SWE 645 Assignment 3

Group Members:

- Avinash Arunachalam A Murugappan
- Rushil Nandhan Dubey
- Rashi Varshney
- Mithilaesh Jayakumar

Exposing JPA entity class as RESTful web service using jersey

Step 1- Download a Reference Implementation to JAX-RS

In order to get started, you need to download a reference implementation of JAX-RS provided by the GlassFish project. Jersey provides a servlet that analyzes the incoming HTTP request and selects the correct class and method to respond to this request. This selection is based on annotation in the class and methods.

You can download Jersey zip file (core dependencies) from the Jersey website.

Step 2 - Create Your Java Project

Using Eclipse, create the project's directory structure (i.e., with Eclipse, use Dynamic Web Project).
Next, copy all the Jersey core-dependency jars into the WEB-INF/lib folder of the project.

Step 3 - Register the Jersey Reference Implementation Servlet with the Project

In order to get Jersey intercepting the requests, you need to modify the Web.xml file like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
<display-name>jaxrscrud</display-name> <servlet>
<servlet-name>Jersey REST Service</servlet-name>
<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
<init-param> <param-name>com.sun.jersey.config.property.packages</param-name>
<param-value>crudOperations</param-value> </init-param>
<load-on-startup>1</load-on-startup> </servlet>
```

```
<servlet-mapping> <servlet-name>Jersey REST Service</servlet-name>
<url-pattern>/rest/*</url-pattern> </servlet-mapping> </web-app>
```

Set Up a RDS Database

Step 1: Create a VPC with Private and Public Subnets
Step 2: Create Additional Subnets
Step 3: Create a VPC Security Group for a Public Web Server
Step 4: Create a VPC Security Group for a Private DB Instance
Step 5:

Create a DB Subnet Group

To launch a MySQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you want to create the DB instance. This example uses the US West (Oregon) Region.
3. In the navigation pane, choose Databases.
4. Choose Create database.
5. On the Create database page, shown following, make sure that the Standard Create option is chosen, and then choose MySQL.
6. In the Templates section, choose Dev/Test.
7. In the Settings section, set these values:
    - DB instance identifier – tutorial-db-instance
    - Master username – tutorial_user
    - Auto generate a password – Disable the option
    - Master password – Choose a password.
    - Confirm password – Retype the password.

8. In the DB instance size section, set these values:
    - DB instance performance type – Burstable
    - DB instance class – db.t2.small
9. In the Storage and Availability & durability sections, use the default values.
10. In the Connectivity section, open Additional connectivity configuration and set these values:
    - Virtual Private Cloud (VPC) – Choose an existing VPC with both public and private subnets, such as the tutorial-vpc (vpc-identifier) created in Create a VPC with Private and Public Subnets

Note

- The VPC must have subnets in different Availability Zones.
- Subnet group – The DB subnet group for the VPC, such as the tutorial-db-subnet-group created in Create a DB Subnet Group

- Publicly accessible – No
- VPC security groups – Choose an existing VPC security group that is configured for private access, such as the tutorial-db-securitygroup created in Create a VPC Security Group for a Private DB Instance.
  Remove other security groups, such as the default security group, by choosing the X associated with each.
- Availability zone – No Preference
- Database port – 3306

11. Open the Additional configuration section, and enter a sample for Initial database name. Keep the default settings for the other options.
12.To create your Amazon RDS MySQL DB instance, choose Create database.
13.Your new DB instance appears in the Databases list with the status Creating.
14.Wait for the Status of your new DB instance to show as Available. Then choose the DB instance name to show its details.
15.In the Connectivity & security section, view the Endpoint and Port of the DB instance.

<u>Containerizing the application using docker:</u>

<u>Step 1 - Installing Docker</u>

1. Login to your server and update the software repository.

   ```
   ssh root@192.168.1.248
   ```

   ```
   apt-get update
   ```

2. Install docker.io with this apt command:

   ```
   apt-get install docker.io
   ```

3. When the installation is finished, start the docker service and enable it to start at boot time:

   ```
   systemctl start docker
   ```

   ```
   systemctl enable docker
   ```

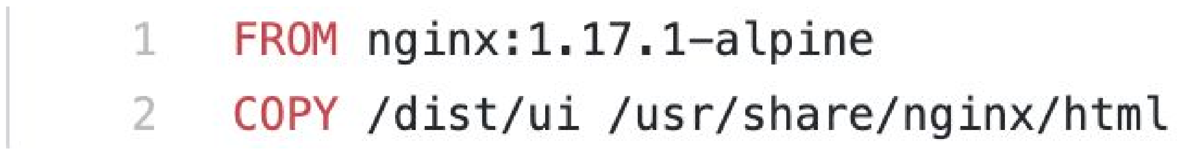
4. Docker has been installed and is running on the ec2 system.

<u>Step 2 - Create Dockerfile</u>

1. Export war file
2. Build the docker file.

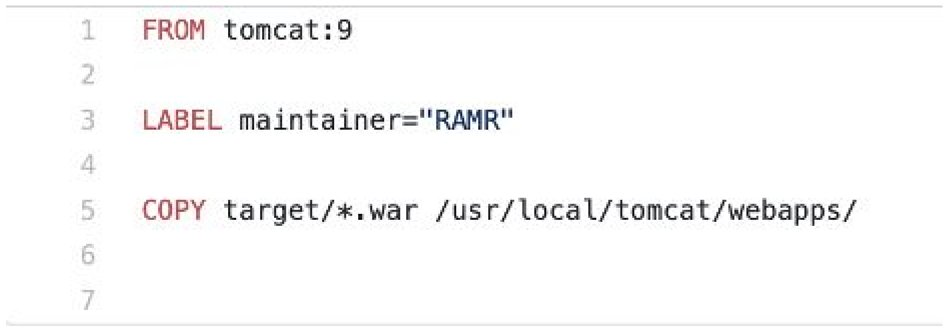
Dockerfile:

Dockerfile for frontend:



```
FROM nginx:1.17.1-alpine
COPY /dist/ui /usr/share/nginx/html
```

Dockerfile for backend:



```
FROM tomcat:9

LABEL maintainer="RAMR"

COPY target/*.war /usr/local/tomcat/webapps/


```

Step 3 - Build New Docker Image and Create New Container Based on it

1. The Dockerfile and all required config files have been created, now we can build a new docker image based on Ubuntu and our dockerfile with the docker command below:

   docker build -t ‘Image-name’.

2. When the command completed successfully, we can check the new image 'image-name' with the docker command below:

   docker images

3. Now run the new container with the command below:

   docker run -p 80:80 --name image-name

4. Then we can check that the new container based on 'image-name' is running:

   docker ps

Setting up Jenkins on ec2

Creating the ec2 for Jenkins:

1. Create AWS account
2. Navigate to ec2 and choose instances and click launch instance
3. Choose Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type
4. Select t2.micro and proceed, just use the default options provided and proceed until configure the security group.
5. Choose ports 8080, 22, 8081 and launch the instance.

Note: Since we need Jenkins to build docker images for us, we have to install Jenkins on the same ec2 where docker was installed mentioned on the above steps.

Installing Jenkins on ec2:

Note: Must have Java8 to run Jenkins. Java8 can be installed by using the following command, Or if you have multiple java versions, then choose the appropriate version that is java8.

```
[ec2-user ~]$ sudo yum install java-1.8.0
```

1. To ensure that your software packages are up to date on your instance, use the following command to perform a quick software update:

```
[ec2-user ~]$ sudo yum update –y
```

2. Add the Jenkins repo using the following command:

```
[ec2-user ~]$ sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
```

3. Import a key file from Jenkins-CI to enable installation from the package:

```
[ec2-user ~]$ sudo rpm — import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
```

4. Install Jenkins:

```
[ec2-user ~]$ sudo yum install jenkins -y
```

5. Start Jenkins as a service.

```
[ec2-user ~]$ sudo service jenkins start
```

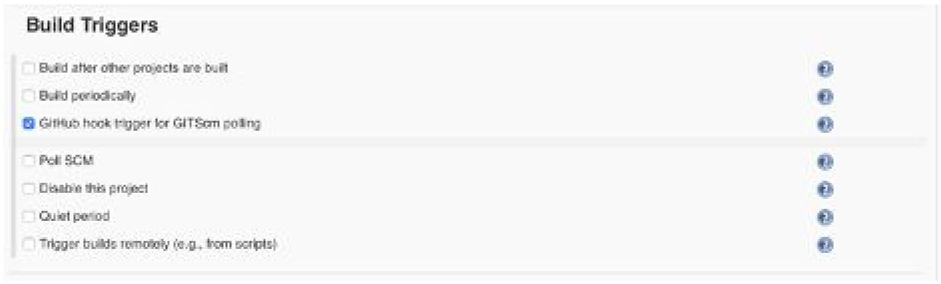
Note : Before running the Jenkins, make sure your 8080 port is available or else you could run Jenkins on any other available port by simply changing the port inside the configuration file of CentOS rpm based linux i.e. /etc/sysconfig/jenkins file(The location in debian based linux is /var/default/jenkins).

Visit the following address to use Jenkins, http://server-ip-address:8080/

[Note: We changed the port, our Jenkins is running on 8081]

<u>Automated deployment to k8's with jenkins:</u>

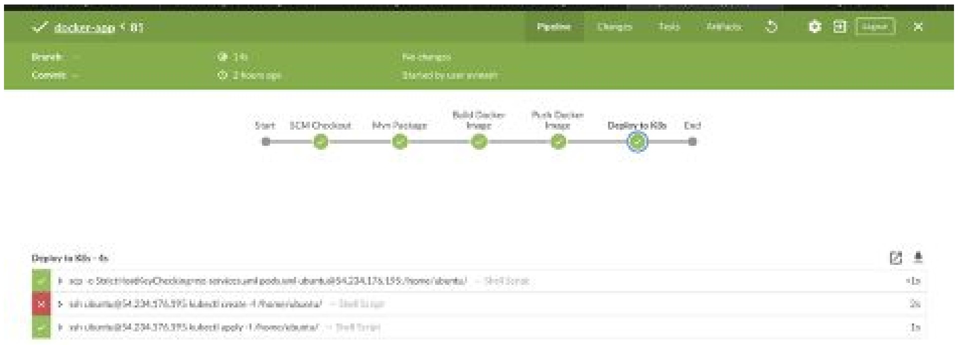
1. To make builds on Jenkins automatically, we have to configure build triggers and choose GitHub hook trigger. So, each time we commit to our GitHub repo, Jenkins will trigger to build automatically.



Build Triggers

- ☐ Build after other projects are built
- ☐ Build periodically
- ☑ GitHub hook trigger for GITScm polling
- ☐ Poll SCM
- ☐ Disable this project
- ☐ Quiet period
- ☐ Trigger builds remotely (e.g., from scripts)

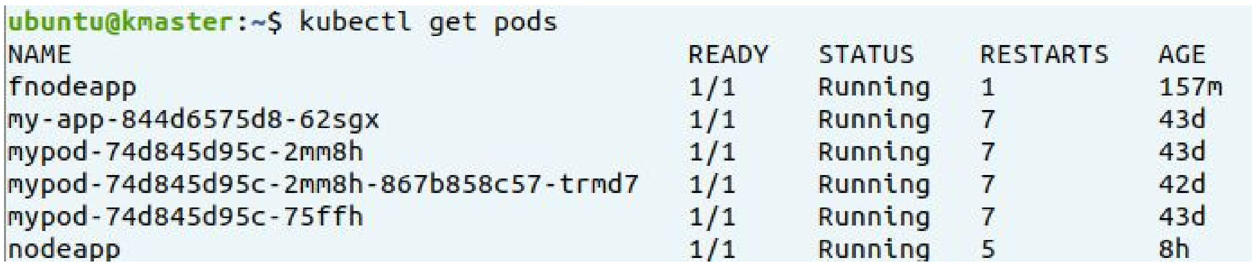
2. Jenkins will build the ci/cd for deployment to k8's, We have used blue ocean to monitor the health of each build with detailed information. You can see all the stages have been successfully run



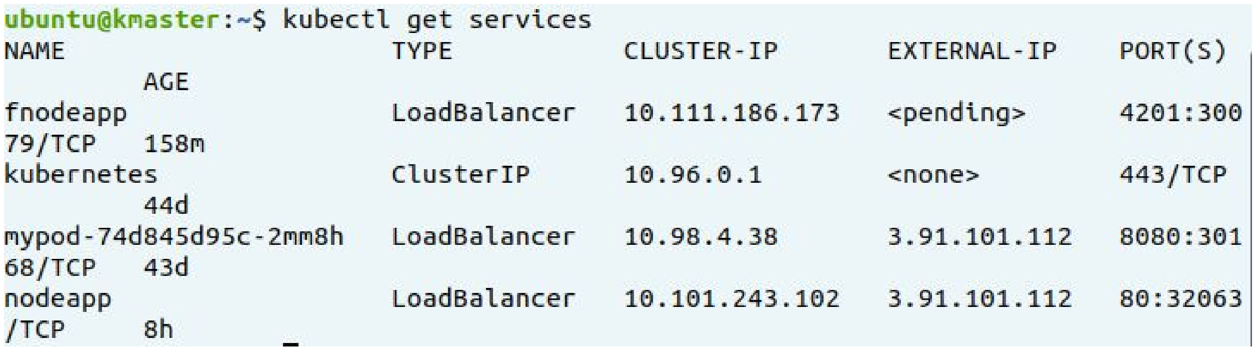
```
scp -o StrictHostKeyChecking=no services.yml pods.yml ubuntu@54.234.176.195:/home/ubuntu/ — Shell Script   <1s
ssh ubuntu@54.234.176.195 kubectl create -f /home/ubuntu/ — Shell Script   2s
ssh ubuntu@54.234.176.195 kubectl apply -f /home/ubuntu/ — Shell Script   1s
```

3. After sshing into the master node, we can see our application deployed 'nodeapp' on the cluster and exposed to the external-ip

<u>screenshot of kubernetes pods</u>



```
ubuntu@kmaster:~$ kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
fnodeapp                                    1/1     Running   1          157m
my-app-844d6575d8-62sgx                     1/1     Running   7          43d
mypod-74d845d95c-2mm8h                      1/1     Running   7          43d
mypod-74d845d95c-2mm8h-867b858c57-trmd7     1/1     Running   7          42d
mypod-74d845d95c-75ffh                      1/1     Running   7          43d
nodeapp                                     1/1     Running   5          8h
```



```
ubuntu@kmaster:~$ kubectl get services
NAME                      TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)
         AGE
fnodeapp                  LoadBalancer   10.111.186.173   <pending>      4201:300
79/TCP   158m
kubernetes                ClusterIP      10.96.0.1        <none>         443/TCP
         44d
mypod-74d845d95c-2mm8h    LoadBalancer   10.98.4.38       3.91.101.112   8080:301
68/TCP   43d
nodeapp                   LoadBalancer   10.101.243.102   3.91.101.112   80:32063
/TCP     8h
```

Links:

Frontend: http://34.229.53.170:8082/

Backend: http://3.91.101.112:32063/

Github :

- Frontend code : https://github.com/RAMR645/swefront
- Backend code : https://github.com/RAMR645/sweback

S3 links:

Avinash’s s3: http://swe645spring2020.s3-website-us-east-1.amazonaws.com

Rushil’s s3: http://swe645assign.s3-website-us-east-1.amazonaws.com/

Rashi’s s3: http://swe645-rv.s3-website-us-east-1.amazonaws.com/

Mithilaesh’s s3: http://mjayakum.s3-website-us-east-1.amazonaws.com/

References:

Containerizing angular application
https://medium.com/@wkrzywiec/build-and-run-angular-application-in-a-docker-container-b65dbbc50be8

Setting up RDS Database
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateVPC.html#CHAP_Tutorials.WebServerDB.CreateVPC.VPCAndSubnets

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateDBInstance.html

Building docker image from the war file:

https://aspetraining.com/resources/blog/deploying-your-first-web-app-to-tomcat-on-docker

To push docker image into the docker hub:

https://ropenscilabs.github.io/r-docker-tutorial/04-Dockerhub.html

Kubernetes installation:

https://phoenixnap.com/kb/install-kubernetes-on-ubuntu

Deploying docker image into kubernetes as a service:

https://codeburst.io/getting-started-with-kubernetes-deploy-a-docker-container-with-kubernetes-in-5-minutes-eb4be0e96370

Setting up Jenkins: https://www.youtube.com/watch?v=jmm8DsosBqw

Contributors: [Everyone contributed equally towards the assignment.]

- Avinash Arunachalam Arunachalachettiar Murugappan[G01163980]
- Rushil Nandhan Dubey[G01203932]
- Rashi Varshney[G01225299]
- Mithilaesh Jayakumar[G01206238]