**CAPSTONE PROJECT**

You have been hired Sr. Devops Engineer in Abode Software. They want to implement Deveops Lifecycle in their company. You have been asked to implement this lifecycle as fast as possible abode software is a product-based company, their product is available on this GitHub link.

<https://github.com/hshar/website.git>

Following are the specifications of the lifecycle:

1. Git Workflow has to be implemented
2. Code Build should automatically be triggered once commit is made to master branch develop branch.

If commits is made to master branch, test and push to prod

If commit is made to develop, just test the product, do not push to prod

1. The Code should be containerized with the help of a Dockerfile. The Dockerfile should be built every time there is a push to Git-Hub. Use the following pre-built container for your application.

Hshar/webapp

The code should reside in ‘/var/www/html’

1. The above tasks should be defined in a Jenkins Pipeline, with a following jobs.

Job 1 – Building website

Job2 – Testing website

Job3 – Push to production

1. Since you are setting up the server for the first time, ensure the following file exists on both Test and Prod server in /home/Ubuntu/config-management/status.txt. This file will be used by a third-party tool. This should basically have the info whether apache is installed on the system or not.

If apache is installed => Apache is installed on this System”

If apache is not installed => “Apache is not installed on this System”

Architectural Advice:

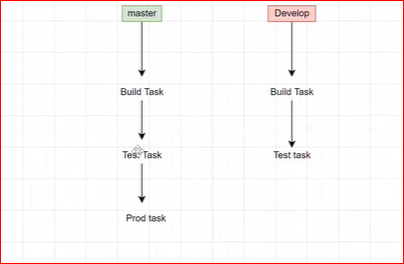
Create 3 servers on AWS “t2.micro”

Server 1 –should have Jenkins Master, Puppet Master and Nagios Installed

Server 2 – Testing Server, Jenkins Slave

Server 3 – Prod Server, Jenkins Slave

Below is project Architecture and diagram. Jenkins configured as below diagram.



First created Controller Instance where I control terraform and Ansible from this server.

Install Terraform and Ansible on this server using below command.

Terraform:

curl -O https://releases.hashicorp.com/terraform/1.0.11/terraform\_1.0.11\_linux\_amd64.zip

sudo apt install unzip -y

unzip terraform\_1.0.11\_linux\_amd64.zip

sudo mv terraform /usr/local/bin

Note: complete aws architecture

vpc

subnet

sg

route table

IGW

ec2 instance

**Go to IAM and Create User with Access key – Programmatic access. And select “AdminstratorAccess” . Attach this Existing policy to this user. And used this Access key in provider.**

**vi vpc.tf**

resource "aws\_vpc" "main" {

cidr\_block = "10.0.0.0/16"

tags={

Name = "demo\_vpc\_terraform"

}

}

resource "aws\_subnet" "public\_subnet" {

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.1.0/24"

map\_public\_ip\_on\_launch = "1"

availability\_zone\_id = "aps1-az1"

tags = {

Name = "public\_tf\_subnet"

}

}

resource "aws\_security\_group" "securtiy\_group" {

vpc\_id = aws\_vpc.main.id

ingress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "tf-security\_grp"

}

}

resource "aws\_internet\_gateway" "gw" {

vpc\_id = aws\_vpc.main.id

tags = {

Name = "tf-IGW"

}

}

#route table for public subnet with IGW

resource "aws\_route\_table" "table\_public" {

vpc\_id = "${aws\_vpc.main.id}"

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = "${aws\_internet\_gateway.gw.id}"

}

tags = {

Name = "rt\_public"

}

}

# route table association public subnet

resource "aws\_route\_table\_association" "association\_rt\_public" {

subnet\_id = aws\_subnet.public\_subnet.id

route\_table\_id = aws\_route\_table.table\_public.id

}

**# launch an instance using controller instance**

**Vi instance.tf**

provider "aws" {

access\_key = "AKIAVK…………"

secret\_key = "fZloTmQuGal……………."

region = "ap-south-1"

}

resource "aws\_instance" "web" {

ami = "ami-08ee6644906ff4d6c"

instance\_type = "t2.micro"

vpc\_security\_group\_ids= ["${aws\_security\_group.securtiy\_group.id}"]

subnet\_id= aws\_subnet.public\_subnet.id

key\_name= "latest\_key"

tags = {

Name = "master"

}

}

resource "aws\_instance" "web1" {

ami = "ami-08ee6644906ff4d6c"

instance\_type = "t2.micro"

vpc\_security\_group\_ids= ["${aws\_security\_group.securtiy\_group.id}"]

subnet\_id= aws\_subnet.public\_subnet.id

key\_name= "latest\_key"

tags = {

Name = "worker1"

}

}

resource "aws\_instance" "web2" {

ami = "ami-08ee6644906ff4d6c"

instance\_type = "t2.micro"

vpc\_security\_group\_ids= ["${aws\_security\_group.securtiy\_group.id}"]

subnet\_id= aws\_subnet.public\_subnet.id

key\_name= "latest\_key"

tags = {

Name = "worker2"

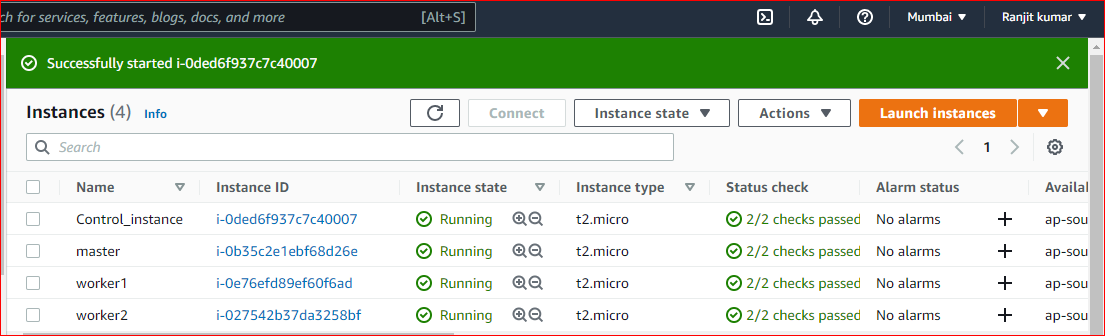
}

}

terraform init

terrafrom plan

terrafrom apply # Now All master and worker server is created using terraform as below screen.



**Now I configured Ansible on Control Instance and using ansible I configured all software to all server.**

**Install Ansible**:

$ sudo apt update

$ sudo apt install software-properties-common

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

$ sudo apt install ansible

$ ssh-keygen # id\_rsa\_key.pub key generated and copy content to all server in authorize.key file.

**inventory file /etc/ansible/hosts**

[server]

host1 ansible\_host=13.233.166.116 # master server IP

host2 ansible\_host=13.235.19.100 # Worker1 server IP

host3 ansible\_host=13.127.154.198 # Worker2 server IP

ansible all -m ping # to test connection

**On controller Server: write ansible playbook and software installation script file.**

**jenkins.sh**

sudo apt-get update

sudo apt install ca-certificates

sudo apt-get install openjdk-11-jdk -y

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install jenkins -y

**java.sh**

sudo apt-get update

sudo apt-get install openjdk-11-jdk -y

**docker.sh**

sudo apt-get update

sudo apt-get install docker.io –y

**Vi ansible.yml**

---

- hosts: **host1** # master machine

become: yes

name: Installing apache2 on slave1

tasks:

- name: executing a script file for installing jenkins on master machine.

script: jenkins.sh

- name: executing a script file for installing java on master machine.

script: java.sh

- name: executing a script file for installing docker on master machine.

script: docker.sh

- hosts: **host2** # worker1 machine

become: yes

name: Installing apache2 on slave1

tasks:

- name: executing a script file for installing java on master machine.

script: java.sh

- name: executing a script file for installing docker on master machine.

script: docker.sh

- hosts: **host3** # worker2 machine

become: yes

name: Installing apache2 on slave1

tasks:

- name: executing a script file for installing java on master machine.

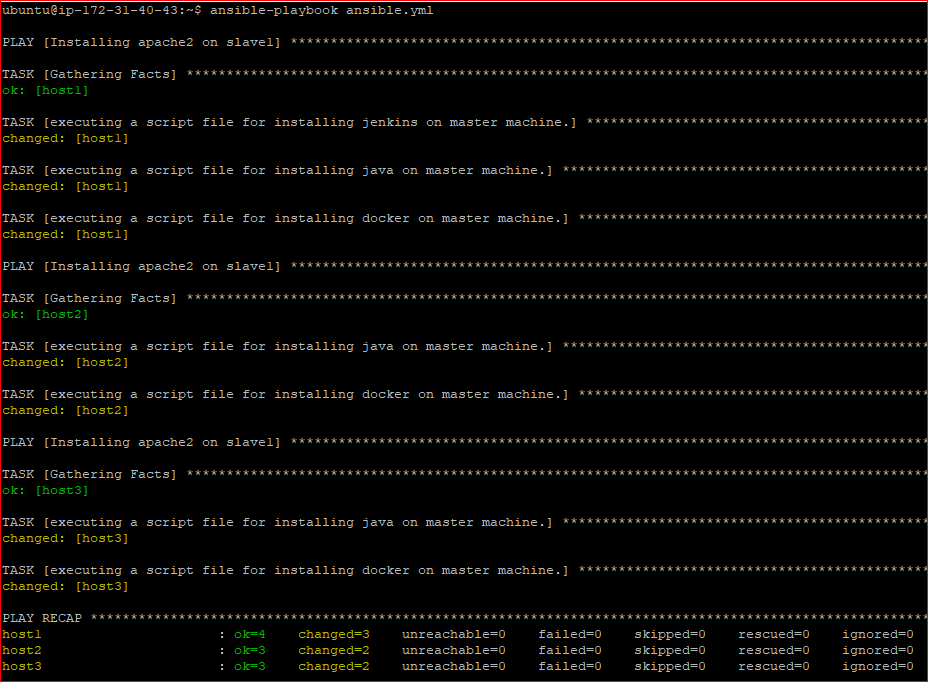
script: java.sh

- name: executing a script file for installing docker on master machine.

script: docker.sh

ansible-playbook ansible.yml --syntax-check

ansible-playbook ansible.yml # execute as in blow screen. And installed all required software.



Now All software configured in all server using Ansible.

URL: masterIP:8080 # login Jenkins admin/admin

manage Jenkins > configure global security > Agents Random >apply and save

**# Now I am going to create Jenkins Slave1 and Slave2 to distribute the load as below steps.**

**On Master Node**: first generate ssh-keygen. And copy key (id\_rsa.pub) and past in ec2 worker1/worker2 in authorize-key.

Manage Node > New Node > Node name: **worker1** > permanent agent > ok

Number of executer:2 , Remote root dir: /home/Ubuntu/Jenkins , Launch method : Select “launch agent via SSH” , Host: private ip for worker1

Add credentials: Select “SSH Username with private key”

id: Ubuntu (anything)

desc: ,,

Username: Ubuntu

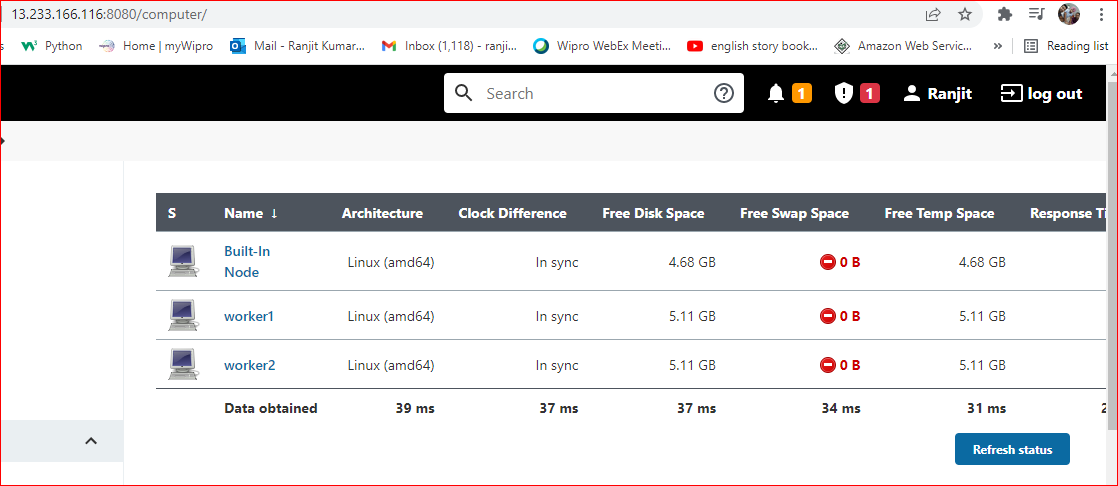
private key (Enter directly): > add > in second box, past of .ssh/id\_rsa (key of master machine) > add

Select Ubuntu key (created key)

Host key verification strategy: select “Non verify verification strategy” > save

Refresh status # worker1 slave activated as below screen.

Same configure Slave for **worker2 also.**



**# Now I am going to create Jenkins Jobs as below steps. (job1, job2, job3)**

**Go to create New job job1** (free style)

Description: develop branch, task build and test

Checked GitHub project: https://github.com/ranjit8569/website.git

Checked Restrict where this project: select “**worker1**” Slave.

Source Code Management:

checked Git: https://github.com/ranjit8569/website.git

branch: \*/develop # job execute for develop branch

checked GitHub hook trigger for GITScm polling

apply and save

**# when build job then create workspace folder**

worker1 ec2> cd Jenkins/workspace/job1 ; ls # Docker file, images, index.html

configure job1 > Build > execute shell >

sudo docker rm -f c1\_test

sudo docker build /home/ubuntu/jenkins/workspace/job1 -t app1

sudo docker run -itd --name c1\_test -p 82:80 app1

Apply and save.

build job1 . # success

**Create new Jenkins job2**

description: master branch, task build and test

checked GitHub project > project url: https://github.com/ranjit8569/website.git (master branch path)

Checked Restrict where this project can be run

Label Expression: worker2

Source Code Management:

Checked Git : https://github.com/ranjit8569/website.git

branch: \*/master

checked GitHub hook trigger for GITScm polling

Apply and save

build job2 : to create workspace folder on worker2 node # project file exist there and copy path

configure job2 > Build > Execute Shell >

sudo docker rm -f test\_c2

sudo docker build /home/ubuntu/jenkins/workspace/job2/ -t app2

sudo docker run -itd --name test\_c2 -p 83:80 app2

Apply and save

build job2 . # success

**Go to create job3**

description: post build job ----> job2

task prod task running a container at port 80

checked GitHub project https://github.com/ranjit8569/website.git # master branch path

checked: Restrict where this project can be run

Label: worker2

Source Code Management

URL : https://github.com/devopsintellipaat/website.git

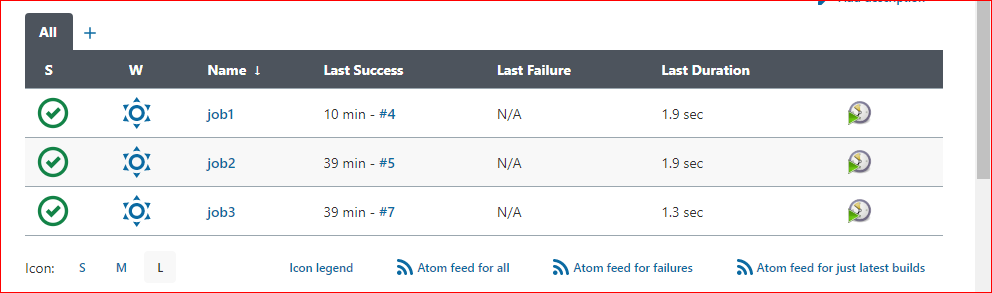
Build > Execute Shell >

#sudo docker rm -f prod

sudo docker run -itd --name prod -p 80:80 app2

Apply and save

build job3 and success.



**goto job2** and configure > **Add post-build action** > Build other project > projects to build: select “**job3**” > Apply and save.

**build job2 and execute job2 as well job3, and got success.**

Apt install git –y

Git init # created git repository

Git remote add origin <https://github.com/ranjit8569/website.git>

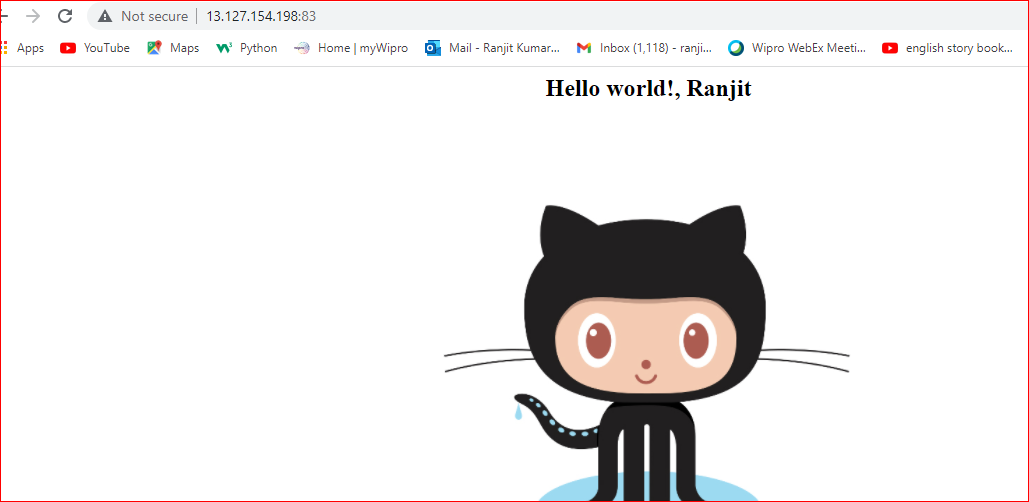
Git clone <https://github.com/ranjit8569/website.git>

Vi index.hml # edit

Git add .

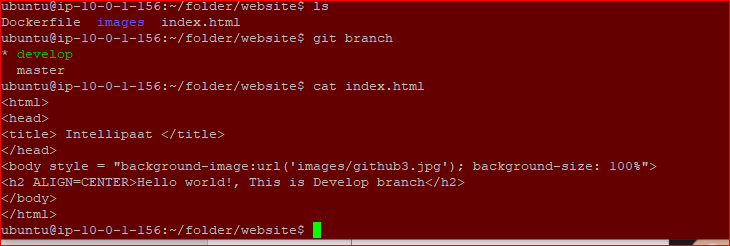
Git commit –m “first commit”

Git branch develop



**goto github.com > setting > webhook > add webhook > payload URL: jenkin url (http://ip:8080/github-webhook/ > Add webhook**

git checkout develop # switched branch

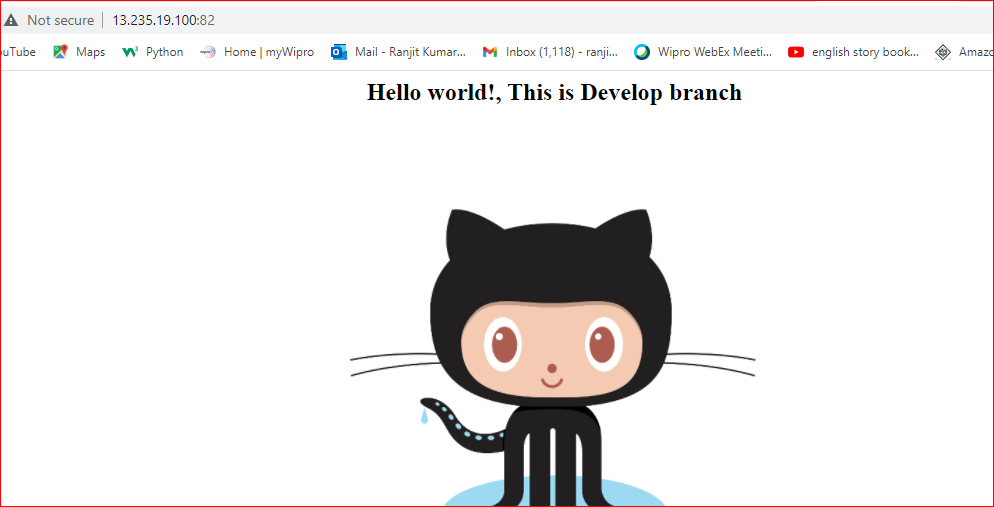


Vi index.hml # edit

Git add .

Git commit –m “develop branch”

Git push origin develop



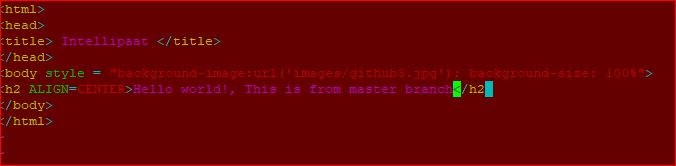
goto master node > Git checkout master

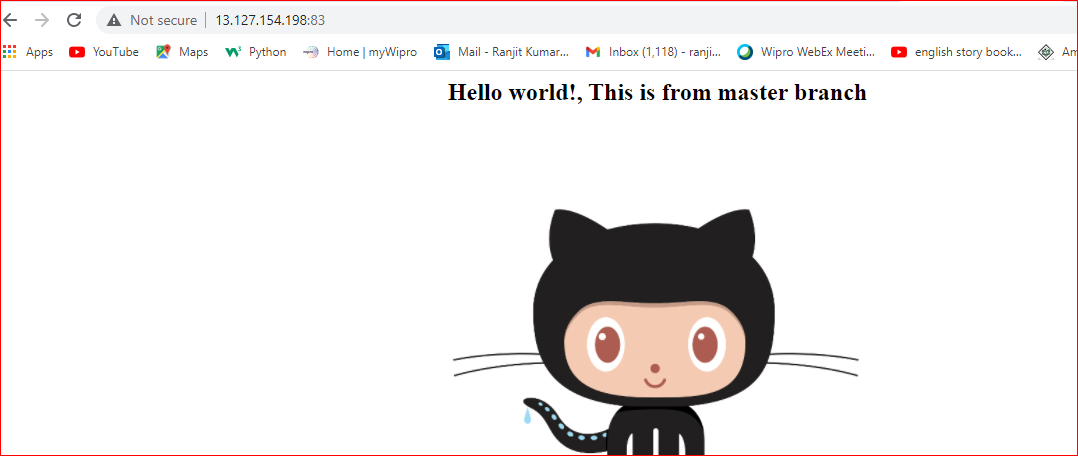
cd website ; vi index.html # edit some content ( this is from master branch)

git add .

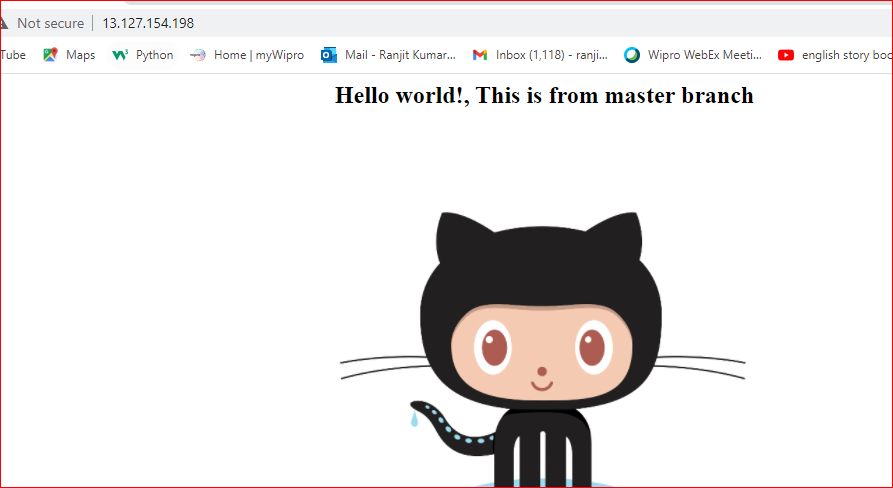
git commit -m "changed"

git push origin master # automatically job executed.





All port is running fine.



Below is container running fine after Jenkins job execute.

