**Capstone Project**

**Problem Statement**

You are hired as a DevOps engineer for Analytics Pvt Ltd. This company is a product-based organization which uses Docker for their containerization needs within the company. The final product received a lot of traction in the first few weeks of launch. Now with the increasing demand, the organization needs to have a platform for automating deployment, scaling, and operations of application containers across clusters of hosts, As a DevOps engineer, you need implement a DevOps life cycle, such that all the requirements are implemented without any change in the Docker containers in the testing environment.

Up until now, this organization used to follow a monolithic architecture with just 2 developers. The product is present on https://github.com/hshar/website.git

Following are the specifications of life-cycle:

1. Git workflow should be implemented. Since the company follows monolithic architecture of Development you need to take care of version control. The release should happen only on 25th of every month.

2. Code build should be triggered once the commits are made in the master Branch.

3. The code should be containerized with the help of the Docker file, The Dockerfile should be built every time if there is a push to Git-Hub. Create a custom Docker image using a Dockerfile.

4. As per the requirement in the production server, you need to use the Kubernetes cluster and the containerized code from Docker hub should be deployed with 2 replicas. Create a NodePort service and configure the same for port 30008.

5. Create a Jenkins pipeline script to accomplish the above task.

6. For configuration management of the infrastructure, you need to deploy the configuration on the servers to install necessary software and configurations.

7. Using Terraform accomplish the task of infrastructure creation in the AWS cloud provider.

**Architectural Advice**

Software’s to be installed on the respective machines using configuration management.

**Worker1**: Jenkins, Java.

**Worker2**: Docker, Kubernetes.

**Worker3**: Java, Docker, Kubernetes

**Worker4**: Docker, Kubernetes.

**Solution Approach**

**Step 1: Create and Setup ec2 instances**

* We will create 3 ec2 instances: 1 Master, 2 Slave nodes for prod and test
* We will set up the connections such that we can ssh from the Master into all Slave nodes

**Step 2: Setting up Master-Slave architecture for ansible**

Used the following set of codes to set up master slave architecture between the Master and 2 Slaves

1. which python3 # check if python is installed

2.

3. # Install Ansible only in the master node

4. sudo yum install -y ansible

5.

6. # Create a new user with the name "ansible" in the MASTER node

7. sudo useradd ansible

8. sudo passwd ansible # Generate a password for the user

9. su – ansible # log in to the user

10.

11. # Create a new user with the name "ansible" in the both slave nodes

12. sudo useradd ansible

13. sudo passwd ansible # Generate a password for the user

14. su - ansible

15.

16. # Provide sudo access to the ansible user in the both the slave nodes

17. cd /etc/ # go to folder

18. sudo vi sudoers # open sudoers in a text editor

19. # add this to wheel: ansible ALL=(ALL) NOPASSWD: ALL

20.

21. # Generate a public and private key in the MASTER node

22. su - ansible # log in to the user

23. ssh-keygen # command to generate keypairs

24.

25. # In order to establish the ssh connection, we need the public IP of the Slave nodes

26. curl ifconfig.me

27.

28. # Run this in SLAVE node to enable ssh port

29. cd /etc/ssh # go to ssh folder

30. sudo vi sshd\_config # the file sshd\_config has details that needs to be changed

31. # Search for the setting "PasswordAuthentication" and change the setting from "no" to "yes"

32. sudo systemctl restart sshd # restart sshd service

33.

34. # Now try gaining remote access to both servers from Master

35. ssh ansible@13.235.79.77 # Slave 1

36. ssh ansible@52.66.239.74 # Slave 2

37.

38. # Copy public key from master to remote server

39. ssh-copy-id ansible@{slave public ip}

**Step 3: Creating all ansible setup file**

1. Creating the configuration file as follows:

1. [defaults]

2. inventory = /etc/ansible/hosts.ini

3. become = True

4. become\_method = sudo

5. become\_user = root

6. fork = 5

7. timeout = 30

1. Creating the hosts.ini file as follows:

1. [slave]

2. 13.235.79.77

3. 52.66.239.74

1. Creating the ansible playbooks
   1. The first notebook will install Java and Jenkins in the master node

---

- name: Install Jenkins on AWS Linux EC2 Master Instance

hosts: localhost

become: yes

tasks:

- name: Update all packages

yum:

name: "\*"

state: latest

- name: Install Java (required by Jenkins)

command: sudo dnf install java-17-amazon-corretto -y

- name: Download Jenkins repository

get\_url:

url: https://pkg.jenkins.io/redhat-stable/jenkins.repo

dest: /etc/yum.repos.d/jenkins.repo

mode: '0644'

- name: Import Jenkins GPG key

rpm\_key:

state: present

key: https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

- name: Check for package upgrades

yum:

name: "\*"

state: latest

- name: Install Jenkins

yum:

name: jenkins

state: present

- name: Start Jenkins service

service:

name: jenkins

state: started

enabled: true

3.2 The second notebook will install Java, Git and Docker on both the slave nodes

---

- name: Install Jenkins on AWS Linux EC2 Master Instance

hosts: slave

become: yes

tasks:

- name: Update all packages

yum:

name: "\*"

state: latest

- name: Install Java 17 Amazon Corretto

command: sudo dnf install java-17-amazon-corretto -y

- name: Install Docker

yum:

name: docker

state: present

- name: Start and enable Docker service

systemd:

name: docker

state: started

enabled: true

- name: Install Git

yum:

name: git

state: present

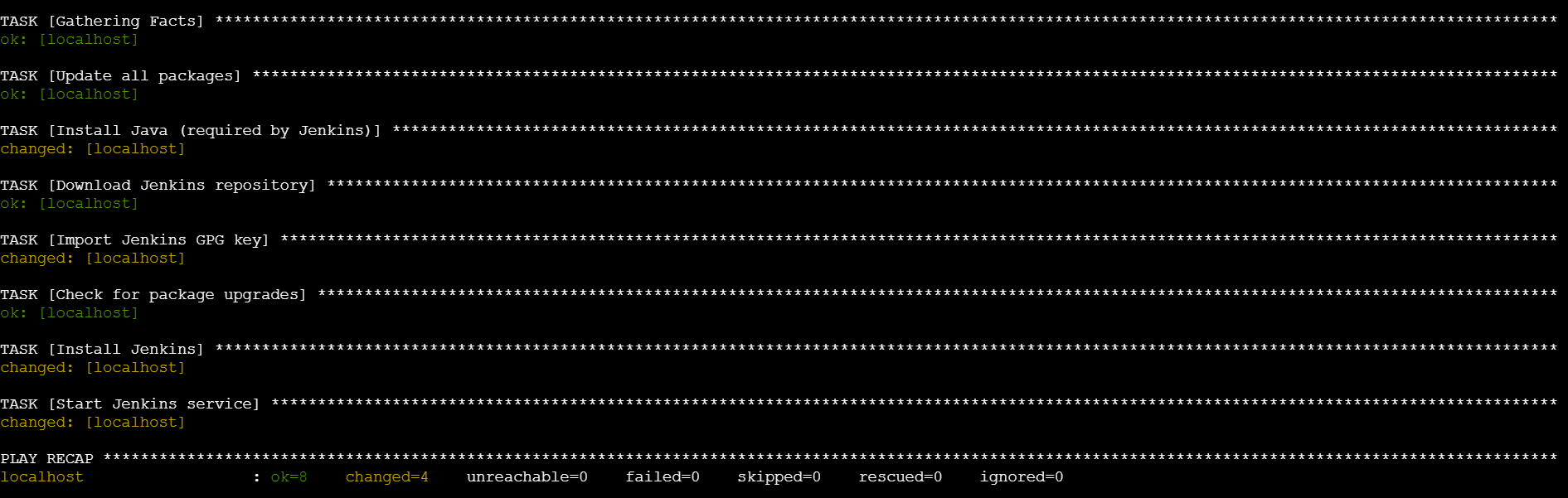
**Step 4: Executing the Ansible Playbook**

The ansible playbook was executed using the below commands:

1. ansible-playbook playbook\_master.yml --syntax-check   # code to check for syntax error

2. ansible-playbook playbook\_master.yml --check          # dry run on terminal

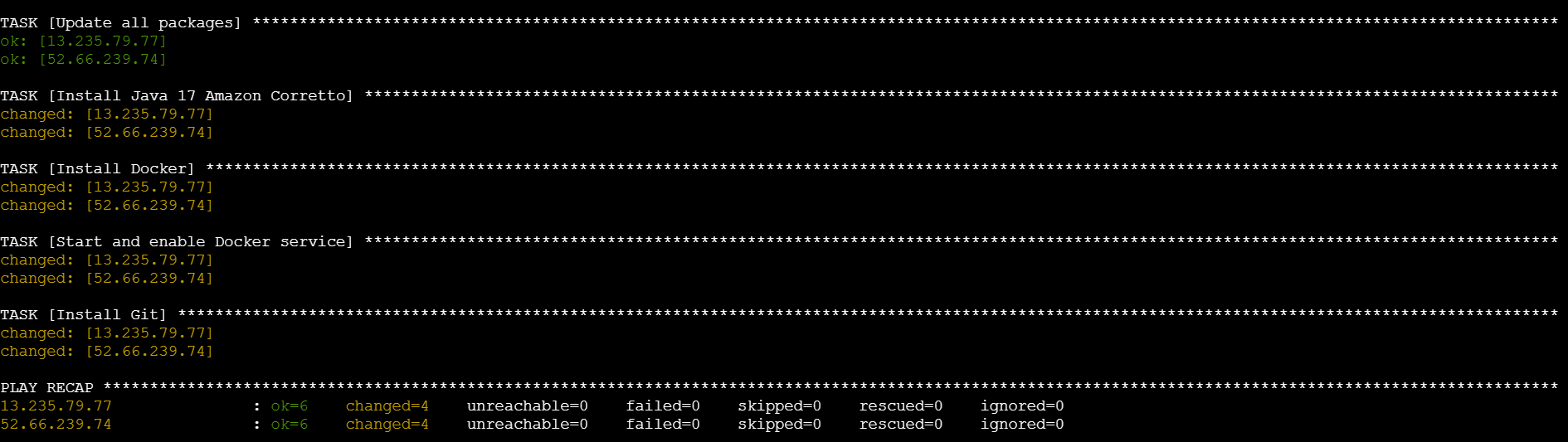
3. ansible-playbook playbook\_master.yml                  # final run on the server



1. ansible-playbook playbook\_slave.yml --syntax-check   # code to check for syntax error

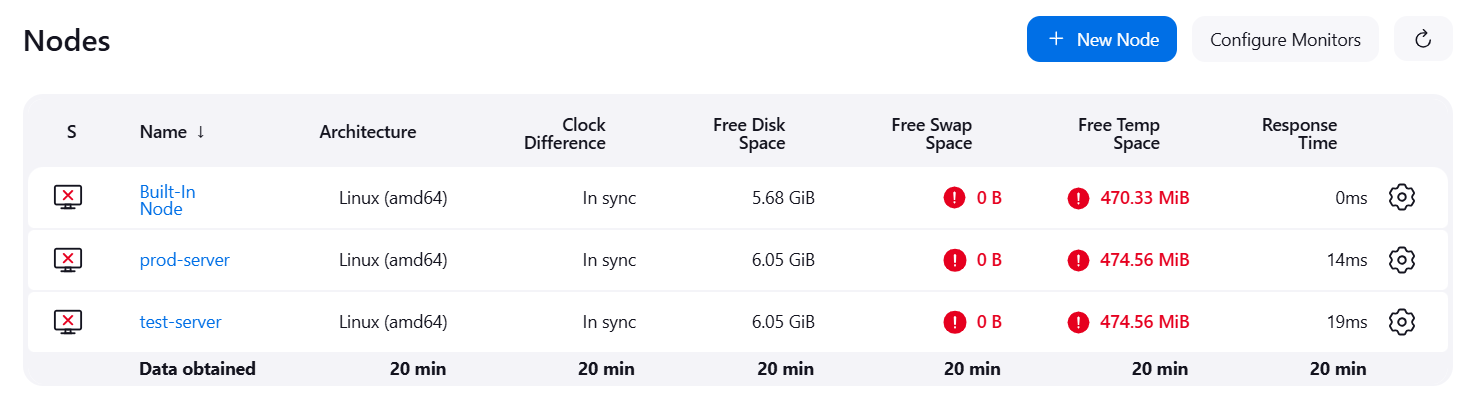
2. ansible-playbook playbook\_slave.yml --check          # dry run on terminal

3. ansible-playbook playbook\_slave.yml                  # final run on the server



**Step 5: Connect to Jenkins and set up Nodes**

1. Setting up the *test-server* node:
   * Name: *test-server*
   * Number of executors: *1*
   * Remote root directory: */home/ec2-user/jenkins/*
   * Labels: *test*
   * Usage: *Use this node as much as possible*
   * Launch Method: *Launch agents via SSH*
     + Host: *172.31.7.92 (Jenkins test private IP)*
     + Credentials: *ec2-user*
     + Host key verification strategy: *Non verifying*
   * Availability: *Keep this agent online as much as possible*
2. Setting up the *prod-server* node:
   * Name: *prod-server*
   * Number of executors: *1*
   * Remote root directory: */home/ec2-user/jenkins/*
   * Labels: *test*
   * Usage: *Use this node as much as possible*
   * Launch Method: *Launch agents via SSH*
     1. Host: *172.31.9.91 (Jenkins prod private IP)*
     2. Credentials: *ec2-user*
     3. Host key verification strategy: *Non verifying*
   * Availability: *Keep this agent online as much as possible*

****

**Step 6: Clone Git Repository, Setup Git Workflow, Create Dockerfile**

Clone the GitHub repository provided in the problem statement. Create a new Dockerfile as shown below and push the file to Github

*Dockerfile*

FROM ubuntu

RUN apt-get update

RUN apt-get install apache2 -y

ADD . /var/www/html/

ENTRYPOINT apachectl -D FOREGROUND

The repository just has a master branch as of now. We also create a *develop* branch as well that will be used for testing purposes

All these changes are then pushed back to GitHub.

**Step 7: Create Jobs on Jenkins and set-up GitHub Webhook**

1. Setting up the *Job 1*:
   * Create new item as freestyle project: *push-to-test*
   * Description: *Trigger build and push to test server when change made in develop*
   * GitHub project: <https://github.com/vyombhatt/website.git>
   * Restrict where the project is run: *test-server*
   * Source code management
     1. Repository URL: <https://github.com/vyombhatt/website.git>
     2. Credentials: *creds not needed as repo is public*
     3. Branch Specifier: *\*/develop*
   * Build Triggers: *GitHub hook trigger for GitScm polling*
   * Build Steps: Execute Shell

*sudo docker rm -f $(sudo docker ps -a -q)*

*sudo docker build . -t job1\_docker\_image*

*sudo docker run -itd -p 80:80 job1\_docker\_image*

1. Setting up the *Job2*:
   * Create new item as freestyle project: *push-to-prod*
   * Description: *Trigger build and push to test server when change made in master*
   * GitHub project: <https://github.com/vyombhatt/website.git>
   * Restrict where the project is run: *test-server*
   * Source code management
     1. Repository URL: <https://github.com/vyombhatt/website.git>
     2. Credentials: *creds not needed as repo is public*
     3. Branch Specifier: *\*/master*
   * Build Triggers: *GitHub hook trigger for GitScm polling*
   * Build Steps: Execute Shell

*sudo docker rm -f $(sudo docker ps -a -q)*

*sudo docker build . -t job1\_docker\_image*

*sudo docker run -itd -p 80:80 job1\_docker\_image*

1. Setting up the *Job3*:
   * Create new item as freestyle project: *push-to-prod*
   * Description: *Trigger build and push to prod server when change made in master*
   * GitHub project: <https://github.com/vyombhatt/website.git>
   * Restrict where the project is run: *prod-server*
   * Source code management
     1. Repository URL: <https://github.com/vyombhatt/website.git>
     2. Credentials: *creds not needed as repo is public*
     3. Branch Specifier: *\*/master*
   * Build Triggers: *GitHub hook trigger for GitScm polling*
   * Build Steps: Execute Shell

*sudo docker rm -f $(sudo docker ps -a -q)*

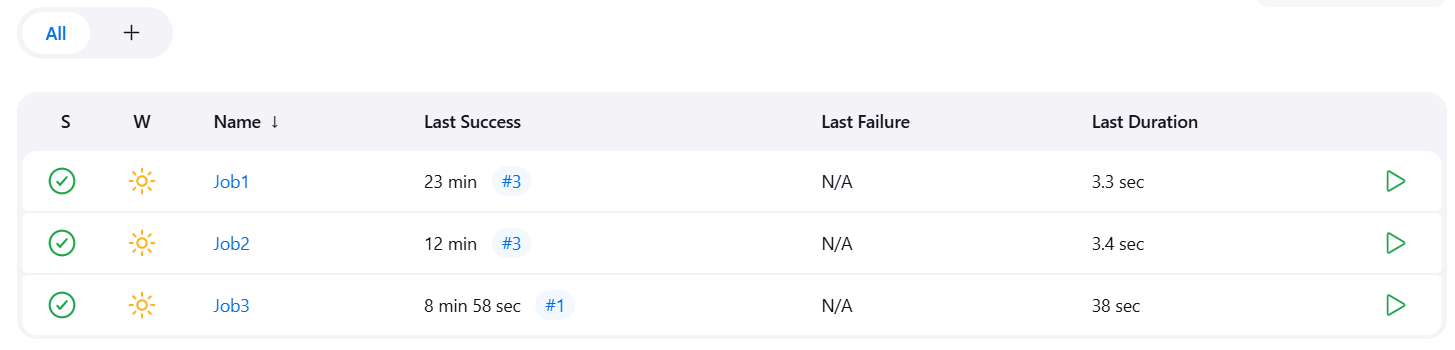
*sudo docker build . -t job1\_docker\_image*

*sudo docker run -itd -p 80:80 job1\_docker\_image*

Creating Webhook on GitHub:

* Payload URL: <http://13.201.222.193:8080/github-webhook/> (http://{jenkins\_url}/github-webhook/)
* Content type: application/x-www-form-urlencoded

**Step 8: Test whether the Jobs get triggered on making commits to Develop and Master**



Jobs work fine!