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.

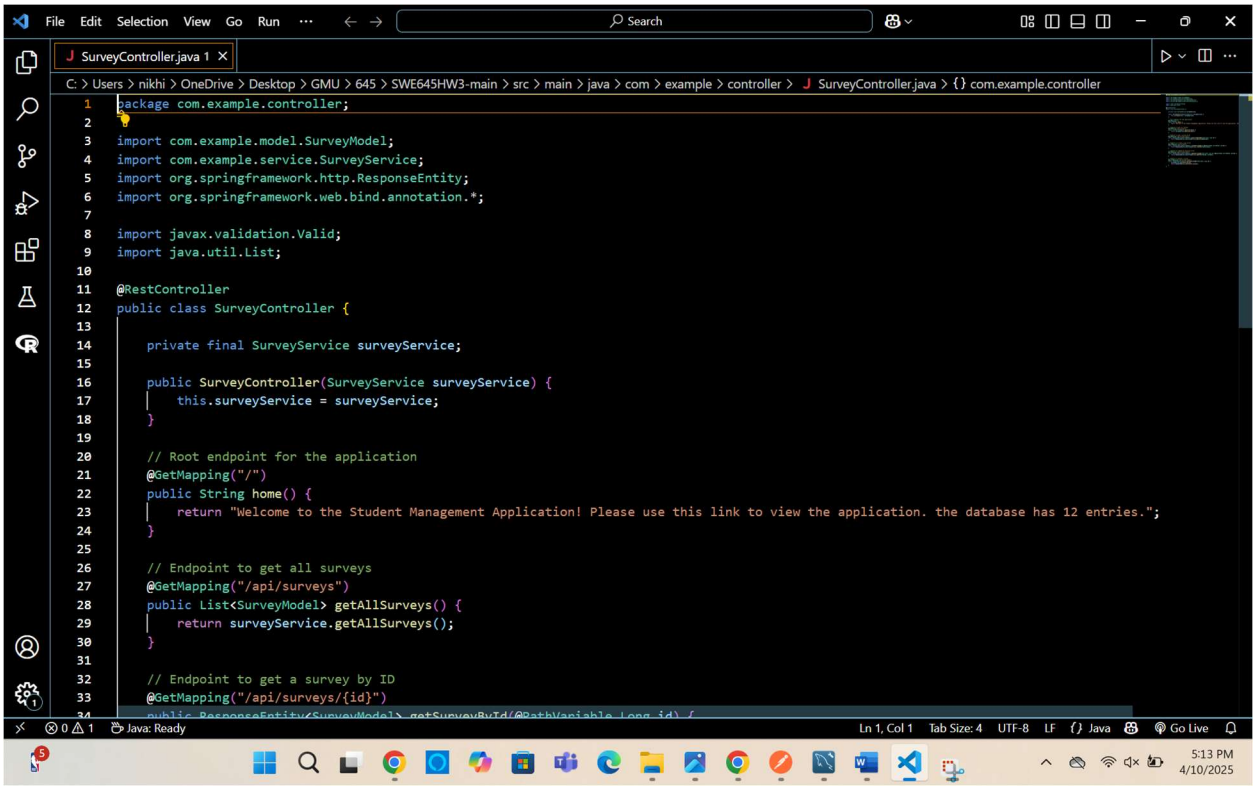
4. **Application Deployment via Rancher UI**: Leverage the intuitive Rancher user interface to deploy the application seamlessly. Set up and manage a Kubernetes cluster through the UI, enabling automated scaling and resilience for the deployed application.

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:



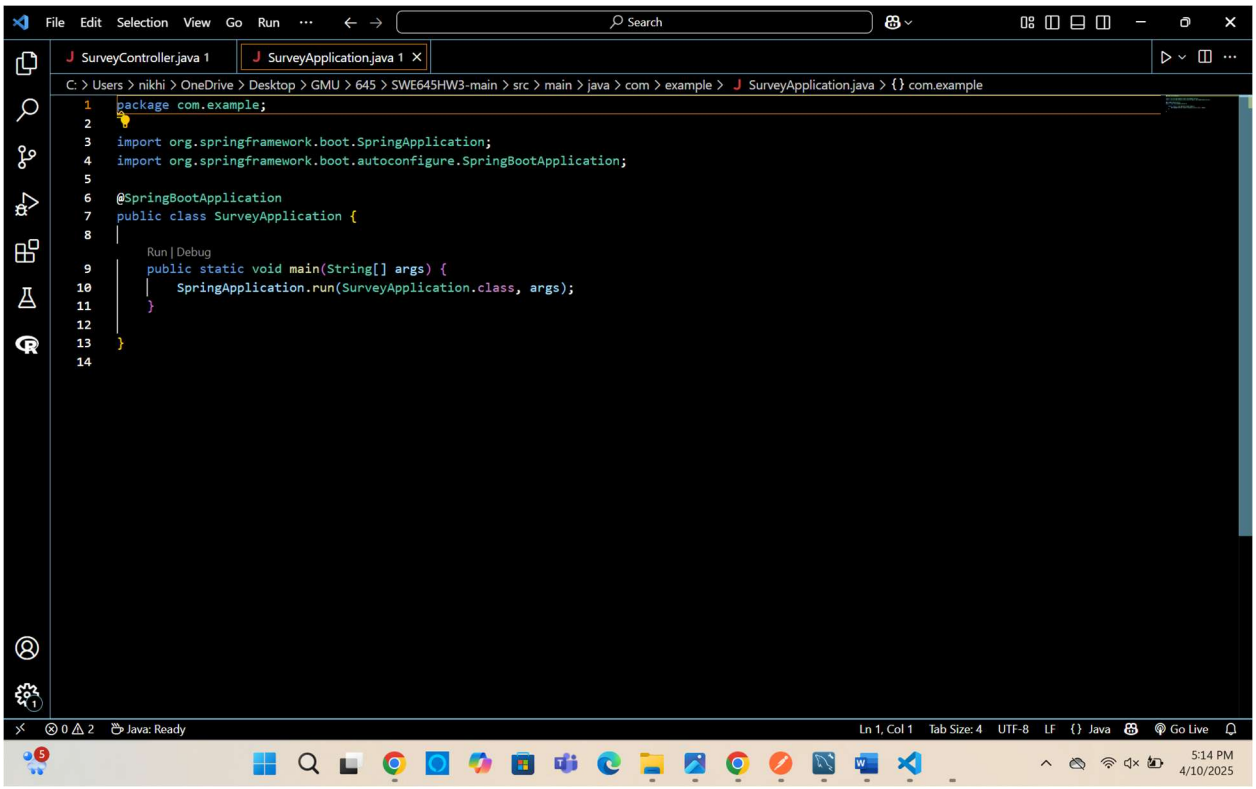
1. **FileStructure:**
   The project follows a clear and organized structure, with folders like com.example.controller, com.example.service, and com.example.model, ensuring modularity and readability.

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.



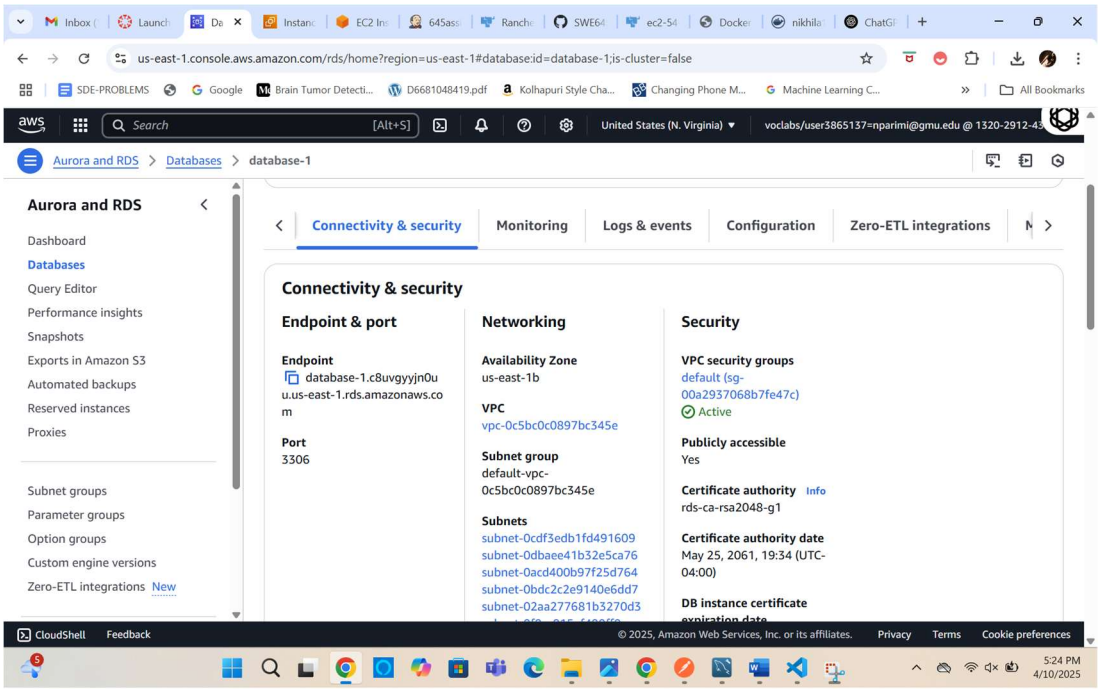
1. **MainClass:**
   The SurveyApplication class serves as the main entry point for the Spring Boot application, marked with the @SpringBootApplication annotation.

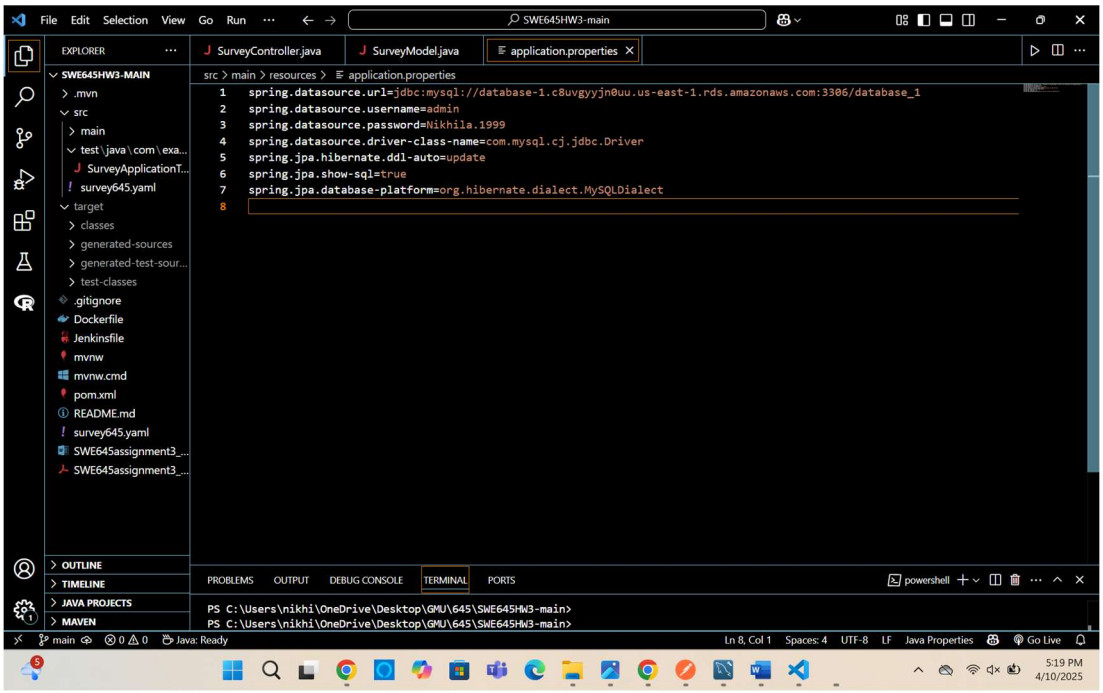
2. **SpringBootFeatures:**
   The @SpringBootApplication annotation enables essential Spring Boot features, including auto-configuration, component scanning, and configuration setup.

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

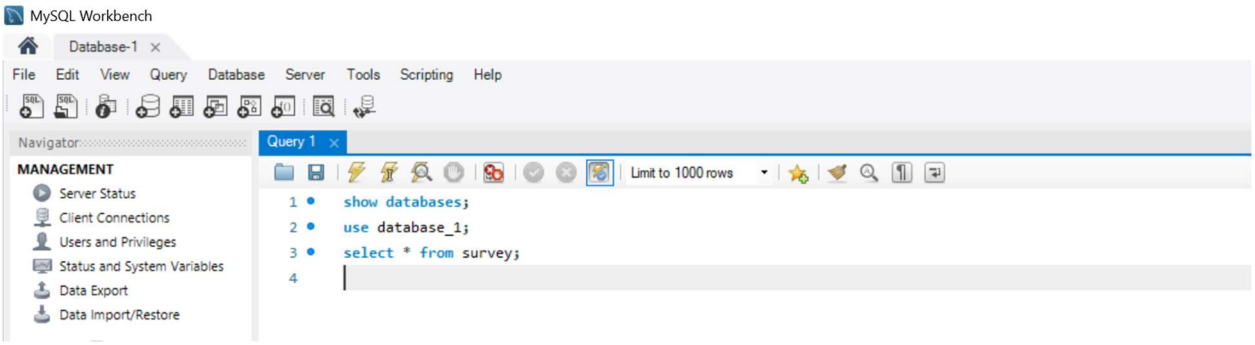
1. **AmazonRDSDashboardOverview:**
   The RDS management console displays details of a MySQL database instance, with a focus on the *Connectivity & security* tab.

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. **SSLEncryption:**
   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**:
   a. Go to AWS Console.
2. **Navigate to RDS**:
   a. Search for **RDS** in the AWS Management Console and open it.
3. **Create a New Database**:
   a. Click on **Create database**.
   b. Select **MySQL** as the database engine.

    c. Choose **Standard create**.
    d. Specify settings like:
        i. **DB Instance Identifier**: A name for your RDS instance (e.g., mydb).
        ii. **Master Username**: Set a username (e.g., admin).
        iii. **Master Password**: Set a strong password.
4. **Configure Connectivity**:
    a. Under **Connectivity**, ensure that:
        i. A **VPC** is selected.
        ii. **Public Access** is set to **Yes** (if you want to connect from your local machine).
        iii. 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:
        sudo apt update
        sudo apt install mysql-client

    b. For macOS:
        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:

    mysql -h <endpoint> -P <port> -u <username> -p

    Replace:

        a. <endpoint>: Your RDS endpoint (e.g., database-1.c8uvgyyjn0uu.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).

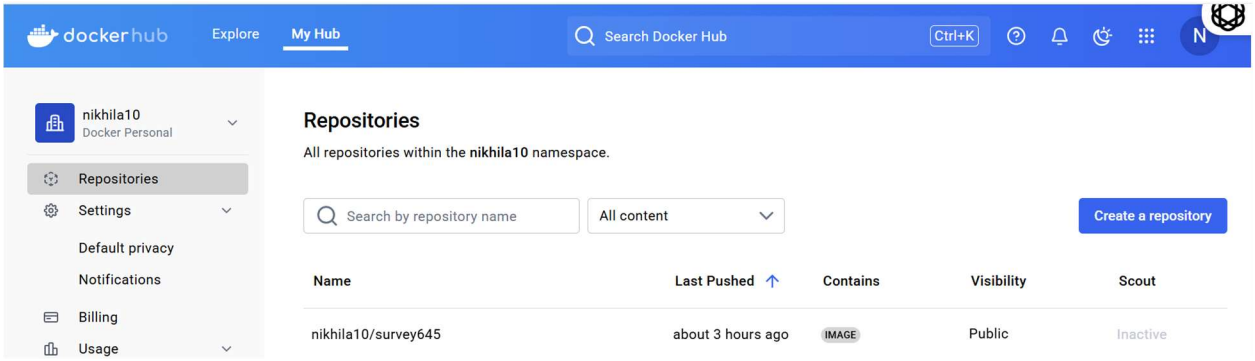
    Example:

    database-1.c8uvgyyjn0uu.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:
        mysql>

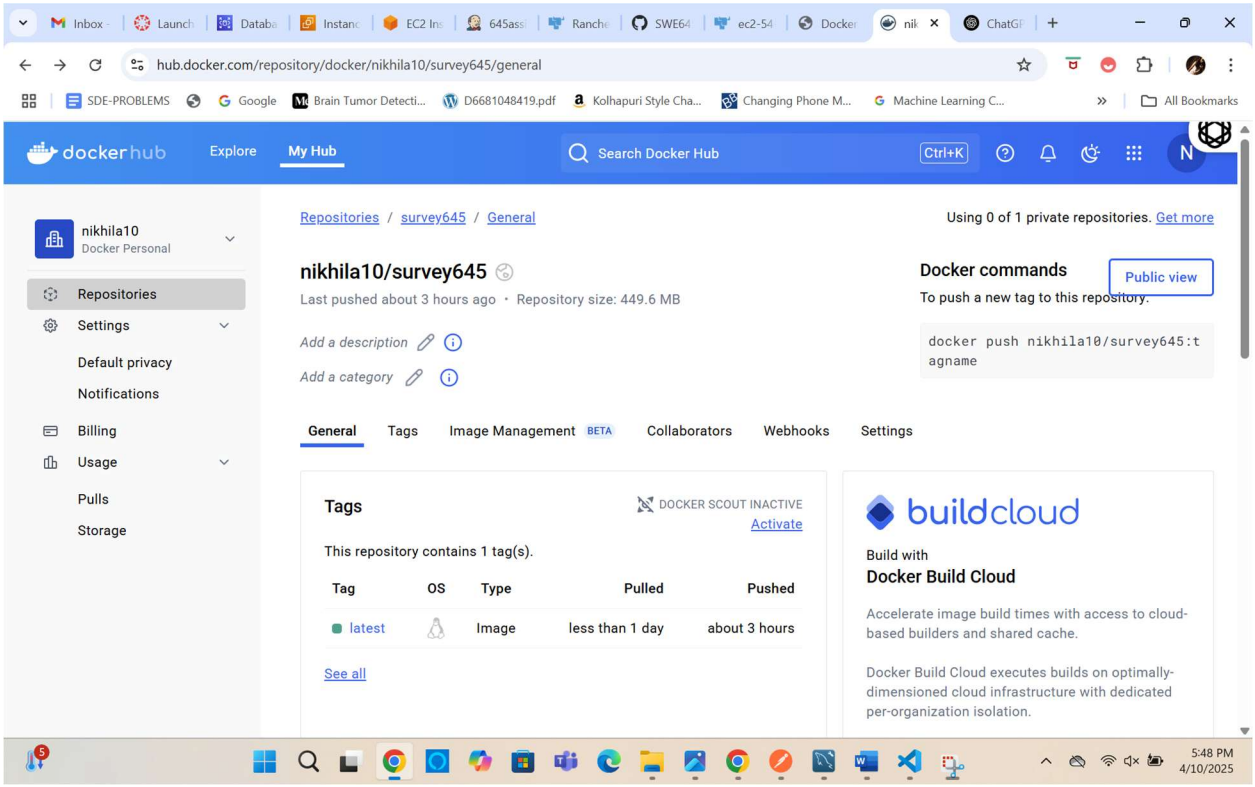
5. Test the Connection

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



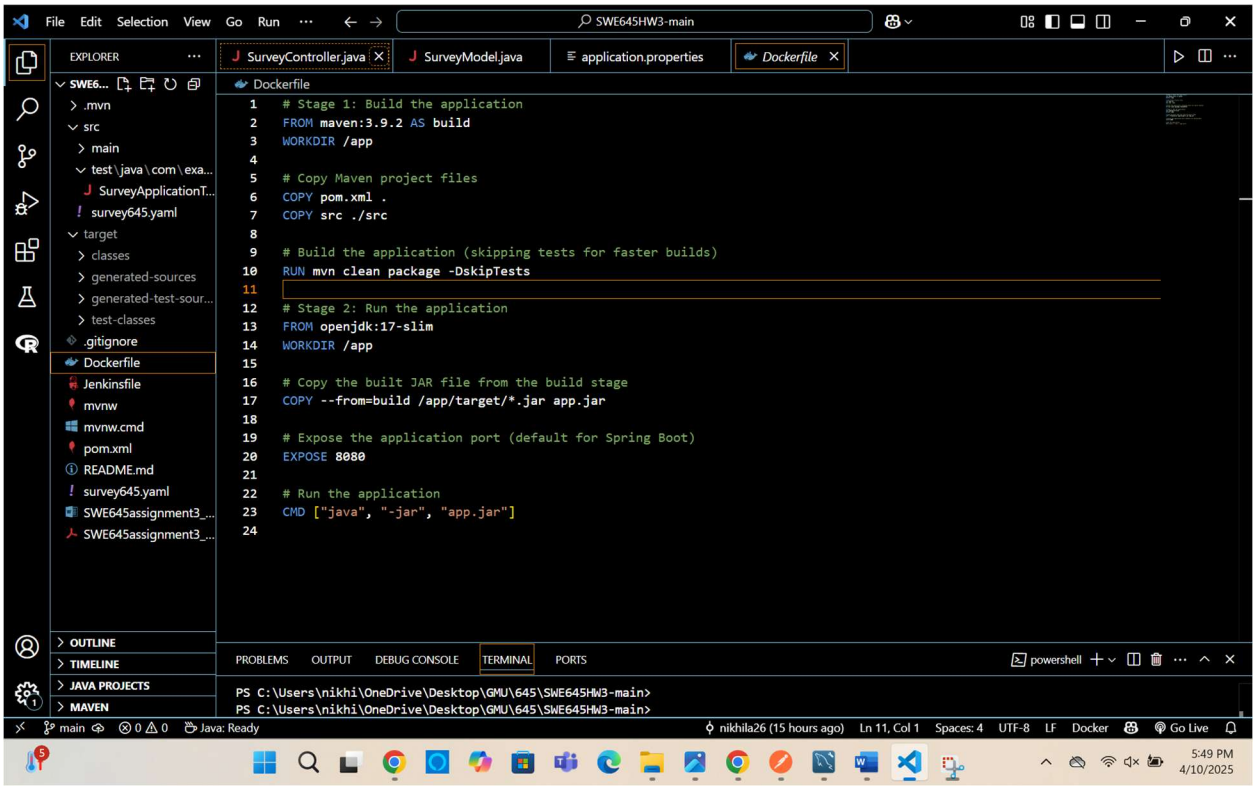
Docker image on dockerhub:

- 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 "nikhila10/survey.war" file are located in the same folder.



nikhila10/survey645:latest

Creation of docker file:

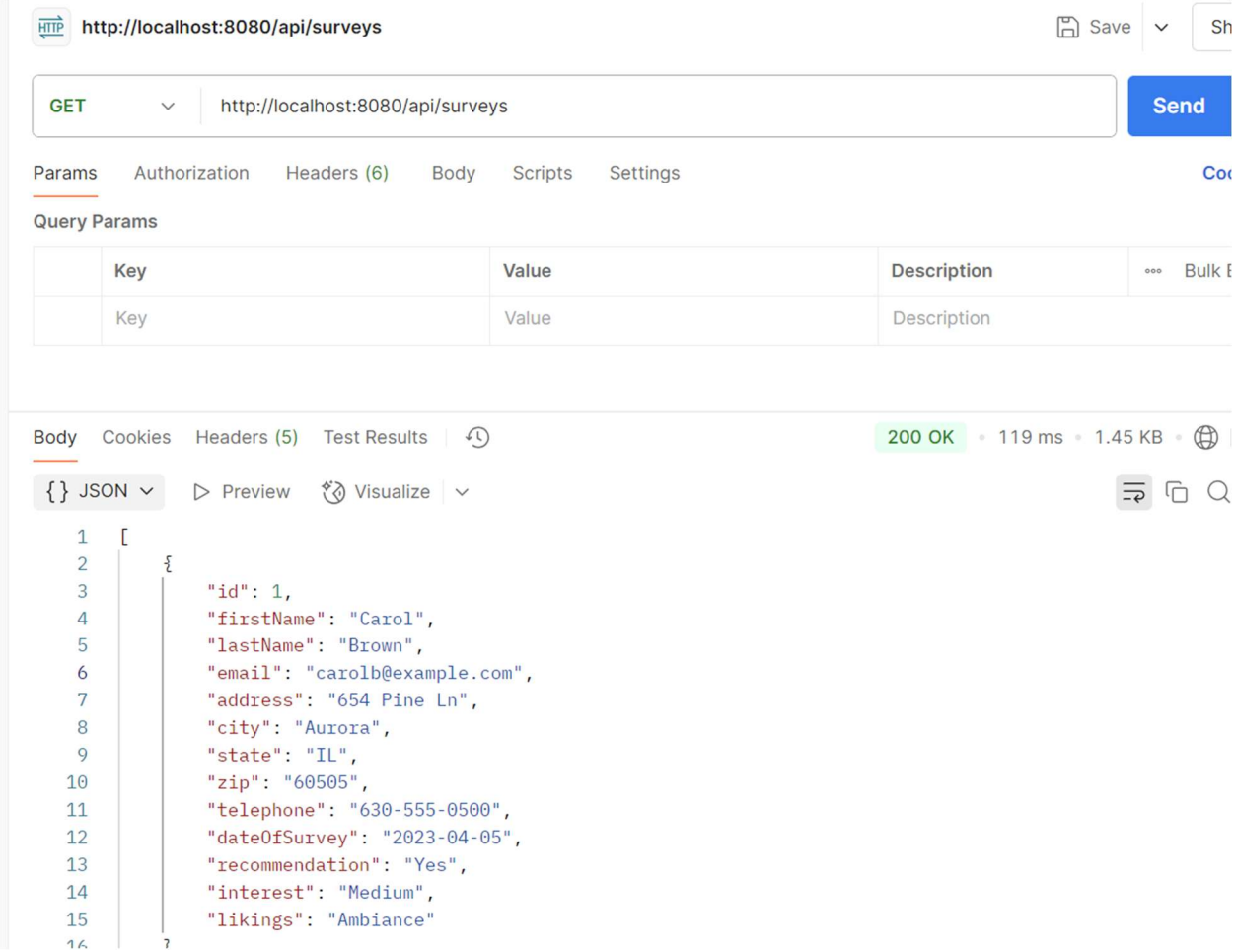


1. Open the command prompt in the directory where the Dockerfile and nikhila10/student.war file are located.
2. To create a Docker image, run the command:
3. docker build -t nikhila10/survey645 .
4. Docker Desktop will build an image for the application on your local machine. To run the application on localhost, execute:
5. docker run -it -p 8081:8080 nikhila10/survey645
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 /survey645, which corresponds to the .war file created using the command:

   The application's webpage should appear in the browser.

   URL: http://localhost:8080/api/surveys
8. Import ant Docker commands are:

   Docker build -t and docker push



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 http://localhost:8080/api/surveys 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 firstName, lastName, email, address, city, state, and zip.
4. **Successful Request**:
   The response suggests that the server is running on localhost:8080, 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.

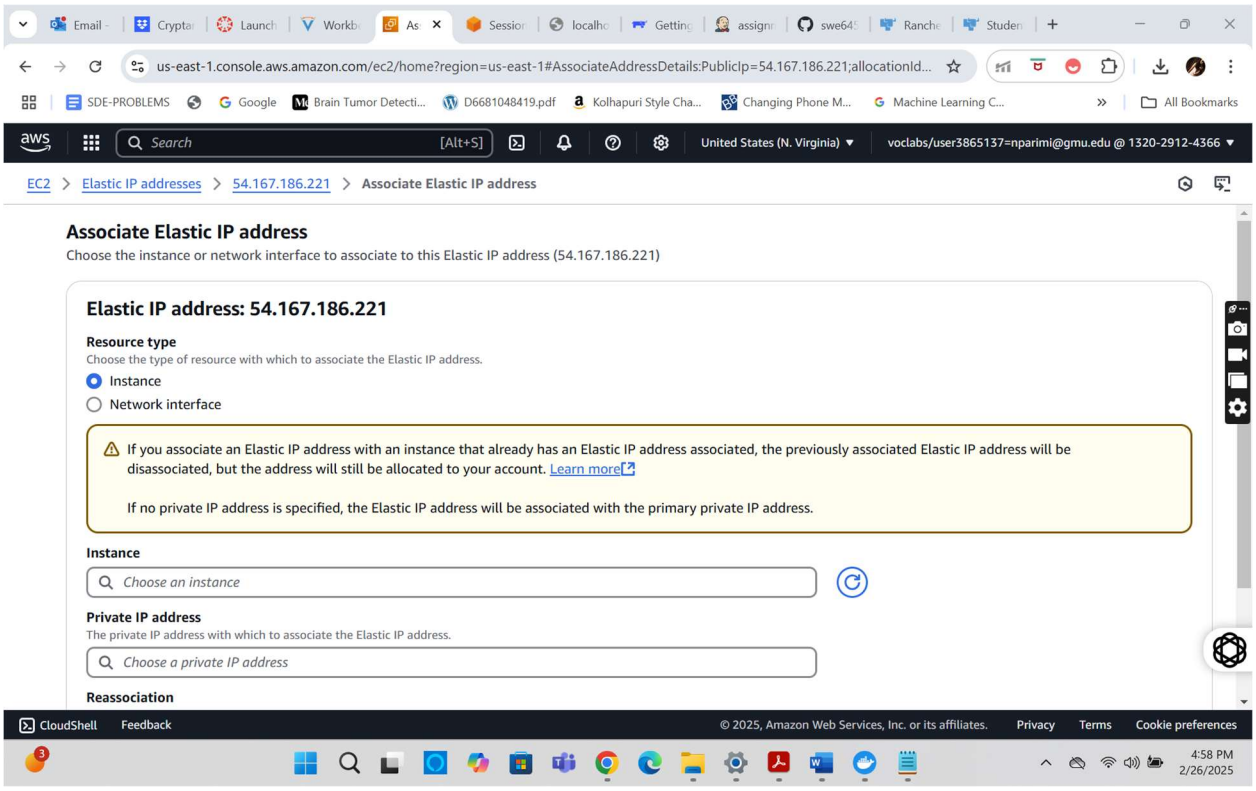
2. Click on "Launch Instance" to create a new instance.

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".

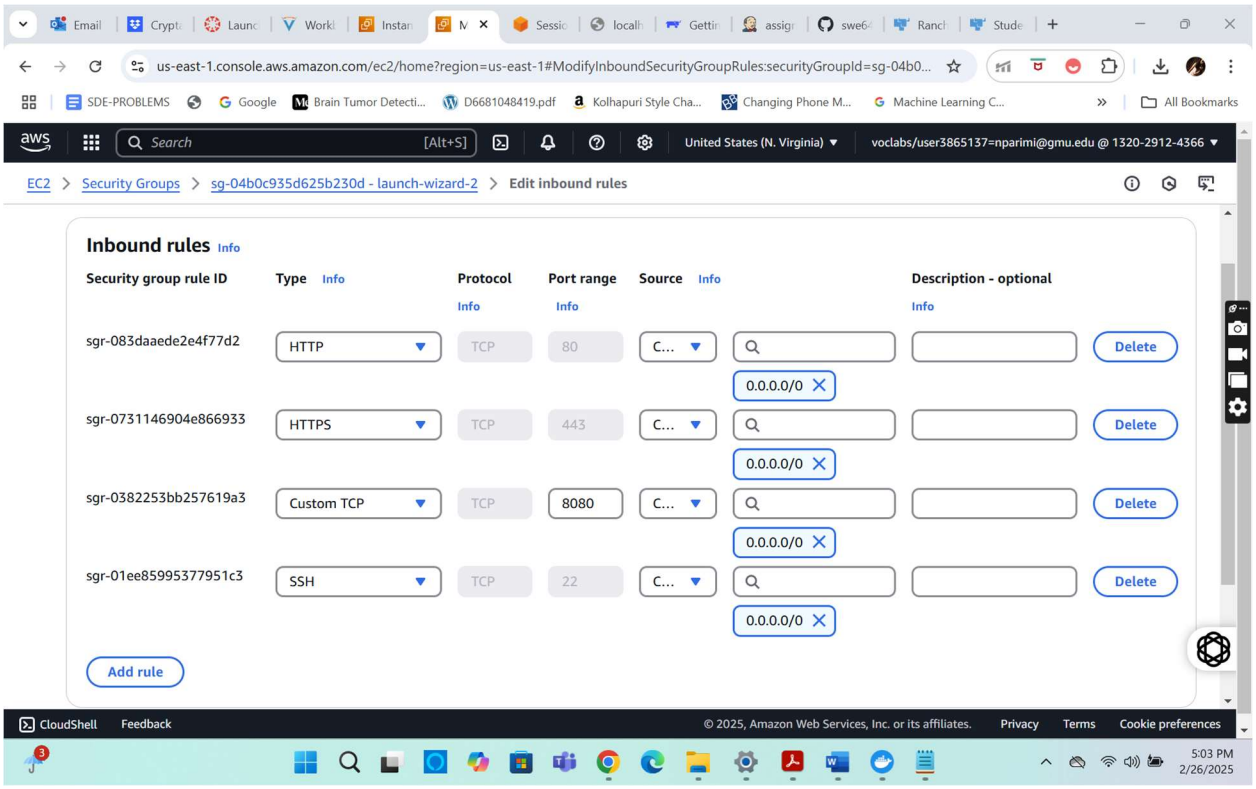
4. Select an existing key pair, in this example, it is "645". Check the boxes to allow HTTP and HTTPS traffic from the internet.

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”.



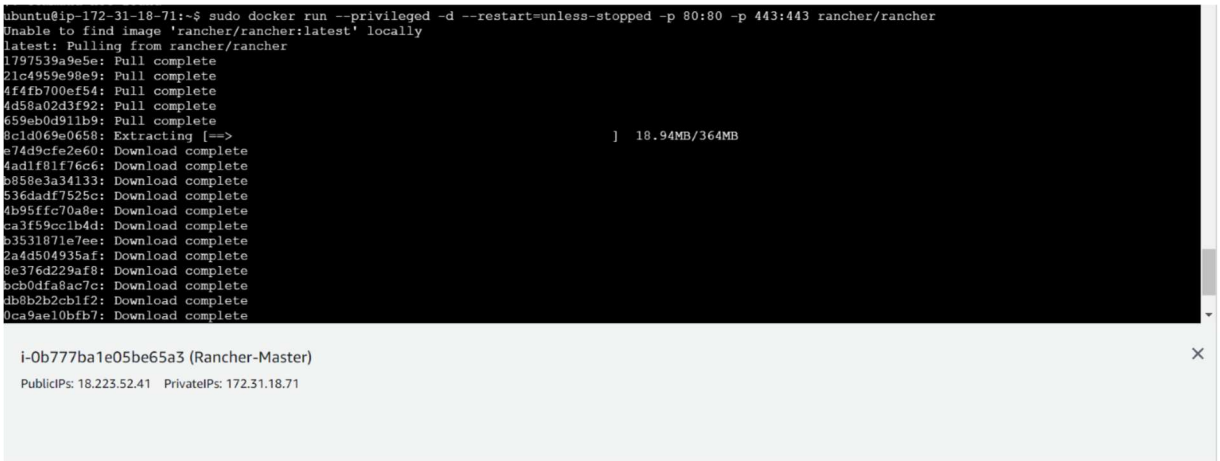
Next, select the EC2 instance and click on "Connect". Navigate to the "EC2 Instance Connect" tab, enter the username "ubuntu", and click on "Connect". This will open a shell in a new tab.

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`.



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-172-31-7-26:~$ sudo docker ps
CONTAINER ID   IMAGE             COMMAND           CREATED          STATUS         PORTS
               NAMES
36571a0d8424   rancher/rancher   "entrypoint.sh"   11 seconds ago   Up 6 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp,
:::443->443/tcp   beautiful_tharp
ubuntu@ip-172-31-7-26:~$
```

i-0652adf76ef505255 (Rancher-Master)

PublicIPs: 54.167.186.221 PrivateIPs: 172.31.7.26

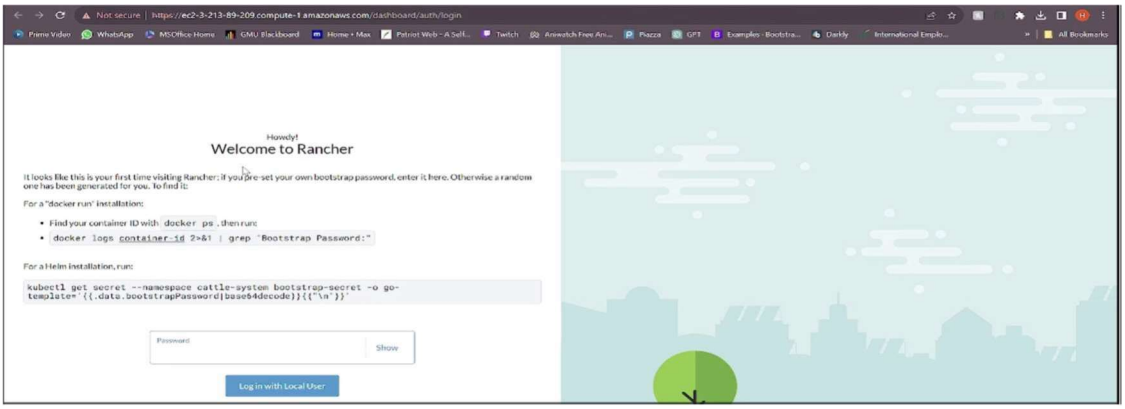
```
ubuntu@ip-172-31-7-26:~$ sudo docker logs 36571a0d8424 2>&1 | grep "Bootstrap Password"
2025/04/10 04:42:44 [INFO] Bootstrap Password: 6b425tw6dpjsq4mczdlmtct5vjgp2jdqtc2xqz78brm45dcdn98jtj
ubuntu@ip-172-31-7-26:~$
```

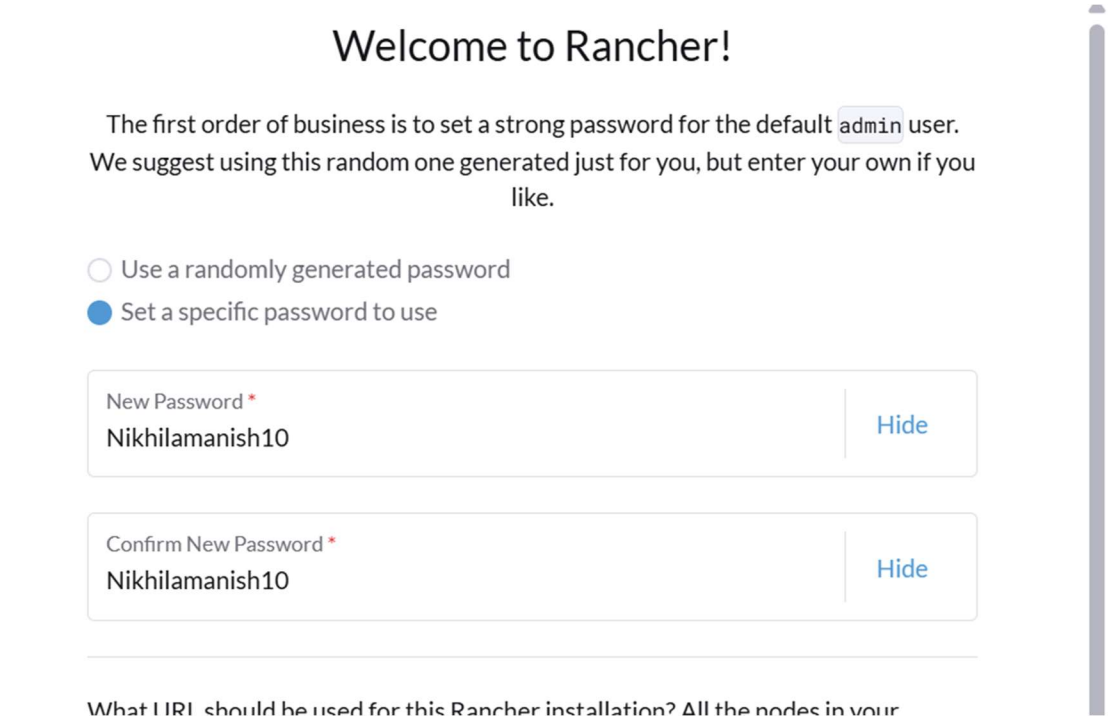
i-0652adf76ef505255 (Rancher-Master)

PublicIPs: 54.167.186.221 PrivateIPs: 172.31.7.26

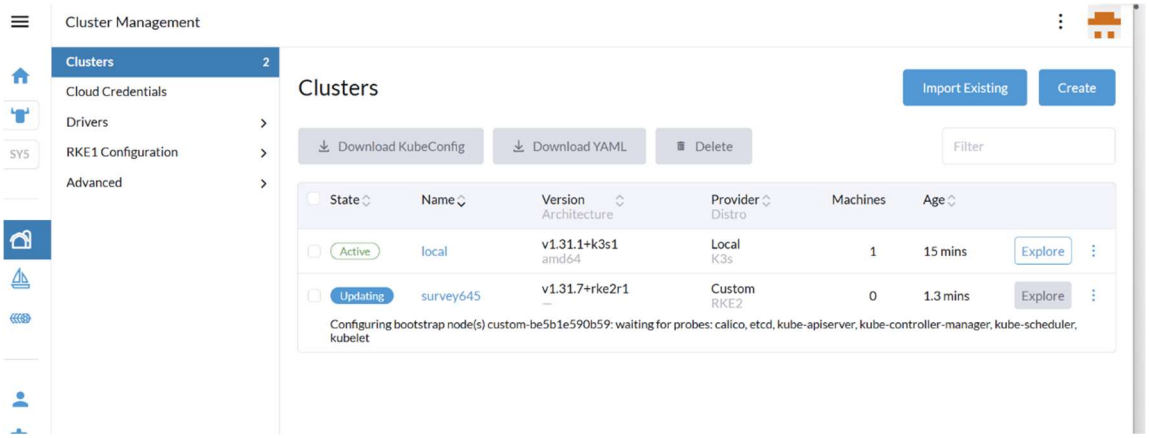
Creation of instances:

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.



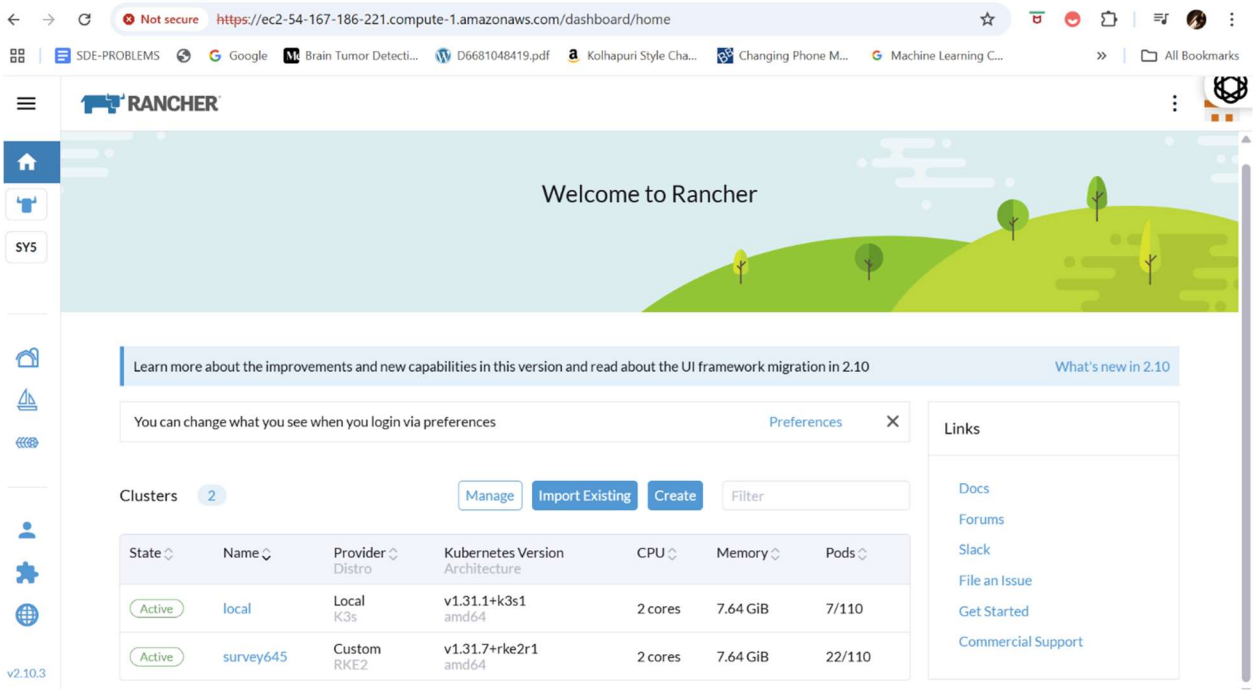


Once the cluster is in the "Active" state, click on the "Explore" button which is next to the cluster.



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:
    - **Namespace**: default
    - **Name**: name to it
    - **Replicas**: 3 (this is the number of pods)
    - **Container Image**: nikhila10/survey645:latest (make sure this matches the name of the image on Docker Hub)

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



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

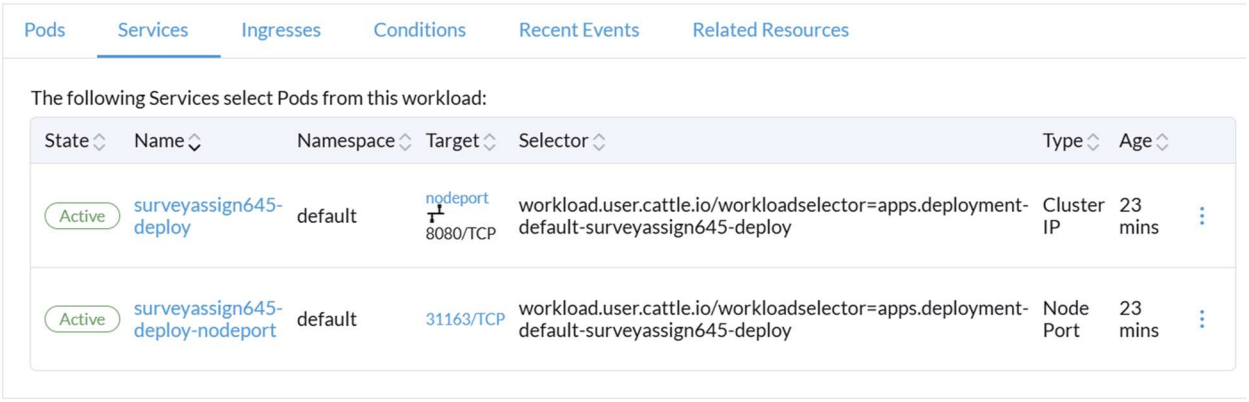
Fill in the following details:

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

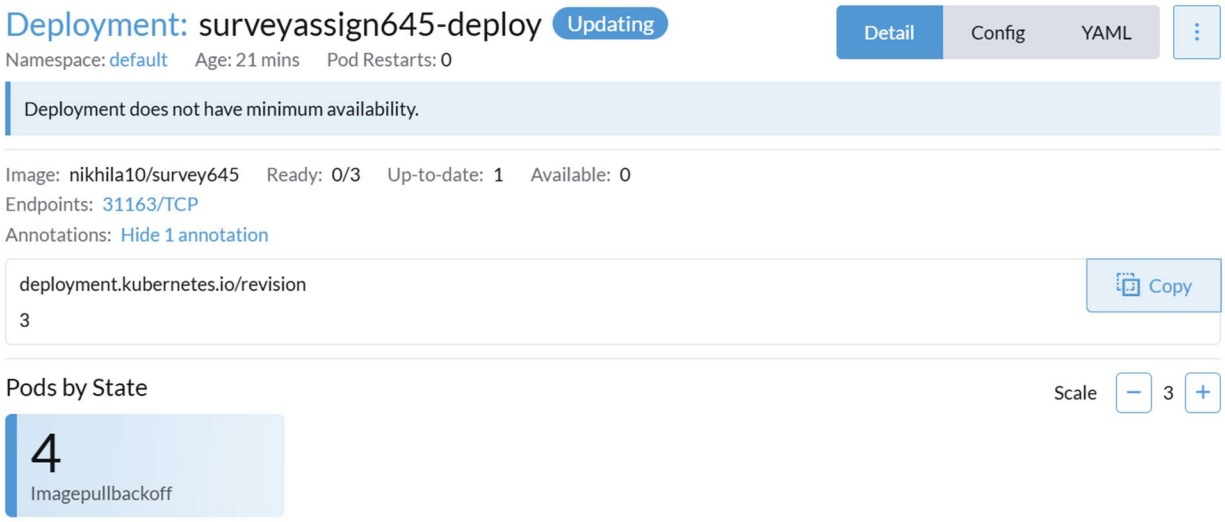
Leave all other settings as default and click on **Create**.

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.



Once everything is adjusted properly you can see the 4 instances deployed properly.



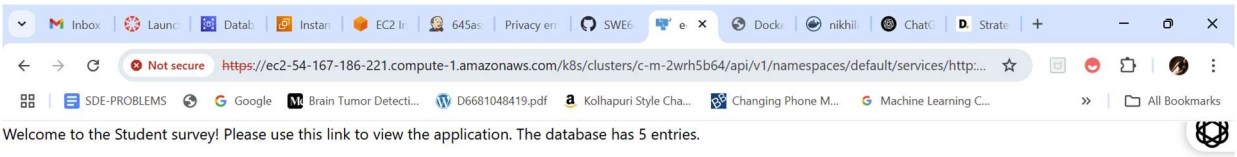
Testing the link deployed on nodeport in kubernetes:

Accessing the Service: The image shows a web browser accessing a service running on a Kubernetes cluster via the

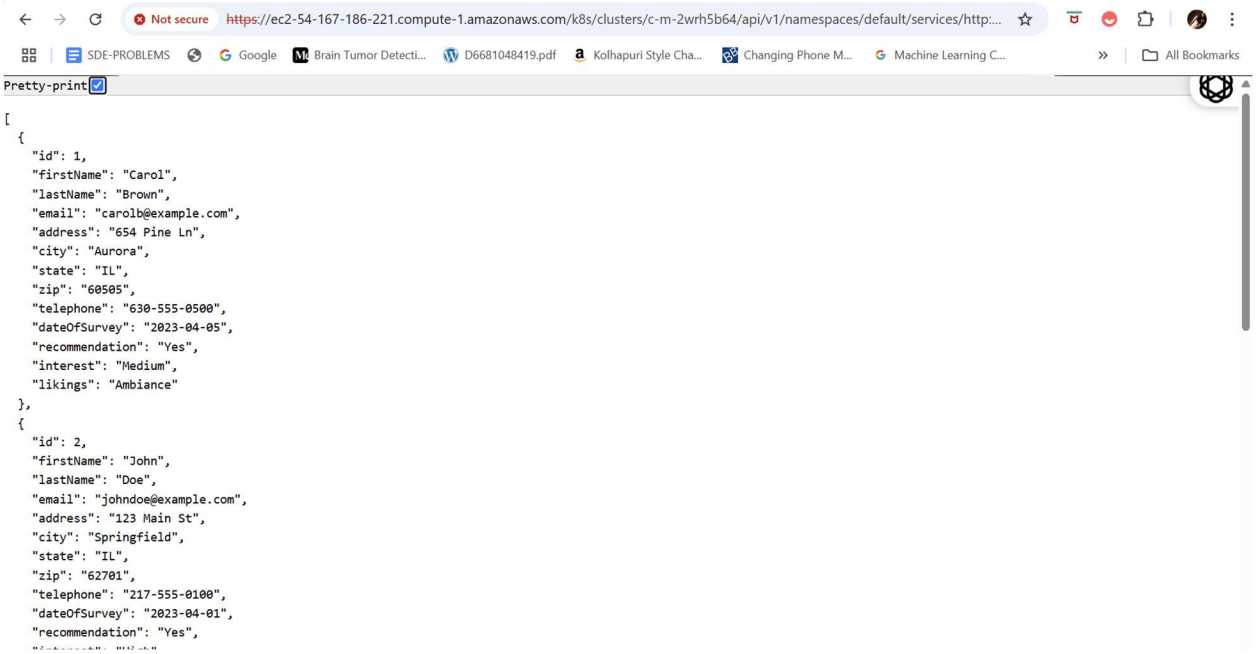
URL: https://18.232.206.204/k8s/clusters/c-m-v5mz5hgq/api/v1/namespaces/default/services/http:surveyassign3-deploy:8080/proxy/. This URL structure indicates that the user is leveraging a Kubernetes proxy to reach the service.

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.



URL : https://ec2-54-167-186-221.compute-1.amazonaws.com/k8s/clusters/c-m-2wrh5b64/api/v1/namespaces/default/services/http:surveyassign645-deploy:8080/proxy/



The URL for viewing all the surveys is: https://ec2-54-167-186-221.compute-1.amazonaws.com/k8s/clusters/c-m-2wrh5b64/api/v1/namespaces/default/services/http:surveyassign645-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)

```
ubuntu@ip-172-31-86-97:~$ sudo apt install jenkins
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Start Jenkins using the command:

- sudo systemctl start jenkins.service

```
ubuntu@ip-172-31-86-97:~$ sudo systemctl start jenkins.service
ubuntu@ip-172-31-86-97:~$ sudo ufw allow 8080
Rules updated
Rules updated (v6)
ubuntu@ip-172-31-86-97:~$
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
7ce996ea15de4093b9611537eb28a3ea
```

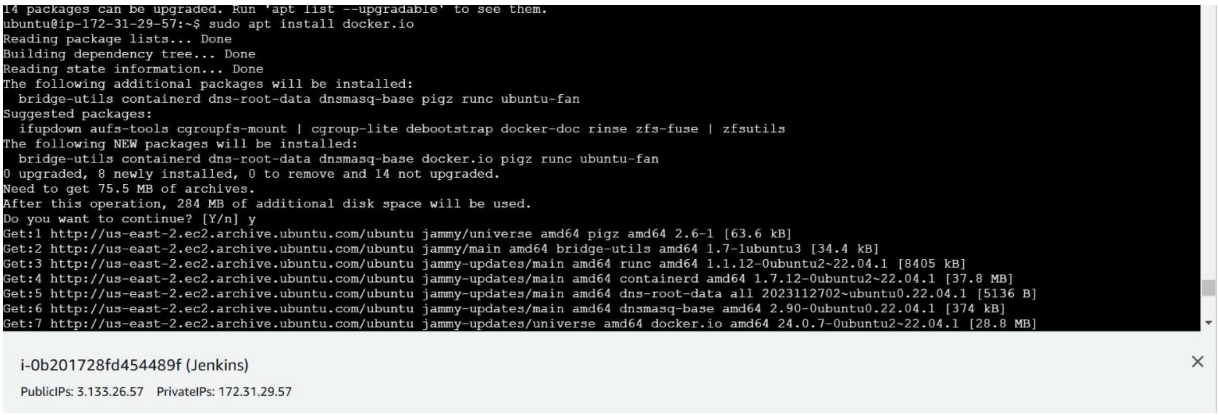
Give the default admin command to enter to jenkins

Now download the suggested plugins:

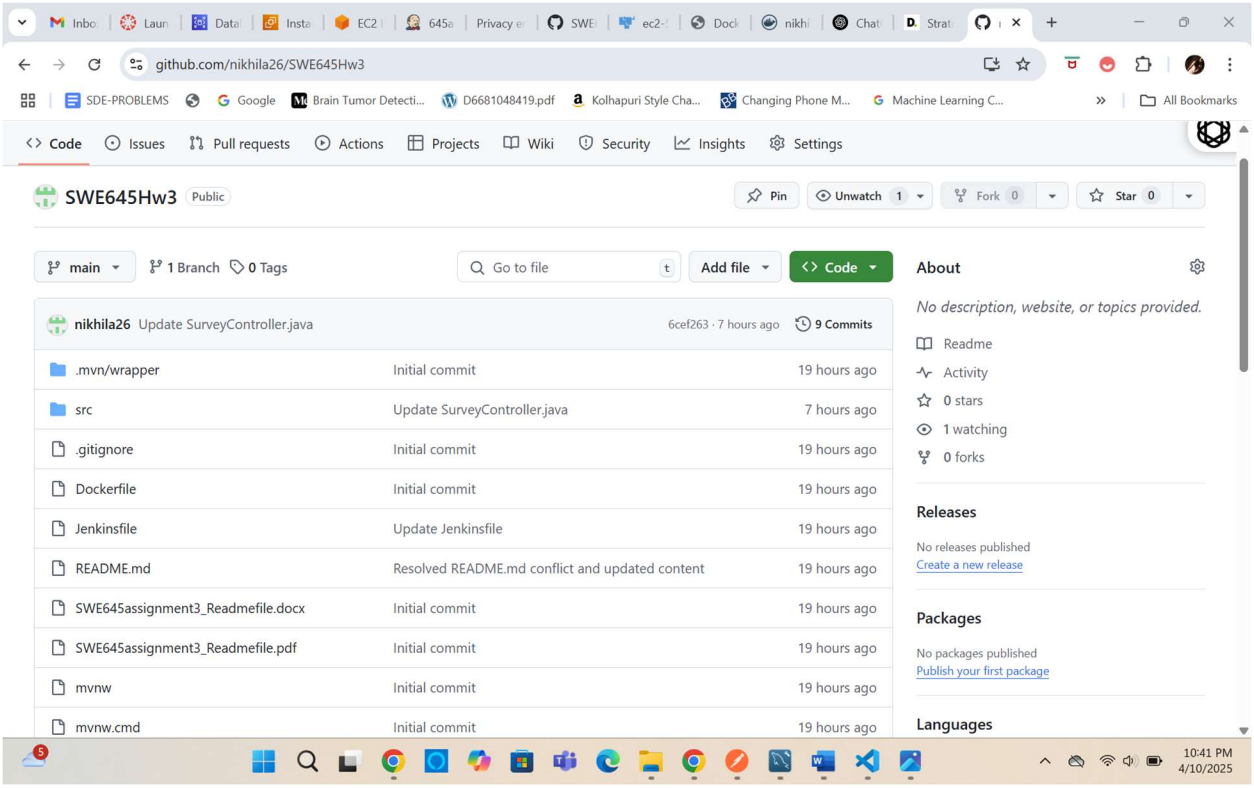
- Return to the EC2 console to create a config file.
- Run sudo su jenkins to switch to the Jenkins home.
- Navigate to the root directory: cd ../../.
- Go to the Jenkins directory: cd /var/lib/jenkins.
- Create a directory and enter it: mkdir .kube followed by cd .kube.
- Create and open the config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



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/nikhila26/SWE645Hw3



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.

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.

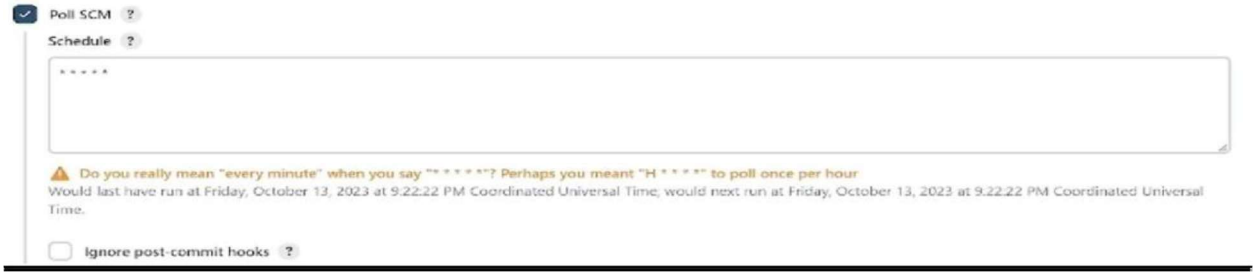
1. **Save GitHub Credentials**:
    - Navigate to the "Credentials" section under "Manage Jenkins".
    - Select the "global" option and click on "Add credentials".
    - Set the **Kind** to "Username and password".
    - Enter your GitHub username.
    - Use your GitHub PAT (Personal Access Token) as 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/nikhila26/SWE645Hw3
- Scroll down and check the "Poll SCM" checkbox.

    Give the Schedule as: "* * * * *"



Scroll down to the “Pipeline” section.

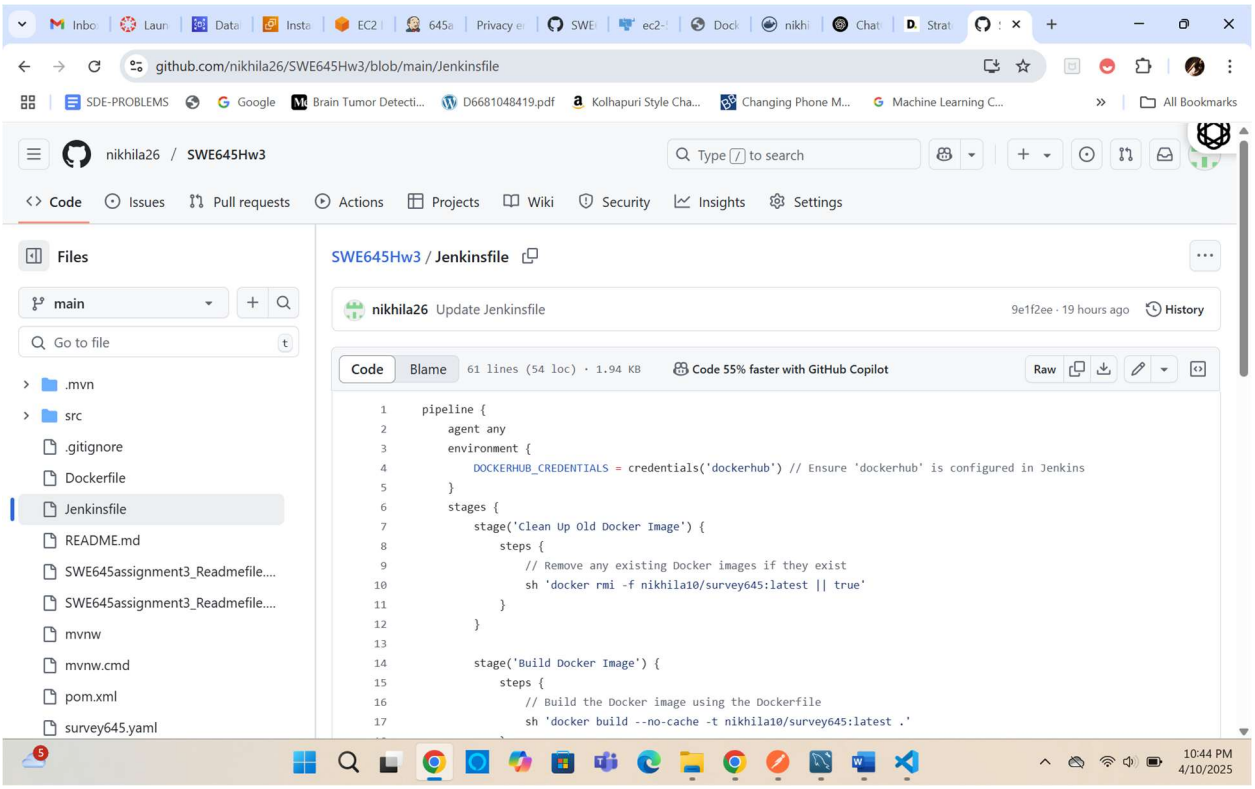
Select the definition: “Pipeline from SCM” and SCM: “git”.

Give the Github repository: https://github.com/nikhila26/SWE645Hw3
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.



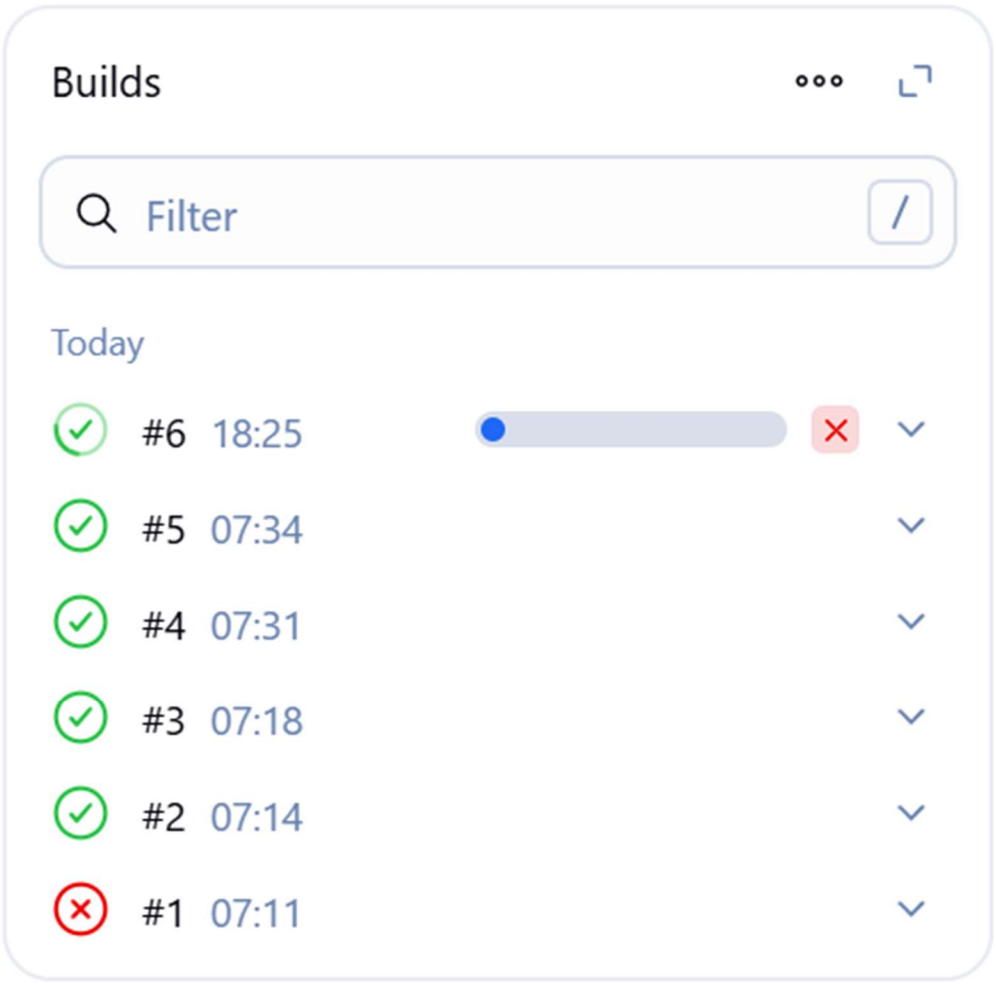
Creation of jenkins file in github:

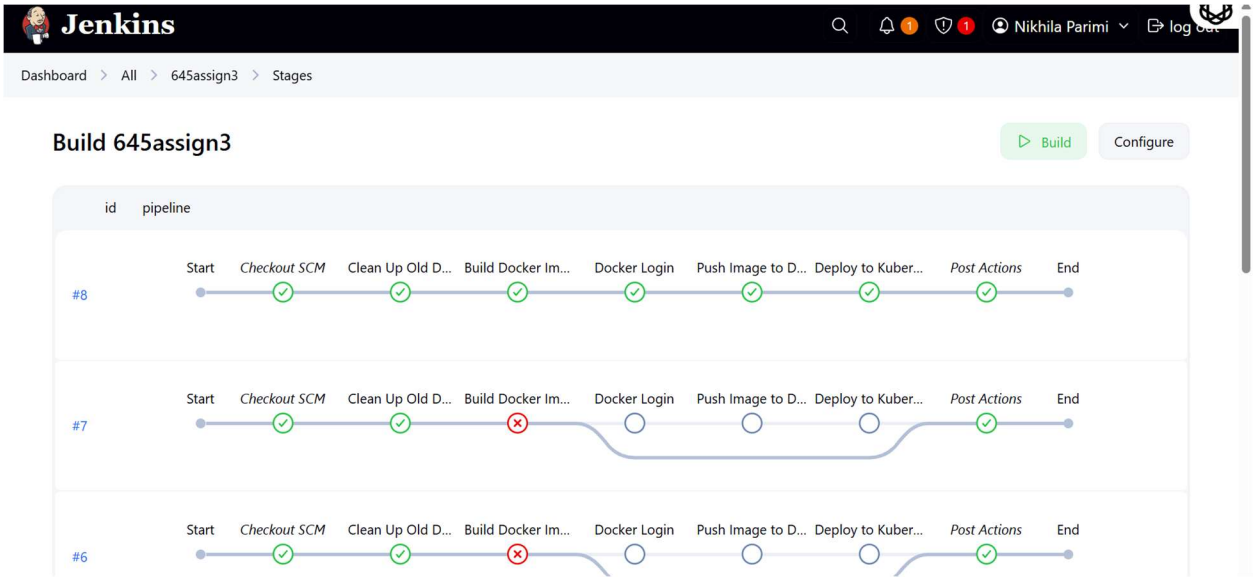
You will get multiple errors while creating, keep on correcting those errors and updates and committing the jenkins file.

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.


Builds
Filter
/
Today
#6 18:25
#5 07:34
#4 07:31
#3 07:18
#2 07:14
#1 07:11



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://ec2-54-167-186-221.compute-1.amazonaws.com/k8s/clusters/c-m-2wrh5b64/api/v1/namespaces/default/services/http:surveyassign645-deploy:8080/proxy/students

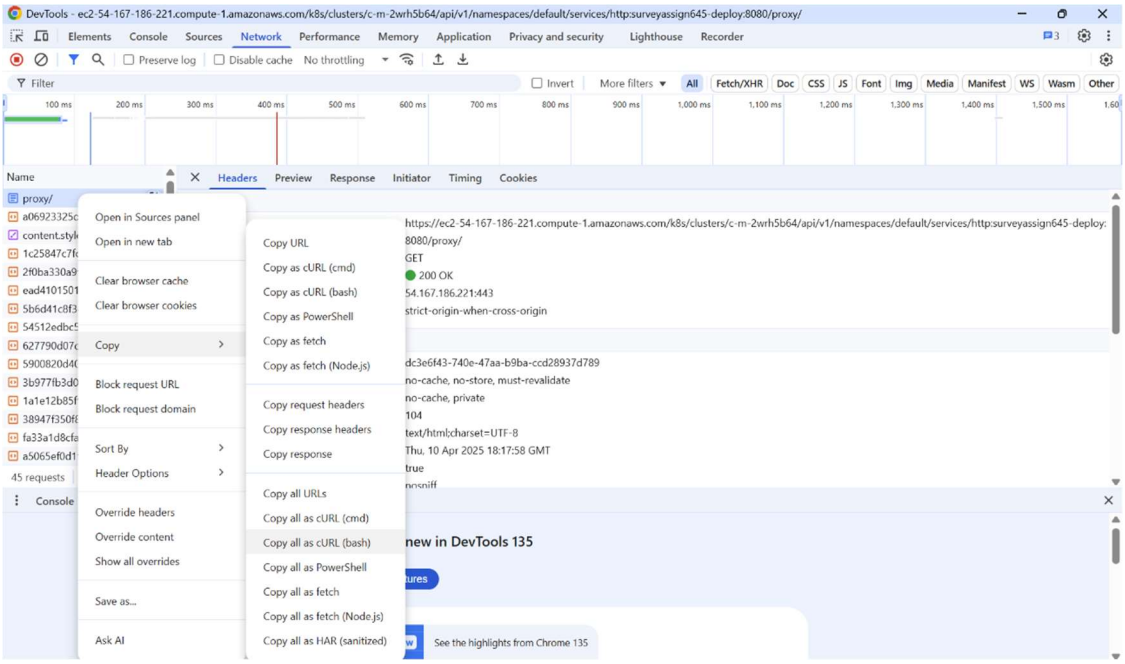
3. Successful Response: The response status is `200 OK`, indicating the request was successful.

4. Response Data: The response body contains JSON data with the following details:

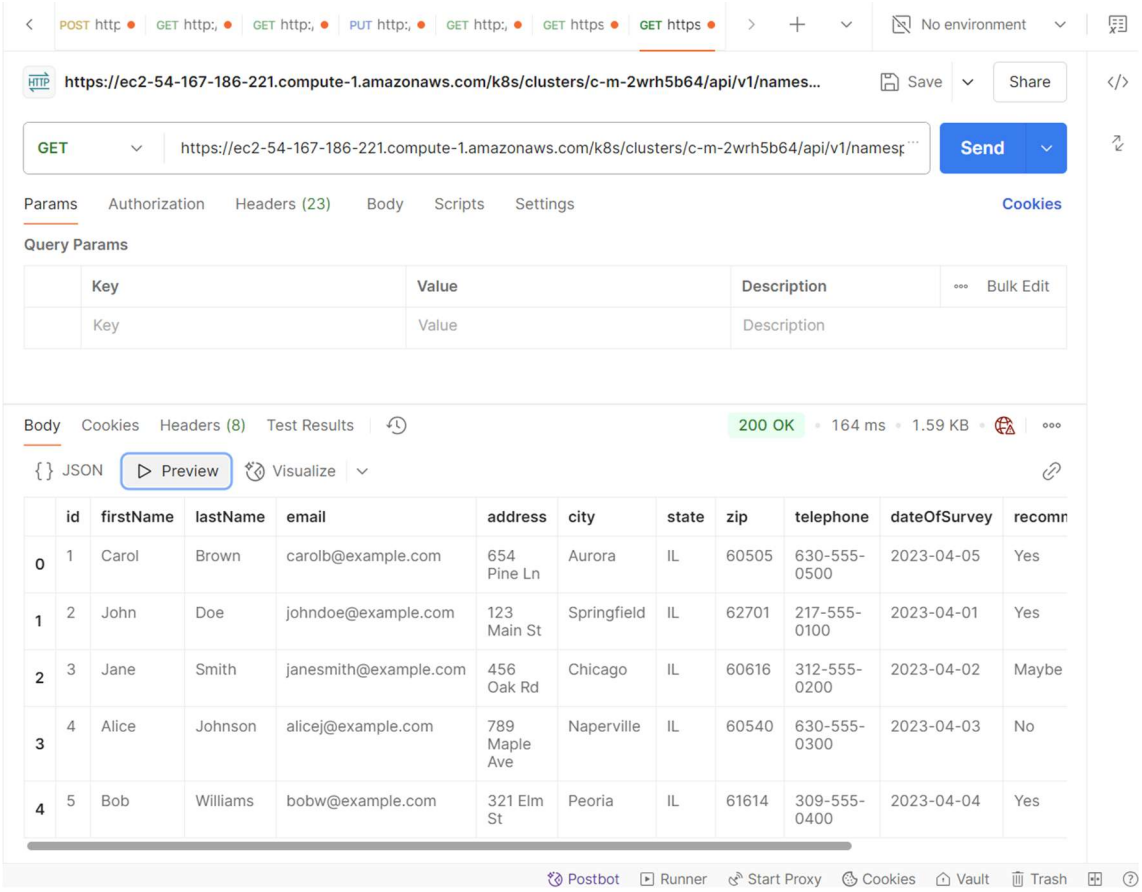
“Welcome to student survey! Please use this link to view the application. The database has a few entries.”

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

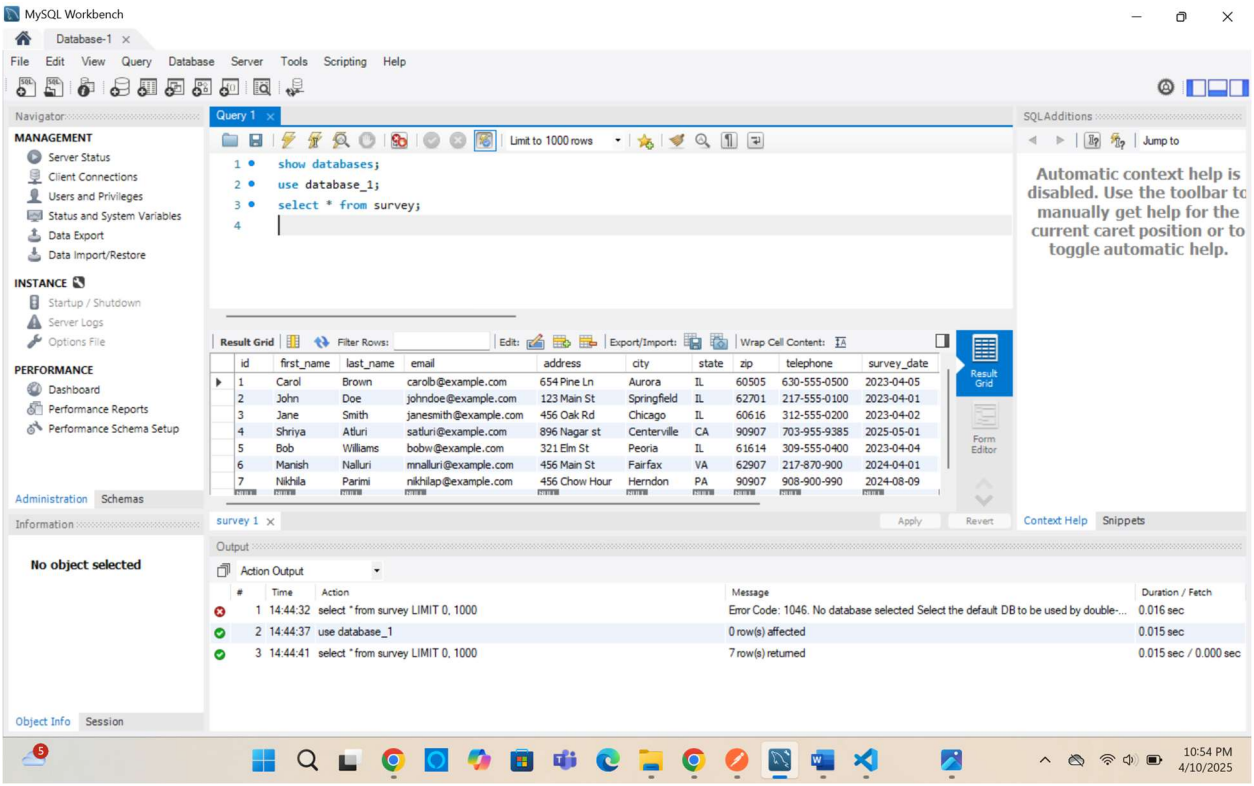
```html
Welcome to the Student Management Application! Please use this database to view the student information.
```



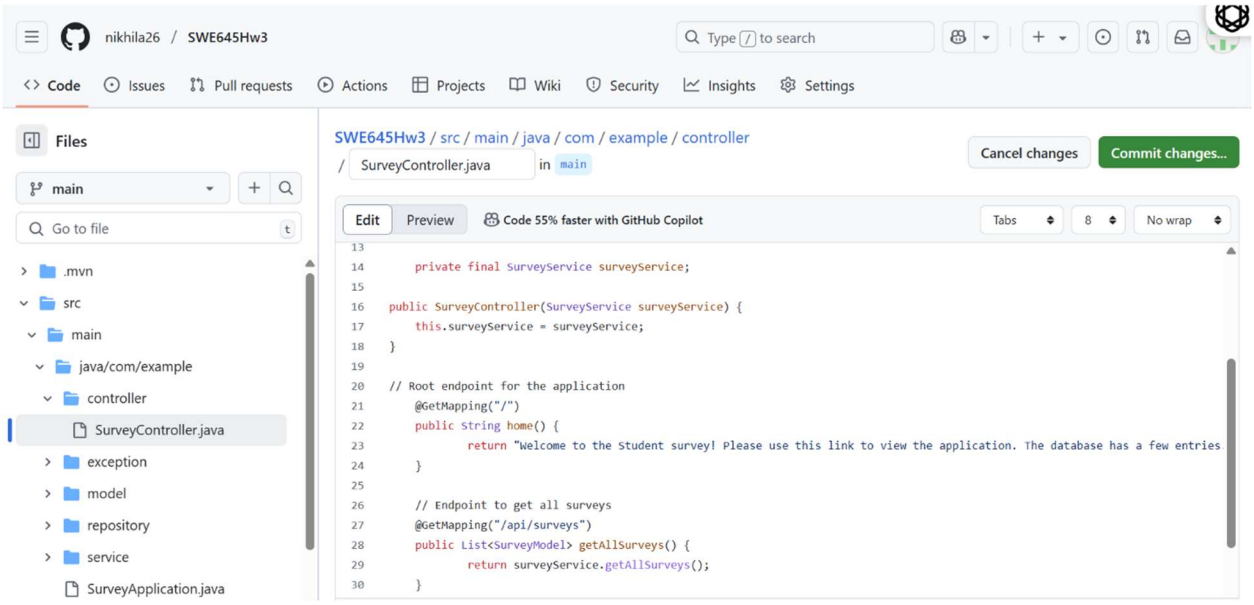
Here to get the link in the postman goto copy >>copy all as URL and then past the link in the post man to get the required output



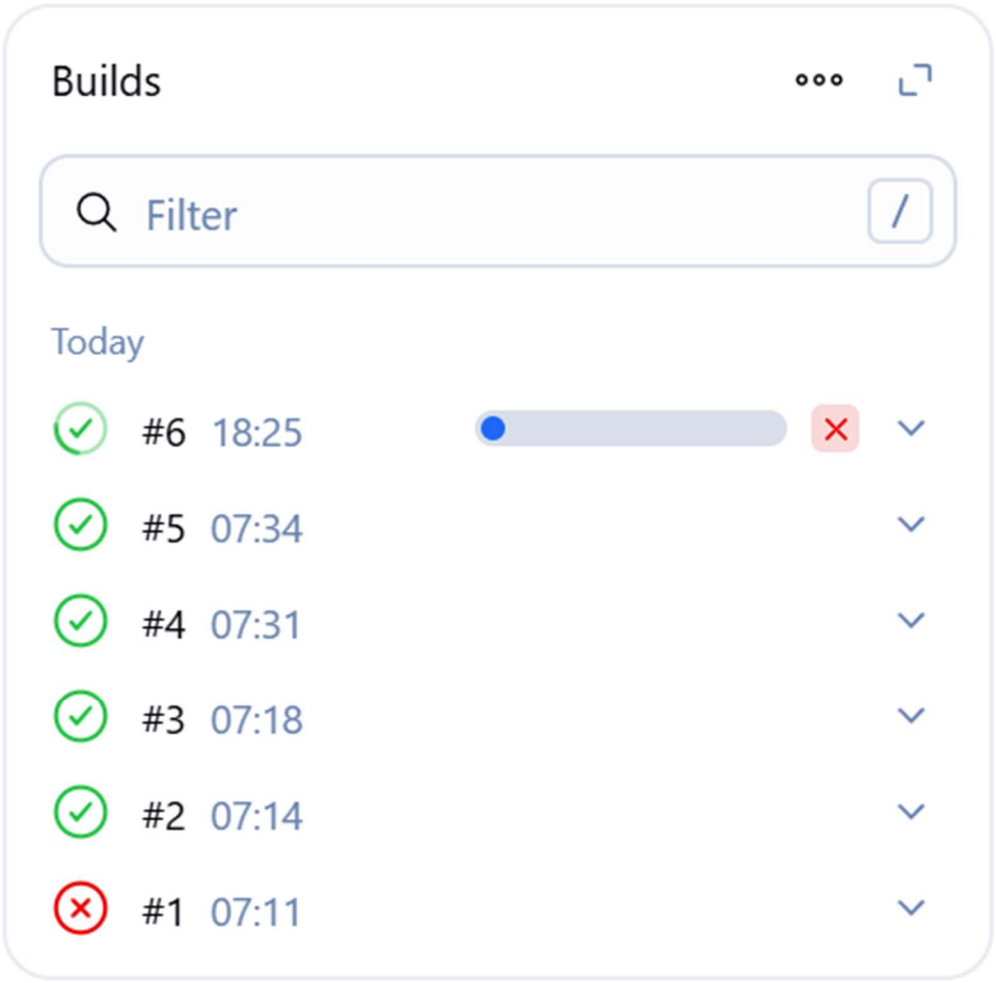
The above picture shows the successful running of API



Now, lets check the same in MYSQL database.

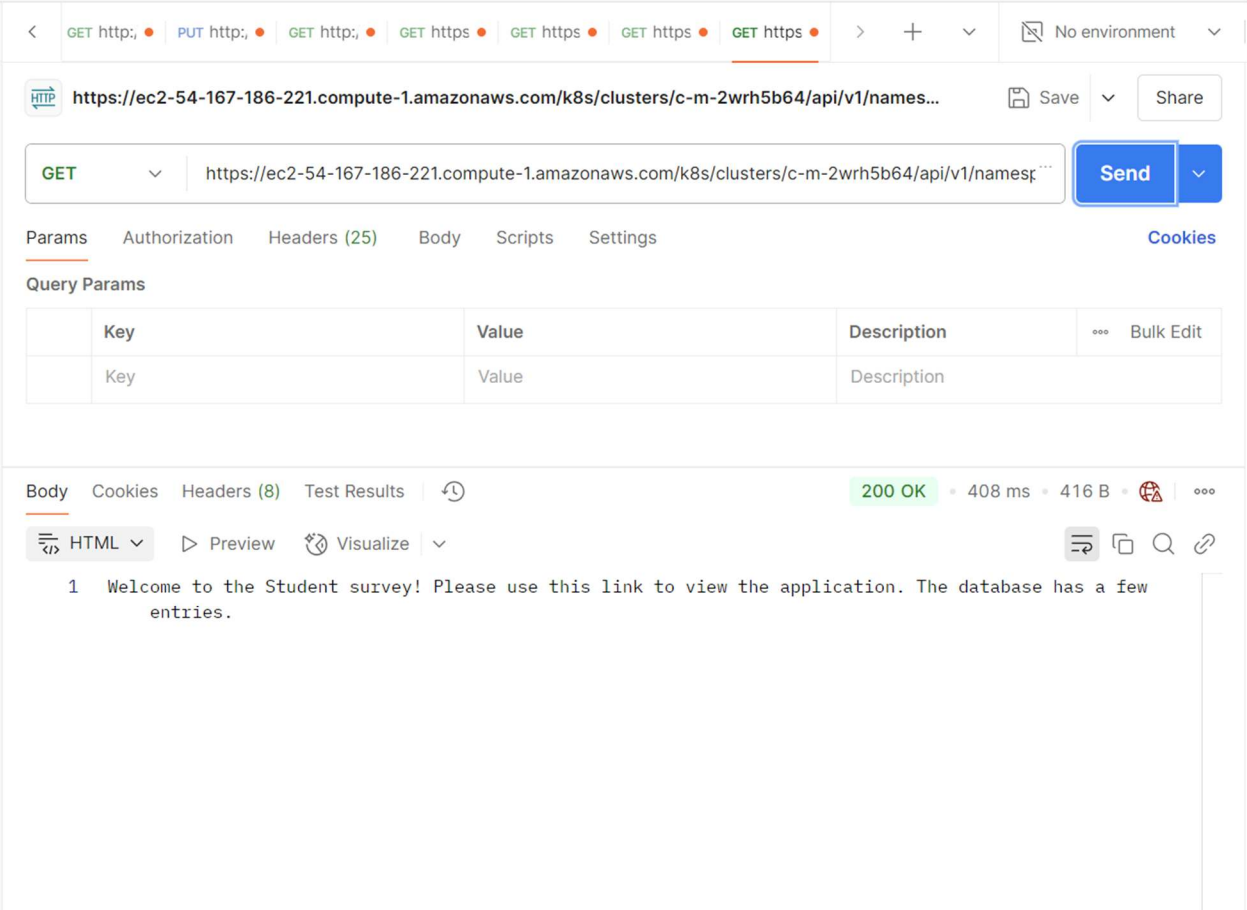


A small change is performed on github, to check if it reflects in the pipeline. And then checked with postman. Above, you can see that the changes are perfectly getting reflected.



The pipeline is getting built.

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



The change is successfully reflected.

URL : https://ec2-54-167-186-221.compute-1.amazonaws.com/k8s/clusters/c-m-2wrh5b64/api/v1/namespaces/default/services/http:surveyassign645-deploy:8080/proxy/

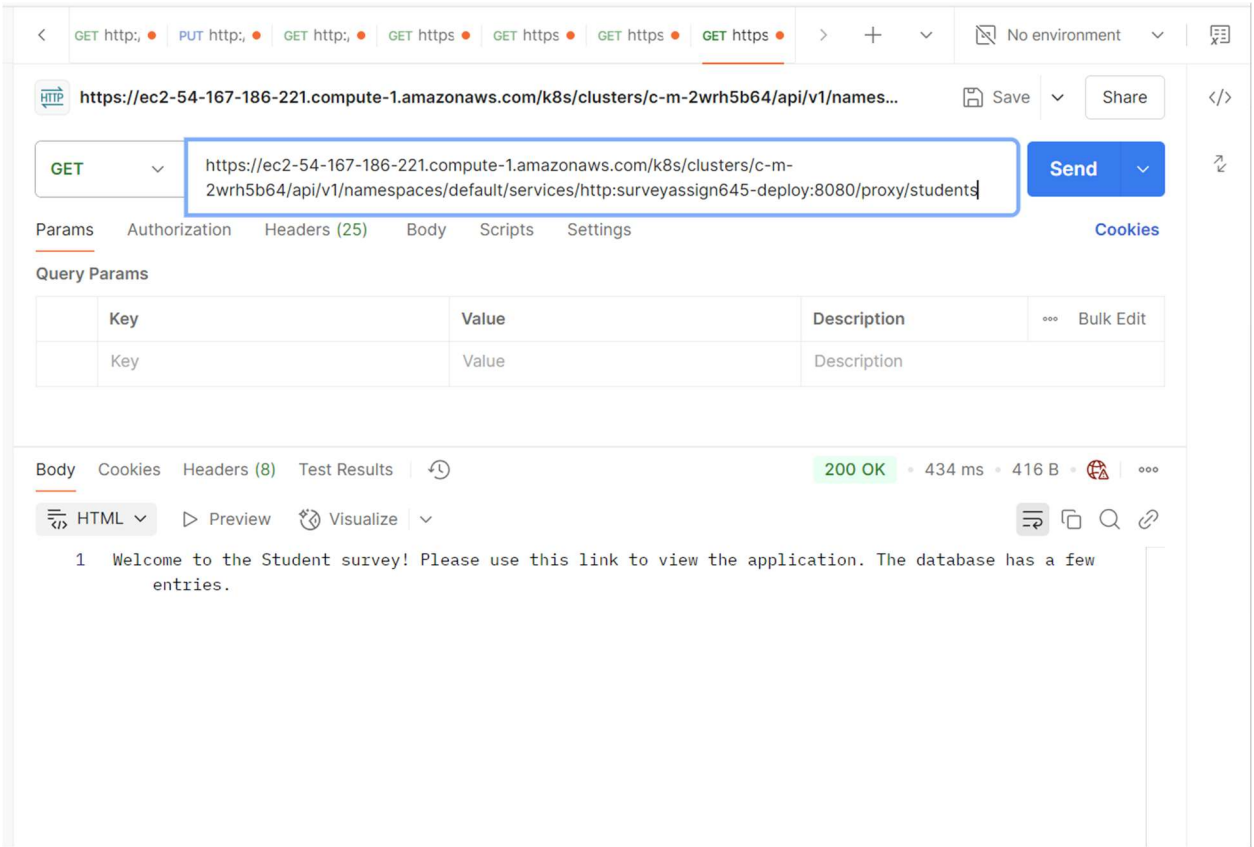
When changes are made to the data in GitHub, and these changes are reflected not only in Postman but also in the database, here's a detailed explanation of how this process typically works:

Workflow Overview

1. **Update Code in GitHub**: Changes are made to the source code, configuration files, or database schema in the GitHub repository.
2. **Jenkins Pipeline Trigger**: The Jenkins CI/CD pipeline detects these changes and initiates a build process.
3. **Build and Deploy**: Jenkins builds the updated code, creates a new Docker image, and deploys it to the Kubernetes cluster.
4. **Database Migration/Update**: As part of the deployment process, database migrations or updates are applied to reflect the new changes.

5. **Verify with Postman**: Use Postman to verify that the changes are successfully reflected in the application's API responses.
6. **Database Verification**: Confirm that the updates are correctly applied to the database.

Above picture stating that we can fetch the changes even after changing the path. Here we are changing the path from " / " to "/students"



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

https://ec2-54-167-186-221.compute-1.amazonaws.com/k8s/clusters/c-m-2wrh5b64/api/v1/namespaces/default/services/http:surveyassign645-deploy:8080/proxy/students

Here's a concise description of the process:

1. **Postman Setup**: Open Postman and check the API endpoint mappings to determine the correct URL to test, such as /api/surveys or /student.
2. **Send Request**: Based on the mappings, use Postman to send a request to the specified endpoint.
3. **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. **NIKHILA PARIMI (G0156266):** 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. **MANISH NALLURI (G01453450):** 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. **VIGNESH REDDY MOLUGU (G01502803):** 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. **VENNA SUJITH REDDY (G01515456):** 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.