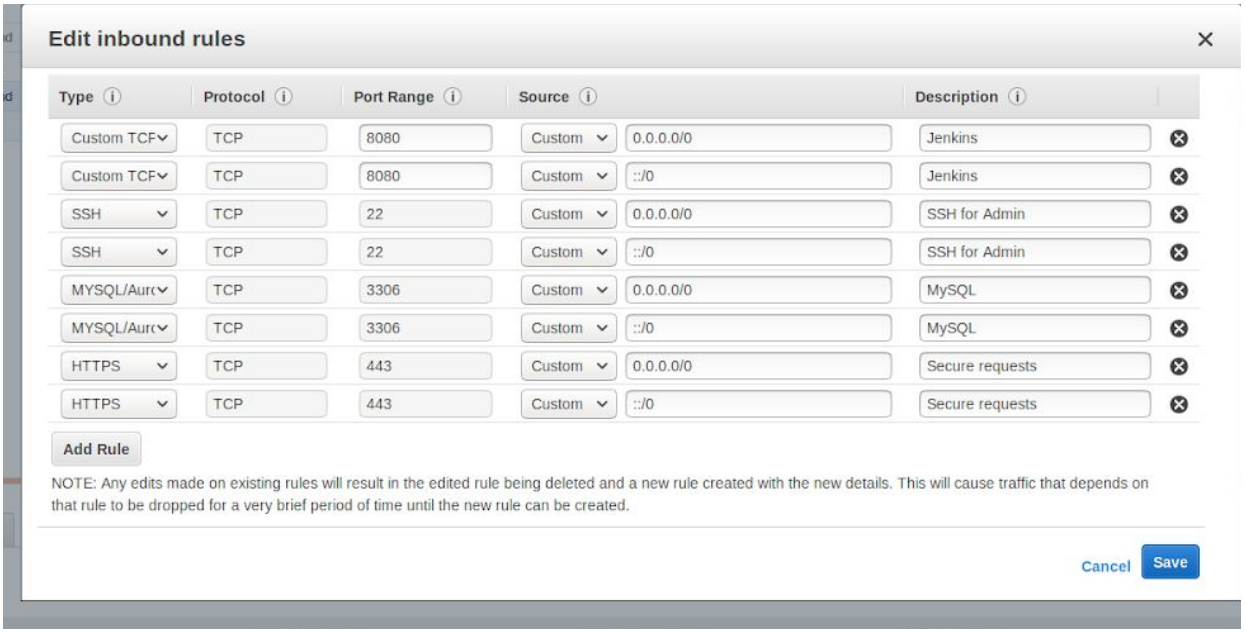
DevOps Project

1.) Set up VPC and subnets -

- Create a VPC (10.0.0.0/16)
- Create a subnet for each Availability zone within London region
- The CIDR blocks for the subnets are (10.0.1.0/24, 10.0.2.0/24, 10.0.3.0/24)

2.)Create Security groups for front end and back end and attach to the VPC created in step one



- **change source from anywhere to IP at a later point so that this is not publicly accessible.

Separate security groups were created for both the front end and back end instance.

3. Create IAM role with RDS Full access, RDS data full access and rdsconnect policies

4. Create RDS on AWS
   - Select MySQL latest version 8.0.16
   - free tier
   - name your database
   - Database instance class – burstable classes
   - Storage- general purpose (SSD)
   - enable autoscaling
   - connect your VPC made in step one
   - Ensure RDS is not publicly accessible
   - Add security group for back-end (made in step 2)
   - Database is on default port 3306

5. Create a new role to give EC2 access to RDS
   - Assign the policy from step 3 to give full access to RDS

6. Create an EC2 instance for back end (with the previously created role)
   - Select AMI: **Ubuntu Server 18.04 LTS (HVM), SSD Volume Type** -free-tier

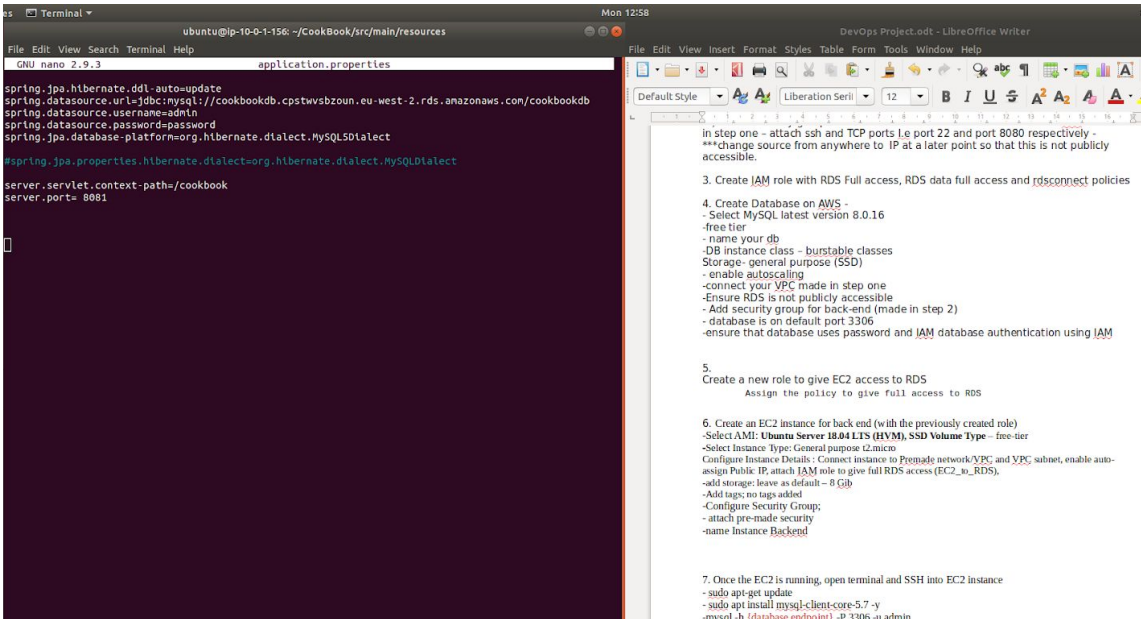
- Select Instance Type: General purpose t2.micro
- Configure Instance Details : Connect instance to Premade network/VPC and VPC subnet, enable auto-assign Public IP, attach IAM role to give full RDS access (EC2_to_RDS),
- add storage: leave as default – 8 Gib
- Add tags; no tags added
- Configure Security Group; attach pre-made security group for back end
- name Instance Backend

7. Once the EC2 is running, open terminal and SSH into EC2 instance

- sudo apt-get update
- sudo apt install mysql-client-core-5.7 -y
- mysql -h {database endpoint} -P 3306 -u admin

- Install docker
- Make docker start up on boot
- Clone back end repository
- Change connection details to use RDS
- Build back end image
- Run backend container with --restart flag
- Use Postman to test the endpoint
- Stop the instance
- Start the instance
- Test again with postman that the docker container started up automatically

8. Change endpoint in application props of project



9. Install Maven and Docker

- sudo apt-get update
- sudo apt install maven

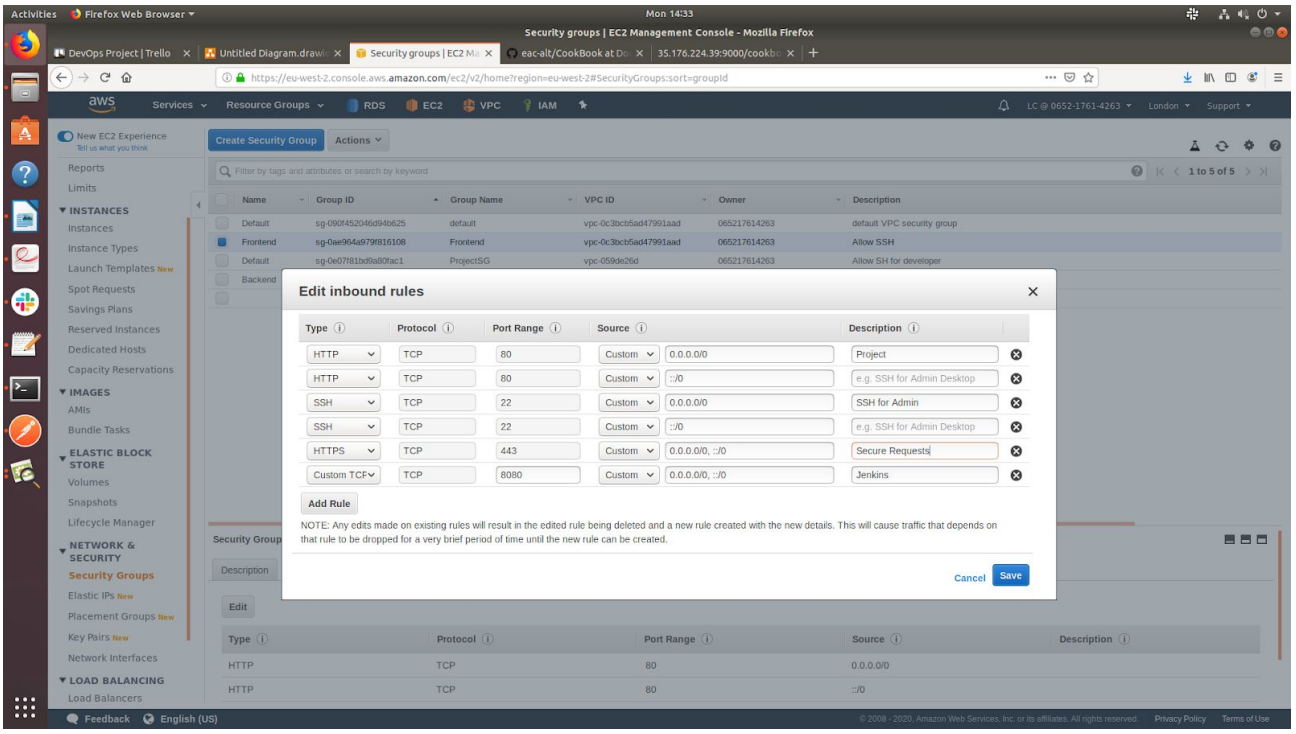
- sudo apt-get remove docker
- docker-engine docker.io
- sudo apt install docker.io
- docker version
- sudo usermod -aG docker $(whoami)

10. Run mvn build of project, run docker build and create container

- sudo mvn clean package
- docker build -t app-name .
- docker run -d -p 9000:8081 --name app app

11.Test using postman or via entering IP address in browser

13. Launch new EC2 instance for front-end and attach pre-made front-end security group



14. Install docker and containerise the front end- also ensure the Nginx.conf file is pointing to the back end instance(use the back-end instance IP)

15. Test front and back end connection using front end instance IP in browser and sending HTTP requests via the front end.

16. Jenkins

Create Jenkinsfile for back end as below→

```
pipeline {
    agent any
    stages {

        stage('--Mvn clean package--') {
                steps {
                    sh "mvn clean package deploy"
                }
```

15.Create new instance for Jenkins

16. Within the Jenkins EC2 instance install JDK and Maven:

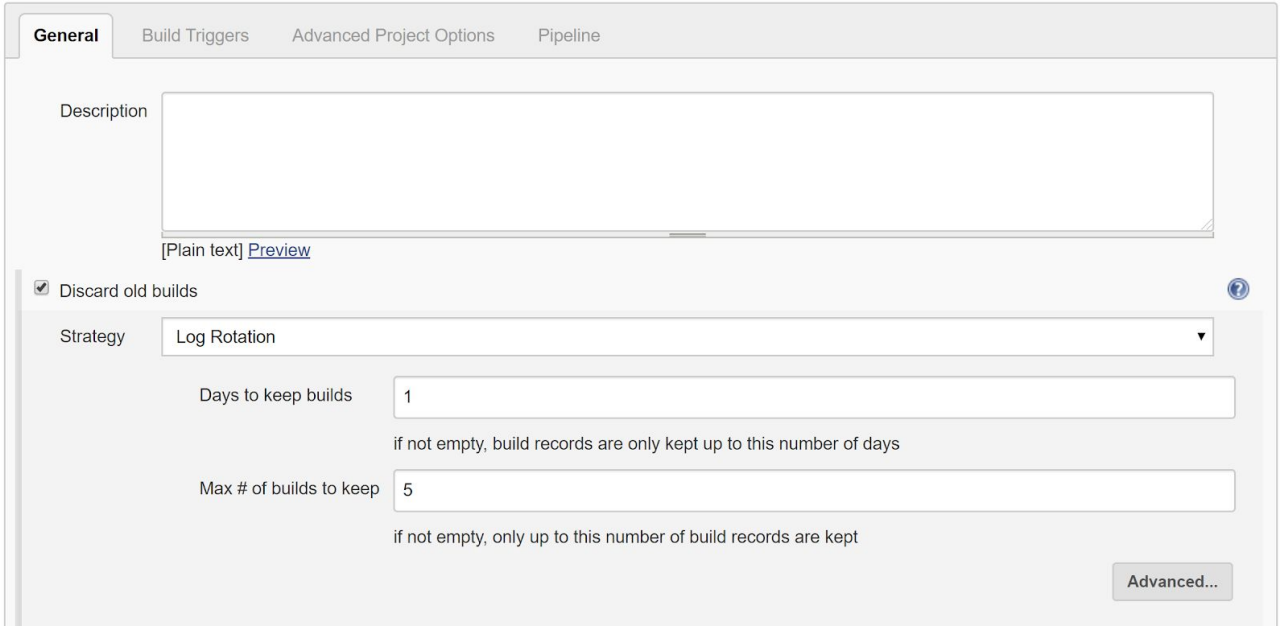
sudo apt install default-jdk -y
sudo apt install maven -y

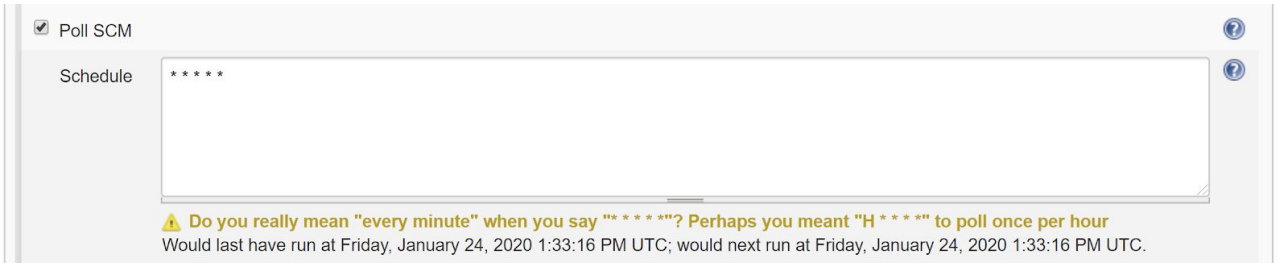
- jenkins > Credentials > add credentials > add dockerhub username and password> then save.

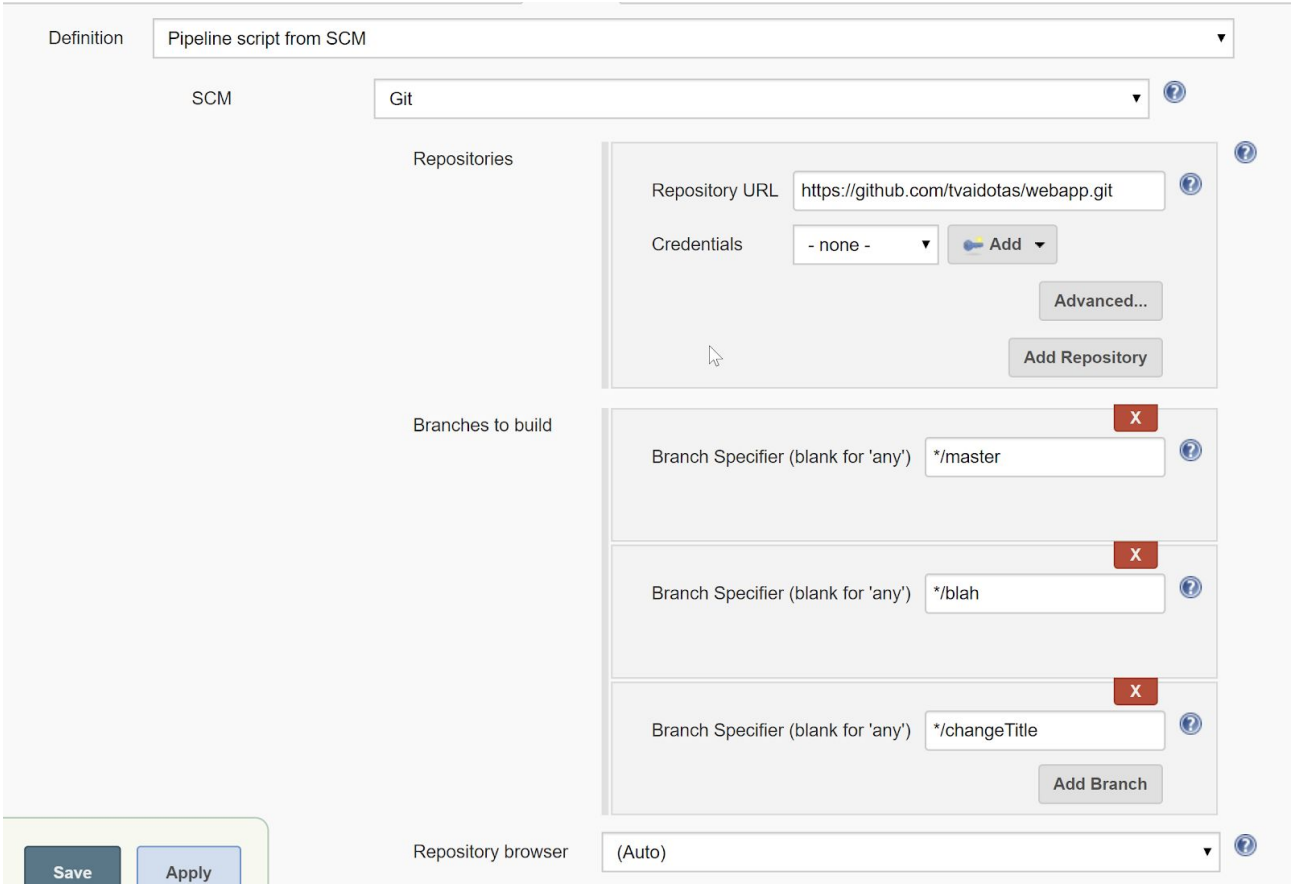
- within Jenkins instance run Jenkins script file
- create jenkins username and password;
  username; admin
  password; password

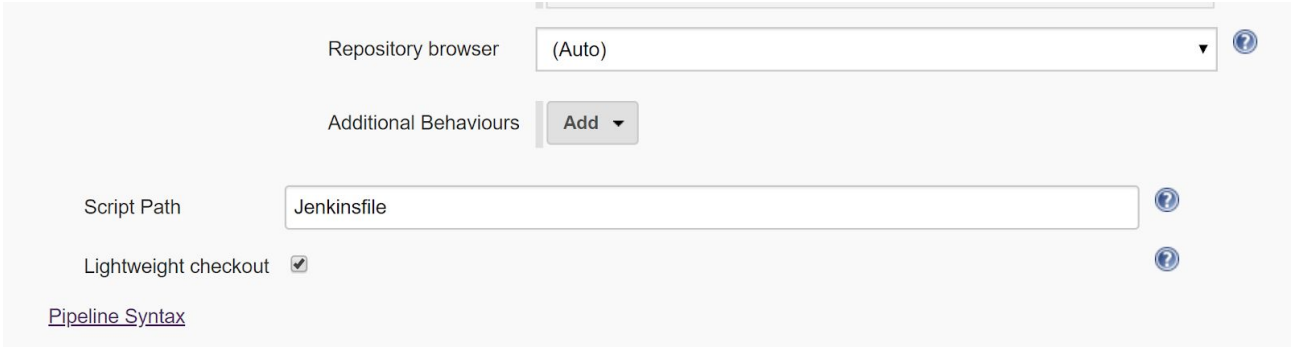
Manage Jenkins > Manage Plugins > available > search maven integration > download now and install after restart

Select Create new item and then pipeline




Poll SCM
Schedule
* * * * *
Do you really mean "every minute" when you say "* * * * *"? Perhaps you meant "H * * * *" to poll once per hour
Would last have run at Friday, January 24, 2020 1:33:16 PM UTC; would next run at Friday, January 24, 2020 1:33:16 PM UTC.


Definition
Pipeline script from SCM
SCM
Git
Repositories
Repository URL
https://github.com/tvaidotas/webapp.git
Credentials
- none -
Add
Advanced...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/master
Branch Specifier (blank for 'any')
*/blah
Branch Specifier (blank for 'any')
*/changeTitle
Add Branch
Repository browser
(Auto)
Save
Apply


Repository browser
(Auto)
Additional Behaviours
Add
Script Path
Jenkinsfile
Lightweight checkout
Pipeline Syntax

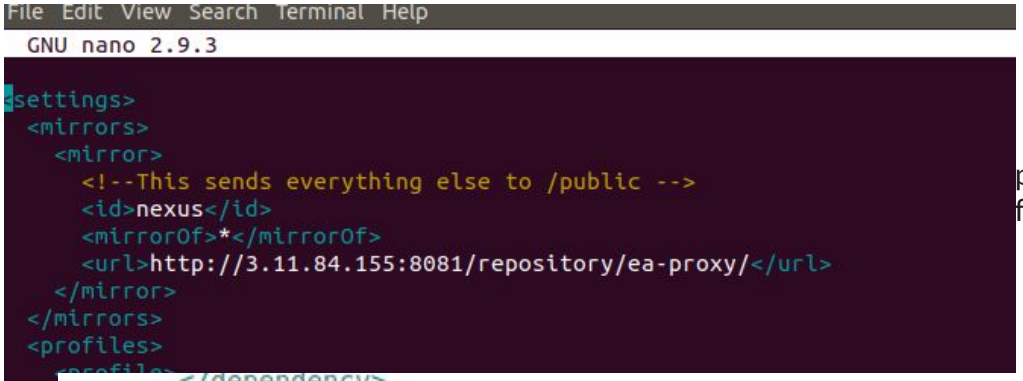
within jenkins create two jobs

Two for dev (front and back end)

Nexus

Go to Nexus VM link
Within Jenkins VM > go to M2 dir > create a settings.xml > set IP of URL to URL of Nexus VM : http://3.11.84.155:8081/repository/ea-proxy/

username: ea
password: password
hosted is for pom.xml
proxy is for settings.xml



change pom.xml to the following:

-

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
        <finalName>spring-boot-probook</finalName>
    </build>

    <distributionManagement>
        <snapshotRepository>
            <id>nexus</id>
            <name>maven-snapshots</name>
            <url>http://3.11.84.155:8081/repository/ea-hosted/</url>
        </snapshotRepository>
    </distributionManagement>
```

Make sure

Nexus is running prior to Jenkins build.
- use Nexus to build snapshot

Dockerhub

Modify Jenkinsfile in BE with

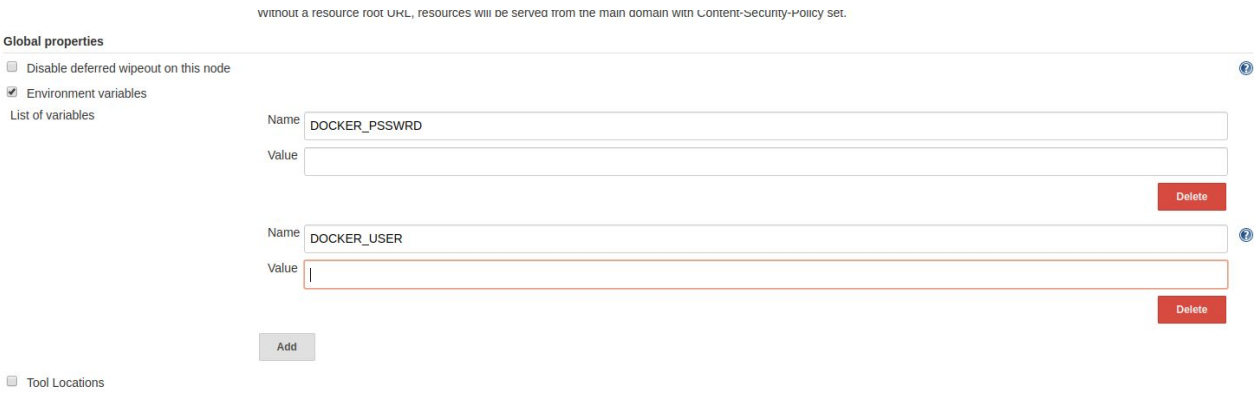
In

Jenkinsfile

```
pipeline {
    agent any
    stages {

        stage('--Mvn clean package--') {
                steps {
                    sh "mvn clean package deploy"
                    }
            }
            stage('--Build back-end--') {
                steps {
                    sh "docker build -t app-test ."
                    }
            }
        stage('--Deploy--') {
              steps {
                    sh "docker login -u ${env.DOCKER_USER} -p ${env.DOCKER_PSSWRD}"
                    sh "docker tag app-test alwinthomas/app-test"
                    sh "docker push alwinthomas/app-test"
                    }
              }
    }
}
```

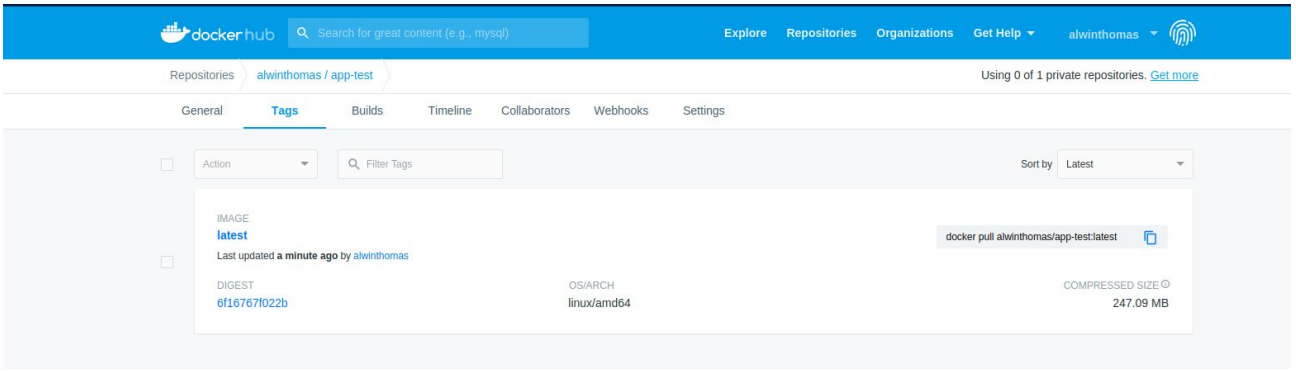
Jenkins> manage Jenkins > Config system > select environment variables> press add > type name and value as below for your dockerhub account



Apply save
Go into EC2 Jenkins instance and run the following commands:

```
sudo apt update
sudo apt install docker.io -y
sudo usermod -aG docker $USER
sudo chmod 777 /var/run/docker.sock
sudo systemctl enable docker
```

Build via Jenkins
Check Dockerhub for images



Testing and Production environments

**

→ Change Jenkinsfile to the following for the backend :

**

→ Change Jenkinsfile to the following for the frontend :

-launch new EC2 instance
→ run docker script

sudo apt update
sudo apt install docker.io -y
sudo usermod -aG docker $USER
sudo chmod 777 /var/run/docker.sock
sudo systemctl enable docker
→ then pull docker image from docker hub
→ containerise back-end

→ create a new EC2 instance for the front end

→ repeat earlier steps for front end inluding Jenkinsfile- remove mvn package steps and change build type to front end and push image to Docker Hub
→ pull docker image from docker hub
→ run docker script
→ containerise front-end
→ change nginx.config to IP address of new instance
→ stop front and back end containers

Selenium tests:

1. Create new repo for selenium tests
2. Run selenium tests in FE instance
3. In FE instance run:

sudo apt update
wget https://dl.google.com/linux/direct/google-chrome stable_current_amd64.deb
sudo dpkg -i google-chrome-stable_current_amd64.deb
rm google-chrome-stable_current_amd64.deb

Make sure chrome is running the same version as Driver
String Host in Constants should be IP/DNS name

run mvn test in FE instance to conduct selenium tests (ensure selenium tests are set to headless)

4. Adapt Jenkinsfile to run a job that creates selenium tests→ if tests pass ssh into production ECT

wget
http://3.11.84.155:8081/service/rest/v1/search/assets/download?repository=ea-hosted&group=com.bae.cookbook=cookbook-app&0.0.1 SNAPSHOT&sort=version&maven.extension=jar

start up two new instances for back-end and front-end production environments
Add Jenkinfile to master branch and change docker image name within the Jenkinsfile

SSH into instances

scp= use to copy files between two systems

1. scp -i "project.pem" front-end.pem ubuntu@3.11.97.214:/home/jenkins/
(This command moves front end key into Jenkins for access in Jenkinsfile)

copy over front-end key into Jenkins instance in order to add the ssh command from Jenkins job

2. secure copy over the back-end key into Jenkins instance in order to add the ssh command from Jenkins job
→ scp -i "project.pem " backend-end.pem ubuntu@3.11.97.214:/home/jenkins/

3. Additional ssh step to Jenkinsfile for frontend and back-end

stage('--test-deploy--') {
steps {

sh "ssh -T -i /home/jenkins/Project.pem
ubuntu@ec2-35-176-134-117.eu-west.compute.amazonaws.com ./docker-back-end.sh"
}
}

4. Test by running Jenkins build – fails build so enable host key verification and change credentials;

In /home/jenkins/ directory
run:

sudo chown jenkins:jenkins "key.pem"
sudo passwd jenkins
su jenkins
ssh -T -i /home/jenkins/back-end-RDS.pem
ubuntu@ec2-3-8-173-215.eu-west-2.compute.amazonaws.com
ssh -T -i /home/jenkins/front-end.pem
ubuntu@ec2-3-10-227-78.eu-west-2.compute.amazonaws.com
-T creates a new virtual terminal

Selenium test are run in FE instance rather than Jenkins because FE instance must be up and running before running selenium tests

BE EC2 instance Script for test environment

```
#!/bin/bash

sudo apt update

curl https://get.docker.com | sudo bash

sudo usermod -aG docker $(whoami)

sudo usermod -aG docker $USER

sudo chmod 777 /var/run/docker.sock

sudo systemctl enable docker

docker pull alwinthomas/app-backend:latest

docker run -d -p 9090:8089 --restart unless-stopped --name probook-app alwinthomas/app-
backend:latest
```

FE EC2 instance Scripts for test environment

Selenium script

```
#!/bin/bash

docker pull alwinthomas/app-frontend:latest

docker run -d -p 80:80 --restart unless-stopped --name probook-front-end alwinthomas/app-
frontend:latest

git clone https://github.com/AlwinThomaz/DevOps-Selenium.git --branch dev_v2 --single-branch

cd DevOps-Selenium

mvn test > /home/ubuntu/selenium-logs.txt
```

Install script

```
#!/bin/bash

sudo apt update

sudo apt install default-jdk -y

sudo apt install maven -y

curl https://get.docker.com | sudo bash

sudo usermod -aG docker $(whoami)

sudo usermod -aG docker $USER

sudo chmod 777 /var/run/docker.sock

sudo systemctl enable docker

wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb

sudo dpkg -i google-chrome-stable_current_amd64.deb

sudo apt install fonts-liberation libappindicator3-1 libcairo2 libcairo2 libgdk-pixbuf2.0-0
libgtk-3-0 libpango-1.0-0 libpangocairo-1.0-0 libxcursor1 libxss1 xdg-utils

sudo apt --fix-broken install

rm google-chrome-stable_current_amd64.deb
```

In total should be running 4 instances - two for the test environment and two for the production environment . Each has their own script that are run upon SSHing into the respective instance from Jenkins via the SSH stage in each Jenkinsfile.