# SWE 645: Assignment 3 RESTful Web Services and JPA

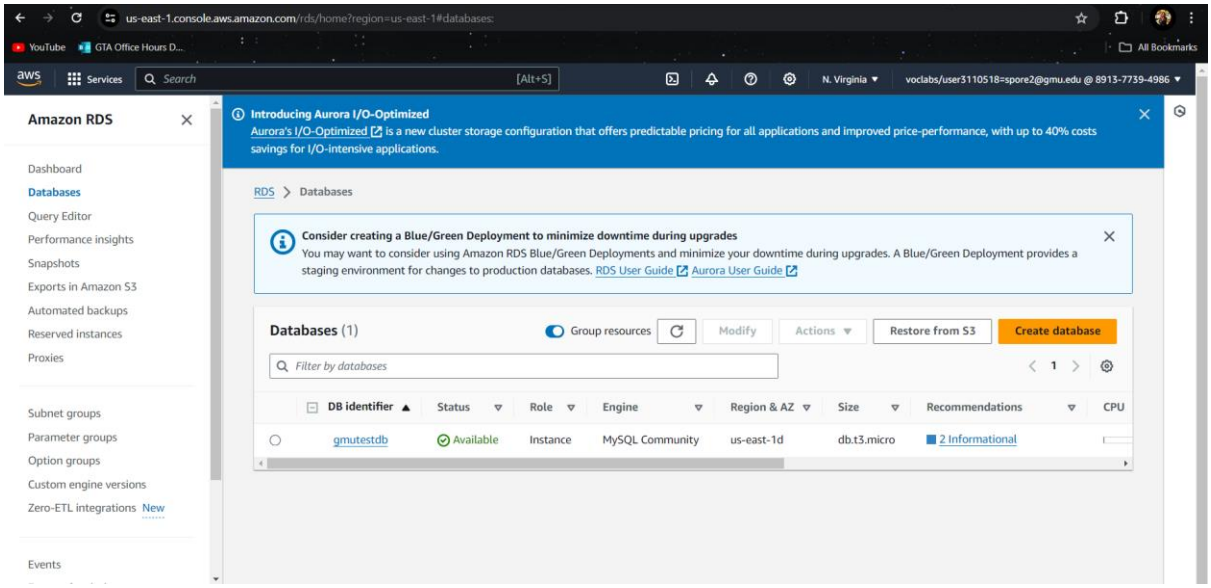
**Shivani Pore – G01333617**

Step 1: Initialize Project Using Spring Initializr

- Go to Spring Initializr.
- Select Maven Project, Java, and the version of Spring Boot you prefer.
- Add dependencies: Spring Web, Spring Data JPA, MySQL Driver.
- Generate the project and download the zip file.
- Extract the zip file and import the project into VSCode.

Step 2: Set Up Database

- Log in to your AWS Management Console and navigate to the RDS service.
- Create a new MySQL database instance.
- Configure the database with appropriate security settings and obtain the endpoint URL, username, and password.




Step 3: Configure Application Properties

- In your Spring Boot application, locate the application.properties file.
- Add the following properties to connect to your Amazon RDS instance:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://gmutestdb.ct2mg8sc0l3o.us-east-1.rds.amazonaws.com/gmutestdb
spring.datasource.username=admin
spring.datasource.password=swe645ass3
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
```

Step 4: Implement CRUD Operations

- Define a Survey entity class with annotations to map it to the database table.
- Create a repository interface extending JpaRepository for database operations.
- Develop a service class to handle business logic.
- Create a REST controller with mappings for CRUD operations (GET, POST, PUT, DELETE).

- Tested using postman locally.

Step 5: Containerize the Application

- Create a Dockerfile in the root of your project:

```
FROM --platform=linux/amd64 openjdk:latest
EXPOSE 8080
COPY target/*.jar test.jar
ENTRYPOINT ["java","-jar","/test.jar"]
```

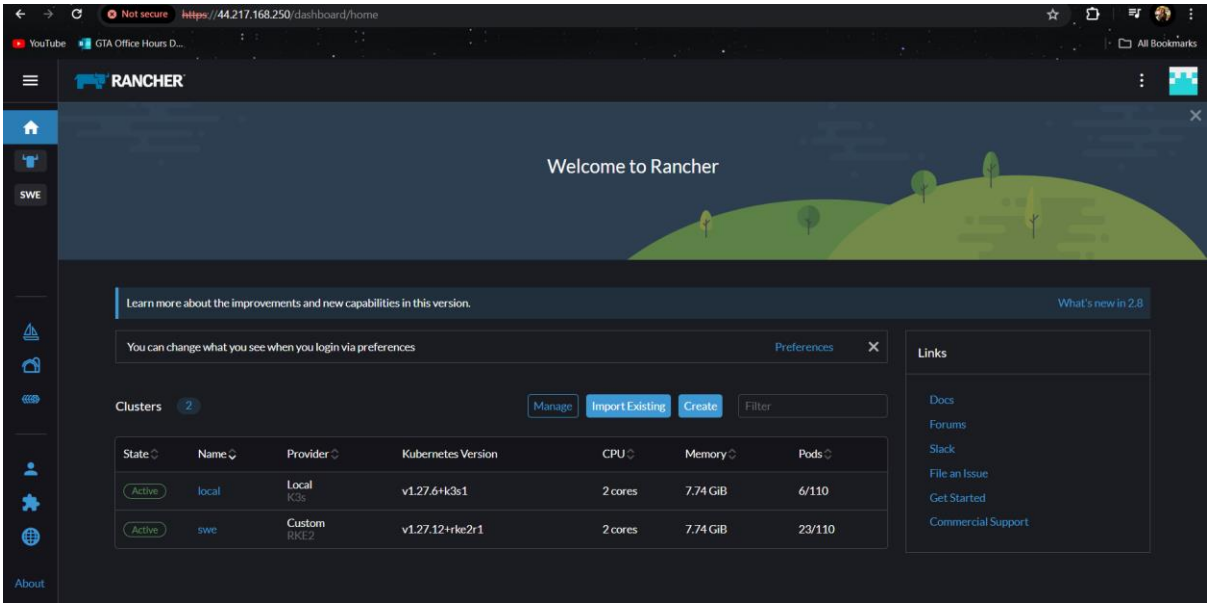
- Build the Docker image:
  `docker build -t swe .`

- Pushed to Docker Hub:
  `sudo docker login`
  `sudo docker tag student-form-img spore2/swe:latest`
  `sudo docker push spore2/swe:latest`

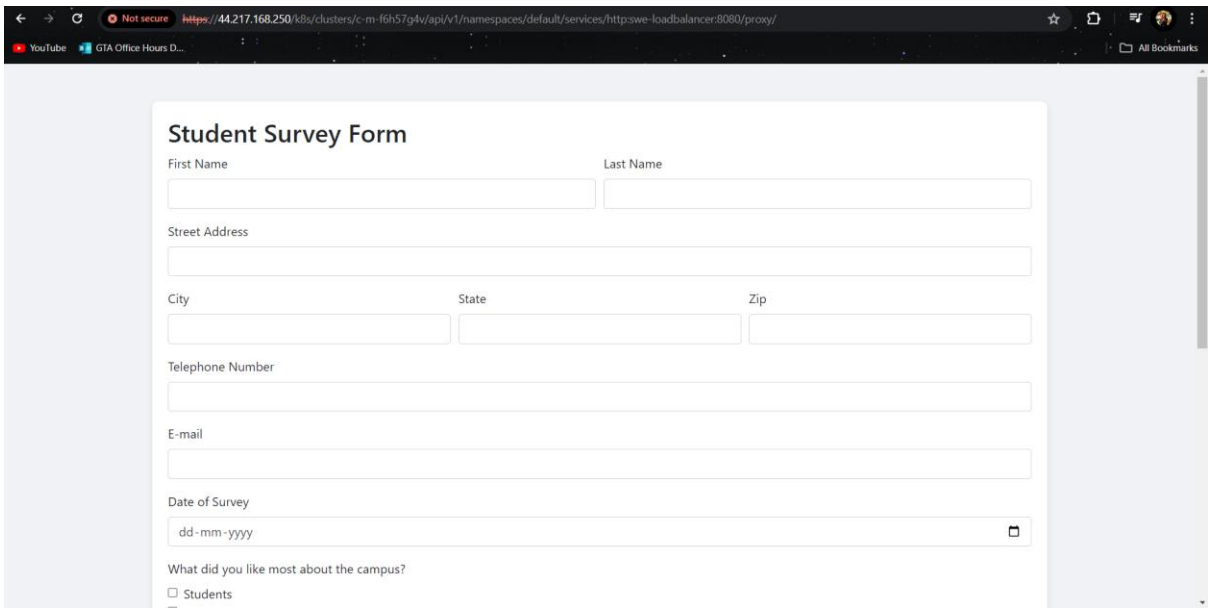
- Tested the Docker image locally to ensure it works as expected.
  `sudo docker run -d -p 8080:8080 swe`
  http://44.217.168.250:8080/

Step 6 :Deploy the Containerized Application on Kubernetes:

- Installed Kubernetes on the cluster:
  `sudo snap install kubectl –classic`
- Installed Rancher:
  `sudo docker run -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher`
- Accessed the Rancher UI by navigating to https://44.217.168.250/dashboard/home in web browser.
- Followed the on-screen instructions to set up Rancher.
- Clicked on Clusters in the top menu, then click Add Cluster.
- Chose From existing nodes (Custom) as the cluster type.
- Entered a name for cluster(swe) and did a basic setup.
- Clicked Next, then clicked Save.




- Rancher provided a command to run on EC2 instance to join the cluster.
- In the Rancher UI, navigated to cluster(ass2) and clicked Workloads in the top menu.
- Clicked Deploy.
- Entered a name for workload, select the namespace, and specify the number of replicas (e.g., 3) to ensure at least three pods are running all the time.
- In the Containers section, click Add Container and configured container(swe) with the image(spore2/swe) I pushed to the DockerHub.
- Launched Verified that the application is running correctly on Kubernetes by accessing the service endpoint

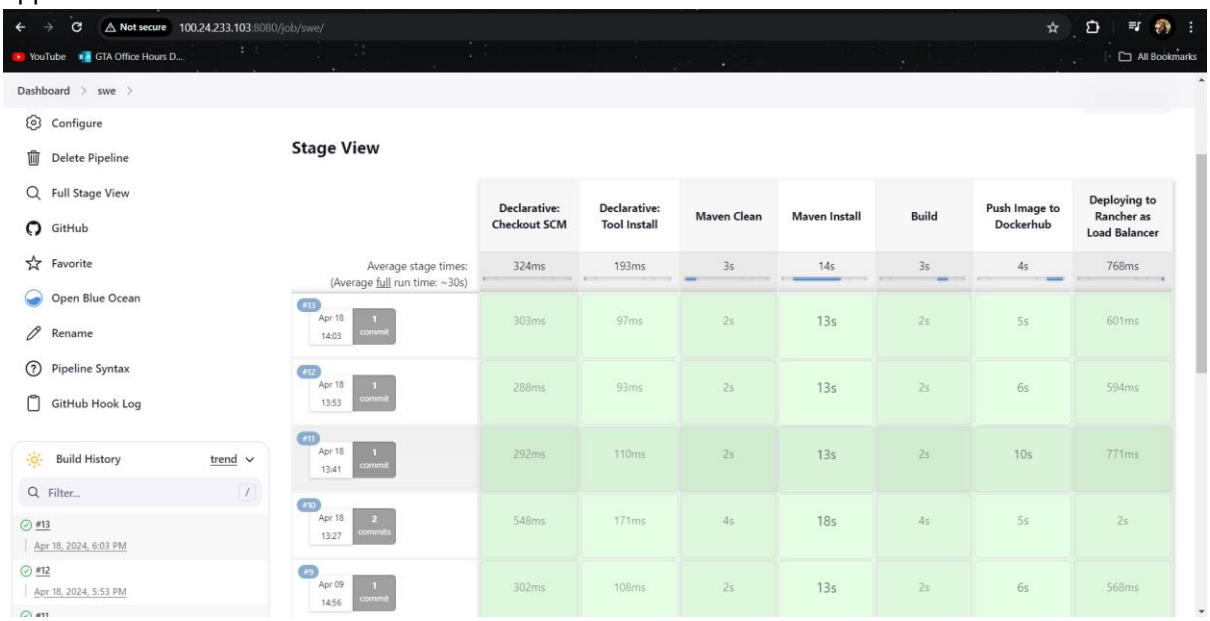



Step 7: Establish a CI/CD Pipeline with GitHub and Jenkins:

- Set up a GitHub repository to host web application code. https://github.com/spore2gmu/SWE645-3
- Install and configure Jenkins on a second EC2 instance (jenkins).

```
sudo apt update sudo apt install openjdk-11-jdk
curl -fsSL https://pkg.jenkins.io/debian□stable/jenkins.io-2023.key | sudo tee
\ /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins□keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt update
sudo apt install jenkins
sudo systemctl start jenkins
sudo systemctl enable Jenkins
```

- Opened Jenkins in the url : http://100.24.233.103:8080/
- Installed the necessary plugins in Jenkins, such as the Git, GitHub, Docker and Kubernetes. Created a Jenkins pipeline script (Jenkinsfile) that defined CI/CD workflow.
- Configured a webhook in your GitHub repository to trigger the Jenkins pipeline on every push to the repository.
- Navigated to the first cluster(rancher) and click on Kubeconfig File under Cluster Actions. Copied the contents of the kubeconfig file.
- In Jenkins pipeline script (Jenkinsfile), added a withCredentials block to securely pass the kubeconfig file as a variable.
- Used the file credential type to pass the kubeconfig file.
- Used the KUBECONFIG environment variable to set the kubeconfig file path in the Jenkins pipeline.
- Tested the CI/CD pipeline by making a changes to the application code, committing it to the GitHub repository, and verifying that Jenkins automatically builds and deploys the updated application to Kubernetes




Step 8: Tested CRUD Operations on Postman

- Added the Kubernetes load-balancer endpoint in the URL section for the GET, PUT, DELETE, CREATE operations https://44.217.168.250/k8s/clusters/cmf6h57g4v/api/v1/namespaces/default/services/http:swe-loadbalancer:8080/proxy/

- For authorization I added the bearer token that I got from the Kubeconfig file.