AnsibleWor(ks)X - A Graphical User Interface Configuration Management Automation Tool

Introduction

Ansible awx is an open source community project that provides a web based user interface and API to manage organization's Ansible Playbook, Inventories, Vault, and Credentials. It is an open source version of Red Hat Ansible Tower.
Ansible AWX makes Ansible more comfortable for IT teams who are not comfortable with the command lines by providing GUI version of ansible

Ansible Tower is **the enterprise version of Ansible** that also provides web based user interface,role-based access control (RBAC), workflows, and continuous integration and continuous delivery (CI/CD) for helping your team scale with more efficiency and flexibility. .

There are two ways to install AWX

- Ansible AWX services will be deployed inside containers, and for that, we need to install docker and docker-compose to run multiple container images.
  But it was supported till the version v17 . currently it is no longer supported by the Red Hat team
- From version 18.x of AWX , the recommended installation method is via AWX operator.
  This operator installation procedure requires kubernetes cluster/ minikube
  Setting up awx in a kubernetes cluster/minikube is pretty easy to get up and running.

Basic setup configuration

- Docker should be installed in server machine where awx installed
- Python 3 should be installed on awx server and all target machines
- Here we are using awx-ee:21.11.0 version
- Here our target machine/hosts is localhost and container

Ansible : install AWX on Ubuntu using kubernates/minikube cluster

minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes.
Here we are using a minicube for AWX .
For installation of awx in minikube, we have followed the procedure given in the link below
https://github.com/ansible/awx-operator with some additional steps .

1. Update and upgrade your Debian System before you install Ansible AWX using command
sudo apt update && sudo apt -y full-upgrade

2.To create a minikube cluster first we have To install the latest minikube stable release on x86-64 Linux using below command.

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64.
sudo install minikube-linux-amd64 /usr/local/bin/minikube

3. From a terminal with administrator access (but not logged in as root), run below command to start kubernetes/minikube with certain cpu and ram.

minikube start --cpus=4 --memory=6g --addons=ingress

```
$ minikube start --cpus=4 --memory=6g --addons=ingress
😄  minikube v1.23.2 on Fedora 34
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🏃  Updating the running docker "minikube" container ...
🐳  Preparing Kubernetes v1.22.2 on Docker 20.10.8 ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image k8s.gcr.io/ingress-nginx/controller:v1.0.0-beta.3
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.0
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.0
🔎  Verifying ingress addon...
🌟  Enabled addons: storage-provisioner, default-storageclass, ingress
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Figure 1: screenshot of minikube started

4. Once Minikube is deployed, check if the node(s) and kube-apiserver communication is working as expected using command

minikube kubectl -- get nodes

5. To checks kubernetes has started some pods or not using below command:

minikube kubectl -- get pods -A

```
$ minikube kubectl -- get nodes
NAME       STATUS   ROLES                  AGE    VERSION
minikube   Ready    control-plane,master   113s   v1.22.2

$ minikube kubectl -- get pods -A
NAMESPACE       NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx   ingress-nginx-admission-create--1-kk67h     0/1     Completed   0          2m1s
ingress-nginx   ingress-nginx-admission-patch--1-7mp2r      0/1     Completed   1          2m1s
ingress-nginx   ingress-nginx-controller-69bdbc4d57-bmwg8   1/1     Running     0          2m
kube-system     coredns-78fcd69978-q7nmx                    1/1     Running     0          2m
kube-system     etcd-minikube                               1/1     Running     0          2m12s
kube-system     kube-apiserver-minikube                     1/1     Running     0          2m16s
kube-system     kube-controller-manager-minikube            1/1     Running     0          2m12s
kube-system     kube-proxy-5mmnw                            1/1     Running     0          2m1s
kube-system     kube-scheduler-minikube                     1/1     Running     0          2m15s
kube-system     storage-provisioner                         1/1     Running     0          2m11s
```

Figure 2: screenshot of minikube started making pods

Note:It is not required for kubectl to be separately installed since it comes already wrapped inside a minikube.

6. Once Kubernates started making pods, Let's create an alias for easier usage using command
alias kubectl="minikube kubectl --"

7. Once we have a running Kubernetes cluster, we can deploy AWX Operator into our cluster using Kustomize.
To Install Kustomize by downloading precompiled binaries use below command:
Curl -s "https://raw.githubusercontent.com/kubernetes-sigs/kustomize/master/hack/install_kustomize.sh" | bash

8. Confirm installation of Kustomize by checking the version using command:
kustomize version

9. Than , create a file called **kustomization.yaml** with the following content:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  # Find the latest tag here: https://github.com/ansible/awx-operator/releases
  - github.com/ansible/awx-operator/config/default?ref=1.1.4
  # Add this extra line:
  - awx-demo.yaml

# Set the image tags to match the git version from above
images:
  - name: quay.io/ansible/awx-ee:21.11.0
    newTag: 1.1.4

# Specify a custom namespace in which to install AWX
namespace: awx
```

Figure 3: kustomization.yaml file

In place of **newTag and ref** pass the latest version of awx-operator which can be found by going to the **https://github.com/ansible/awx-operator/releases** link. Here we are using awx-operator version 1.1.4

We can also Save the latest version from **AWX Operator releases** as *RELEASE_TAG* variable And pass that variable instead of passing the hardcoded latest awx-operator version .
The AWX Operator is used to manage one or more AWX instances in any namespace within the cluster.

10. Install the manifests by running below command:
kustomize build . | kubectl apply -f -

```
$ kustomize build . | kubectl apply -f -
namespace/awx created
customresourcedefinition.apiextensions.k8s.io/awxbackups.awx.ansible.com created
customresourcedefinition.apiextensions.k8s.io/awxrestores.awx.ansible.com created
customresourcedefinition.apiextensions.k8s.io/awxs.awx.ansible.com created
serviceaccount/awx-operator-controller-manager created
role.rbac.authorization.k8s.io/awx-operator-awx-manager-role created
role.rbac.authorization.k8s.io/awx-operator-leader-election-role created
clusterrole.rbac.authorization.k8s.io/awx-operator-metrics-reader created
clusterrole.rbac.authorization.k8s.io/awx-operator-proxy-role created
rolebinding.rbac.authorization.k8s.io/awx-operator-awx-manager-rolebinding created
rolebinding.rbac.authorization.k8s.io/awx-operator-leader-election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/awx-operator-proxy-rolebinding created
configmap/awx-operator-awx-manager-config created
service/awx-operator-controller-manager-metrics-service created
deployment.apps/awx-operator-controller-manager created
```

Figure 4: screenshot of install the manifests

11. Wait for some minutes and we should have the **awx-operator** running command:
We can check if awx-operator deployed or not using command :
kubectl get pods -n awx
If its status shows Running, it means our awx-operator is deployed successfully.

```
$ kubectl get pods -n awx
NAME                                                READY   STATUS    RESTARTS   AGE
awx-operator-controller-manager-66ccd8f997-rhd4z    2/2     Running   0          11s
```

Figure 5: screenshot of successfully deployed awx-operator in minikube

12.Since, we don't want to keep repeating -n awx, let's set the current namespace for kubectl using command: **kubectl config set-context --current --namespace=awx**

13. now,create a file named **awx-demo.yaml** in the same folder with content below. The name we give in metadata.name will be the name of the resulting AWX deployment.

```yaml
---
apiVersion: awx.ansible.com/v1beta1
kind: AWX
metadata:
  name: awx-demo
spec:
  service_type: nodeport
  # default nodeport_port is 30080
  nodeport_port: <nodeport_port>
```

Figure 6: awx-demo.yaml file

Note: In nodeport_port give the port no. on which you want to run awx

14. Make sure to add this new file to the list of "resources" in your kustomization.yaml file:

```
...
resources:
  - github.com/ansible/awx-operator/config/default?ref=<tag>
  # Add this extra line:
  - awx-demo.yaml
...
```

Figure 7: kustomization.yaml file content

15. Finally, run kustomize again to create the AWX instance in your cluster using below command: **kustomize build . | kubectl apply -f -**

16.After a few minutes, the new AWX instance will be deployed. You can look at the operator pod logs in order to know where the installation process is at using below command:
kubectl logs -f deployments/awx-operator-controller-manager -c awx-manager

17. After a few minutes, the new AWX instance will be deployed in cluster and we can monitor its installation logs using below command:
kubectl get pods -l "app.kubernetes.io/managed-by=awx-operator"
kubectl get svc -l "app.kubernetes.io/managed-by=awx-operator"

```
$ kubectl get pods -l "app.kubernetes.io/managed-by=awx-operator"
NAME                        READY   STATUS    RESTARTS   AGE
awx-demo-77d96f88d5-pnhr8   4/4     Running   0          3m24s
awx-demo-postgres-0         1/1     Running   0          3m34s

$ kubectl get svc -l "app.kubernetes.io/managed-by=awx-operator"
NAME                TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
awx-demo-postgres   ClusterIP   None           <none>        5432/TCP       4m4s
awx-demo-service    NodePort    10.109.40.38   <none>        80:31006/TCP   3m56s
```

Figure 8: screenshot of resource creation by awx-operator

18. Once awx deployed in cluster, the AWX instance will be accessible by running below command: **minikube service -n awx awx-demo-service --url**

19. By default, the **username is admin** and to retrieve the password use below command:
kubectl get secret awx-demo-admin-password -o jsonpath="{.data.password}" | base64 --decode ; echo

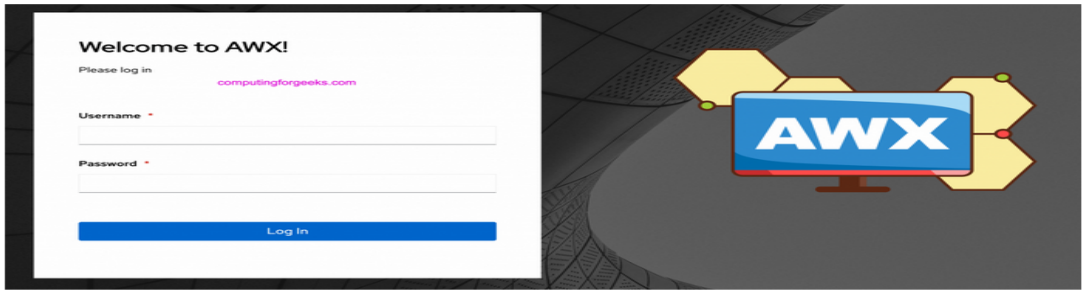
Now we are done with basic installation , Now just go to the url and login using **admin** as username and password which is generated by above command.

How to use run Ansible Playbook using AWX

- We can get url of Ansible AWX Dashboard using below command:
  minikube service -n awx awx-demo-service --url

Ansible AWX web portal is now accessible on **http://hostip_or_hostname:30080** (by default portno. is 30080 if we have not set the port no. of awx in awx-demo.yml file)

- Launch your browser to access the dashboard and browse the above url. You will get a screen as shown in figure 9.



F **Figure 9:**AWX Login Page

- We have to use **admin** as username and we can get password by running the below command :
  kubectl get secret awx-demo-admin-password -o jsonpath="{.data.password}" | base64 --decode ; echo
  Sample output of above command: ***LkyWUKDwKdnhiEcvFe0zRQ9jOJCz7eM***

- After login using username and password we enter into AWX AdministrationDashboard which is shown in figure 10 . Now we can start adding inventory, credential, hosts,projects ,Templates, Ansible roles and automate our infrastructure and application deployment.

DashBoard

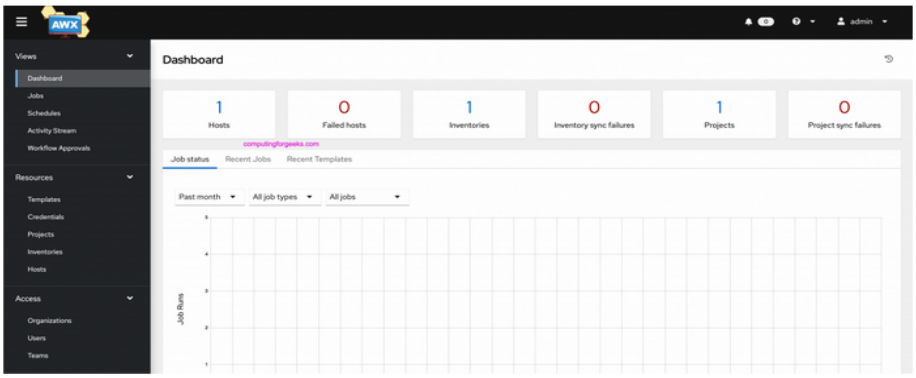


Figure 10: Dashboard

- Now,select the organization tab at left side of screen ,click the add button to add new organization , Than add name of organization and click save as shown in figure 11

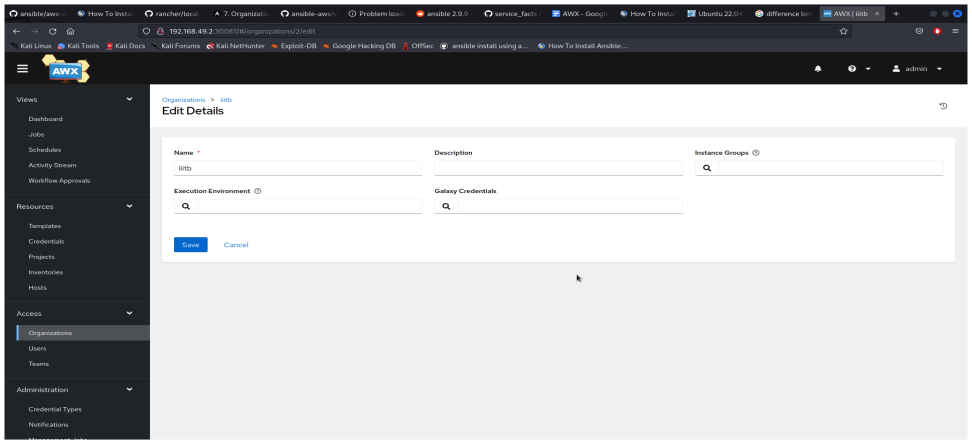


Figure 11: screenshot to Add Organization

Inventory

- Now go to inventory and click on the add button to create inventory.
- Enter inventory name and if you want to create this inventory for a specific organization, you can select the created organization or else choose default. Then save it.
- In variable,we pass those parameters which we want to apply to all hosts connected to that inventory . here we want to connect with hosts/target machine using ssh .We have to pass ansible_connection: ssh in variable
- If you will get any error related to python interpreter while executing template then Add ansible_python_interpreter: '{{ ansible_playbook_python }}' in variables of that inventory.

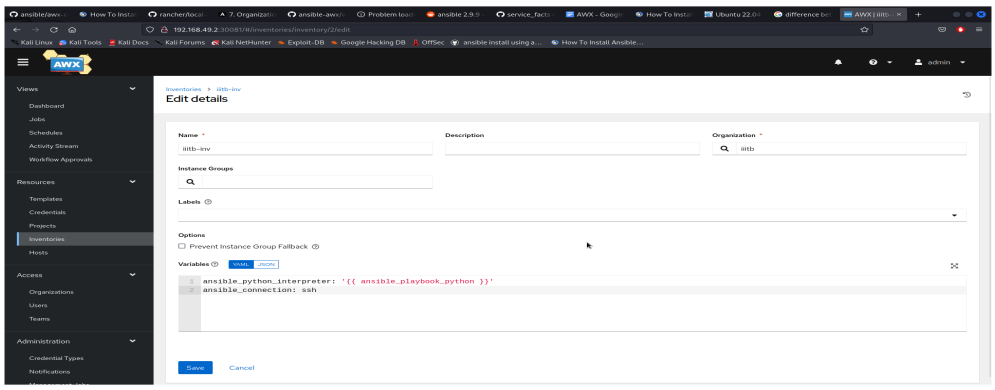


Figure 12: screenshot To add Inventory

Hosts

- To add a host, which you want to access using awx, click hosts and then select add button. Add the IP address or host name of the host/target machine, choose the inventory to which you want to add this host, then add below details of that host and save it as shown in figure 13.

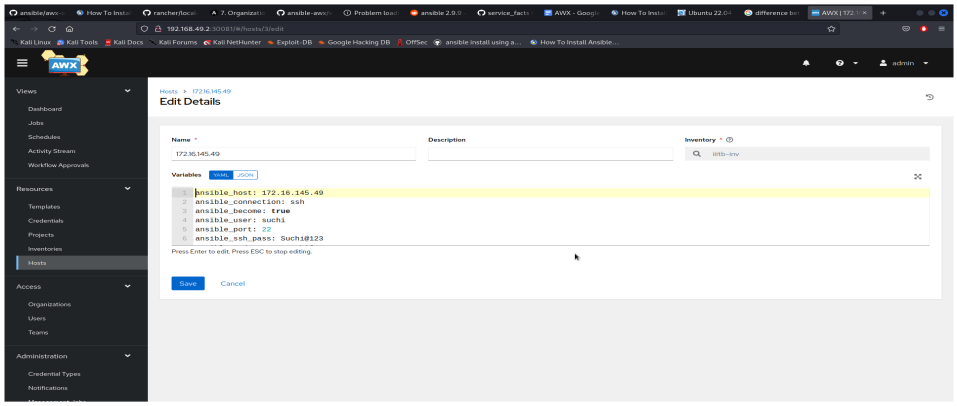


Figure 13: screenshot To add Hosts

We have to pass variables in hosts as shown below :

ansible_host: 172.16.145.49 # ip address of host
ansible_user: suchi # enter username of host you are accessing
ansible_port: 22 # default ansible_port is 22 so no need to write
ansible_become: true # Ansible_become used for privilege escalation.
ansible_ssh_pass: Suchi@123 # password of ansible_user of host machine
ansible_sudo_pass: Suchi@123

Note: if the ansible_user is root we don't need to pass ansible_sudo_pass.
If you want to connect with the target machine/hosts with ssh and not password, it is not required to pass ansible_ssh_pass variables in host configuration.

Credentials

- If you want to access the target machine/hosts using password we can directly select demo credentials and no need to configure anything.
- But if you want to access the target machine/hosts using ssh without password , we have to use the key sharing mechanism .
- Go inside minikube where awx installed using command **Docker exec -it *minikube_container_id* /bin/bash**
- Generate public and private key of minikube machine using command : **ssh-keygen**
- Now, copy the public key of minikube in all target machines/hosts using command **ssh-copy-id *user@ip*** in from minikube terminal . It will copy the public key of the minikube in that ip machine .
- Here ***user*** is the user of the host you are accessing and its ***ip*** which we have mentioned in that host configuration.

- Now, we have to add the private key of minikube in awx credential . This private key can be found inside minikube by using below command :
  cd ~/.ssh
  cat id_rsa
- Now copy all content of id_rsa file which contain private key of minikube
- Now in awx go to the credential tab , click add button. Give credential name. Paste that copied key in ssh private key.
- In username add login user of awx who will execute templates . In our case we are using admin .
- Select **sudo** in privilege escalation method and **root** in privilege escalation username to give root permission to execute. Then save it as shown in figure 14.

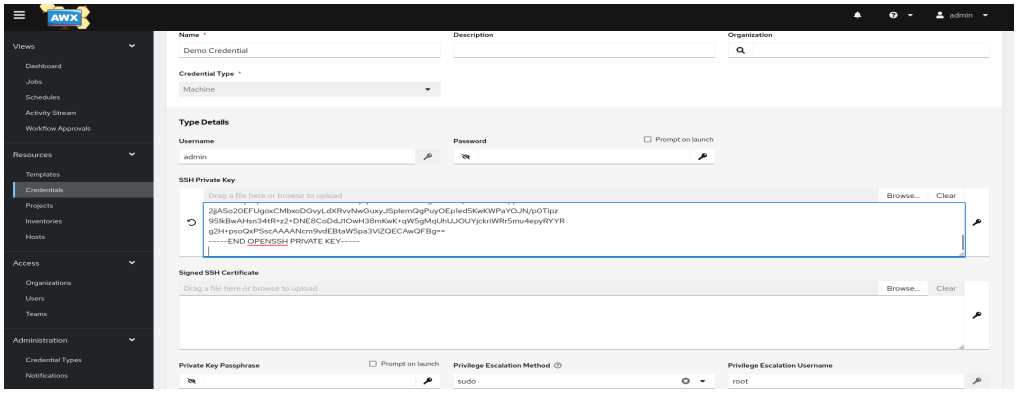


Figure 14: screenshot To add credentials

Projects

- Now select the project tab and click the add button . Basically the project is the collection of playbooks which you want to execute into your host machine using awx.
- Give your project name and select your organization . Here we are fetching playbooks from github so, select git in source control type and enter the url of github playbook repository where all the playbooks are present.
- It is recommended to use github to fetch playbook instead of fetching from machine as there will be problem of maintaining version control in local machine
- Enable update revision on launch so that whenever we launch our project i.e run our template it will always fetch the latest version of playbook so even in case of updation of playbook in github it will always fetch updated version .
- Now save it as shown in figure 15.

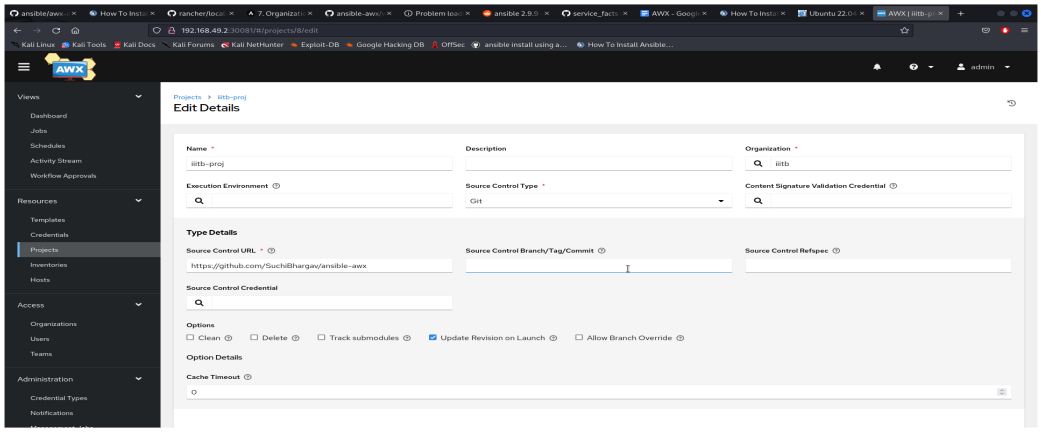


Figure 15:screenshot To add Project

- This is the github account from where we are fetching playbooks in awx which is shown in figure 16 . This url has to be used in source control url in project shown in figure 15
- https://github.com/SuchiBhargav/ansible-awx here all playbooks are present

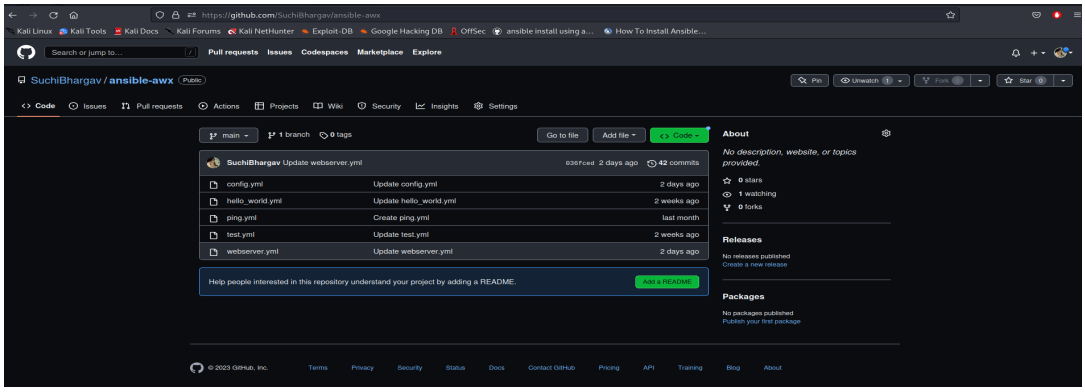


Figure 16: Github Repo where all playbooks are present

Templates

- Now select the Template tab,click add button then enter template name and select your project , inventory and playbook that is present in your github repo. Also enable the privilege Escalation option at the bottom of the template then save it.
- Templates are basically jobs where we define what playbook we want to run and what inventory source the hosts that we want to run against are in.

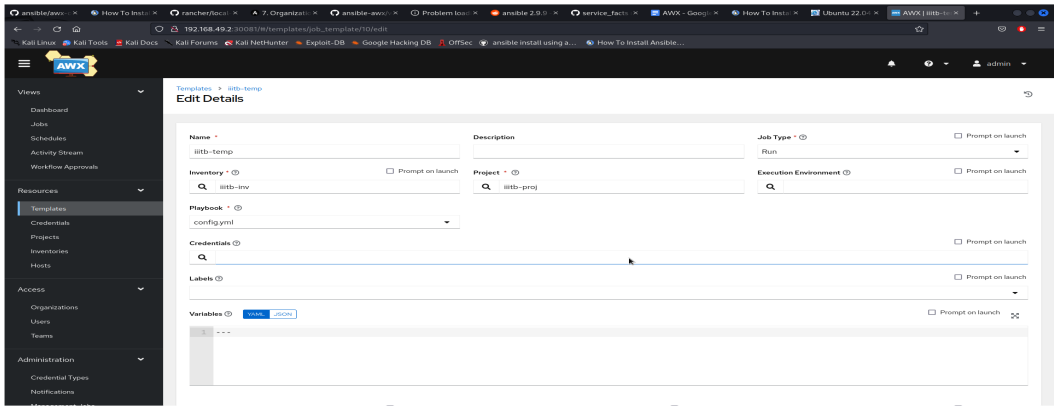


Figure 17: screenshot To add Templates

Execution of Playbook using Template Launch

- Now our Template is ready to launch just click the rocket icon or launch button to run the playbook.

Some points to be noted when using awx for accessing Hosts

- if we want to access any container and deploy anything inside it using ansible awx. we must create a docker network and connect both minikube and container in the same network so that they can communicate with each other.
- While minikube can directly access and deploy inside localhost without the need to connect both in the same network.
- Commands to be used to create network and connect minikube and container in same network as shown below:
    1. **docker network create -d bridge *network_name***
    2. **docker network connect *network_name container_id***
    3. **docker network connect *network_name container_id_of_minikube***
- Now , they can communicate with each other
- If your minikube stopped for some reason , first disconnect your minikube from your created network using below command otherwise minikube will not start and gives you error : **docker network disconnect *network_name container_id_of_minikube***
- Than only start your minikube again using command:
  minikube start --cpus=4 --memory=6g --addons=ingress
- After it start than again connect minikube with your network using command
  docker network connect *network_name container_id_of_minikube*
- Also we have to start **ssh** service in both localhost and container so that minikube can communicate with both using ssh
- We can start ssh using the command : **Service ssh start**
- If ssh is not working , it might be possible that ssh is not installed so use command:
  apt install openssh-server and again try to start ssh

- We can also cross check if ssh working or not by going inside minikube awx container using command : **docker exec -it *container_id* /bin/bash**
  Execute command **ssh root@*ip*** **or** **ssh user@ip** here ip is the ip address of hosts to which awx wants to communicate to check if it is able to ssh into that hosts or not and user is the username of the host you want to communicate with.
- But for accessing the root user of the host/target machine using ssh by awx, you have to give permission by changing the file of the host by going inside /etc/ssh/ and updating file sshd_config by enabling the below parameters.
  permitrootlogin yes
  PubkeyAuthentication yes
  PasswordAuthentication yes
- And again restart ssh service using command : **service ssh restart**
- To create and run container use command:
  docker run -it -d –name *container_name image_name*
- If you want to give any particular **user** sudo privilege ,Install sudo if inside folder **/etc sudoers** file is not there
- If it is, add user below user privilege **all:all and no passwd** as shown in below figure 18

```
# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
suchi   ALL=(ALL) ALL
suchi   ALL=(ALL) NOPASSWD: ALL
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:
```

Figure 18: sudoers file

Here we are giving username: suchi a sudo privilege
***<user>* ALL=(ALL) ALL**
***<user>* ALL=(ALL) NOPASSWD: ALL**

Alternatively, we can add the user to the sudo group using below command
sudo usermod -aG sudo *<user>*

replace **<user>** with the **actual user.**

Types of ERROR You might get during execution :

1. while doing ssh If you are getting error of **permission denied** even after entering correct password Or getting error of **Host key verification failed**. Use command **ssh-keygen -R *ip*** here ip is the ip address of client you want to ssh And try again command **ssh root@*ip***

2. while executing playbook if you are getting an error if **The module fails to execute correctly, you probably need to set the interpreter.\nSee stdout/stderr for the exact error.**

You have to install python in cd /usr/bin/ in hosts

1. Go to **cd /usr/bin/**
2. Than run command **apt install python 3**

Playbooks which we executed using awx

Playbook 1: Ping.yml

This playbook will ping all the hosts which are attached with inventory

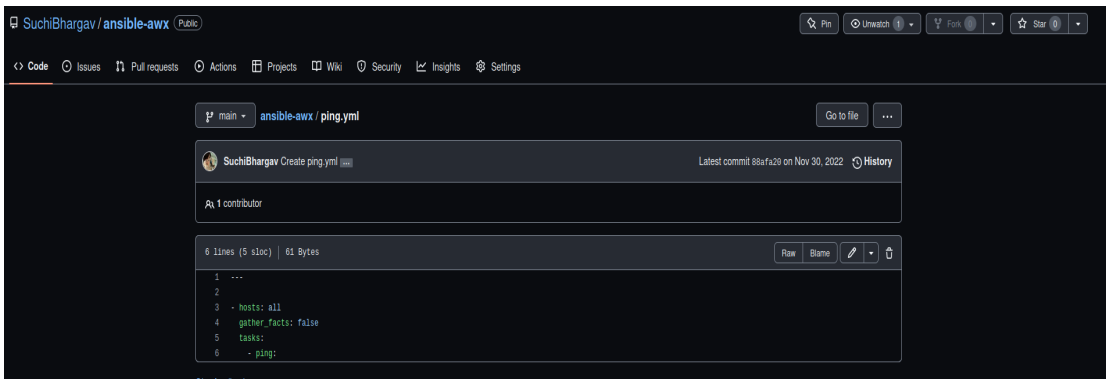


Figure 19 : playbook1

After Launching Template to execute it

Output:

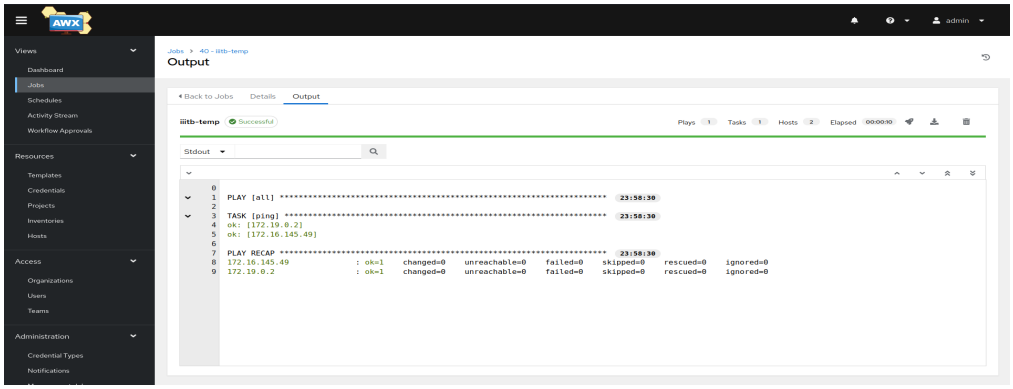


Figure 20: Output of playbook1

Playbook 2: Hello_world.yml

This playbook will print hello world than install nginx server inside localhost and container and also start it

SuchiBhargav Update hello_world.yml

1 contributor

13 lines (12 sloc) | 294 Bytes

```
- name: Hello World Sample
  hosts: all
  become: true
  connection: ssh
  tasks:
    - name: Hello Message
      debug:
        msg: "Hello World!"
    - name: ensure nginx is at the latest version
      apt: name=nginx state=latest
    - name: start nginx
      command: service nginx start

```

Figure 21: Playbook2

Output:

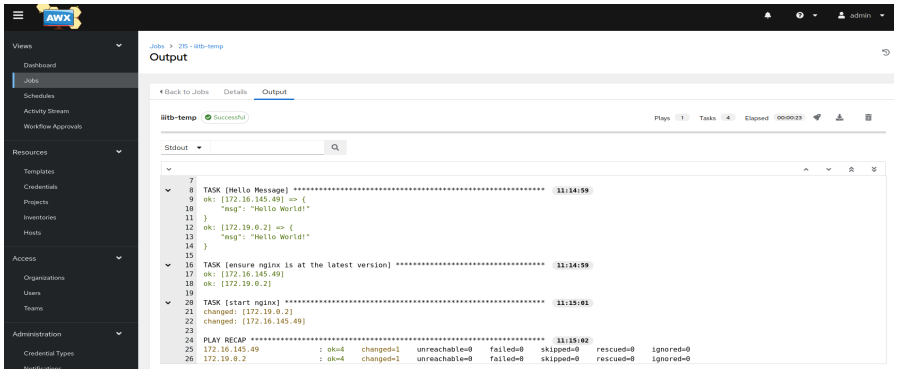


Figure 22: output after playbook2 execution

By using command : service –status-all we can confirm if nginx is started or not after execution of above playbook.

Playbook 3: Config.yml

This playbook will create a file inside hosts and also show its ip address

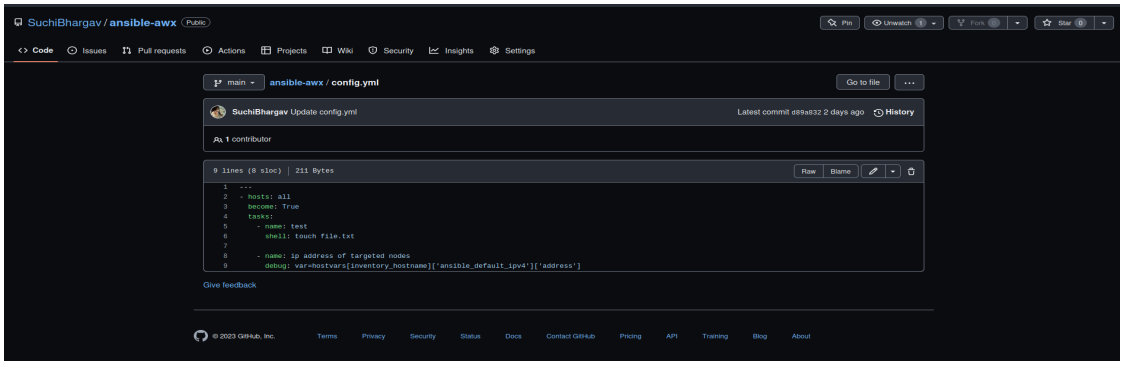


Figure 23: Playbook3

Output:

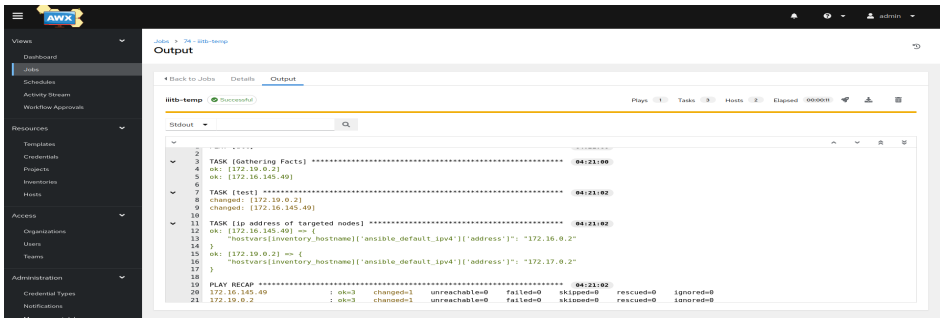
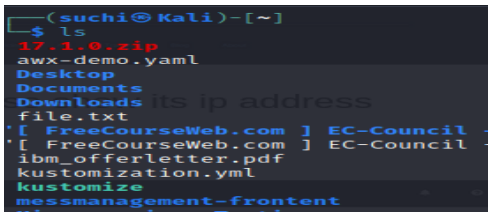


Figure 24: Output after playbook3 execution

Here we can see, In localhost and container db1a3d5b9edd ***file.txt*** is created which can be seen using **ls** command in localhost and container

As shown in figure 25 and 26



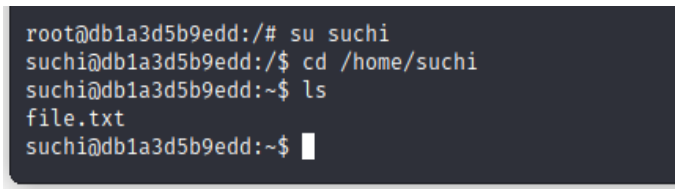


Figure 25 :Localhost ls output

Figure 26: Container ls output

Playbook 4: test.yml

This playbook when executed will display all the folders and files present in it.

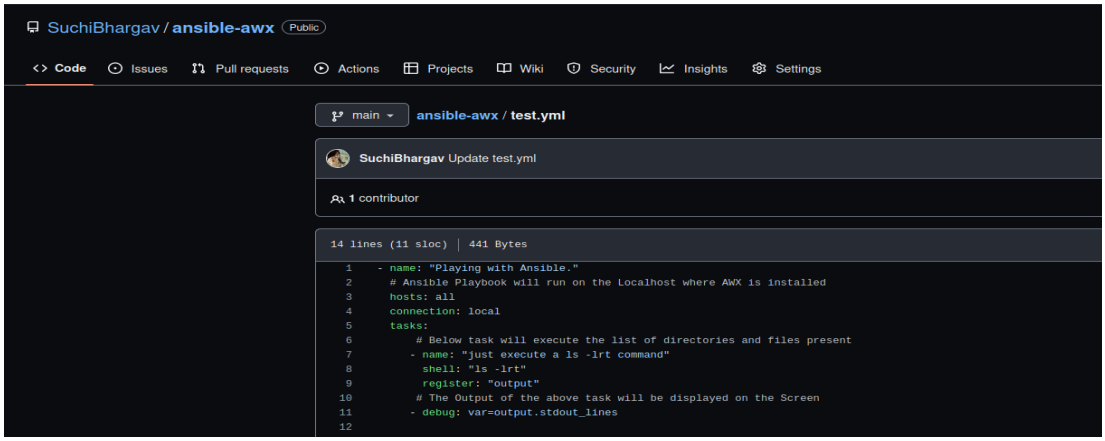


Figure 27: Playbook 4

Output:

Figure 28: Output of playbook 4

Output of ls -al in localhost and container is the same as output we got after executing playbook which can be seen in figure 29.

Figure 29: Command ls -al output in localhost and container

Conclusion:

We can see everything is working as expected and almost 90% of jobs of ansible can be done using ansible awx.

Reference:

- https://github.com/ansible/awx-operator
- https://www.howtoforge.com/ansible-awx-guide-basic-usage-and-configuration/
- https://www.ibm.com/docs/en/pm-and-q/2.5.1?topic=pip-generating-copying-rsa-keys-among-all-node-computers
- https://www.howtoforge.com/tutorial/how-to-install-ansible-awx-with-docker-on-centos/
- https://unix.stackexchange.com/questions/23291/how-to-ssh-to-remote-server-using-a-private-key
- https://www.baeldung.com/linux/sudo-privileges-user