

README 645 HW3

Group Members:

Srini Jammula – G01426257 (sjammul3)
Tejeswar Sadanandan – G01451353 (tsadanan)
Aditya Gottipati – G01459535 (agottipa)
Sruthi Sivasamy – G01453460 (ssivasa)

Survey API Url- <http://ec2-52-45-191-29.compute-1.amazonaws.com:30448/api/surveys>

Github Repository Url- <https://github.com/srinijammula/swe645asg3>

RDS Url- jdbc:mysql://rdshw3demo.c7scmo2s80q1.us-east-1.rds.amazonaws.com:3306/dbname

Rancher Public IPv4 DNS- <http://ec2-3-218-63-246.compute-1.amazonaws.com/dashboard>

Credentials username- admin, password- University@123

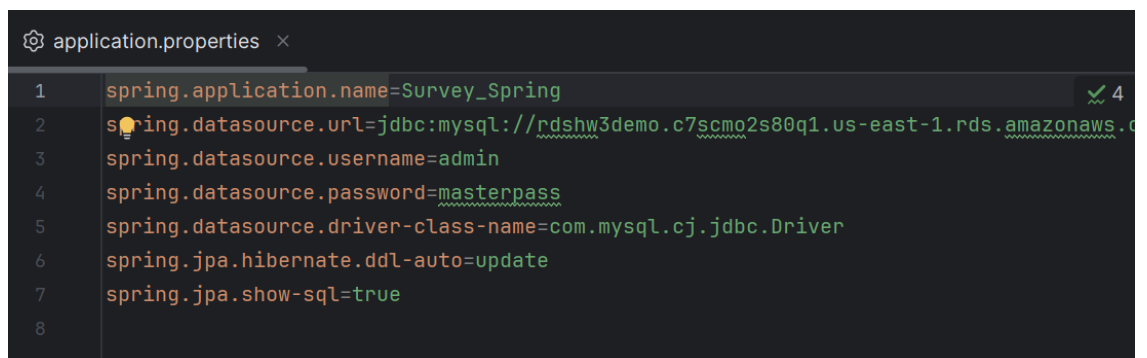
Jenkins Public IPv4 DNS- <http://ec2-54-159-234-5.compute-1.amazonaws.com:8080/>

Credentials username- srinijammula, password- University@123

• SPRING BOOT

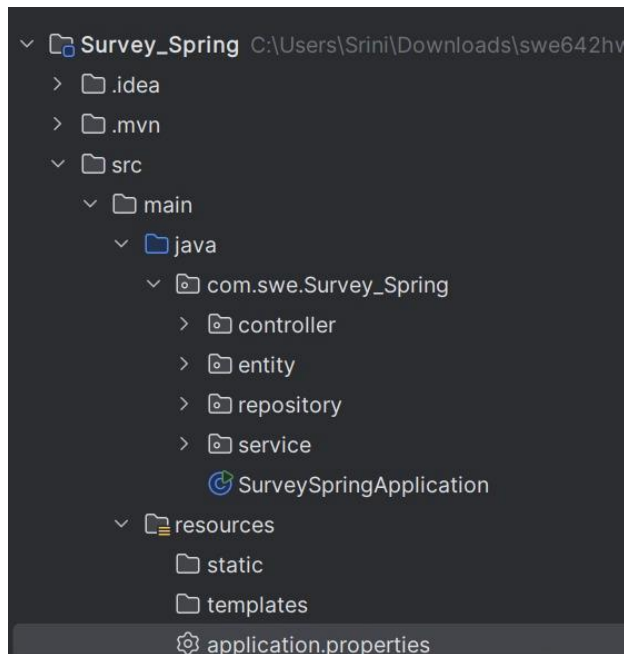
○ Setting up Spring boot.

- Create an enterprise-level spring application from Spring Initializer by going to the website <https://start.spring.io/>
- Select maven and keep the language in Java and the spring boot version be 3.1.4 in project Metadata set a name for the group and artifact in our case we gave it Survey_Spring and set the Packaging to Jar and Java to 17.
- Add some dependencies we need to add “Spring Web” “Lombok” “Spring Data JPA” “MySQL Driver”
- Then click on Generate and a folder will be downloaded to your working laptop, extract the folder and open the folder in IntelliJ IDE
- In that folder, we will see a resources folder in which there will be application properties where we will write configurations to connect our spring boot application to our RDS database. Create your RDS in AWS before this step.

A screenshot of the 'application.properties' file in an IntelliJ IDE. The file contains the following properties:

```
1 spring.application.name=Survey_Spring
2 spring.datasource.url=jdbc:mysql://rdshw3demo.c7scmo2s80q1.us-east-1.rds.amazonaws.com:3306/dbname
3 spring.datasource.username=admin
4 spring.datasource.password=masterpass
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6 spring.jpa.hibernate.ddl-auto=update
7 spring.jpa.show-sql=true
8
```

- In IntelliJ IDE In our main -> Java -> com.swe.Survey_Spring folder we are creating a few sub packages namely “controller”, “entity”, “repository”, and “service”.



○ Post survey API in Spring Boot

- In IntelliJ we have created a few sub packages, in the “entity” package we are creating a java class named Survey.java and we will be annotating it with @Data, @Entity, @Table(name of the table) this annotation “@Entity” is used to make this class as a table and once we run it, it will create the table in the database “survey.db”.
- Before running the application we need to create an interface SurveyRepository.java in the repository package, which extends JPA Repository to take arguments SurveyClass and Data type of ID, to implement CRUD operations in the application, and another java class in the service package namely SurveyService.java we will annotate it with “@Service” which will act as a service for the application.
- In the controller package, create a java class namely SurveyController.java and annotate it with “@RestController”, “@RequestMapping(“/API””, “@RequiredArgsConstructor”
- After this when you run the application, you can check to see if the table has been created or not.
- To check this we can use the API from Postman we need to enter the URL of our backend application and then enter the URL of our API and then attach the body in the JSON format of our form data and then click on send.

○ Get All surveys API in Spring Boot

- First, we will create an API to get all the surveys from Spring Boot and then we will check the same API from Postman. We have our class SurveyService where we will add a new method named getAllSurveys() that will return a list of all the surveys.
- Here, we will use the same surveyRepository instance to call the findAll() method that will return all records from our database.
- Next, we will call this method from our SurveyController. So we will create another method here called getAllSurveys() with List as a return type. This method will invoke the getAllSurveys() method that we wrote in our surveyService.
- We will add a ‘@GetMapping’ annotation with this method, where we add the URL of this API ‘/surveys’.

- To make sure that the API is running from the backend, we will then test this API from the backend by creating a GET request with the URL of this application. We will be able to get a response with all the records from the database.
 - Now, to implement this API, we will create a method called `getAllSurveys()` in the `survey.service.ts` where we have to pass the URL of this API which is `‘/API/surveys’`.
- **Update Survey**
 - So now when we get the ID from the URL then we need to get the API from our backend application we can do that by adding the method in our `Survey.service.ts`, and annotate with `‘@PutMapping’` which should receive the number i.e. ID of the certain user data.
 - **Delete Survey**
 - For the backend part, we will create a new method in our survey service called `deletesurvey()` to invoke the `deleteByID()` method. This method will be invoked by another method in our controller, which we will name `deleteSurvey()` and we will annotate this method by `‘@DeleteMapping’` and we will give the required API URL along with it and this particular record will be deleted from the table.

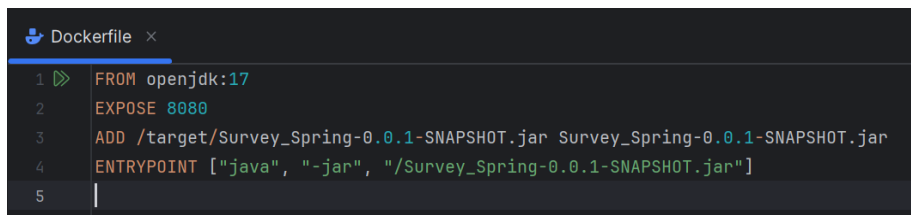
• DEPLOYING APPLICATION TO AWS CLOUD

1. Create a jar file

Install maven in your system and execute `mvn clean install` command in terminal of IntelliJ, and your jar file will be created under target folder.

2. Create Dockerfile

Create a file with name `‘Dockerfile’`. It will automatically give a docker symbol beside. Now paste this in your Dockerfile.



```

1 FROM openjdk:17
2 EXPOSE 8080
3 ADD /target/Survey_Spring-0.0.1-SNAPSHOT.jar Survey_Spring-0.0.1-SNAPSHOT.jar
4 ENTRYPOINT ["java", "-jar", "/Survey_Spring-0.0.1-SNAPSHOT.jar"]
5

```

This file is used to build your image from jdk and your jar file.

3. Check your image in localhost

This step is to verify if you can build an image correctly on given Dockerfile. To see if your image is working in local host, install docker desktop and open it for the Docker Engine to run. Check if docker is installed with command `‘docker --version’`. Now open cmd at your project folder and run below commands.

To build an image,

`docker build -t image-name .`

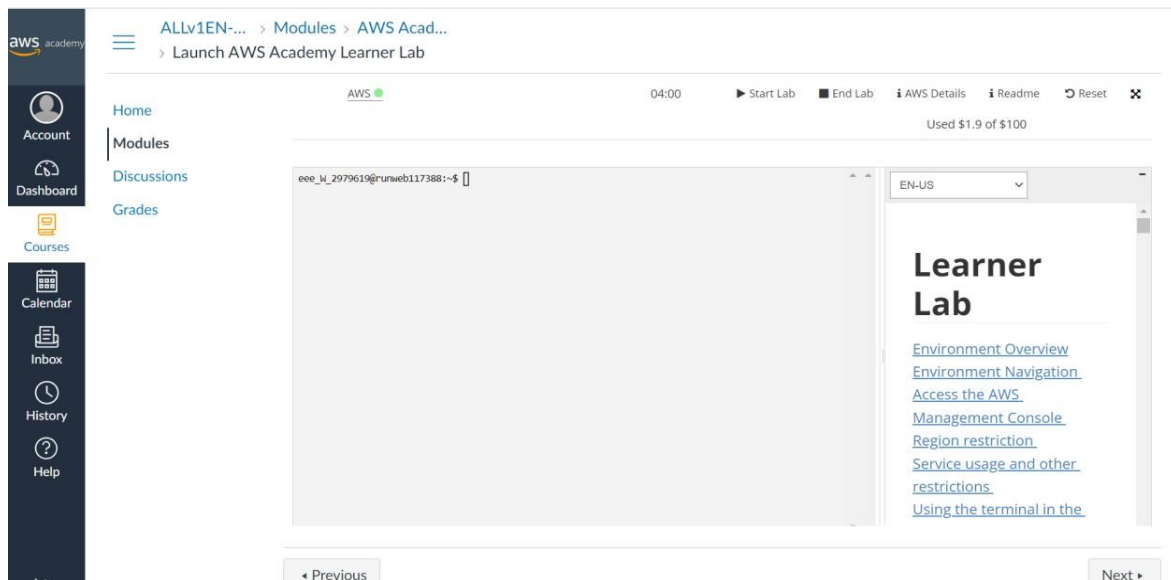
To create a container,

`docker run -d -it -p 8080:8080 image-name`

Check if you can see your survey list in <http://localhost:8080/your-api>

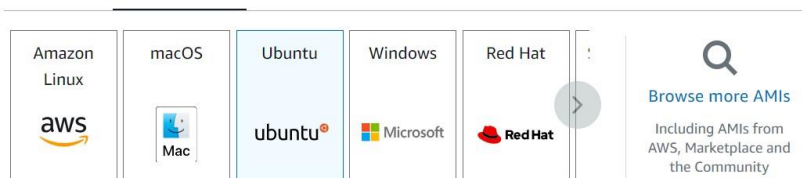
4. Create EC2 instances

Launch your AWS Lab. Got to modules and launch AWS Academy learner lab, start lab. After you can see a green dot near AWS click on it, it will redirect you to AWS console.

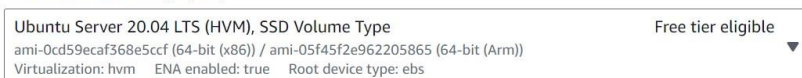


Search for EC2->Launch instance. Create three different instances with same configurations mentioned below. (asg3, clusterasg3, jenkinsasg3)

For AMI choose Ubuntu 20.04 LTS,



Amazon Machine Image (AMI)

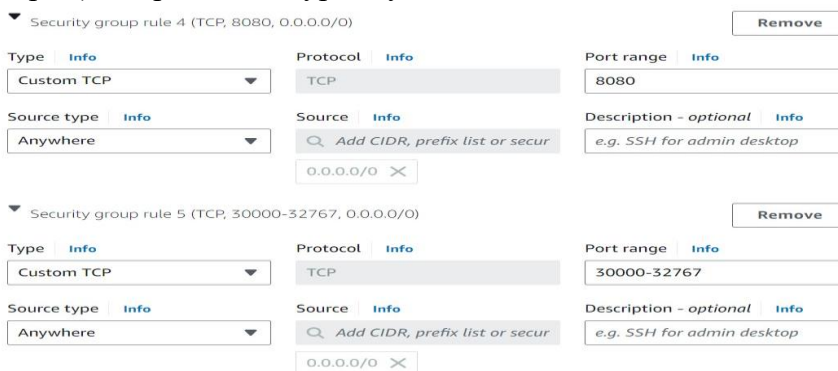


Description

Instance type- t2.large

Create a new key-pair or use an existing one (I am using demo1.pem).

In network settings check on all allow and click on edit to add port 8080 and 30000-32767 (for nodeport) and put source type anywhere.



Configure storage to 30gb and Launch instance.

Instances (3) Info							Launch instances	
<input type="text" value="Find Instance by attribute or tag (case-sensitive)"/>				All states		< 1 >		
<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm st		
<input type="checkbox"/>	jenkinsasg3	i-05e0f0f28268593a0	Running	t2.large	2/2 checks passed	View alar		
<input type="checkbox"/>	clusterasg3	i-0f9cb82d080a2d9f3	Running	t2.large	2/2 checks passed	View alar		
<input type="checkbox"/>	asg3	i-02cf6376cb06b9ce5	Running	t2.large	2/2 checks passed	View alar		

Search for elastic ips in ec2 and allocate each for each instance. Just select instance name and default settings. Give names to IP.

<input type="checkbox"/>	Name	Allocated IPv4 addr...	Type	Allocation ID
<input type="checkbox"/>	elastic1	3.218.63.246	Public IP	eipalloc-05afcc0e14d0b51f6
<input type="checkbox"/>	elastic2	52.45.191.29	Public IP	eipalloc-07d853181f3fcbf00
<input type="checkbox"/>	elastic3	54.159.234.5	Public IP	eipalloc-08fd7a3061811bb8a

Now go to asg3 instance and click on connect, run below commands.

Install docker, *sudo apt-get*
update sudo apt install
docker.io

Get your war and Dockerfile from your system to ec2 using gitbash.

scp -i /path/to/your-key.pem /path/to/your-file.war ec2-user@[your-ec2-instanceip]:/path/on/ec2/where/to/put/file

Change your pem file permissions if it doesn't allow access,
chmod 400 ~/Downloads/your-key.pem

```
scp -i ~/Downloads/demo1.pem ~/Downloads/SWE642hw3_Backup/SWE642hw3/Survey_Spring/Survey_Spring-0.0.1-SNAPSHOT.jar
ubuntu@ec2-3-218-63-246.compute-1.amazonaws.com: /home/ubuntu
```

```
scp -i ~/Downloads/demo1.pem ~/Downloads/SWE642hw3_Backup/SWE642hw3/Survey_Spring/Dockerfile ubuntu@ec2-3-218-63-246.compute-1.amazonaws.com: /home/ubuntu
```

Now check in your asg3 instance if the files are transferred using ls command.

Build an image and create a container, tag and push it to docker hub.

```
# Build the Docker image
sudo docker build -t your-image-name .

# Tag the Docker image for Docker Hub
sudo docker tag your-image-name your-dockerhub-username/your-repo-name:your-tag

# Log in to Docker Hub
sudo docker login

# Push the image to Docker Hub
sudo docker push your-dockerhub-username/your-repo-name:your-tag

# Run a container from your image, mapping port 8080 from the container to port 8080
sudo docker run -d -it -p 8080:8080 your-image-name
```

Check your image in public ipv4 dns address:8080/app

5. Creating cluster and deploying pods

Now execute below commands in asg3 to start rancher and login using public ipv4 dns address of asg3.

To start rancher,

```
sudo docker run --privileged=true -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher
```

To get the container-id of rancher image,

```
sudo docker ps
```

Get the password to enter for first time using this and change it later,

```
docker logs your-container-Id 2>&1 | grep "Bootstrap Password:"
```

Paste it in rancher website opened using public ipv4 dns address of asg3.

Go to navigation bar-> Cluster management-> Custom-> Create-> give a name-> create.

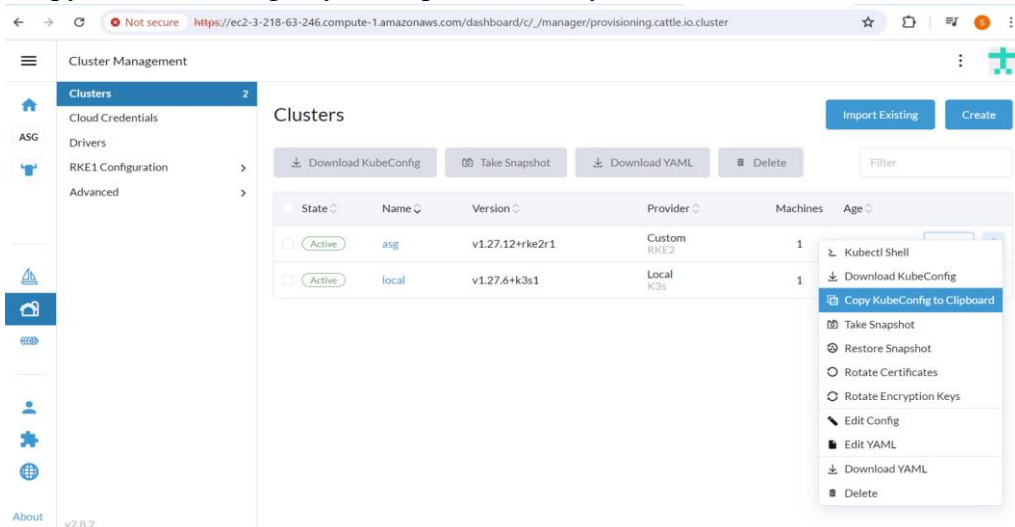
Copy the curl command under registration after checking insecure.

Now to clusterasg3 and connect. Paste the command and run.

Install kubectl,

```
snap install kubectl --classic
```

Copy the kubeconfig to your clipboard from you cluster



Create a .kube directory and paste it in config.

```
mkdir .kube
```

```
nano config
```

Ctrl+O to save, enter and Ctrl+X to exit.

Come out of your .kube directory and create deployment.yaml and service.yaml using nano command, customize them accordingly.

```

Y deployment.yaml x
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: deployment
5  spec:
6    replicas: 3
7    selector:
8      matchLabels:
9        app: asg
10   template:
11     metadata:
12       labels:
13         app: asg
14     spec:
15       containers:
16       - name: asg
17         image: srinijammula/asg3
18         ports:
19         - containerPort: 8080
20

```

```

Y deployment.yaml Y service.yaml x
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: service
5  spec:
6    type: NodePort
7    selector:
8      app: asg
9    ports:
10     - protocol: TCP
11       port: 8080
12       targetPort: 8080
13       nodePort: 30448

```

Apply both using `kubectl apply -f deployment.yaml` and `kubectl apply -f service.yaml`. Check your pods are running using `kubectl get pods`, you can also see them in rancher->worknodes->deployment. Now check if you can see the list from,

http://publicipv4_addressof_clusterinstance:nodeport/containername

Deployment: deployment (Active)

Namespace: default Age: 1 day Pod Restarts: 6

[Detail](#)[Config](#)

Image: srinijammula/asg3 Ready: 3/3 Up-to-date: 3 Available: 3

Endpoints: 30448/TCP

Annotations: [Show 2 annotations](#)

Pods by State

Scale 3

3

Running

[Pods](#) [Services](#) [Ingresses](#) [Conditions](#) [Recent Events](#) [Related Resources](#)

Download YAML


Delete

<input type="checkbox"/> State	Name	Image	Ready	Restarts	IP	Node	Age	
<input type="checkbox"/> Running	deployment-6f89d554fb-95d7j	srinijammula/asg3	1/1	2 (3h42m ago)	10.42.237.191	ip-172-31-59-10	21 hours	
<input type="checkbox"/> Running	deployment-6f89d554fb-nlgxt	srinijammula/asg3	1/1	2 (3h42m ago)	10.42.237.136	ip-172-31-59-10	21 hours	
<input type="checkbox"/> Running	deployment-6f89d554fb-wfd6g	srinijammula/asg3	1/1	2 (3h42m ago)	10.42.237.150	ip-172-31-59-10	21 hours	


6. Github repository


Create a github account if you don't have one. Click on create new repository and give a name like swe645asg3 and click on create.


Connect from your local system to git repository and push your index.html, war file, Dockerfile, Jenkins file (create one), deployment.yaml and service.yaml to your repository.


swe645asg3


Public


Pin


Unwatch 1


master


1 Branch

0 Tags

Go to file


Add file










Code

srinijammula

made a change in api

2913a9d · yesterday

3 Commits

	.mvn/wrapper	initial commit msg	yesterday
	out/artifacts/Survey_Spring.jar	initial commit msg	yesterday
	src	made a change in api	yesterday
	.gitignore	initial commit msg	yesterday
	.gitignore.txt	initial commit msg	yesterday
	Dockerfile	initial commit msg	yesterday
	Jenkinsfile	initial commit msg	yesterday
	Survey_Spring-0.0.1-SNAPSHOT.jar	initial commit msg	yesterday
	deployment.yaml	initial commit msg	yesterday

7. Setup Jenkins

Open jenkinsasg3 instance and execute these commands.

To update and install jdk,

```
sudo apt update
```

```
sudo apt install openjdk-17-jre
```

Add Jenkins repository,

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```


Update and install Jenkins,

```
sudo apt-get update
```

```
sudo apt-get install jenkins
```

Start Jenkins and check its status,

```
sudo systemctl start jenkins
```

```
sudo systemctl status Jenkins
```

Install docker to run containers,

```
sudo apt-get update
```

```
sudo apt-get install docker.io
```

```
sudo systemctl status docker
```

Add Jenkins User,

```
sudo usermod -a -G docker Jenkins
```

Install kubectl,

```
sudo snap install kubectl --classic
```

Configure Kubernetes Access,

```
cd /var/lib/jenkins/.kube
```

Create Config file in .kube and paste kubeconfig contents from cluster you created in rancher,

```
sudo nano config
```

Verify Jenkins Status,

```
sudo systemctl status Jenkins
```

Access Jenkins Web Interface by public ipv4 dns of jenkinsasg3 instance.

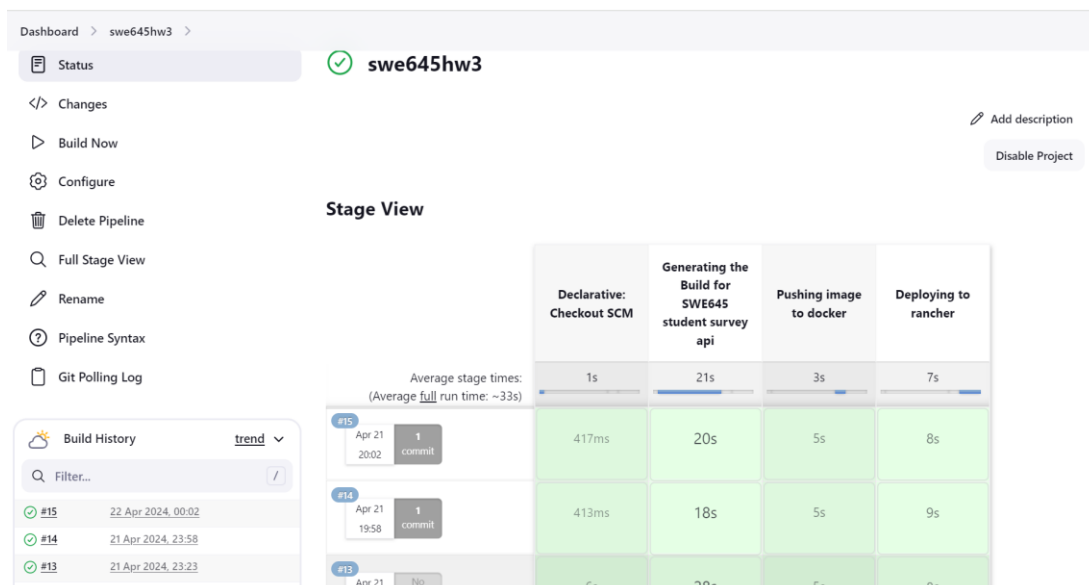
Initial Admin password to unlock Jenkins and create account ,

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Create a Jenkins Pipeline following these steps,

- Select "+ New Item", choose "Pipeline", and give it a name.
- Under "Build Trigger", select "Poll SCM" and enter * * * * * (runs every minute).
- Under "Pipeline", choose "Pipeline script from SCM", select Git as SCM, provide your repository URL and branch, and set the Jenkinsfile path.
- Save the pipeline configuration.

Check the build of Jenkins if everything is deployed correctly. If you get any errors refer the console at the bottom (click on failed build) to know the exact error.



CONTRIBUTIONS

- By Sruthi Sivasamy and Aditya Gottipati,
 - Spring Boot (CRUD)
 - Connecting to RDS
 - Jar file
- By Tejeswar Sadanandan and Srini Jammula,
 - Deploying application to cloud (EC2, K8s, Rancher)
 - Repository and Pipeline (GitHub and Jenkins)

REFERENCES

- <https://stackoverflow.com/questions/70541720/jenkins-has-no-installation-candidate-error-while-trying-to-install-jenkins-on>
- <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux>
- <https://www.jenkins.io/doc/book/pipeline/jenkinsfile/>
- <https://www.jenkins.io/blog/2023/03/27/repository-signing-keys-changing/>

PS: We have not uploaded **out** folder of spring application as it is very large and taking longer time to compress or extract after that. Kindly reach out to any of us if that is required or if you face any issue in running the application. Thank you!