CONTENTS

1. Setup and develop the React application.

2. Setup docker on AWS and Dockerize the react application.

3. Kubernetes Cluster Creation.

4. Deploy react application to Kubernetes Cluster.

**SETUP & DEVELOP THE REACT APPLICATION**

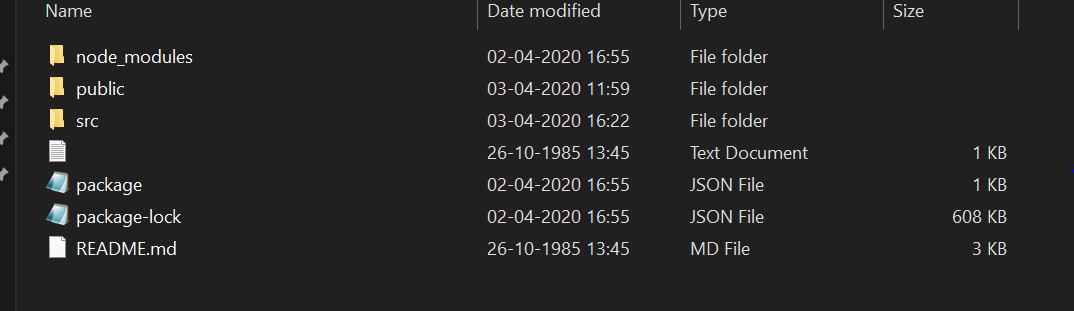
**Download & Install Nodejs :**

1. Go to the site <https://nodejs.org/en/download/> and download the necessary binary files.
2. Double click on the downloaded .msi file to start the installation. Click the Run button on the first screen to begin the installation. Accept the default components during the installation.
3. After the installation is successful, check the node and npm version using below commands.
4. node -v 🡪check node version
5. npm -v 🡪 check npm version

**Create the React application:**

React application can be easily created by using the create-react-app. We can do it as below

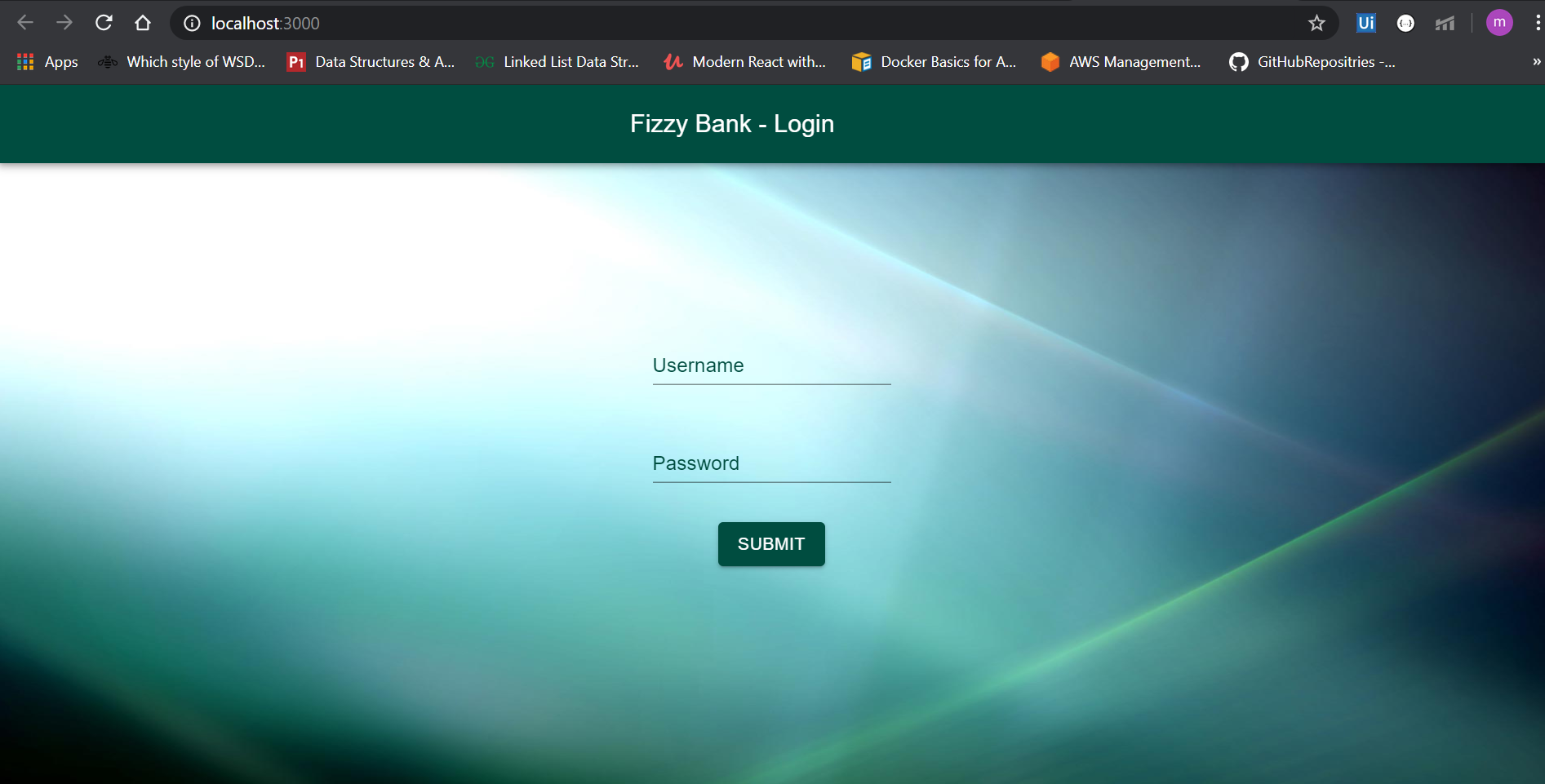
* npm install -g create-react-app
* create-react-app Pentagon



The main application structure is created as shown above and we can create additional files as per the application requirement. Once the application development is done we can run the react application with the below command

* cd Pentagon
* npm start

Once the application is started we can view it in the browser at <http://localhost:3000(3000> is default port where react application runs) as shown below



You can find all of the application code in git repository <https://github.com/GitHubRepositries/Pentagon>

**SETUP DOCKER ON AWS AND DOCKERIZE THE REACT APPLICATION**

**Create the AWS account and EC2 instance:**

Go to the link <https://aws.amazon.com/> and create a free tier account. The free tier account is valid for 12 months.

**To launch the EC2 instance and mount an EFS file system**

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose Launch Instance.
3. Choose an Amazon Machine Image (AMI), find an Amazon Linux AMI at the top of the list and choose Select.
4. Choose an Instance Type, choose Next: Configure Instance Details.
5. Configure Instance Details, provide the following information:

* For Network, choose the entry for the same VPC that you noted when you created your EFS file system in Step 1: Create Your Amazon EFS File System.
* For Subnet, choose a default subnet in any Availability Zone.
* For File systems, make sure that the EFS file system that you created in Step 1: Create Your Amazon EFS File System is selected. The path shown next to the file system ID is the mount point that the EC2 instance will use, which you can change. Choose Add to user data to mount the file system when the EC2 is launched.
* Under Advanced Details, confirm that the user data is present in User data.

1. Choose Next: Add Storage.
2. Choose Next: Add Tags.
3. Name your instance and choose Next: Configure Security Group.
4. Configure Security Group, set Assign a security group to Select an existing security group. Choose the default security group to make sure that it can access your EFS file system.
5. You can't access your EC2 instance by Secure Shell (SSH) using this security group. SSH access isn't required for this exercise. To add access by SSH later, you can edit the default security and add a rule to allow SSH. Or you can create a new security group that allows SSH. You can use the following settings to add SSH access:
   * + Type: SSH
     + Protocol: TCP
     + Port Range: 22
     + Source: Anywhere 0.0.0.0/0
6. Choose Review and Launch.
7. Choose Launch.
8. Select the check box for the key pair that you created, and then choose Launch Instances.

Connect to the above created EC2 instance by SSH using PUTTY and then download and install node, git and docker into the ec2 instance.

**Install Docker:**

* sudo yum update -y
* sudo yum install -y docker
* sudo service docker start

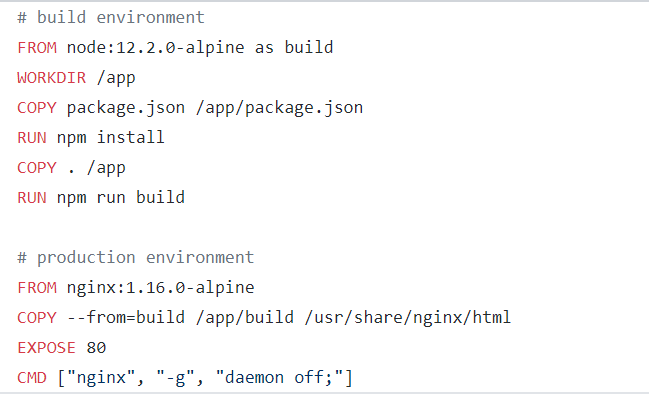
**Install Node:**

* curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
* . ~/.nvm/nvm.sh
* nvm install node --- install latest version, specify version if you need other version
* node -e "console.log('Running Node.js ' + process.version)"

**Install git:**

* sudo yum update -y
* sudo yum install git -y
* git version

Create a docker file for the react application so we can dockerize it. Please find the docker file below..



To build the image using the docker file, go to directory where the docker file is and run below command.

* docker build -t react-app:1111 .

Run a container using the image build above using below command

* docker run -p 8080:80 react-app:1111

We can access the react application by accessing the link as http://<instance-ip>:8080/ .

**KUBERNETES CLUSTER CREATION**

1. Launch Linux EC2 instance in AWS.

2. Create and attach IAM role to EC2 Instance.

Kops need permissions to access

S3

EC2

VPC

Route53

Autoscaling

etc..

3. Install Kops on EC2

curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d '"' -f 4)/kops-linux-amd64

chmod +x kops-linux-amd64

sudo mv kops-linux-amd64 /usr/local/bin/kops

4. Install kubectl

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

5. Create S3 bucket in AWS

S3 bucket is used by kubernetes to persist cluster state, lets create s3 bucket using aws cli **Note:** Make sure you choose bucket name that is uniqe accross all aws accounts

aws s3 mb s3://reactsample.in.k8s --region <<region-of-kops-machine>>

6. Create private hosted zone in AWS Route53

1. Head over to aws Route53 and create hostedzone
2. Choose name for example (javahome.in)
3. Choose type as privated hosted zone for VPC
4. Select default vpc in the region you are setting up your cluster
5. Hit create

7 Configure environment variables.

Open .bashrc file

vi ~/.bashrc

Add following content into .bashrc, you can choose any arbitary name for cluster and make sure buck name matches the one you created in previous step.

export KOPS\_CLUSTER\_NAME=javahome.in

export KOPS\_STATE\_STORE=s3://javahome.in.k8s

Then running command to reflect variables added to .bashrc

source ~/.bashrc

8. Create ssh key pair

This keypair is used for ssh into kubernetes cluster

ssh-keygen

9. Create a Kubernetes cluster definition.

kops create cluster \

--state=${KOPS\_STATE\_STORE} \

--node-count=2 \

--master-size=t2.micro \

--node-size=t2.micro \

--zones=<<instance-zone>> \

--name=${KOPS\_CLUSTER\_NAME} \

--dns private \

--master-count 1

10. Create kubernetes cluster

kops update cluster --yes

Above command may take some time to create the required infrastructure resources on AWS. Execute the validate command to check its status and wait until the cluster becomes ready

kops validate cluster

For the above above command, you might see validation failed error initially when you create cluster and it is expected behaviour, you have to wait for some more time and check again.

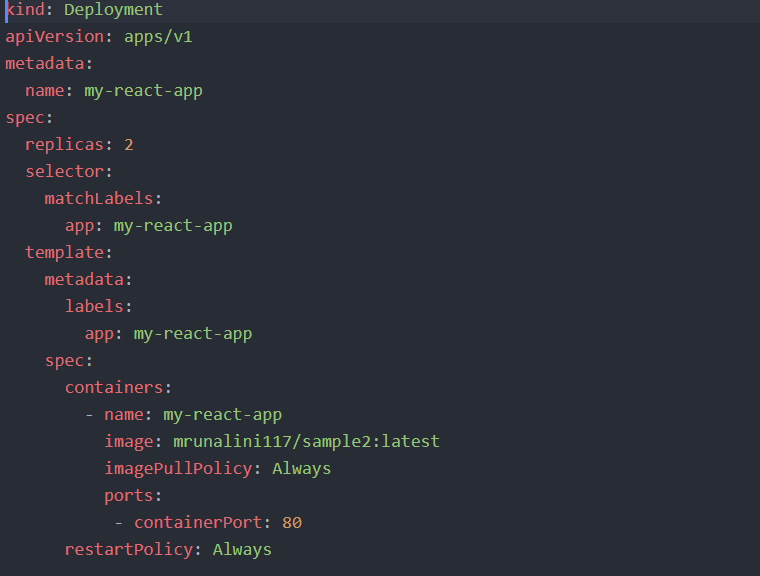
11. To connect to the master

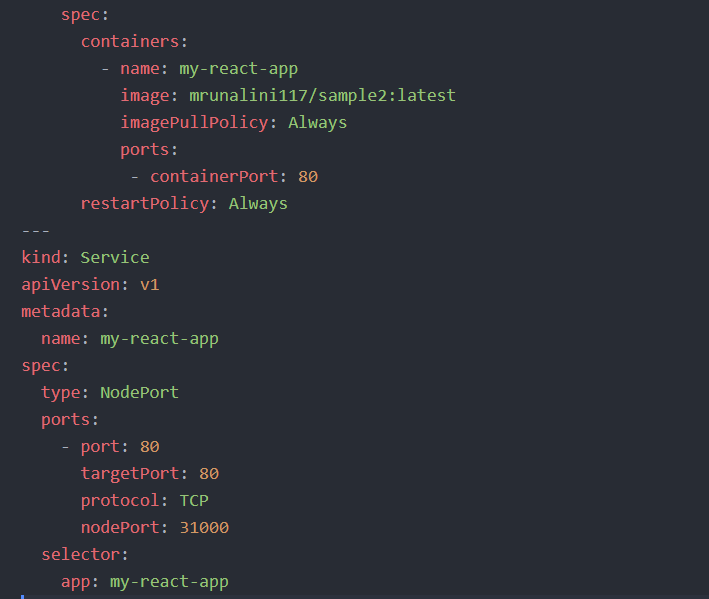
ssh admin@api.javahome.in

**DEPLOY APPLICATION TO KUBERNETES CLUSTER**

1. **Deploy using kubectl:**

To deploy the react application onto Kubernetes cluster we need to create the resources on Kubernetes cluster. We need to specify the type and all other configuration in a yaml file and run file on the Kubernetes cluster to deploy the application. Create a file deploy.yaml with below contents…





Login into Kubernetes master from the kops-machine using ssh and run the below command.

* Kubectl apply -f deploy.yaml

Running the above command creates pods, deployment and a service using which we can access the application from browser. We can check them with below commands..

* Kubectl get pods
* Kubectl get deployments
* Kubectl get services

The application is now up and running and we can access it using http://<<master-ip>>:31000/ as 31000 is specified as node port.

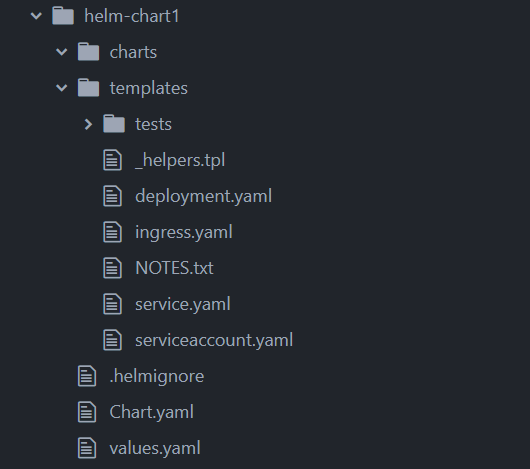
1. **Deploy using helm-charts:**

**Install helm:**

* curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
* chmod 700 get\_helm.sh
* ./get\_helm.sh
* helm version

To create the initial helm chart structure, run the below command

* helm create helm-chart1



The above screenshot shows the structure of the helm-chart created by above command. Changes needs to be made as per the application needs. Then we can check the helm-chart created by running the below command

* helm lint helm-chart

we can execute a dry-run to create the resources from the chart,

* helm install <<release-name>> helm-chart1/ --dry-run --debug

Then we can run the above command without –dry-run flag for the actual deployment

* helm install <<release-name>> helm-chart1/ --dry-run --debug

Verify the helm release that has been created by using the “helm ls” command which lists the release details. We can access the application in browser using http://<<master-ip>>:31000 .

**BUILDING DEPLOYMENT PIPELINE**

Create another ec2 Instance and install docker as mentioned above. We will be using this to setup Jenkins.

Install Jenkins:

1. Update the yum package

sudo yum update -y

1. Check if java is installed, if not install

java -version

sudo yum install java-1.8.0

To check and select one out of multiple java versions available

sudo /usr/sbin/alternatives --config.java

1. Download latest Jenkins code packages

sudo wget -O /etc/yum.repos.d/jenkins.repo

<http://pkg.jenkins.io/redhat/jenkins.repo>

1. Import a key file from Jenkins-CI to enable instllation from the package

sudo rpm --import <http://pkg.jenkins.io/redhat/jenkins.io.key>

1. Install Jenkins

sudo yum install jenkins

1. Start Jenkins

sudo service jenkins start

1. Access Jenkins server using the public DNS of your ec2 on port(8080)

http://{ec2-public-dns}:8080

1. To start jenkins on a diff port, update port number in /etc/sysconfig/Jenkins
2. To fetch initial admin password

sudo su -

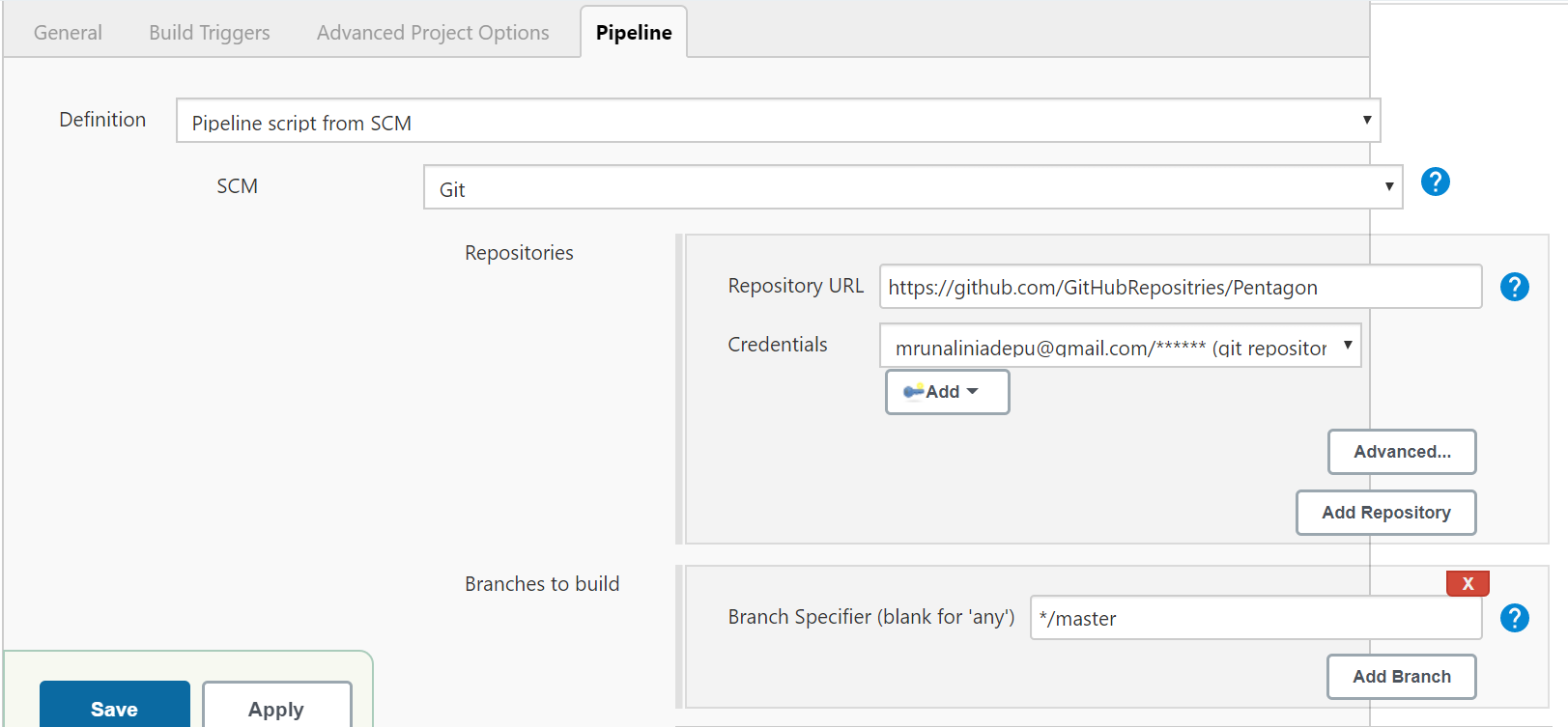
cd /var/lib/jenkins/secrets/

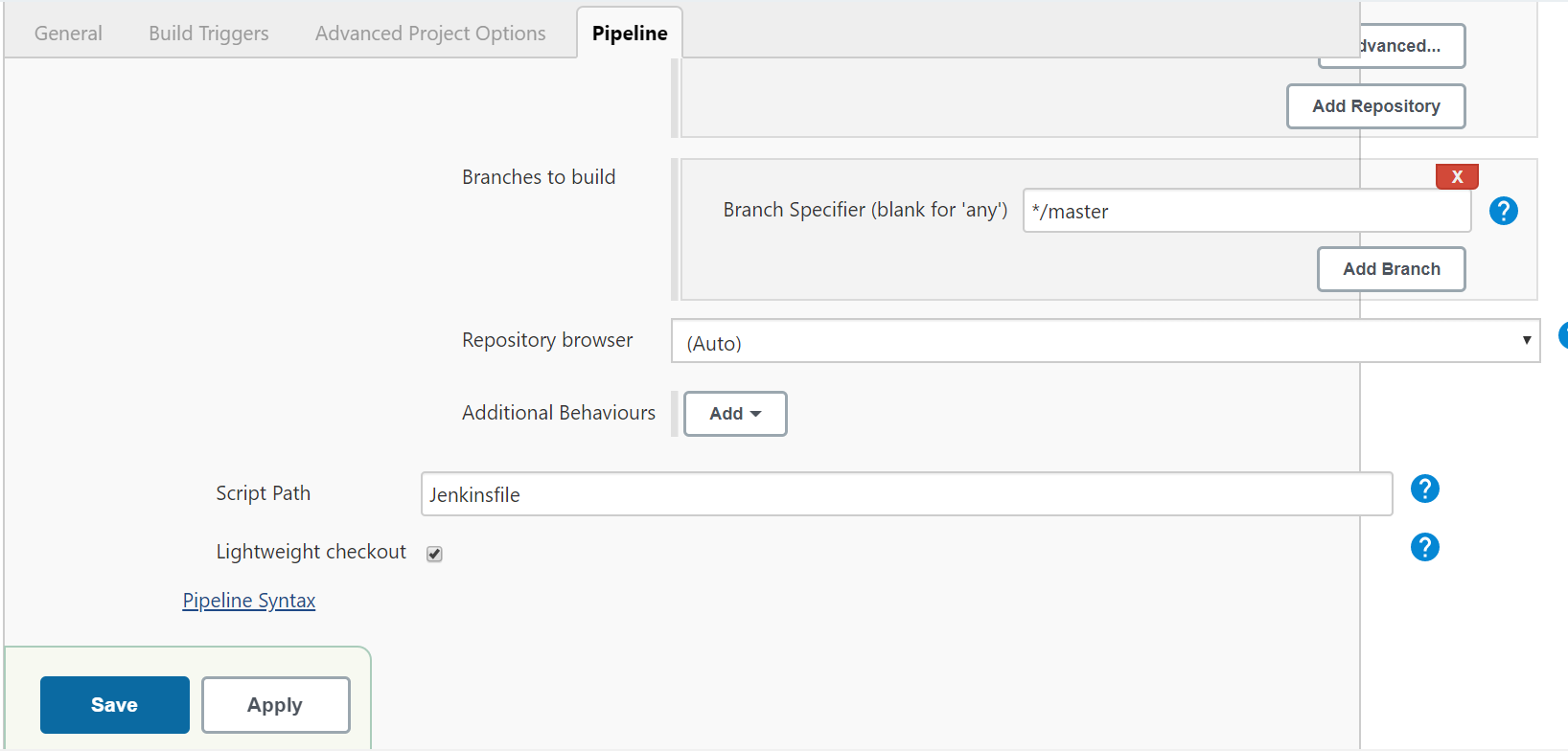
cat initialAdminPassword

1. To stop jenkins

sudo service jenkins stop

Launch the Jenkins server and create a new pipeline job as give the pipeline script location in the jenkins job as below





This job runs the Jenkinsfile that is placed in the git repository to deploy the application onto the Kubernetes cluster.