer.cisco.com/site/sandbox/

Once your lab is reserved, the VPN credentials are sent by email as shown below:

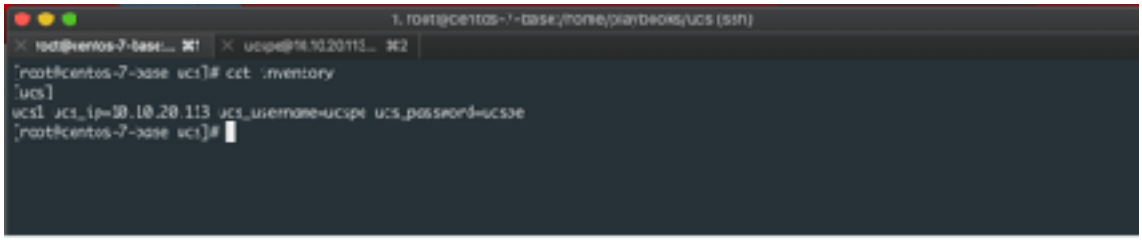
You have to set up the VPN connection:



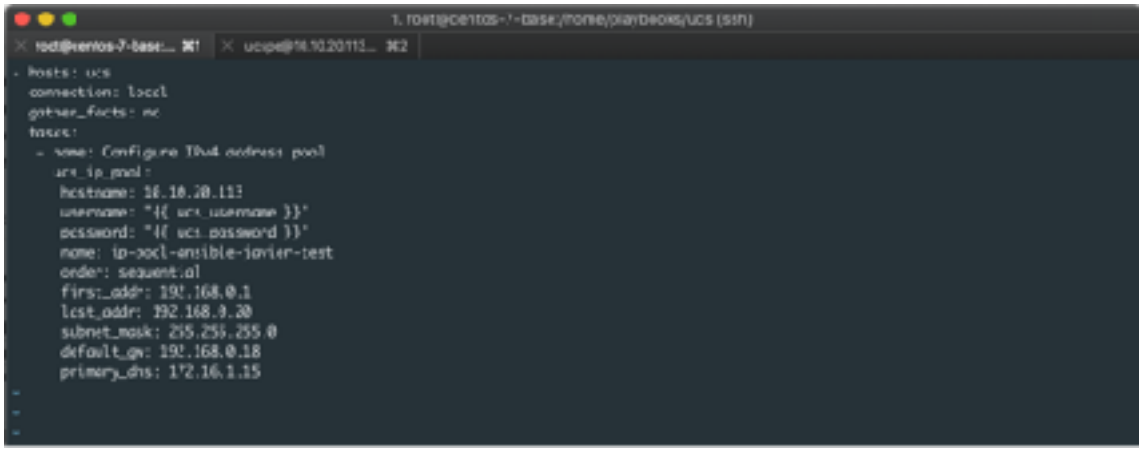
In order to interwork with UCS Manager, the "ucsmsdk" must be installed:



For testing purposes, I have created the inventory file below:

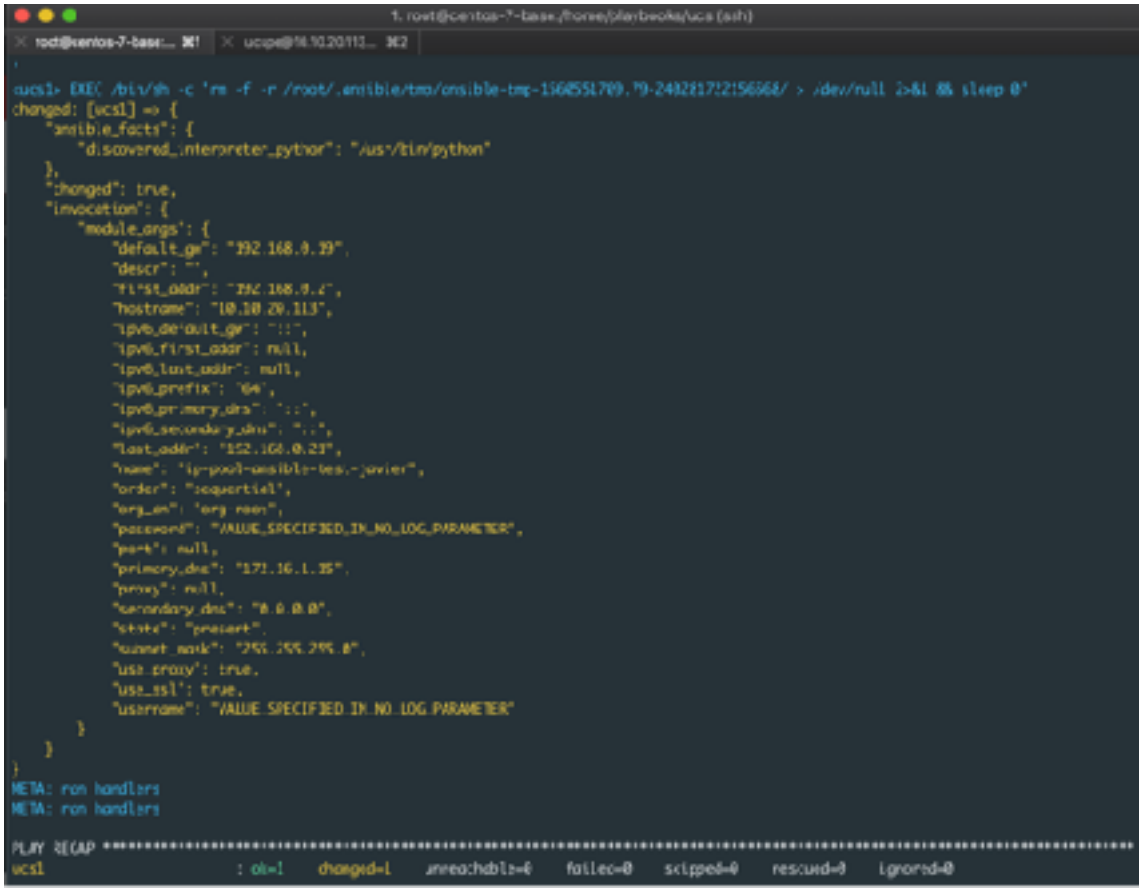


The following playbook configures an address pool in UCS.

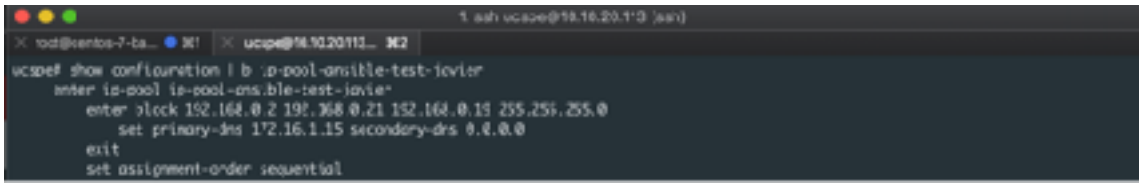


Running the playbook:

 # ansible-playbook create-pool-yaml -i inventory -vvv

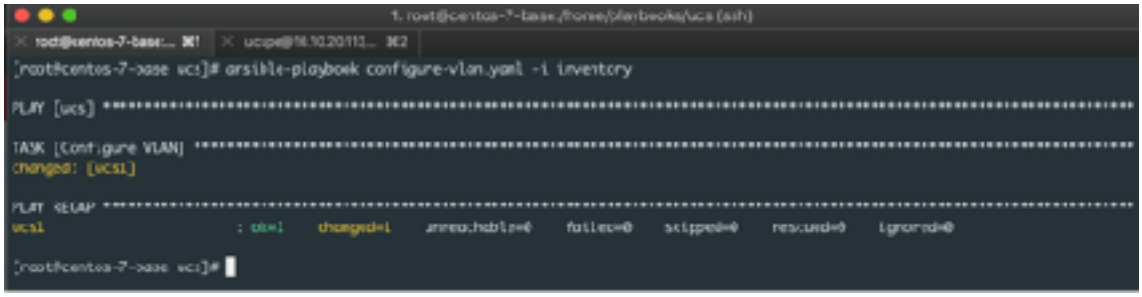


You can establish an SSH connection towards UCS Manager and check that the pool is created:



Similarly, the following playbook creates a VLAN.

Running the playbook:



Validating that the VLAN is created:

Cisco NXOS Management

Please refer to: https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html?highlight=nxos

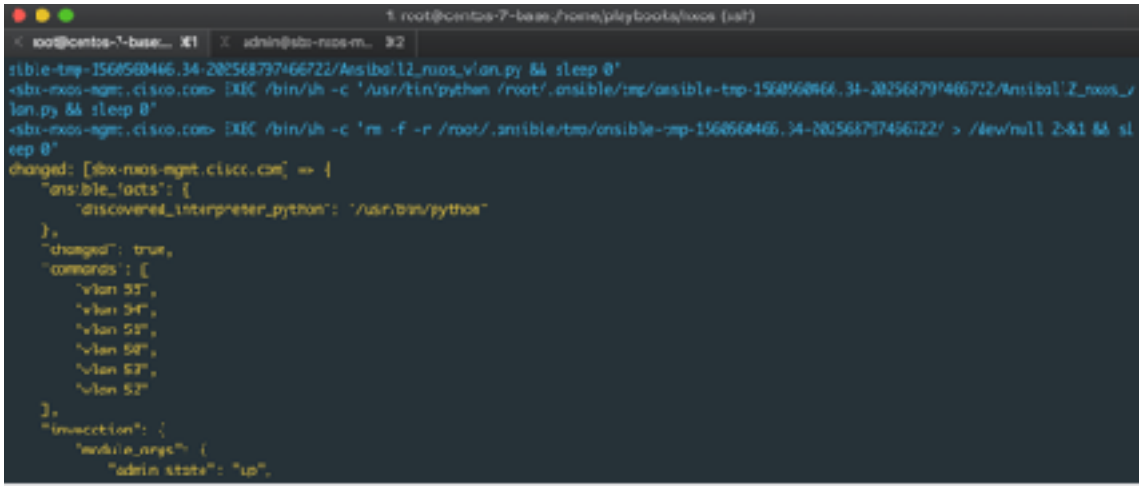
Playbooks

Create VLANs

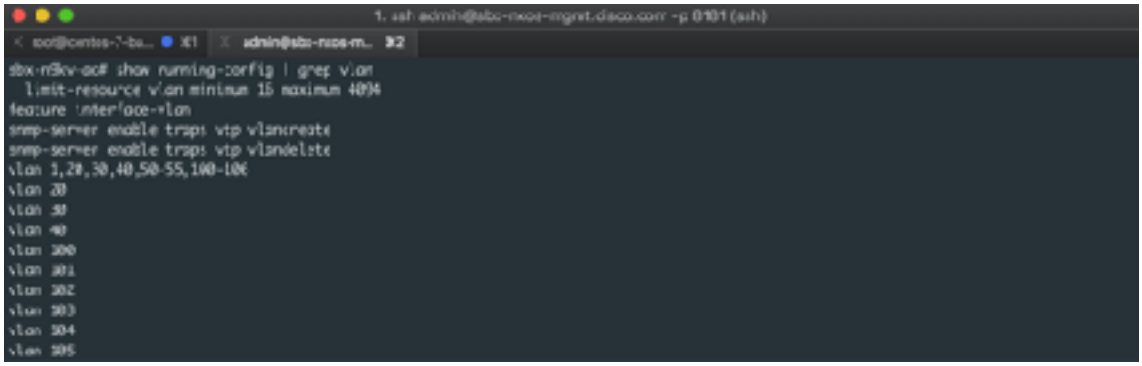
This is the initial NXOS device configuration:

The playbook below creates a range of VLANs:

Running the playbook:

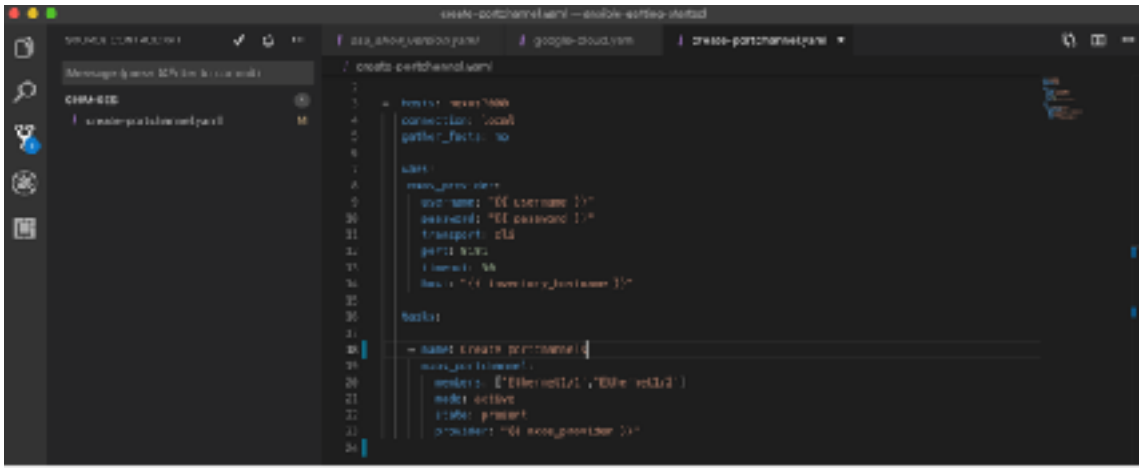


This is the list of VLANs once the playbook is executed:



Configure Portchannels

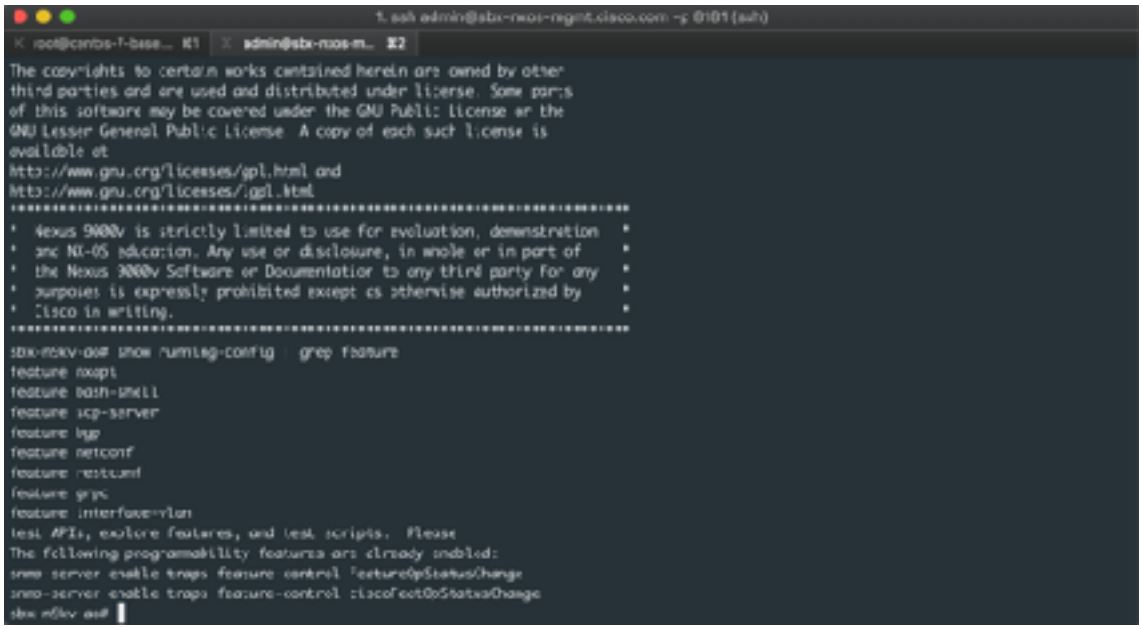
```
tasks:
- name: Create portchannel 200
  nxos_portchannel:
   group: 200
   members: ['Ethernet1/1','Ethernet1/2']
   mode: 'active'
   host: "{{ inventory_hostname }}"
   state: present
```



Feature Enablement

```
tasks:
- name: Ensure DHCP is enabled
  nxos_feature:
   feature: dhcp
   state: enabled
   host: "{{ inventory_hostname }}"
```

This is the list of features enabled:

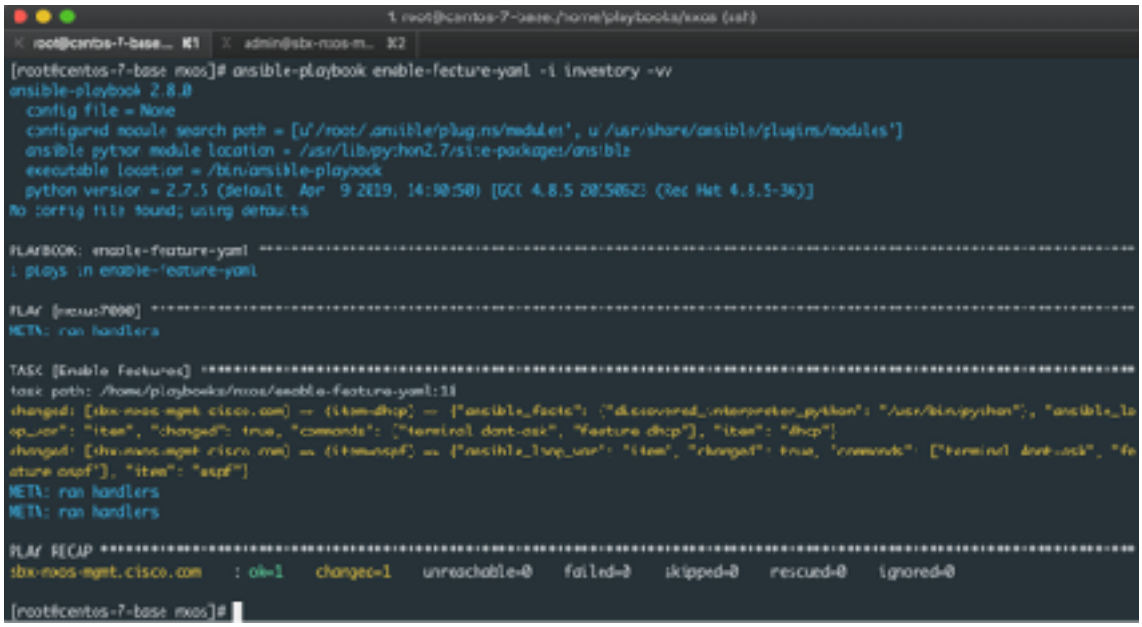


The inventory file includes the Nexus 9K hostname, username and password:

The playbook enables two new features: DHCP and OSPF:

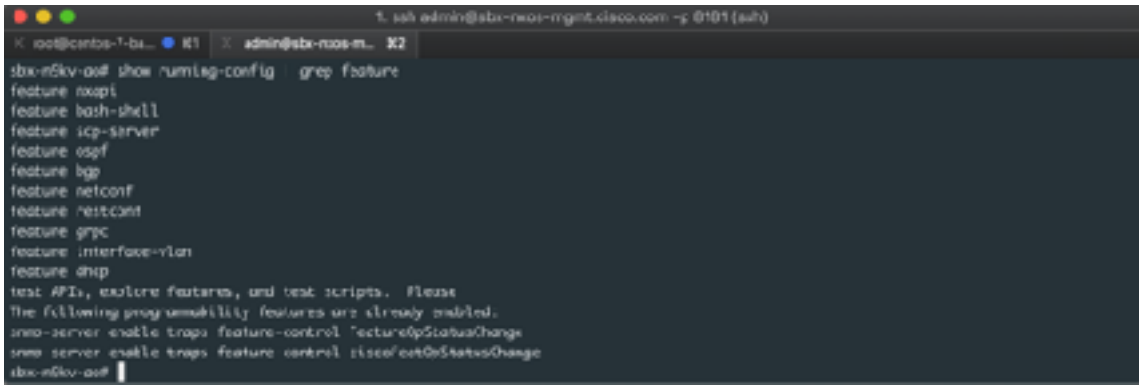
```
[root@centos-7-base nxos]# cat enable-feature-yaml
---

  hosts: nexus7000
  connection: local
  gather_facts: no

  vars:
    nxos_provider:
      username: "{{ username }}"
      password: "{{ password }}"
      transport: cli
      port: 8181
      timeout: 30
      host: "{{ inventory_hostname }}"

  tasks:

    - name: Enable features
      nxos_feature:
        feature: '{{ item }}'
        state: enabled
        provider: '{{ nxos_provider }}'
      with_items:
        - dhcp
        - ospf
[root@centos-7-base nxos]#
```

Running the playbook:



```
[root@centos-7-base nxos]# ansible-playbook enable-feature-yaml -i inventory -vv
ansible-playbook 2.8.0
  config file = None
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /bin/ansible-playbook
  python version = 2.7.5 (default, Apr  9 2019, 14:30:50) [GCC 4.8.5 20150623 (Red Hat 4.8.5-36)]
No config file found; using defaults

PLAYBOOK: enable-feature-yaml **********************************************************************
1 plays in enable-feature-yaml

PLAY [nexus7000] ***********************************************************************************
META: ran handlers

TASK [Enable Features] *****************************************************************************
task path: /home/playbooks/nxos/ansible-feature-yaml:18
changed: [sbx-nxos-mgmt.cisco.com] => (item=dhcp) => {"ansible_facts": {"discovered_interpreter_python": "/usr/bin/python"}, "ansible_lo
op_var": "item", "changed": true, "commands": ["terminal dont-ask", "feature dhcp"], "item": "dhcp"}
changed: [sbx-nxos-mgmt.cisco.com] => (item=ospf) => {"ansible_loop_var": "item", "changed": true, "commands": ["terminal dont-ask", "fe
ature ospf"], "item": "ospf"}
META: ran handlers
META: ran handlers

PLAY RECAP *****************************************************************************************
sbx-nxos-mgmt.cisco.com    : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[root@centos-7-base nxos]#
```

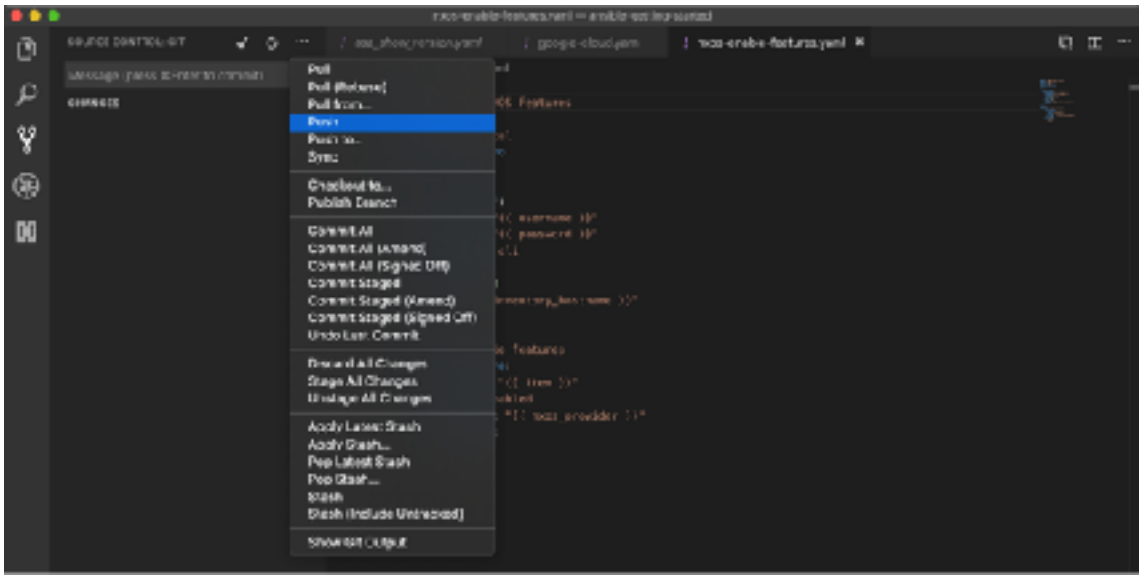
The new features are enabled:

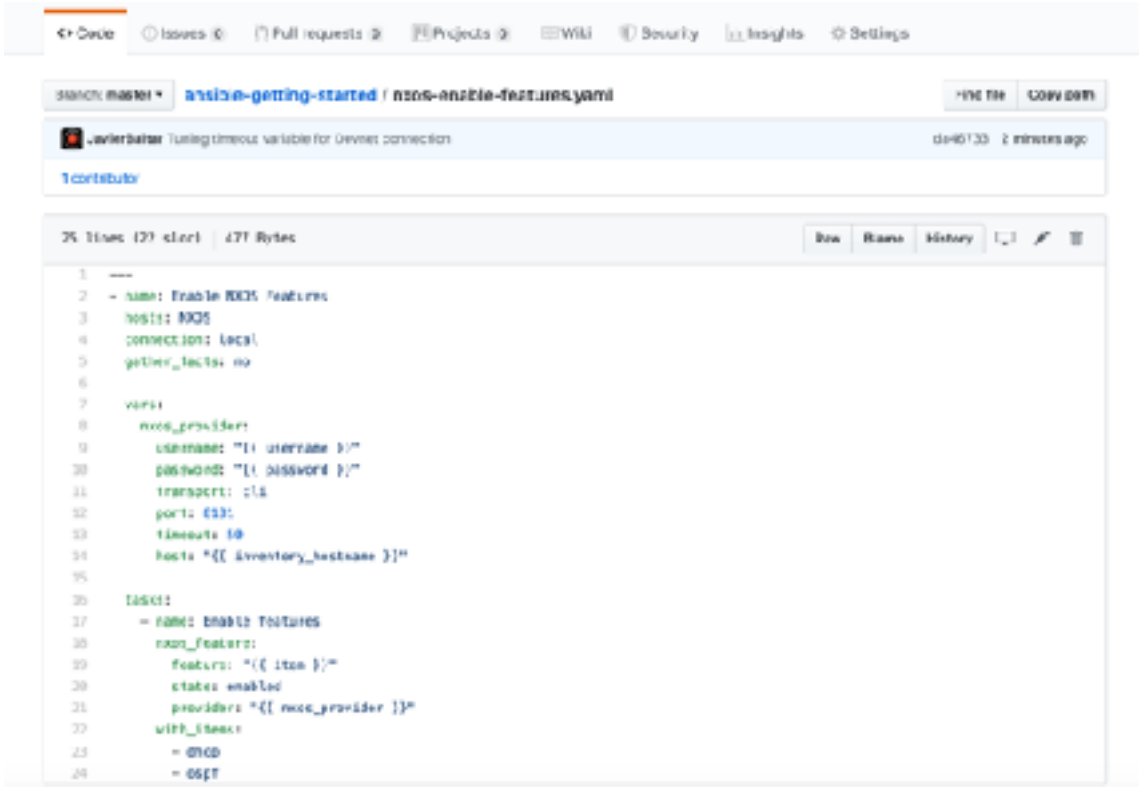


```
sbx-n9kv-ao# show running-config | grep feature
feature nxapi
feature bash-shell
feature scp-server
feature ospf
feature bgp
feature netconf
feature restconf
feature grpc
feature interface-vlan
feature dhcp
test APIs, explore features, and test scripts.  Please
The following programmability features are already enabled.
snmp-server enable traps feature-control ExtSrcOpStatusChange
snmp server enable traps feature control CiscoEntOpStatusChange
sbx-n9kv-ao#
```

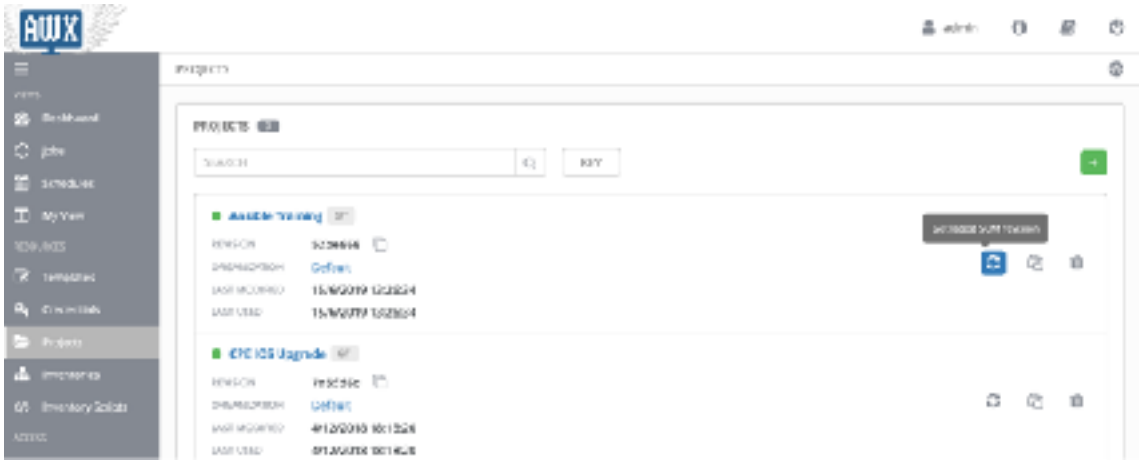
It is time to get into AWX and play around with all these possibilities.

You have to push your code to the Git repository:

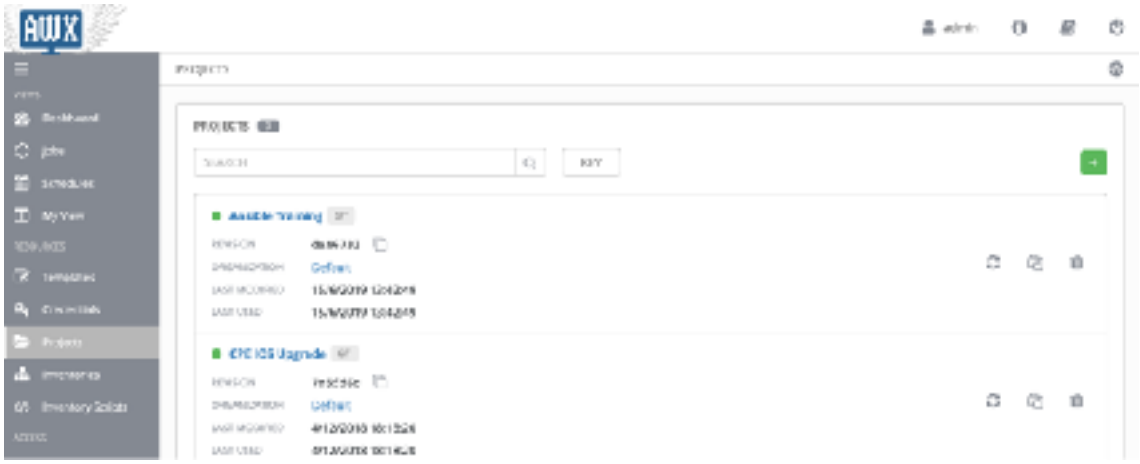




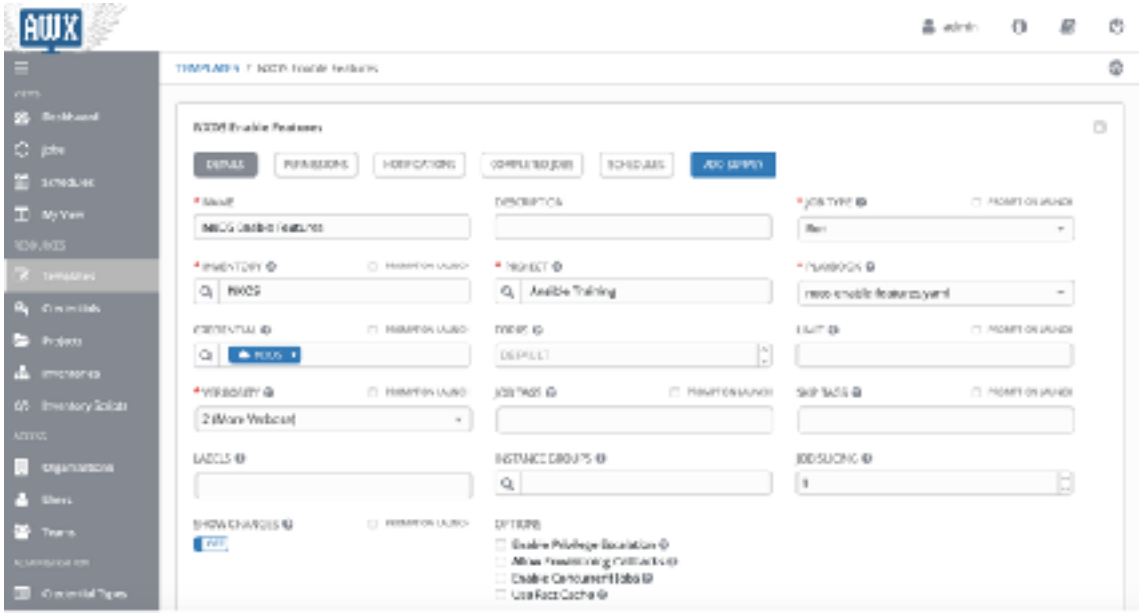
Ensure that AWX project is updated by clicking on Sync:



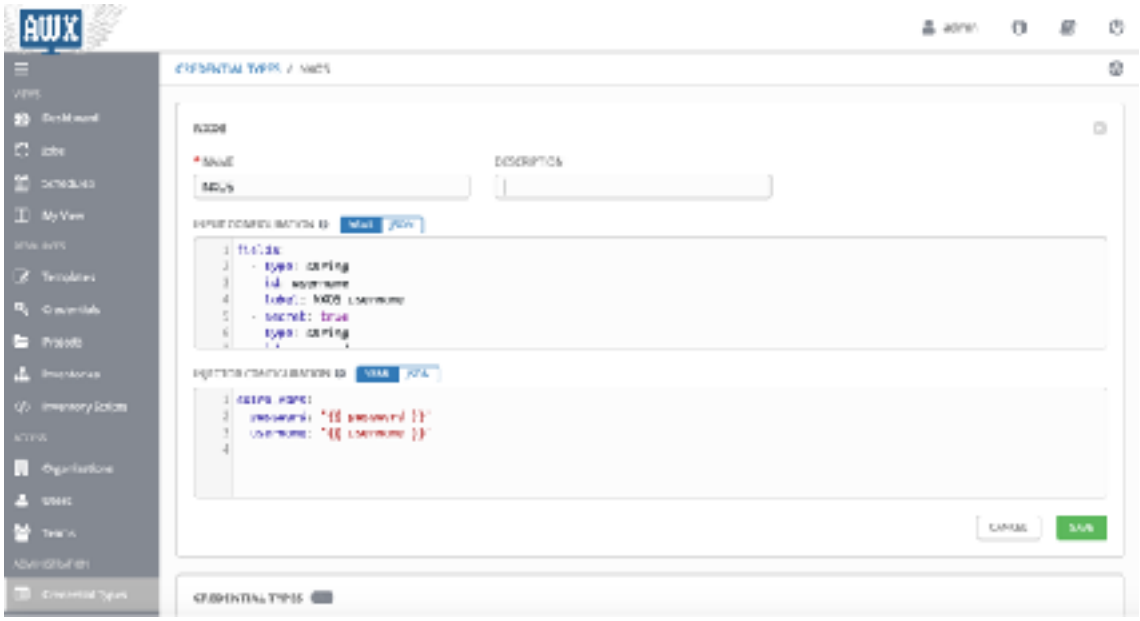
The revision matches with Github:



Before creating the template, you have to provision the credentials.

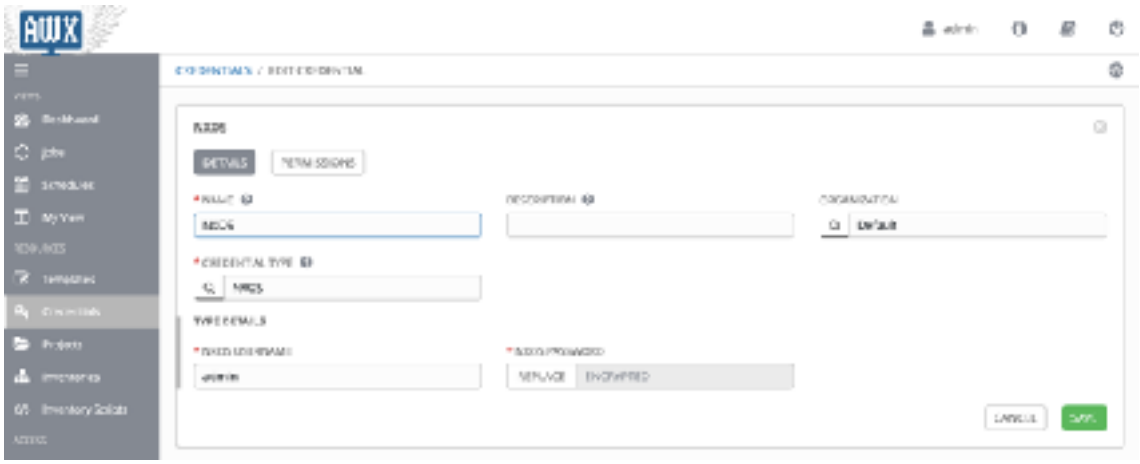


Click on Credentials Types

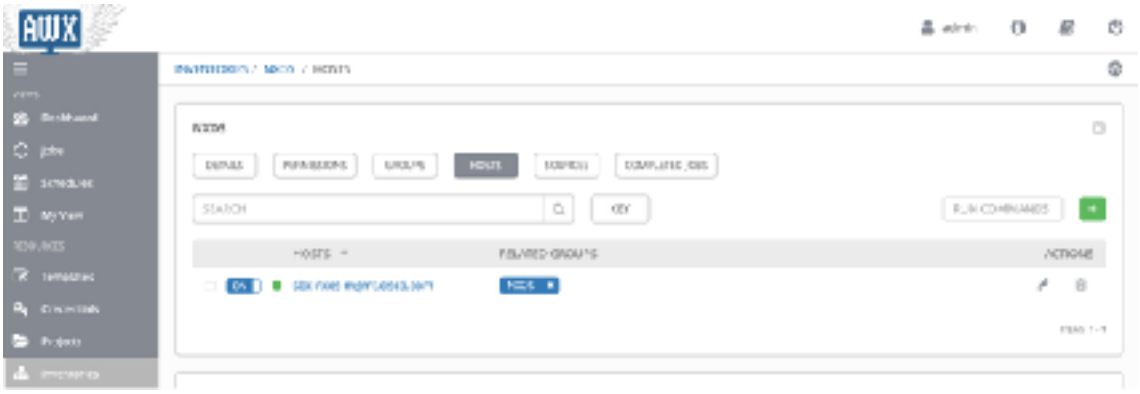


Input Configuration:

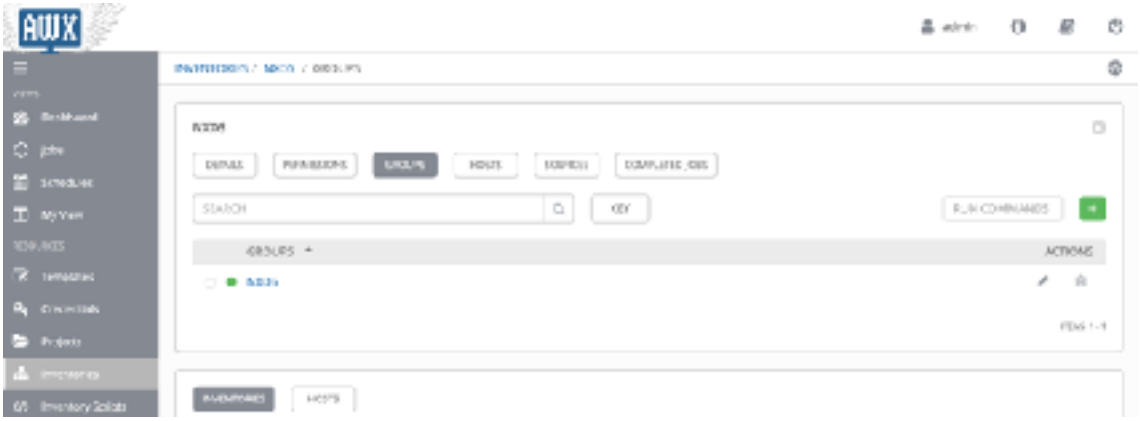
fields:
- type: string
id: username
label: NXOS username
- secret: true
type: string
id: password
label: NXOS password
required:
- username
- password



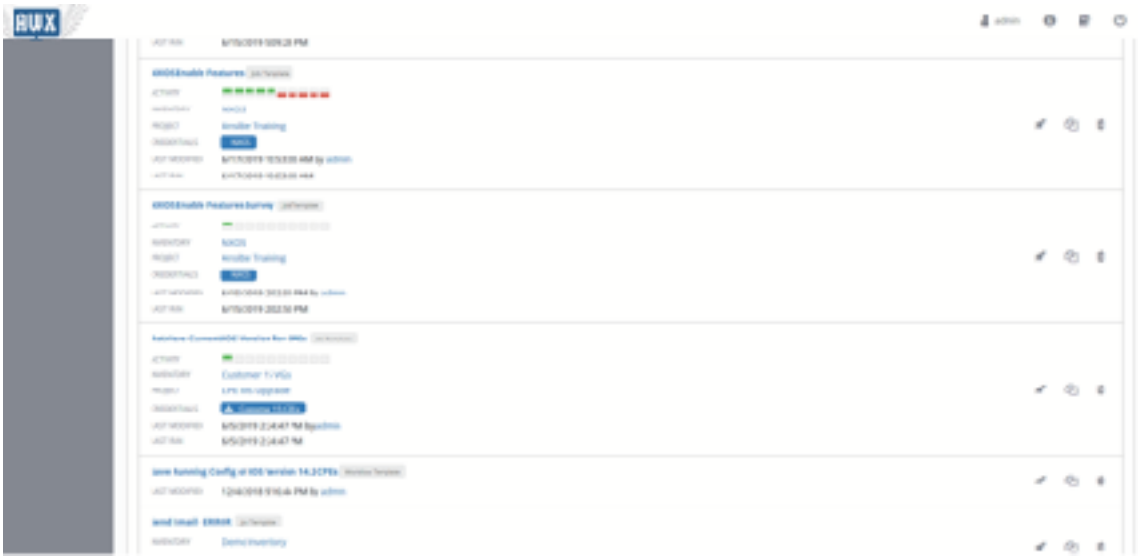
Click on Inventory and add the NXOS host:



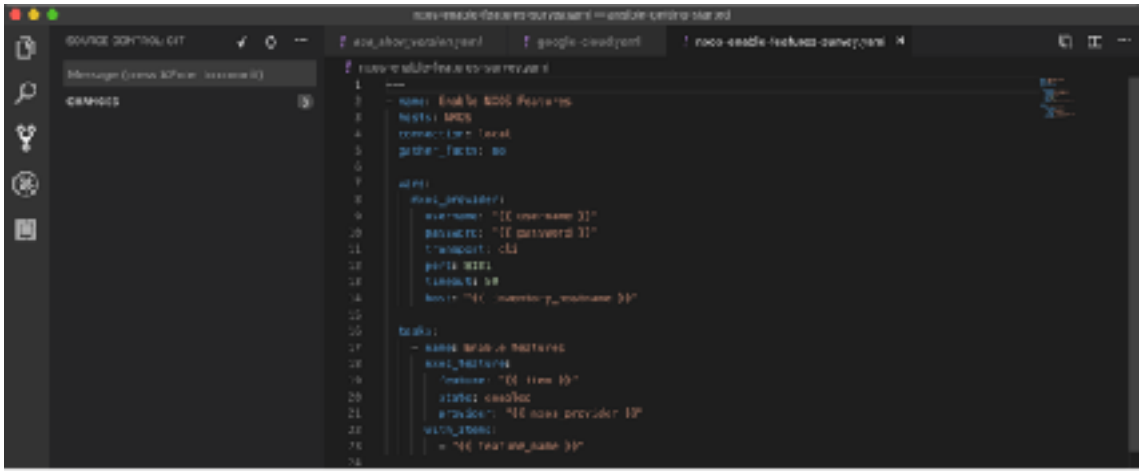
Create the corresponding group:



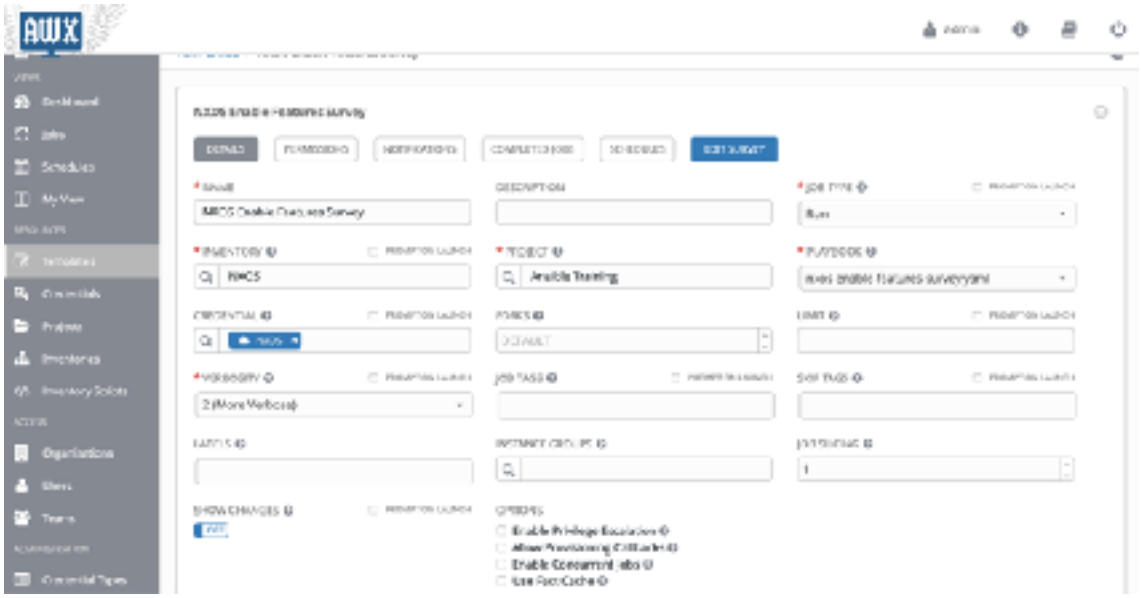
Let´s run the plyabook:



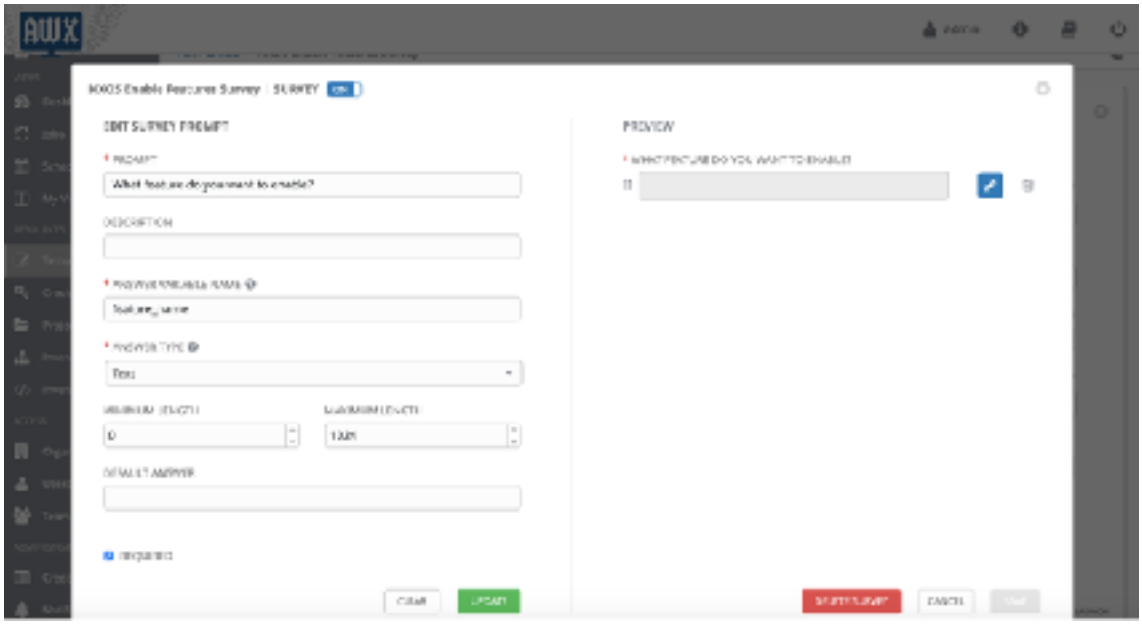
You can also add a survey, which passes a variable value to the playbook execution.
I am adding a new variable called "feature_name".



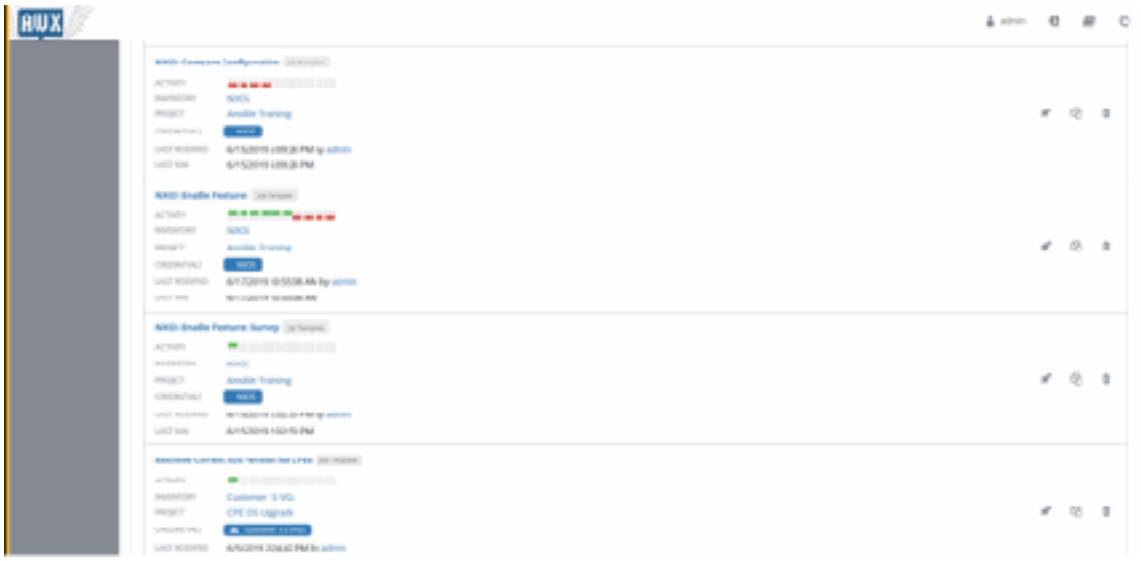
Click on your template and Edit Survey:



Add a prompt, variable name and answer type:



In this case, you have to specify the feature before running the playbook as shown below:



Cisco ASA Management

Please refer to: https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html?highlight=asa

Playbooks

Create Network object

```
---
- hosts: asa
  gather_facts: no
  connection: network_cli

vars:
  object_hosts:
   - 10.45.16.79
   - 10.45.16.72

  object_group: TEST

  tasks:
   - name: "CREATE HOST AND GROUPS ON {{ ansible_ssh_host }}"
      asa_config:
       lines:
        - network-object host {{ item }}
        parents: object-group network {{ object_group }}
       with_items: "{{ object_hosts }}"
```

Cisco IOS Management

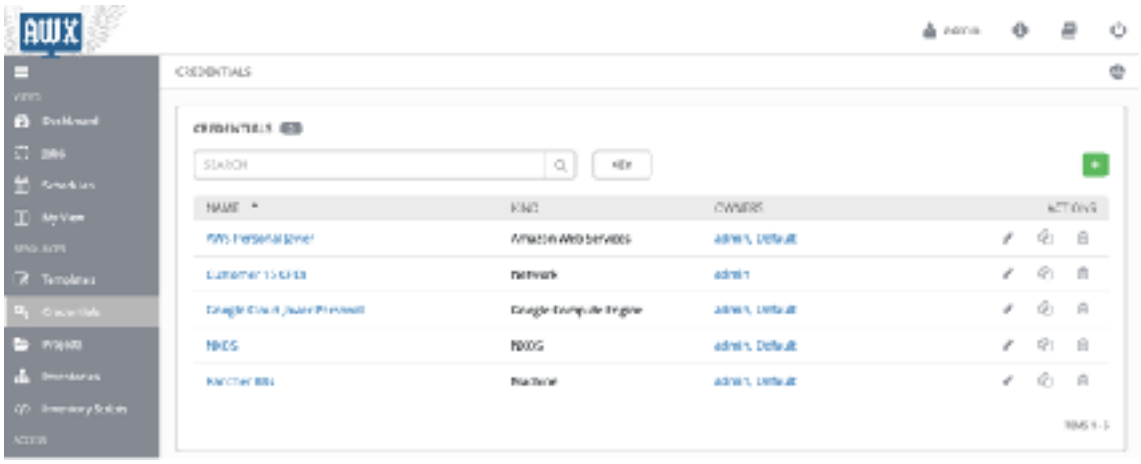
Please refer to: https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html?highlight=ios_

Deploying to Amazon Web Services

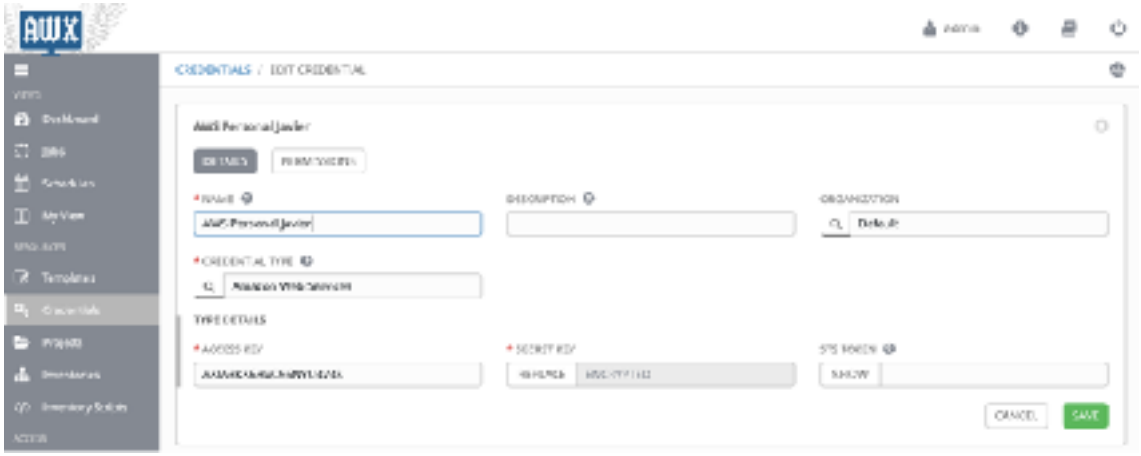
Inventory

You can configure a dynamic inventory syncing with your AWS resources.

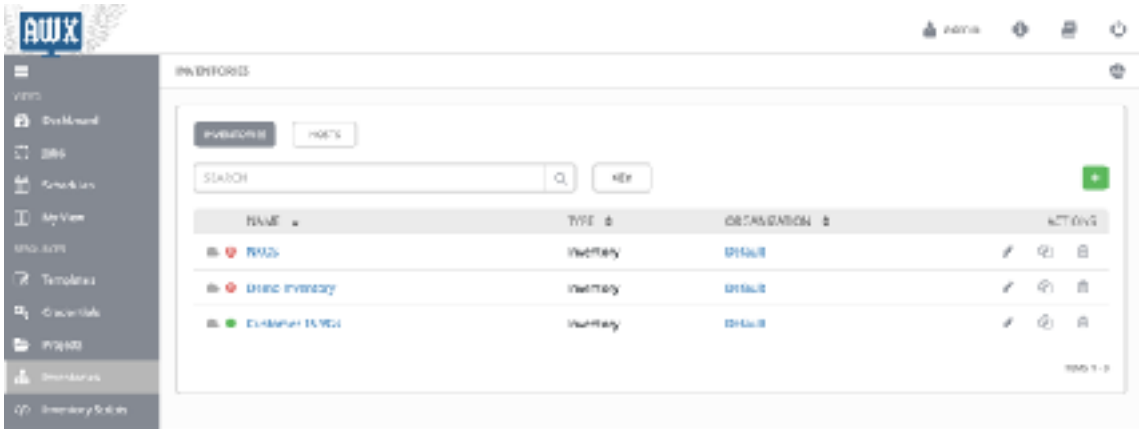
First, configure your AWS account credentials:



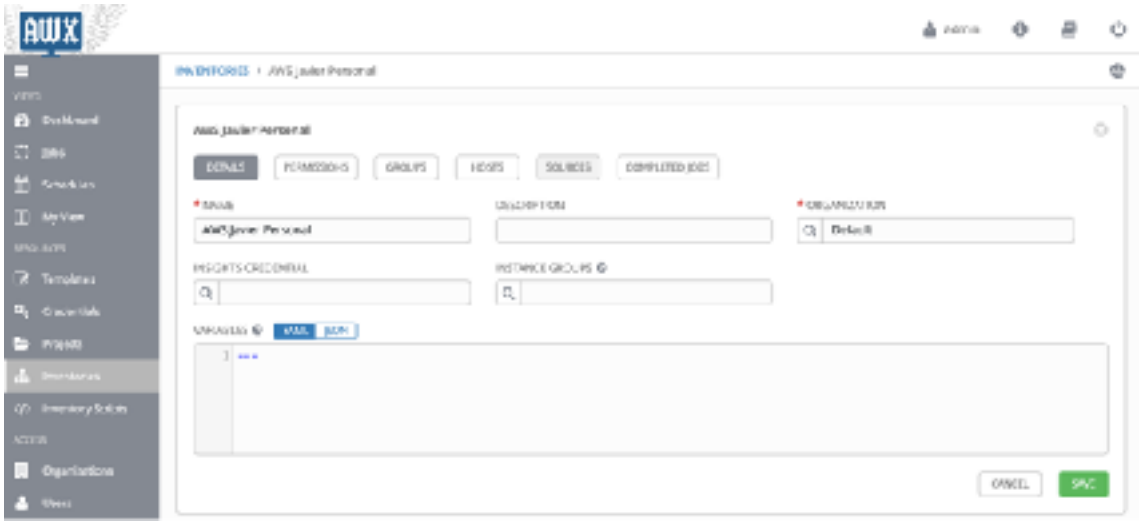
Add your access and secrets keys.



Click on Inventories:



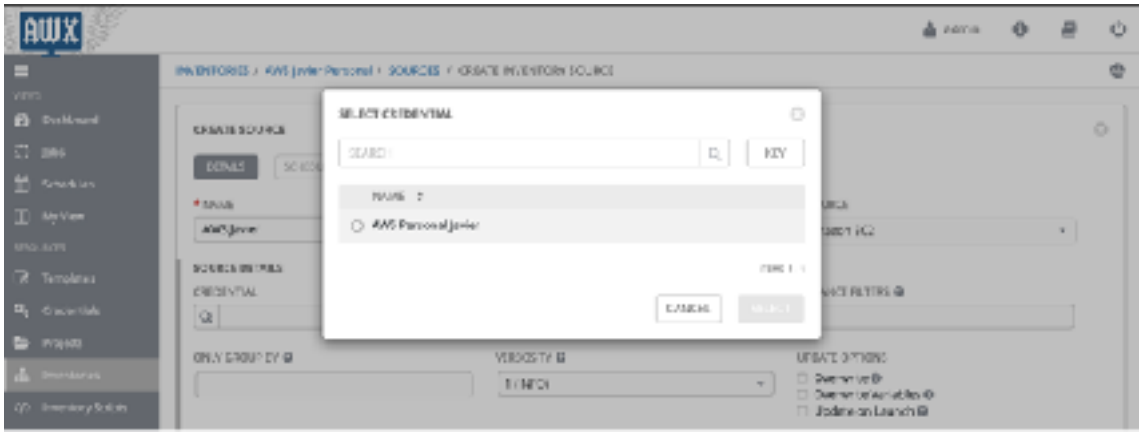
Click on Sources.



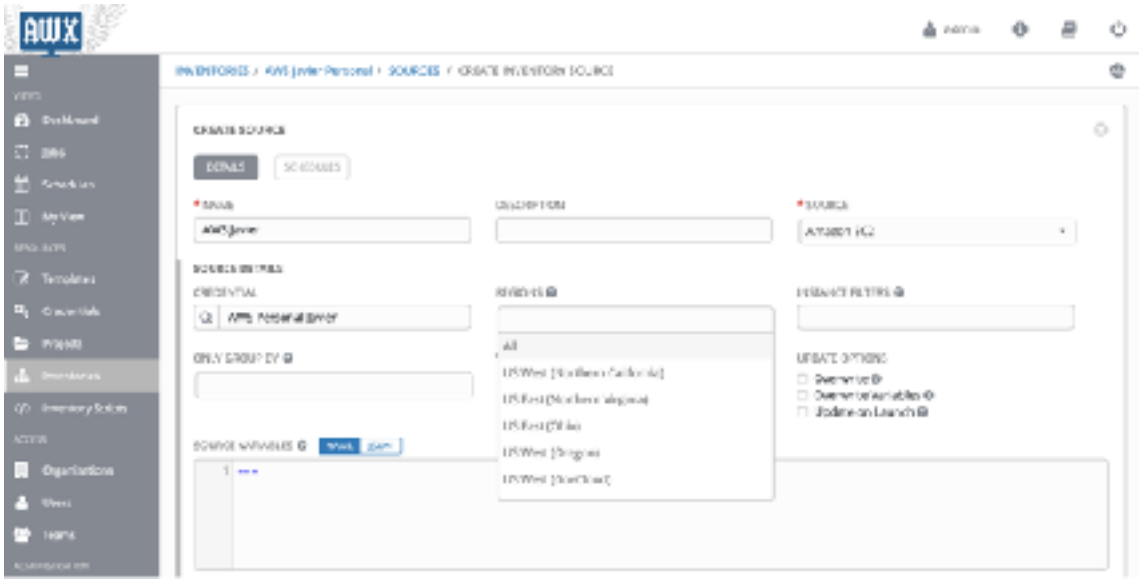
Choose the Amazon EC2 source.

NOTE: VMware vCenter is also supported.

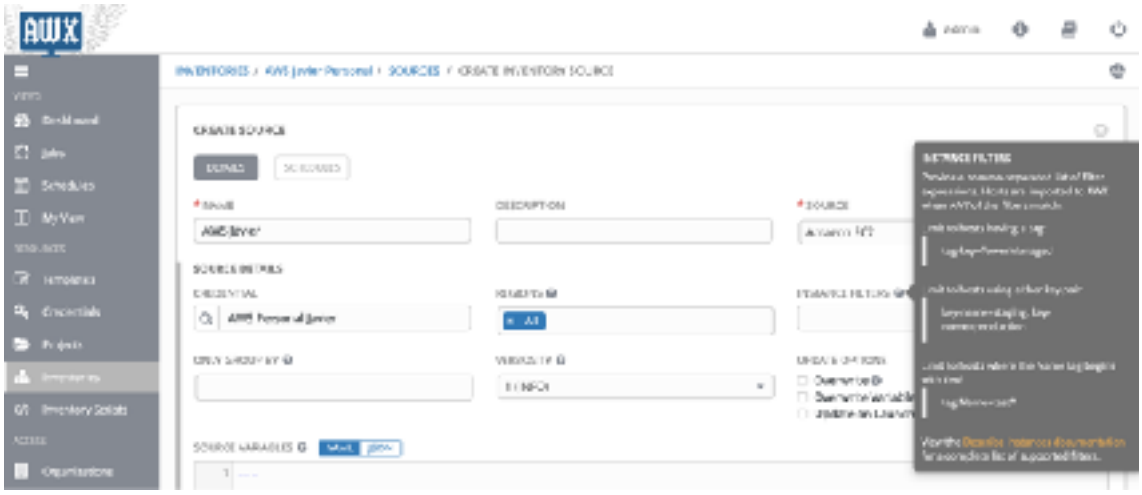
Choose your AWS credentials:



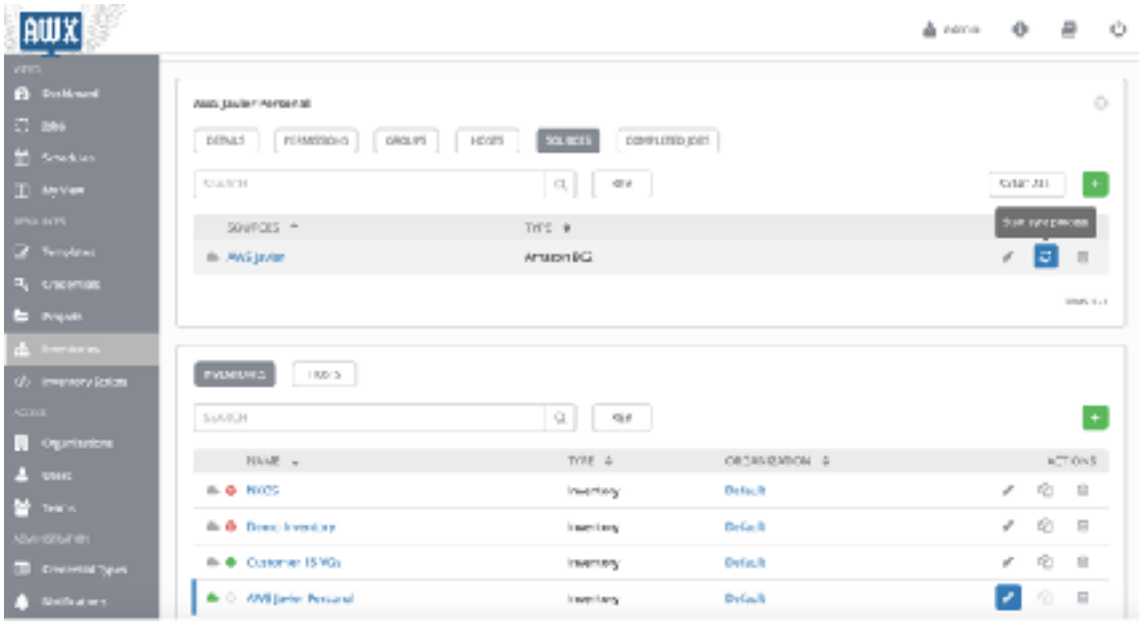
Select your AWS regions:



You can also apply filters:



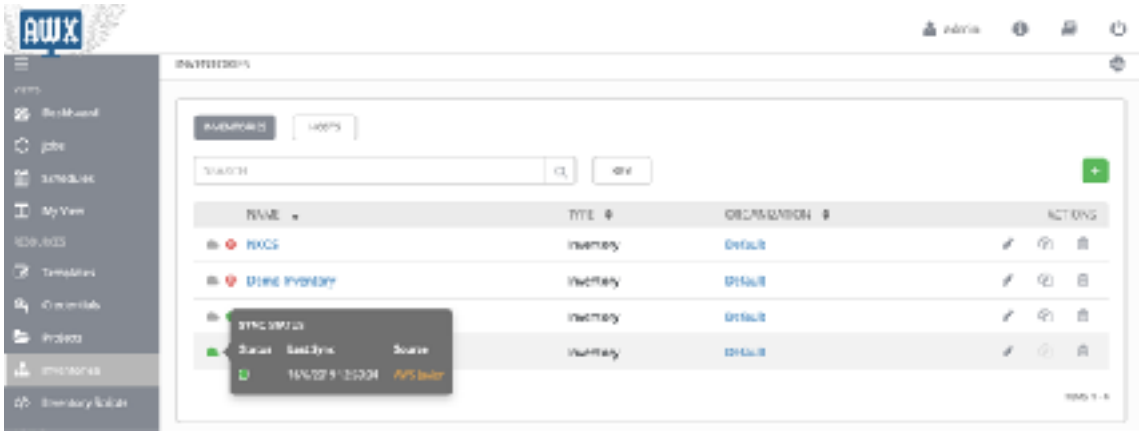
Start the sync process:

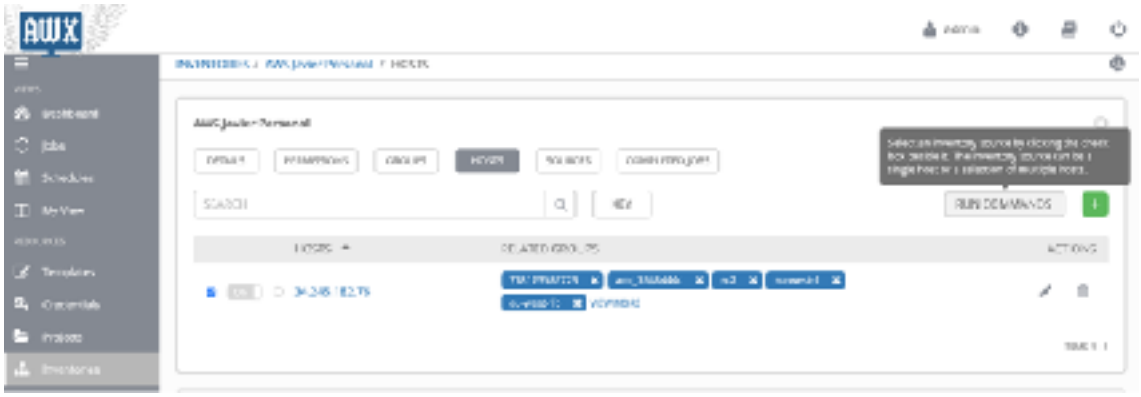


Your inventory is synced:

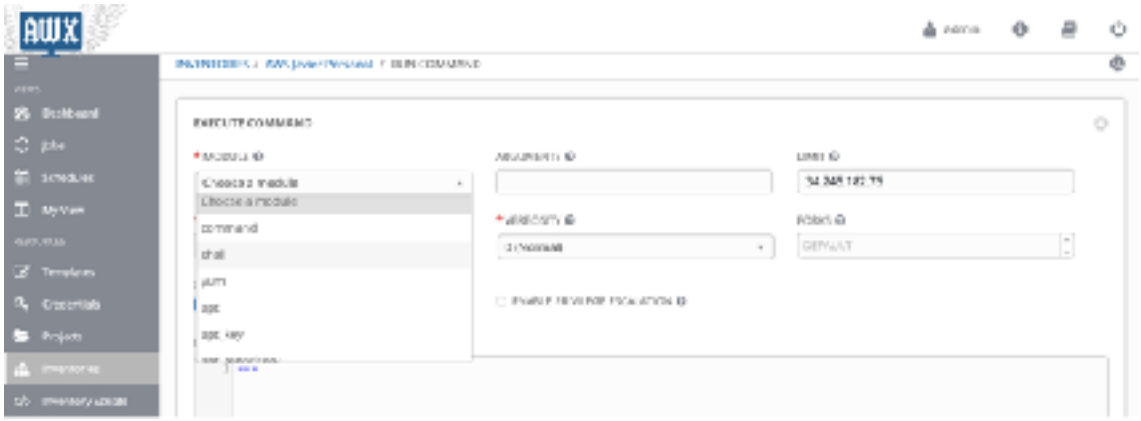
Click on hosts and check that the hosts are imported:

You can check the latest sync details.





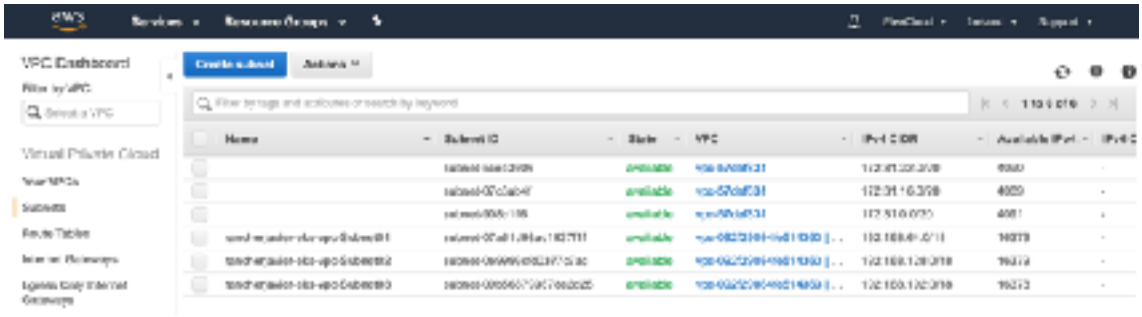
In addition, you can run ad-hoc commands as shown below:



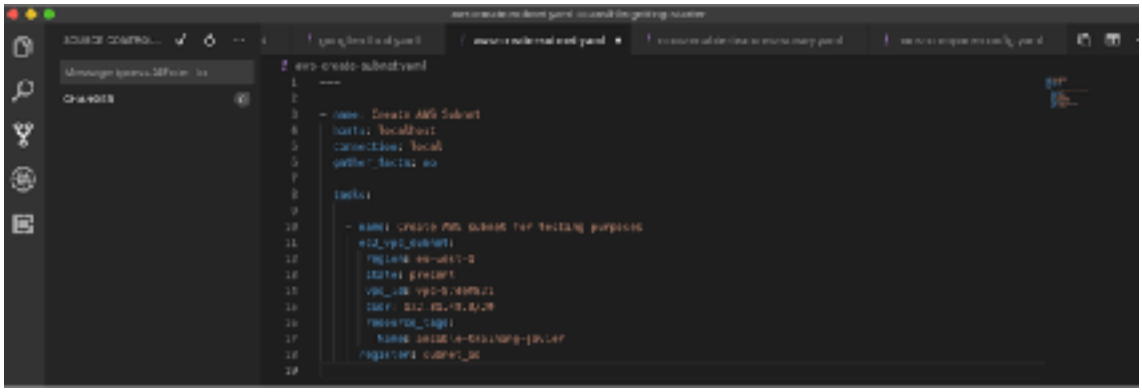
Playbooks

In the following example, an AWS subnet is created.

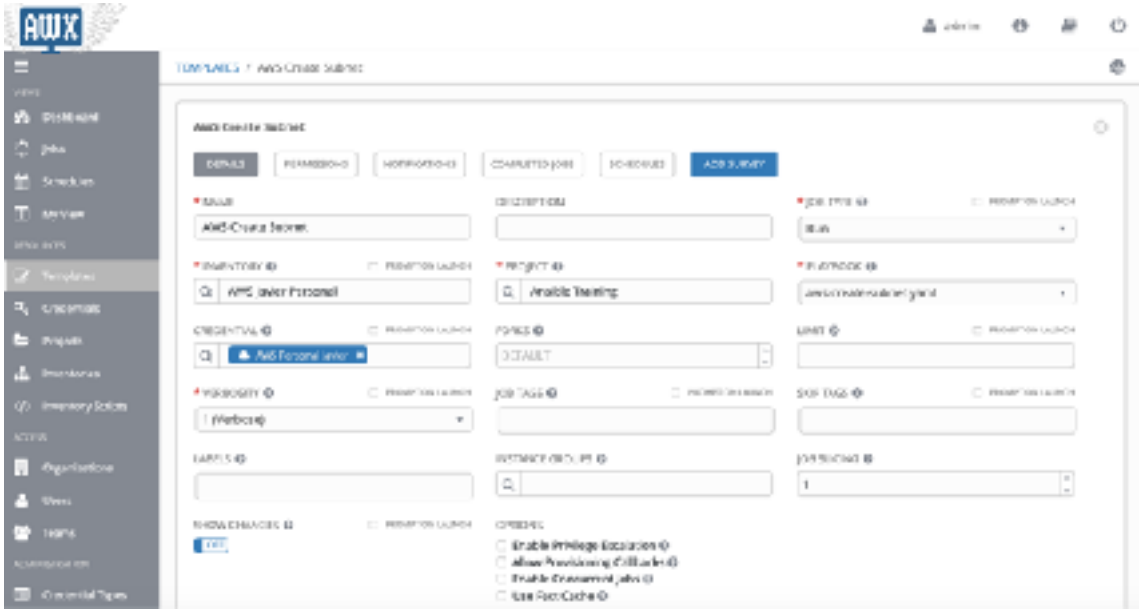
This is the list of subnets before running my playbook:



For testing purposes, I am just including the values hardcoded.

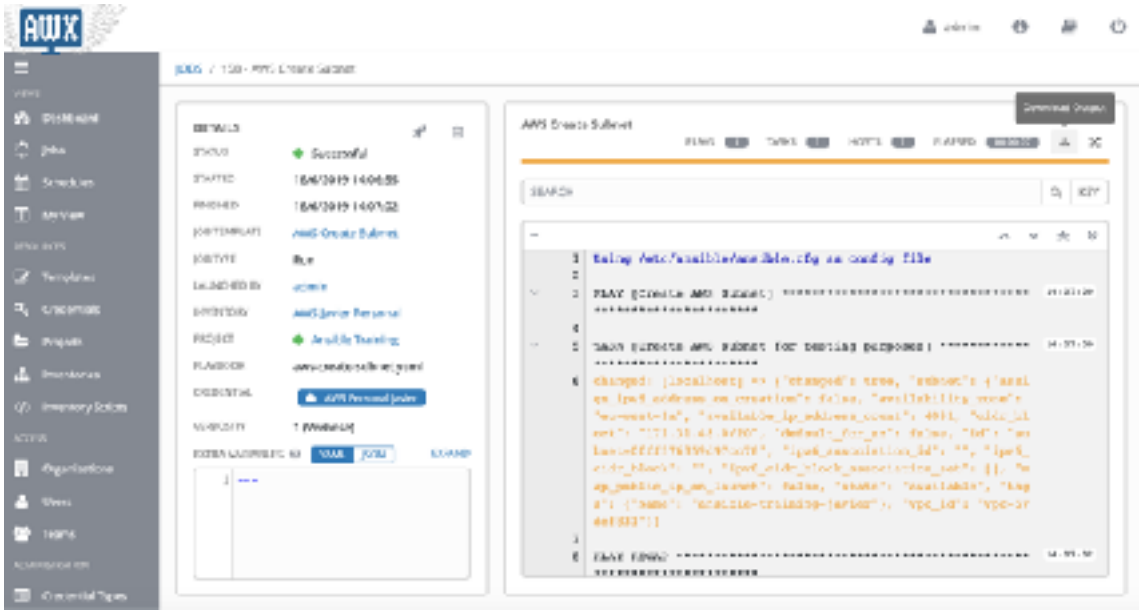


Lets create the template in AWX:

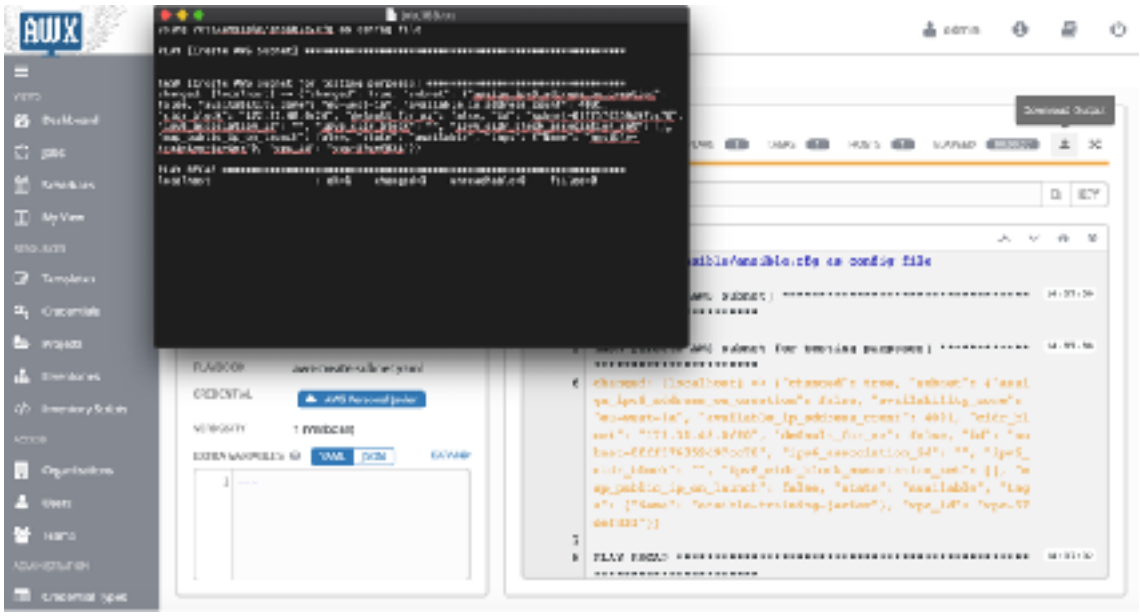


Running the playbook:

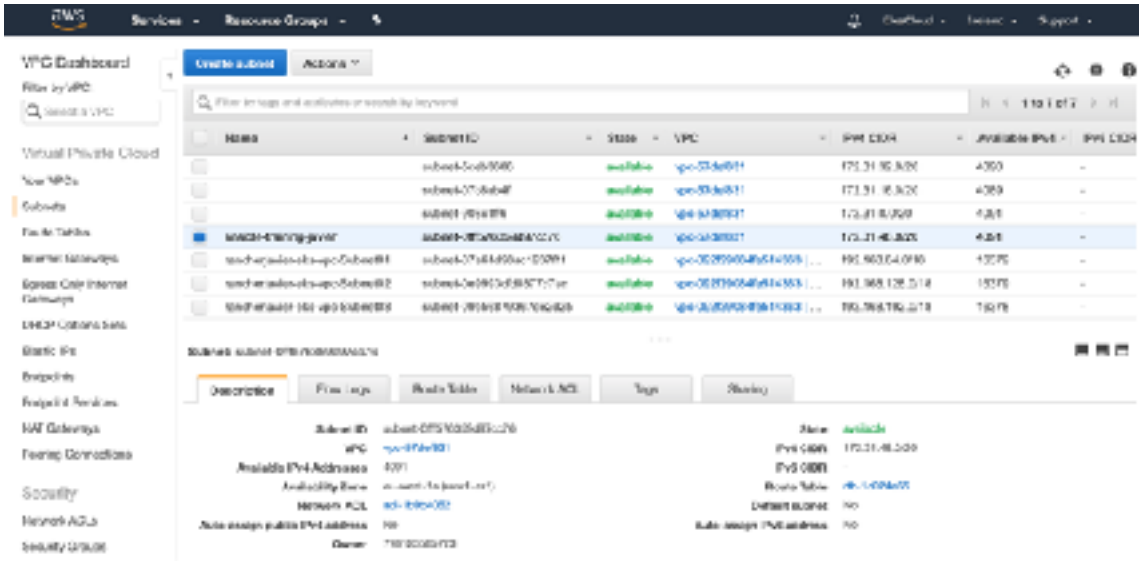
NOTE: The output can be downloaded and shared:



A text file with the output is generated:



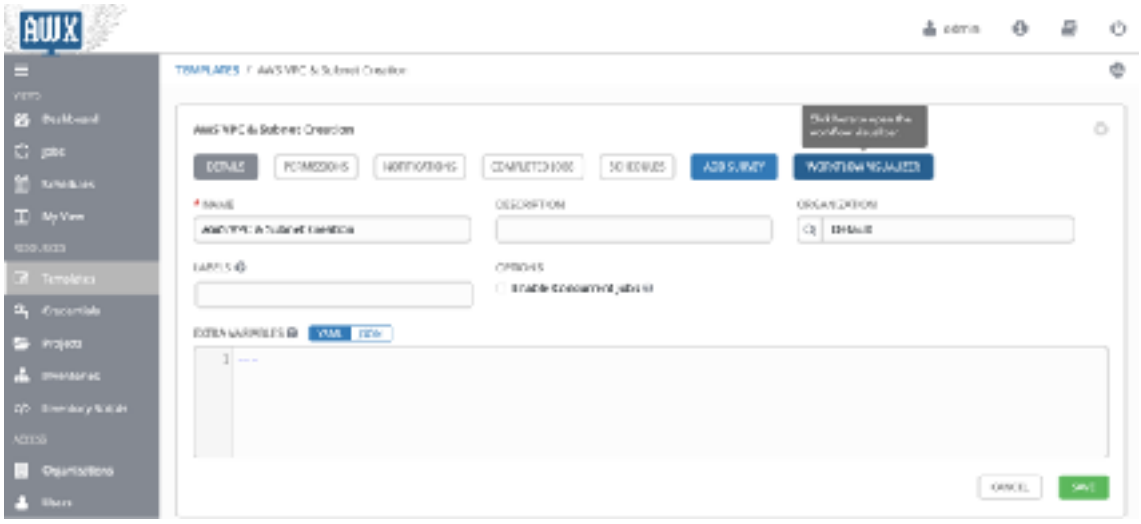
The new AWS subnet is created:



Let's take a glance of workflow templates and discuss how to inherit variables.

Click on Templates and add a new Workflow Template:

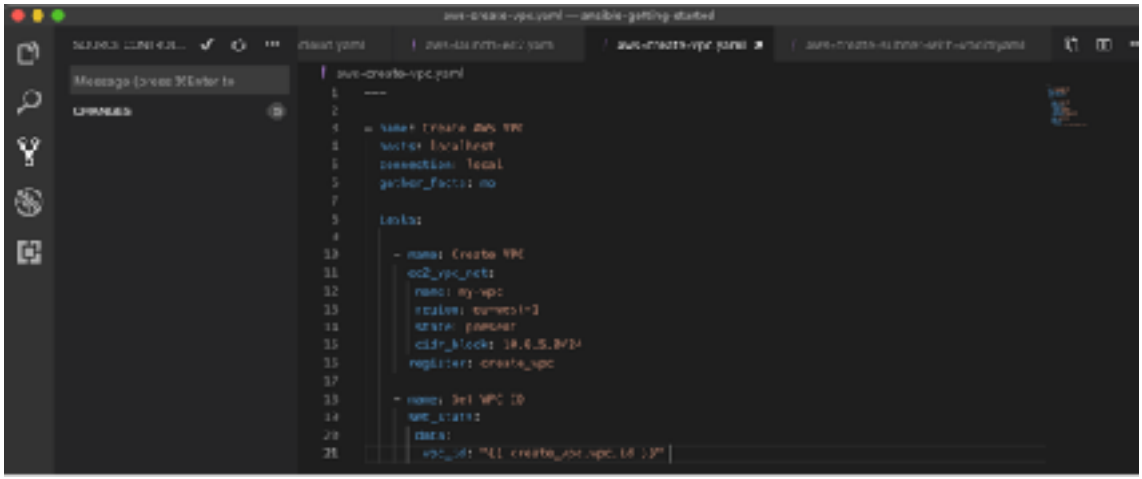
Once the name is saved, click on Workflow Visualizer



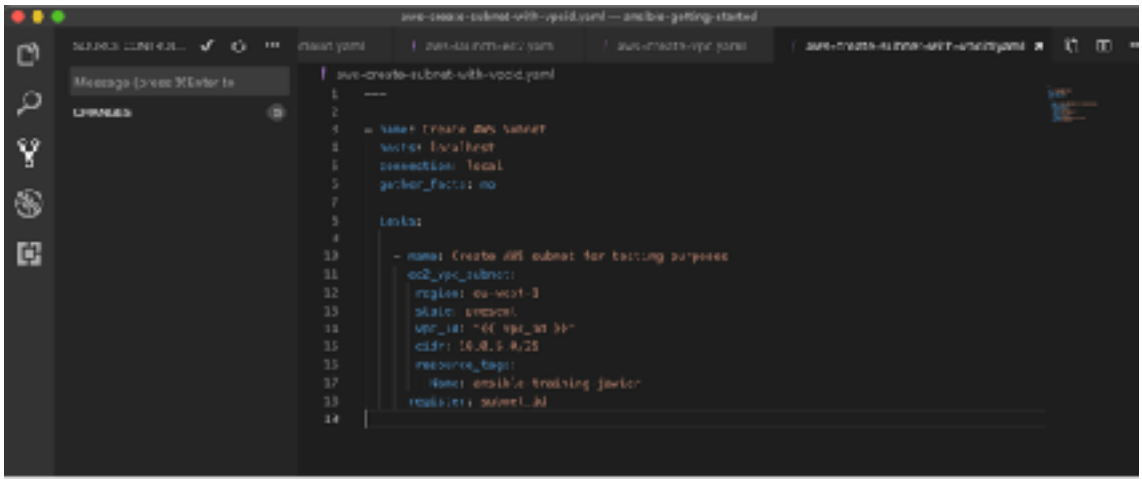
I have created the workflow below:

In order to add templates, click over a template and select a job, project sync or inventory sync.

In order to pass the "vpc_id" from the AWS Create VPC playbook to "AWS Create Subnet with VPC ID", you can use the "set_stats" module:

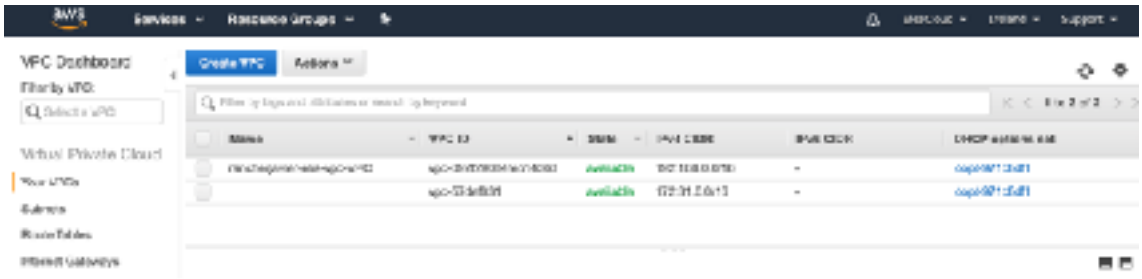


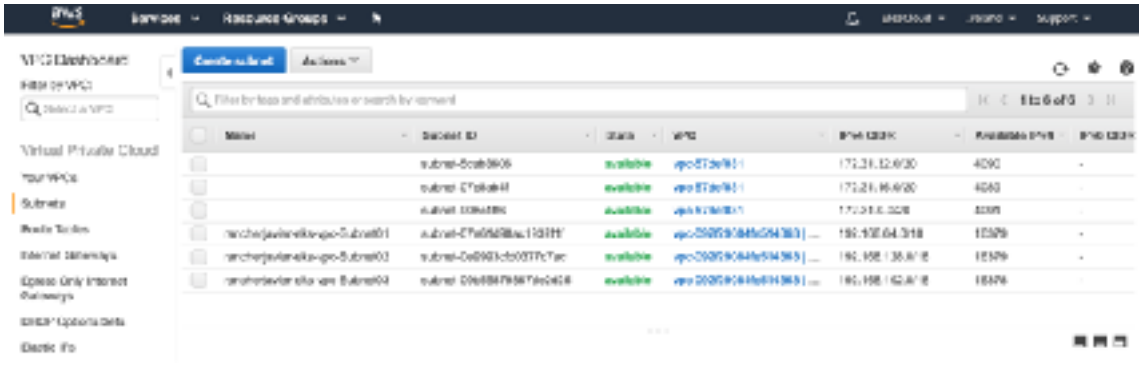
The create subnet playbook inherits that value.



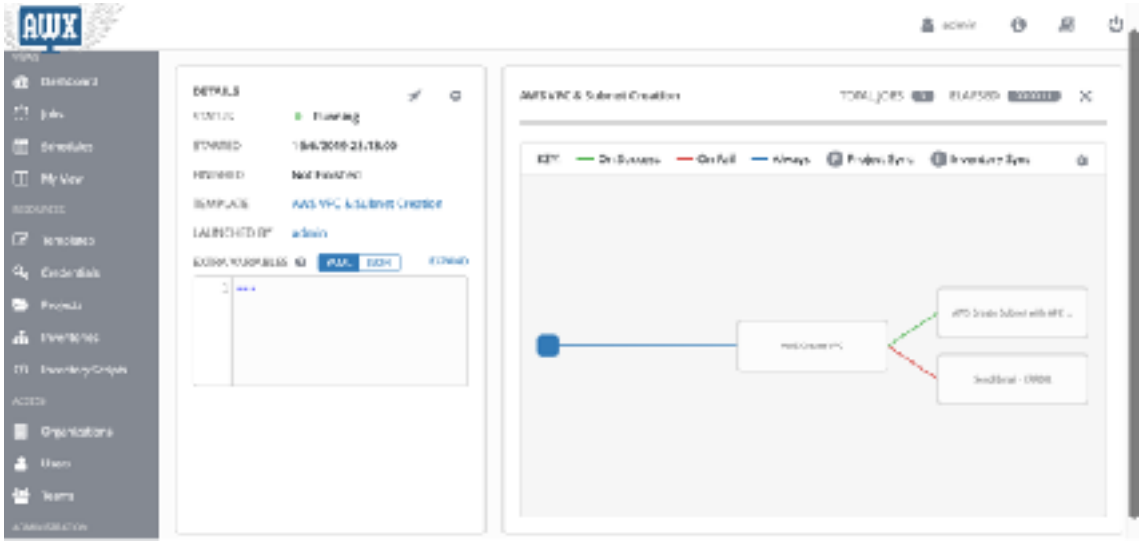
Let´s run the workflow.

This is the list of VPCs and subnets created.

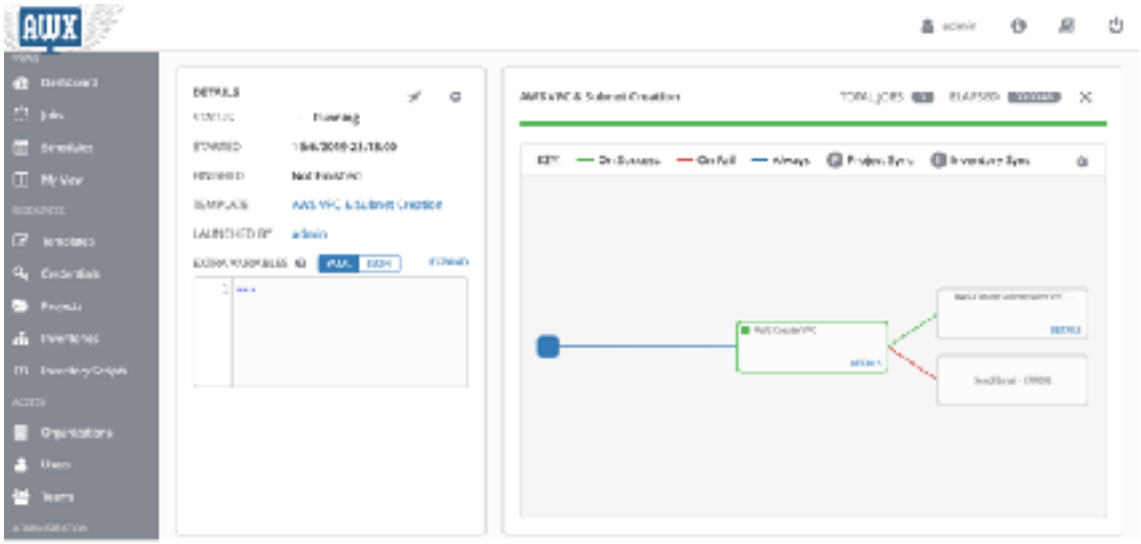




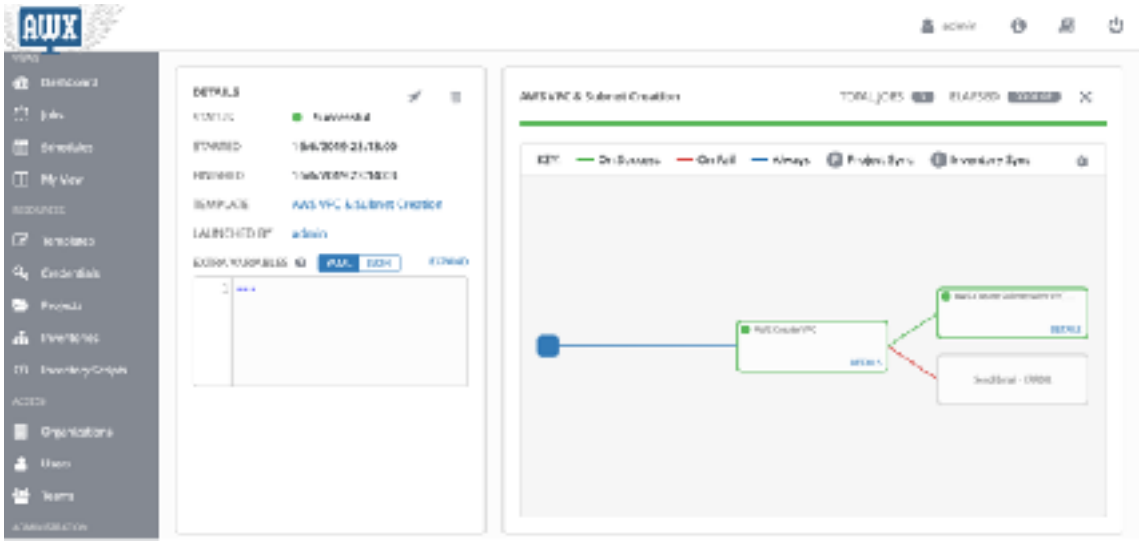
Running the workflow:



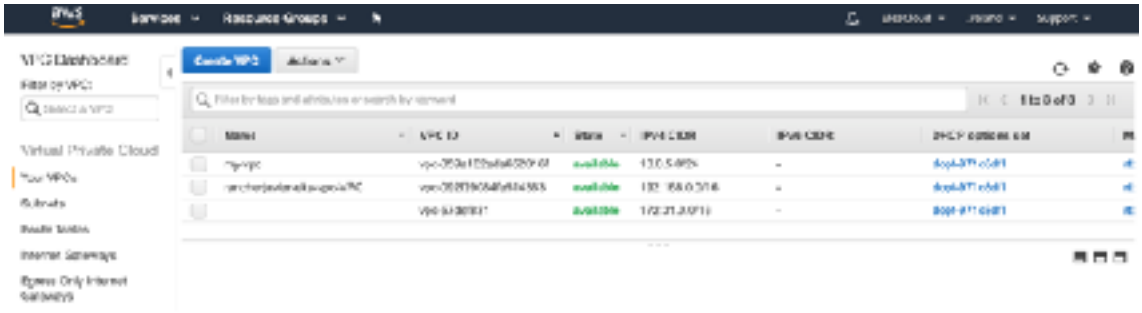
First job is successfully completed:



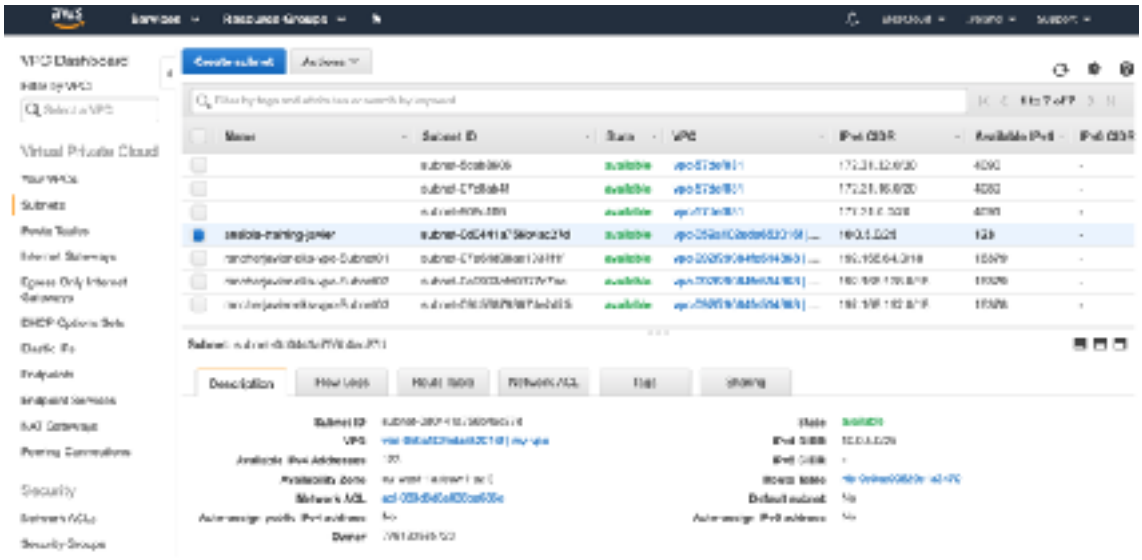
Now, the second job is also executed:



The new VPC called "my-vpc" is created.



A new subnet called "ansible-training-javier" is also provisioned in the VPC created in the same workflow.



Deploying to Google Cloud Platform

Inventory

You will need to install additional packages in order to gather information about GCP-based hosts:

pip install requests google-auth apache-libcloud

First, you have to create a new service account for Ansible. Open the Google Cloud dashboard and select your project from the top header.

Select IAM & admin then go to the Service Accounts section. You can create a new service account by clicking on the Create Service Account button at the top.

From the pop-up, we can create a service account and download the JSON credentials file as shown below.

Considering Ansible Engine, once we have the credentials we should put them in roles/gce/vars/secrets.yml:

```
---
gs_access_key: XXXXXXXXXXXXXXXXXX
gs_secret_key: XXXXXXXXXXXXXXXXXX
```

Now, we encrypt them:

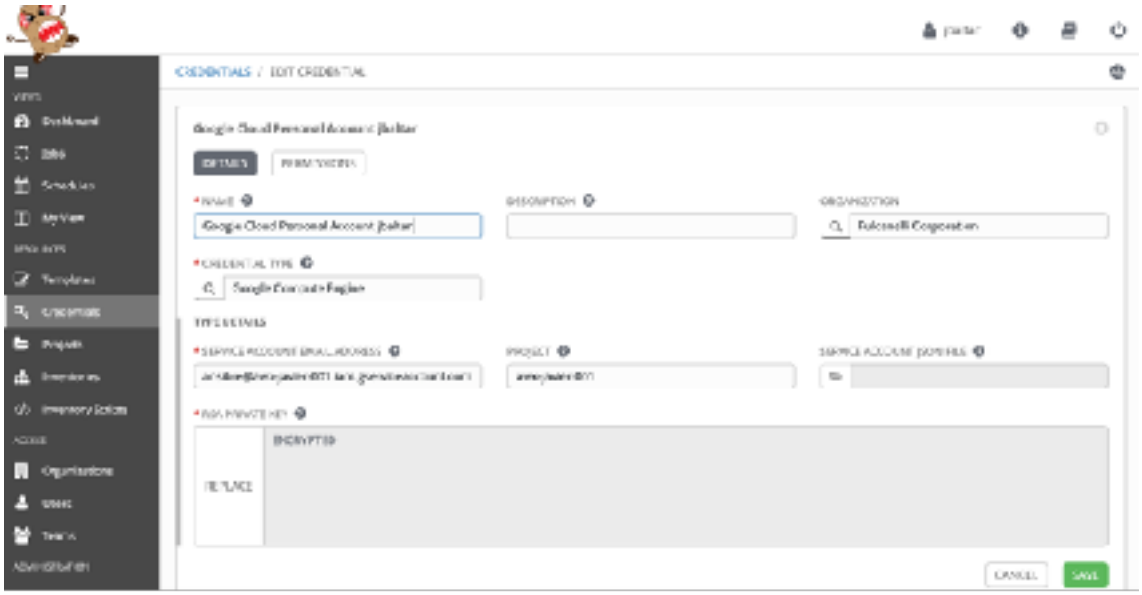
$ ansible-vault encrypt roles/gce/vars/secrets.yml

From AWX perspective, the configuration is similar to AWS.

Click on credentials and choose Google Compute Engine.

Add the service account details retrieved from the file downloaded:

Once the credential is saved, it is automatically encrypted.



You can generate a ssh key for Ansible and uploaded it to the Google Cloud console:
Click on Compute Engine > Metadata > SSH Keys and add the new key.

The inventory configuration is similar to the one we did for AWS.

Click on Inventories > Sources and add a Google Compute Engine source type.

As an example, the following playbook launched a new VM in GCP.

```
- name: Create Google Cloud VM
  hosts: localhost
  gather_facts: no
  connection: local

  vars:
    machine_type: n1-standard-1
    image: centos7
    service_account_email: ansible@awx-javier-001.iam.gserviceaccount.com
    credentials_file: /home/googlecloud/playbooks/key.json
    project_id: AWX-Javier-001

  tasks:
    - name: Launch VM
      gce:
          instance_names: instance-1
          machine_type: "{{ machine_type }}"
          image: "{{ image }}"
          service_account_email: "{{ service_account_email }}"
          credentials_file: "{{ credentials_file }}"
          project_id: "{{ project_id }}"
```

Running the playbook, the new virtual machine is created: