

SWE-645-ASSIGNMENT 3

We need to execute the following steps which are required to complete Assignment 3:

a. Set Up the Project

1. Use [Spring Initializr](#) to create a Maven project:
 - Dependencies: Spring Web, Spring Data JPA, MySQL Driver, Spring Boot DevTools.
 - Import the project into your IDE (Eclipse EE/IntelliJ IDEA).

b. Implement the Microservices

- **Entities:** Create a StudentSurvey class to represent the survey data fields.
- **Repository:** Define a JPA repository for CRUD operations.
- **Service Layer:** Create a service class for business logic.
- **Controller Layer:** Implement RESTful endpoints for CRUD operations.

c. Database Configuration

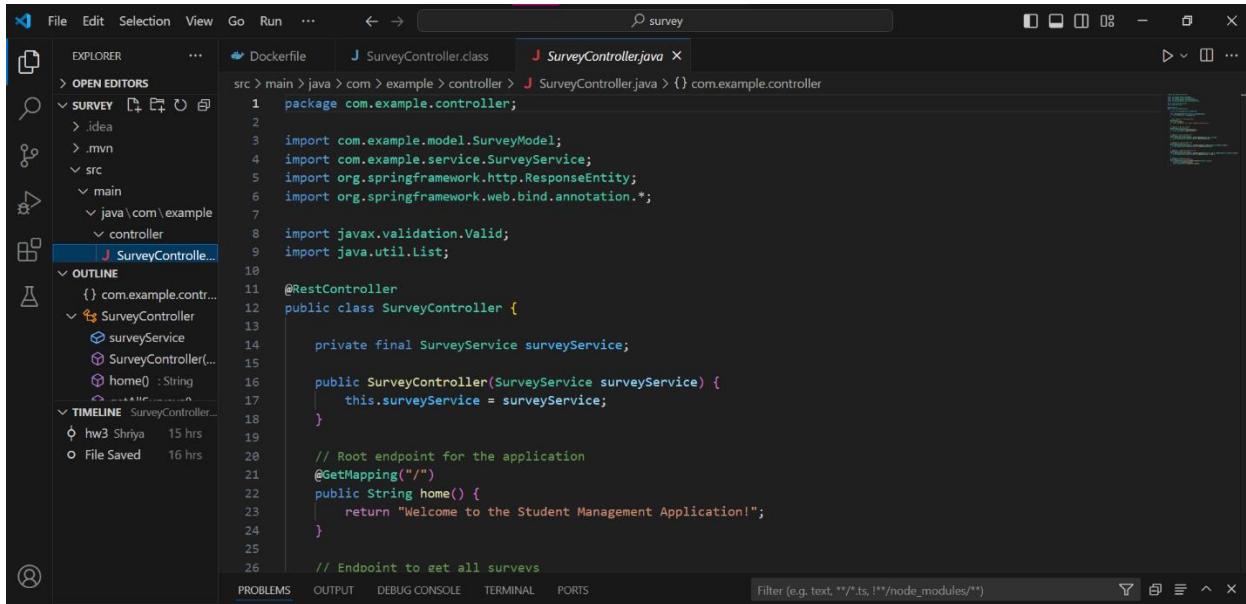
- Use Amazon RDS for a MySQL database:
 - Create an RDS instance in **Development mode**.

Then, we need to perform the following steps:

1. **Creating and Managing Docker Images:** Develop a containerized image of the application using Docker and ensure its proper configuration. Upload the finalized image to Docker Hub using Docker Desktop for efficient sharing and deployment.
2. **AWS EC2 Configuration for Kubernetes Deployment:** Configure AWS EC2 instances to serve as the infrastructure for deploying the application within a Kubernetes cluster. Utilize Rancher as a management platform to streamline cluster operations and deployment processes.
3. **Rancher Installation and Configuration:** Install and configure Rancher, a robust Kubernetes management tool, to simplify cluster setup and application orchestration. Ensure all necessary components are properly aligned to support the deployment pipeline.
4. **Application Deployment via Rancher UI:** Leverage the intuitive Rancher user interface to deploy the application seamlessly. Set up and manage a Kubernetes cluster through the UI, enabling automated scaling and resilience for the deployed application.
5. **Jenkins Installation on AWS EC2:** Provision an AWS EC2 instance specifically for installing and operating Jenkins, a powerful automation server. Ensure the instance is optimized for running CI/CD pipelines with secure and scalable configurations.

6. **GitHub Repository Setup for Project Codebase:** Create and configure a dedicated GitHub repository to host the project's source code and related assets. Enable version control and collaboration tools for effective team contributions and project management.
7. **Building and Executing a CI/CD Pipeline with Jenkins:** Construct a comprehensive CI/CD pipeline using Jenkins to automate the build, test, and deployment processes. Fully configure the pipeline to streamline application updates and maintain a reliable release workflow.

Described below is the screenshot of how the assignment is executed:



```

File Edit Selection View Go Run ...
Dockerfile SurveyController.class SurveyController.java
src > main > java > com > example > controller > SurveyController.java > {} com.example.controller
1 package com.example.controller;
2
3 import com.example.model.SurveyModel;
4 import com.example.service.SurveyService;
5 import org.springframework.http.ResponseEntity;
6 import org.springframework.web.bind.annotation.*;
7
8 import javax.validation.Valid;
9 import java.util.List;
10
11 @RestController
12 public class SurveyController {
13
14     private final SurveyService surveyService;
15
16     public SurveyController(SurveyService surveyService) {
17         this.surveyService = surveyService;
18     }
19
20     // Root endpoint for the application
21     @GetMapping("/")
22     public String home() {
23         return "Welcome to the Student Management Application!";
24     }
25
26     // Endpoint to get all surveys
}

```

The screenshot shows the IntelliJ IDEA interface with the SurveyController.java file open in the editor. The code defines a REST controller for a survey service, with endpoints for getting all surveys and a root endpoint returning a welcome message. The IDE also shows the project structure in the Explorer panel and a timeline of recent changes in the Timeline panel.

1. FileStructure:

The project follows a clear and organized structure, with folders like com.example.controller, com.example.service, and com.example.model, ensuring modularity and readability.

2. ControllerFocus:

The SurveyController.java file, located in the controller package, serves as the main focus, handling the controller layer in the Spring Boot architecture.

3. RESTAPIConfiguration:

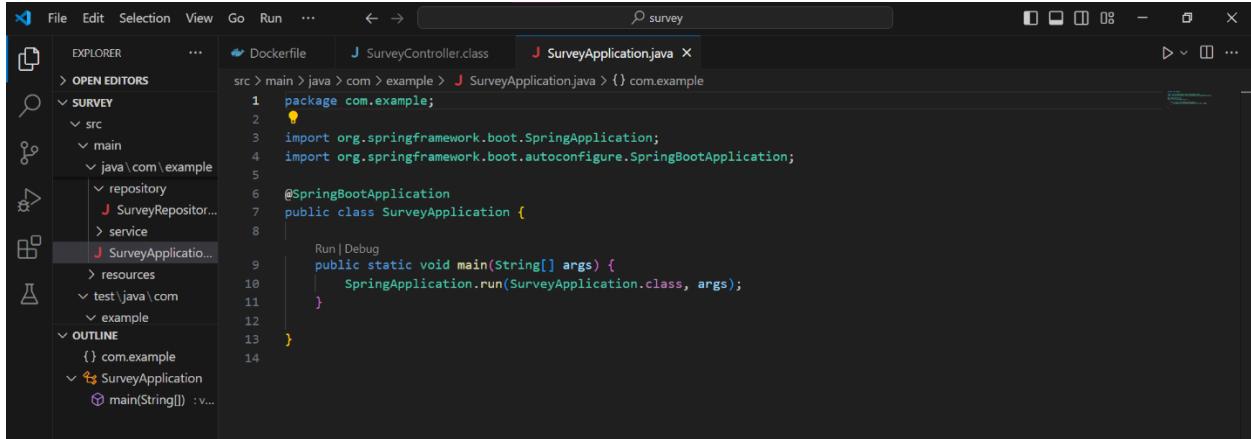
Annotated with @RestController, the SurveyController manages HTTP requests and responses in JSON format, with a root endpoint ("") returning a welcome message: "Welcome to the Student Management Application!".

4. ServiceLayerIntegration:

The SurveyController utilizes a SurveyService instance, injected via constructor-based dependency injection, ensuring a robust link between the controller and service layers.

5. PostmanTesting:

Endpoints are ready for testing using Postman, with GET / serving as a health check endpoint, returning the welcome message to verify application functionality.



The screenshot shows a dark-themed IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** survey
- Explorer:** OPEN EDITORS, SURVEY (src, main, java\com\example, repository, service, SurveyApplication.java, resources, test.java\com, example)
- Outline:** OUTLINE (SurveyApplication, main(String[]))
- Editor:** SurveyApplication.java (Content shown below)

```
src > main > java > com > example > SurveyApplication.java > {} com.example
1 package com.example;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class SurveyApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(SurveyApplication.class, args);
11     }
12 }
13
14 }
```

1. MainClass:

The SurveyApplication class serves as the main entry point for the Spring Boot application, marked with the @SpringBootApplication annotation.

2. SpringBootFeatures:

The @SpringBootApplication annotation enables essential Spring Boot features, including auto-configuration, component scanning, and configuration setup.

3. ApplicationInitialization:

The SpringApplication.run(SurveyApplication.class, args) method starts the application, using SurveyApplication as the primary configuration source.

4. FileOrganization:

The SurveyApplication.java file is located in the com.example package, showcasing a clean and professional project structure aligned with Spring Boot best practices.

5. RunandDebugOptions:

IDE integration provides convenient options to run or debug the application directly, streamlining testing and troubleshooting processes.

The screenshots demonstrate the integration of an AWS RDS MySQL database instance into an application. The top screenshot shows the AWS RDS Connectivity & security tab, where the database endpoint is listed as `database-1.c3kyesugi7z8.us-east-1.rds.amazonaws.com` and the port is `3306`. The VPC and subnet group information are also visible. The bottom screenshot shows an IDE editor displaying the `application.properties` file, which contains the database connection details:

```

spring.datasource.url=jdbc:mysql://database-1.c3kyesugi7z8.us-east-1.rds.amazonaws.com:3306/database_1
spring.datasource.username=admin
spring.datasource.password=Shriyaatluri2112
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

```

The above screenshots show the setup and connectio of the AWS RDS to the application.

1. AmazonRDSDashboardOverview:

The RDS management console displays details of a MySQL database instance, with a focus on the *Connectivity & security* tab.

2. DatabaseConnectionDetails:

The instance provides a database endpoint (`database-1.c1xksyeqjz78.us-east-1.rds.amazonaws.com`) and port number (3306), essential for connecting the application to the RDS database.

3. NetworkingConfiguration:

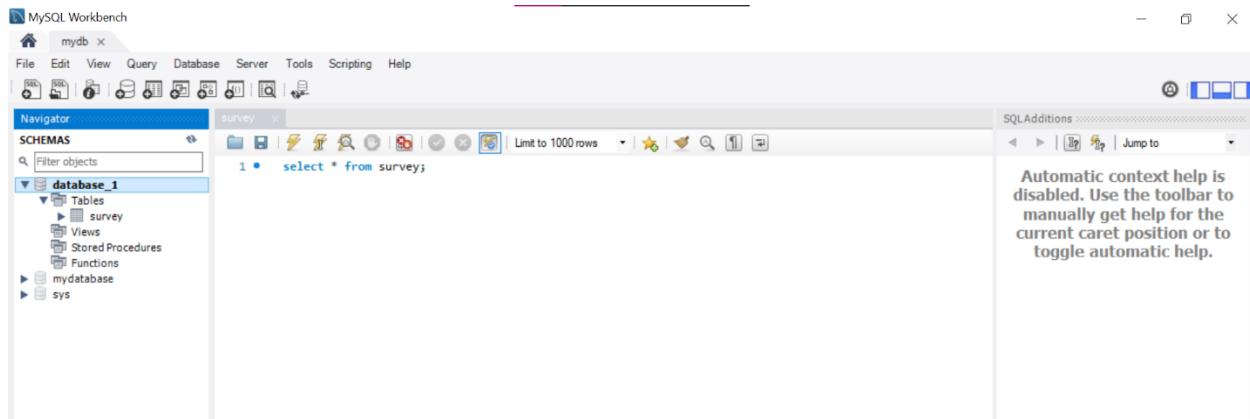
The database resides in the `us-east-1a` availability zone, within a specific VPC (`vpc-0a1bbf8066c194c8d`) and subnet setup.

4. SecuritySetup:

The database instance is associated with a security group (`default sg-0a546454e510e9f5f`) and is publicly accessible, allowing external connections while maintaining access control through credentials and security rules.

5. SSL Encryption:

SSL certificate authority details are included to ensure secure, encrypted connections between the client and the database, safeguarding sensitive data during communication.



Here, we are connecting an AWS RDS MySQL database to our local MySQL client which involves several steps. Here's a comprehensive guide:

1. Set Up Your AWS RDS Instance

1. Log in to AWS Management Console:

- Go to [AWS Console](#).

2. Navigate to RDS:

- Search for **RDS** in the AWS Management Console and open it.

3. Create a New Database:

- Click on **Create database**.
- Select **MySQL** as the database engine.
- Choose **Standard create**.
- Specify settings like:
 - DB Instance Identifier:** A name for your RDS instance (e.g., mydb).
 - Master Username:** Set a username (e.g., admin).
 - Master Password:** Set a strong password.

4. Configure Connectivity:

- Under **Connectivity**, ensure that:
 - A **VPC** is selected.
 - Public Access** is set to **Yes** (if you want to connect from your local machine).
 - Add a **VPC security group** to allow inbound traffic.

5. Create the Database:

- a. Review the settings and click **Create database**.
- b. Wait for the database instance to launch.

2. Configure Security Group

1. **Open the EC2 Dashboard:**
 - a. In the AWS Management Console, go to **EC2** and navigate to **Security Groups**.
2. **Find the Security Group for Your RDS Instance:**
 - a. Locate the security group associated with your RDS instance.
3. **Edit Inbound Rules:**
 - a. Add a new rule:
 - i. **Type:** MySQL/Aurora
 - ii. **Protocol:** TCP
 - iii. **Port Range:** 3306
 - iv. **Source:** Choose **My IP** to restrict access to your current IP address or **0.0.0.0/0** (not recommended for production) to allow access from anywhere.
4. Save the rule.

3. Retrieve Endpoint and Port

1. **Go to Your RDS Instance:**
 - a. Navigate back to the RDS service in AWS.
2. **Find the Endpoint:**
 - a. Select your RDS instance and look for the **Endpoint** under the **Connectivity & security** section.
 - b. Note the **Endpoint** and **Port** (default is 3306).

4. Connect to AWS RDS Using MySQL Client

1. **Install MySQL Client** (if not installed):
 - a. For Ubuntu/Debian:

```
bash  
Copy code  
sudo apt update  
sudo apt install mysql-client
```

- b. For macOS:

```
bash  
Copy code  
brew install mysql
```

- c. For Windows, download and install MySQL Workbench or MySQL Shell.
- 2. **Connect to RDS:** Open a terminal or MySQL client and run:

```
bash  
Copy code  
mysql -h <endpoint> -P <port> -u <username> -p
```

Replace:

- a. <endpoint>: Your RDS endpoint (e.g., mydb.123456789012.us-east-1.rds.amazonaws.com).
- b. <port>: The port number (default: 3306).
- c. <username>: The username you set during RDS setup (e.g., admin).

Example:

```
bash  
Copy code  
mysql -h mydb.123456789012.us-east-1.rds.amazonaws.com -P 3306 -u admin -p
```

- 3. **Enter Password:**
 - a. You will be prompted to enter the password. Enter the master password you set during the RDS setup.
- 4. **Verify Connection:**
 - a. If successful, you'll see the MySQL prompt:

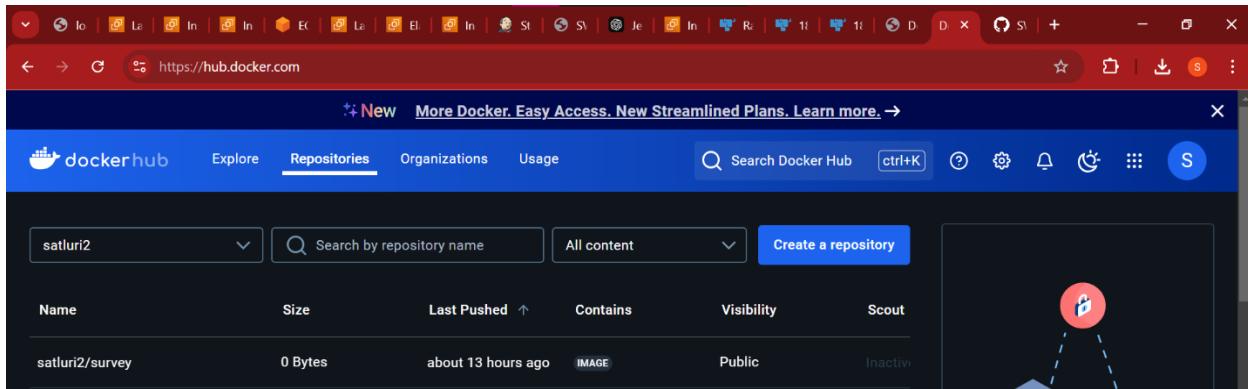
```
bash  
Copy code  
mysql>
```

5. Test the Connection

- Run SQL commands to test the connection. For example:

```
sql  
Copy code
```

SHOW DATABASES;



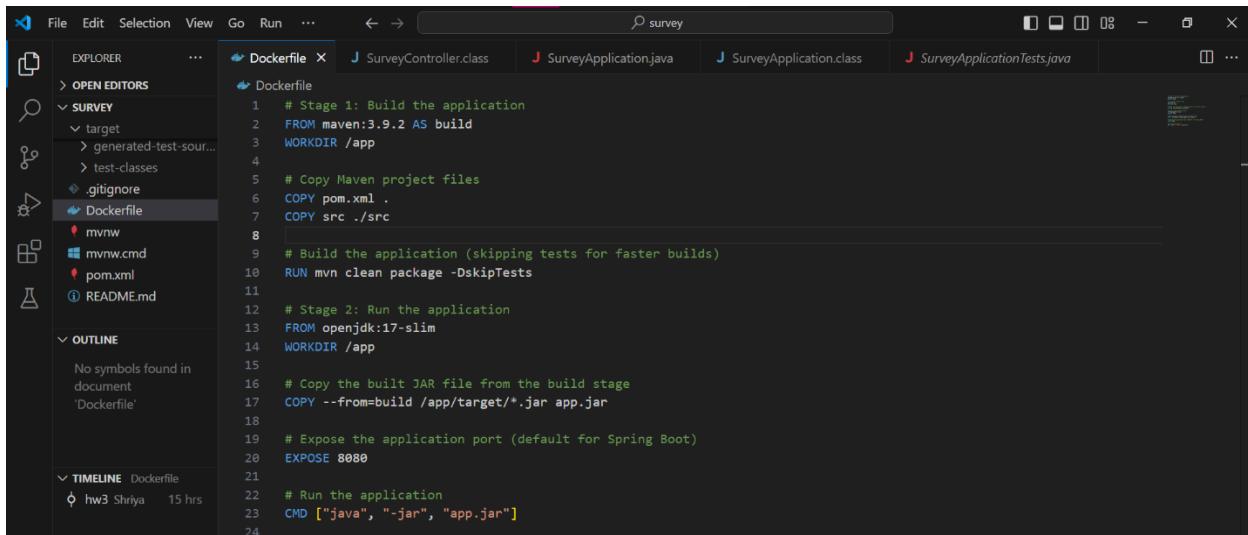
Docker image on dockerhub :

- Visit [Docker Hub](#) and create an account. Additionally, download and install Docker Desktop for seamless operation.
- Use your newly created account to log in to Docker Desktop on your computer and Docker Hub via a web browser.
- Create a "Dockerfile" in the same directory as your ".war" file. Ensure both the "Dockerfile" and the "satluri2/survey.war" file are located in the same folder.

A screenshot of a web browser displaying the Docker Hub repository page for 'satluri2/survey'. The URL is https://hub.docker.com/repository/docker/satluri2/survey/general. The page shows basic repository information: General tab selected, Tags, Builds, Collaborators, Webhooks, and Settings. It displays the repository name 'satluri2/survey' with a link to 'Public View'. Below this, it says 'Last pushed about 13 hours ago' and 'This repository does not have a description' with an edit icon. There's also a note 'This repository does not have a category' with an edit icon. On the right, there's a 'Docker commands' section with the command 'docker push satluri2/survey:tagname'. Below this, there's a 'Tags' section showing one tag: 'latest' (Image type, pushed 13 hours ago). To the right, there's an 'Automated builds' section with a note about connecting to GitHub or Bitbucket for automatic builds.

satluri2/survey:latest

Creation of docker file:



The screenshot shows the Docker Desktop interface with the Dockerfile tab selected. The Dockerfile content is as follows:

```
1 # Stage 1: Build the application
2 FROM maven:3.9.2 AS build
3 WORKDIR /app
4
5 # Copy Maven project files
6 COPY pom.xml .
7 COPY src ./src
8
9 # Build the application (skipping tests for faster builds)
10 RUN mvn clean package -DskipTests
11
12 # Stage 2: Run the application
13 FROM openjdk:17-slim
14 WORKDIR /app
15
16 # Copy the built JAR file from the build stage
17 COPY --from=build /app/target/*.jar app.jar
18
19 # Expose the application port (default for Spring Boot)
20 EXPOSE 8080
21
22 # Run the application
23 CMD ["java", "-jar", "app.jar"]
```

1. Open the command prompt in the directory where the Dockerfile and satluri/survey2.war file are located.
2. To create a Docker image, run the command:
3. docker build -t satluri/survey .
4. Docker Desktop will build an image for the application on your local machine. To run the application on localhost, execute:
5. docker run -it -p 8081:8080 satluri/survey
6. Ensure the server is running, and the application is accessible on localhost:8080.
7. Open a browser and navigate to localhost:8080. Append the URL with /survey, which corresponds to the .war file created using the command:

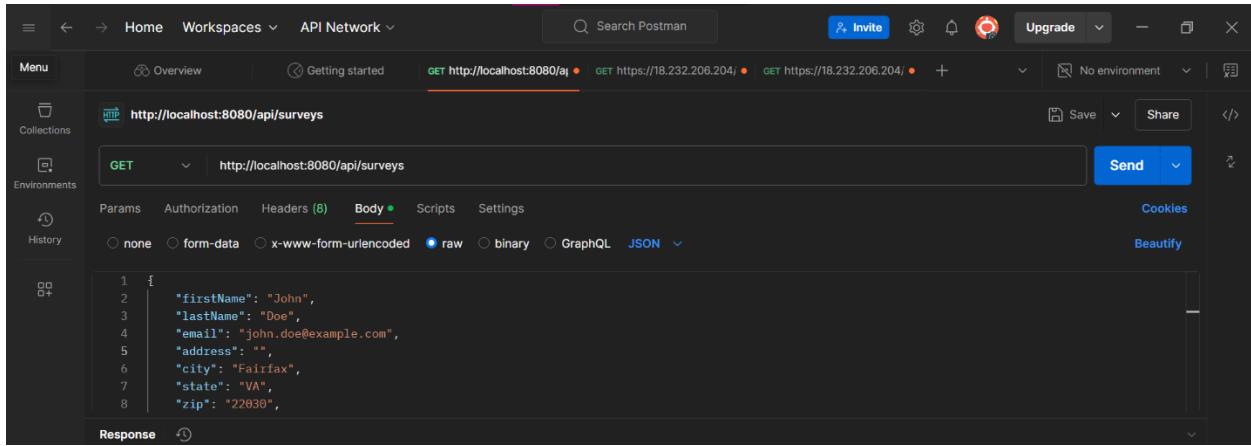
The application's webpage should appear in the browser.

URL:

<http://localhost:8080/survey/>

8.Import ant Docker commands are:

Docker build -t and docker push



The screenshot shows the Postman application interface. In the top navigation bar, there are links for Home, Workspaces, API Network, and a search bar labeled "Search Postman". On the right side of the header, there are buttons for Invite, Upgrade, and a close button. Below the header, the main workspace shows a collection named "Overview" and a specific request for "http://localhost:8080/api/surveys". The request method is set to "GET". The "Body" tab is selected, showing a raw JSON payload:

```
1 {
2   "firstName": "John",
3   "lastName": "Doe",
4   "email": "john.doe@example.com",
5   "address": "",
6   "city": "Fairfax",
7   "state": "VA",
8   "zip": "22030",
```

On the right side of the interface, there are buttons for Save, Share, Send, Cookies, and Beautify.

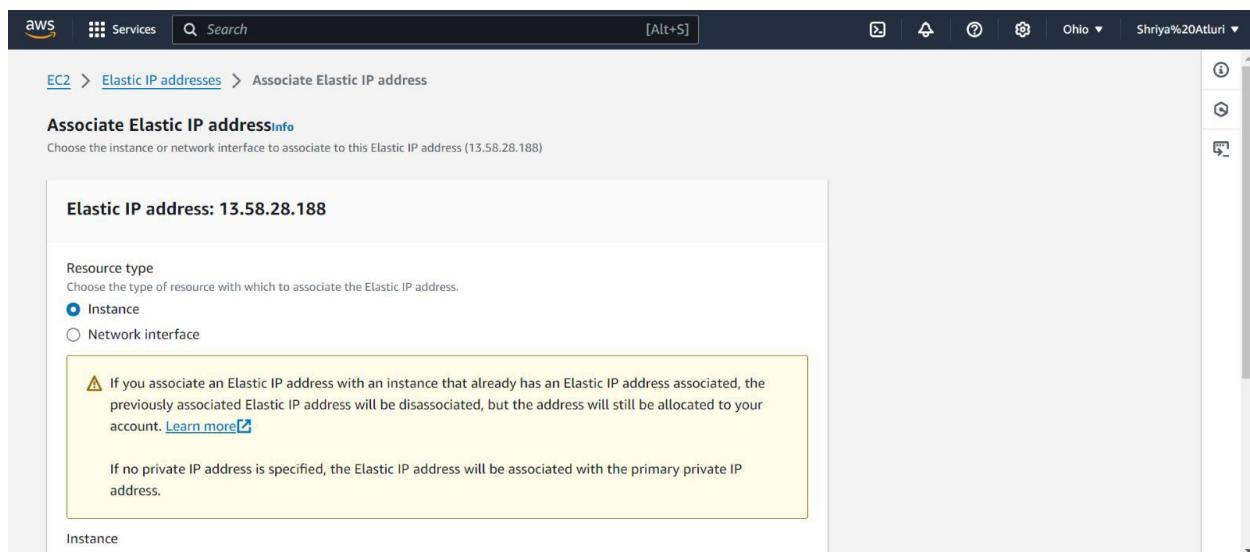
Go to postman and check the link : <http://localhost:8080/api/surveys>

- | | | | |
|--|----------------|-----------|-----------------|
| 1. API | Request | in | Postman: |
| The user is testing the REST API endpoint <code>http://localhost:8080/api/surveys</code> using Postman. | | | |
| 2. HTTP | | | Method: |
| The GET method is used, which retrieves data from the server at the specified endpoint. | | | |
| 3. Response | | | Payload: |
| The server responds with a JSON object containing survey-related data fields like <code>firstName</code> , <code>lastName</code> , <code>email</code> , <code>address</code> , <code>city</code> , <code>state</code> , and <code>zip</code> . | | | |
| 4. Successful | | | Request: |
| The response suggests that the server is running on <code>localhost:8080</code> , and the API is functioning correctly by returning valid data in JSON format. | | | |

Setting up of Rancher:

For this assignment, we'll require two AWS EC2 instances.

1. Log in to AWS Academy and access AWS. Navigate to the EC2 service and open the dashboard.
2. Click on "Launch Instance" to create a new instance.
3. Name the instance "Rancher-Master". Choose the AMI as "Ubuntu Server 22.04 LTS (HVM) SSD Volume Type". Set the instance type to "t3.large".
4. Select an existing key pair, in this example, it is "645". Check the boxes to allow HTTP and HTTPS traffic from the internet.
5. Increase the default storage from 8 GiB to 30 GiB.
6. Finally, click the "Launch instance" button to start the instance.



- Now, click on the instance and go to the Security tab. Here, in the “inbound rules”/“Outbound rules” section, click on the security group wizard.
- Scroll down and from either of the “inbound”/“outbound” tab click on edit rule option.
- Here, add a rule using the following configuration, Type : “Custom TCP”, Port range : “8080”, Source : “custom” and “0.0.0.0/0” and click on “Save rules”.

| Inbound rules Info | | | | | | |
|------------------------------------|---------------------------|----------------------|----------------------|-----------------------------|--|------------------------|
| Security group rule ID | Type Info | Protocol | Port range | Source Info | Description - optional | |
| | | Info | Info | | Info | |
| sgr-08db697053a133ff9 | SSH | TCP | 22 | Cus... ▼ | <input type="text"/> Q X | Delete |
| sgr-06aead81f8cac817d | Custom TCP | TCP | 8080 | Cus... ▼ | <input type="text"/> Q X | Delete |
| sgr-09a267c9daa91c649 | HTTPS | TCP | 443 | Cus... ▼ | <input type="text"/> Q X | Delete |
| sgr-0ffb32172306e5426 | HTTP | TCP | 80 | Cus... ▼ | <input type="text"/> Q X | Delete |

[Add rule](#)

Next, select the EC2 instance and click on "Connect". Navigate to the "EC2 Instance Connect" tab, enter the username "ubuntu", and click on "Connect". This will open a shell in a new tab.

In the shell, execute the following commands to install Docker on the instances:

1. Run `sudo apt-get update`
2. Follow with `sudo apt install docker.io`
3. When prompted, grant permission by typing `Y`.

```
ubuntu@ip-172-31-19-71:~$ sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher
Unable to find image 'rancher/rancher:latest' locally
latest: Pulling from rancher/rancher
1797539a9e5e: Pull complete
21c4959e98e9: Pull complete
4fafb700ef54: Pull complete
4d58a02d3f92: Pull complete
659eb0d911b9: Pull complete
0c1d069e0658: Extracting [==>]
e74dc9fe2e60: Download complete
4ad1f81f76cc: Download complete
b858e3a34133: Download complete
536dadf7525c: Download complete
4b95ffc70a8e: Download complete
ca3f59cc1b4d: Download complete
b3531871e7ee: Download complete
2a4d504935af: Download complete
9e376d229af8: Download complete
bcb0dfa8ac7c: Download complete
db5b2b2eb1f2: Download complete
0ca9ae10bf7: Download complete

i-0b777ba1e05be65a3 (Rancher-Master)
Public IPs: 18.223.52.41 Private IPs: 172.31.18.71
```

Now that we have both the instances up and in “running” state, we will proceed to set up “rancher” on the master node i.e., “Rancher-Master” instance.

- Open the browser and go to the following website: <https://www.rancher.com/quick-start> . Here, scroll down to the “Start the Server” section under “Deploy Rancher”.
- Copy the command present there, which is : “ \$ sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher ”.

With both instances now in a "running" state, let's proceed to set up Rancher on the master node, “Rancher-Master”.

1. Open your web browser and navigate to the following website: Rancher Quick Start Guide.
2. Scroll down to the "Start the Server" section under "Deploy Rancher"
3. Next, establish a connection to the “Rancher-Master” instance following the same steps outlined earlier.
4. This involves:
 - **Connecting:** Select the "Rancher-Master" instance from your EC2 dashboard and click on "Connect".
 - **Instance Connect:** Go to the "EC2 Instance Connect" tab, enter the username "ubuntu", and click "Connect". This will open a shell session in a new browser tab.
 - Once connected, you can proceed with the Rancher setup.

On the "Rancher-Master" instance, execute the following command to get details of the running containers:

```
ubuntu@ip-172-31-6-37:~$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
e641edf3db85 rancher/rancher "entrypoint.sh" 15 seconds ago Up 10 seconds 0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp determined_margulis
ubuntu@ip-172-31-6-37:~$
```

i-0b43f5099b889340a (Rancher-Master)

Public IPs: 18.232.206.204 Private IPs: 172.31.6.37

```
ubuntu@ip-172-31-6-37:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
e641edf3db85 rancher/rancher "entrypoint.sh" 32 minutes ago Up 32 minutes 0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp determined_margulis
ubuntu@ip-172-31-6-37:~$ docker logs e641edf3db85 2>&1 | grep "Bootstrap Password:"
2024/11/23 22:47:48 [INFO] Bootstrap Password: wj22jrvwzpg9npstlmx9zp4jvzb7r5lh5q5bldw64x9t628687hls
ubuntu@ip-172-31-6-37:~$
```

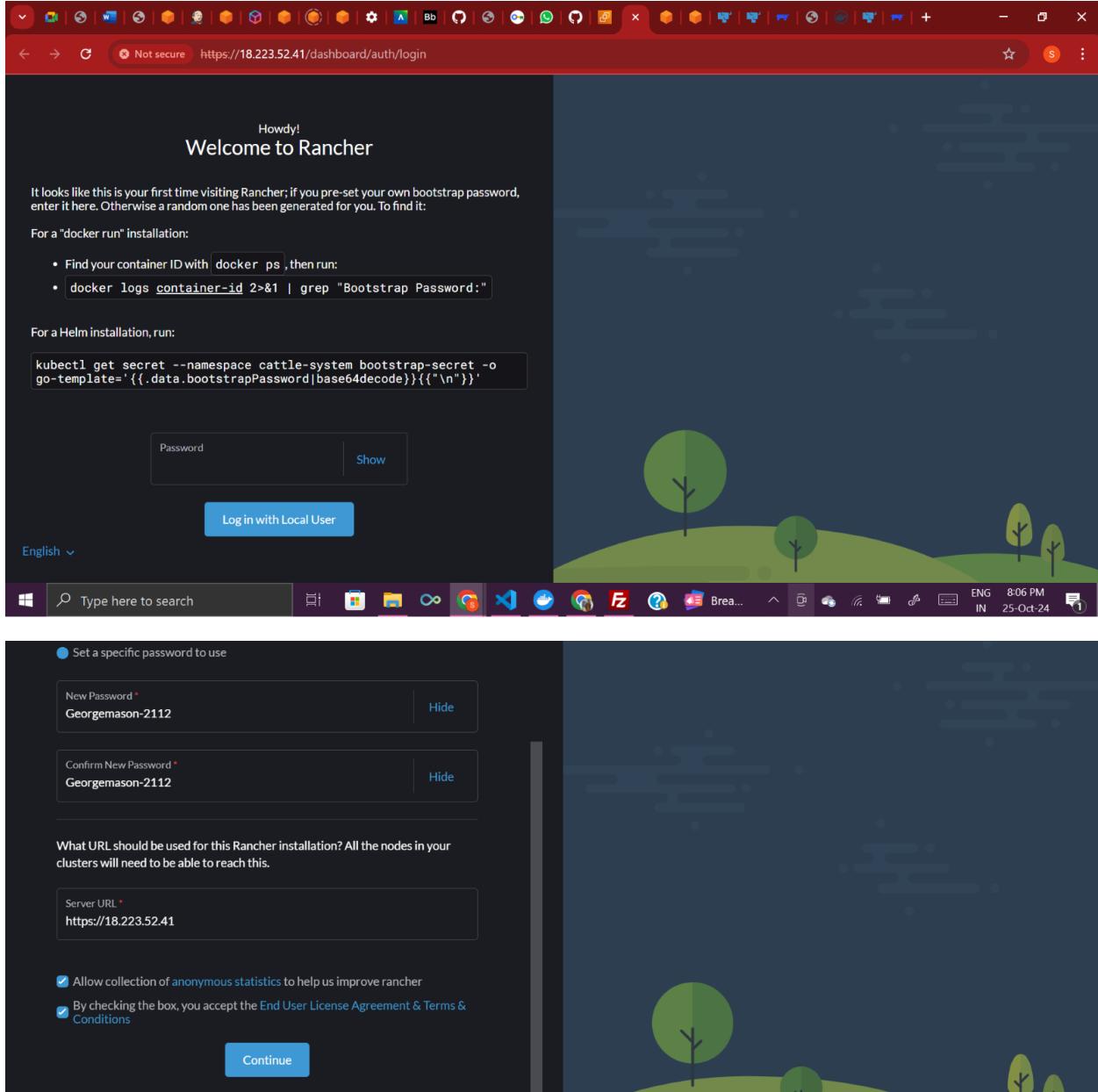
i-0b43f5099b889340a (Rancher-Master)

Public IPs: 18.232.206.204 Private IPs: 172.31.6.37

Creation of instances :

Now, navigate to the EC2 instance page and open the "Public IPv4 DNS" URL in a new browser tab. If prompted with a security warning, proceed to the URL despite it being marked as unsafe.

Once the page loads, you should see the Rancher login page, similar to what we did in the second assignment.



Once the cluster is in the “Active” state, click on the “Explore” button which is next to the cluster.

The screenshot shows the Cattle.io Cluster Management interface. On the left, there's a sidebar with icons for Home, Cloud Credentials, Drivers, RKE1 Configuration, Advanced, Workloads (highlighted), Services, Events, and Help. The main area is titled 'Clusters' and lists two clusters: 'local' (Active, v1.31.1+k3s1, Local K3s, 1 machine, 44 mins old) and 'swe645hw3' (Updating, v1.31.2+rke2r1, Imported RKE2, 0 machines, 3.1 mins old). Buttons for 'Import Existing' and 'Create' are at the top right, along with a 'Filter' input field.

| State | Name | Version | Provider | Machines | Age |
|----------|-----------|-----------------------|------------------|----------|----------|
| Active | local | v1.31.1+k3s1 amd64 | Local K3s | 1 | 44 mins |
| Updating | swe645hw3 | v1.31.2+rke2r1 | Imported RKE2 | 0 | 3.1 mins |

Configuring bootstrap node(s) custom-71a809fe665d: waiting for cluster agent to connect

1. On the left pane, go to **Workloads** and select **Deployments**.
2. Click on the **Create** button to open the form.
3. Fill out the form with the following details:
 - o **Namespace:** default
 - o **Name:** name to it
 - o **Replicas:** 3 (this is the number of pods)
 - o **Container Image:** satluri2/survey:latest (make sure this matches the name of the image on Docker Hub)

Once done, proceed with the setup to deploy your application.

The screenshot shows the 'Clusters' section of the Cattle Manager. On the left, a sidebar lists 'Clusters', 'Cloud Credentials', 'Drivers', 'RKE1 Configuration', and 'Advanced'. The main area displays a table of clusters with columns: State, Name, Version, Provider, Machines, and Age. Two clusters are listed: 'local' (v1.31.1+k3s1, Local K3s, 1 machine, 47 mins old) and 'swe645hw3' (v1.31.2+rke2r1, Custom RKE2, 1 machine, 5 mins old). Buttons for 'Import Existing' and 'Create' are at the top right.

The screenshot shows the 'Deployment' creation interface for cluster 'swe645hw3'. The left sidebar shows 'Workloads' (selected), 'CronJobs', 'DaemonSets', 'Deployments' (selected), 'Jobs', 'StatefulSets', and 'Pods'. The main area shows a form for creating a deployment named 'surveyassign3-deploy' with 3 replicas. The 'General' tab is selected, showing fields for Container Name ('container-0'), Image ('Container Image: satluri2/survey:latest', 'Pull Policy: Always'), and other deployment settings. A 'Create' button is at the bottom right.

Under the **Networking** section, click on the **Add port or service** button.

Fill in the following details:

- **Service type:** Node Port
- **Name:** nodeport
- **Private container port:** 8080
- **Protocol:** TCP

Leave all other settings as default and click on **Create**.

Now that the deployment is complete, it will initially be in the “Updating” state. Allow it a few minutes to transition, and soon it should be in the “Active” state.

Once the deployment is active, your application should be up and running, accessible via the configured Node Port.

The screenshot shows the Cattle UI interface. On the left, there's a sidebar with various icons and a cluster name 'swe645hw3'. The main area is titled 'Deployments' and shows a table with one row. The row details are:

| State | Name | Image | Ready | Up To Date | Available | Restarts | Age | Health |
|--------|----------------------|-----------------|-------|------------|-----------|----------|---------|---|
| Active | surveyassign3-deploy | saturni2/survey | 3/3 | 3 | 3 | 0 | 10 mins | <div style="width: 100%; height: 10px; background-color: green;"></div> |

Once everything is adjusted properly you can see the 3 instances deployed properly.

The screenshot shows the Cattle UI interface. On the left, there's a sidebar with various icons and a cluster name 'swe645hw3'. The main area is titled 'Services' and shows a table with two rows. The rows details are:

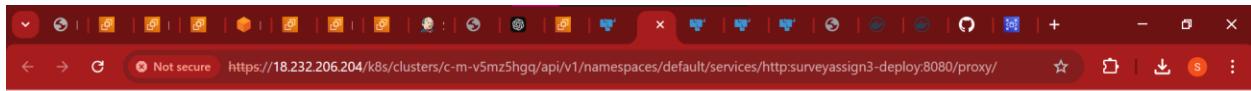
| State | Name | Namespace | Target | Selector | Type | Age |
|--------|-------------------------------|-----------|-----------|---|------------|---------|
| Active | surveyassign3-deploy | default | nodeport | workload.user.cattle.io/workloadselector=apps.deployment-default-surveyassign3-deploy | Cluster IP | 12 mins |
| Active | surveyassign3-deploy-nodeport | default | 31389/TCP | workload.user.cattle.io/workloadselector=apps.deployment-default-surveyassign3-deploy | Node Port | 12 mins |

Testing the link deployed on nodeport in kubernetes :

Accessing the Service: The image shows a web browser accessing a service running on a Kubernetes cluster via the URL: <https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/>. This URL structure indicates that the user is leveraging a Kubernetes proxy to reach the service.

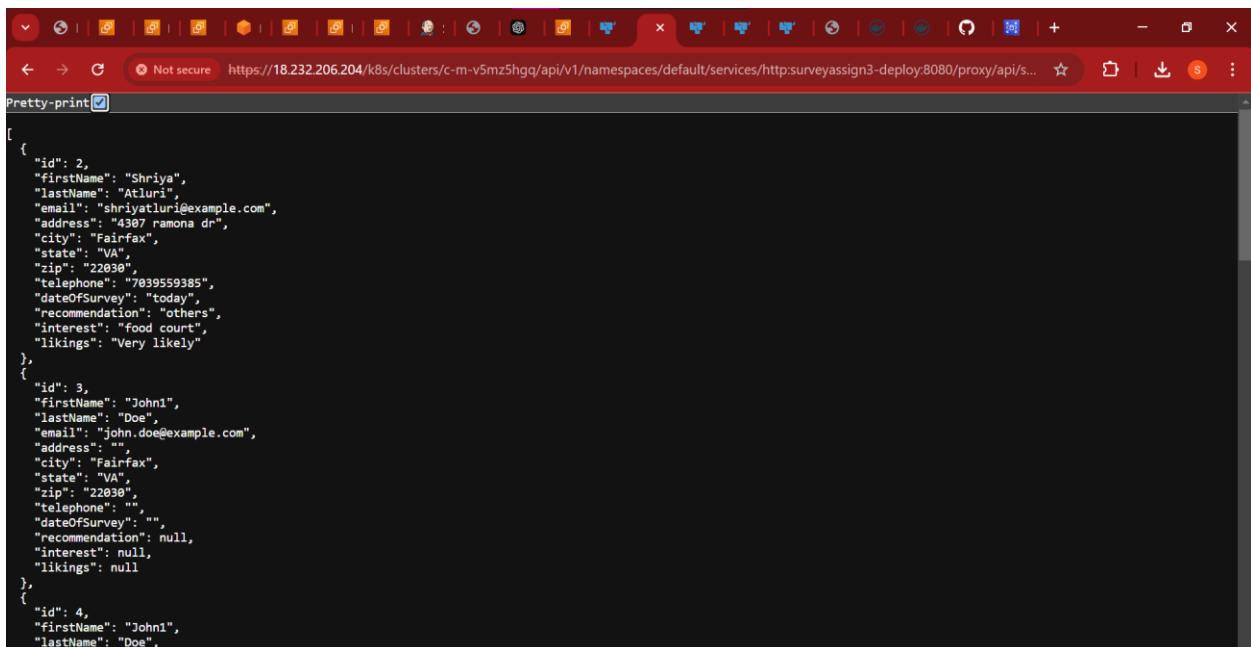
Service Name and Port: The URL includes http:surveyassign3-deploy:8080, indicating that the service name is surveyassign3-deploy, and it is being accessed on port 8080.

Application Content: The webpage displays a message: "Welcome to the Student Management Application! Please use this database to view the student information." This implies that the application is a student management system.



Welcome to the Student Management Application! Please use this database to view the student information.

URL : <https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/>



The URL for viewing all the surveys is: <https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/api/surveys>

Now, setting the jenkins instance :

Setting up an AWS EC2 instance for Jenkins installation and operation:

- We'll focus on developing a CI/CD pipeline using Jenkins to automatically build and update our source code as needed.
- To start, we need to install Jenkins on our machine. We'll begin by setting up an AWS EC2 instance in our AWS lab.
- We will create this instance in the same manner as we did with the previous two EC2 instances, to deploy our application on the Kubernetes cluster.
- Name the EC2 instance "swe645hw2_jenkins," follow the same setup steps, assign an elastic IP, and adjust the security group accordingly.

Use **EC2 Instance Connect** to establish a connection to the instance and verify that it is in the "running" state.

Before using Jenkins, install Java by running the following commands:

- sudo apt update
- sudo apt install openjdk-17-jdk -y (This installs Java JDK 17)

Install Jenkins by running these commands:

- sudo wget -O /usr/share/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io2023.key
- echo "deb [signedby=/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-get update
- sudo apt-get install jenkins -y (This installs Jenkins)

Start Jenkins using the command:

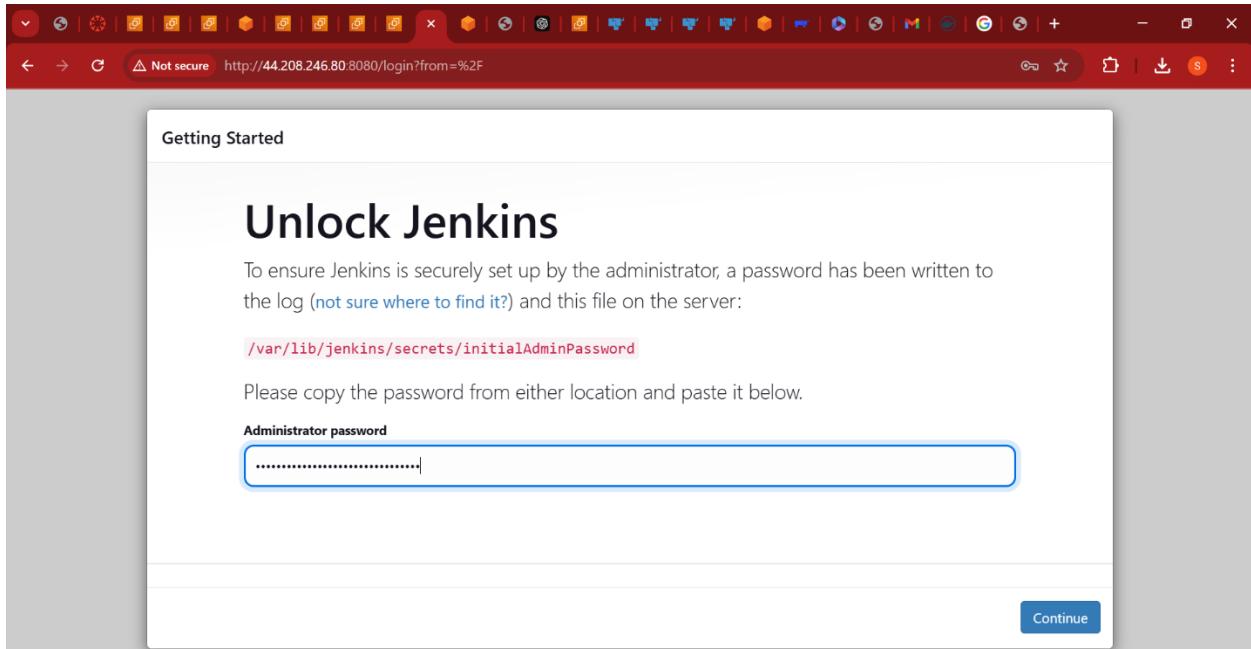
- sudo systemctl start jenkins.service

```
ubuntu@ip-172-31-4-183:~$ sudo ufw allow 8080
Rules updated
Rules updated (v6)
ubuntu@ip-172-31-4-183:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
033d39d37bb844149664012a660a7e5d
ubuntu@ip-172-31-4-183:~$
```

i-0a67f2fea7cd95f78 (Jenkins)

Public IPs: 44.208.246.80 Private IPs: 172.31.4.183

Give the default admin command to enter to jenkins



Now download the suggested plugins:

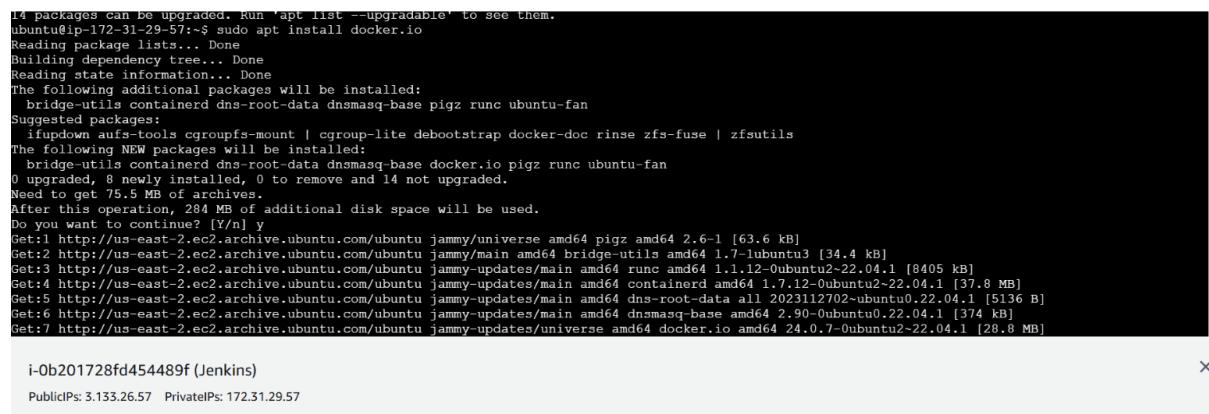


- Return to the EC2 console to create a config file.
- Run sudo su jenkins to switch to the Jenkins home.
- Navigate to the root directory: cd ../../
- Go to the Jenkins directory: cd /var/lib/jenkins.
- Create a directory and enter it: mkdir .kube followed by cd .kube.

- Create and open the config file: vi config.
- Copy and paste the contents from the previously downloaded “KubeConfig” file into this config file.
- Save the file by typing :wq.

Next, exit the jenkins mode using :“exit” command.

Now proceed to install docker using the commands : “sudo apt-get update” and “sudo apt update” followed by “sudo apt install docker.io”. Give permission as “Y” when asked.



```

14 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-29-57:~$ sudo apt install docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 14 not upgraded.
Need to get 75.5 MB of archives.
After this operation, 284 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-2.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6-1 [63.6 kB]
Get:2 http://us-east-2.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 bridge-utils amd64 1.7-lubuntu3 [34.4 kB]
Get:3 http://us-east-2.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 runc amd64 1.1.12-0ubuntu2-22.04.1 [8405 kB]
Get:4 http://us-east-2.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 containerd amd64 1.7.12-0ubuntu2-22.04.1 [37.8 kB]
Get:5 http://us-east-2.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 dns-root-data all 2023112702-ubuntu0.22.04.1 [5136 B]
Get:6 http://us-east-2.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 dnsmasq-base amd64 2.90-0ubuntu0.22.04.1 [374 kB]
Get:7 http://us-east-2.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 docker.io amd64 24.0.7-0ubuntu2-22.04.1 [28.8 MB]

i-0b201728fd454489f (Jenkins)
Public IPs: 3.133.26.57 Private IPs: 172.31.29.57

```

Creating github for CI/CD pipeline:

To create our pipeline, we need the source code stored in a location where it can be attached and monitored for changes. We will use GitHub for this purpose.

1. Visit GitHub and create an account if you don't already have one. Log in to your account.
2. Create a new repository named SWE645HW3.git
3. Add your source files, Dockerfile, and the WAR file to the repository, then commit these changes.
4. We will use this repository to set up our pipeline.
5. The URL for the GitHub repository is: <https://github.com/shriya2112/SWE645HW3.git>

The screenshot shows a GitHub repository page for 'shriya2112/SWE645HW3'. The 'Code' tab is selected. A commit from 'shriya2112' is highlighted, showing changes to SurveyController.java and other files like .mvn/wrapper, Dockerfile, Jenkinsfile, README.md, mvnw, mvnw.cmd, and pom.xml. The commit message is 'Update SurveyController.java' and it was made 'yesterday'.

| File | Description |
|--------------|------------------------------|
| .mvn/wrapper | added API code |
| src | Update SurveyController.java |
| .gitignore | added API code |
| Dockerfile | hw3 |
| Jenkinsfile | Update Jenkinsfile |
| README.md | hw3 |
| mvnw | added API code |
| mvnw.cmd | added API code |
| pom.xml | hw3 |

Now, the creation of CI/CD pipeline:

Navigate to Jenkins Dashboard: Go to the Jenkins dashboard.

Manage Jenkins: Click on “Manage Jenkins”.

Credentials: Scroll down and select the “Credentials” option.

Global Credentials: Choose the “global” option and then click on “Add credentials”.

Add Credentials: Set the following:

- **Kind:** Select “Username and password”.
- **Username:** Enter your Docker Hub username.
- **Password:** Enter your Docker Hub password.
- **ID:** Set the ID to “dockerhub”.

Create: Click on “Create” to save these credentials.

The screenshot shows the Jenkins 'Manage Jenkins' interface under 'Global credentials (unrestricted)'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc.)'. The 'Username' field contains 'satluri2'. The 'Password' field is redacted with dots. The 'ID' field contains 'dockerhub'. The 'Description' field is empty. A blue 'Create' button is visible at the bottom.

Repeat the above steps to save GitHub credentials as well, give the id as “github”. For github, we will have to use PAT (Personal Access Tokens) as the password. • Now go to the dashboard and click on “New Item”. • Give the name as “assignment3swe645” and click on “Pipeline” option.

Repeat the above steps to save GitHub credentials, this time giving the ID as “github”. For GitHub, use a Personal Access Token (PAT) as the password.

1. Save GitHub Credentials:

- Navigate to the “Credentials” section under “Manage Jenkins”.
- Select the “global” option and click on “Add credentials”.
- Set the **Kind** to “Username and password”.
- Enter your GitHub username.
- Use your GitHub PAT (Personal Access Token) as the password.
- Set the ID to “github”.
- Click on “Create”.

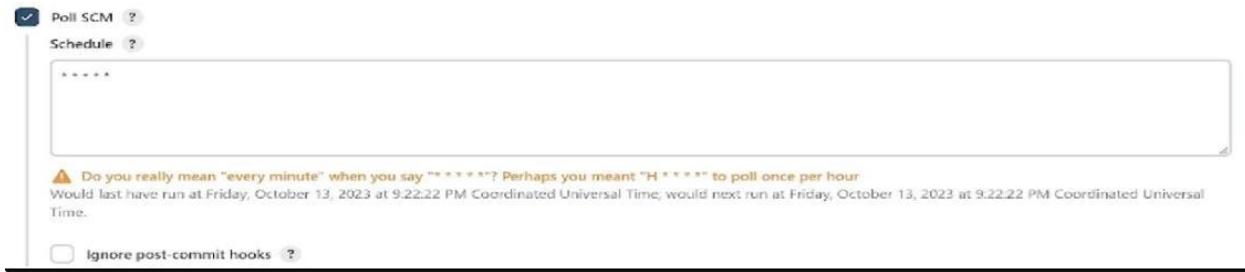
2. Create a New Pipeline:

- Go to the Jenkins dashboard and click on “New Item”.

Give name as : assignment3swe645 and select pipeline.

Now, on the next page, check the “Github Project” checkbox, and in the project URL, paste the URL from

- Github link : <https://github.com/shriya2112/SWE645HW3.git>
- Scroll down and check the “Poll SCM” checkbox.
Give the Schedule as : “* * * * *”



Scroll down to the “Pipeline” section.

Select the definition: “Pipeline from SCM” and SCM: “git”.

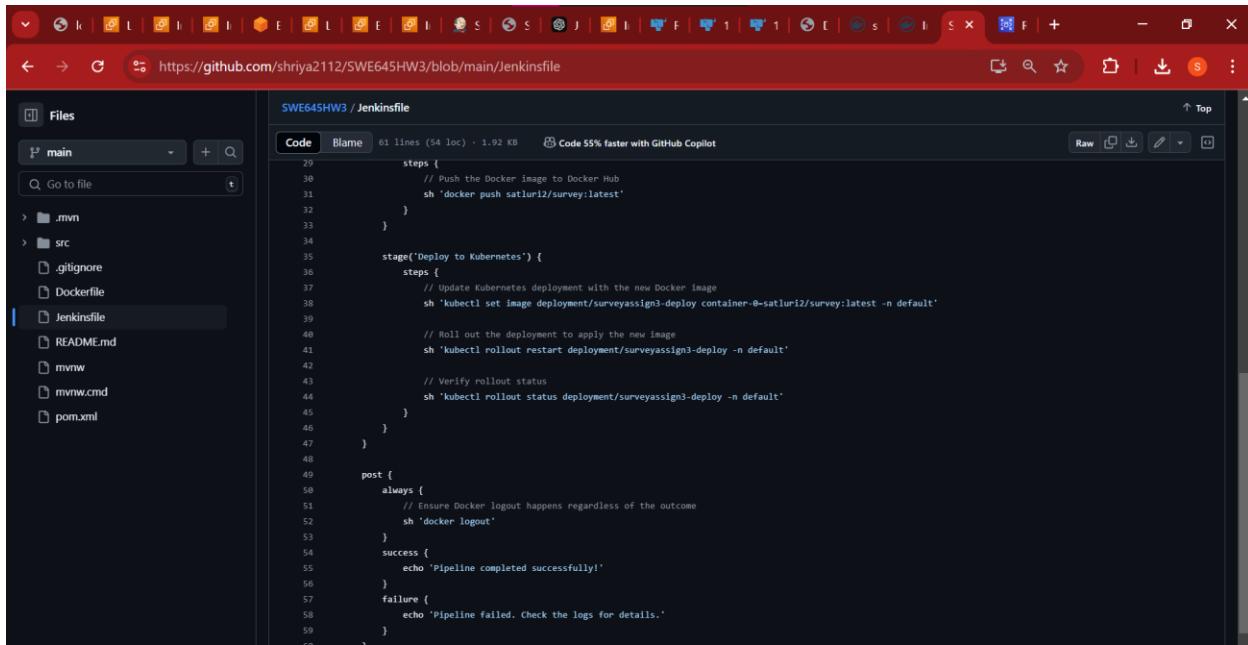
- Give the Github repository : <https://github.com/shriya2112/SWE645HW3.git>

Under Credentials, select the github credentials.

Then, change the branch specifier : “*/main”.

1. Navigate to your GitHub repository and create a new file named Jenkinsfile. Commit the changes.
2. Click on **Save**.
3. This will trigger an automatic build on your Jenkins pipeline.
4. Go back to the GitHub repository, locate the Jenkinsfile, and click on the edit option.
5. Enter the commands shown in the provided screenshot. Commit the changes.

These commands will outline the steps to be executed whenever there is a commit or change to the files in the repository. They will include commands to generate a new WAR file, create a Docker image, log in to Docker, push the image, and deploy the image on a Kubernetes cluster.



```
steps {
    // Push the Docker image to Docker Hub
    sh 'docker push satluri2/survey:latest'
}

stage('Deploy to Kubernetes') {
    steps {
        // Update Kubernetes deployment with the new Docker image
        sh 'kubectl set image deployment/surveyaassign3-deploy container-0=satluri2/survey:latest -n default'

        // Roll out the deployment to apply the new image
        sh 'kubectl rollout restart deployment/surveyaassign3-deploy -n default'

        // Verify rollout status
        sh 'kubectl rollout status deployment/surveyaassign3-deploy -n default'
    }
}

post {
    always {
        // Ensure Docker logout happens regardless of the outcome
        sh 'docker logout'
    }
    success {
        echo 'Pipeline completed successfully!'
    }
    failure {
        echo 'Pipeline failed. Check the logs for details.'
    }
}
```

Creation of jenkins file in github:

You will get multiple errors while creating, keep on correcting those errors and updates and committing the jenkins file.

As you proceed with your builds, you may encounter some errors in the paths specified within the Jenkinsfile. These errors can occur due to various reasons, such as incorrect file paths or commands.

Here's what you should do:

1. Review the Error Logs: When a build fails, Jenkins provides detailed logs. Review these logs to understand the nature of the errors.
2. Edit the Jenkinsfile: Based on the errors identified, edit the Jenkinsfile to correct the paths or commands.
3. Test Locally: If possible, try to replicate the commands locally on your machine to ensure they work as expected before updating the Jenkinsfile.
4. Commit Changes: After making necessary adjustments, commit the changes to the Jenkinsfile in your GitHub repository.
5. Trigger a New Build: Jenkins will automatically detect the changes in the Jenkinsfile and trigger a new build. Monitor this build to ensure the errors are resolved.

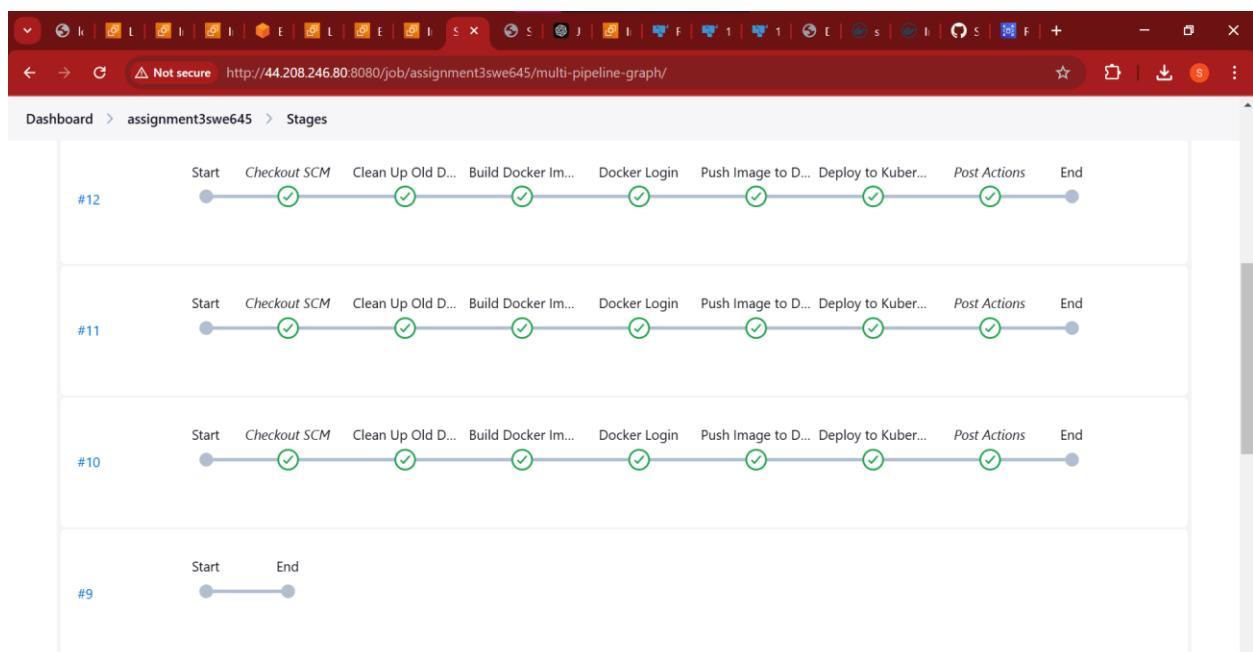
Not secure http://44.208.246.80:8080/job/assignment3swe645

Builds

Filter

Today

- #12 3:43 AM
- #11 3:43 AM
- #10 3:41 AM
- #9 3:38 AM
- #8 3:38 AM
- #7 3:28 AM
- #6 3:27 AM
- #5 3:25 AM
- #4 3:24 AM
- #3 3:22 AM
- #2 3:22 AM
- #1 3:13 AM



The screenshot shows the Postman application interface. In the top navigation bar, 'Home' and 'Workspaces' are visible. The main workspace shows a GET request to the URL <https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/api/surveys>. The 'Body' tab is selected, showing a JSON response with fields like 'firstName', 'lastName', 'email', 'address', 'city', 'state', and 'age'. Below the body, the status bar indicates a 200 OK response with 347 ms latency and 412 B size. The bottom section shows the raw HTML response content.

1. API Request: You are using Postman to test an API endpoint.
2. GET Request: The request being made is a GET request to the URL: [`https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/api/surveys`](https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/api/surveys).
3. Successful Response: The response status is `200 OK`, indicating the request was successful.
4. Response Data: The response body contains JSON data with the following details:

```
```json
{
 "firstName": "Sampreeth",
 "lastName": "Joshi",
 "email": "sampreethjoshi@example.com",
 "address": "10111 Gainsborough",
 "city": "Fairfax",
 "state": "VA",
 "age": 30
}
```

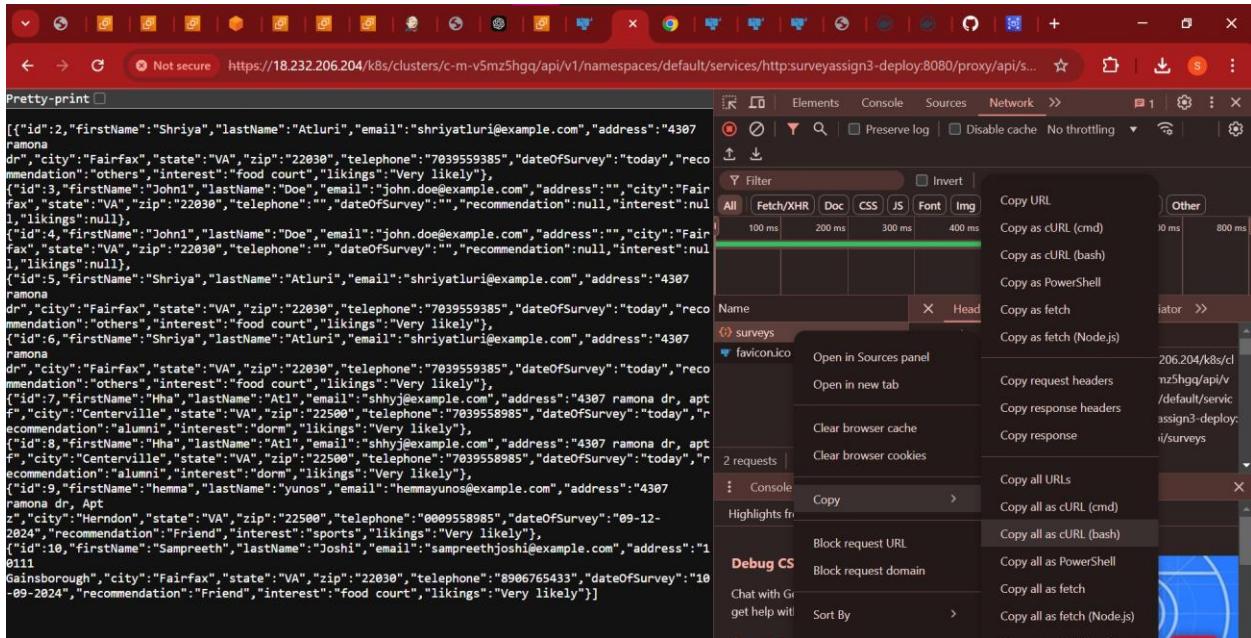
```

5. HTML Preview: Additionally, there is a message displayed in the HTML preview section:

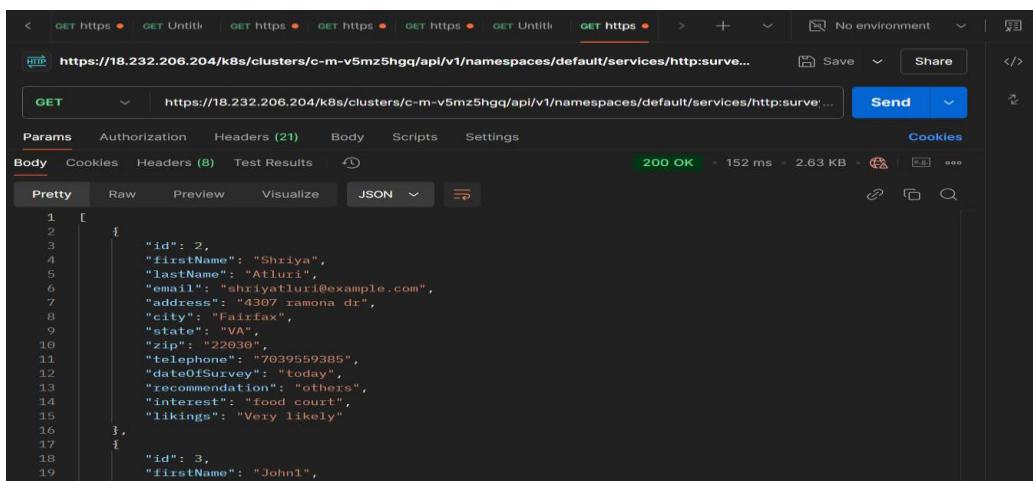
```html

Welcome to the Student Management Application! Please use this database to view the student information.

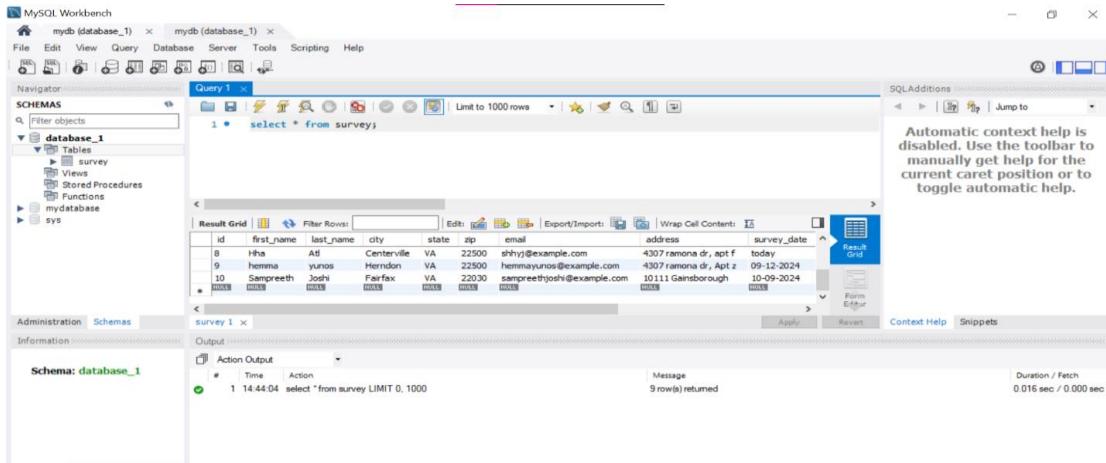
```



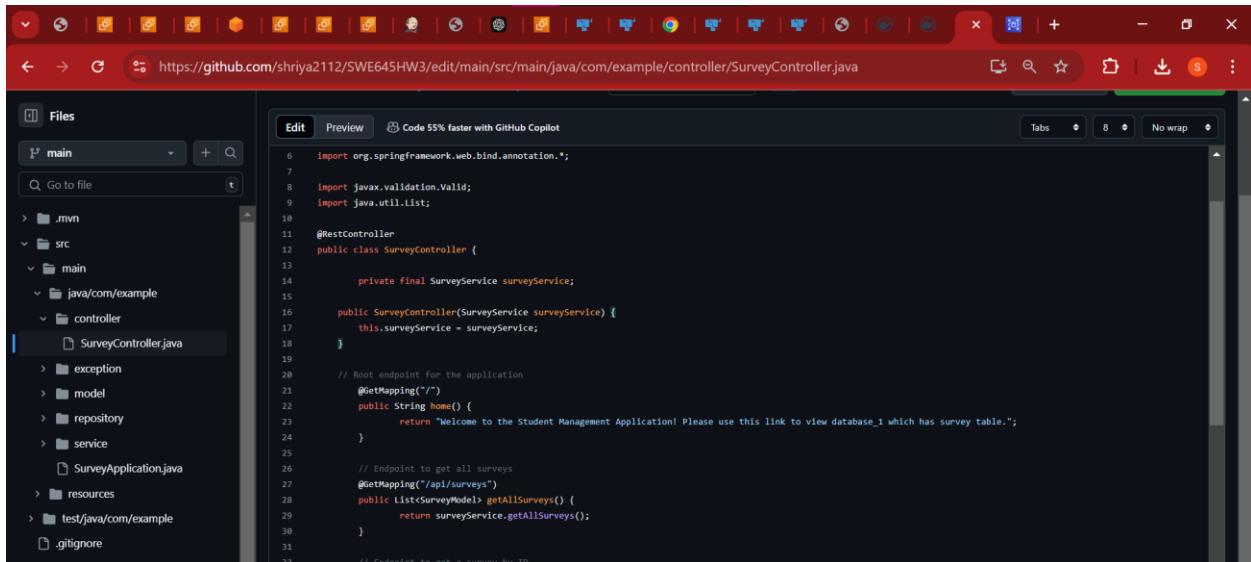
Here to get the link in the postman goto copy >>copy all as URL and then past the link in the post man to get the required output



The above picture shows the successful running of API



Now, lets check the same in MYSQL database.



A small change is performed on github, to check if it reflects in the pipeline. And then checked with postman. Above, you can see that the changes are perfectly getting reflected.

The pipeline is getting built.

Here, the image depicts when the changes are pushed onto github and the build process takes place successfully.

Started by user Shriya Aturi
Started 45 sec ago
Took 38 sec

Revision: 0189fa19e1c0ad02c305fd4b3ff635f27f04ef7c
Repository: <https://github.com/shriya2112/SWE645HW3.git>

Changes

1. Update SurveyController.java (commit: 0189fa1) ([details](#) / [githubweb](#))

The screenshot shows the Postman interface with a successful HTTP GET request. The URL is <https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/api/surveys>. The response status is 200 OK, and the body contains a welcome message: "Welcome to the Student Management Application! Please use this link to view database_1 which has survey table."

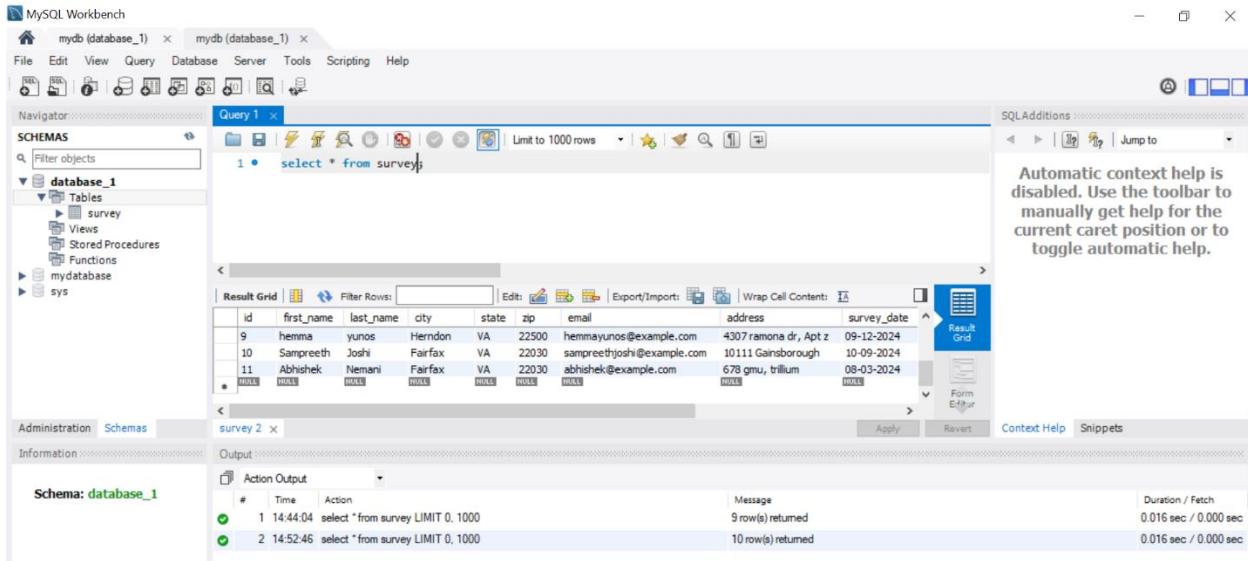
The change is successfully reflected.

The screenshot shows the Postman interface with a successful HTTP POST request. The URL is the same as the previous screenshot. The request body is a JSON object representing a survey record:

```
4 "email": "abhishek@example.com",
5 "address": "678 gmu, trillium",
6 "city": "Fairfax",
7 "state": "VA",
8 "zip": "22030",
9 "telephone": "7654897890",
10 "dateOfSurvey": "08-03-2024".
```

The response status is 200 OK, and the body shows the newly created survey record.

URL : <https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/api/surveys>



When changes are made to the data in GitHub, and these changes are reflected not only in Postman but also in the database, here's a detailed explanation of how this process typically works:

Workflow Overview

- Update Code in GitHub:** Changes are made to the source code, configuration files, or database schema in the GitHub repository.
- Jenkins Pipeline Trigger:** The Jenkins CI/CD pipeline detects these changes and initiates a build process.
- Build and Deploy:** Jenkins builds the updated code, creates a new Docker image, and deploys it to the Kubernetes cluster.
- Database Migration/Update:** As part of the deployment process, database migrations or updates are applied to reflect the new changes.
- Verify with Postman:** Use Postman to verify that the changes are successfully reflected in the application's API responses.
- Database Verification:** Confirm that the updates are correctly applied to the database.

```

11 package com.example.controller;
12
13 public class SurveyController {
14
15     private final SurveyService surveyService;
16
17     public SurveyController(SurveyService surveyService) {
18         this.surveyService = surveyService;
19     }
20
21     // Root endpoint for the application
22     @GetMapping("/student")
23     public String home() {
24         return "Welcome to the Student Management Application! Please use this link to view database_1 which has survey table.";
25     }
26
27     // Endpoint to get all surveys
28     @GetMapping("/surveys")
29     public List<Survey> getAllSurveys();

```

Above picture stating that we can fetch the changes even after changing the path. Here we are changing the path from “ / “ to “/student”

The output in post when the path is changed is depicted in the above figure

<https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surve...>

Here's a concise description of the process:

- Postman Setup:** Open Postman and check the API endpoint mappings to determine the correct URL to test, such as /api/surveys or /student.
- Send Request:** Based on the mappings, use Postman to send a request to the specified endpoint.
- Backend Verification:** Verify that the backend is correctly implemented by checking the response data returned by the API.

This summarizes the backend implementation and testing process for the assignment.

Reference tools used in this assignment:

1. **Spring Initializr**: For creating a Maven project with necessary dependencies.
2. **Spring Boot**: To develop and run the backend services.
3. **Spring Data JPA**: For managing CRUD operations and database interactions.
4. **MySQL Driver**: To connect and interact with the MySQL database.
5. **Amazon RDS**: For hosting the MySQL database in a scalable environment.
6. **Docker**: For containerizing the application.
7. **Docker Hub**: For sharing Docker images.
8. **AWS EC2**: To deploy Kubernetes clusters and other components.
9. **Rancher**: For managing Kubernetes deployments.
10. **Postman**: For API testing.
11. **Jenkins**: For creating and managing CI/CD pipelines.
12. **GitHub**: For version control and hosting the project repository.

These tools collectively ensure the successful implementation of the assignment requirements.

CONTRIBUTIONS

1. **SHRIYA ATLURI (G01476041)**: Contributed in the Spring Initializr, Spring Boot, Spring Data JPA, MySQL Driver setup. Helped in creating the database and setting up the AWS RDS installation. Docker and Dockerhub is created on this account. Helped in instance creation, Rancher setup and testing the application on postman. Documentation and video recording were also done.
2. **SAI SAMPREETH JOSHI (G01408085)**: Contributed in the Spring Initializr, Spring Boot, Spring Data JPA, MySQL Driver setup. Helped in creating the database and setting up the AWS RDS installation. Helped in instance creation, Rancher setup and testing the application on postman. Helped in testing the pipelines also.
3. **PRAGATHI REDDY GUDURI (G01478980)**: Helped in creating the database and setting up the AWS RDS installation. Helped in instance creation, Rancher setup and testing the application on postman. Helped in setting up jenkins and testing the pipelines also. Video recording is also done.
4. **SAI ABHISHEK NEMANI (G01462099)**: Helped in creating the database and setting up the AWS RDS installation. Helped in instance creation, Rancher setup and testing the application on postman. Helped in setting up jenkins and testing the pipelines also. Documentation is also done.