SWE 645 - Assignment-3

Team Name : Momtrimo

Team members :

1. Satya Subrahmanya Gautama Shastry Bulusu Venkata (G01477340)
2. Omtri Mohan Maheedhar Sai (G01478890)

1. Application Access URLs

- **Kubernetes Endpoint URL :** http://ec2-107-20-78-237.compute-1.amazonaws.com:31325/api/surveys

- **Source Code (GitHub) : https://github.com/GautamaShastry/SWE645_assign3**

2. Amazon RDS – MySQL Setup

Steps to Configure:

1. Log into your AWS Management Console.

2. Navigate to the **RDS** service from the search bar or Services menu.

3. Click **"Create Database"** and select the **MySQL** engine. Opt for the **Free Tier** template and choose the latest stable version available.

4. Under the "Settings" section:
   - Set a DB instance identifier (e.g., `database-1`).
   - Choose a master username and password, which will be used later to access the DB. In my case, I used “admin” as username.

5. In the **Instance Configuration**, select **db.t4g.micro** (or any Free Tier-compatible instance).

6. Under **Connectivity**, use the default VPC and subnet group. Ensure **Public Access** is set to **Yes**.

7. Create a new security group and allow public access to required ports by editing the **Inbound Rules**:

   - Add rules for port **3306** (MySQL) and choose **Anywhere** as the source.

8. Enable **Password Authentication**, assign a database name, and finalize by clicking **"Create Database"**.

MySQL Workbench Setup:

- Download the latest version of **MySQL Workbench** from MySQL Downloads.
- Install and launch the application.
- Set up a new connection:
  - **Hostname:** RDS endpoint
  `database-1.cjyikqgki6ok.us-east-1.rds.amazonaws.com`
  - **Username:** (your configured RDS username)
  - **Password:** (your RDS password)

3. Building a Student Survey Application Using Spring Boot

Overview:

We are creating a RESTful microservice using **Spring Boot** that performs full CRUD operations (Create, Read, Update, Delete) on student survey data stored in a MySQL database.

Project Setup via Spring Initializr

1. Navigate to Spring Initializr.
2. Choose the following options:
   - **Project Type:** Maven
   - **Language:** Java
   - **Spring Boot Version:** Latest stable release
   - **Group:** com.example
   - **Artifact:** student-survey(or any other name, I used momtrimo)
   - **Packaging:** JAR
   - **Dependencies:**
     - Spring Web
     - Spring Data JPA
     - MySQL Driver
3. Click **Generate**, then extract the downloaded `.zip` file.

IDE Setup and Configuration

1. Import the extracted project into **IntelliJ IDE for Enterprise Java Developers**.

2. Open `src/main/resources/application.properties` and configure database settings:

   spring.datasource.url=jdbc:mysql://<your-rds-endpoint>:3306/<db_name>

   spring.datasource.username=<your-username>

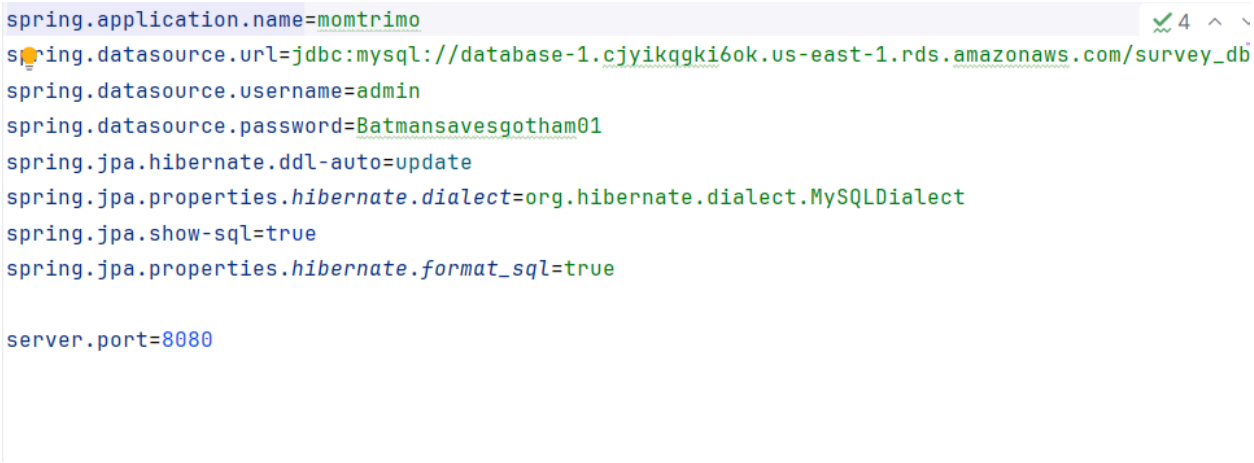
   spring.datasource.password=<your-password>

   spring.jpa.hibernate.ddl-auto=update

   spring.jpa.show-sql=true

   spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

   Below, is the screenshot of my application.properties environment variables:



   a. In my applications, there are mainly 4 packages: model(for defining the database entities, this shows how the Survey page details is stored in the database), repository(database connection), service(define the business logic, in this case, the main CRUD operations involved in my application, like creating the survey page, updating, etc), and finally, controllers(to define the API endpoints)
   b. Inside the `repository` package, define a JPA entity for the survey table with appropriate annotations (`@Entity`, `@Id`, etc.).
   c. Create a **Repository Interface** that extends `JpaRepository<Survey, Long>` to interact with the database.
   d. In the `controller` package, create a **REST Controller** that includes endpoints for:

   - Creating a new survey (POST)

- Retrieving all surveys (GET)
- Fetching a survey by ID (GET)
- Updating a survey (PUT)
- Deleting a survey (DELETE)

Testing the API with Postman(first, we check if the endpoints are working, in our localhost 8080, where the server runs, then, when we configure ec2 instance, and create kubernetes cluster, we can use that url):

Operation	**Method**	**Endpoint**
Create Survey	POST	`http://localhost:8080/api/surveys`
View All Surveys	GET	`http://localhost:8080/api/surveys`
Get Survey by ID	GET	`http://localhost:8080/api/surveys/{id}`
Update Survey	PUT	`http://localhost:8080/api/surveys/{id}`
Delete Survey	DELETE	`http://localhost:8080/api/surveys/{id}`

Use **JSON format** in the request body for POST and PUT operations:

Components Summary

- **Survey Entity:** Maps the survey table in MySQL using JPA

```java
// define the Survey entity, surveys is the table name in the MySQL database

package com.survey.momtrimo.model;
import jakarta.persistence.*;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

import java.time.LocalDate;

@Entity  Gautam
@Table(name = "surveys")
public class Survey {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "First Name is required")  2 usages
    @Column(nullable = false, name = "first_name")
    private String firstName;

    @NotBlank(message = "Last Name is required")  2 usages
    @Column(nullable = false, name = "last_name")
    private String lastName;

    @NotBlank(message = "Email is required")  2 usages
    @Email(message = "Invalid email format")
    @Column(nullable = false, name = "email")
```

- SurveyRepository: Interface extending **JpaRepository**, handles DB operations.

```java
// repository layer: connection to the database

package com.survey.momtrimo.repository;

import com.survey.momtrimo.model.Survey;
import org.springframework.data.jpa.repository.JpaRepository;

public interface SurveyRepository extends JpaRepository<Survey, Long> {  3 usages  Gautam
}
```

- **SurveyController:** Exposes REST endpoints and handles API logic.

```java
// define api endpoints

package com.survey.momtrimo.controller;



import com.survey.momtrimo.model.Survey;
import com.survey.momtrimo.service.SurveyService;
import jakarta.validation.Valid;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController  Gautam
@RequestMapping("/api/surveys")
public class SurveyController {
    private final SurveyService surveyService;  6 usages

    public SurveyController(SurveyService surveyService) { this.surveyService = surveyService; }

    @GetMapping  Gautam
```

- **SurveyService:** defines the CRUD operations performed on the application

```java
// service layer: define the business logic (CRUD operations)
C:\Users\gauta\OneDrive\Desktop\SWE645-Assignments\Assignment3\SWE645_assign3\momtrimo\Dockerfile
package com.survey.momtrimo.service;


import com.survey.momtrimo.model.Survey;
import com.survey.momtrimo.repository.SurveyRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service  3 usages  Gautam
public class SurveyService {
    private final SurveyRepository surveyRepository;  8 usages

    public SurveyService(SurveyRepository surveyRepository) { this.surveyRepository = surveyRepos

    public List<Survey> getAllSurveys() { return surveyRepository.findAll(); }
```

4. Dockerizing the Spring Boot Application

Objective:

We'll containerize the Spring Boot survey application using Docker, enabling easy deployment and portability across environments.

Steps to Dockerize the Application :

1. Install and Set Up Docker

- Download and install **Docker Desktop** from Docker Hub.
- Once installed, open your terminal and log in to Docker:

  docker login -u gautam26

2. Create a Dockerfile :
Inside the root of your Spring Boot project, create a file named Dockerfile with the following content:

```
FROM openjdk:23-jdk-slim

WORKDIR /app

COPY target/momtrimo-0.0.1-SNAPSHOT.jar momtrimo-0.0.1-SNAPSHOT.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "momtrimo-0.0.1-SNAPSHOT.jar"]
```

3. Build the Docker Image :

Navigate to the root of the project (where the Dockerfile is located) and run:

docker build -t gautam26/swe645:1.0

4. Run the Docker Container Locally

To verify it works:

docker run -p 8080:8080 gautam26/swe645:1.0

5. Push Image to Docker Hub

To make your image available for Kubernetes/Rancher:

docker push gautam26/swe645:1.0

5. Deploying the Containerized Application on Kubernetes Using Rancher

Overview

This section covers the deployment of the Dockerized Spring Boot application onto a Kubernetes cluster using **Rancher**. Rancher helps in managing and orchestrating the Kubernetes clusters efficiently.

Step-by-Step Process for Kubernetes Deployment via Rancher

a. Setup EC2 Instances for Kubernetes and Rancher

- Log into the **AWS EC2 Console** and create three EC2 instances. Configure them with the following:
    - **AMI:** Ubuntu Server 22.04 LTS (HVM), SSD volume type.
    - **Security Groups:** Allow inbound traffic on ports `8080`, `80`, `443`, and `22` from anywhere.
    - **Storage:** Assign 30 GB storage to each instance.
    - **Outbound Rules:** Allow all traffic.
- Assign **Elastic IPs** to each instance for public access.

b. Install Docker on EC2 Instances

SSH into **Instance 1** and **Instance 2**, then run the following commands to install Docker:

sudo su -

sudo apt-get update

sudo apt install docker.io

c. Start the Rancher Server on Instance 1

- On **Instance 1**, run the following command to start the Rancher server:

- sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher
- Once the installation is complete, run sudo docker ps to view the current docker instance, and note the container ID.
- In the UI, copy the password command and paste in instance1 console to get the default password to enter in the RancherUI for the first time login. The noted container-ID must be replaced in the below command sudo docker logs container-ID 2>&1 | grep “Bootstrap Password:”
- Use the generated password to login and once login, select the option “Set a specific password to use” to set a custom password for future

d. Setting up the Kubernetes Cluster on Instance 2

- On **Instance 2**, log in to Rancher UI, create a new cluster using **Custom Nodes** (RKE - Rancher Kubernetes Engine).

- Assign the following roles to the nodes in your cluster:

    - **etcd** (Cluster data storage)

    - **Control Plane** (Master node)

    - **Worker** (Application node)

- Select **Insecure:** to bypass TLS verification if you use a self-signed certificate.

- Rancher will provide a **registration command** for **Instance 2**. Run this on Instance 2’s terminal to register it to the cluster.

Once the cluster state switches to **active**, the cluster is ready.

e. Install Kubernetes Tools on Instance 2

Install **kubectl** (Kubernetes CLI) on **Instance 2** to interact with the Kubernetes cluster:

snap install kubectl --classic

Next, download the **KubeConfig file** from Rancher and copy it to the appropriate directory:

mkdir -p ~/.kube

mv <downloaded-kubeconfig-file> ~/.kube/config

chmod 600 ~/.kube/config

f. Deploy the Application on Kubernetes

Now that your cluster is set up, deploy the application using **deployment.yaml** and **service.yaml** files.

1. **Create `deployment.yaml`:** This file defines how the application is deployed on the Kubernetes cluster, including the Docker image and resource configurations.
2. **Create `service.yaml`:** The `service.yaml` exposes the deployed application to the external network.

Apply the deployment and service files using: **kubectl apply -f deployment.yaml** and **kubectl apply -f servie.yaml**

g. Configure Security Groups and Expose the Application

Once the service is applied, use the following command to retrieve the **NodePort**:

kubectl get service

Add the NodePort to the EC2 instance's **Security Group** as a custom TCP rule and allow access from anywhere.

h. Access the Application

You can now access the deployed application by visiting the following URL:

http://ec2-107-20-78-237.compute-1.amazonaws.com:31325/api/surveys

6. Automating the Deployment Pipeline with Jenkins

Overview

This section demonstrates how to use **Jenkins** to automate the build, test, and deployment of a Spring Boot application onto a Kubernetes cluster. Jenkins enables continuous integration and delivery (CI/CD), streamlining the deployment process from code commit to production.

Step-by-Step Jenkins CI/CD Pipeline Setup

a. Launch an EC2 Instance for Jenkins

- Go to your **AWS EC2 Console** and launch a new EC2 instance.

- Use the following configuration:
    - **AMI:** Ubuntu Server 22.04 LTS
    - **Security Group:** Allow inbound traffic on ports 22, 8080, and 443
    - **Storage:** Minimum 30 GB
- Assign an **Elastic IP** for stable access to the Jenkins dashboard.

b. Install Jenkins

SSH into the EC2 instance and run the following commands:

ssh -i survey-backend-key.pem ubuntu@<IP address of instance3>

c. Access Jenkins Dashboard

- `Install the necessary packages like docker, jenkins using sudo apt-install`
- `Install jdk using sudo apt install openjdk-17-pre-headless`
- `Install and add jenkins repository key using  curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null`
- `Start the jenkins service using sudo systemctl start jenkins`
- Open your browser and go to: `http://ec2-18-208-7-119.compute-1.amazonaws.com:8080`
- Get the **Administrator password**: `sudo cat var/lib/jenkins/secrets/initialAdminPassword`
- Paste the password into the Jenkins setup wizard.
- Install **Suggested Plugins** and complete the initial setup.

d. Configure Jenkins with Required Tools

Install necessary tools in Jenkins:

- **Docker** (to build images)
- **Git** (for source control)
- **Kubernetes CLI (kubectl)** (for deployments)

e. Install Jenkins Plugins

From the Jenkins dashboard:

- Go to **Manage Jenkins > Plugins**
- Install the following:
    - **Docker Pipeline**
    - **Kubernetes CLI Plugin**
    - **Pipeline**
    - **Git Plugin**

f. Add the credentials of Github and DockerHub using username and password, and kubeconfig file using the secret key.

g. Create a Jenkins Pipeline Job

1. Click **New Item** > Select **Pipeline** > Name it accordingly.(cicdPipeline)

   Under **Build Triggers**, select **Poll SCM** and set the schedule to `* * * * *` for every minute

2. Scroll down to the **Pipeline** section.
3. Choose **Pipeline script** as follows:

   select **Pipeline script from SCM** and choose **Git** as the SCM. - Provide the GitHub repository URL and select the saved credentials.

h. Triggering the Pipeline

- Make a commit to the Github to trigger the build.

7. Verifying the Complete CI/CD Workflow

Objective

This section walks through the process of validating the entire DevOps pipeline—from code changes in GitHub to automated deployment on Kubernetes—ensuring that everything is functioning as expected.

Step 1: Make a Code Change

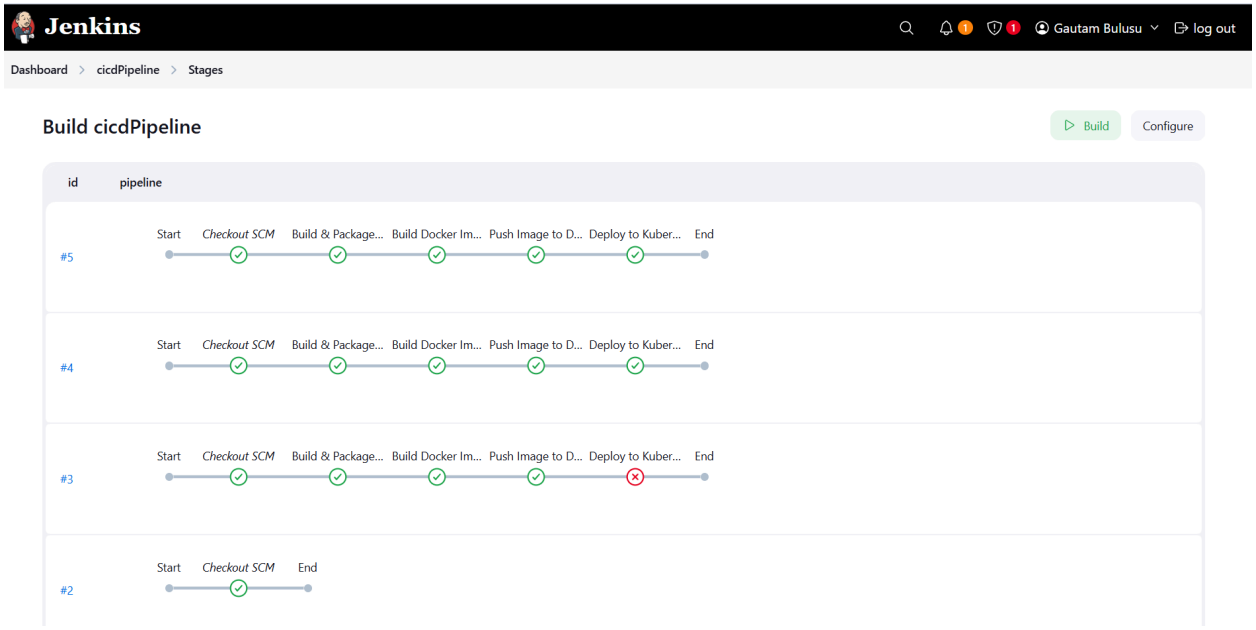
- Navigate to your local clone of the Spring Boot application or modify files directly in your GitHub repository.
- For testing, make a small change such as:
    - Updating the welcome message
    - Editing HTML or CSS styles
    - Modifying application logic
- Commit and push the changes to the **main** branch of your GitHub repo.

```
git add .

git commit -m "commit message"

git push
```

Step 2: Observe Jenkins Pipeline Execution

- Open your **Jenkins dashboard**.
- Under your pipeline job, you should see a new build triggered automatically (if webhooks are configured).
- The pipeline will follow these stages:
    1. Clone updated code from GitHub
    2. Build the Spring Boot application
    3. Create a Docker image and tag it
    4. Push the image to Docker Hub
    5. Apply Kubernetes manifests for deployment

Each stage should display logs to verify success.



Step 3: Check Docker Hub for the Updated Image

- Go to hub.docker.com and open your repository.
- Confirm the presence of the new image with the updated timestamp or commit hash.

Step 4: Validate Deployment on Kubernetes

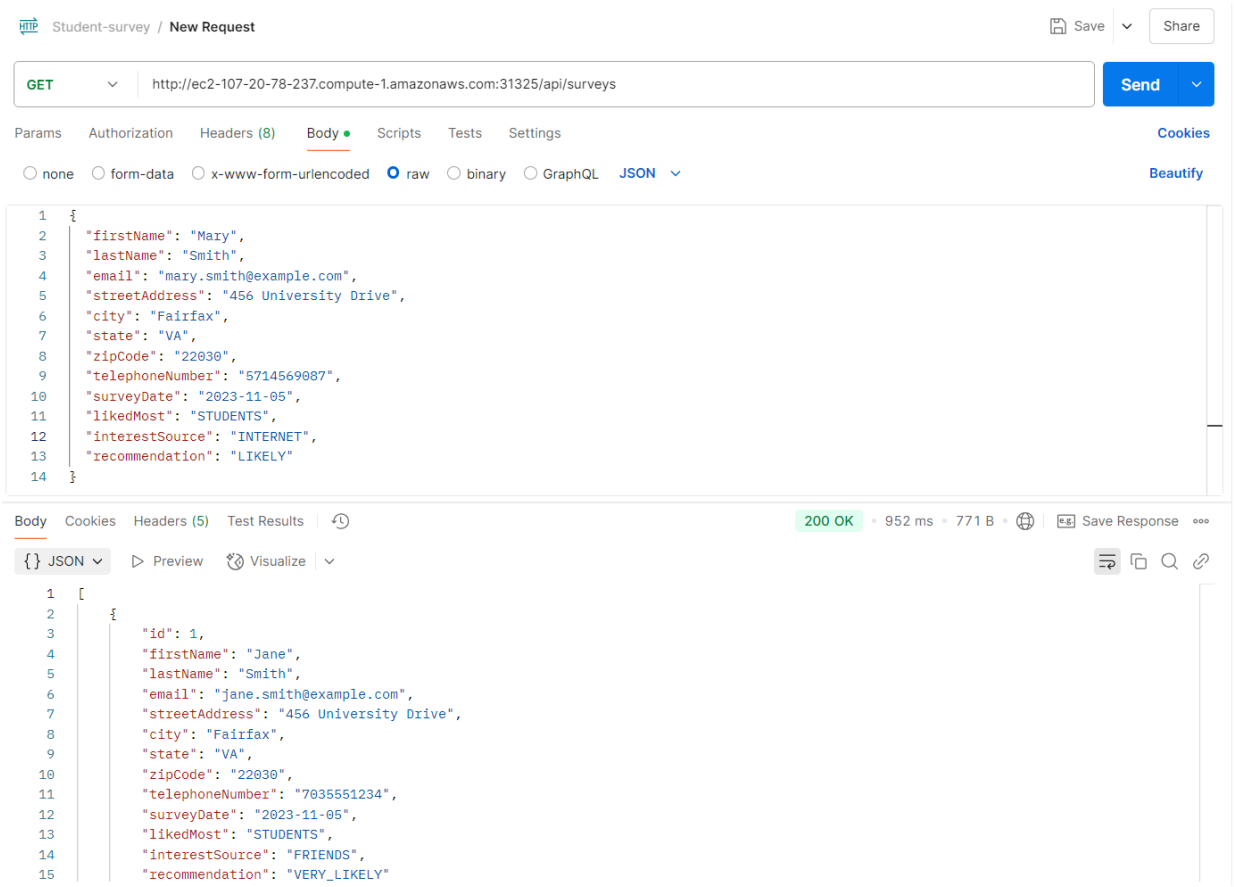
- SSH into your Kubernetes master node and run:

```
kubectl get pods

kubectl get svc
```

- Check that the updated pod is running and the service is active.
- If a LoadBalancer or NodePort is used, access the app using the provided external IP or port.

Step 5: Open the Deployed Application in Postman

- Use the public IP or DNS of your Kubernetes cluster (or AWS Load Balancer if used) to view the application.
- You should now see the updated content reflecting your latest commit.



Team Contributions:

1. **Gautama Sastry Bulusu (G01477340) :** Spearheaded the setup and deployment of the Spring Boot microservices, integrating REST APIs with MySQL for CRUD operations. Took charge of configuring Docker containers and managing image builds, ensuring smooth execution within local and cloud environments. integrating GitHub and Docker Hub to automate builds and deployments efficiently.

2. **Omtri Mohan Maheedhar Sai (G01478890) :** Handled the Kubernetes and Rancher deployment pipeline, creating and applying YAML configurations to launch services. Set up Jenkins for CI/CD workflows. Also includes Documentation and Video presentation.