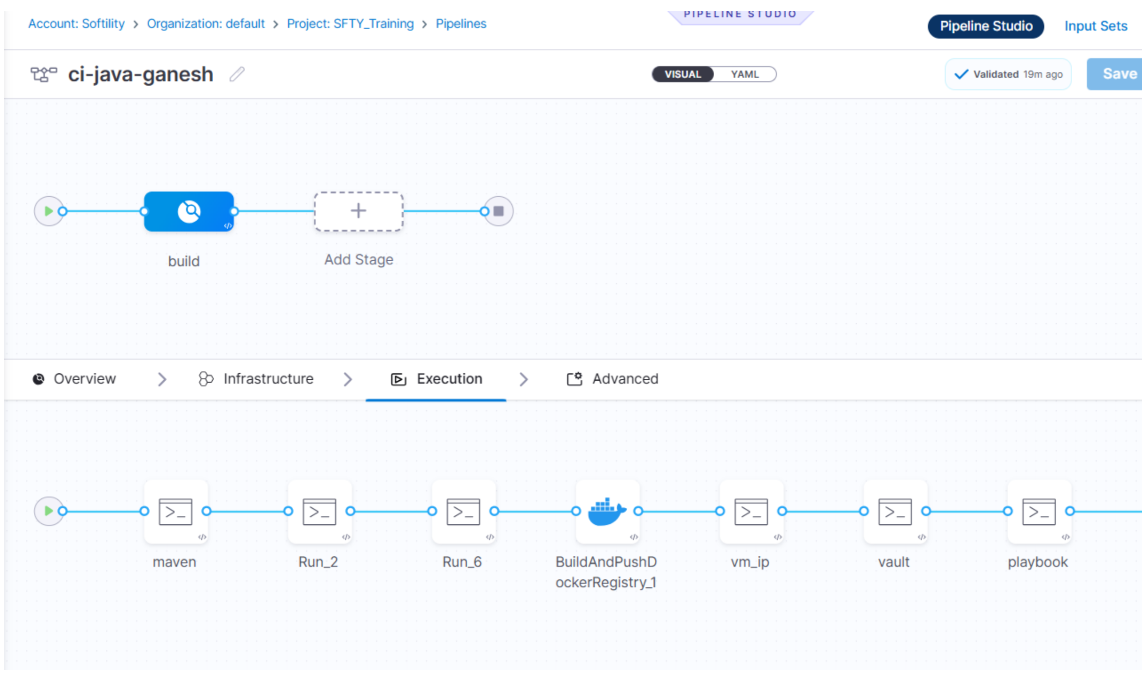
Maven artifact push to jfrog without plugin of jfrog



Here we required

1. Settings.xml
   - Here in this settings.xml we give the username and access token of that use
2. Pom.xml
   - In pom.xml we give them some decency like jfrog repo url and id here id in pom.xml and settings.xml id must match then only it was pushed into repo
3. Here the setting files can be present in current working dir or .m2/ dir
   - If it was present in current dir then command is

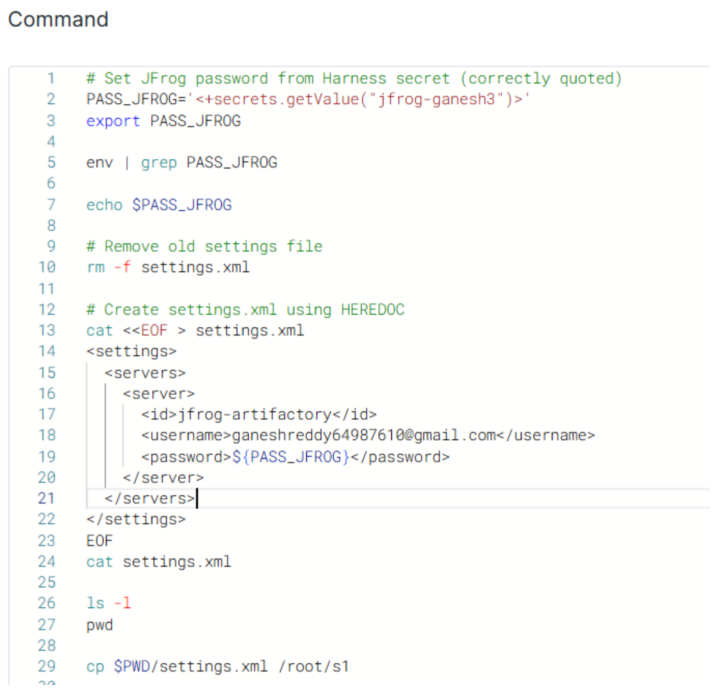
     mvn clean package –settings=settings.xml
   - If it was present on in /root/.m2/ then

1. Mvn clean package

Steps 1: mvn clean package

```
# Set JFrog password from Harness secret (correctly quoted)
PASS_JFROG='<+secrets.getValue("jfrog-ganesh3")>'
export PASS_JFROG

env | grep PASS_JFROG

echo $PASS_JFROG

# Remove old settings file
rm -f settings.xml

# Create settings.xml using HEREDOC
```

```
cat <<EOF > settings.xml
<settings>
  <servers>
    <server>
      <id>jfrog-artifactory</id>
      <username>ganeshreddy64987610@gmail.com</username>
      <password>${PASS_JFROG}</password>
    </server>
  </servers>
</settings>
EOF
cat settings.xml

ls -l
pwd

cp $PWD/settings.xml /root/s1

# Run Maven with settings
mvn clean deploy --settings=settings.xml
```



Command

```
# Set JFrog password from Harness secret (correctly quoted)
PASS_JFROG='<+secrets.getValue("jfrog-ganesh3")>'
export PASS_JFROG

env | grep PASS_JFROG

echo $PASS_JFROG

# Remove old settings file
rm -f settings.xml

# Create settings.xml using HEREDOC
cat <<EOF > settings.xml
<settings>
  <servers>
    <server>
      <id>jfrog-artifactory</id>
      <username>ganeshreddy64987610@gmail.com</username>
      <password>${PASS_JFROG}</password>
    </server>
  </servers>
</settings>
EOF
cat settings.xml

ls -l
pwd

cp $PWD/settings.xml /root/s1
```

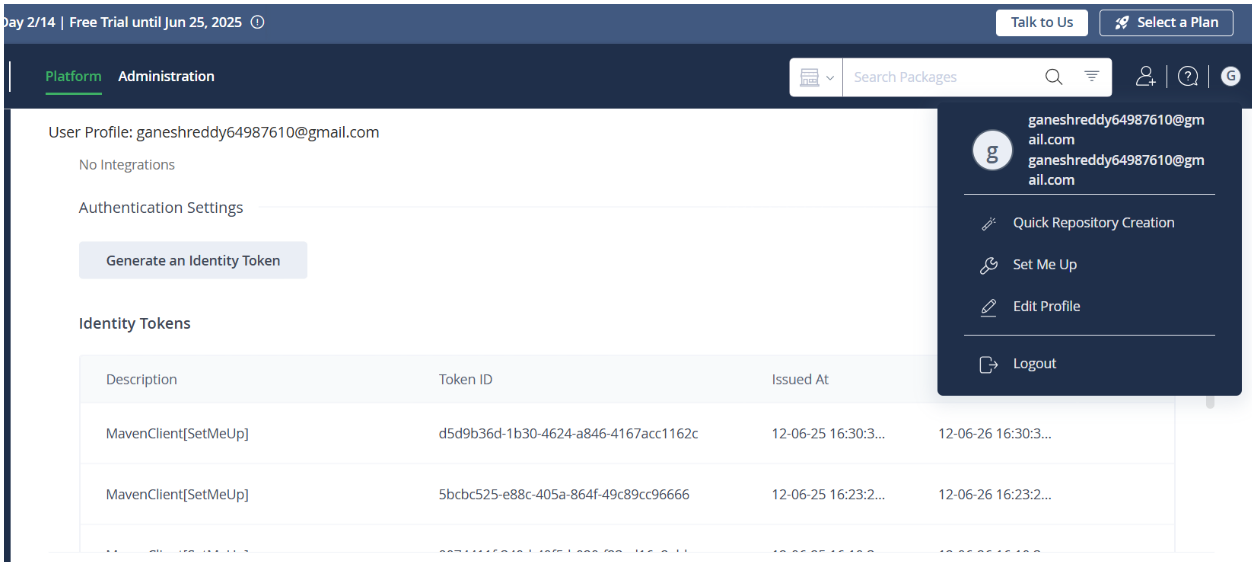
Here we store the password in secrets of harness. We take jfrog access token and pass to setting.xml the we perform the mvn clean package

Step 2: we take the artifact from jfrog

```
pwd

rm -rf $PWD/new99
ls -l
mkdir -p  $PWD/new99
mv  $PWD/Dockerfile $PWD/new99/
cd $PWD/new99

PASS_JFROG='<+secrets.getValue("jfrog-ganesh3")>'
export PASS_JFROG

REPO_BASE_URL="https://trialuu1981.jfrog.io/artifactory/repo3"
GROUP_PATH="com/example/demo"
VERSION="0.0.1-SNAPSHOT"

# Download maven metadata
curl -u "ganeshreddy64987610@gmail.com:${PASS_JFROG}" \
     -s "${REPO_BASE_URL}/${GROUP_PATH}/${VERSION}/maven-metadata.xml" -o
maven-metadata.xml

# Extract timestamp and buildNumber
TIMESTAMP=$(xmllint --xpath "string(//snapshot/timestamp)" maven-
metadata.xml)

BUILDNUM=$(xmllint --xpath "string(//snapshot/buildNumber)" maven-
metadata.xml)

# Construct JAR file name
JAR_FILE="demo-${TIMESTAMP}-${BUILDNUM}.jar"

# Download JAR
curl -u "ganeshreddy64987610@gmail.com:${PASS_JFROG}" \
     -O "${REPO_BASE_URL}/${GROUP_PATH}/${VERSION}/${JAR_FILE}"

ls -l

pwd


ls -l
```

here we create the new dir and pass the artifact to that dir, here we use timestamp and buildnum because we don't know the exact time and buildnum of artifact sos we take it from xml of metadata in jfrog and base on that time and build number the artifact was fetch from jfrog snapshot folder in jfrop of repo then we pass this to docker it will create the image

jfrog



Day 2/14 | Free Trial until Jun 25, 2025
Talk to Us
Select a Plan
Platform
Administration
Search Packages
ganeshreddy64987610@gmail.com
ganeshreddy64987610@gmail.com
Quick Repository Creation
Set Me Up
Edit Profile
Logout

User Profile: ganeshreddy64987610@gmail.com

No Integrations

Authentication Settings

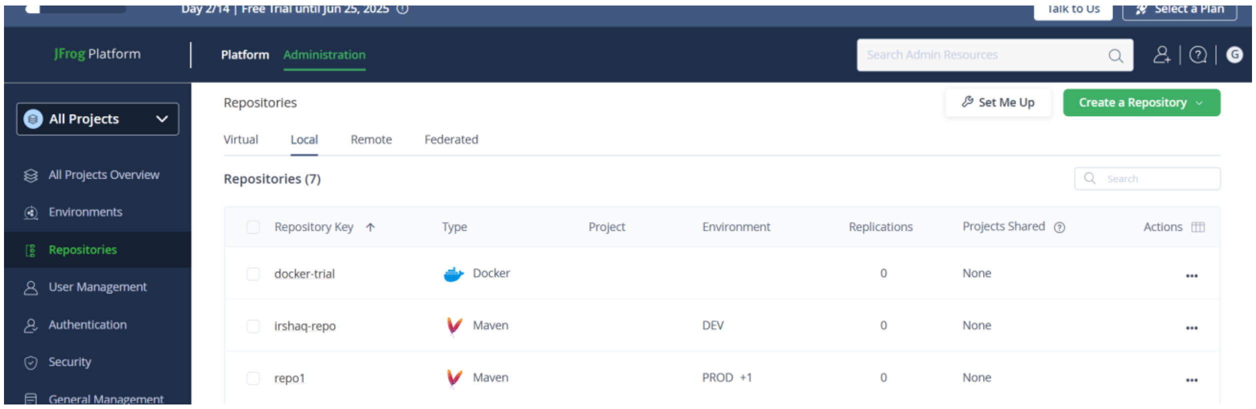
Generate an Identity Token

Identity Tokens

Description	Token ID	Issued At	
MavenClient[SetMeUp]	d5d9b36d-1b30-4624-a846-4167acc1162c	12-06-25 16:30:3...	12-06-26 16:30:3...
MavenClient[SetMeUp]	5bcbc525-e88c-405a-864f-49c89cc96666	12-06-25 16:23:2...	12-06-26 16:23:2...

login to jfrog -> navigate to edit profile -> then generate an identity token

To create the repo in jfrog:

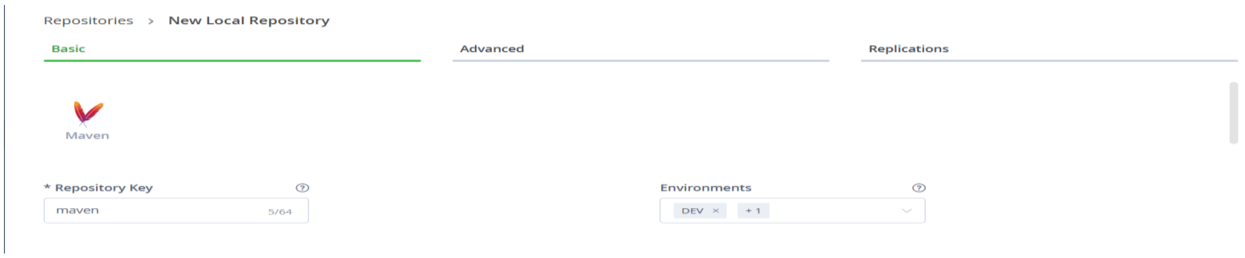


Repositories

Repositories (7)

Repository Key	Type	Project	Environment	Replications	Projects Shared	Actions
docker-trial	Docker			0	None	...
irshaq-repo	Maven		DEV	0	None	...
repo1	Maven		PROD +1	0	None	...

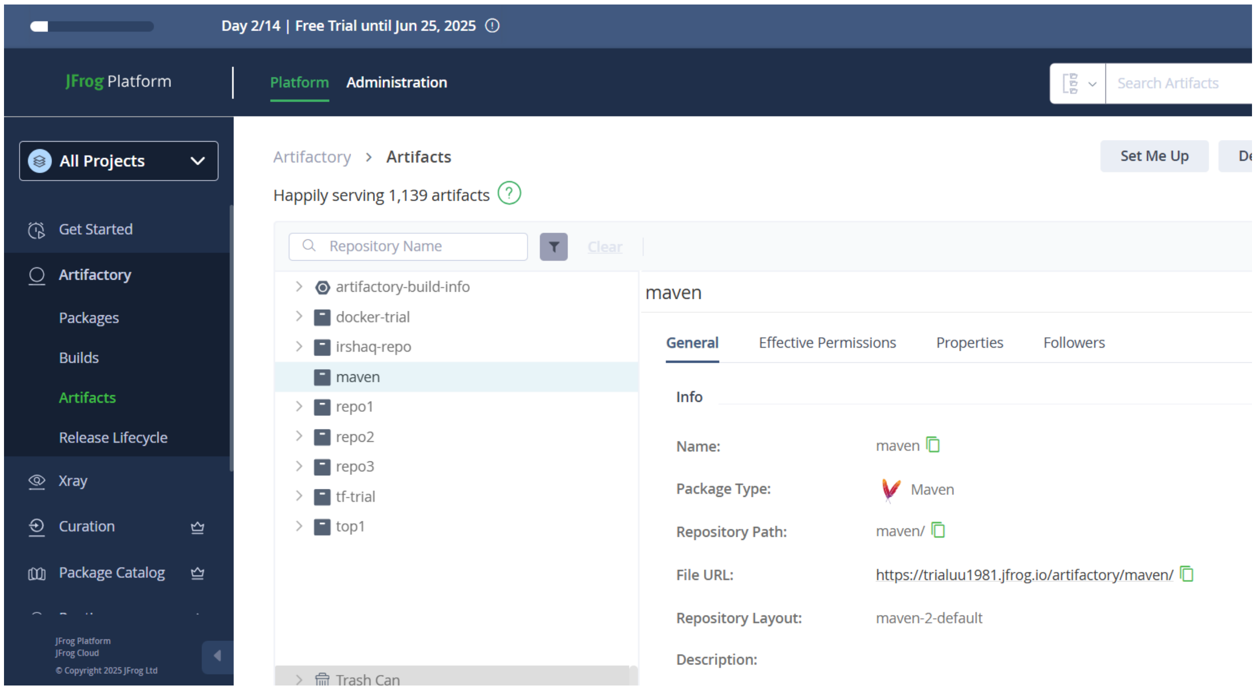
2. Got to artifact or repositories and click on create repo



* Repository Key: maven 5/64

Environments: DEV × +1

4. take the file url and past in pom.xml



Artifactory > Artifacts

Happily serving 1,139 artifacts

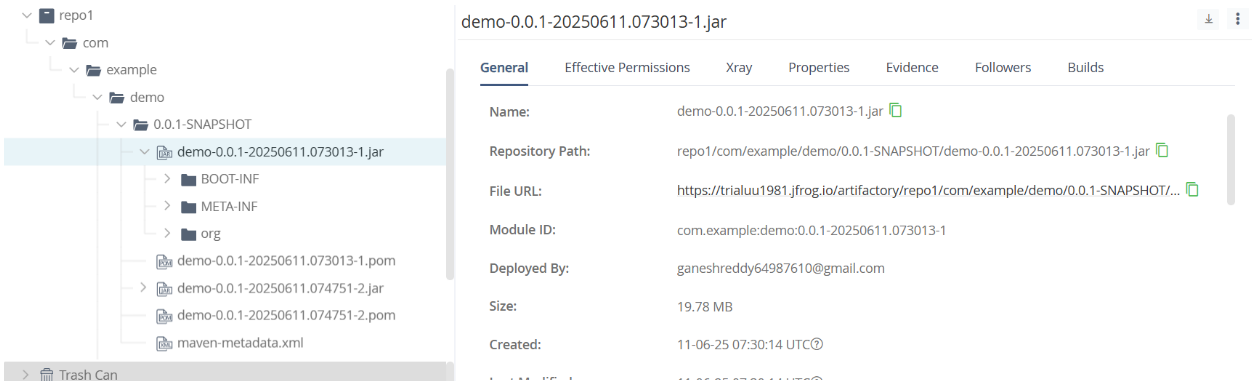
- artifactory-build-info
- docker-trial
- irshaq-repo
- maven
- repo1
- repo2
- repo3
- tf-trial
- top1

maven

Info

Name:	maven
Package Type:	Maven
Repository Path:	maven/
File URL:	https://trialuu1981.jfrog.io/artifactory/maven/
Repository Layout:	maven-2-default
Description:	

5 run maven and us the some folders in the repo which was store the artifact



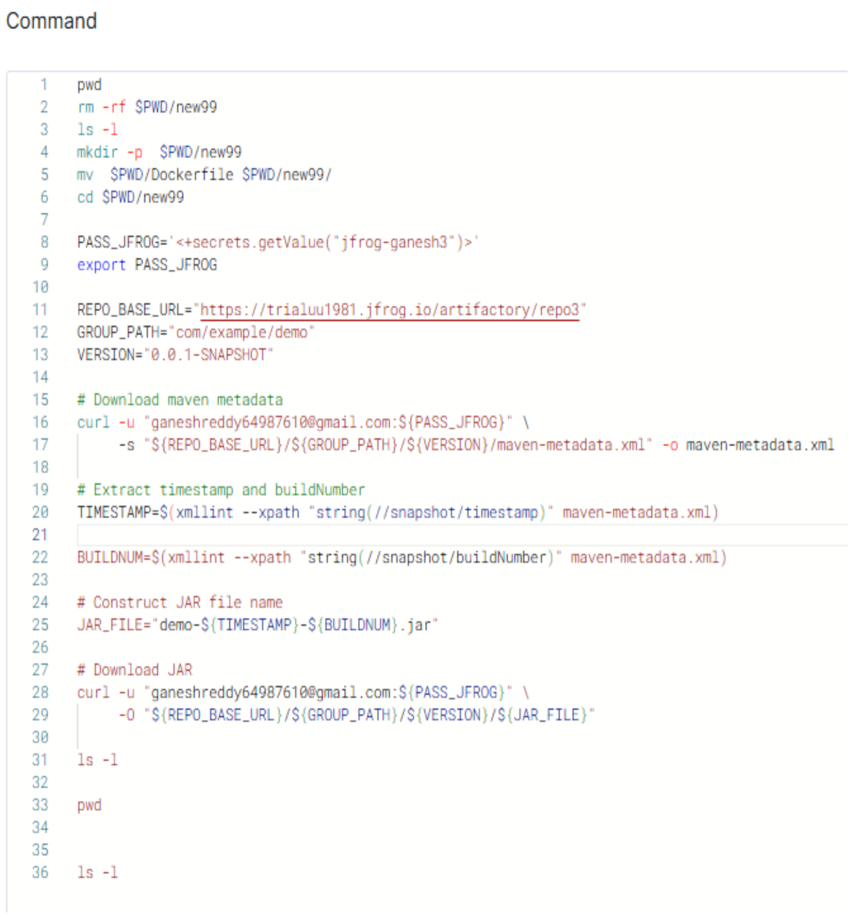
```
repo1
  com
    example
      demo
        0.0.1-SNAPSHOT
          demo-0.0.1-20250611.073013-1.jar
            BOOT-INF
            META-INF
            org
          demo-0.0.1-20250611.073013-1.pom
          demo-0.0.1-20250611.074751-2.jar
          demo-0.0.1-20250611.074751-2.pom
          maven-metadata.xml
```

demo-0.0.1-20250611.073013-1.jar

Name:	demo-0.0.1-20250611.073013-1.jar
Repository Path:	repo1/com/example/demo/0.0.1-SNAPSHOT/demo-0.0.1-20250611.073013-1.jar
File URL:	https://trialuu1981.jfrog.io/artifactory/repo1/com/example/demo/0.0.1-SNAPSHOT/...
Module ID:	com.example:demo:0.0.1-20250611.073013-1
Deployed By:	ganeshreddy64987610@gmail.com
Size:	19.78 MB
Created:	11-06-25 07:30:14 UTC

Here in this way it was stored base on time and build number it was stored in repo

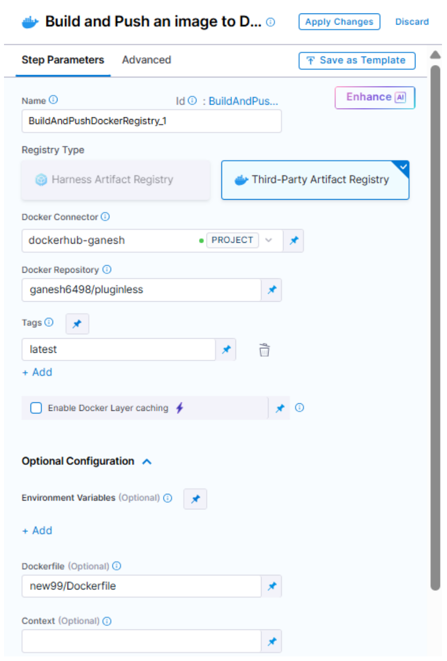
So we use the time and build num



Command

```
pwd
rm -rf $PWD/new99
ls -l
mkdir -p  $PWD/new99
mv  $PWD/Dockerfile $PWD/new99/
cd $PWD/new99

PASS_JFROG='<+secrets.getValue("jfrog-ganesh3")>'
export PASS_JFROG

REPO_BASE_URL="https://trialuu1981.jfrog.io/artifactory/repo3"
GROUP_PATH="com/example/demo"
VERSION="0.0.1-SNAPSHOT"

# Download maven metadata
curl -u "ganeshreddy64987610@gmail.com:${PASS_JFROG}" \
     -s "${REPO_BASE_URL}/${GROUP_PATH}/${VERSION}/maven-metadata.xml" -o maven-metadata.xml

# Extract timestamp and buildNumber
TIMESTAMP=$(xmllint --xpath "string(//snapshot/timestamp)" maven-metadata.xml)

BUILDNUM=$(xmllint --xpath "string(//snapshot/buildNumber)" maven-metadata.xml)

# Construct JAR file name
JAR_FILE="demo-${TIMESTAMP}-${BUILDNUM}.jar"

# Download JAR
curl -u "ganeshreddy64987610@gmail.com:${PASS_JFROG}" \
     -O "${REPO_BASE_URL}/${GROUP_PATH}/${VERSION}/${JAR_FILE}"

ls -l

pwd


ls -l
```

Step 3: run the docker



Build and Push an image to D...

Name: BuildAndPushDockerRegistry_1

Id : BuildAndPus...

Registry Type: Harness Artifact Registry [✓ Third-Party Artifact Registry]

Docker Connector: dockerhub-ganesh PROJECT

Docker Repository: ganesh6498/pluginless

Tags: latest

+ Add

☐ Enable Docker Layer caching

Optional Configuration

Environment Variables (Optional)

+ Add

Dockerfile (Optional): new99/Dockerfile

Context (Optional):

Here we give the docker file path because in previous stage we move the docker file and artifact to new99

Step 3:
Take the vm_ip here we use the docker delegate to connect local system
In that system we also perform the dynamic inventory
So we can if we don't need we can skip the step

```
gcloud compute instances list   --filter="name~sam"   --
format="value(networkInterfaces.accessConfigs.natIP)" | tr -d "[]'"  >
/etc/ansible/hosts
cat /etc/ansible/hosts
```

step 4: vault to take the private key from the vault gcs storage

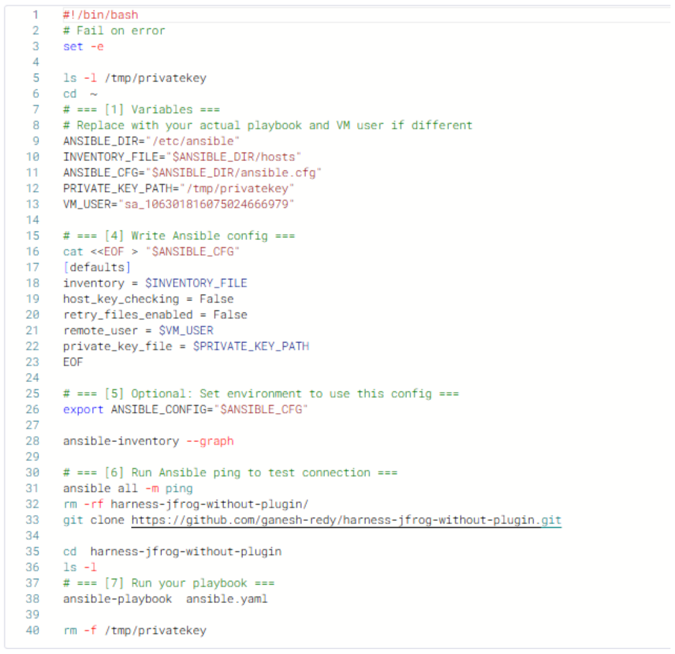
```
echo '<+secrets.getValue("private-key-ganesh")>' > /tmp/privatekey
chmod 600 /tmp/privatekey
ls -l /tmp
```

step 5: run the playbook

```
#!/bin/bash

# Fail on error
set -e

ls -l /tmp/privatekey
cd  ~

# === [1] Variables ===
# Replace with your actual playbook and VM user if different
ANSIBLE_DIR="/etc/ansible"
INVENTORY_FILE="$ANSIBLE_DIR/hosts"
ANSIBLE_CFG="$ANSIBLE_DIR/ansible.cfg"
PRIVATE_KEY_PATH="/tmp/privatekey"
VM_USER="sa_106301816075024666979"



# === [4] Write Ansible config ===
cat <<EOF > "$ANSIBLE_CFG"
[defaults]
inventory = $INVENTORY_FILE
host_key_checking = False
retry_files_enabled = False
remote_user = $VM_USER
```

```
private_key_file = $PRIVATE_KEY_PATH
EOF

# === [5] Optional: Set environment to use this config ===
export ANSIBLE_CONFIG="$ANSIBLE_CFG"


ansible-inventory --graph

# === [6] Run Ansible ping to test connection ===
ansible all -m ping
rm -rf harness-jfrog-without-plugin/
git clone https://github.com/ganesh-redy/harness-jfrog-without-plugin.git

cd  harness-jfrog-without-plugin
ls -l

# === [7] Run your playbook ===
ansible-playbook  ansible.yaml

rm -f /tmp/privatekey
```

Command



```
#!/bin/bash
# Fail on error
set -e

ls -l /tmp/privatekey
cd  ~
# === [1] Variables ===
# Replace with your actual playbook and VM user if different
ANSIBLE_DIR="/etc/ansible"
INVENTORY_FILE="$ANSIBLE_DIR/hosts"
ANSIBLE_CFG="$ANSIBLE_DIR/ansible.cfg"
PRIVATE_KEY_PATH="/tmp/privatekey"
VM_USER="sa_106301816075024666979"

# === [4] Write Ansible config ===
cat <<EOF > "$ANSIBLE_CFG"
[defaults]
inventory = $INVENTORY_FILE
host_key_checking = False
retry_files_enabled = False
remote_user = $VM_USER
private_key_file = $PRIVATE_KEY_PATH
EOF

# === [5] Optional: Set environment to use this config ===
export ANSIBLE_CONFIG="$ANSIBLE_CFG"

ansible-inventory --graph

# === [6] Run Ansible ping to test connection ===
ansible all -m ping
rm -rf harness-jfrog-without-plugin/
git clone https://github.com/ganesh-redy/harness-jfrog-without-plugin.git

cd  harness-jfrog-without-plugin
ls -l
# === [7] Run your playbook ===
ansible-playbook  ansible.yaml

rm -f /tmp/privatekey
```

Ansible dynamic inventory

Pre requirements
First we need install ansible , python gcloud , auth

```
sudo yum install ansible
sudo yum install epel-release
yum install -y python3-pip
pip install requests google-auth
```

step :

ansible –config init –disable > /etc/ansible/ansible.cfg

or

we can directly use that file with out disable

step 2:

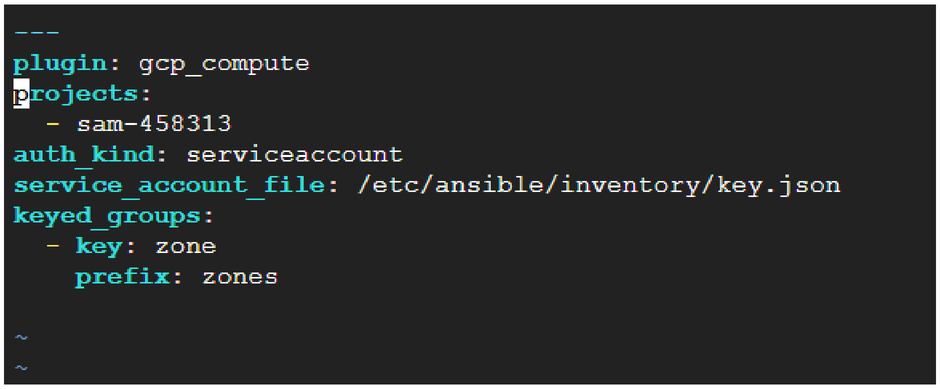
create on dir in /etc/ansible/

mkdir inventory

cd inventory

vim gcp.json or sam.gcp.json

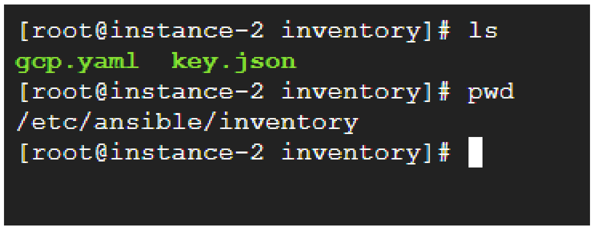
```
plugin: gcp_compute

projects:

    -    project_ip_gcp

auth_kind: serviceaccount
service_account_file: /etc/ansible/inventory/key.json

keyed_groups:

    key: zone

    prefix: sam
```



```
---
plugin: gcp_compute
projects:
  - sam-458313
auth_kind: serviceaccount
service_account_file: /etc/ansible/inventory/key.json
keyed_groups:
  - key: zone
    prefix: zones

~
~
```

Step 3:

Download the gcp service account jey an past the location



```
[root@instance-2 inventory]# ls
gcp.yaml  key.json
[root@instance-2 inventory]# pwd
/etc/ansible/inventory
[root@instance-2 inventory]#
```

Step 4: Go to ansible.cfg, then

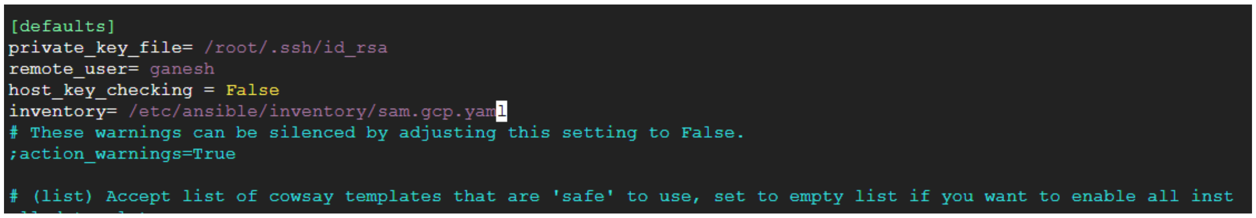
Enable_plugin = gcp_compute

Remote_user = ansible (if you create the ssh key with same ansible comment name the u can give here this name)

Host_key_checking =False

Become = true

Private_key_file = /root/.ssh/id_rsa

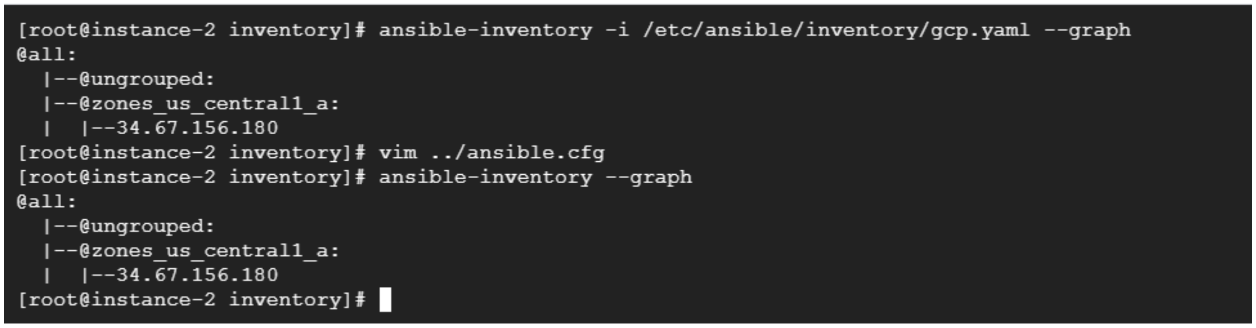


```
[defaults]
private_key_file= /root/.ssh/id_rsa
remote_user= ganesh
host_key_checking = False
inventory= /etc/ansible/inventory/sam.gcp.yaml
# These warnings can be silenced by adjusting this setting to False.
;action_warnings=True

# (list) Accept list of cowsay templates that are 'safe' to use, set to empty list if you want to enable all inst
```

Step 5:

Check the ip fetching or not

Ansible-inventory --graph



```
[root@instance-2 inventory]# ansible-inventory -i /etc/ansible/inventory/gcp.yaml --graph
@all:
  |--@ungrouped:
  |--@zones_us_central1_a:
  |  |--34.67.156.180
[root@instance-2 inventory]# vim ../ansible.cfg
[root@instance-2 inventory]# ansible-inventory --graph
@all:
  |--@ungrouped:
  |--@zones_us_central1_a:
  |  |--34.67.156.180
[root@instance-2 inventory]#
```