**Project Objectives:**

**Goal**: To integrate **Terraform** and **Ansible** over a **Jenkins** (CI/CD) declarative pipeline to instantiate an **AWS EC2** instance, install **Docker** & **Docker-compose**

to build and run the Application/APIs (**Spring Pet Clinic**) on the provisioned instance. **Git** tool will be used to fetch the application delivered on a project repository maintained in

**GitHub**

*Use DevOps tools*

* Docker (Containerization tool - that is it will containerize the application as an image into a container)
* Jenkins ( Integration tool to provide the CI/CD pipeline)
* Ansible (Configuration Management tool) -> it will ssh (login) into the newly created instance and install Docker, Docker-Compose
* Terraform (Provisioning tool) -> simply create an AWS EC2 instance, with allied resources (VPC, Sybnets, Routing Tables, I-G/w)
* Git
* AWS Cli

*What are we going to do stepwise :*

* Create an AWS EC2 instance as a **Master server** (Mumbai - ***ap-south-1***) and install **Jenkins**, **Ansible**, **Terraform**, **Git**, **AWS-client**, **Java,** **tree -** Manual work
* Extract/Clone the source-code from ***Github repository*** of the project into the AWS instance created above/Windows/Mac
* Push the repository into your individual Git hub account
* Create **Terraform** provisioning files/scripts to provision an EC2 instance with VPC, Nat, Subnets, Route Tables, Internet G/W resources into the Singapore region
* Create **Ansible** playbook and other configuration files to install the application available from Git hub as **Docker** images and run the containers with **Docker-compose**
* **Create Jenkins CI/CD Pipeline declarative script to integrate Terraform and Ansible, to enable an automated pipeline job to achieve the goal of the project**
* Verify the application in the backend of newly created instance as well as call the REST APIs as URLs on the browser [Customer UAT -> user acceptance testing or Client presentation ]

An overall Project definition – refer the embedded document



**Installation & configuration of the S/Ws on the Master Server in Mumbai (ap-south-1)/N-Virginia:**

**Jenkins Installation**

$> sudo adduser jenkins

$> sudo usermod -aG sudo jenkins

$> sudo su -l ubuntu

[Please execute this step and do not skip]

$> sudo apt update

$> sudo apt install **openjdk-17-jdk** -y

[ Please install Java 17 instead of Java 11 - this is required for the new version of Spring Pet Clinic application ]

$> curl -fsSL <https://pkg.jenkins.io/debian/jenkins.io-2023.key> | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

$> echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

$> sudo apt update

$> sudo apt install jenkins -y

$> sudo systemctl status jenkins

**Ansible Installation**

$ sudo useradd -m -s /bin/bash ansible

$ sudo passwd ansible

$ sudo su -

$ echo -e 'ansible\tALL=(ALL)\tNOPASSWD:\tALL' > /etc/sudoers.d/ansible

$ exit

$ sudo apt update

$ sudo apt install software-properties-common

$ sudo apt-add-repository --yes --update ppa:ansible/ansible

$ sudo apt install ansible -y

$ ansible --version

**Terraform Installation**

$ sudo apt update

$ sudo apt install -y gnupg software-properties-common curl

$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

$ sudo apt update

$ sudo apt install terraform -y

$ terraform -help plan

$ which terraform

$ terraform --version

**AWS-client Installation**

$ sudo curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

$ sudo apt install unzip -y

$ unzip awscliv2.zip

$ sudo ./aws/install

$ aws --version

**tree Installation**

$ sudo apt update

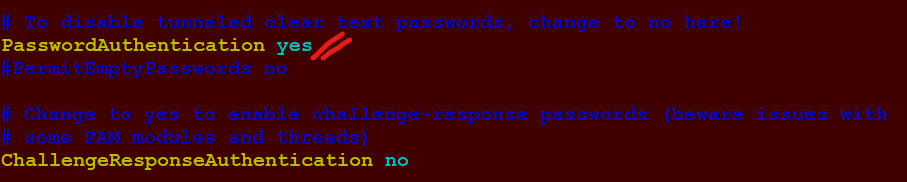
$ sudo apt install tree -y

**Key-Pair in ap-southeast-1 region**

* Create a new **key-pair** in AWS management console for the region Singapore (ap-southeast-1)
* Name the generated .pem file as **spring-petclinic.pem** and download in a Windows/Mac filesystem
* On the Master instance with Jenkins user created do the following steps:
  + /home/ubuntu > su -l jenkins
  + Password: jenkins
  + /home/jenkins > ssh-keygen

This command “ssh-keygen” will create the .ssh folder under the home folder of the user Jenkins, as well as create 2 key files -> id\_rsa & id\_rsa.pub

* + /home/jenkins > cd .ssh
  + Check id\_rsa files created
  + /home/jenkins > exit
  + /home/ubuntu > sudo vi /etc/ssh/sshd\_config
  + Make the passwordauthentication to ‘yes’ from ‘no’



* + /home/ubuntu > **sudo systemctl restart sshd**

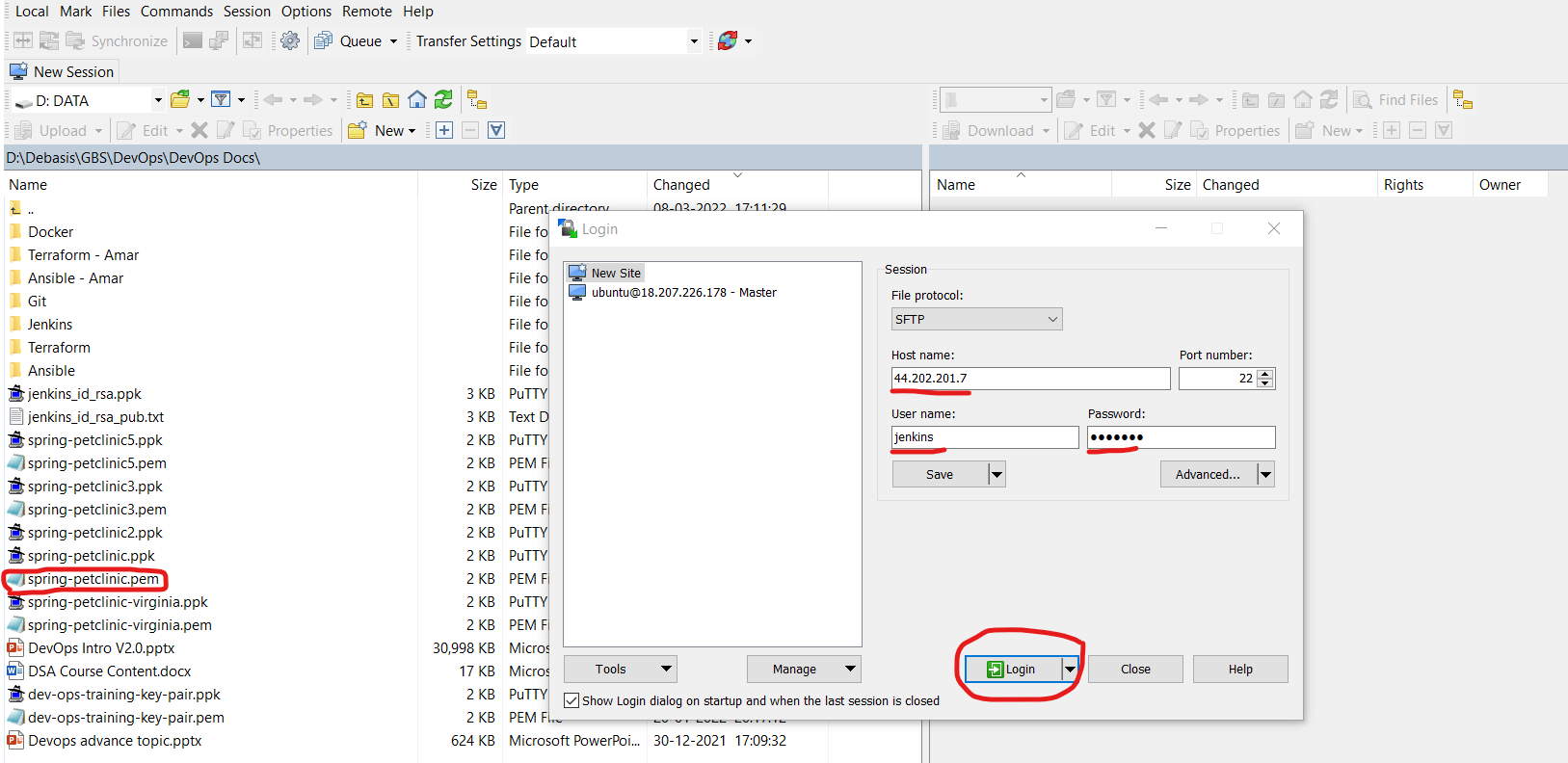
This command will register the changes done in the sshd\_config file

**On windows using pscp**

* + Go to the folder c:\aws\aws-key-files
  + invoke Gitbash
  + $> pscp -pw jenkins spring-petclinic.pem jenkins@3.110.217.245:/home/jenkins/.ssh

**Alternative using WinScp**

* + Use **WinScp** tool to transfer the spring-petclinic.pem to /home/jenkins of the master server/instance

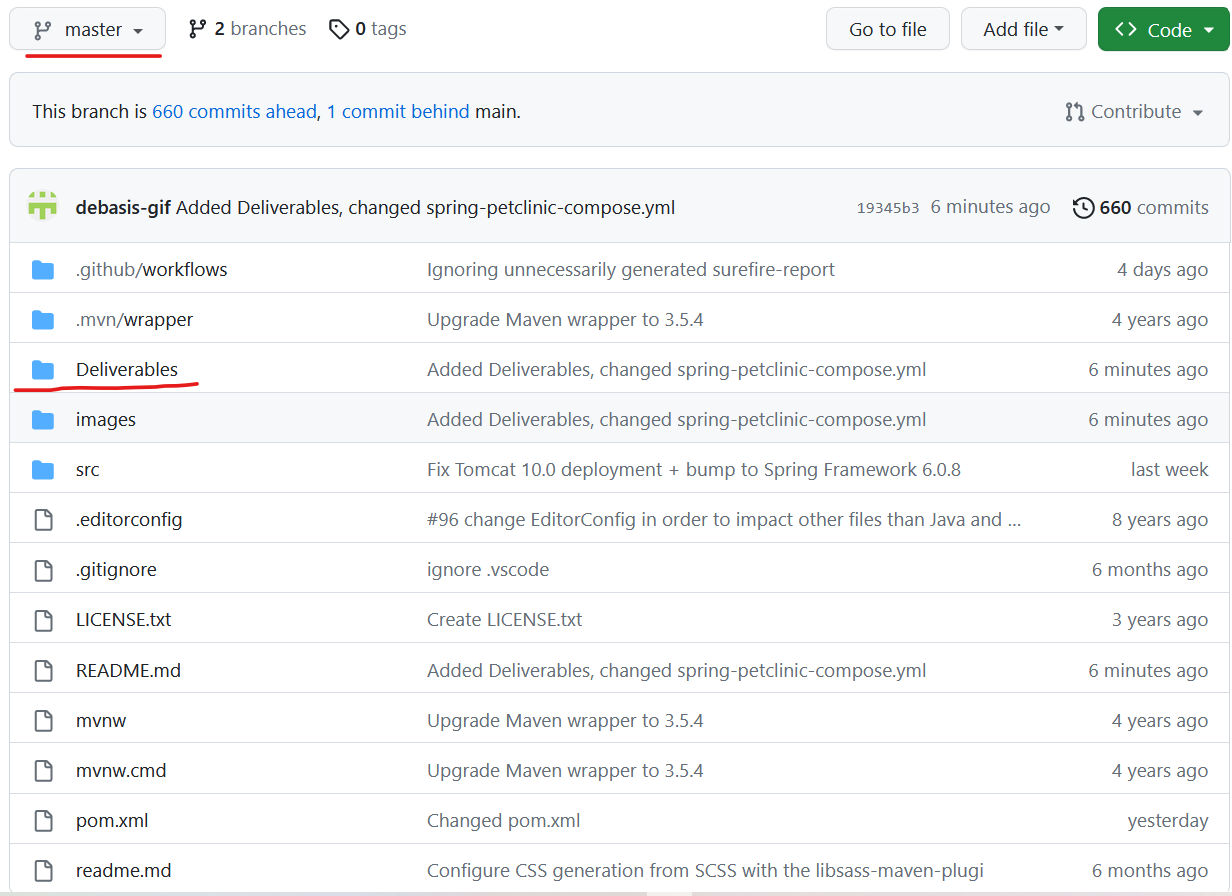


* + /home/ubuntu > su -l jenkins
  + Password: jenkins
  + /home/jenkins > chmod 0400 spring-petclinic.pem
  + /home/jenkins > mv spring-petclinic.pem .ssh/. [ In case the file transfer was done with WINSCP ]
  + /home/jenkins > exit
  + /home/ubuntu > sudo vi /etc/ssh/sshd\_config
  + Make the passwordauthentication back to ‘no’ from ‘yes’
  + /home/ubuntu > sudo systemctl restart sshd

As mentioned above, this command will register the changes done in the ***sshd\_config*** file

**Git Repo of our project:**

<https://github.com/debasis-gif/spring-framework-petclinic.git>



**Git commands to Clone the repository and push to your own GitHub Repository**

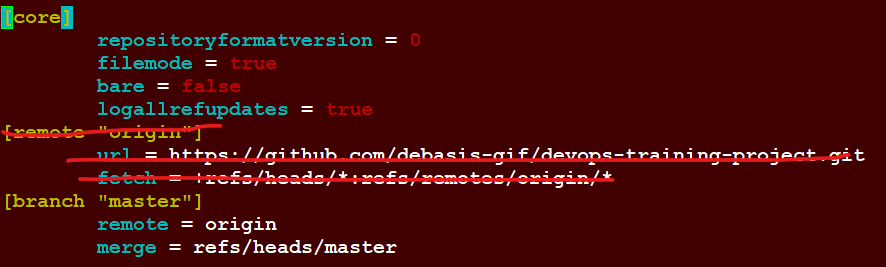
$> /home/ubuntu > git clone --branch master <https://github.com/debasis-gif/spring-framework-petclinic.git>

$> cd spring-framework-petclinic

$> /home/ubuntu/spring-framework-petclinic > cd .git

$> /home/ubuntu/spring-framework-petclinic/.git > vi config

Remove the 3 lines as shown below:



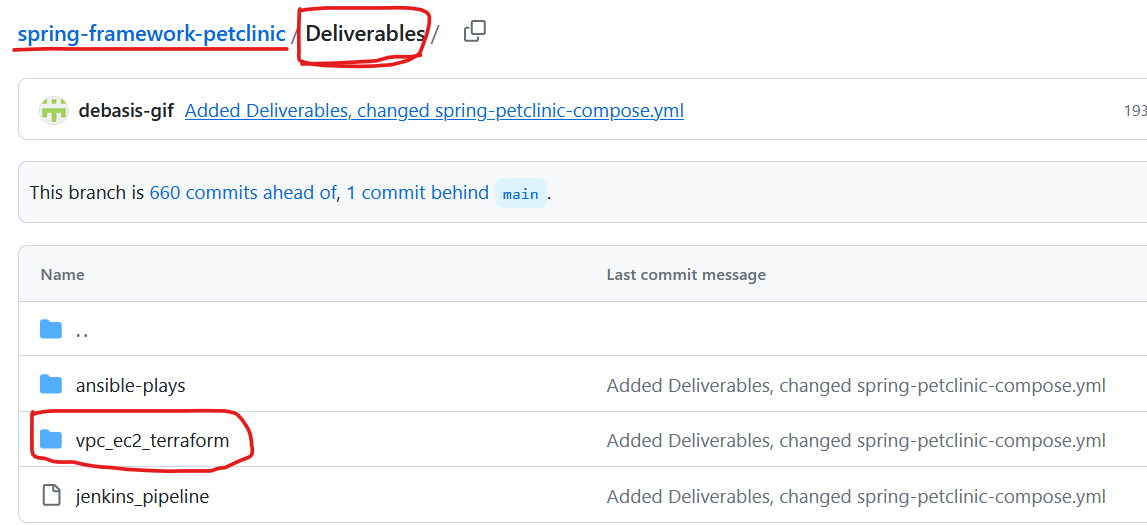
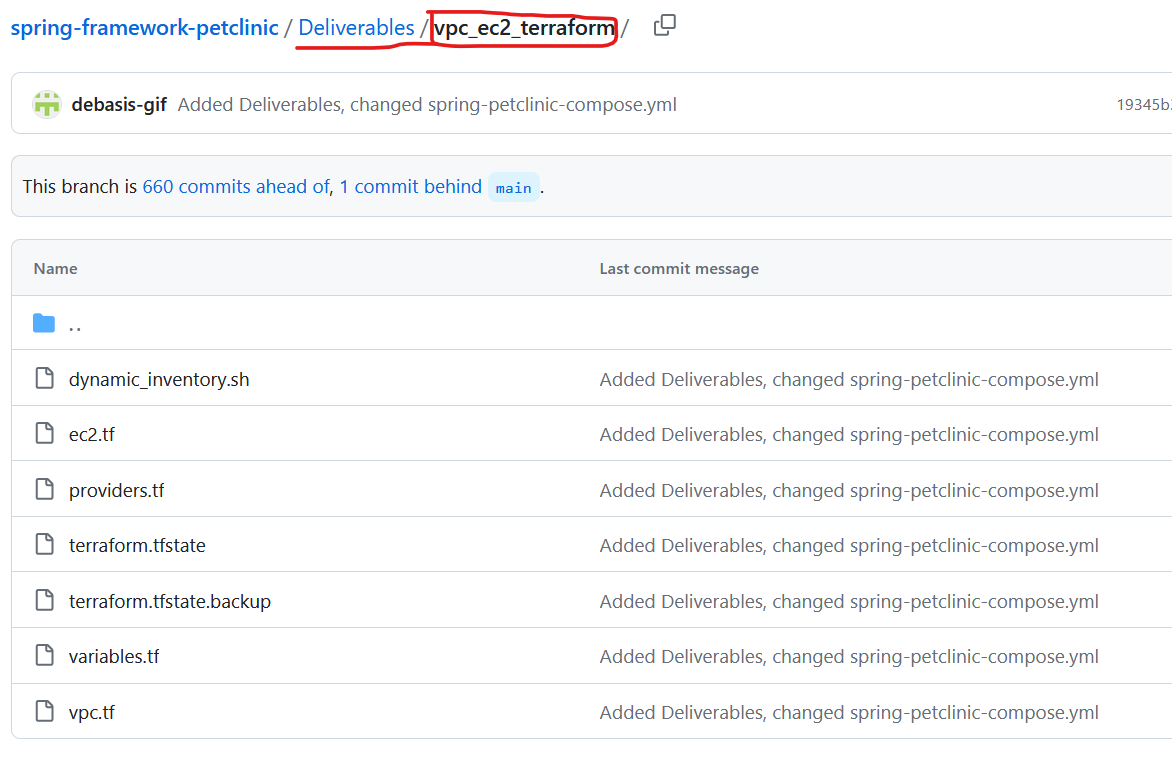
$ /home/ubuntu/spring-framework-petclinic/.git > cd ..

$ /home/ubuntu/spring-framework-petclinic > git remote add originhttps://github.com/<account-name>/<repository>

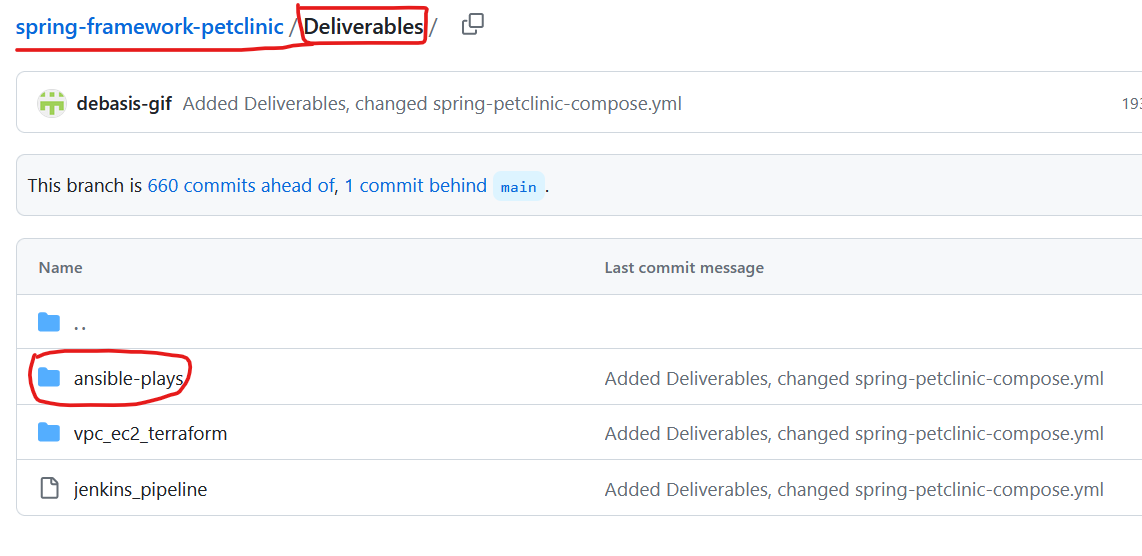
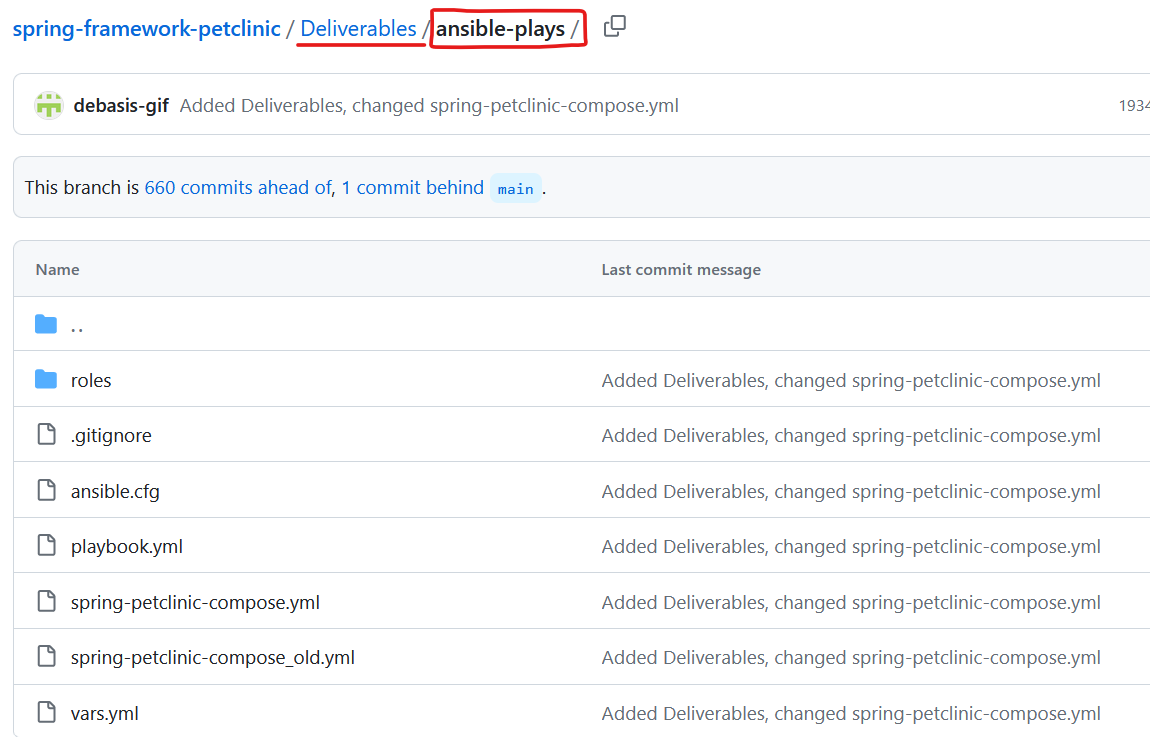
***It is suggested that you create a new repository for this project***

$ /home/ubuntu/spring-framework-petclinic > git push origin master

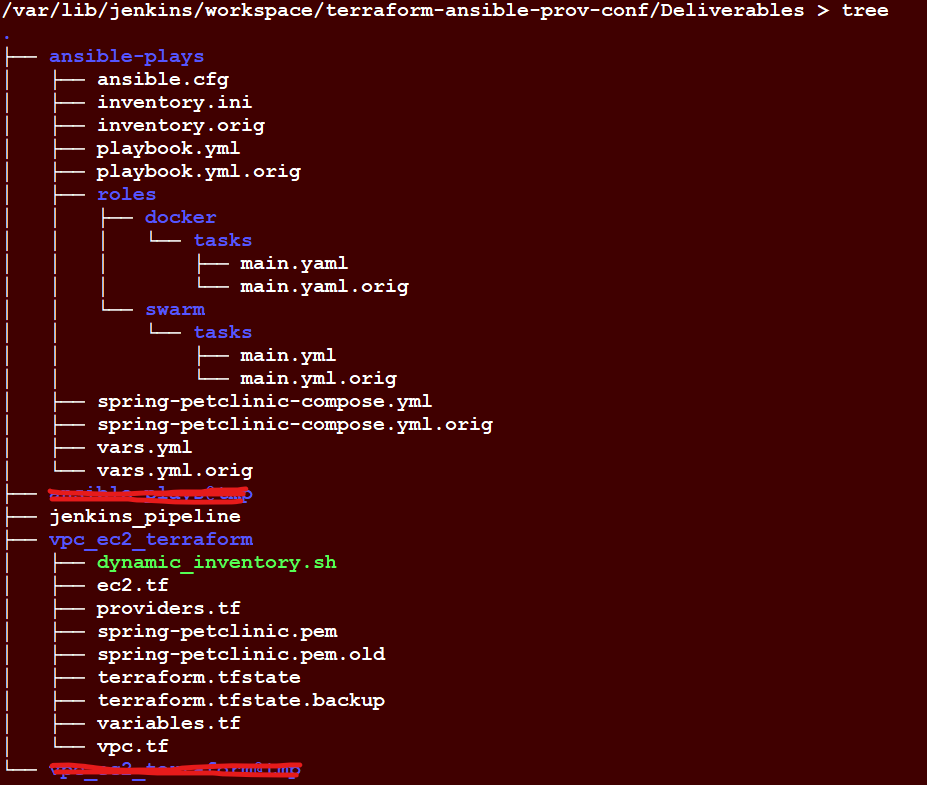
**TerraForm Scripts location:**

**Ansible Scripts location:**

**Tree View of file structure**



----------------------------------------------------------

**Creating an AWS Secret Access Credentials and using them to extract the public IP address of the instance created on the run-time:**

This is to allow **AWS-CLI** backend commands to programmatically extract the ***public IP address*** of the newly provisioned Instance by the Terraform tool

* Create AWS credentials => **Access Key ID** & **Secret Access Key** for your AWS account. This key is valid globally for all the AWS Regions and AZs for your AWS account
* To run the aws cli commands for getting the EC2 instance details, issue the following command for configuring the credentials for AWS
* $> ***aws configure***
* Extract the Public IP address from the created instance from Terraform File run using asw-cli commands as below :-
  + Step 1: $ aws ec2 describe-instances --filters Name=tag:Name,Values=**tf-instance** --region ap-southeast-1
  + Step 2: Extract a level deep :

$ aws ec2 describe-instances --filters Name=tag:Name,Values=**tf-instance** --region ap-southeast-1 | grep "PublicIpAddress" | awk '{print $2}' | sed 's/"//g'

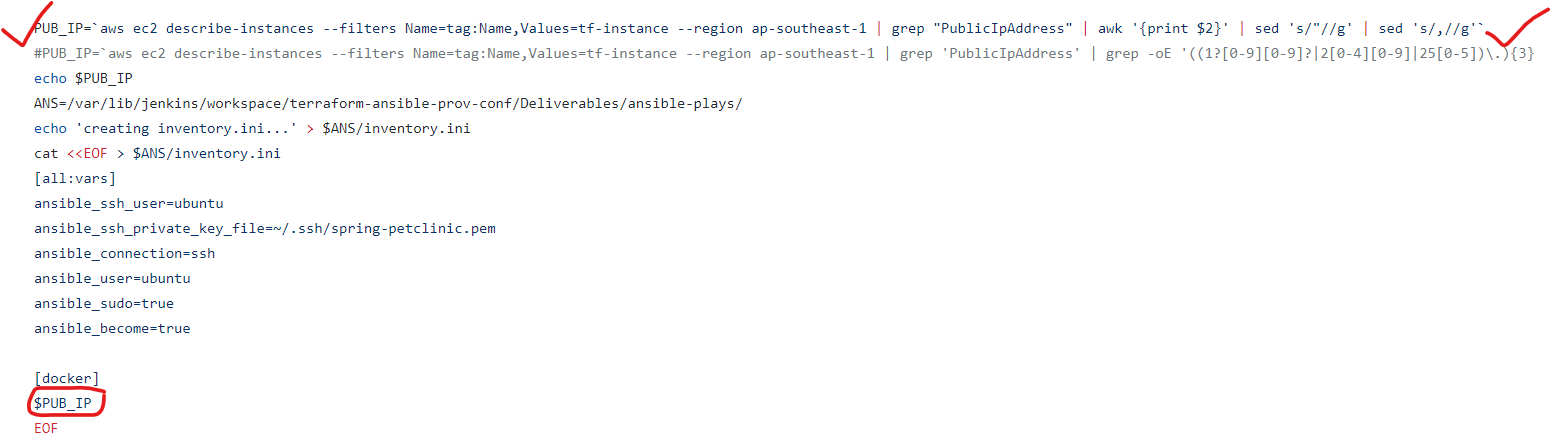
* + Step 3: Extract a further level deep [Final shape of the *aws command* which will be placed in the *dynamic\_inventory.sh*] :

$ aws ec2 describe-instances --filters Name=tag:Name,Values=**tf-instance** --region ap-southeast-1 | grep "PublicIpAddress" | awk '{print $2}' | sed 's/"//g' | sed 's/,//g'

***Note***: we will use the **above aws command** when creating the “inventory.ini” *dynamically* at the point when the new AWS instance **tf-instance** is created by Terraform files in the region **ap-southeast-1**

This aws-cli command syntax is applied in the shell script “**dynamic\_inventory.sh**” under the folder vpc\_ec2\_terraform [as the above tree view]

Here is a screen shot of the shell script (“dynamic\_inventory.sh”) :

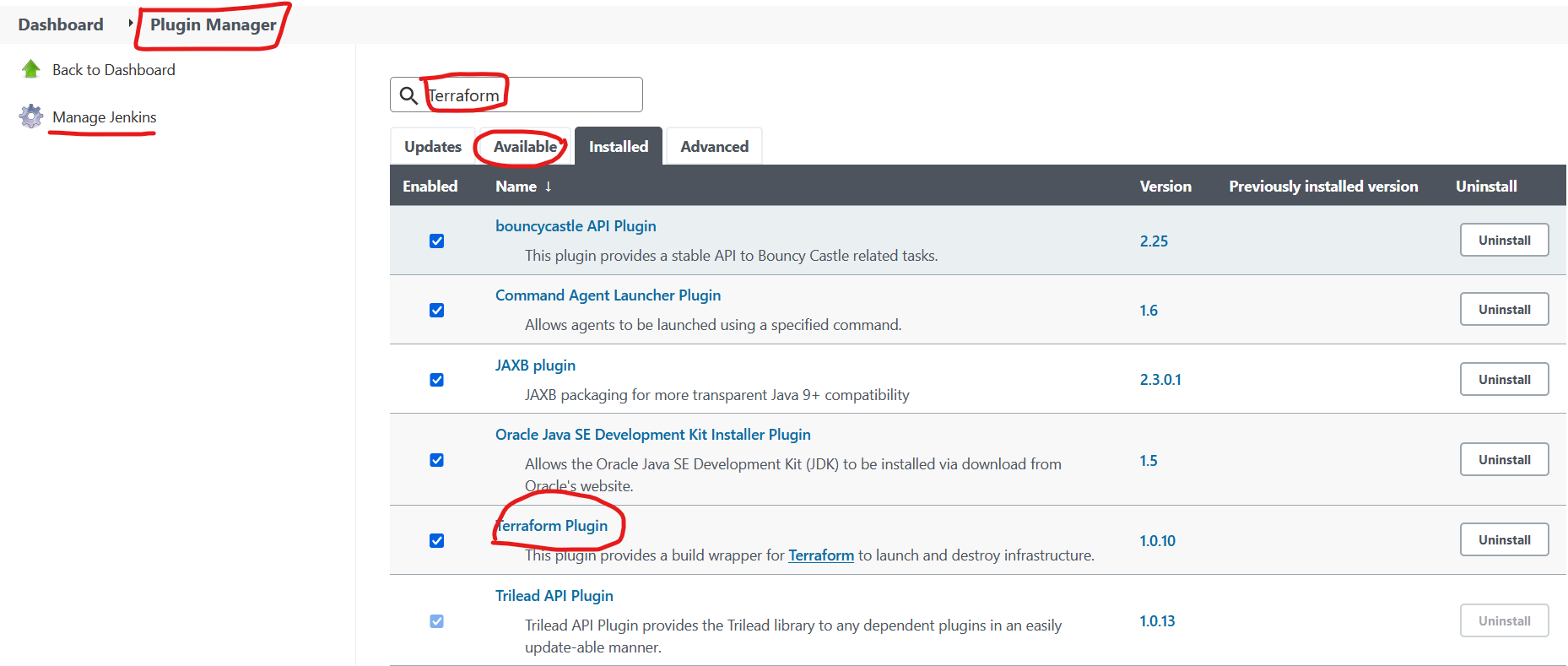


**Jenkins Setup for building & running the Pipeline:**

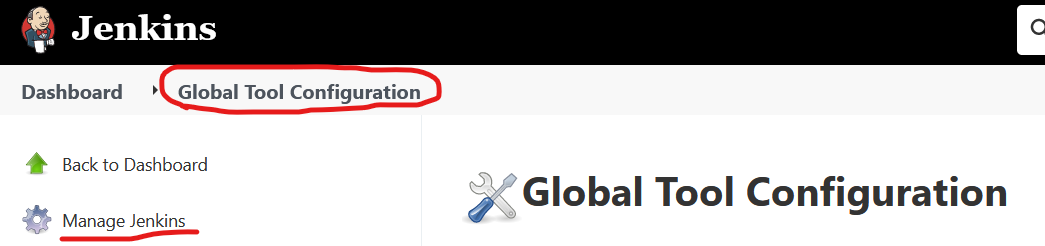
**Plugins for Terraform & Ansible:**

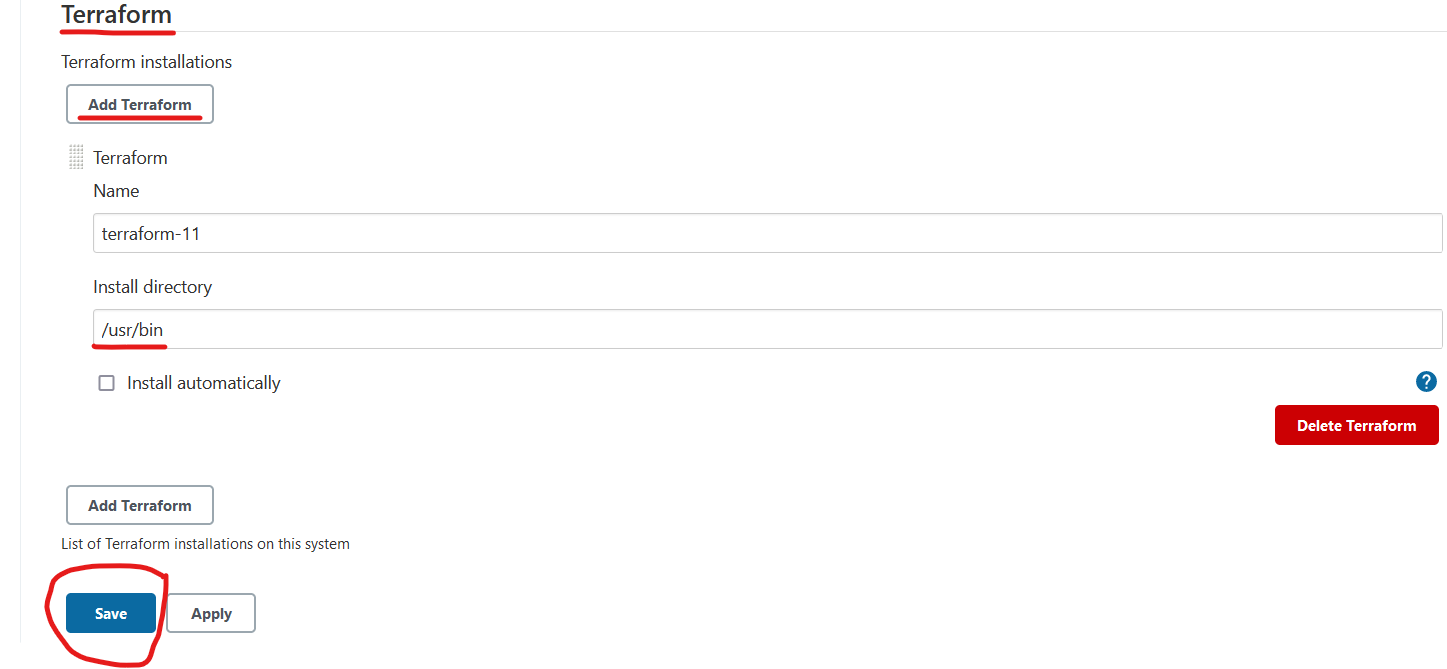
Jenkins Dashboard -> Manage Jenkins -> **Manage Plugins**

**Terraform Plugin**



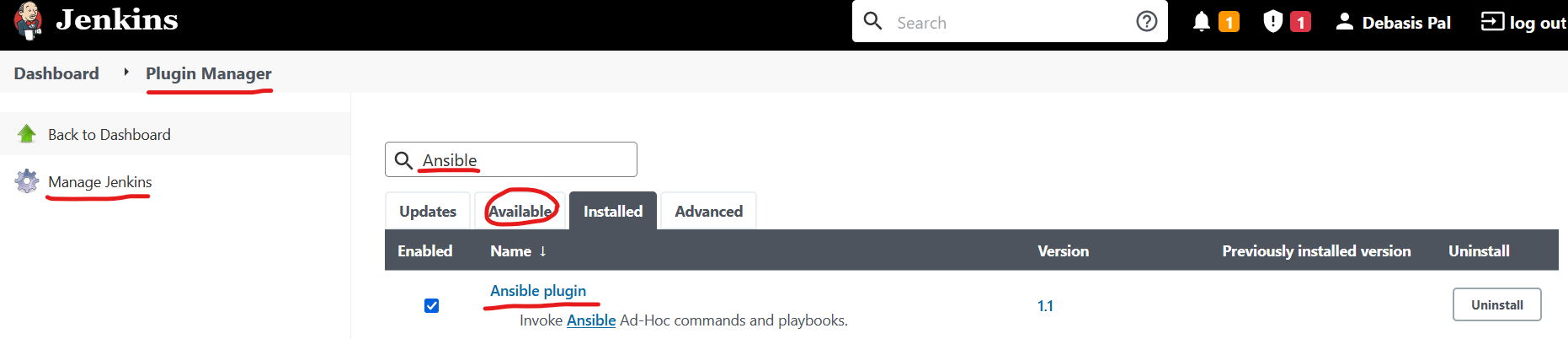
Jenkins Dashboard -> Manage Jenkins -> **Global Tool Configuration - Terraform**



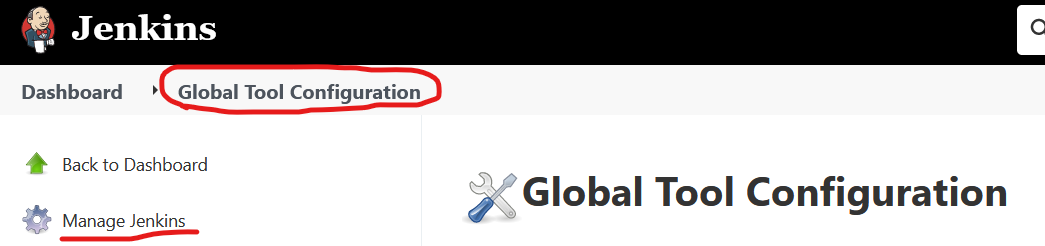


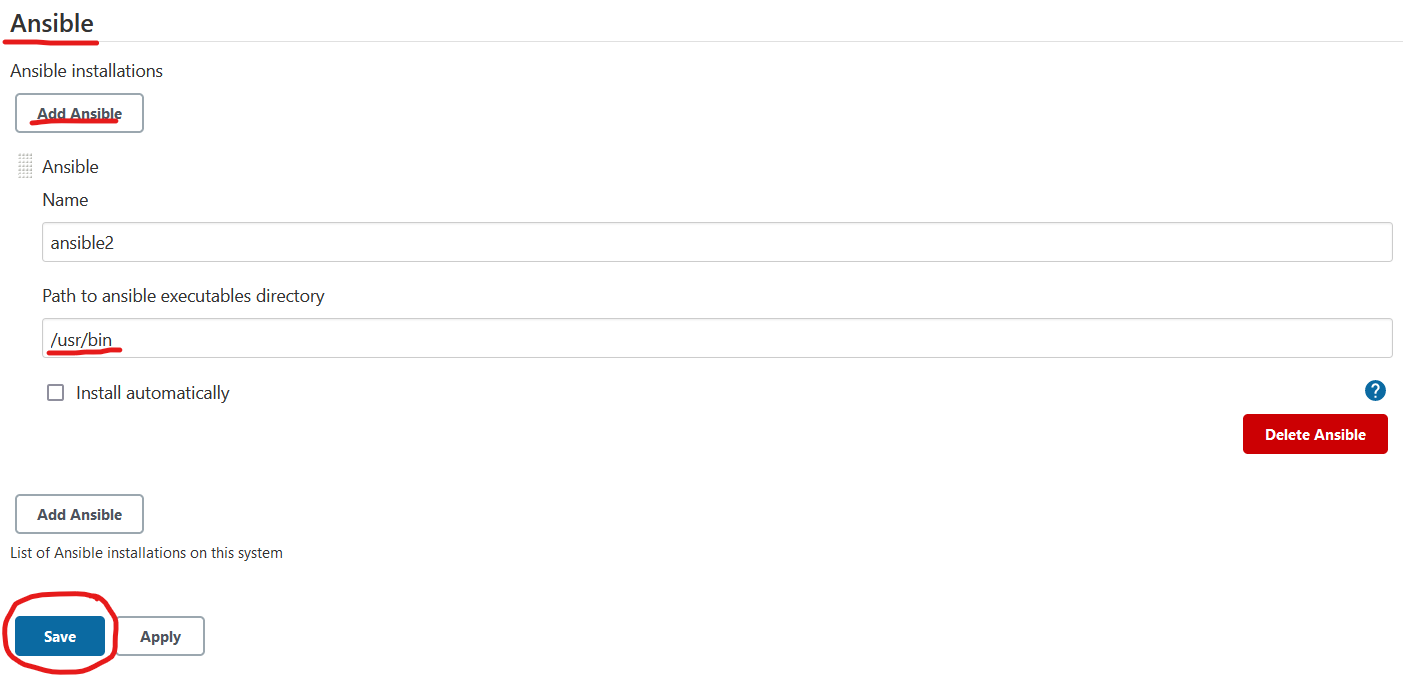
Jenkins Dashboard -> Manage Jenkins -> **Manage Plugins**

**Ansible Plugin**

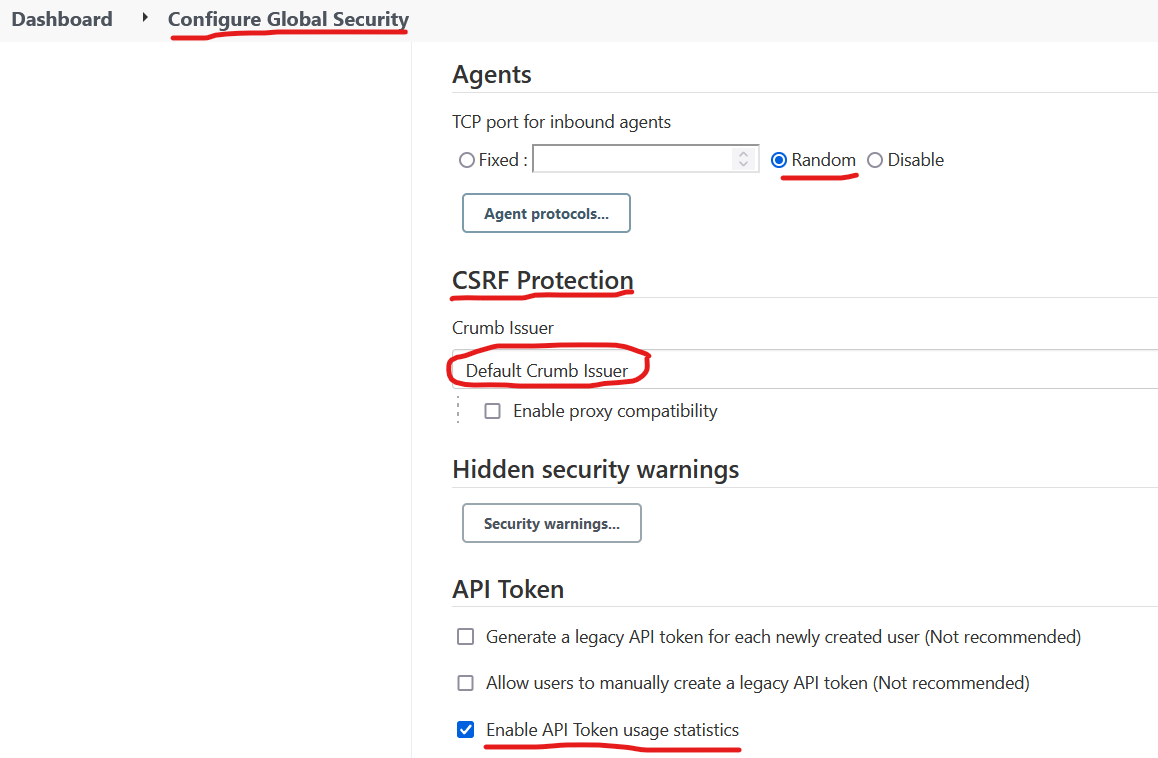


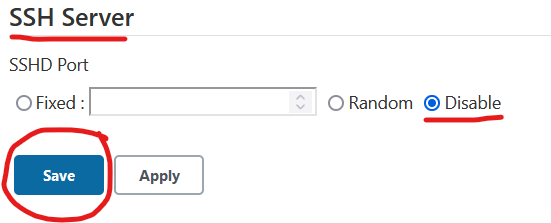
Jenkins Dashboard -> Manage Jenkins -> **Global Tool Configuration - Ansible**





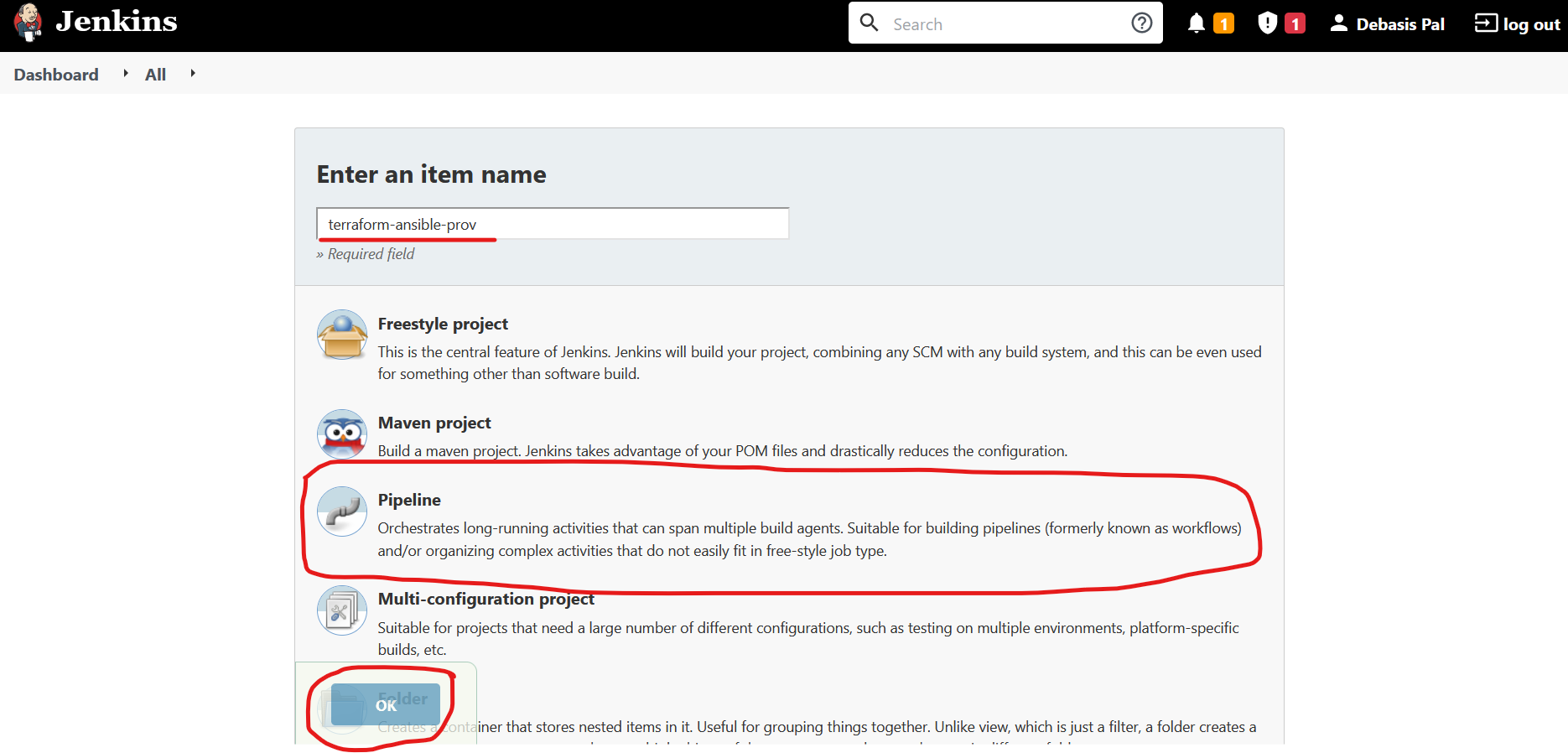
Jenkins Dashboard -> Manage Jenkins -> **Configure Global Security**



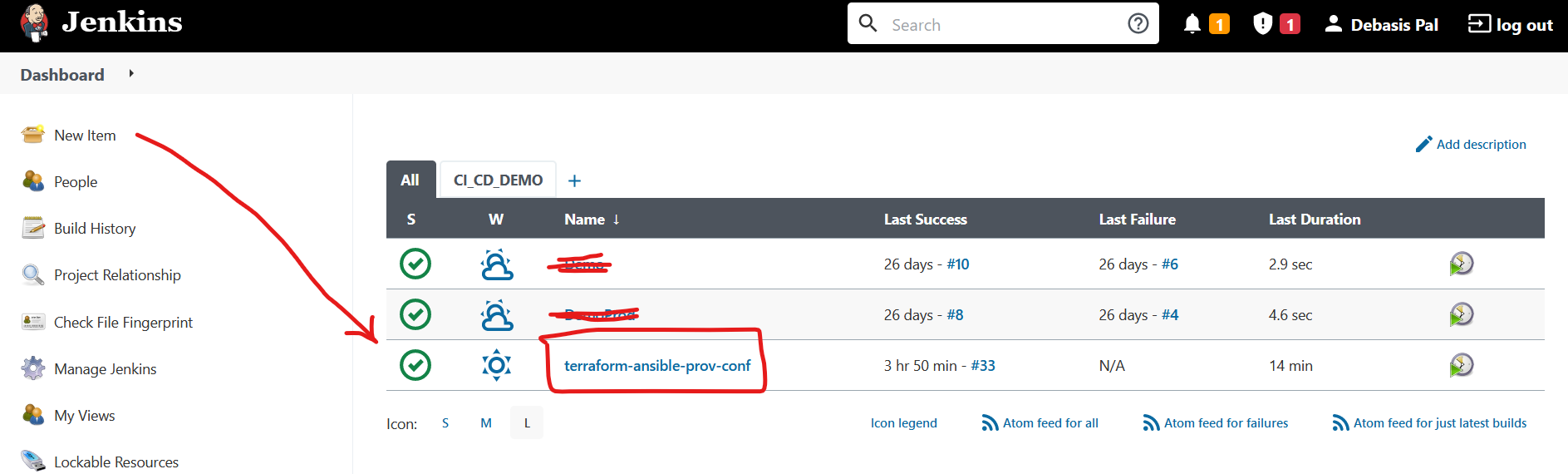


**Create a Jenkins Pipeline Job**

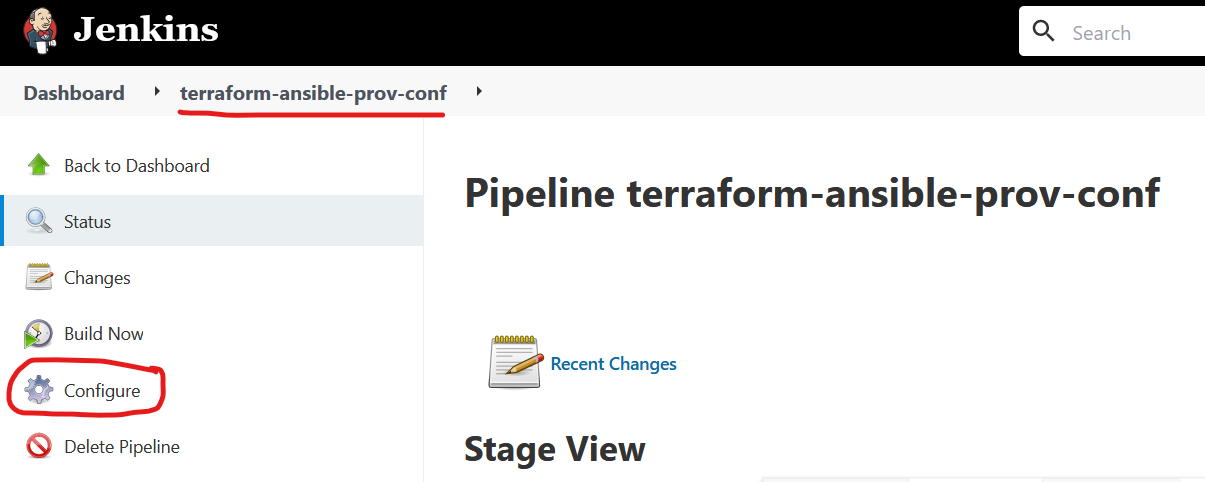
Jenkins Dashboard -> New Item



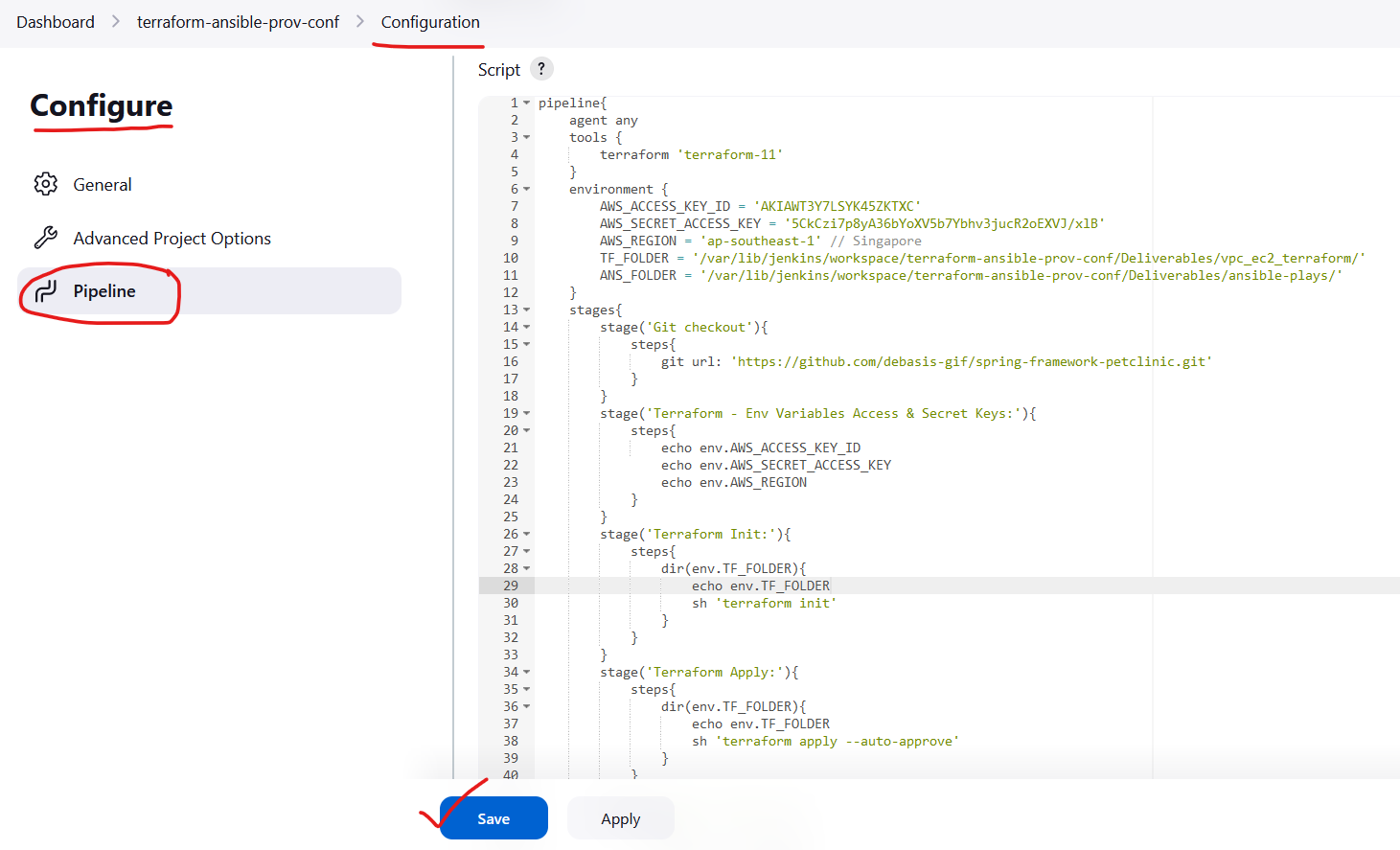
**Note:** Please create the Jenkins Pipeline Job by the name “**terraform-ansible-prov-conf**” as this job folder is being used in the pipeline declarative script



Jenkins Dashboard -> terraform-ansible-prov-conf



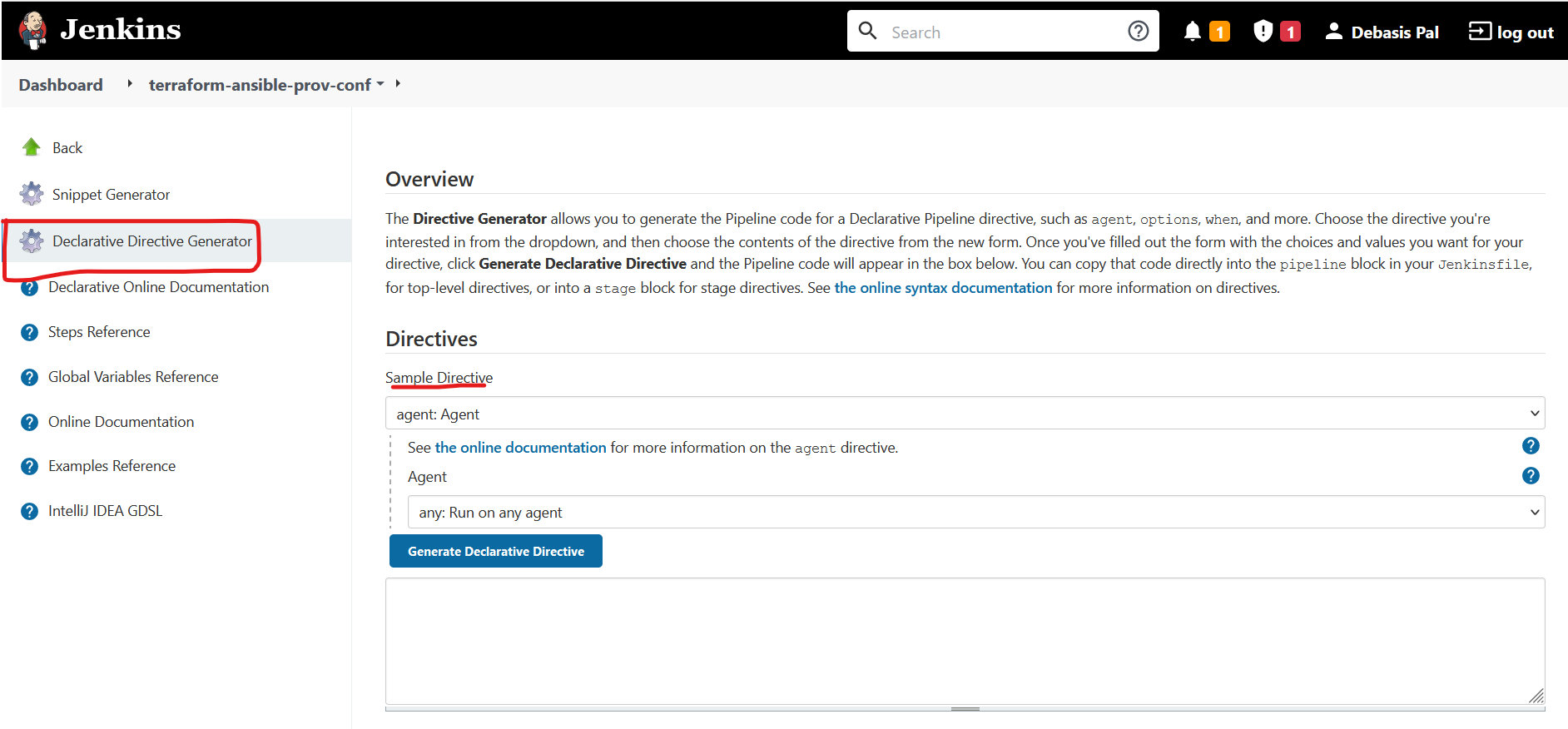
Jenkins Dashboard -> terraform-ansible-prov-conf -> Configure

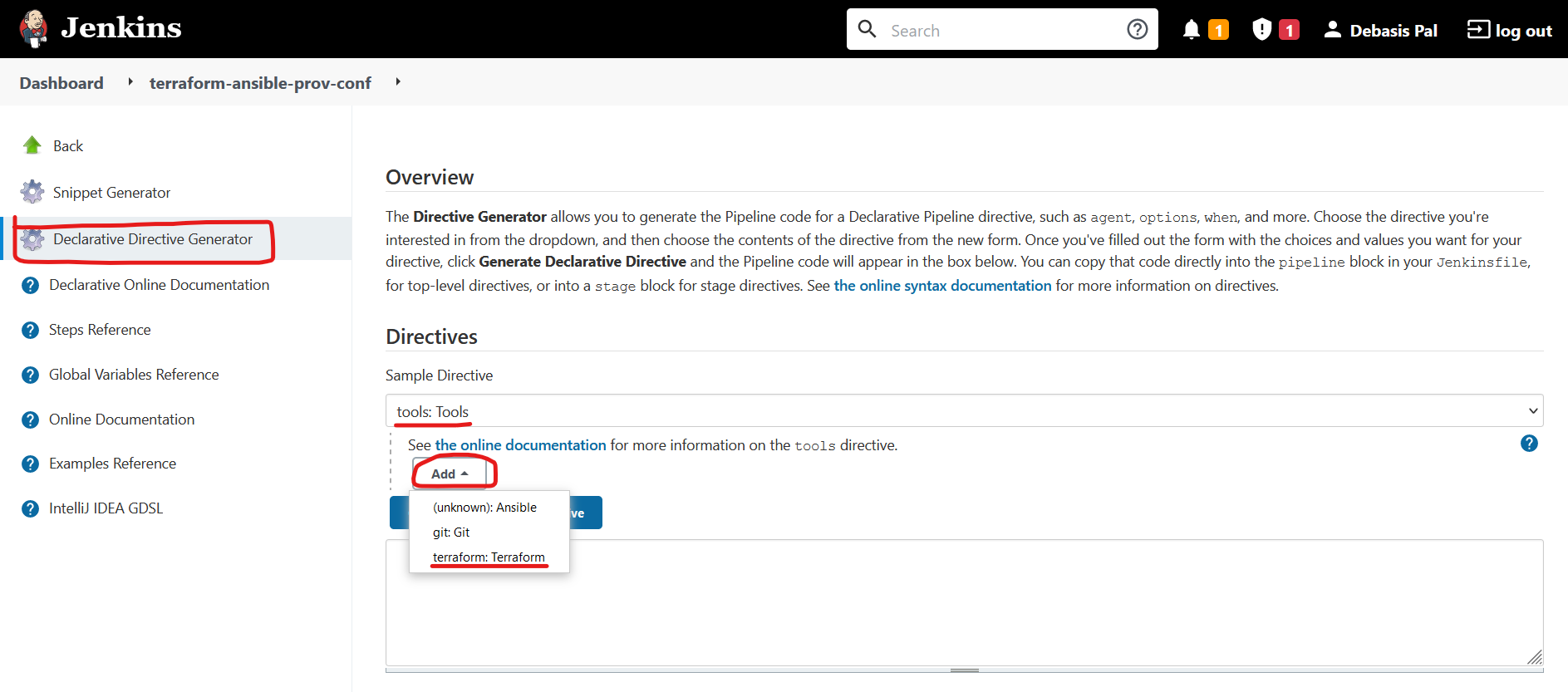


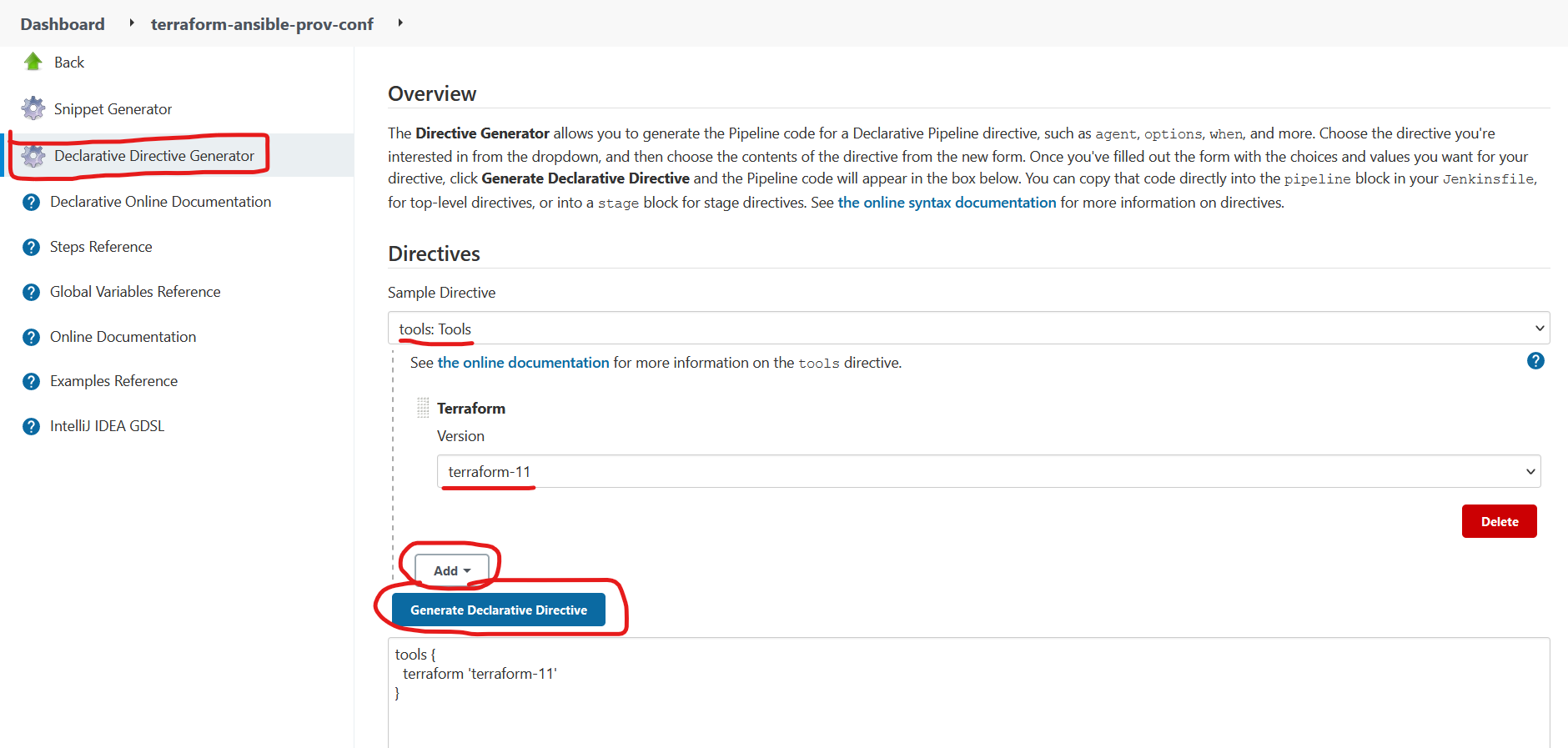
Pipeline Script as below:

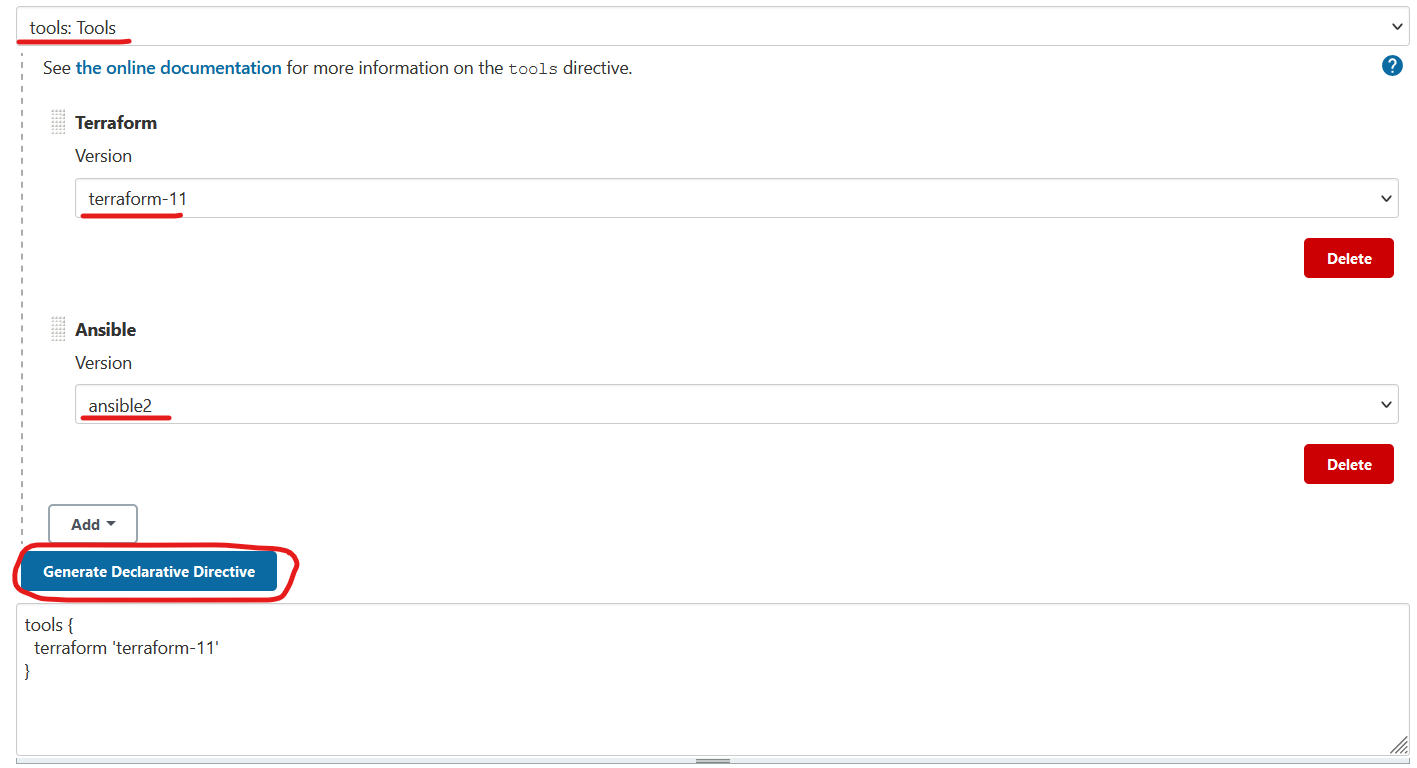


**Using Pipeline Syntax – Declarative Directive Generator**







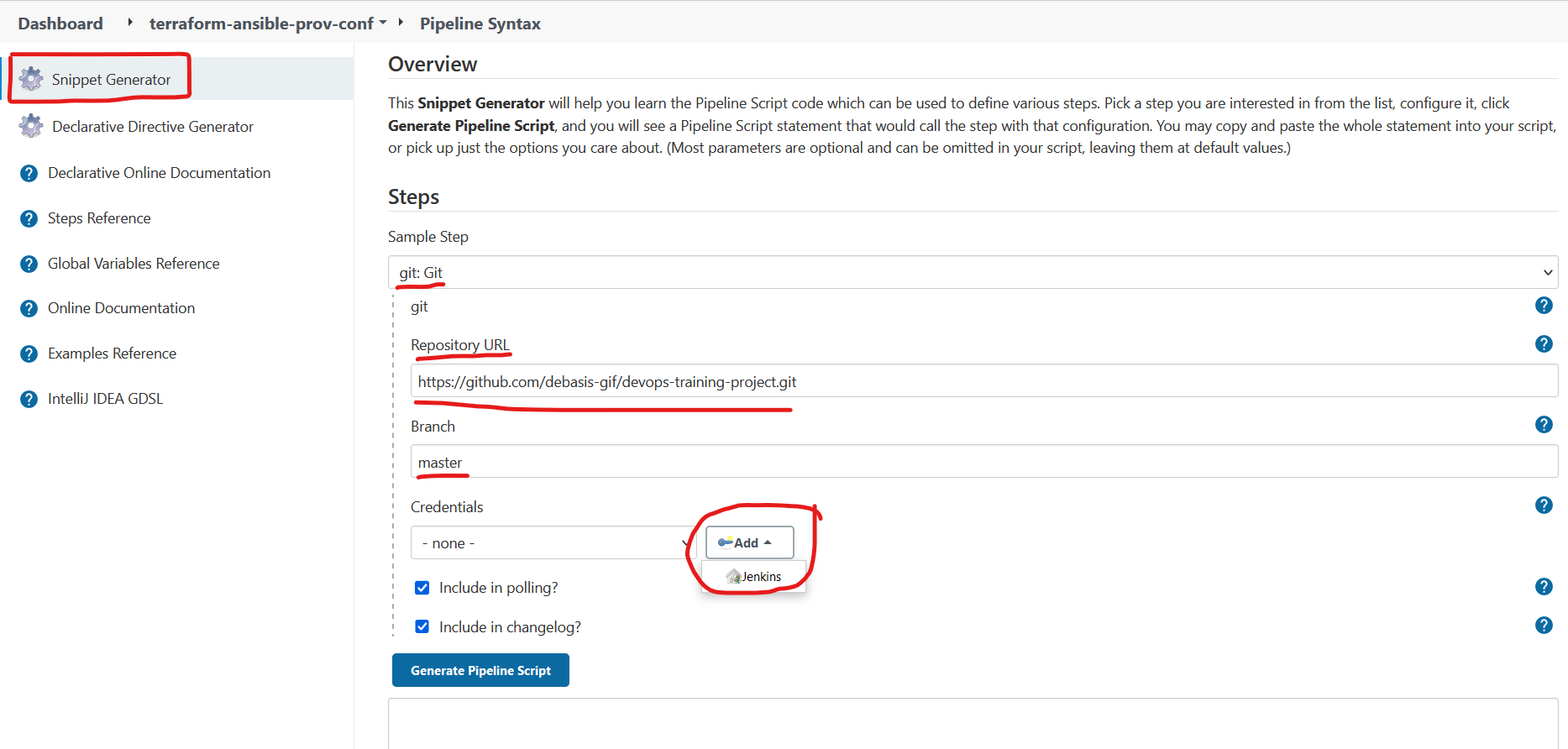


**Note**: there is no declarative Directive for the Ansible Tool

Copy the Declarative Directive from the text box and paste in the pipeline script

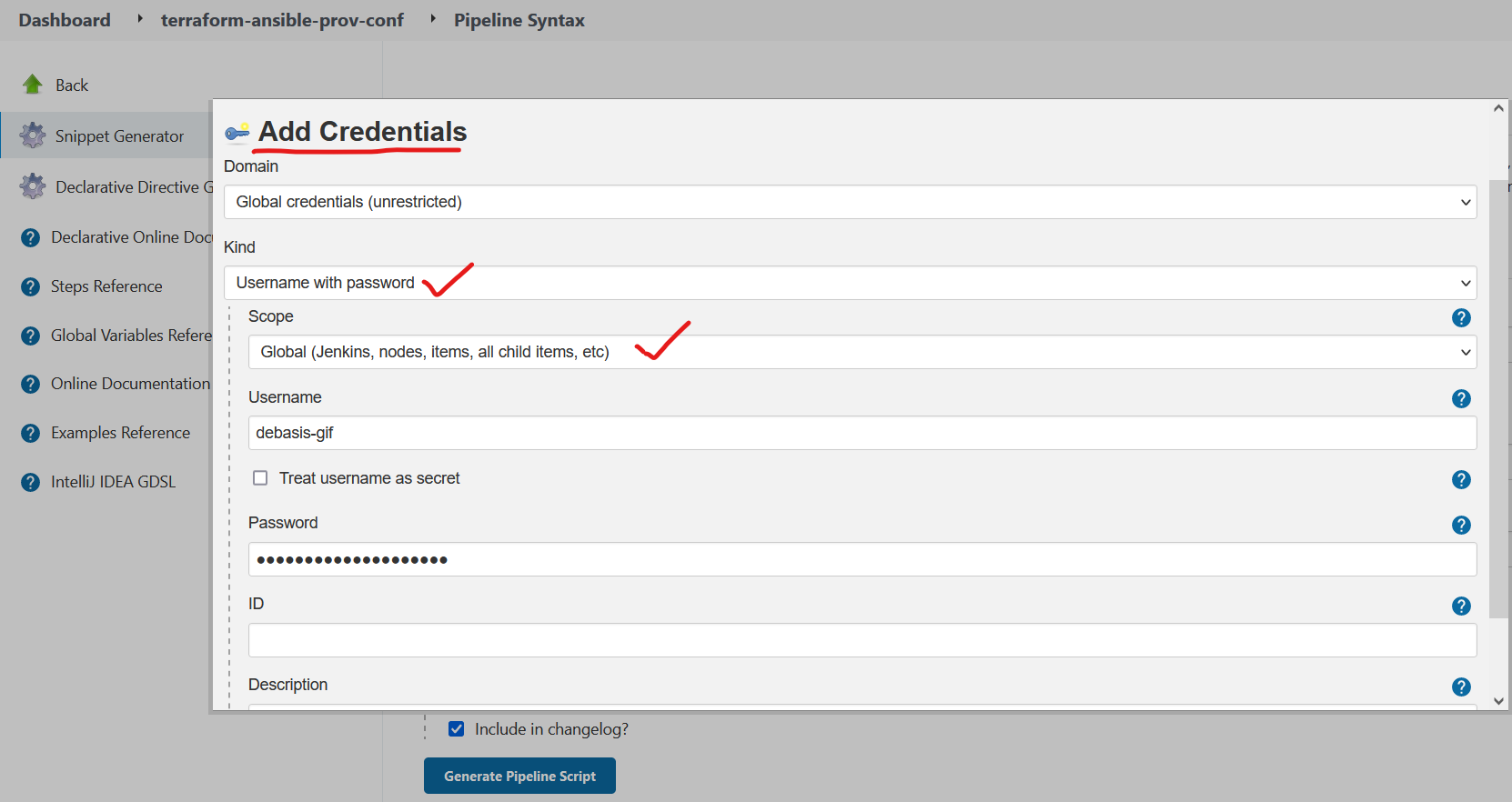
**NOT NEEDED**

**Using Pipeline Syntax – Snippet Generator**



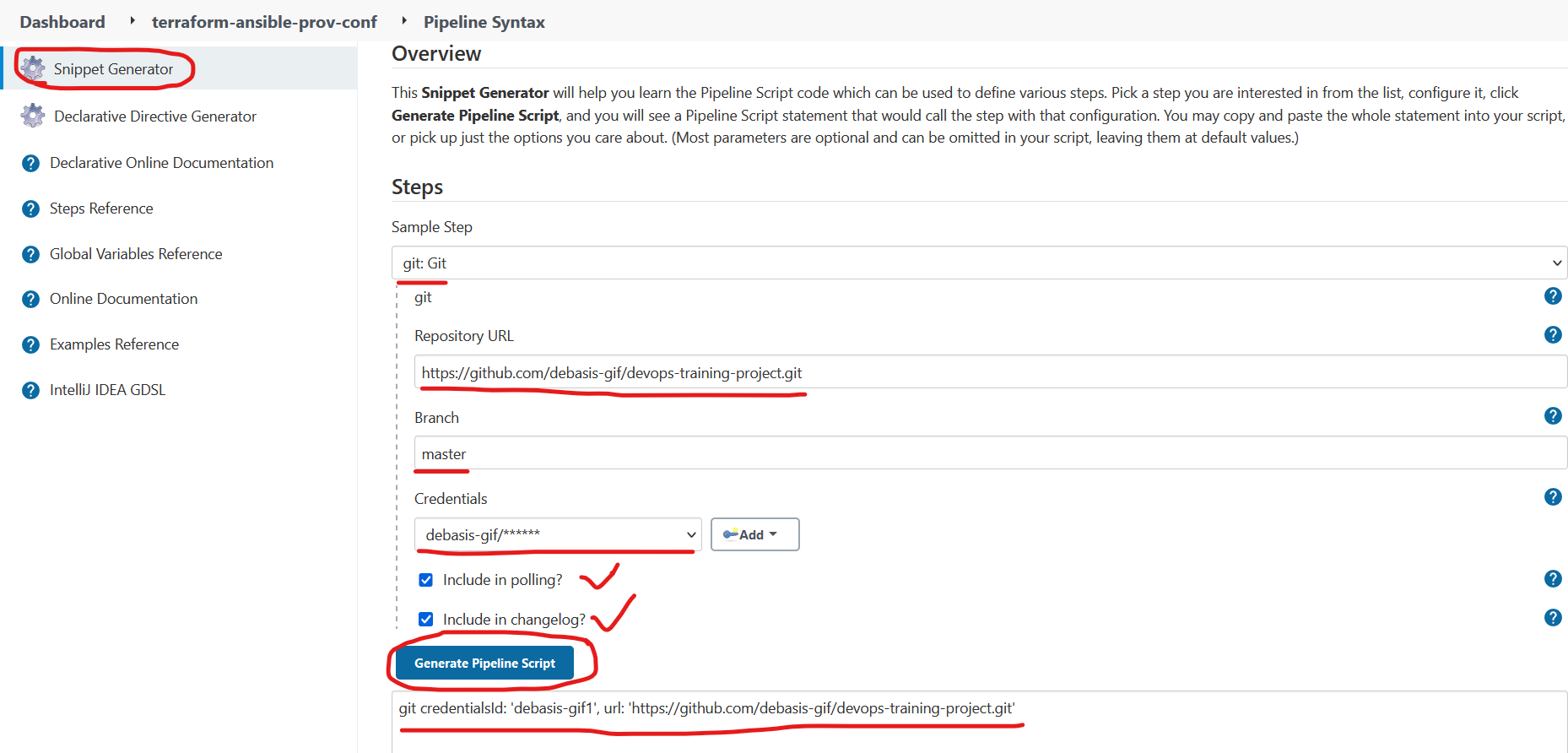
**NOT NEEDED**

Add the Credentials for Git in Jenkins (so as there is automation while no interruption for user-name & password) -> click on Jenkins





**NOT NEEDED**



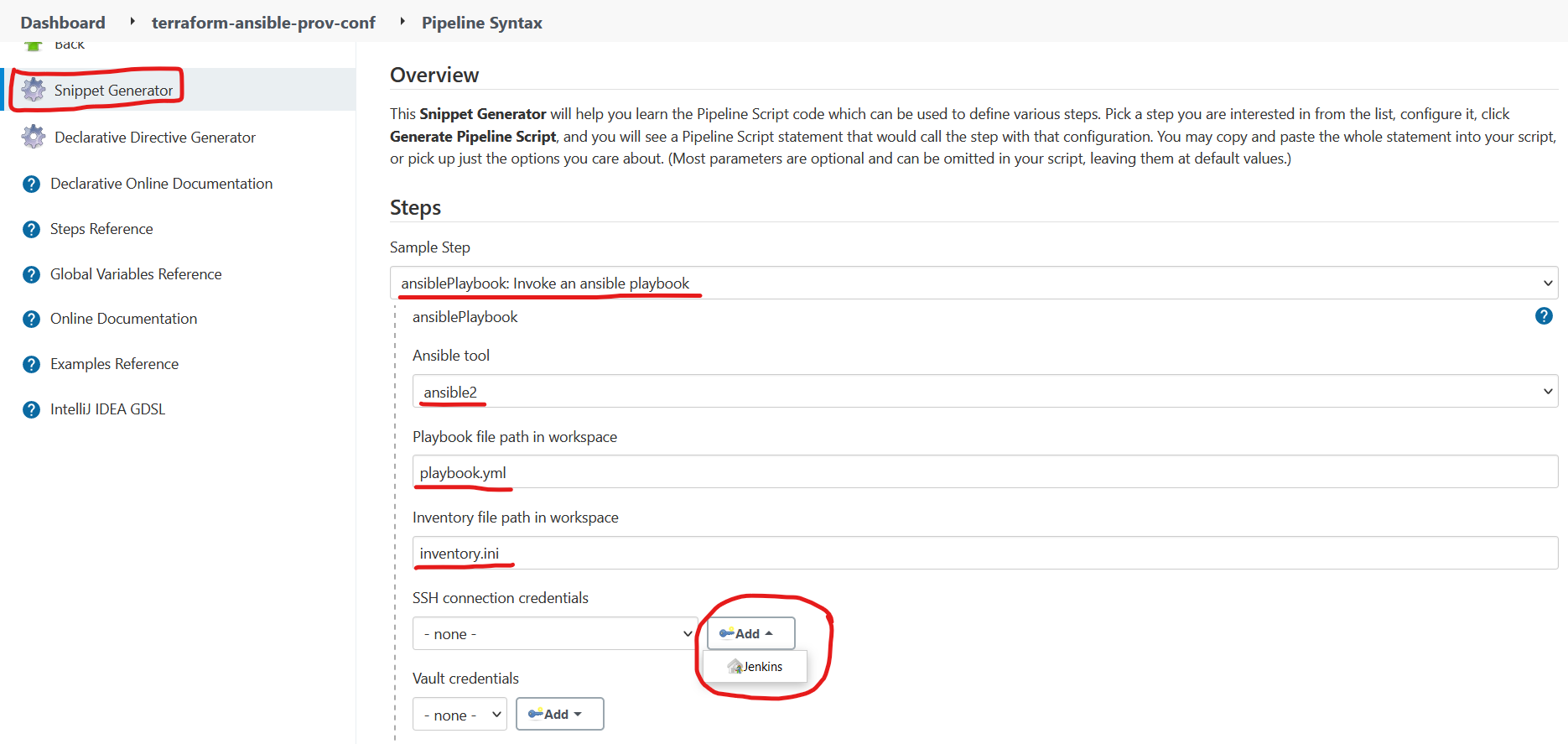
**NOT NEEDED**

Copy the snippet generate Pipeline script for the Git: step -> paste in the pipeline script

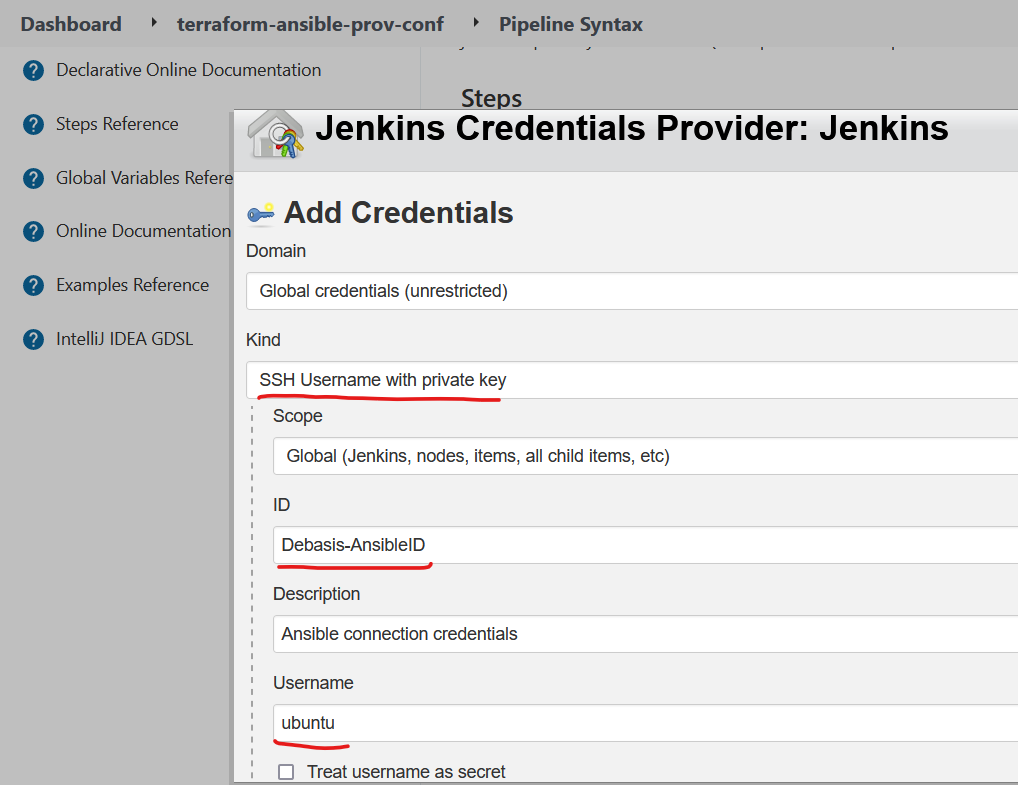
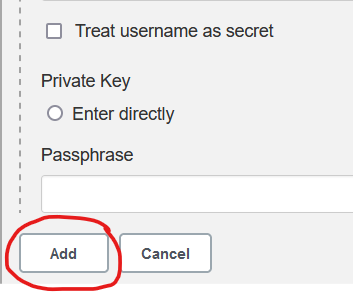
**NOT NEEDED**

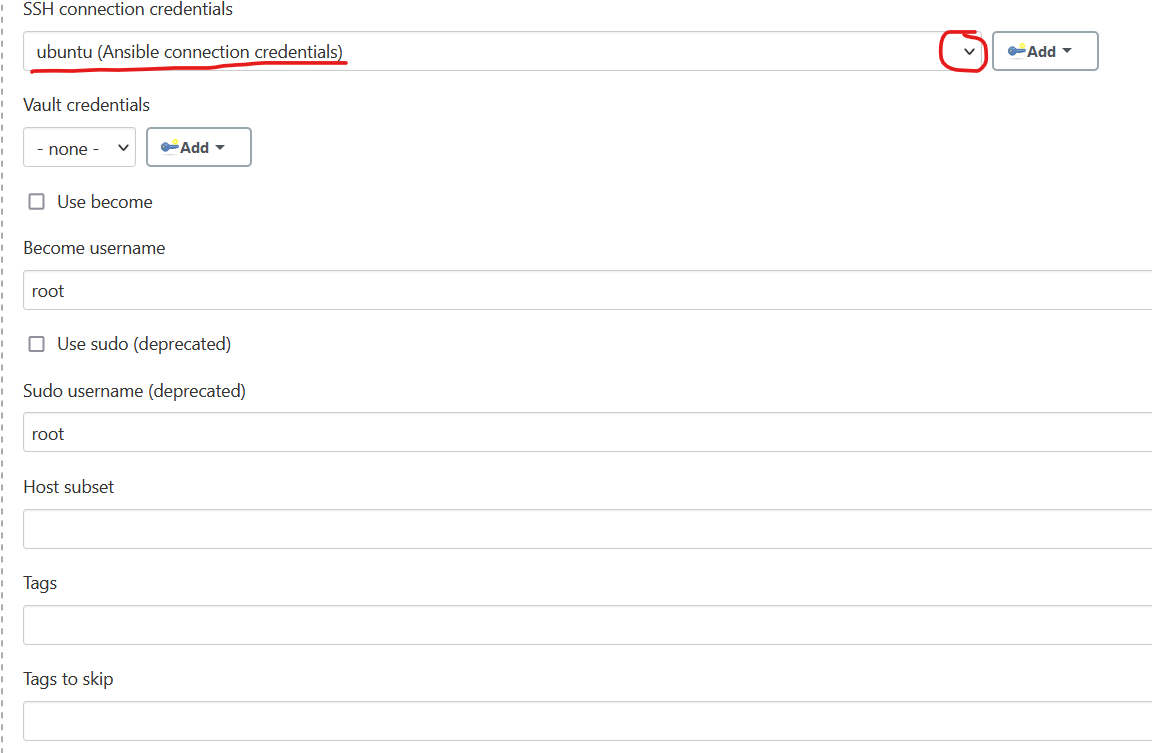
Add the Credentials for Ansible Playbook in Jenkins (also add the SSH credentials so as there is automation

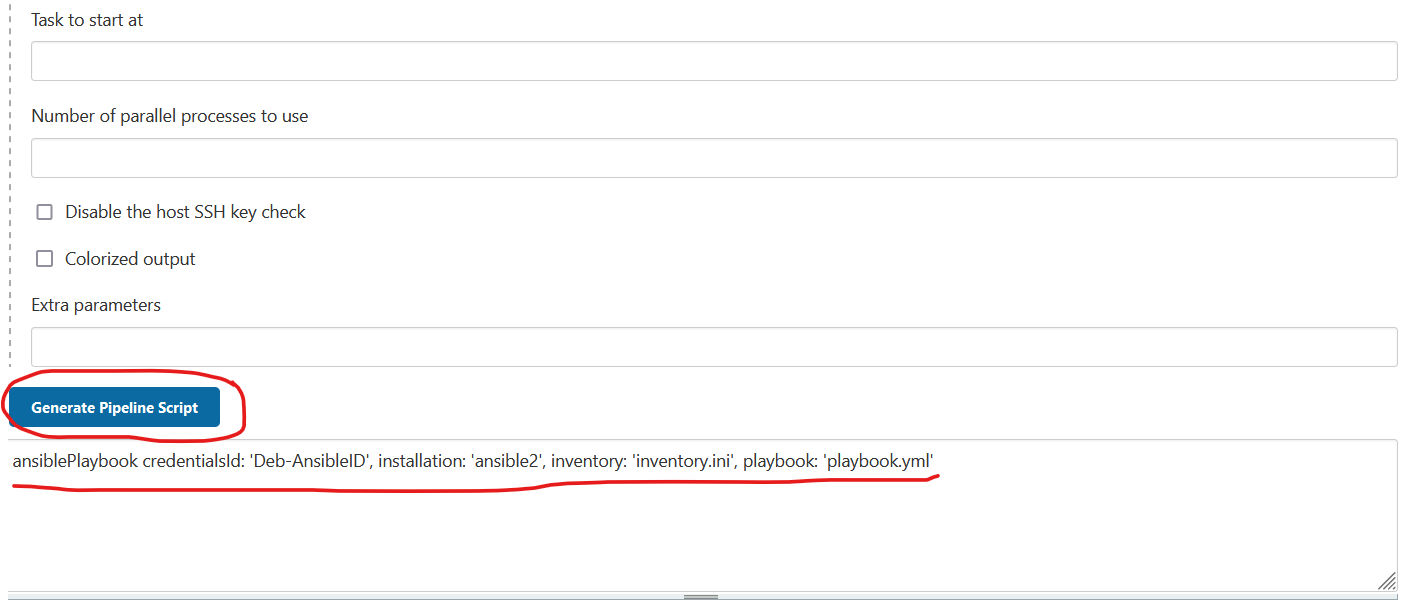
while no interruption for public key authentication) -> click on Jenkins



**NOT NEEDED**



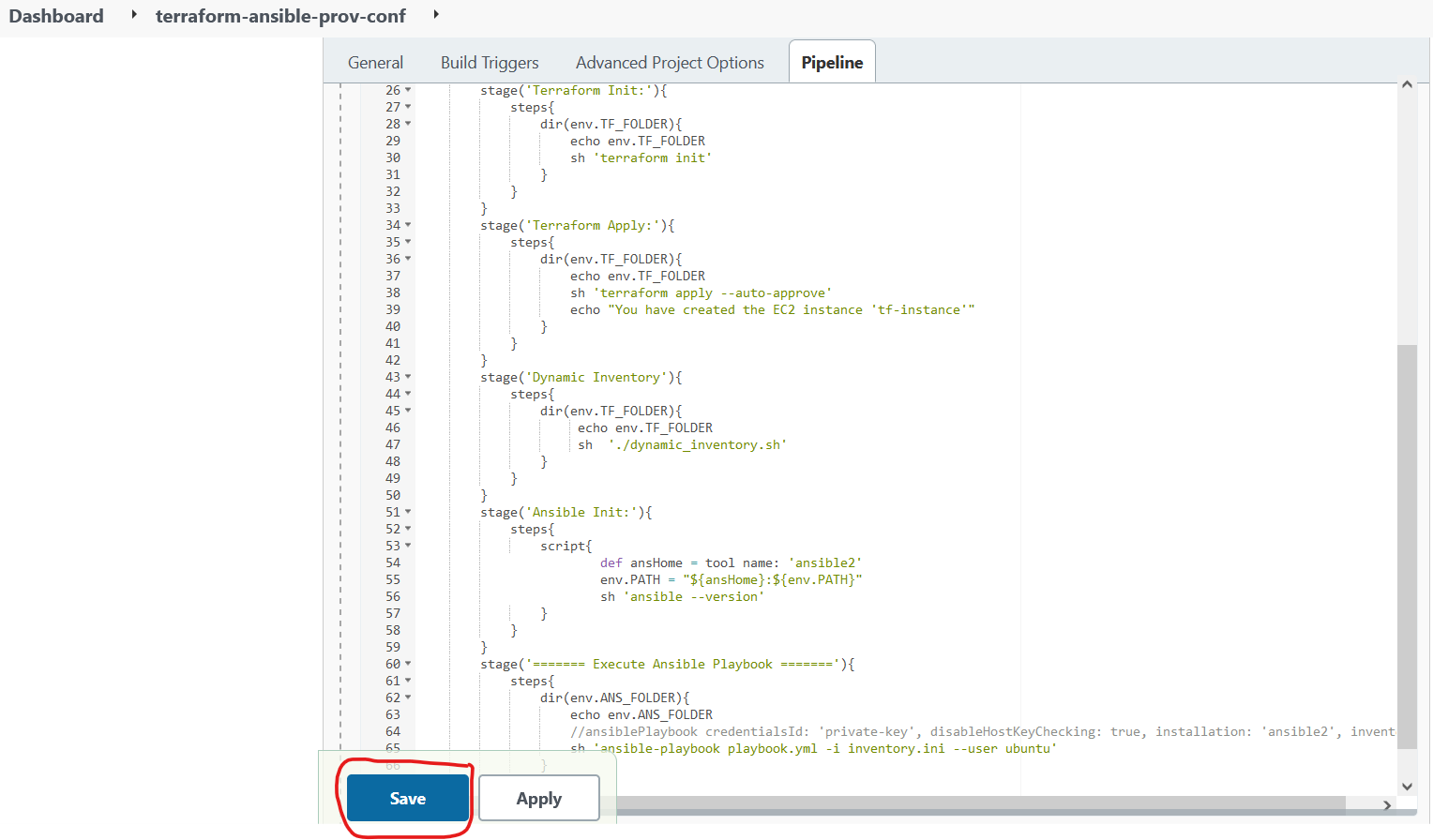


**NOT NEEDED**

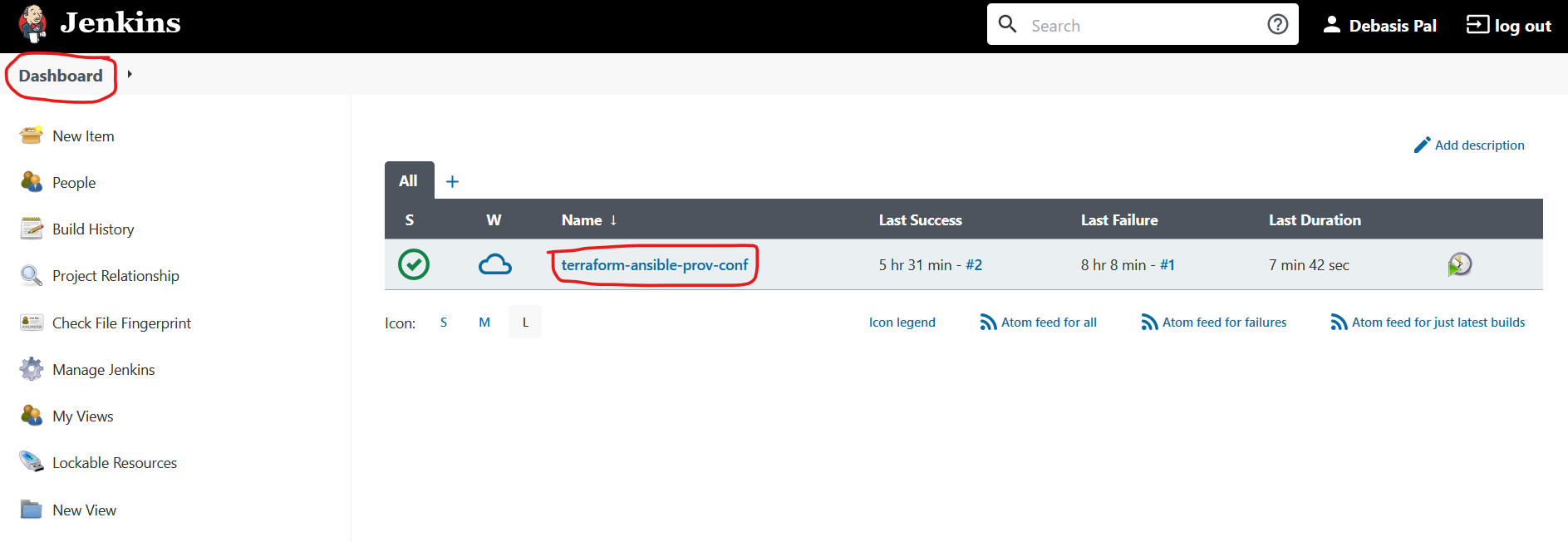
Copy the snippet generate Pipeline script for the ansiblePlaybook: step -> paste in the pipeline script

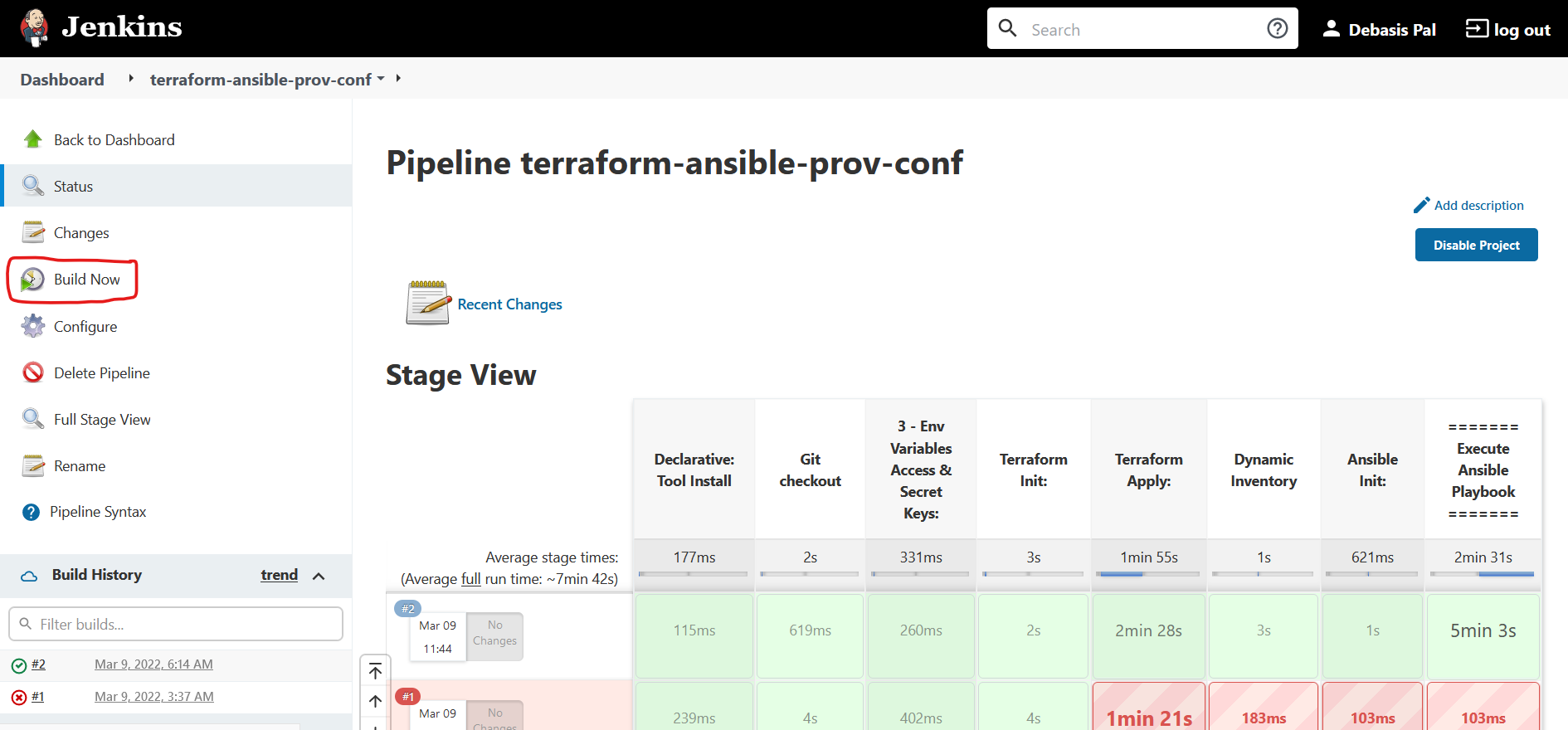
**Note:** We have not used this snippet in our pipeline script, instead we have used the ‘**sh**’ to execute the ansible commands directly without the help of ansible plugin,

Anyway one can always use the snippet code in place of directly executing the ansible commands through sh



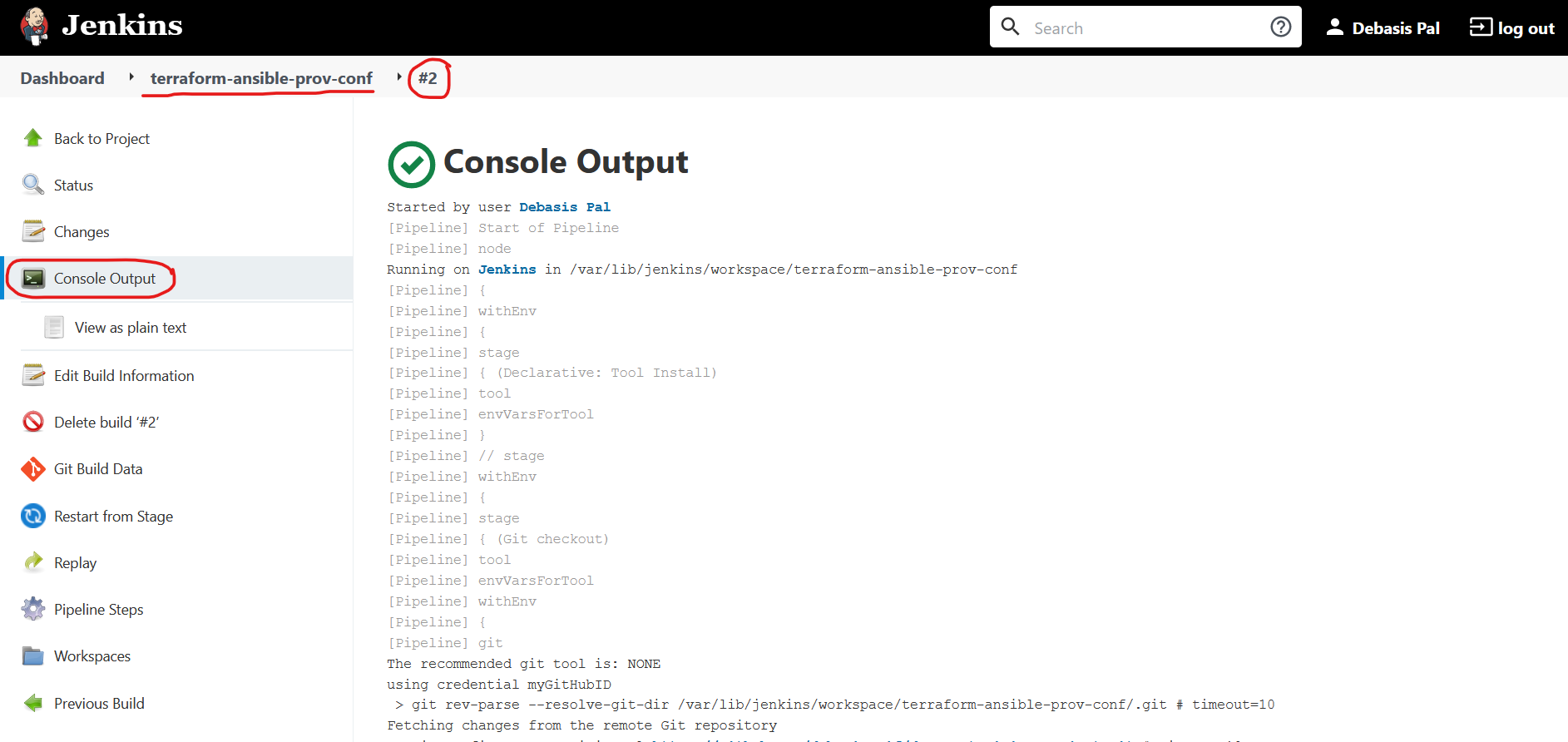
After completing the Pipeline scripting click the **Save** button as shown above





Finally click on the **Build Now** option to build the job and run the job.

As the job runs click on the #job number -> To open the page for the job run view -> click on the pipeline -> Console for the job run opens up to give the view of the job run



Attached: The Console log for the successful job run:



Note: All the DevOps processes like Ansible, Terraform, Git, Docker, Jenkins are **idempotent** in nature, that is a method or process is idempotent if an identical request made once or several times in a row with the same effect while leaving the server in the same state, that is there would no state change for the same operation over and over again.

All the exchange of information in AWS-Cli (backend) over to client (browser/terminal) is in form of **JSON** (javascript object notation) data. This concept has been utilized

In designing our shell script for dynamic\_inventory.sh for extracting the Public-IP-Address

Note:-

* Check the docker-compose container which must be running by now :-
  + $> docker ps
  + ***Now, run in the browser with the url for the Spring Pet Clinic application running on the mapped port -> 8090***
  + http://<public-ip-address>:8090/

Additional Reference for GitHub: <https://github.com/spring-petclinic/spring-framework-petclinic.git>

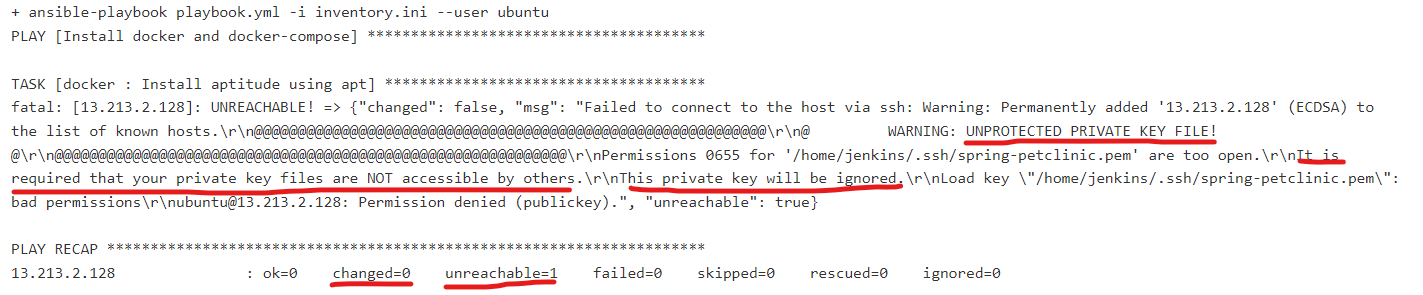
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Errors** **& Solution Section**

**=====================**

**Error - 1**

**=======**



Trouble-Shoot the above problem – the Solution

========================================

$> su -l jenkins

password: jenkins

$> cd .ssh

* *you’ll need to reset the permissions of the following key files generated after ssh\_keygen for jenkins user:*

$> sudo chmod 600 ~/.ssh/id\_rsa

$> sudo chmod 600 ~/.ssh/id\_rsa.pub

* *you may need to set the permissions of the known\_hosts file if there is any to 644:*

$> sudo chmod 644 ~/.ssh/known\_hosts

* *Finally, you may need to adjust the directory permissions as well:*

$> sudo chmod 755 ~/.ssh

* *Also, check the permissions of the spring-petclinic.pem file, it should only have read permission for the owner only, if not it should change:*

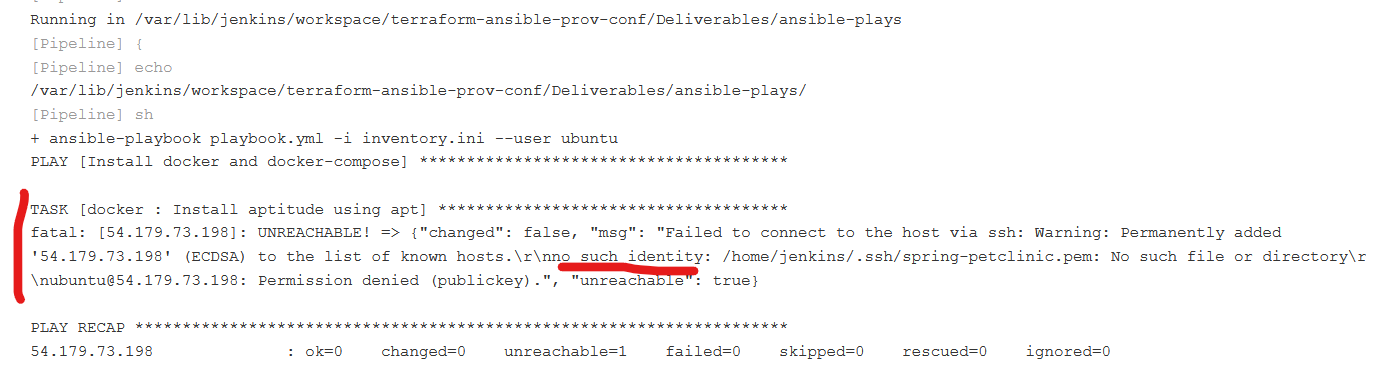
$> chmod 0400 spring-petclinic.pem

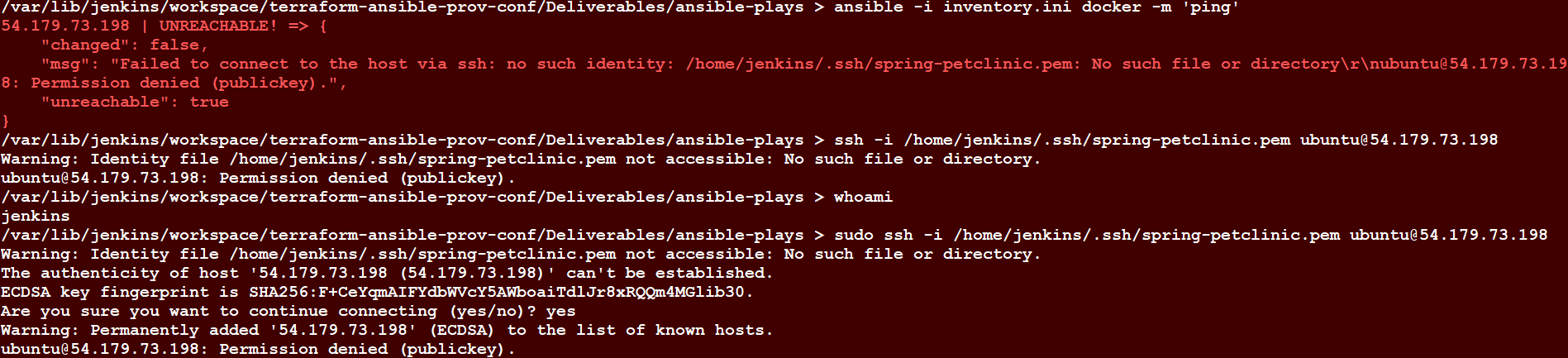
* Re-run the Jenkins job (Build now)

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Error - 2**

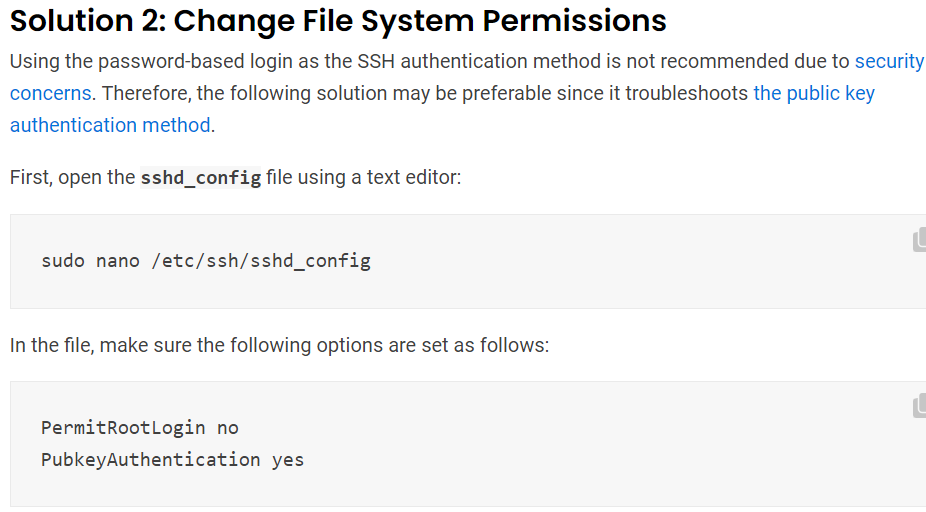
**=======**

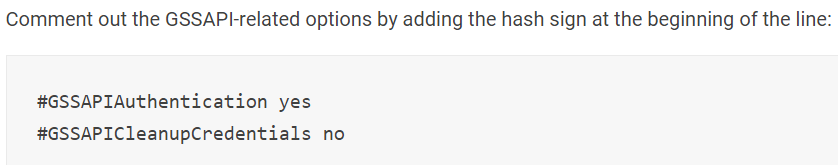


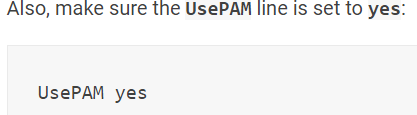


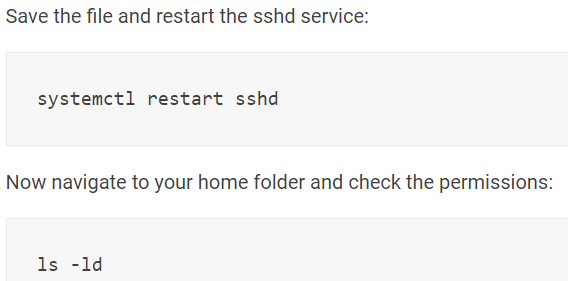
Trouble-Shoot the above problem – the Solution

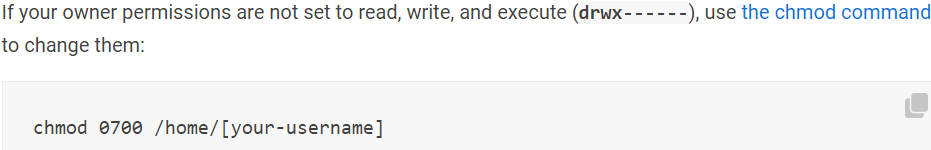
========================================

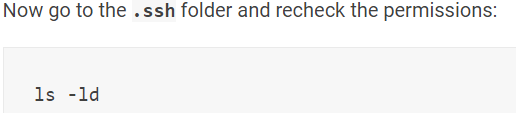


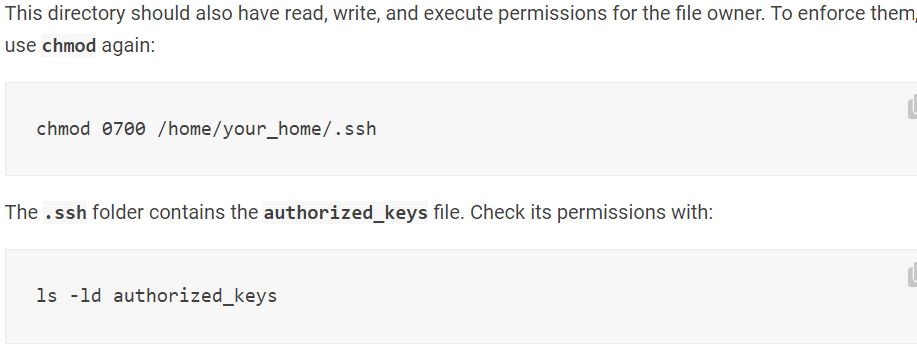


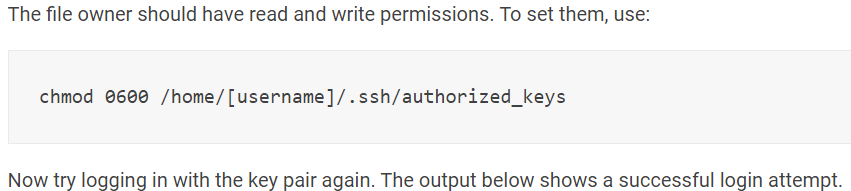












$ ssh -i <key-file> <user-name>@host-public-ip-addr

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Error - 3**

**=======**

**Problem**

Docker Compose Containers are exiting immediately after it is made up and running with docker-compose up -d

============================================================================================

**Solution**

**Docker: Prevent Container From Exiting Immediately After Creation**

Posted July 25, 2020 by [Yaroslav Grebnov](https://yaroslavgrebnov.com/contributors/yaroslav-grebnov/) ‐ **2 min read**

For some projects, it is useful to create one or several Docker images serving as parts of the environment, containing all software which is required to run the application project your team is working on.

In case such image is built with the container entry point like: CMD [“bin/sh”], the container may exit immediately after creation.

If the container is created using docker run -it, like in this example:

docker run -it ygrebnov/node-angularcli:latest ash

you can access it and work with it. As soon as you exit the container, it is stopped.

docker ps -a

You can see the Exited (0) About a minute ago status displayed for your container.

If the container is created using docker-compose with basic configuration, the container status is Exited(0) after:

docker-compose up -d

If there is a restart policy defined for the service in the docker-compose.yml, the container will be constantly restarting. In case of Exited(0) container status, most likely that docker logs <container\_id> command will show nothing.

Such behavior is explained by the fact that the created docker container process with PID 1 is started and successfully finished with the return code 0. Docker container is exited as soon as the process with PID 1 finishes. In our case, the process with PID 1 is /bin/sh.

In order to prevent the docker container from exiting immediately after creation, tty should be set to true in the docker-compose.yml file. **tty: true** in docker-compose corresponds to the docker run -it.

With **tty: true** for the service, the created by docker-compose up -d container status is Up.

An example of the docker image with CMD [“bin/sh”]: ygrebnov/node-angularcli. Docker Hub: <https://hub.docker.com/repository/docker/ygrebnov/node-angularcli>

An example of the docker-compose.yml using ygrebnov/node-angularcli image: <https://github.com/ygrebnov/mean-setup>.