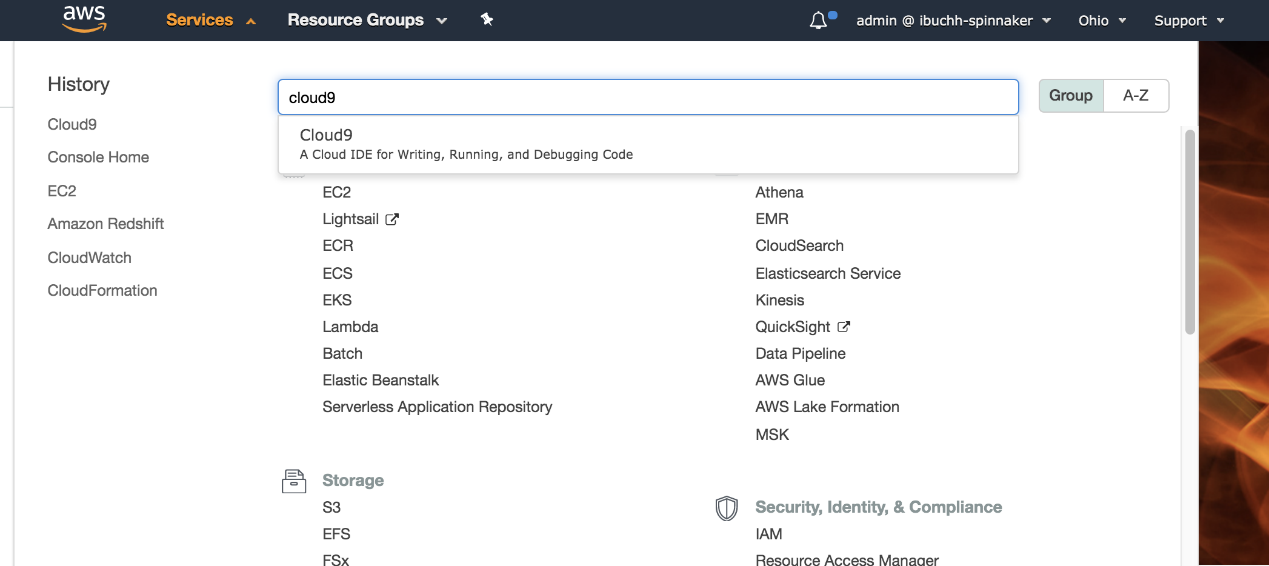
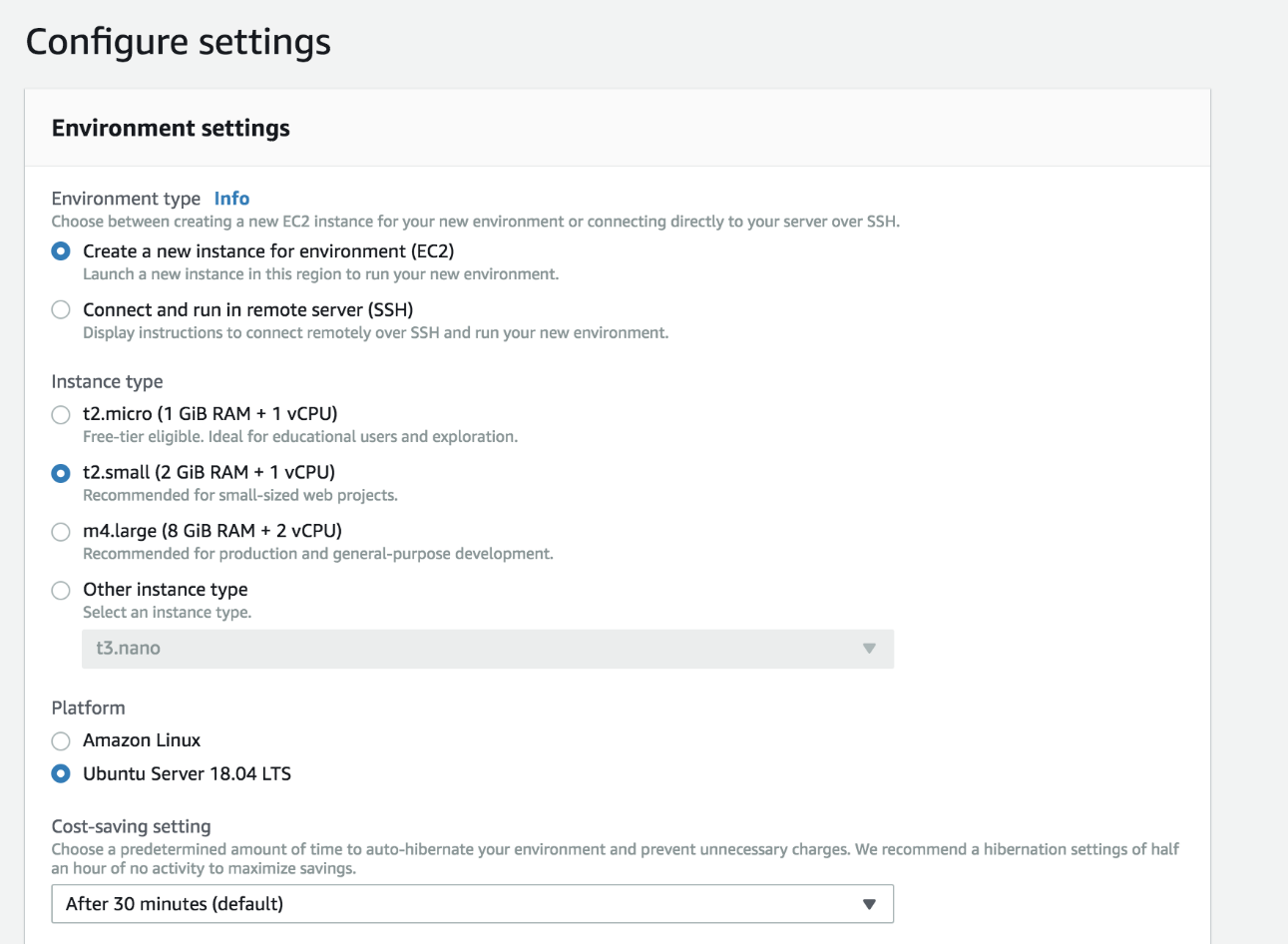
**Create an AWS Cloud9 environment**

Log into the AWS Management Console and search for Cloud9 services in the search bar:



*Fig 2. AWS Management Console*

Click Cloud9 and create an AWS Cloud9 environment in the us-east-2 region based on Ubuntu Server 18.04 LTS (Halyard is not supported on Amazon Linux yet). Choose the settings as shown below where the platform should be Ubuntu Server 18.04 LTS.



**Configure the AWS Cloud9 environment**

Launch the AWS Cloud9 IDE. In a new terminal session, follow the instructions to configure the AWS Cloud9 environment.

**1. Install and configure Kubectl**

Install kubectland aws-iam-authenticatoron the AWS Cloud9 Ubuntu machine:

#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

#curl -o aws-iam-authenticator https://amazon-eks.s3-us-west.amazonaws.com/1.13.7/2019-06-11/bin/linux/amd64/aws-iam-authenticator

#chmod +x ./aws-iam-authenticator

#mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$HOME/bin:$PATH

#echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc

#aws-iam-authenticator help

The script verifies that aws-iam-authenticator is working by displaying the help contents of aws-iam-authenticator.

**2. Upgrade awscli**

#aws –version

#pip install awscli --upgrade --user

**3. Install eksctl**

#curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest\_release/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

#sudo mv /tmp/eksctl /usr/local/bin

**4. Install Terraform**

#wget https://releases.hashicorp.com/terraform/0.12.4/terraform\_0.12.4\_linux\_amd64.zip

#unzip terraform\_0.12.4\_linux\_amd64.zip

#sudo mv terraform /usr/local/bin/

#export PATH=$PATH:/usr/local/bin/terraform

**5. Install Halyard**

#curl -O https://raw.githubusercontent.com/spinnaker/halyard/master/install/debian/InstallHalyard.sh

#sudo bash InstallHalyard.sh

#sudo update-halyard

#hal -v

**Create Amazon EKS clusters**

To make a complete environment, I will create three AWS EKS clusters including one for production, one for UAT, and one for Spinnaker installation. Inside the AWS Cloud9 IDE, run the following commands to create these Amazon EKS clusters. (You can choose your preferred regions; for this post I shall use us-east-2 to provision the Amazon EKS cluster for Spinnaker deployment and us-east-1 region to provision the UAT and production Amazon EKS clusters.

**1. Create the Production Amazon EKS cluster**

#eksctl create cluster --name=eks-prod --nodes=3 --region=us-east-1 --write-kubeconfig=false

**2. Create the UAT Amazon EKS cluster**

#eksctl create cluster --name=eks-uat --nodes=3 --region=us-east-1 --write-kubeconfig=false

**3. Create the Spinnaker Amazon EKS cluster**

#eksctl create cluster --name=eks-spinnaker --nodes=2 --region=us-east-2 --write-kubeconfig=false

[eksctl](https://aws.amazon.com/blogs/opensource/eksctl-eks-cli/) is a simple CLI tool for creating clusters on Amazon EKS which creates the following components of the Amazon EKS cluster architecture:

**Install and configure Spinnaker**

This section will walk you through the process of installing and configuring Spinnaker for use with Amazon EKS. I prefer to use Armory Spinnaker because:

* Armory provides an installer that does many of the configurations required with a command hal armory init.This configuration supports AWS Simple Storage Service S3.
* Armory provides [pipelines as code](https://docs.armory.io/docs/spinnaker/using-dinghy/) so that you can store pipeline configurations in source control and have a consistent, versioned method of application deployment. In the op you can only create pipelines through the UI.
* [Armory develops native integrations of Spinnaker with third party tools](https://www.armory.io/armory-spinnaker/integrations/).

**1. Retrieve Amazon EKS cluster kubectl contexts**

# aws eks update-kubeconfig --name eks-spinnaker --region us-east-2 --alias eks-spinnaker

#aws eks update-kubeconfig --name eks-uat --region us-east-1 --alias eks-uat

#aws eks update-kubeconfig --name eks-prod --region us-east-1 --alias eks-prod

**2. Check halyard version**

hal -v

**3. Create and configure a Docker registry**

#hal config provider docker-registry enable

#hal config provider docker-registry account add ibuchh-docker --address index.docker.io --username ibuchh --password

This command will prompt you to enter your docker account password.

**4. Add and configure a GitHub account**

#hal config artifact github enable

#hal config artifact github account add spinnaker-github --username ibuchh --password --token

This command will prompt you to enter your GitHub token that you can get from the GitHub account setting.

**5. Add and configure Kubernetes accounts**

*Production Amazon EKS account:*

Set the Kubernetes provider as enabled:

#hal config provider kubernetes enable

#kubectl config use-context eks-prod

A context element in a kubeconfig file is used to group access parameters under a convenient name. Each context has three parameters: cluster, namespace, and user. By default, the kubectl command line tool uses parameters from the current context to communicate with the cluster.

CONTEXT=$(kubectl config current-context)

We will create service accounts for the three Amazon EKS clusters.

#kubectl apply --context $CONTEXT -f https://spinnaker.io/downloads/kubernetes/service-account.yml

Extract the secret token of the spinnaker-service-account:

TOKEN=$(kubectl get secret --context $CONTEXT \

$(kubectl get serviceaccount spinnaker-service-account \

--context $CONTEXT \

-n spinnaker \

-o jsonpath='{.secrets[0].name}') \

-n spinnaker \

-o jsonpath='{.data.token}' | base64 --decode)

Set the user entry in kubeconfig:

#kubectl config set-credentials ${CONTEXT}-token-user --token $TOKEN

#kubectl config set-context $CONTEXT --user ${CONTEXT}-token-user

Add eks-prod cluster as a Kubernetes provider.

#hal config provider kubernetes account add eks-prod --provider-version v2 \

--docker-registries ibuchh-docker --context $CONTEXT

*UAT Amazon EKS account:*

#kubectl config use-context eks-uat

#CONTEXT=$(kubectl config current-context)

kubectl apply --context $CONTEXT -f https://spinnaker.io/downloads/kubernetes/service-account.yml

Extract the secret token of the spinnaker-service-account:

TOKEN=$(kubectl get secret --context $CONTEXT \

$(kubectl get serviceaccount spinnaker-service-account \

--context $CONTEXT \

-n spinnaker \

-o jsonpath='{.secrets[0].name}') \

-n spinnaker \

-o jsonpath='{.data.token}' | base64 --decode)

Set the service account entry in kubeconfig file:

#kubectl config set-credentials ${CONTEXT}-token-user --token $TOKEN

#kubectl config set-context $CONTEXT --user ${CONTEXT}-token-user

Add eks-uat cluster as a Kubernetes provider.

#hal config provider kubernetes account add eks-uat --provider-version v2 --docker-registries ibuchh-docker --context $CONTEXT

*Spinnaker Amazon EKS account:*

#kubectl config use-context eks-spinnaker

#CONTEXT=$(kubectl config current-context)

kubectl apply --context $CONTEXT -f https://spinnaker.io/downloads/kubernetes/service-account.yml

Extract the secret token of the spinnaker-service-account:

#TOKEN=$(kubectl get secret --context $CONTEXT \

$(kubectl get serviceaccount spinnaker-service-account \

--context $CONTEXT \

-n spinnaker \

-o jsonpath='{.secrets[0].name}') \

-n spinnaker \

-o jsonpath='{.data.token}' | base64 —decode)

Set the service account entry in the Kubeconfig file:

#kubectl config set-credentials ${CONTEXT}-token-user --token $TOKEN

#kubectl config set-context $CONTEXT --user ${CONTEXT}-token-user

Add eks-spinnaker cluster as a Kubernetes provider.

#hal config provider kubernetes account add eks-spinnaker --provider-version v2 --docker-registries ibuchh-docker --context $CONTEXT

**6. Enable artifact support**

#hal config features edit --artifacts true

**7. Configure Spinnaker to install in Kubernetes**

For our environment we will use a distributed Spinnaker installation onto the Kubernetes cluster. This installation model has Halyard deploy each of the Spinnaker microservices separately. A distributed installation helps to limit update-related downtime, making it recommended for use in production environments.

#hal config deploy edit --type distributed --account-name eks-spinnaker

**8. Configure Spinnaker to use AWS S3**

You will need your AWS account access key and secret access key.

#export YOUR\_ACCESS\_KEY\_ID=<access-key>

#hal config storage s3 edit --access-key-id $YOUR\_ACCESS\_KEY\_ID --secret-access-key --region us-east-2

Enter your AWS account secret access key at the prompt.

#hal config storage edit --type s3

**9. Choose the Spinnaker version**

To identify the latest version of Spinnaker to install, run the following to get a list of available versions:

#hal version list

At the time of writing, 1.15.0 is the latest Spinnaker version:

export VERSION=1.15.0

#hal config version edit --version $VERSION

Now we are finally ready to install Spinnaker on the eks-spinnaker Amazon EKS cluster.

#hal deploy apply

**10. Verify the Spinnaker installation**

#kubectl -n spinnaker get svc

**11. Expose Spinnaker using Elastic Loadbalancer**

I shall expose the Spinnaker API (Gate) and the Spinnaker UI (Deck) via Load Balancers by running the following commands to create the spin-gate-public and spin-deck-public services:

#export NAMESPACE=spinnaker

#kubectl -n ${NAMESPACE} expose service spin-gate --type LoadBalancer \

--port 80 --target-port 8084 --name spin-gate-public

#kubectl -n ${NAMESPACE} expose service spin-deck --type LoadBalancer \

--port 80 --target-port 9000 --name spin-deck-public

#export API\_URL=$(kubectl -n $NAMESPACE get svc spin-gate-public \

-o jsonpath='{.status.loadBalancer.ingress[0].hostname}')

#export UI\_URL=$(kubectl -n $NAMESPACE get svc spin-deck-public \

-o jsonpath='{.status.loadBalancer.ingress[0].hostname}')

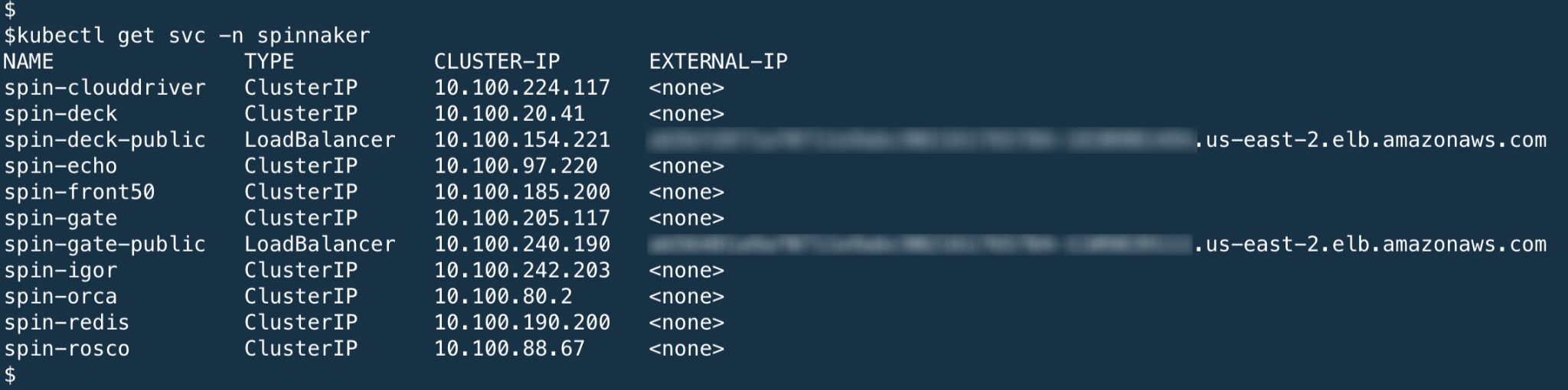
#hal config security api edit --override-base-url http://${API\_URL}

#hal config security ui edit --override-base-url http://${UI\_URL}

#hal deploy apply

**12. Re-verify the Spinnaker installation**

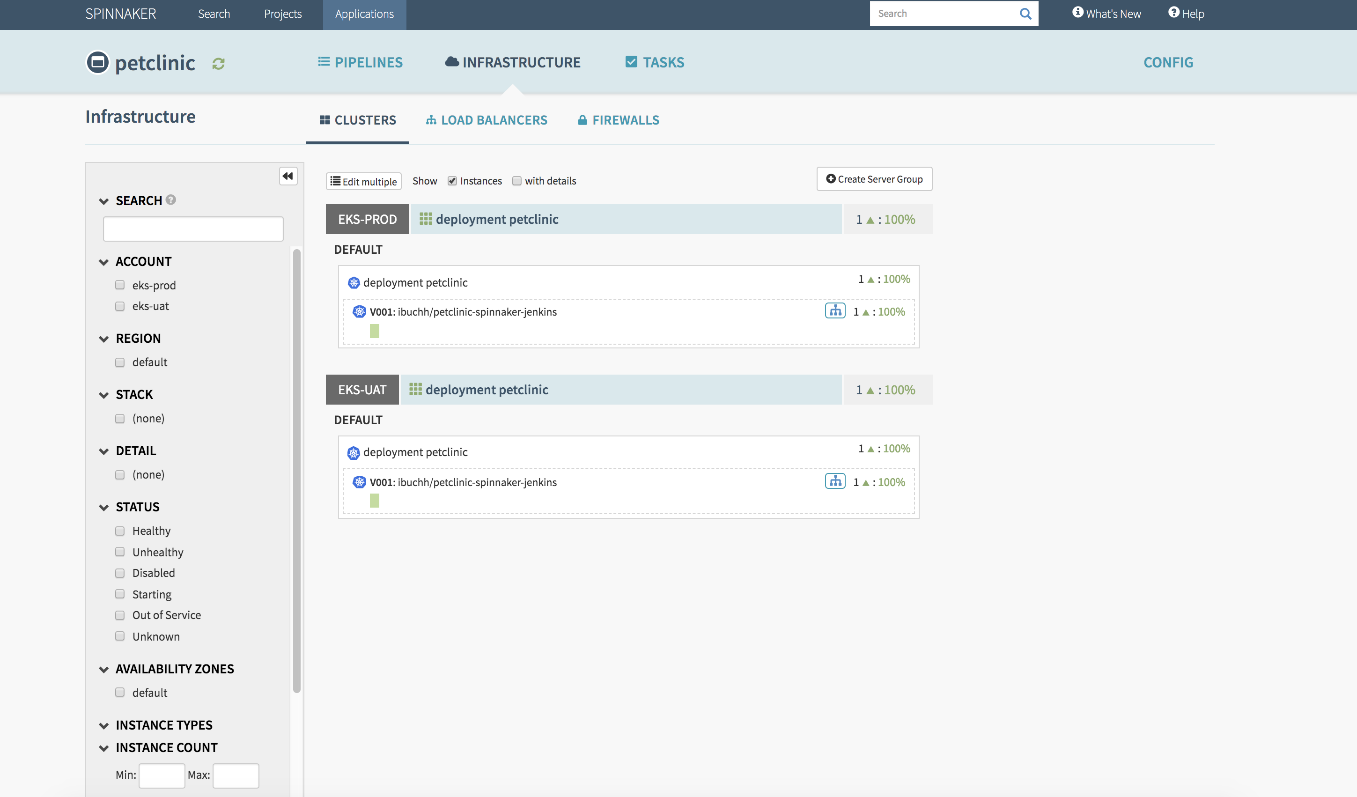
kubectl -n spinnaker get svc



*Fig 7. Spinnaker UI endpoints*

**13. Log in to Spinnaker console**

Using a browser, log in to the Spinnaker UI using the spin-deck-public services endpoint as shown above.



To implement the instructions in this post, you will need the following:

* AWS account
* Docker Hub account
* GitHub account

**Create a Jenkins CI server using Terraform**

Provisioning a Jenkins CI server manually can be error-prone and time-consuming, so I shall be configuring the Jenkins Continuous Server (CI) using Infrastructure as Code (IaC). For this post, I have decided to use Terraform. Log in to the AWS Management Console and [create an EC2 key pair](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html#having-ec2-create-your-key-pair) .Using your GitHub account, fork the code sample repository at  <https://github.com/balasg70/myblock.git>.

From the AWS Cloud9 IDE, open a shell terminal and do the following (replace aws-samples with your GitHub account):

git clone <https://github.com/balasg70/myblock.git>

cd myblock/Eks-jenkins-terraform/terraform/

# terraform init

# terraform plan

# terraform apply -auto-approve

Terraform apply will also output the IP address of the Jenkins CI server as shown above.

Terraform will provision an AWS EC2 instance and install git, Apache Maven, Docker, Java 8, and Jenkins as shown in the install\_jenkins.sh file:

#!/bin/bash

sudo yum -y update

echo "Install Java JDK 8"

sudo yum remove -y java

sudo yum install -y java-1.8.0-openjdk

echo "Install Maven"

sudo yum install -y maven

echo "Install git"

sudo yum install -y git

echo "Install Docker engine"

sudo yum update -y

sudo yum install docker -y

sudo sudo chkconfig docker on

echo "Install Jenkins"

sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo

sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key

sudo yum install -y jenkins

sudo usermod -a -G docker jenkins

sudo chkconfig jenkins on

echo "Start Docker & Jenkins services"

sudo service docker start

sudo service jenkins start

Bash

Using a browser, open the page at [http://jenkins\_ip\_address:8080](http://ip-addess:8080/); the Jenkins admin page will be displayed:

Using the AWS EC2 instance shell terminal, log in to the Jenkins CI server, find the Administrator password by running the following command:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Enter this Administrator password on the Jenkins Console by pasting it into the input box, and click **Next**. Click Install suggested plugin.

**Configure Jenkins**

1. Plugins:

Log in to the Jenkins console, click Manage Jenkins → Manage Plugins → Available. Choose and install Docker plugin and GitHub Integration Plugin, then restart Jenkins by clicking the Restart Jenkins

2. Credentials:

Docker Hub: Click Credentials → global → Add Credentials, choose Username with password as Kind, enter the Docker Hub username and password and use dockerHubCredentials for ID.

GitHub: Click Credentials → Global → Add Credentials , choose Username with password as Kind, enter the GitHub username and password and use gitHubCredentials for ID.

**Configure the Jenkins job and pipeline**

From the Jenkins console, click New item. Choose Multibranch Pipeline, name it petclinic and click OK.

Choose GitHub and from the drop-down select the GitHub credentials. Enter the GitHub URL as shown below and click Save to save the Jenkins job.

The Jenkins build executor will check out and scan the GitHub repository and execute the stages in the pipeline as laid out in the Jenkins file shown below. Make sure that you replace the registry with your Docker registry URL inside the build stage.

Below is a screenshot of the final run; if all goes well, you will see a new Docker image pushed to your Docker registry.

**Create and configure Spinnaker pipelines**

A pipeline is a sequence of stages provided by Spinnaker, ranging from functions that manipulate infrastructure (deploy, resize, disable) to utility scaffolding functions (manual judgment, wait, run Jenkins job) that together precisely define your runbook for managing your deployments. Pipelines help you manage deployments consistently, repeatably, and safely.

1. Log in to the AWS EC2 instance environment and open a new terminal. Run the following command:

kubectl get svc -n spinnaker

2.  Using a browser, log in to the Spinnaker UI using the spin-deck-public services endpoint as shown in the output above.

Select the Applications tab, then Actions → Create Application. Enter petclinic as Name and enter a valid email address, leave the rest of the fields blank.

3.  On the Pipelines tab, click Configure a new pipeline , enter DeployToUAT as the Pipeline Name and click Create.

4.  Click Add Artifact and choose GitHub → Kind , File path → kubernetes/petclinic.yaml, Display name → Petclinic-Manifest, Content URL → <https://github.com/balasg70/myblock/blob/master/Eks-jenkins-terraform/kubernetes/petclinic.yaml>

5.  Click Add Trigger and choose Type → Docker Registry, Registry Name → your Docker registry as configured in Spinnaker, Organization → your Docker registry name, Image → Docker image as created by Jenkins.

6.  Click Add Stage, choose Stage Type → Deploy (Manifest) , Account → eks-uat, Application → petclinic, Manifest Source → Artifact, Manifest Artifact → Petclinic-Manifest, Artifact Account → spinnaker-github.

7.  Click Save to save the changes to the DeployToUAT pipeline.

8.  Under the PIPELINES tab, click Create , enter ManualApproval as the Pipeline Name and click Create. Click Add Trigger and Choose Type → Pipeline, Application → petclinic, Pipeline → DeployToUAT.

9.  Click Add Stage, choose Stage Name → Manual Judgement, under Judgement Inputs add two options Approve and Reject as shown below:

10.  Click Save to save the changes to the ManualApproval pipeline.

11.  Under Pipelines tab, click Create , enter DeployToProd as the Pipeline Name and click **Create**. Click **Add Trigger** and Choose Type → Pipeline, Application → petclinic, Pipeline → DeployToProd.

12.  Click Add Artifact and choose GitHub → Kind , File path → kubernetes/petclinic.yaml, Display name → Petclinic-Manifest, Content URL → <https://github.com/balasg70/myblock/blob/master/Eks-jenkins-terraform/kubernetes/petclinic.yaml>

13.  Click Add Trigger and choose Type → Docker Registry, Registry Name → your Docker registry as configured in Spinnaker, Organization → your Docker registry name, Image → Docker image created by Jenkins.

14.  Click Add Stage, choose Stage Type → Deploy (Manifest) , Account → eks-prod, Application → petclinic, Manifest Source → Artifact, Manifest Artifact → Petclinic-Manifest, Artifact Account → spinnaker-github.

15.  Click Save to save the changes of the DeployToProd pipeline.

**Run Spinnaker pipelines manually**

Now run the three pipelines manually. Click Start Manual Execution, choose Pipeline → DeployToUAT, Type → Tag, Tag → enter a valid tag number. Click Run and watch the pipeline execution.