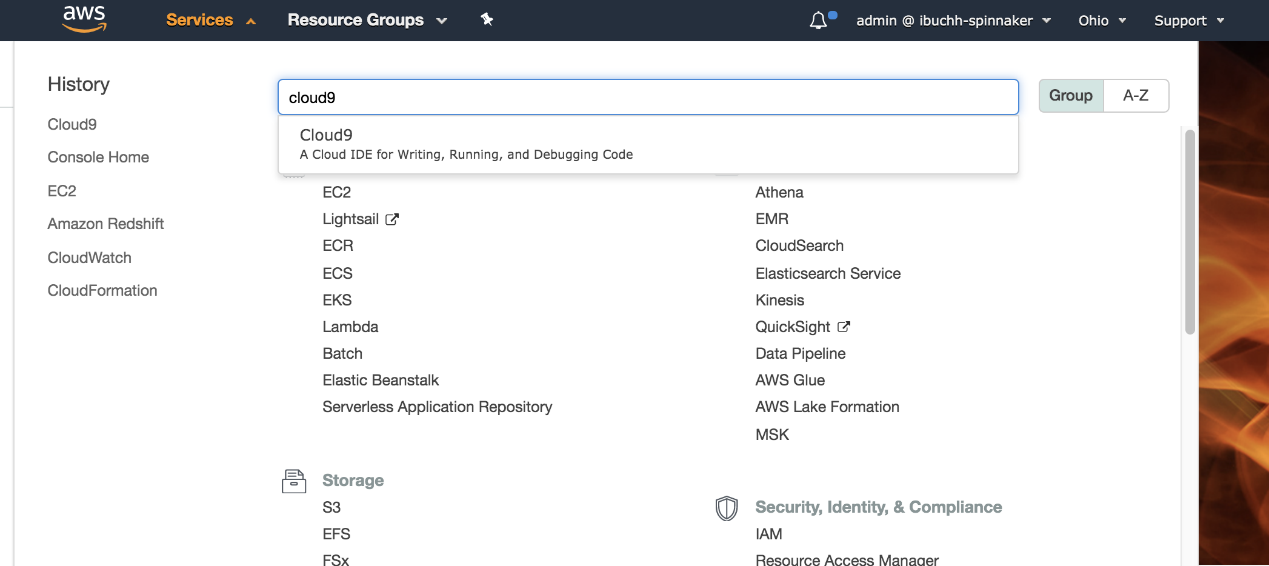
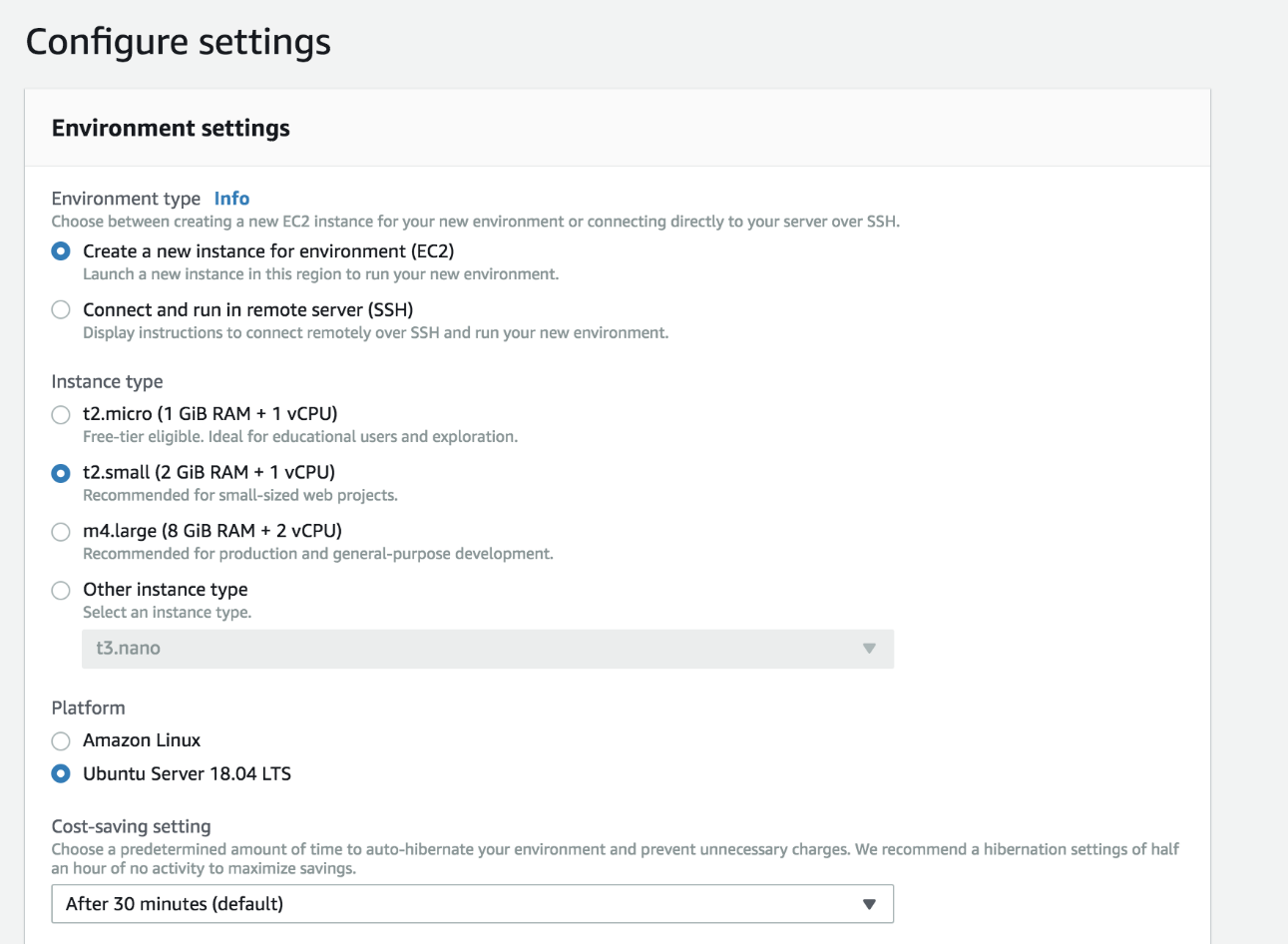
**Create an AWS Cloud9 environment**

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 on 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

**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

#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 UAT Amazon EKS cluster**

#eksctl create cluster --name=eks-uat --nodes=3 --region=us-east-1 --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:

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

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

**5. Add and configure Kubernetes accounts**

*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

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

kubectl -n testapp get svc

https://aws.amazon.com/amazon-linux-2/

2 package(s) needed for security, out of 13 available

Run "sudo yum update" to apply all updates.

-bash: warning: setlocale: LC\_CTYPE: cannot change locale (UTF-8): No such file or directory

[ec2-user@ip-10-0-1-93 ~]$ sudo su -

Last login: Fri Oct 2 09:22:21 UTC 2020 on pts/2

[root@ip-10-0-1-93 ~]# kubectl get services --all-namespaces

NAMESPACE NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

default hello-kubernetes LoadBalancer 10.100.141.170 a96412d3fd7aa4e9e95137b1d8aadc4e-650145602.us-east-1.elb.amazonaws.com 80:30975/TCP 2d1h

default kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 2d19h

kube-system kube-dns ClusterIP 10.100.0.10 <none> 53/UDP,53/TCP 2d19h

testapp hello-kubernetes LoadBalancer 10.100.250.110 aeac6ecbb22d940f2a2ae259784027be-954773184.us-east-1.elb.amazonaws.com 80:31486/TCP 40h

[root@ip-10-0-1-93 ~]# kubectl get all --all-namespaces

NAMESPACE NAME READY STATUS RESTARTS AGE

default pod/hello-kubernetes-fb869d65c-2q7nq 0/1 Pending 0 6m28s

default pod/hello-kubernetes-fb869d65c-8b98v 0/1 Pending 0 6m28s

default pod/hello-kubernetes-fb869d65c-x7phh 0/1 Pending 0 6m29s

kube-system pod/coredns-75b44cb5b4-krm8n 0/1 Pending 0 6m28s

kube-system pod/coredns-75b44cb5b4-rr9px 0/1 Pending 0 6m28s

testapp pod/hello-kubernetes-67c76959d5-b6dlf 0/1 Pending 0 6m28s

testapp pod/hello-kubernetes-67c76959d5-bdkf9 0/1 Pending 0 6m28s

testapp pod/hello-kubernetes-67c76959d5-qqrqv 0/1 Pending 0 6m28s

NAMESPACE NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

default service/hello-kubernetes LoadBalancer 10.100.141.170 a96412d3fd7aa4e9e95137b1d8aadc4e-650145602.us-east-1.elb.amazonaws.com 80:30975/TCP 2d1h

default service/kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 2d19h

kube-system service/kube-dns ClusterIP 10.100.0.10 <none> 53/UDP,53/TCP 2d19h

testapp service/hello-kubernetes LoadBalancer 10.100.250.110 aeac6ecbb22d940f2a2ae259784027be-954773184.us-east-1.elb.amazonaws.com 80:31486/TCP 40h

NAMESPACE NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE

kube-system daemonset.apps/aws-node 0 0 0 0 0 <none> 2d19h

kube-system daemonset.apps/kube-proxy 0 0 0 0 0 <none> 2d19h

NAMESPACE NAME READY UP-TO-DATE AVAILABLE AGE

default deployment.apps/hello-kubernetes 0/3 3 0 2d1h

kube-system deployment.apps/coredns 0/2 2 0 2d19h

testapp deployment.apps/hello-kubernetes 0/3 3 0 40h

NAMESPACE NAME DESIRED CURRENT READY AGE

default replicaset.apps/hello-kubernetes-fb869d65c 3 3 0 2d1h

kube-system replicaset.apps/coredns-75b44cb5b4 2 2 0 2d19h

testapp replicaset.apps/hello-kubernetes-67c76959d5 3 3 0 40h

[root@ip-10-0-1-93 ~]#

**13. Log in to Hello console**

Using a browser, display the screen as shown below.

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.