Capstone project – Orbit Bank

Name: Kotian Rakshith Padmanabha

Email: rakshithpk21@gmail.com

Github link for the project: https://github.com/kotianrakshith/CapstoneProject1

Objective: To deploy a banking application on a Kubernetes cluster from Docker Hub.

Tools to use:

1. Jenkins
2. Github
3. Docker Hub
4. Ansible
5. Kubernetes

Description

Orbit Bank is one of the leading banking and financial service providers and is facing challenges in managing their monolithic applications and experiencing downtime during deployment. The company needs to develop an online banking application that provides private banks with a global accounting foundation, offering electronic banking services to all private banks, and enable private bank clients to carry out their daily transactions.

To address these issues, the company has decided to transition to a microservices architecture and implement a DevOps pipeline workflow using Jenkins, Ansible playbook, and Kubernetes cluster to deploy container on Docker Hub.

Task (Activities)

1. Create the Dockerfile, Jenkinsfile, Ansible playbook, and the source file of the static website and upload it on the GitHub repository
2. Create Jenkins pipeline to perform continuous integration and deployment for a Docker container
3. Set up Docker Hub
4. Set up Kubernetes cluster and configure deployment stage in the pipeline
5. Configure Ansible playbook to deploy container on Docker Host
6. Execute Jenkins build
7. Access deployed application on a Docker container

Steps performed:

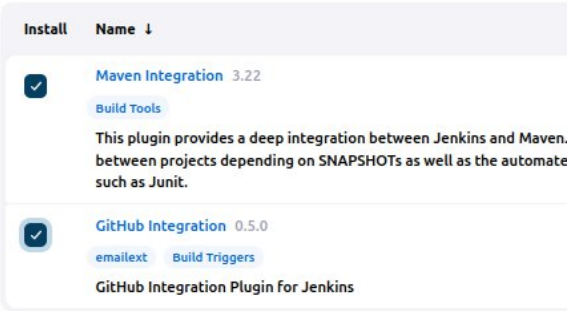
1. <u>Initial testing</u>

Before we use jenkins for the continous integration and deployment, let us just test the build of java application of source code using maven and use the jar file to test the creation of docker image manually

I have saved the source code provided by simplilearn in my github repo :

https://github.com/kotianrakshith/CapstoneProject1

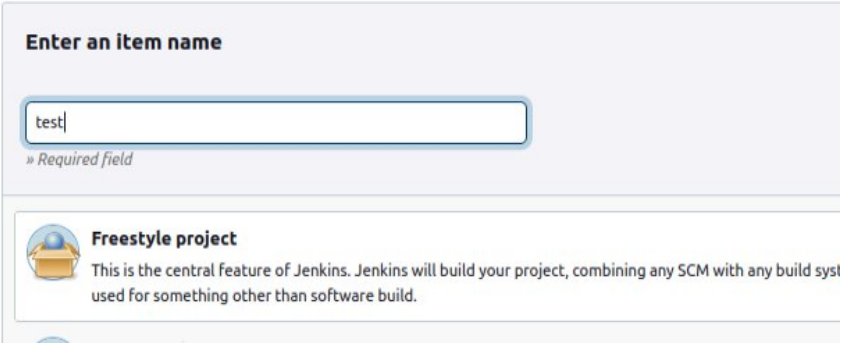
First we will add the required plugins in the jenkins



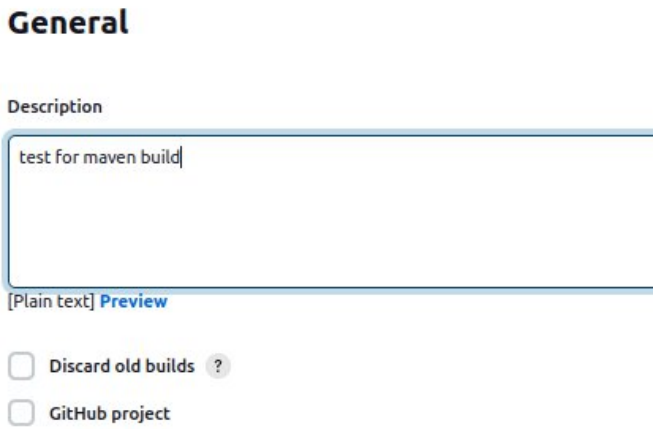
Ohers we can install later incase needed.(Please note that I have added many plugins in the jenkins for the project as we progress but are not documented.)

In the tools add Maven:

Now we create a new test freestyle project

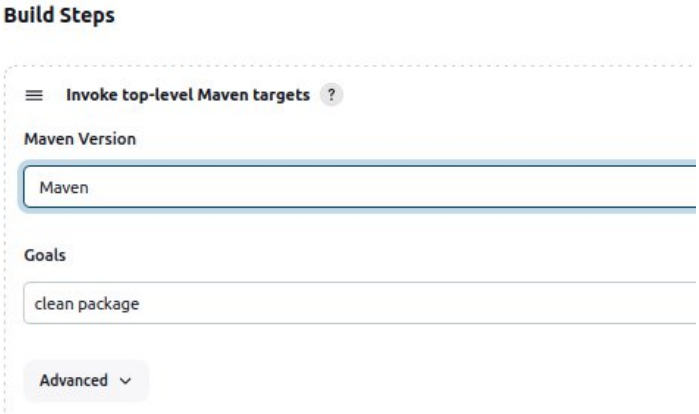


Give a general discription:

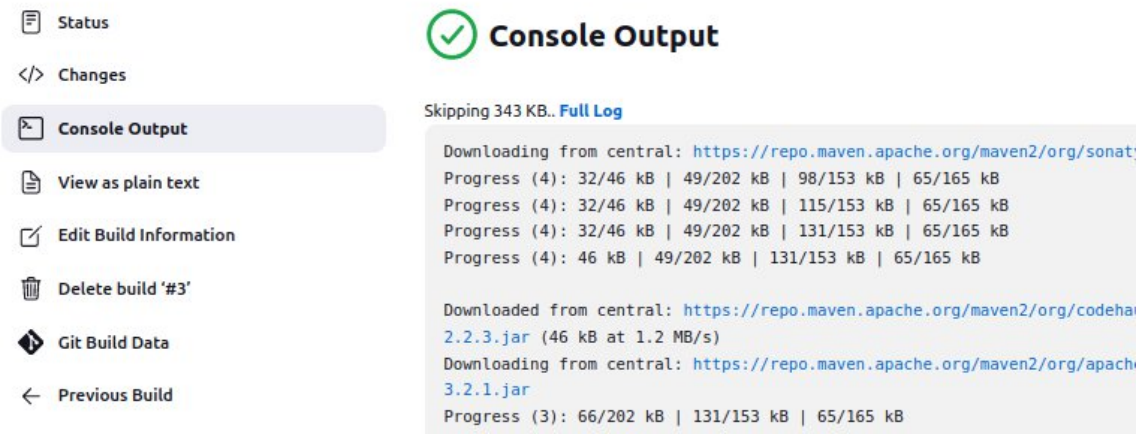


In the repository give the link:

Provide build steps:



Then we can save and build run, and we can see it has run succesfully



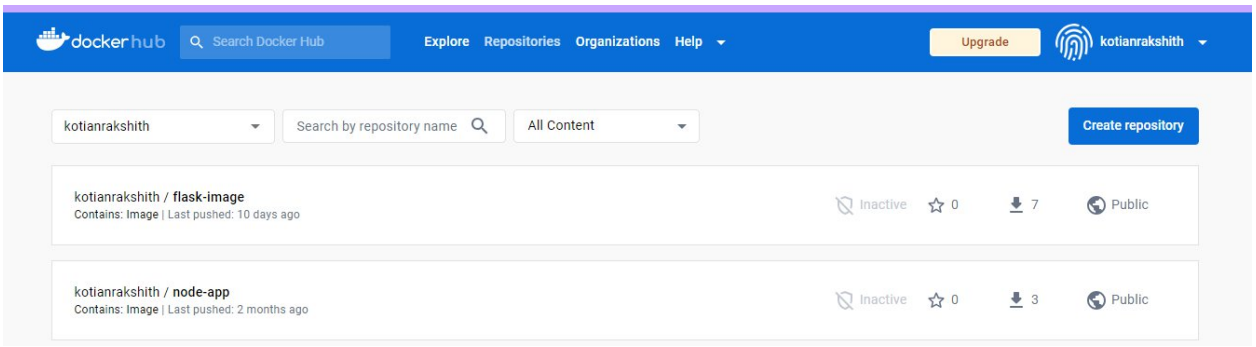
This means that build works correctly.

Now we can proceed with the other tests and configurations before we configure the full pipeline.

Jenkins file we will create in later step by testing each step one by one in pipeline

2. DockerHub Setup

Regarding Dockerhub, I already have a dockerhub account with id: kotianrakshith



So I will use the same account.

For the Dockerfile, I will use below code :

Dockerfile

```
FROM openjdk:8-jdk-alpine

COPY target/*.jar app.jar

EXPOSE 8989

ENTRYPOINT ["java","-jar","/app.jar"]
```

3. Setup kubernetes cluster:

We have three nodes with us, we will chose the system with jenkins installed as master and other two nodes as node 1 and node 2. We will rename it as master, worker node1 and worker node2:

Master:

```
labsuser@ip-172-31-54-86:~$ sudo hostnamectl set-hostname master.example.com
labsuser@ip-172-31-54-86:~$ exec bash
labsuser@master:~$
```

Node1:

```
labsuser@ip-172-31-3-73:~$ sudo hostnamectl set-hostname worker-node-1.example.com
labsuser@ip-172-31-3-73:~$ exec bash
labsuser@worker-node-1:~$
```

Node2:

```
labsuser@ip-172-31-12-234:~$ sudo hostnamectl set-hostname worker-node-2.example.com
labsuser@ip-172-31-12-234:~$ exec bash
labsuser@worker-node-2:~$
```

Now lets do docker configuration in all the three nodes:

```
  cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
        "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
—----------------------------------------------------------
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
sudo swapoff -a
```

```
labsuser@master:~$ cat <<EOF | sudo tee /etc/docker/daemon.json
> {
>   "exec-opts": ["native.cgroupdriver=systemd"],
>   "log-driver": "json-file",
>   "log-opts": {
> "max-size": "100m"
>   },
>   "storage-driver": "overlay2"
> }
> EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
"max-size": "100m"
  },
  "storage-driver": "overlay2"
}
labsuser@master:~$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
labsuser@master:~$ sudo systemctl daemon-reload
labsuser@master:~$ sudo systemctl restart docker
labsuser@master:~$ sudo swapoff -a
labsuser@master:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2023-07-04 05:40:39 UTC; 11s ago
TriggeredBy: ● docker.socket
```

(same output screen in all three nodes)

Now we will do master node initilaization using command:

```
labsuser@master:~$ sudo kubeadm init
I0704 05:42:17.021279  158338 version.go:255] remote version is m
[init] Using Kubernetes version: v1.23.17
[preflight] Running pre-flight checks
```

From here we will copy the link it provides for the worker node initialization:

```
Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.54.86:6443 --token iwsmnp.w5i63e300kus31m1 \
        --discovery-token-ca-cert-hash sha256:d391c5bf37c5d229d0117b2df4eb3c47dc4e1ffa0c8904592d46f5d607ac8d16
labsuser@master:~$
```

Then we will proceed to configure master with changing config permissions:

```
mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

cat ~/.kube/config
```

With the last cat step you should be able to can view config file:

```
labsuser@master:~$   cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMva
JME1Gb1hEVE16TURjd01UQTFOREkwTUZvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pD
VDbVBJRWtPTE1QSk1hZmNJL0w1blFXN0kKY0l2b0JLNExSN1E2UGpRUndxK1JUODRzRktFYXE4N25
lRSWjg5SGF4Z2JVCnhIQzQwTkdycG8rZHRRZUpzMU9ucDM1bnluRFNzTVpGSDVEa2M0dlZlZ0lVaE
Wk5sdHI2Q25BbUwyQldRMTNzQ0F3RUFBYU5aTUZjd0RnWURWUjBQQVFIL0JBUURBZ0trTUE4R0ExV
WMFpYTXdEUVlKS29aSWh2Y05BUUVMQlFBRGdnRUJBSmozQzhoa3hMdnJwSW1FMnpFNgpqMEN6cit5
Z1VVlsSE8xMDYwVkw1NTJETlFVeVNsQU5pYkszSXRrOGpWNUEKV2QzOU40ZTAranhELzZHa0NPRXl
1pjcUVYMU9BdlhOeTRBaVZDSUZSL2JLCmNOM0ZzTlhHcmtac2hlYlNKVVpIdmZOK3JtS01KQk5xZk
    server: https://172.31.54.86:6443
  name: kubernetes
contexts:
```

Then we will install a CNI, here we are chosing calico

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml

```
labsuser@master:~$ kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
```

Now we can do the worker node initialization using the command we copied:

Worker node 1:

```
labsuser@worker-node-1:~$ sudo kubeadm join 172.31.54.86:6443 --token iwsmnp.w5i63e300kus31m1 \
>         --discovery-token-ca-cert-hash sha256:d391c5bf37c5d229d0117b2df4eb3c47dc4e1ffa0c8904592d46f5d607ac8d16
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
W0704 05:48:05.644060    9044 utils.go:69] The recommended value for "resolvConf" in "KubeletConfiguration" is: /run/systemd/
resolv.conf
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

Worker node 2:

```
labsuser@worker-node-2:~$ sudo kubeadm join 172.31.54.86:6443 --token iwsmnp.w5i63e300kus31m1 \
>         --discovery-token-ca-cert-hash sha256:d391c5bf37c5d229d0117b2df4eb3c47dc4e1ffa0c8904592d46f5d607ac8d16
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
W0704 05:48:17.624346    8775 utils.go:69] The recommended value for "resolvConf" in "KubeletConfiguration" is: /run
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

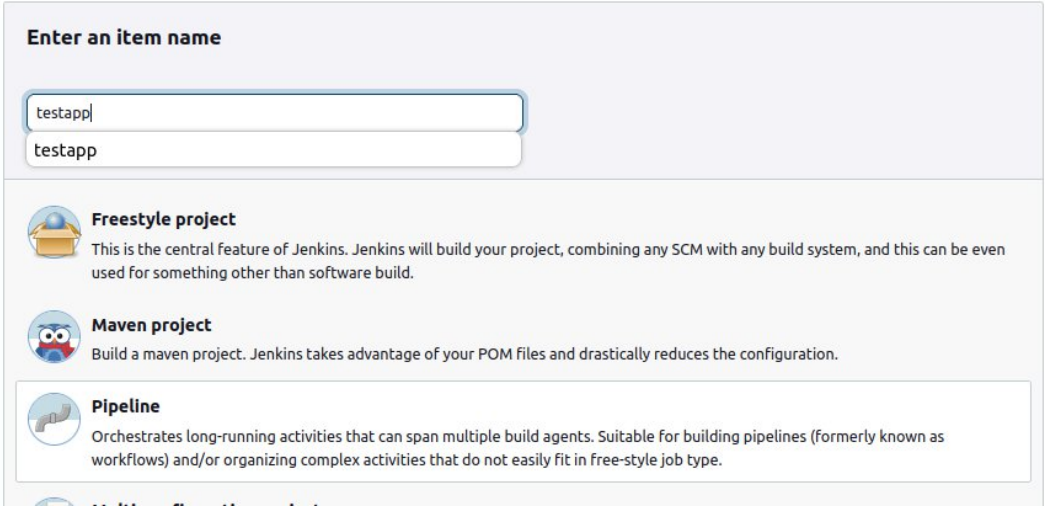
Now in the worker node if we run command ‘kubectl get nodes’ we should see all three nodes:

```
labsuser@master:~$ kubectl get nodes
NAME                      STATUS   ROLES                  AGE     VERSION
master.example.com        Ready    control-plane,master   6m46s   v1.23.4
worker-node-1.example.com Ready    <none>                 89s     v1.23.4
worker-node-2.example.com Ready    <none>                 78s     v1.23.4
labsuser@master:~$
```

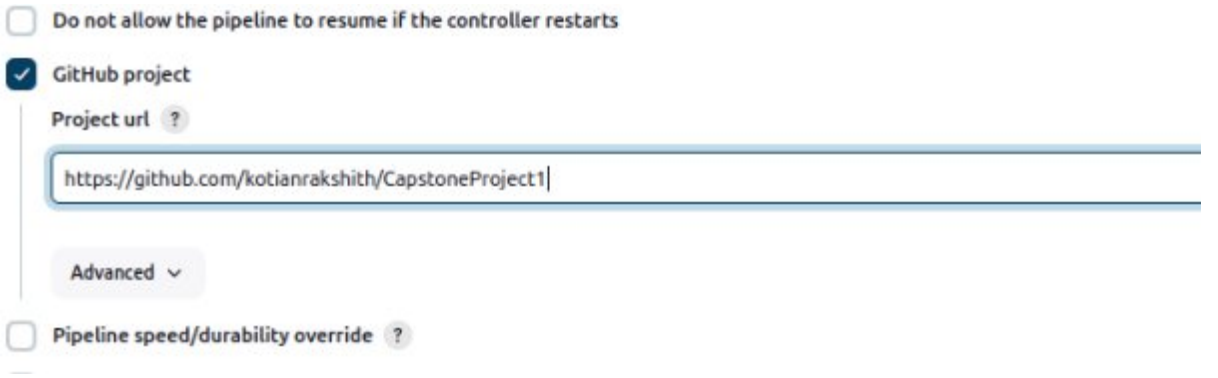
4. Create Jenkins pipeline script stage by stage

Now lets create the jenkins file by creating test pipline with each step one by one:

As our build is already sucessfull lets try build and creating docker image

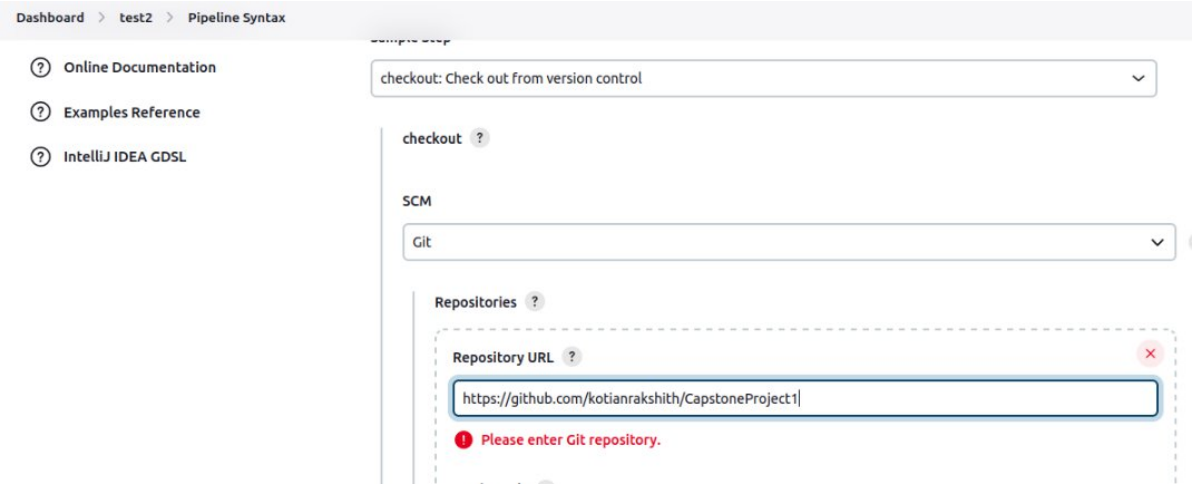


We will chose this as a github project and we will give our github repository link:



For the first stage we will checkout the git and use mvn clean install to build the jar file

To get the checkout script we are using pipeline syntax to populate the script



Please note that I'm using terminology testapp in the script which I will later change for our actual build

With this step we can add below script:

```
stage('Build Maven'){
steps{
checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/kotianrakshith/CapstoneProject1']])
sh 'mvn clean install'
```

```
        }
    }
```

For the docker image build we are using below script:

```
        stage('Build Docker Image'){
        steps{
            script{
                sh 'docker build -t kotianrakshith/testapp .'
            }
        }
    }
```

For now we will test this in the pipeline to see if it works till here:

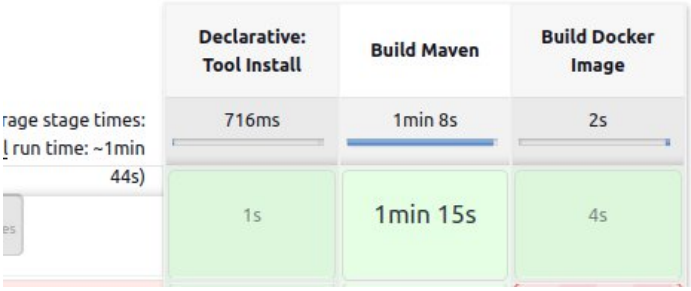
As we had issues with docker build, I have run the below command and reboot the system to give jenkins permission to run docker command:

```
sudo usermod -a -G docker jenkins
```

This solves the issue and maven build and docker image build is completed successfuly

```
[INFO] --- install:2.5.2:install (default-install) @ bank-app ---
[INFO] Installing /var/lib/jenkins/workspace/testapp/target/bank-app-1.0.0.jar to /var/lib/jenkins/.m2/repository
/com/coding/exercise/bank-app/1.0.0/bank-app-1.0.0.jar
[INFO] Installing /var/lib/jenkins/workspace/testapp/pom.xml to /var/lib/jenkins/.m2/repository/com/coding
/exercise/bank-app/1.0.0/bank-app-1.0.0.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  55.761 s
[INFO] Finished at: 2023-07-05T17:43:07Z
[INFO] ------------------------------------------------------------------------
```

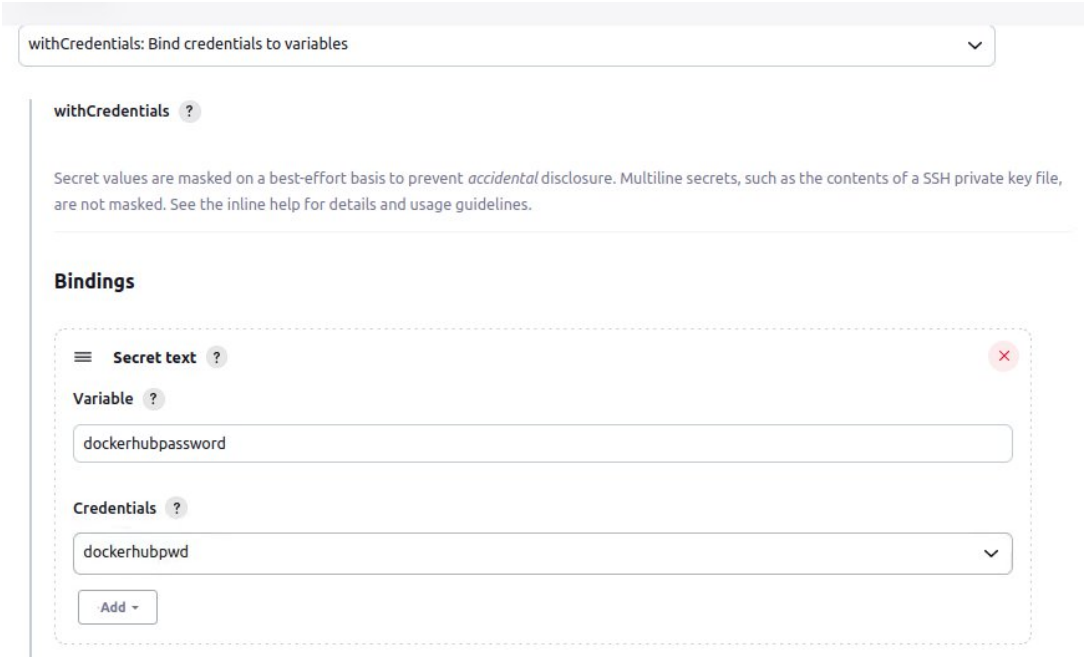
```
 ---> Running in c8c54b6ddd56
Removing intermediate container c8c54b6ddd56
 ---> 3d3f0b9f7e35
Successfully built 3d3f0b9f7e35
Successfully tagged kotianrakshith/testapp:latest
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

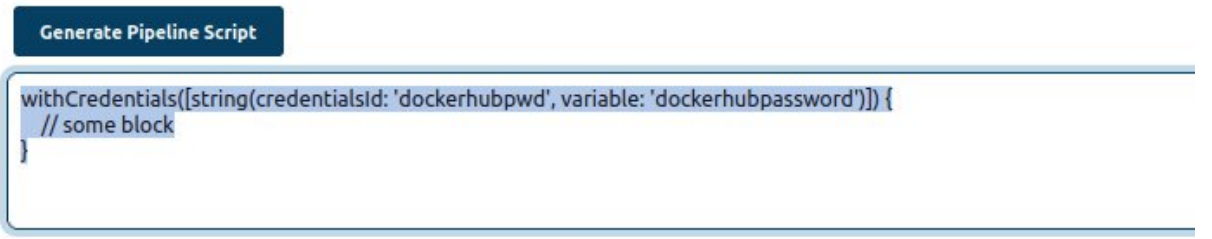


Now I will add the next stage, that is uploading the docker image to the docker hub

As we need password to login I will be creating access token instead of the password.

Usig pipeline variable I'm binding the password to a variable





With this we can write the script as below:

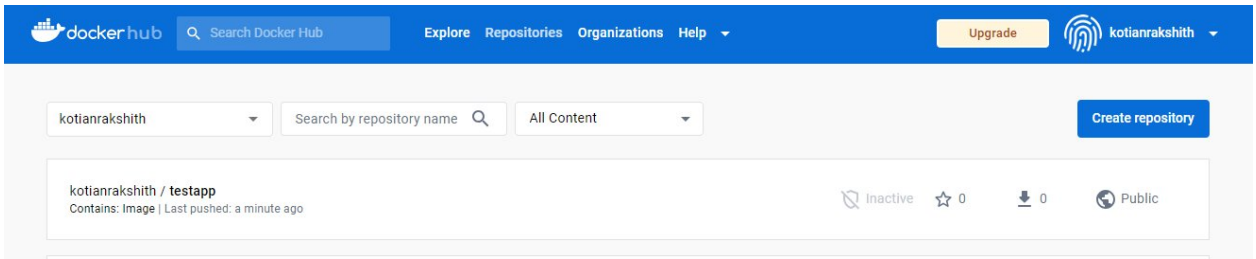
```
stage('Push Docker Image to Dockerhub'){
    steps{
        script{
            withCredentials([string(credentials Id: 'dockerhubpwd', variable: 'dockerhubpassword')])
{
            sh 'docker login -u kotianrakshith -p ${dockerhubpassword}'

            sh 'docker push kotianrakshith/testapp'
            }
        }
    }
}
```

```
18          }
19      }
20      stage('Push Docker Image to Dockerhub'){
21          steps{
22              script{
23                  withCredentials([string(credentialsId: 'dockerhubpwd', variable: 'dockerhubpassword')]) {
24                  sh 'docker login -u kotianrakshith -p ${dockerhubpassword}'
25
26                  sh 'docker push kotianrakshith/testapp'
27                  }
28              }
29          }
30      }
31  }
32 }
```

The pipeline build was successfull till this stage:

Declarative: Tool Install	Build Maven	Build Docker Image	Push Docker Image to Dockerhub
605ms	1min 7s	2s	16s
273ms	1min 6s	5s	16s

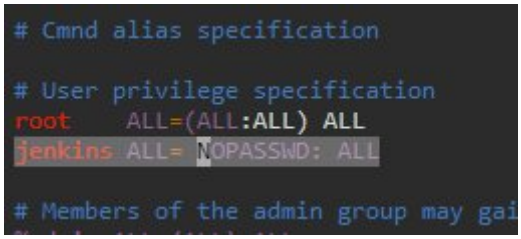
In the dockerhub we can see the testapp image added:



Now the last build stage is executing the ansible playbook to deploy the container in kubernetes using the docker image.

For the authorization we will add the jenkins user in the file /etc/sudoers

```
sudo vim /etc/sudoers
```



As our initial ansible script failed we are using pipline syntax to pass the kube config file in to the jenkins workspace so it can have acccess to run the kubernes commands.

We have modified ansible workbook as below:

```
---
- name: Deploy Kubernetes Deployment and Service
  hosts: localhost
  tasks :
```

```
  - name: Deleting Deployment of testapp

    command: "kubectl delete -f kubewebapp.yaml --kubeconfig=kubeconfig --context=kubernetes-admin@kubernetes"

    ignore_errors: true

  - name: Performing Deployment of testapp

    command: "kubectl apply -f kubewebapp.yaml --kubeconfig=kubeconfig --context=kubernetes-admin@kubernetes"
```

We have added the kuberntes deployment as a file kubewebapp.yaml in github:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: testapp
  name: testapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: testapp
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: testapp
    spec:
      containers:
      - image: docker.io/kotianrakshith/testapp
        name: testapp
        resources: {}
status: {}


---
apiVersion: v1
```

```
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: testapp
  name: testapp
spec:
  ports:
  - port: 8989
    protocol: TCP
    targetPort: 8989
    nodePort: 32000
  selector:
    app: testapp
  type: NodePort
status:
  loadBalancer: {}
```

Now we use pipeline syntax to get the syntax with kube config file:

With this I have written the below stage script below:

```
stage('Execute Ansible Playbook'){
        steps{
            withCredentials([kubeconfigContent(credentialsId: 'Kubernetes', variable:
'KUBECONFIG_CONTENT')]) {
                sh '"echo "$KUBECONFIG_CONTENT" > kubeconfig "'
                sh 'ansible-playbook kubernetesDeploy.yaml'
                sh 'rm kubeconfig'

            }
        }
    }
```

We can see that all our test is complete

Declarative: Tool Install	Build Maven	Build Docker Image	Push Docker Image to Dockerhub	Execute Ansible Playbook
219ms	59s	3s	12s	6s
120ms	1min 1s	4s	11s	8s

And that test app is deployed in kubernets

```
labsuser@master:~$ k get deploy
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
testapp   1/1     1            1           102s
labsuser@master:~$ k get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          46h
testapp      NodePort    10.110.114.118   <none>        8989:32000/TCP   108s
```

5. Executing complete Jenkins pipelin Build

Now let us combine all in a Jenkins file and give correct terminology in all the file:

We are naming the app as orbitbankapp.

We have corrected all the files with the correct name and uploaded in github. Below are the links:

Jekins file link: https://github.com/kotianrakshith/CapstoneProject1/blob/main/Jenkinsfile

Dockerfile link: https://github.com/kotianrakshith/CapstoneProject1/blob/main/Dockerfile

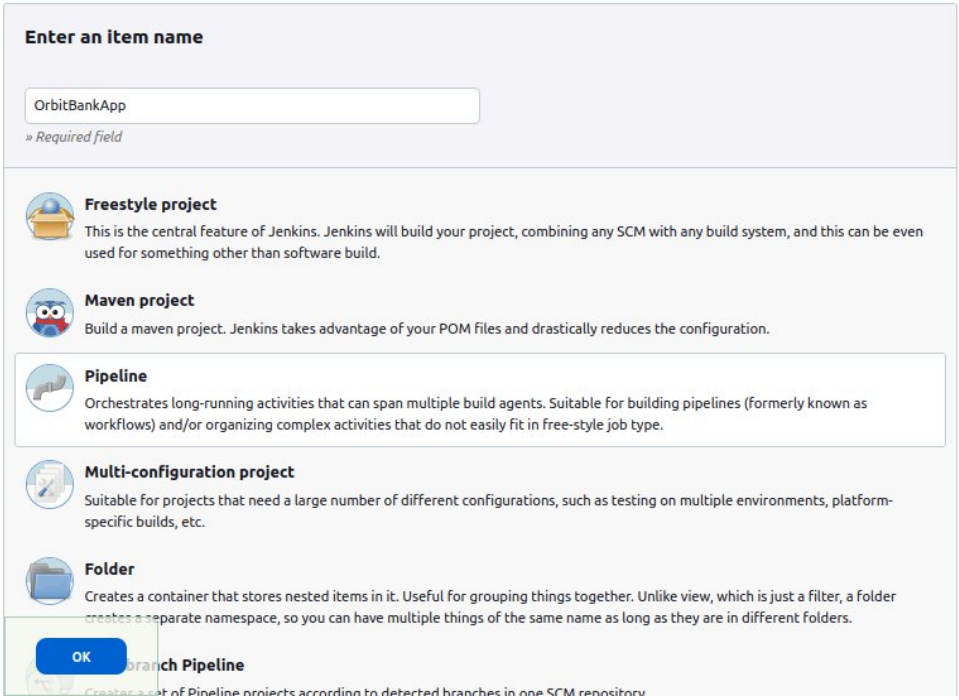
Kubernets file link:
https://github.com/kotianrakshith/CapstoneProject1/blob/main/kubewebapp.yaml

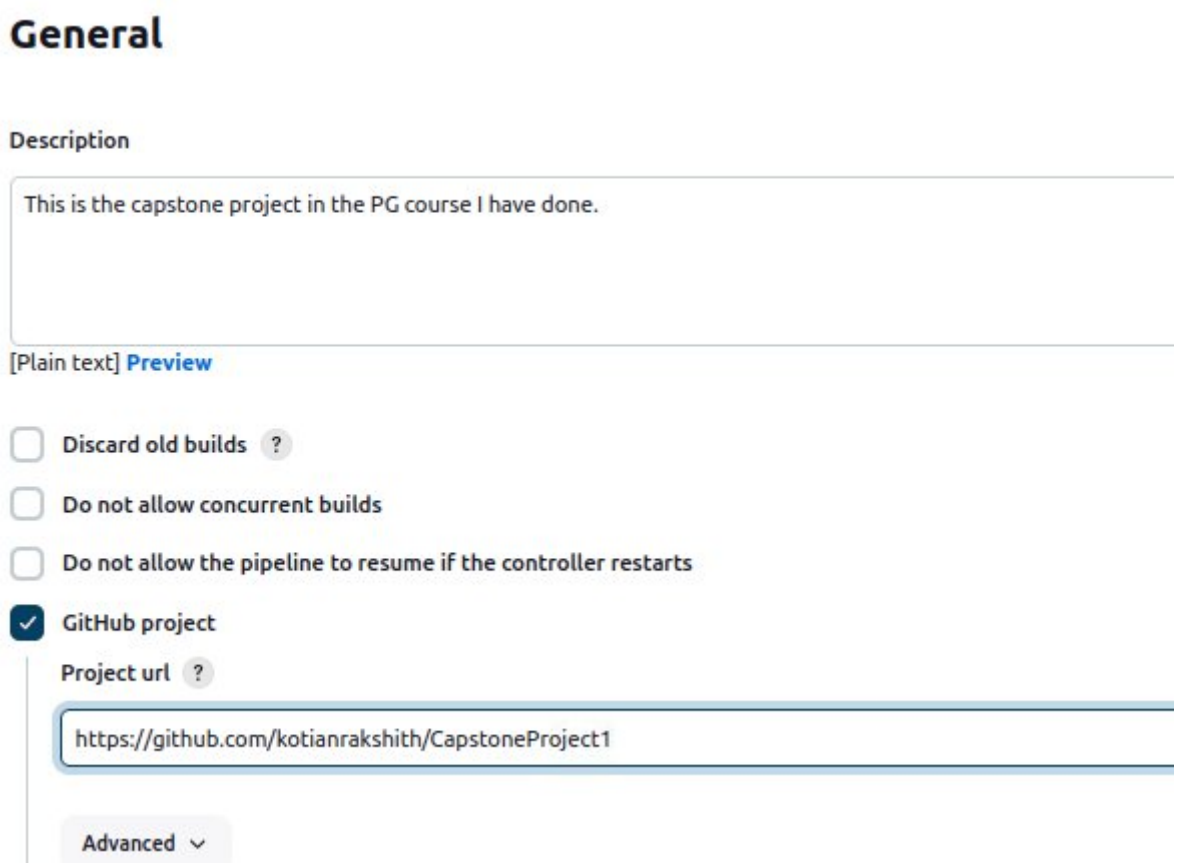
Ansible playbook link:
https://github.com/kotianrakshith/CapstoneProject1/blob/main/kubernetesDeploy.yaml

Total Github link:

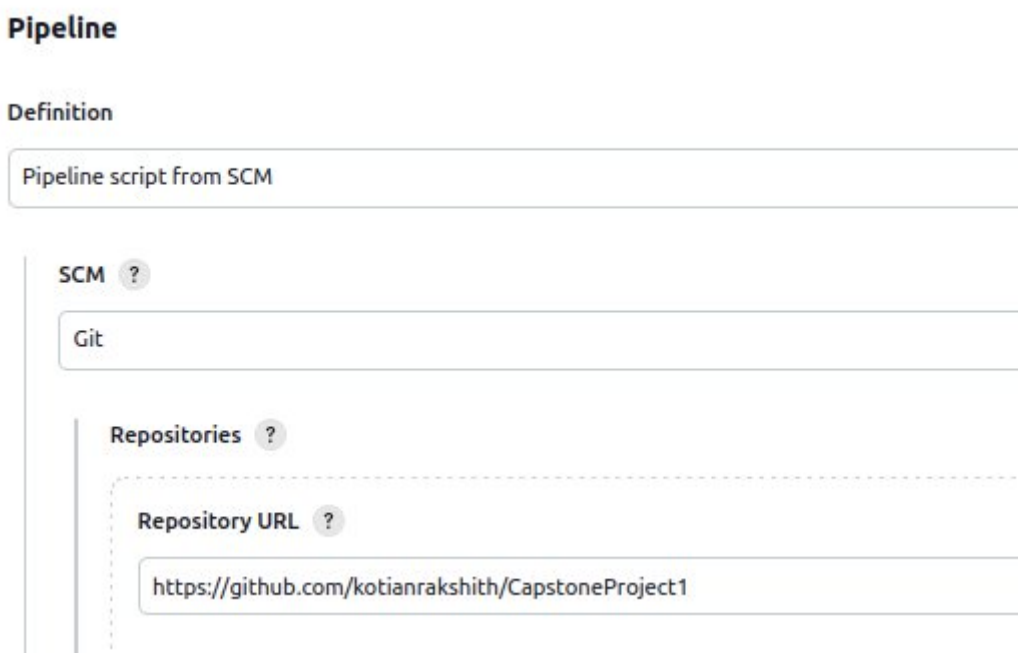
https://github.com/kotianrakshith/CapstoneProject1

Now we have created Jenkins file we can create our actual project pipeline using the file from Github:

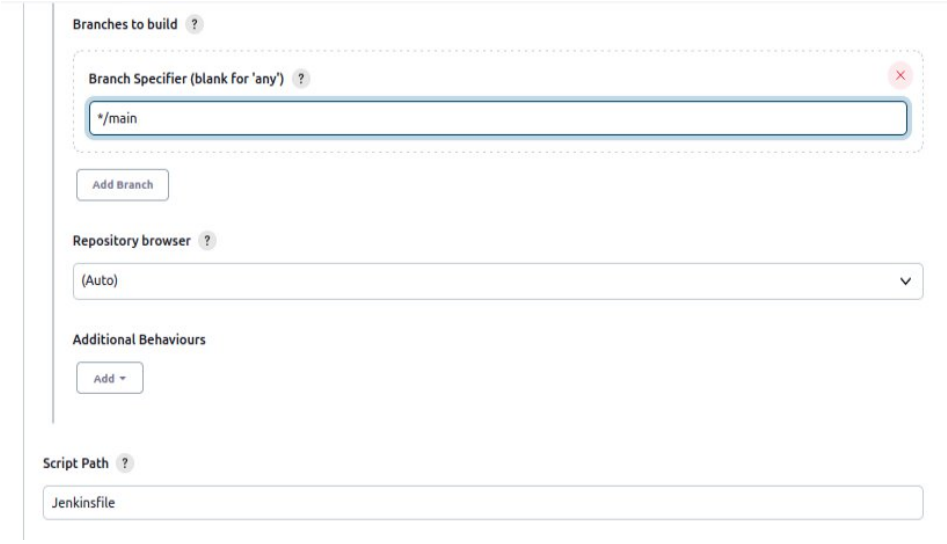




In the pipeline we are choosing pipeline script from SCM:



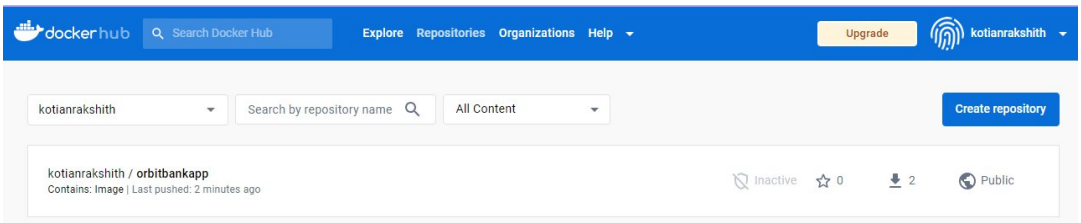
Give the correct branch name and Jenkins file name and save



Once saved you can build the pipeline:

As we have tested all the script before it should work as expected and build correctly.

Now we can check dockerhub if the image has been uploaded:



We can see our orbitbank app image

Now we can check in kubernetes if the deployment and service are present:

```
labsuser@master:~$ k get deploy
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
orbitbankapp   1/1     1            1           2m38s
labsuser@master:~$ k get svc
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP          47h
orbitbankapp   NodePort    10.104.50.227   <none>        8989:32000/TCP   2m41s
labsuser@master:~$
```

As we can see out app in the kubernets is running and service is also exposed in the nodeport 32000

6. Checking the deployment

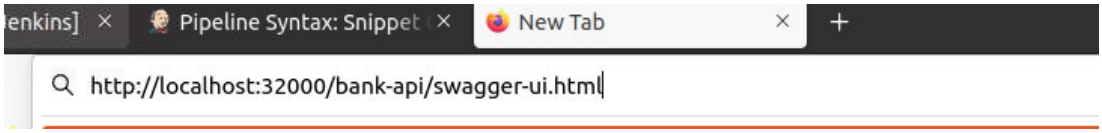
As the instruction provided with the source code we will use the below url to check the app:

http://localhost:<port>/bank-api/swagger-ui.html

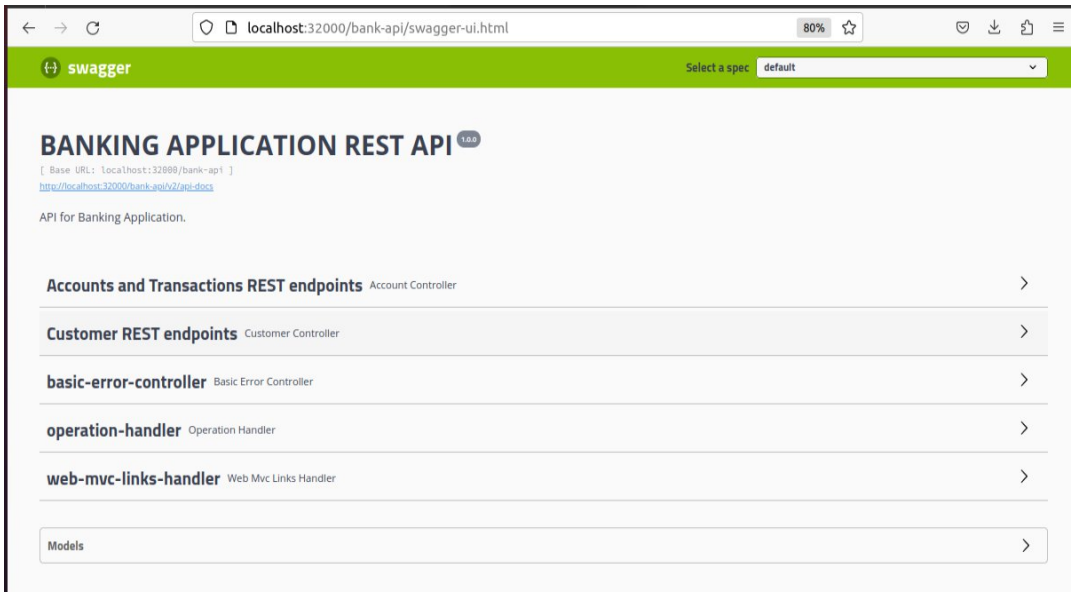
Here as our node port is 32000

We will use the below url to check

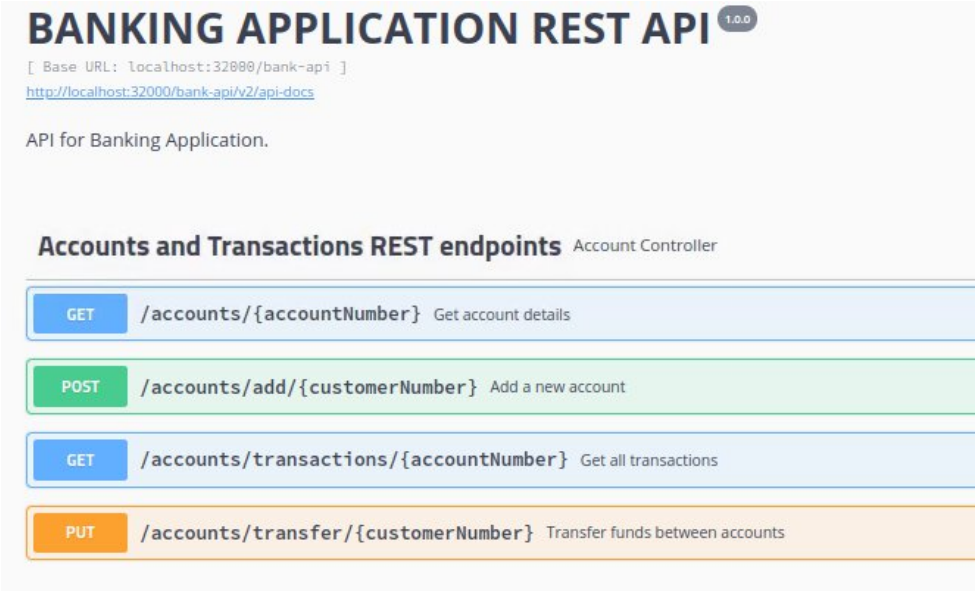
http://localhost:32000/bank-api/swagger-ui.html

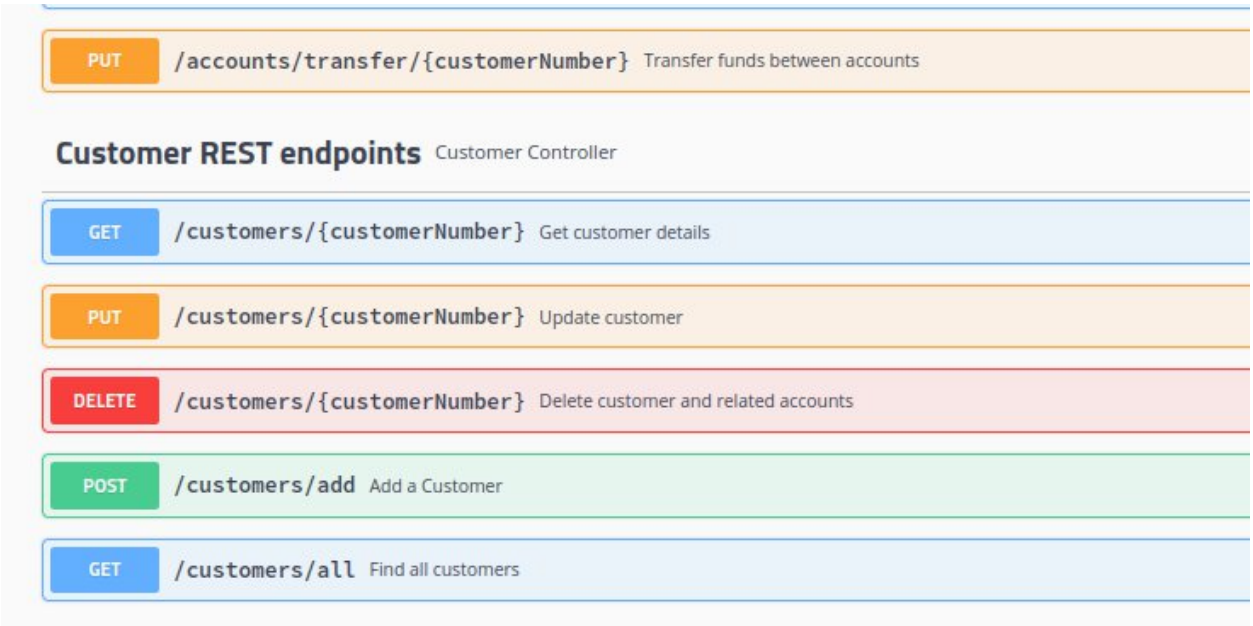


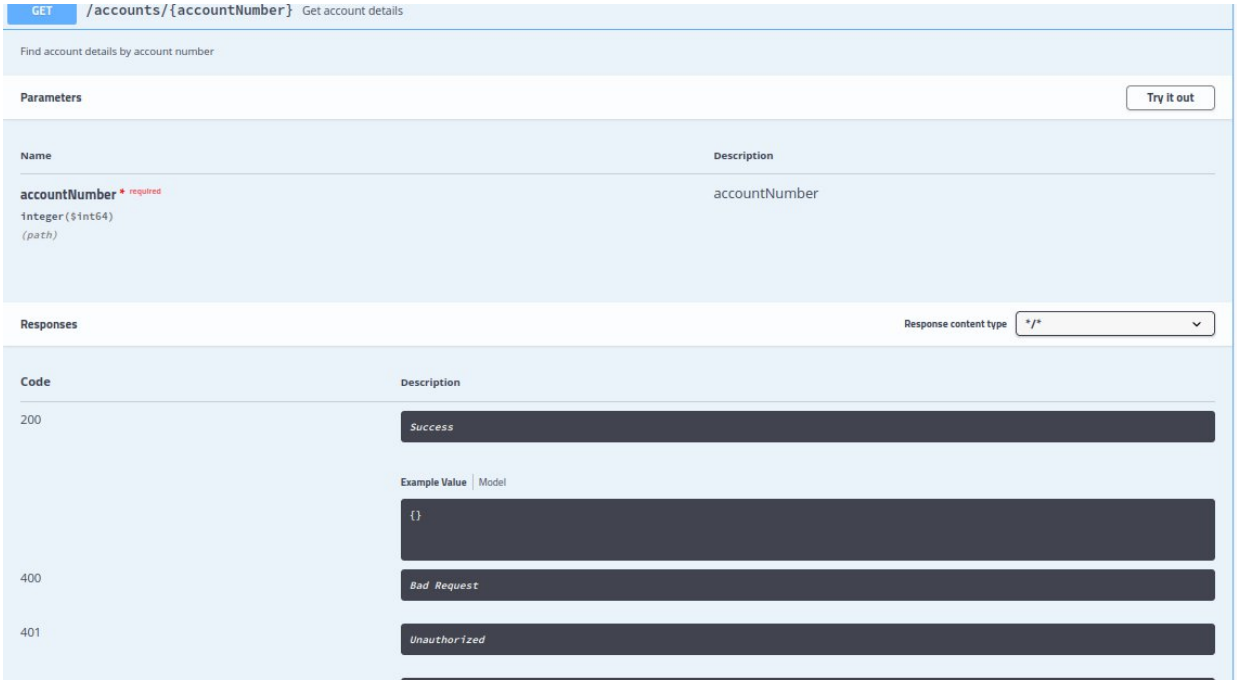
As we can see application is loading correctly:



And we are able to navigate:







So that concludes the project, we can improve on this project by making this an automated build by using poll scm or by using webhooks so it will run whenever there is a build made. But you can also click on build now whenever there is a change done and it should deploy the updated application to the kubernetes.