**Create a RDS:**

* Open the AWS account and open the RDS service by searching for it.
* Go to the database and click on Create Database.

**A screenshot of a computer

Description automatically generated**

* Choose the method as standard and select the MySQL engine.

**A screenshot of a computer

Description automatically generated**

* Under the templates, select the free tier option and change the database identifier, and make sure to keep it unique from the other RDS instances created.
* Under credential settings, change the username and choose self-managed password type and enter a custom password

A screenshot of a computer

Description automatically generated

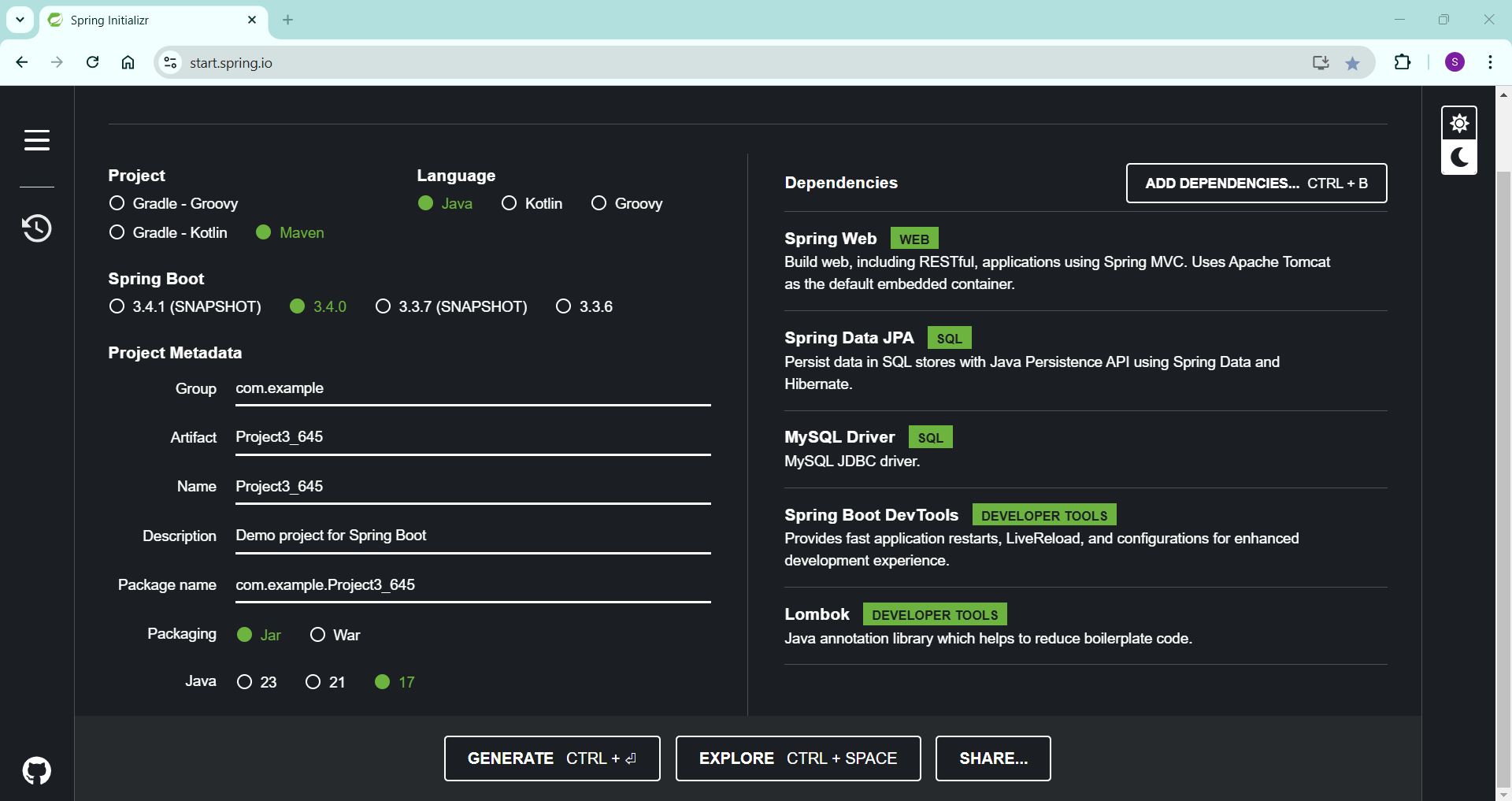
* Under instance configuration, select t4g.micro and default 20 gb ssd instance
* Choose Don’t connect to an EC2 instance and set public accessibility to Yes.
* Create a new VPC security group and click on Create Database.
* After the RDS instance becomes available and it finishes backing up, note down the endpoint URL and keep a note of the username and password.

A screenshot of a computer

Description automatically generated

**Create a Springboot Project:**

* Go to the spring initializer
* Configure the project by selecting the project type as Maven, the language as Java, and the latest stable version of Spring Boot.
* Enter a unique identifier for the group and specify the name of our project.
* Select Java version 11
* Add dependencies- Spring Web, Spring Data JPA, MySQL Driver, Spring Boot DevTools, Lombok
* Click "Generate" to download the project as a .zip file



* Extract the .zip file.
* Import the project into IntelliJ IDEA
* In the src/main/resources, Open the application, and properties and update the username, password, and the endpoint of the RDS

A screenshot of a computer program

Description automatically generated

* Create a controller SurveyController for CRUD operations and handling HTTP requests.
* Created a model Survey to define all the data fields that our survey form has.
* Created a repository called SurveyRepository for interacting with the database

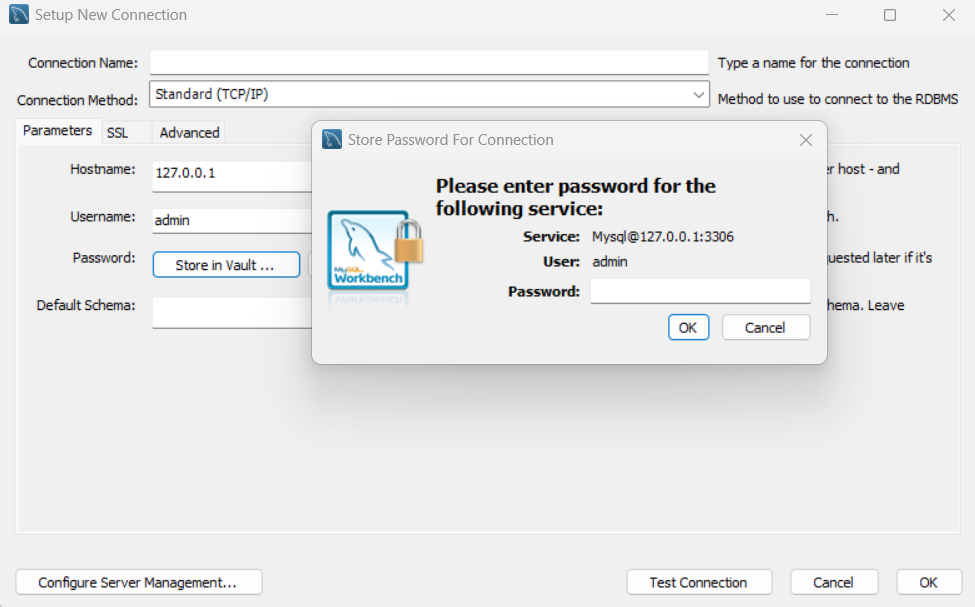
**Connect to Database:**

* Open mysql workbench, and create a new connection.

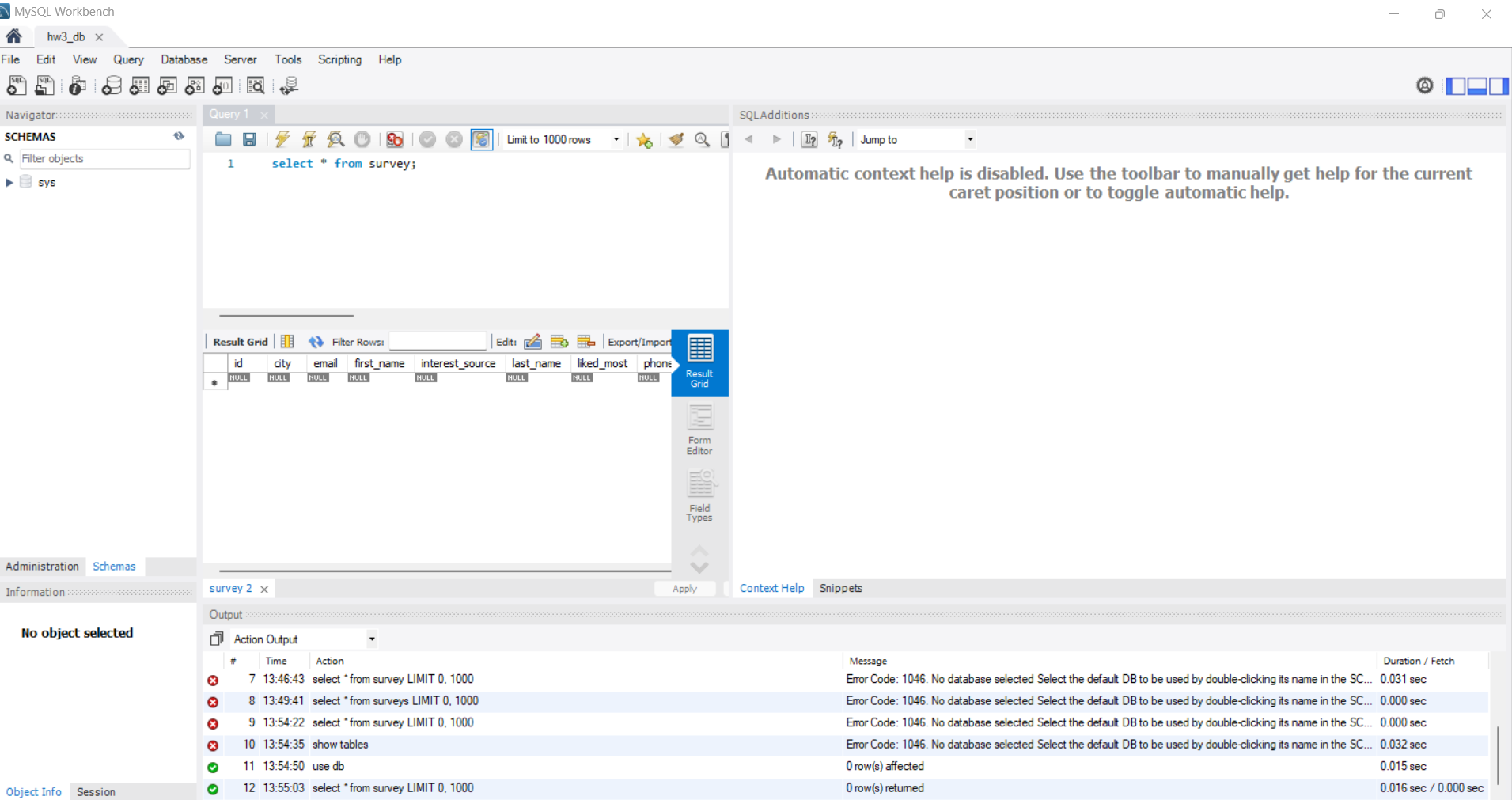
**A screenshot of a computer

Description automatically generated**

* Give the name of the connection, enter the endpoint URL in the hostname, enter the username and password, and connect it.

****

* Create a new database in this, and check if the CRUD operations of our project are working using Postman.

****

**A screenshot of a computer

Description automatically generated**

**Conterizing with Docker:**

* create a Dockerfile and build a Docker image.

docker build -t project3-app:1.0.

* Run the docker container

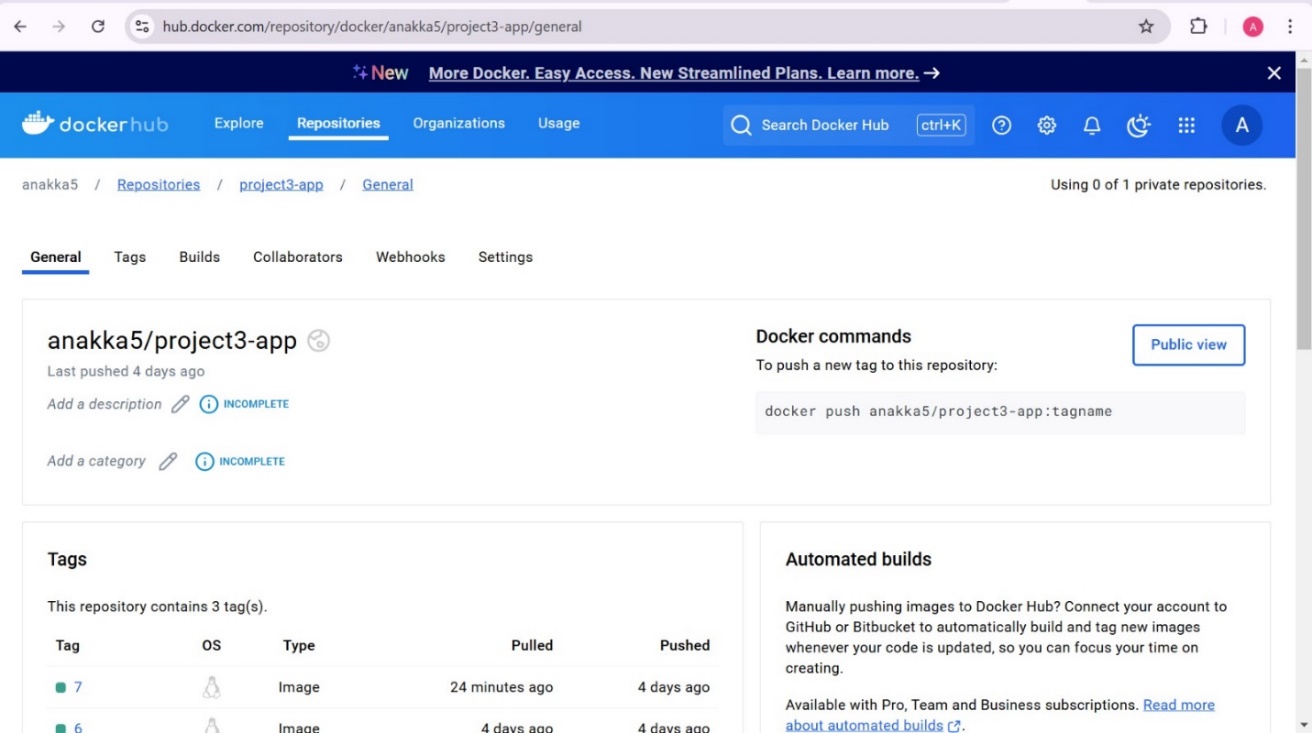
docker run -p 8080:8080 project3-app:1.0

* Use the following command to tag the image

docker tag project3-app:1.0 saranyakorrapati/project3-app:1.0

* Push the docker image

docker push saranyakorrapati/project3-app:1.0

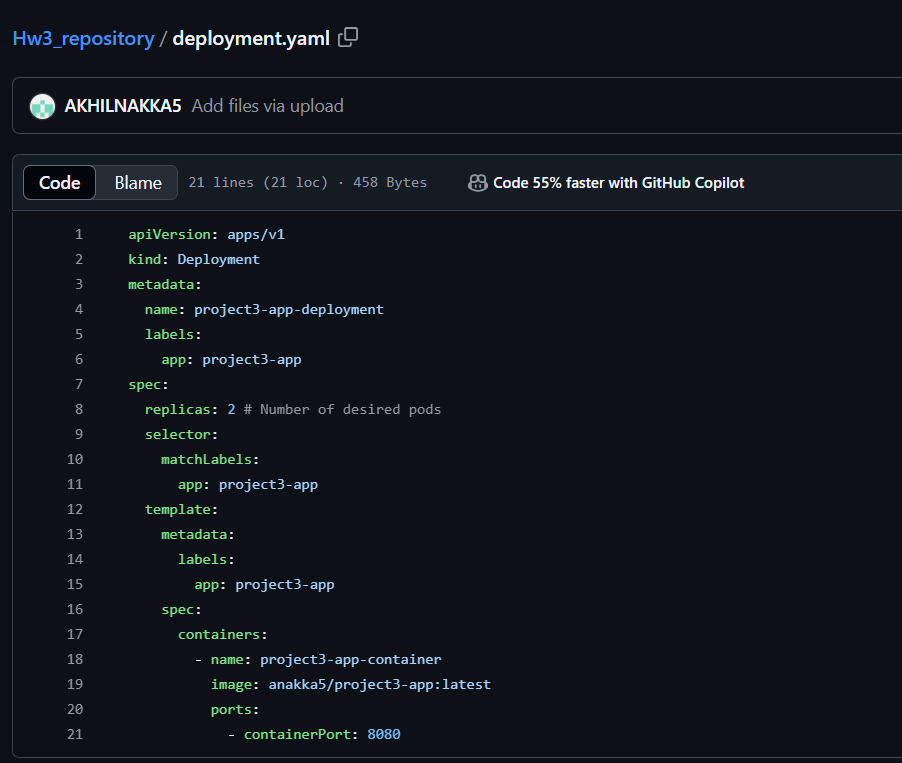


**Deploy the application to Kubernetes:**

* Create Kubernetes Deployment and Service Files
* Apply YAML Files by using

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

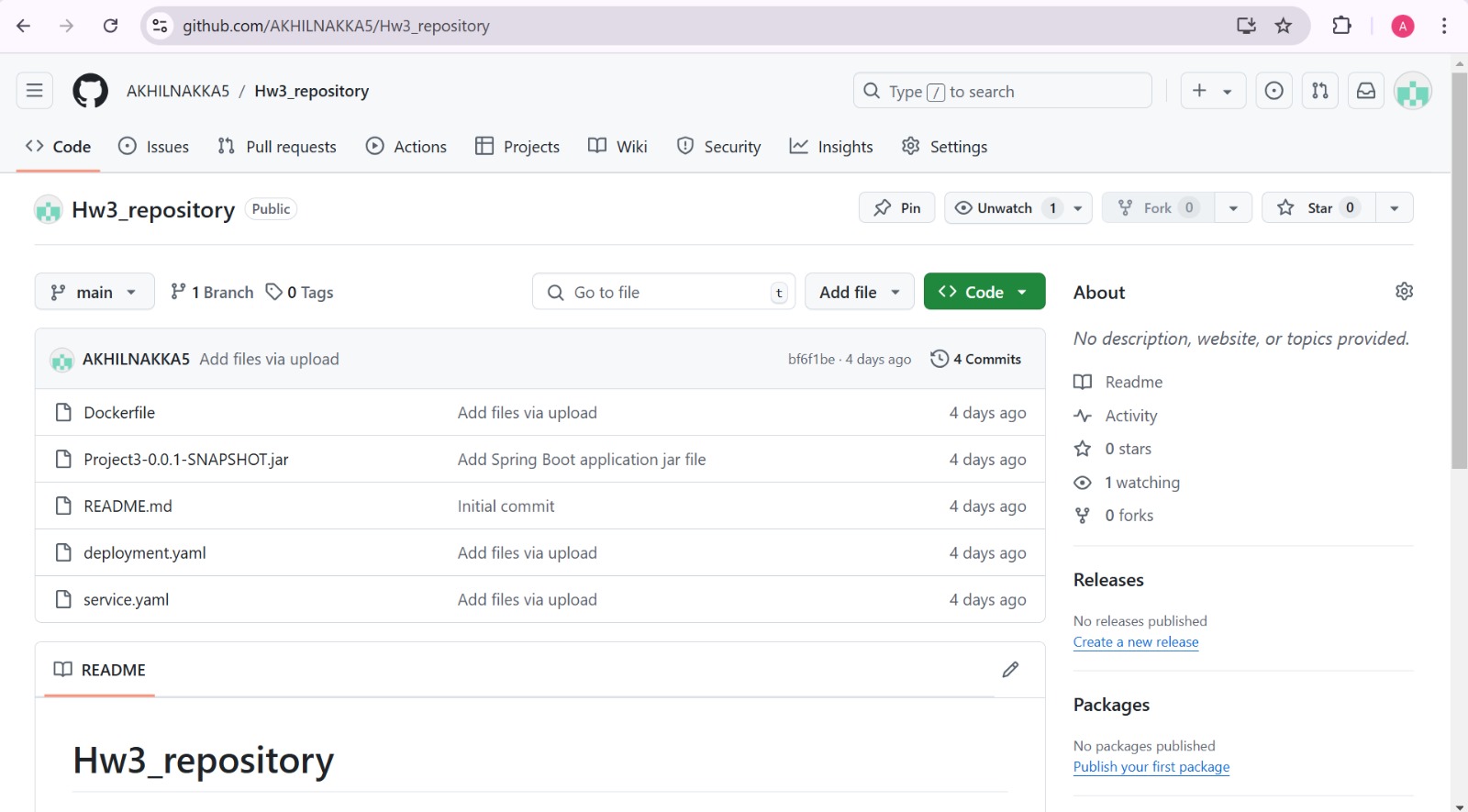


A screenshot of a computer program

Description automatically generated

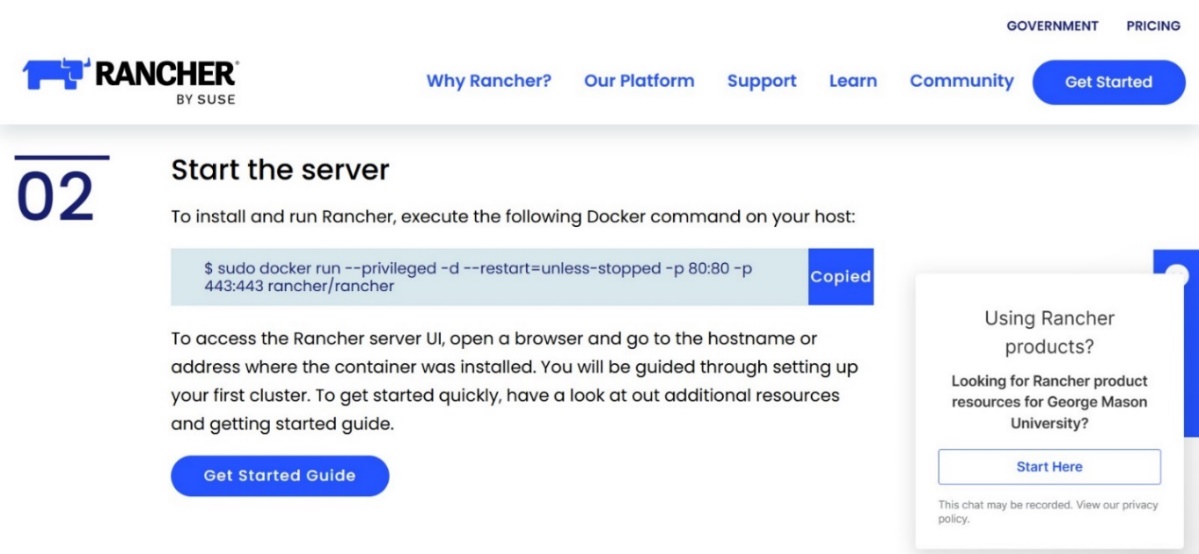
**Setting up the GitHub Repository:**

* We created the Project3\_645 in GitHub.



**Setting up Rancher:**

* Go to Rancher.io, Click on Get Started and click on Deploy Rancher.
* In the Start Server, there will be a URL, copy it to the first instance.



* To get the container ID, use the command
  + $ sudo docker ps
* Use the public IPv4 address od the EC2 instance to open the rancher
* We will retrieve the bootstrap password to login into Rancher
  + $ sudo docker logs container-id 2>&1 | grep “Bootstrap password:”
* Login to the rancher using that password.

A screenshot of a login screen

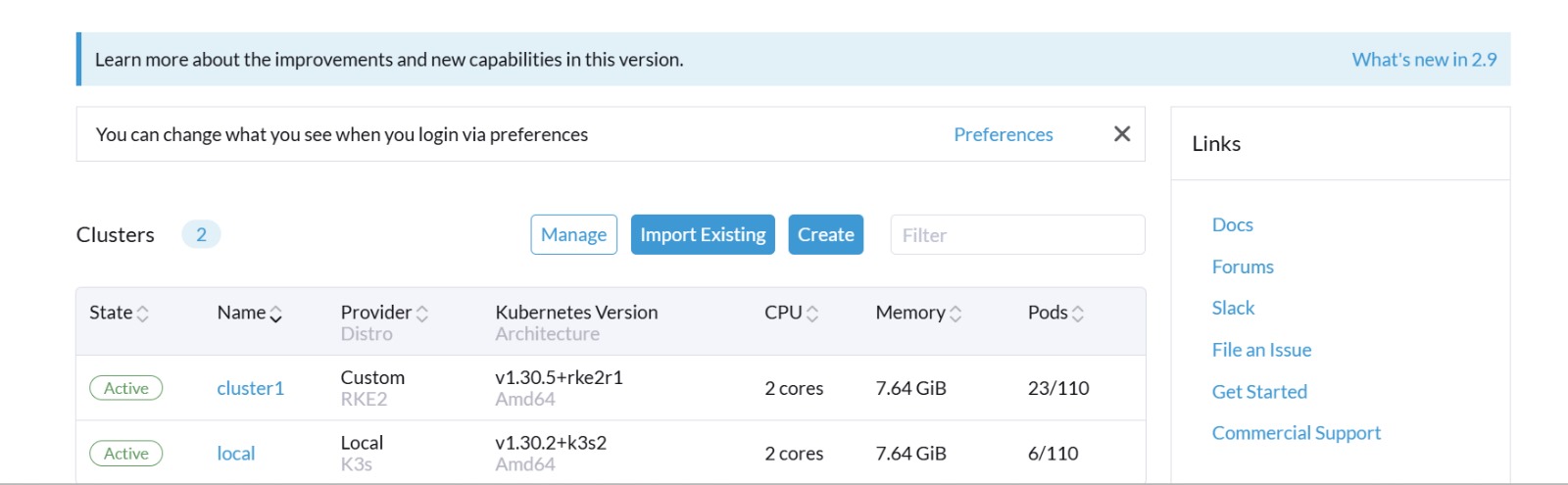
Description automatically generated

* Now, create the cluster using the Create button
* Select Custom, and enter the cluster name (Cluster1)

A screenshot of a computer

Description automatically generated

* Run the given command in the second instance and the instance will be active.



**Installing Jenkins:**

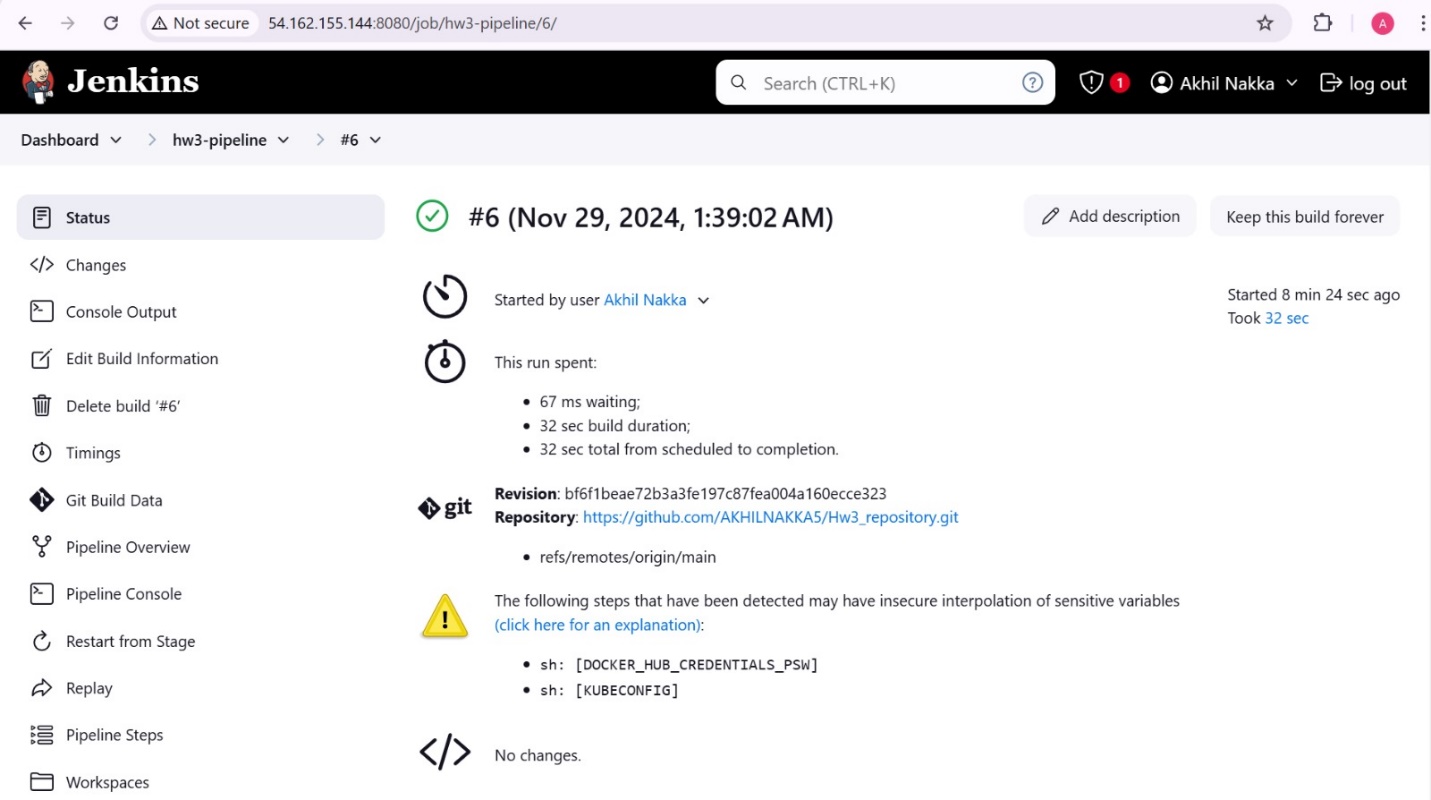
* Install the docker and Kubectl in the third instance, same as above steps
* Install Java 21
  + sudo apt update && sudo apt upgrade -y
  + sudo apt install openjdk-21-jdk -y
* Install Jenkins and start by using the commands
  + sudo apt install jenkins -y
  + sudo systemctl start jenkins
* Check the status of the Jenkins
  + systemctl status Jenkins
* open the browser on the public IPv4 address in port 8080

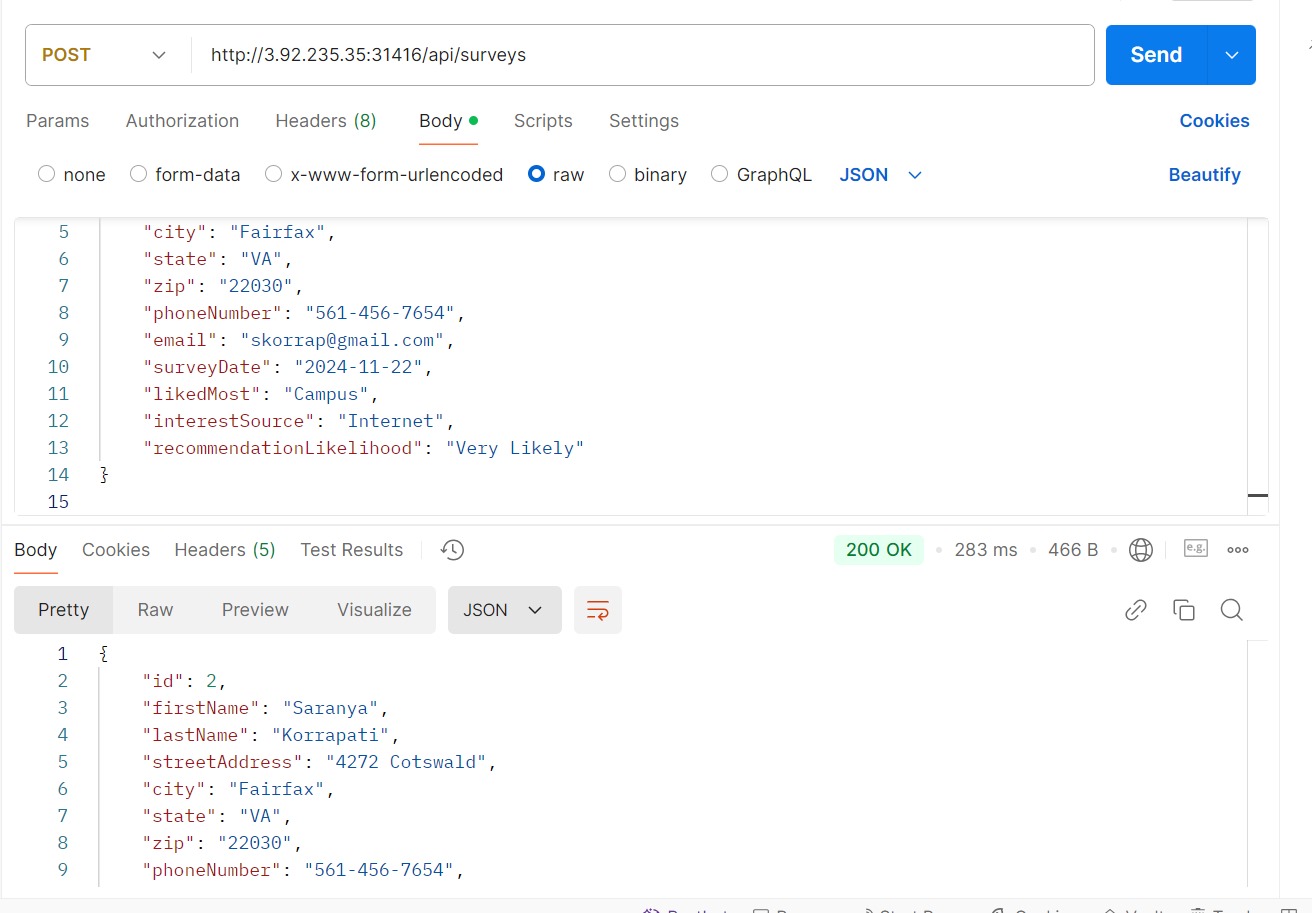
**Setting Up Jenkins:**

* A pipeline that generates the build using the Jenkins file and verifies any modifications to GitHub must be set up.
* In the Jenkins user interface, select "New Items," type a name, and then select "Pipeline."
* Every five minutes, a cron job will be put up to check our git repository for changes. If any are found, a new build will be started.

****

* Save with all other settings left at their default settings.
* Select Configure and Manage Plugins from the Manage Jenkins section. Put the Kubernetes, Docker, and Git plugins on your computer.
* Select credentials from the Manage Jenkins section. Enter the DockerHub username and password to add DockerHub credentials. In the same manner as before, add the git credentials. Choose the secret file option to add the Kubernetes configuration file.
* To build the pipeline, configure the Jenkins file.
* The Jenkins file receives the Kubernetes configuration file and the Docker Hub credentials as environment variables.
* Consequently, it generates the Docker image, tags it with the Jenkins build number, and fetches our Github repository first.
* It will then push the image after logging into DockerHub with the mentioned credentials.
* To deploy to the Kubernetes cluster, we apply the deployment and service yaml files defined in the root directory of the git hub repository using kubectl commands after giving the build number to the deployment yaml in the next step.
* We made some modifications to index.html in the repository and committed them to produce the build.
* The Jenkins pipeline will now use Docker to create an image, which will subsequently be pushed to Docker Hub.
* After that, it will deploy the image to the Kubernetes cluster after retrieving it from Docker Hub.





**Contributions**:

Saranya – Creating RDS, spring boot project, and connecting to database

Akhil – Creating a docker image, setting up EC2 instances, rancher, and Kubernetes cluster

Samuel – Setting up the Jenkins pipeline and validating CRUD operations on Postman