SWE 645 - Assignment 2

TEAM NAME

Tech Bros

TEAM MEMBERS

Anurag Repaka
Abhishek Bodas
Praveen Menon Ambadath
Aneesha Chegoni
Rishita Guduru

Contributions

1. Aneesha and Rishita worked together on setting up project in the docker container for java project and Jenkins container.
2. Anuragh and Abhishake worked on RND and installation and integration of kubecl to AWS and GCP.
3. Praveen worked on integrating AWS EKS with kubectl and the delvelopment.yaml and creating jenkins script for CI/CD

Containerize the application - Docker

Install docker link: https://docs.docker.com/v17.09/engine/installation/

1. For this part we first installed the docker desktop app on the system and setup a docker hub account.
2. Once the docker app is installed we set up a docker tomcat container on our local.
3. Now we have a docker image with tomcat installed. The next part was to setup the part 1 project inside the container inside the /webapps folder
4. To setup the project we had to write a dockerfile.

DockerFile

```
FROM praveenmenon/swe-645:version01

LABEL maintainer="pamenon9@gmail.com"

ENV CATALINA_HOME /usr/local/tomcat

ENV PATH $CATALINA_HOME/bin:$PATH

RUN mkdir -p "$CATALINA_HOME"

WORKDIR $CATALINA_HOME

ADD student-service.war $CATALINA_HOME/webapps/

EXPOSE 8080

CMD ["catalina.sh", "run"]
```

5. The above file is the dockerfile script for inserting the war student-service inside the webapps folder. The java project is exposed to port 8080 by running the catalina.sh file.

After creating the docker file we ran the following command to test the docker container.
docker build -f Dockerfile -t tomcat:9.0 .
docker run -it --rm -p 8888:8080 tomcat:9.0

Here 8888 is the docker container and 8080 is the tomcat running inside docker container.

On a different terminal run the following command to check running docker instances

docker ps -a

output

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b8086524e838	tomcat:9.0	"catalina.sh run"	3 minutes ago	Up 3 minutes	0.0.0.0:8888->8080/tcp	interesting_khayyam

Docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tomcat	9.0	b4ebc4bb3403	8 days ago	626MB

Kubernetes orchestration of docker containers

Install kubernetes: https://kubernetes.io/docs/tasks/tools/install-kubectl/

AWS eks instalation https://docs.aws.amazon.com/eks/latest/userguide/getting-started.html

Now that the docker container is working fine, we created a tag to our container using the commit command

Docker push praveenmenon/swe-645:version01

Here swe-645 is the docker container name and version01 is the tag name

once docker is installed the next step is to install kubectl

Kubectl is used to interact with your kunernetes.

Once kubectl is installed we need to setup aws CLI onto your terminal

with AWS CLI set run the following command on the terminal

aws eks update-kubeconfig --name javacluster
here javacluser is the cluster name that you created on EKS

The update kubeconfig command creates a config file inside the .kube folder

once the kubernetes is setup on the aws account account you need to and aws-eks-cluster.yaml file to connect all the worker nodes to your cluster

aws-eks-cluster.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn:  arn:aws:iam::144763098142:role/javacluster-worker-nodes-NodeInstanceRole-1QNWRZYSKGE4H
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

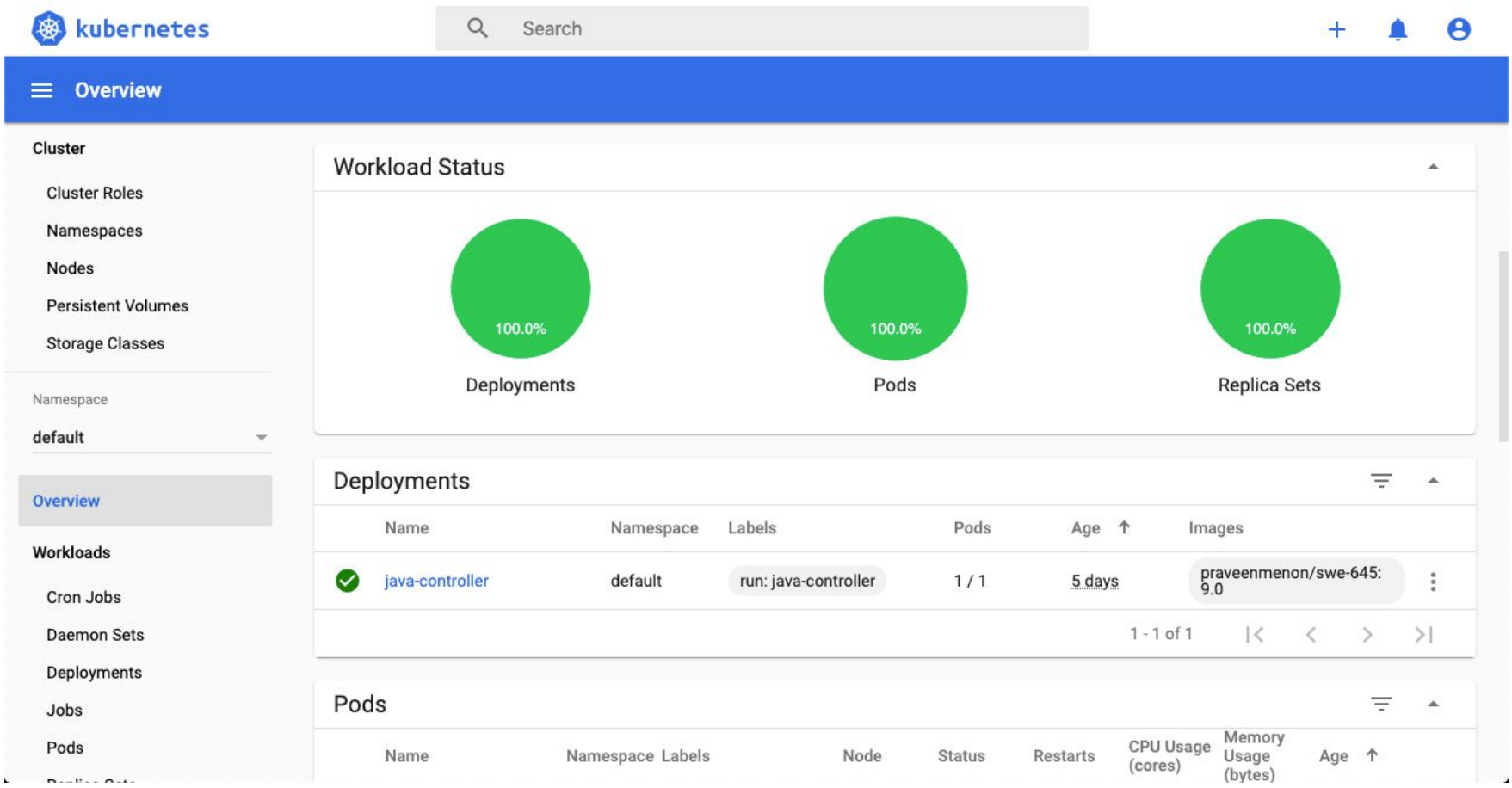
Here we added the ARN of the role used to execute the kubernetes cluster

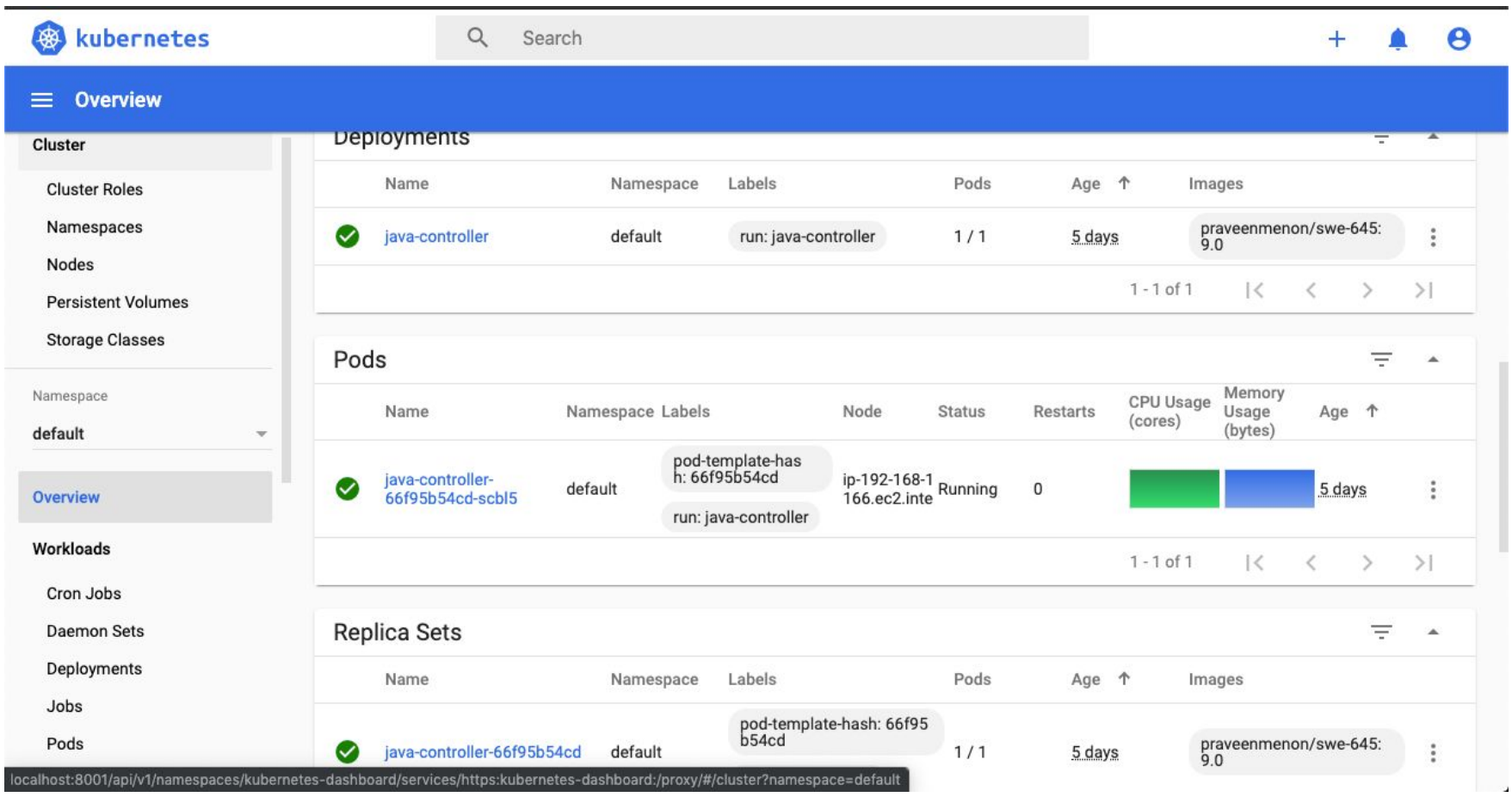
Once all our nodes are visible, we created a deployment.yaml file to deploy our container

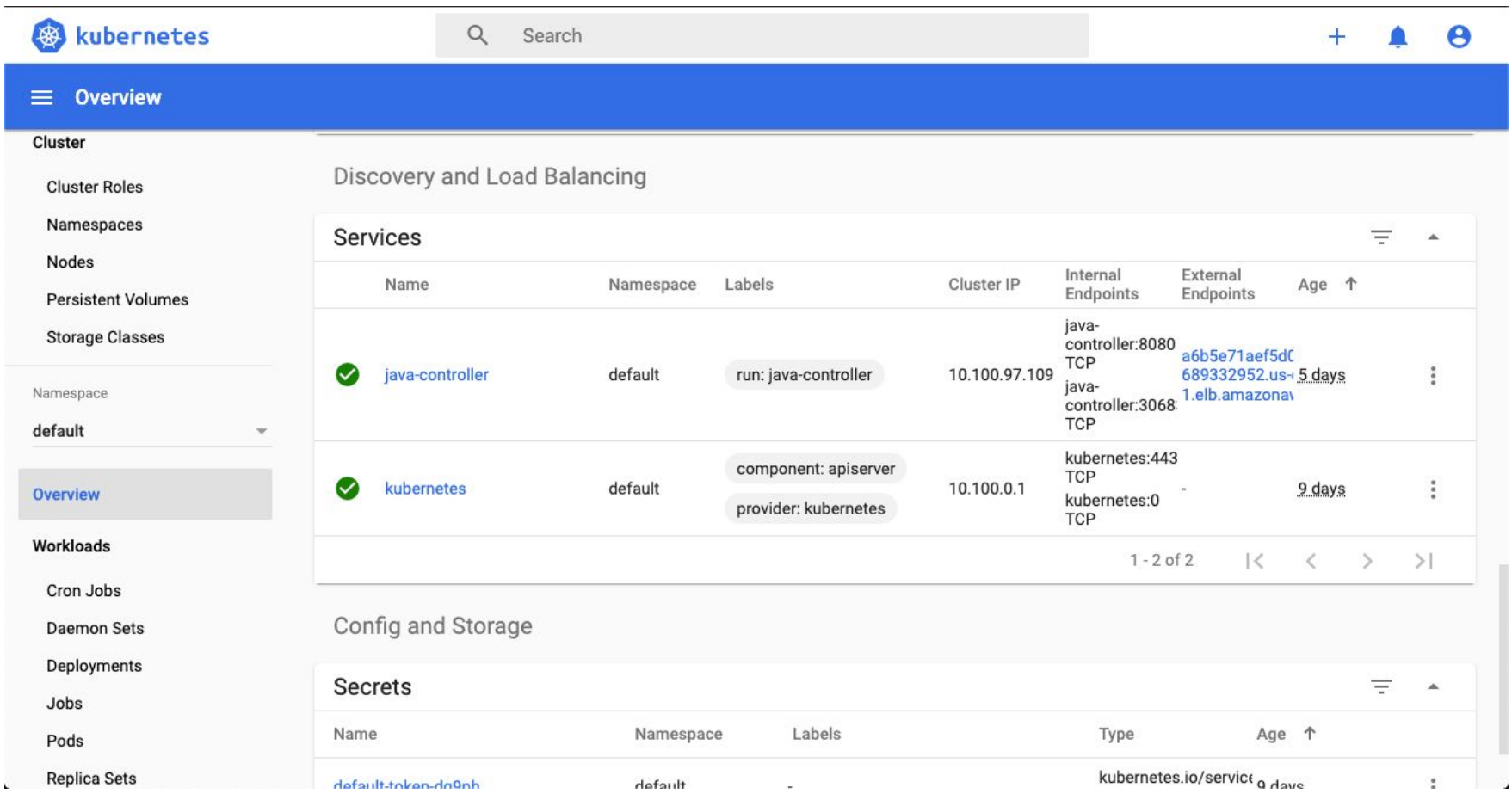
Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    run: java-controller
  name: java-controller
spec:
  replicas: 1
  selector:
    matchLabels:
      run: java-controller
  strategy: {}
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: java-controller
    spec:
      containers:
      - image: praveenmenon/swe-645:9.0
        name: java-controller
        ports:
        - containerPort: 8080
        livenessProbe:
          httpGet:
            path: /student-service
            port: 8080
          initialDelaySeconds: 30
          timeoutSeconds: 1
        resources: {}
status: {}
```







Once the containers are deployed we ran the following command to get the endpoint for docker.

Kubectl get svc

Output

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

java-controller LoadBalancer 10.100.97.109 a6b5e71aef5d011e9a090022e5d4964a-689332952.us-east-1.elb.amazonaws.com 8080:30683/TCP 5d3h

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 9d

This is the link to our docker container.
http://a6b5e71aef5d011e9a090022e5d4964a-689332952.us-east-1.elb.amazonaws.com:8080/student-service/