

SWE 645 – HW-3

Team Members:

Srikanth Reddy Dubba (G01353043) – Spring boot backend application and Rancher setup on AWS

Bhargava Canakapally (G01351583) – Angular application development

Sai Rohith Pasham (G01348426) – AWS MySQL RDS setup, Workbench, testing, and documentation.

Creating EC2 Instance in AWS Academy Learner Lab

- Log into the AWS Learner application. Traverse to the EC2 Instance page. Select Launch Instances.
- Search for the Ubuntu Server 20.24 Instance and select the first one: **Ubuntu Server 20.04 LTS (HVM), SSD Volume Type**.
- Write the name and tags for the instance. Select Instance type t3.large. Create a new key pair or you can use your key pair. The key pair .pem file will automatically save to the system once it is created.

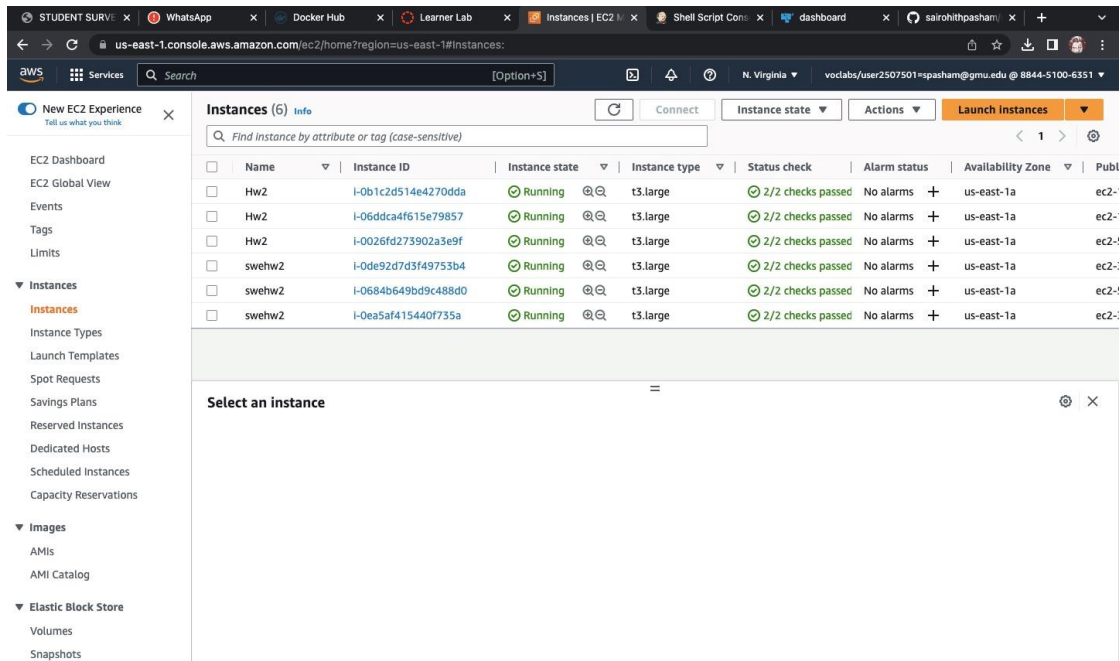
The screenshot displays the AWS Academy Learner Lab interface for creating an EC2 instance. It is divided into two main sections: 'AMI from catalog' and 'Instance type'.

AMI from catalog: This section shows the 'Amazon Machine Image (AMI)' selection process. It includes tabs for 'AMIs from catalog', 'Recents', and 'Quick Start'. The selected AMI is 'ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-20230328' with ID 'ami-0aa2b7722dc1b5612'. It is marked as 'Free tier eligible' and a 'Verified provider'. A table below lists the AMI details:

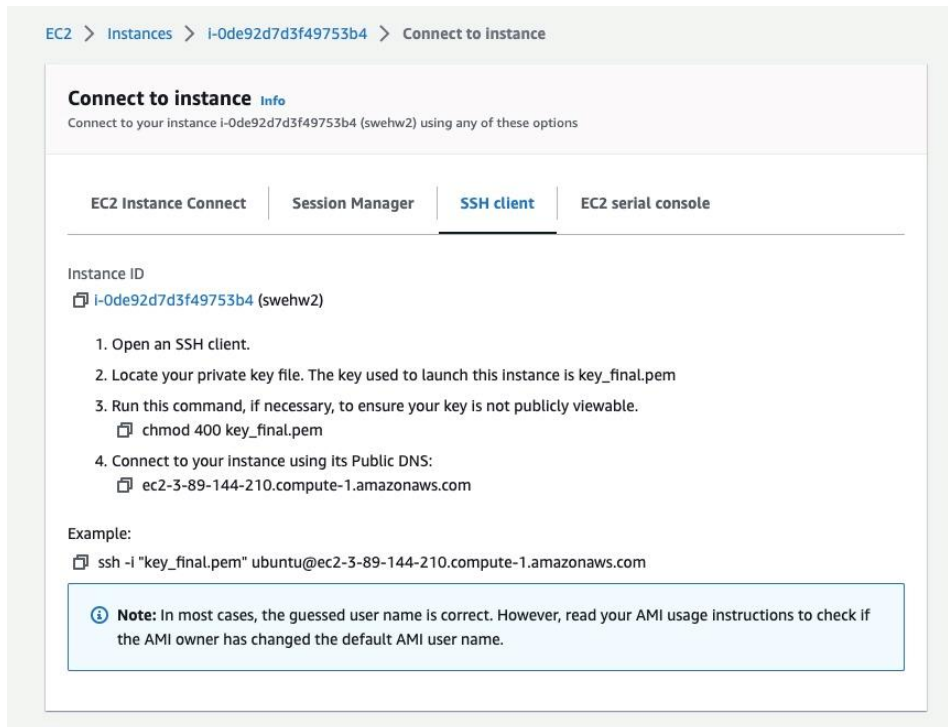
Catalog	Published	Architecture	Virtualization	Root device type	ENA Enabled
Quickstart AMIs	2023-03-28T23:35:33.00Z	x86_64	hvm	ebs	Yes

Instance type: This section shows the 'Instance type' configuration. The selected instance type is 't2.large'. It includes details such as 'Family: t2', '2 vCPU', '8 GiB Memory', and 'Current generation: true'. Pricing information is also displayed: 'On-Demand Windows pricing: 0.1208 USD per Hour', 'On-Demand RHEL pricing: 0.1528 USD per Hour', 'On-Demand SUSE pricing: 0.1928 USD per Hour', and 'On-Demand Linux pricing: 0.0928 USD per Hour'. A 'Compare instance types' link is available.

- Create a new security group and allow for traffic anywhere on ports 8080, 80, 443, and 22. Number of instances to 2, change the subnet to your region (us-east-1) and select the IAM role as LabInstanceProfile. Add storage with the size 16 since we have money in the student account. Click Launch Instances.
- It may take a few minutes for the various instances to be fully deployed and ready. But once they are, the Status check will be green.



- Now, connect both instances. Click that tab if you have SSH capabilities. To use the ssh command, copy and paste it into a PowerShell or terminal window.
- In your console, change the directory (cd) to where your pem file is downloaded too. Run chmod 400 on your pem file to tighten its security rules. Then paste that copied command into the line and hit enter to connect.



- cd </path/to/.pem file>
- chmod 400 name.pem 0
- ssh -i <pem file name> ubuntu@<ec2 instance address>
- In the console type yes for connecting.
- Now that we are in the EC2 instance, update the instance by typing: sudo apt-get update.
- Once it is finished updating, install docker within the instance. Type: sudo apt install docker.io

- Do the above few steps for both instances.

Rancher

- On the global Rancher page, click on 'Create.'
- You can choose any cloud provider you want; we chose Custom Cluster.
- Give your cluster a name. Next, the node options select etcd, control plane, and worker. Copy the commands to the second instance terminal.

Deploy the docker image to the rancher.

- In Rancher, click on the top left side and choose your cluster. Click on Projects/Namespaces. Create a project and a namespace to make it easier to locate your running pods.
- Click on your cluster and choose your project. Click on deployment and choose services. Click on load under the target, select three pods, and check whether the webpage is opening.

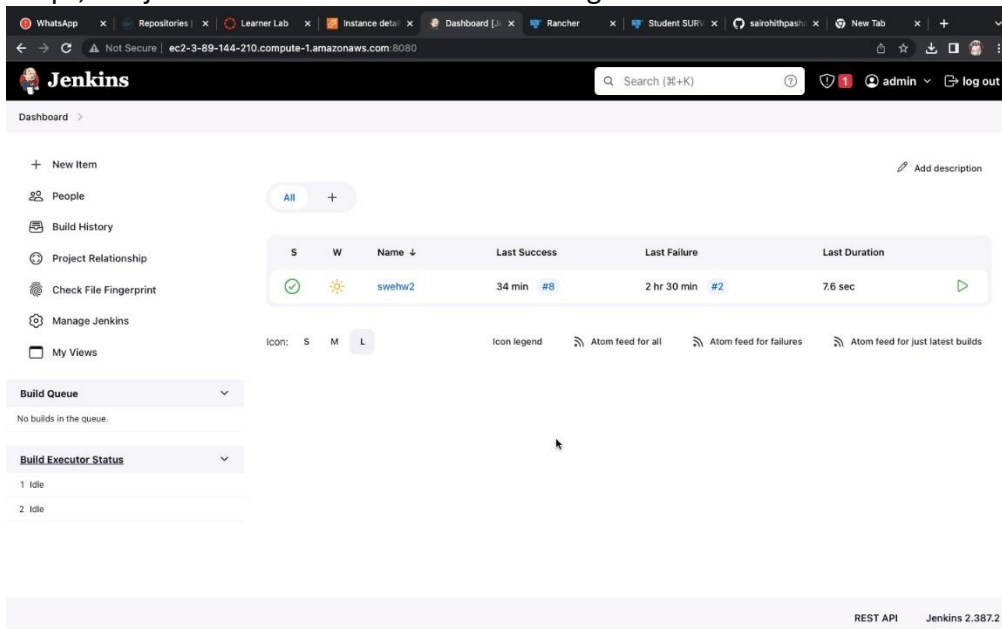
The screenshot shows the Rancher web interface for a deployment named 'deploy' in the 'default' namespace. The deployment is active and has 3 pods running. The pods are listed in a table with columns: State, Name, Image, Ready, Restarts, IP, Node, and Age. All three pods are in a 'Running' state and are using the 'saiohithpasham/swehw:0.1' image.

State	Name	Image	Ready	Restarts	IP	Node	Age
Running	deploy-6cd777d58c-2bvzv	saiohithpasham/swehw:0.1	1/1	0	10.42.0.26	ip-172-31-42-79	23 secs
Running	deploy-6cd777d58c-6x2c5	saiohithpasham/swehw:0.1	1/1	0	10.42.0.27	ip-172-31-42-79	23 secs
Running	deploy-6cd777d58c-f18r4	saiohithpasham/swehw:0.1	1/1	0	10.42.0.25	ip-172-31-42-79	23 secs

Jenkins

- Create another Ubuntu EC2 instance. Install Docker on Jenkins because we will use it later on jenkinsfile.
- Install Docker on Jenkins. JDK 11: 'sudo apt install openjdk-11-jdk'.
- Install Jenkins by running: 'sudo apt update' and then 'sudo apt install jenkins'.
- Check whether Jenkins is installed or not, we run 'systemctl status jenkins' and check if it's active or not.
- As we login into our Jenkins, we use the commands below to enable the kubectl connection with our rancher cluster.

- --\$ cd var/lib/jenkins
- --\$ >. kube (creating. kube directory)
- --\$ > config (creating config file)
- --\$ cat > config [press enter and past the copied kubeconfig file content then press ctrl+d]
- --\$ vi config (to view the config file contents)
- Now that we created and pasted our kubeconfig file, we verify it by running the command 'kubectl config current-context'.
- We now go back to the instance and click on the http://our_ip_or_domain:8080 to access our Jenkins dashboard. Get the password on the terminal using the cat command.
-- \$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
- Copy the password from the terminal, paste it into the "Administrator password" field, and click "Continue". On the next page, click on the "Install suggested plugins" box, and the installation process will start immediately.
- Once plugins are installed, you will be prompted to set up the first admin user. Fill in all the required information and continue.
- The following page will prompt you to enter the URL for your Jenkins instance. - An automatically generated URL will be entered into the field.
- After you confirm the URL by clicking the Save and Finish button, the setup process is complete. – When you click the Start using Jenkins button, you will be redirected to the Jenkins dashboard, where you will be logged in as the admin user you created in one of the previous steps. This indicates that Jenkins was successfully installed on our server.
- In the dashboard, select a new item, give it a name, and pipeline, and then OK. Next, in Jenkins' configuration, we select general->select GitHub project and enter the URL of our GitHub, then select build trigger->select poll SCM and enter '* * * * *' in the description, instructing the Jenkins file to build every one minute.
- Now, in SCM, pipeline->definition->pipeline description: we provided the GitHub URL as well as the script, the jenkinsfile. We leave the remaining fields alone and click Save.



The screenshot shows the Jenkins Dashboard interface. At the top, there's a navigation bar with the Jenkins logo, a search bar, and a user profile dropdown showing 'admin' and a 'log out' button. Below the navigation bar, the main content area displays a table of builds. The table has columns for status (S), warning (W), name, last success, last failure, and last duration. A single build named 'swehw2' is listed with a success status, a warning icon, and a duration of 7.6 sec. To the left of the table, there's a sidebar with various links like 'New Item', 'People', 'Build History', etc. At the bottom, there's a 'Build Queue' section showing 'No builds in the queue' and a 'Build Executor Status' section showing two idle executors.

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	⚠	swehw2	34 min #8	2 hr 30 min #2	7.6 sec

Build Queue: No builds in the queue.

Build Executor Status:

- 1 Idle
- 2 Idle

REST API Jenkins 2.387.2

```

1 pipeline {
2   environment {
3     DOCKERHUB_CREDENTIALS = credentials('dockerid')
4     TIMESTAMP = now Date().format("yyyyMM-dd_HH:mm:ss")
5   }
6   agent any
7
8   stages {
9     stage('Build Image') {
10      steps {
11        script {
12          sh "rm -rf s-war"
13          sh "jar -cvf 645hw2.war -C src/main/webapp/ ."
14          //sh "echo ${BUILD_TIMESTAMP}"
15          sh "docker build -t sairohithashan/studentsurvey:${env.TIMESTAMP} ."
16          sh "echo ${DOCKERHUB_CREDENTIALS_PASSWORD} | docker login --password-stdin"
17        }
18      }
19    }
20
21    stage('Push Image') {
22      steps {
23        script {
24          sh "docker push sairohithashan/studentsurvey:${env.TIMESTAMP}"
25        }
26      }
27    }
28
29    stage('Update Deployment') {
30      steps {
31        script {
32          sh "kubectl set image deployment/nw container=saiprohithashan/studentsurvey:${env.TIMESTAMP}"
33        }
34      }
35    }
36
37    stage('Update LoadBalancer') {
38      steps {
39        script {
40          sh "kubectl rollout restart deploy hw --default"
41        }
42      }
43    }
44  }
45
46  post {
47    always {
48      sh "docker logout"
49    }
50  }
51 }

```

-
-
- Go to the GitHub repository, select Settings from the left-hand side click webhook, and select add webhook, give the ipv4 DNS of the third instance along with the port number, and at the end add github-webhook. Change content type to application/json.
- Now, in Jenkins, select our newly created pipeline and press the 'build now' button. After a successful build, we can see that our new image has been pushed to dockerhub and then to our rancher cluster. If we make any changes to our HTML page, the new image is automatically created, then pushed to dockerhub and updated, and can be viewed from the cluster rancher endpoints.

The screenshot shows the Jenkins web interface for a pipeline named 'swehw2'. On the left is a sidebar with navigation options like Status, Changes, Build Now, Configure, etc. The main area displays the 'Stage View' for the pipeline, which includes a table of build history.

	Declarative: Checkout SCM	Build Image	Push Image	Update Deployment	Update LoadBalancer	Declarative: Post Actions
Average stage times: (Average full run time: ~12s)	435ms	6s	3s	774ms	611ms	346ms
#8 Apr 13 17:47 1 commit	258ms	2s	2s	474ms	418ms	331ms
#7 Apr 13 17:46 No Changes	240ms	1s	9s	415ms	413ms	331ms
#6 Apr 13 16:54 No Changes	392ms	2s	2s	723ms	739ms	346ms
#5 Apr 13 15:53 1 commit	328ms	1s	2s	678ms	725ms	336ms

-
- Now that our CI/CD pipeline is set, we can commit changes to the HTML file present in the src/main/webapp folder in the GitHub repo.
- The changes will be now automatically reflected in the web application.

Environment Setup:

Development Tools:

1. Visual Studio Code
2. Eclipse EE
3. JPA - Spring boot
 - Visual Studio Code, which is used to develop Angular.
 - Spring-Boot and Eclipse EE for creating RESTful Web Services.
 - With JPA, we connected to the My SQL server using the My SQL Workbench.

Installation:

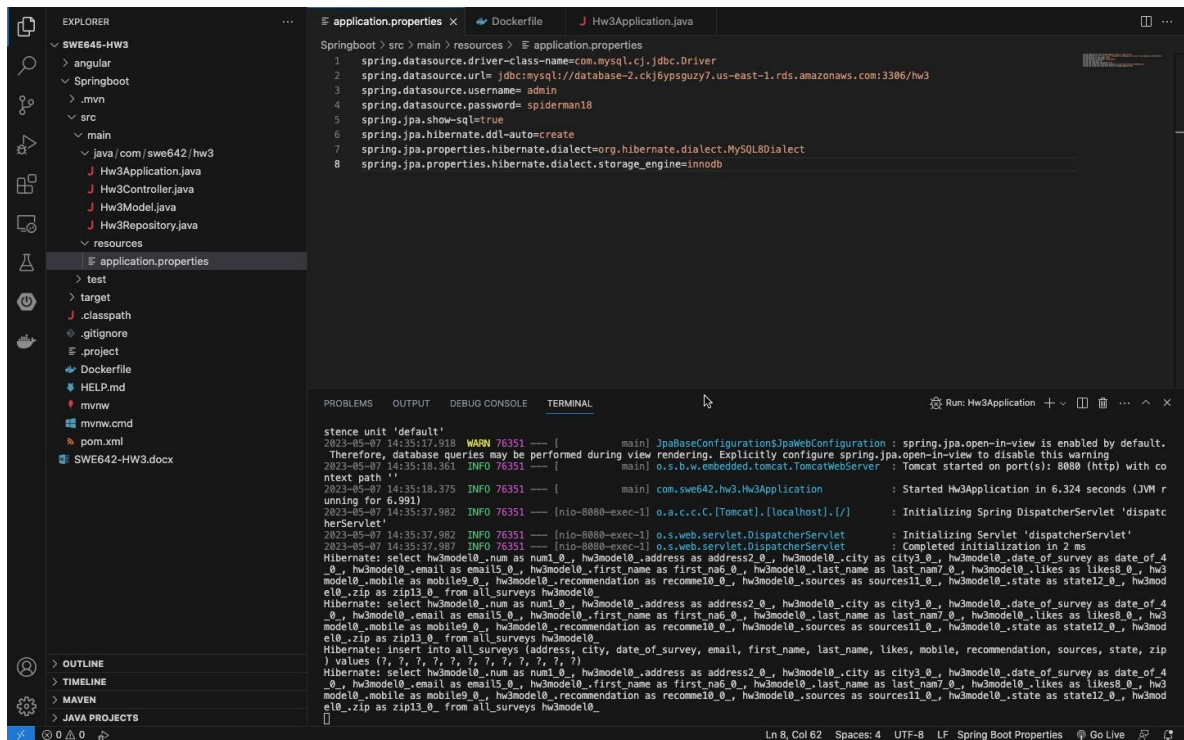
Angular:

1. Firstly, install Node.js from nodejs.org
2. Secondly, install the angular client.
npm install -g@angular/cli
3. Then, create a new angular project.
ng new angular
4. Next, install Bootstrap styling.
npm install bootstrap
5. After that, install the component for the home
ng generate the component home
6. Install the component for the survey.
ng generate component survey
7. Install components for all surveys.
ng generate component allsurveys
8. Install rest API call service.
ng generate service recall
9. Finally, execute **ng serve** to launch a dev server.
Navigate to <http://localhost:8080>. Any changes to the files are automatically reflected.

Spring boot:

1. Create a spring boot application, navigate to <https://start.spring.io/> fill in all the details as mentioned below.
Add the following dependencies:
 - Lombok
 - Spring Web
 - MySQL Driver
 - Spring Data JPA

Then, we create the following packages in the folder:



Maven deployment

1. run the **mvn clean package**.
2. By selecting Run -> Run as -> Java application, you can run Application.java on the server. In the console output, we can see that Tomcat is executing on the 8080 ports.

Steps to set up MySQL RDS:

Here are the steps to create an RDS instance in the AWS Management Console using MySQL:

1. Log into the AWS Management Console and select RDS from the list of AWS services.
2. Click on the "Launch DB instance" button.
3. Select MySQL on the "Select engine" page and click the "Select" button.
4. Select the "Dev/Test" option on the next page to use the AWS free tier.
5. In the following step, choose "db.t2.micro" with 1 vCPU and 1GiB RAM as the instance class to use the free tier database.
6. After selecting the free tier database, please provide a username to identify it and set the access credentials, such as username and password.
7. To configure the security of RDS, you can use Amazon's security groups on the VPC. On the advanced settings page, create a new security group instead of relying on an older one that may or may not have been set up.
8. For the database name, copy the DB Instance Identifier.
9. After clicking the "Launch" button, you will be redirected to a page where the DB instance is created and configured for access to RDS. At the bottom of the page, click on "View Your DB Instances."
10. After some time, you will see the successful creation of the MySQL RDS instance.

The screenshot shows the Amazon RDS console for a database instance named 'database-2'. The instance is in the 'Available' state and is a MySQL Community Edition instance. The console displays various tabs for managing the instance, including Summary, Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance & backups, and Tags. The 'Connectivity & security' tab is selected, showing details about the endpoint, port, networking, and security groups.

Summary			
DB Identifier	database-2	CPU	3.19%
Role	Instance	Current activity	12 Connections
Status	Available	Engine	MySQL Community
Class	db.t3.micro	Region & AZ	us-east-1a

Connectivity & security		
Endpoint & port	Networking	Security
Endpoint	Availability Zone	VPC security groups
database-2.ckj6ypsguzy7.us-east-1.rds.amazonaws.com	us-east-1a	security_group (sg-0c6b027a041bc943b)
Port	VPC	Active
3306	vpc-0a209ace091aa7b98	Publicly accessible
	Subnet group	Yes
	default-vpc-0a209ace091aa7b98	Certificate authority

MySQL Workbench

My SQL Workbench is the official graphical user interface (GUI) tool for MySQL.

To connect the MySQL server hosted on Amazon, we need to provide an Amazon RDS MySQL endpoint in place of the hostname as mentioned below.

Spring-boot:

1. Create a Docker Hub account.(hub.docker.com)
2. Create two Docker files in the account—one for Angular and the other for Spring Boot. Our package.json file is present here, and a spring boot docker file is created.

The screenshot shows an IDE with the Spring Boot application code and the Dockerfile. The code is for a Spring Boot application that uses JPA and Hibernate. The Dockerfile is a multi-stage build that uses the 'openjdk:17' base image and copies the application code to the target directory. The terminal shows the output of the application, including the initialization of the Spring DispatcherServlet and the execution of the application.

```

1 FROM openjdk:17
2 CMD ["mvn clean install"]
3 COPY target/hw3-0.0.1.jar .
4 EXPOSE 8080
5 CMD ["java", "-jar", "hw3-0.0.1.jar"]
6

```

```

stence unit 'default'
2023-05-07 14:35:17.918 WARN 76351 --- [main] JpaBaseConfigurationsJpaWebConfiguration : spring.jpa.open-in-view is enabled by default.
Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2023-05-07 14:35:18.361 INFO 76351 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with co
ntext path ''
2023-05-07 14:35:18.375 INFO 76351 --- [main] com.swe642.hw3.Hw3Application : Started Hw3Application in 6.324 seconds (JVM r
unning for 6.991s)
2023-05-07 14:35:37.982 INFO 76351 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatc
herServlet'
2023-05-07 14:35:37.982 INFO 76351 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-05-07 14:35:37.987 INFO 76351 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
Hibernate: select hw3model0_num as num1_0, hw3model0_address as address2_0, hw3model0_city as city3_0, hw3model0_date_of_survey as date_of_4
_0, hw3model0_email as email5_0, hw3model0_first_name as first_name6_0, hw3model0_last_name as last_name7_0, hw3model0_likes as likes8_0, hw3
model0_mobile as mobile9_0, hw3model0_recommendation as recomme10_0, hw3model0_sources as sources11_0, hw3model0_state as state12_0, hw3mod
el0_zip as zip13_0 from all_surveys hw3model0
Hibernate: select hw3model0_num as num1_0, hw3model0_address as address2_0, hw3model0_city as city3_0, hw3model0_date_of_survey as date_of_4
_0, hw3model0_email as email5_0, hw3model0_first_name as first_name6_0, hw3model0_last_name as last_name7_0, hw3model0_likes as likes8_0, hw3
model0_mobile as mobile9_0, hw3model0_recommendation as recomme10_0, hw3model0_sources as sources11_0, hw3model0_state as state12_0, hw3mod
el0_zip as zip13_0 from all_surveys hw3model0
Hibernate: insert into all_surveys (address, city, date_of_survey, email, first_name, last_name, likes, mobile, recommendation, sources, state, zip
) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: select hw3model0_num as num1_0, hw3model0_address as address2_0, hw3model0_city as city3_0, hw3model0_date_of_survey as date_of_4
_0, hw3model0_email as email5_0, hw3model0_first_name as first_name6_0, hw3model0_last_name as last_name7_0, hw3model0_likes as likes8_0, hw3
model0_mobile as mobile9_0, hw3model0_recommendation as recomme10_0, hw3model0_sources as sources11_0, hw3model0_state as state12_0, hw3mod
el0_zip as zip13_0 from all_surveys hw3model0

```


3. Spring boot image can be built by using:

docker build -t spring-boot:v1.

4. To run the angular image we use:

docker run --rm -d -p 8080:8080 spring-boot:v1

5. Before pushing the image to the docker hub, we need to tag the image with our username:

docker tag spring-boot:v1 sairohithpasham/spring:1.0

6. To view information about currently running docker containers in our docker framework, we use the command:

docker ps

7. Finally, push the image to the docker hub using:

docker push sairohithpasham/spring:1.0

For Angular:

1. In the directory where the package.json file is located, create a Docker file of spring-boot.

2. To build an Angular image, we use the following command:

docker build -t angular:v8

3. To run the angular image, use the following command:

docker run --rm -d -p 8080:8080 angular:v8

4. Before pushing the Angular image to the docker hub, it needs to be tagged with our username:

docker tag angular:v8 sairohithpasham/angular:v8

5. To view the information about currently available docker containers in our docker framework:

docker ps

6. Next, we push the image to the docker hub using:

docker push sairohithpasham/angular:v8

Steps to set up a Rancher:

- To set up the rancher on the instance we created earlier we run the following docker command:
docker run -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher
- To access the Rancher UI, we use the public IPv4 DNS obtained from the previous steps. Once the rancher UI is up and running, we can create a password for the admin user by running the following command: **docker logs**
"container-id" 2>&1 | grep "Bootstrap Password:"

We will be directed to Rancher Console after a valid authenticated password is provided.

Steps to Create Clusters Using Rancher UI:

Here are the steps to add a cluster in the Rancher console:

1. In the top right corner of the Rancher console, click on "Add Cluster."
2. A menu with different options to create a cluster will appear. Select "Custom."
3. Provide a name for your cluster in the "Custom" field, and you can leave the rest of the options as default. Then click on "Next."

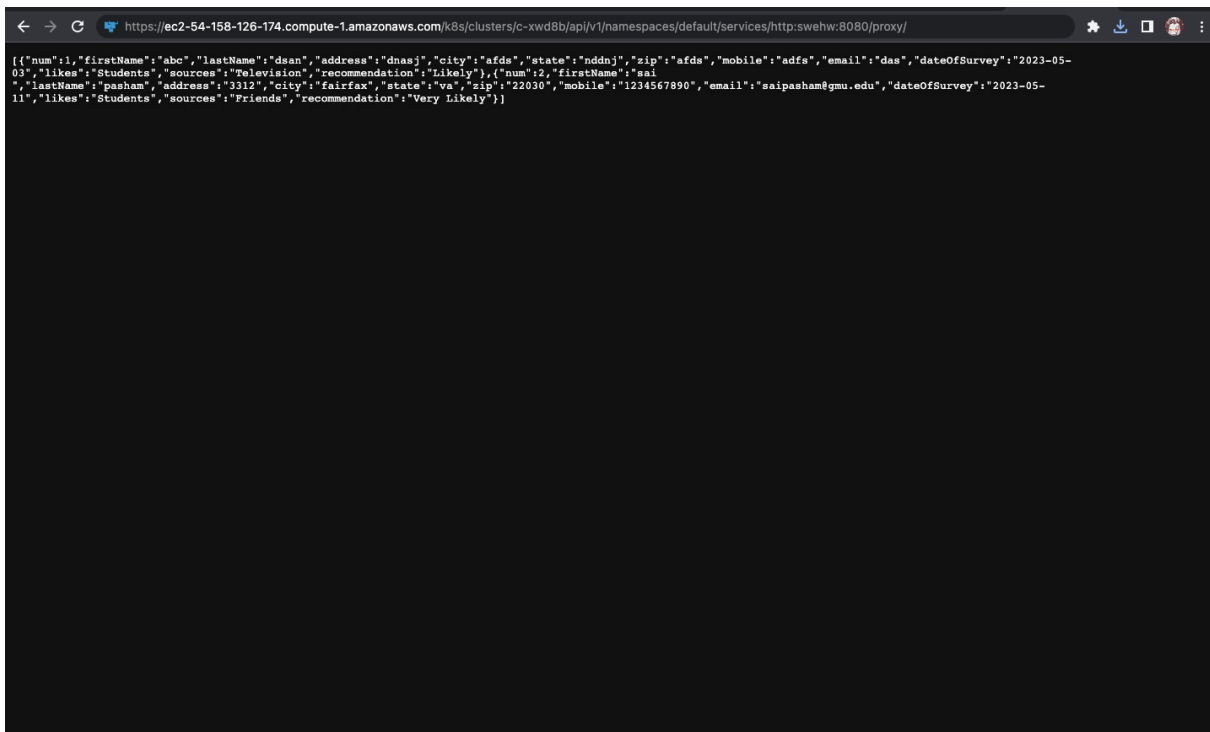
4. On the next page, you will see options to choose the nodes' roles in the cluster. Ensure all three checkboxes are checked: etcd, control plane, and worker.
5. Copy the command provided for the instance created on your machine, and then click "Done."
6. Congratulations! You have successfully created two clusters, one for Angular and the other for Spring Boot.

Deployment of angular and spring-boot:

Now deployment is completed for both the clusters that were created earlier as we can observe in the below screenshots:

We can access our application by pulling the docker images from the hub.

Screenshots of output:



The screenshot shows a web browser window with the address bar displaying the URL: `https://ec2-54-158-126-174.compute-1.amazonaws.com/k8s/clusters/c-xwd8b/api/v1/namespaces/default/services/http.swehw:8080/proxy/`. The main content area of the browser displays a JSON array of two user objects. The first object has a numeric ID of 1, a first name of 'abc', a last name of 'dsan', an address of 'dnasj', a city of 'afds', a state of 'nddnj', a zip of 'afds', a mobile number of 'adfs', an email of 'das', and a date of survey of '2023-05-03'. The second object has a numeric ID of 2, a first name of 'sai', a last name of 'pasham', an address of '3312', a city of 'fairfax', a state of 'va', a zip of '22030', a mobile number of '1234567890', an email of 'saipasham@gmu.edu', and a date of survey of '2023-05-11'. Both objects include 'likes' and 'sources' fields.

```
[{"num":1,"firstName":"abc","lastName":"dsan","address":"dnasj","city":"afds","state":"nddnj","zip":"afds","mobile":"adfs","email":"das","dateOfSurvey":"2023-05-03","likes":"Students","sources":"Television","recommendation":"Likely"}, {"num":2,"firstName":"sai","lastName":"pasham","address":"3312","city":"fairfax","state":"va","zip":"22030","mobile":"1234567890","email":"saipasham@gmu.edu","dateOfSurvey":"2023-05-11","likes":"Students","sources":"Friends","recommendation":"Very Likely"}]
```

GitHub link: <https://github.com/sairohithpasham/hw3>