

SWE645 Homework 2: Report on Containerizing Student Survey Form

Team Members	G#
Pratish Mashankar	G01354094
Sanjeevkumar Hanumantlal Sharma	G01393366
Teja Sajja	G01395837
Sai Ram Dasarapu	G01377310

Links

Task	Link
GitHub	https://github.com/PratishMashankar/cs645.git
Dockerhub	https://hub.docker.com/repository/docker/mythprat/swe645hw2/general
AWS K8s Deployment	http://ec2-54-224-58-34.compute-1.amazonaws.com:31819/student_survey-1/
Rancher node-port	https://ec2-18-211-220-47.compute-1.amazonaws.com/k8s/clusters/c-m-b5xcj9xw/api/v1/namespaces/default/services/http:cs645-hw2:8080/proxy/student_survey-1/
Video and READme Location	Zip File

Introduction

Our project embarked on a journey to enhance the deployment and management of the web application developed during Homework 1 – Part 2. To achieve this, we harnessed the power of Docker container technology to containerize the application, facilitating seamless deployment and scaling. The next step involved orchestrating these containers on the open-source Kubernetes platform, ensuring the application's resilience and scalability. We opted for a baseline configuration with a minimum of three pods continuously running, considering Rancher for Kubernetes cluster setup. Furthermore, our endeavor culminated in establishing a robust CI/CD pipeline, complete with a Git source code repository hosted on GitHub and Jenkins for automated building and deployment of the application on our Kubernetes cluster.

Installation, Implementation, and Setup

1. GitRepo Setup:

1. Create an empty Git repo on GitHub

2. Ensure all files are present in the local folder. Open terminal on the folder and write the following commands:
 - a. `>>git init`
 - b. `>>git remote add origin https://github.com/PratishMashankar/cs645.git`
3. Perform git add, commit, and push
4. The GitRepo is ready

2. Creating and pushing a Docker image to DockerHub:

1. In this part we installed Docker on our machine and created a Docker Hub account so that we can share our project files from our system with the create EC-2 instance
2. To operate the docker there are some thumb rules like the file should be named Dockerfile. So we created a file named Dockerfile in our project directory.
3. After setting up the docker and project directory. From preexisting HW-1 we inserted all the files along with the war file into the present project directory along with the Dockerfile.
4. A docker file consists of a set of commands which the docker runs in a sequential order. We inserted there commands :

FROM tomcat:10.1-jdk11-openjdk

COPY student_survey-1.war /usr/local/tomcat/webapps

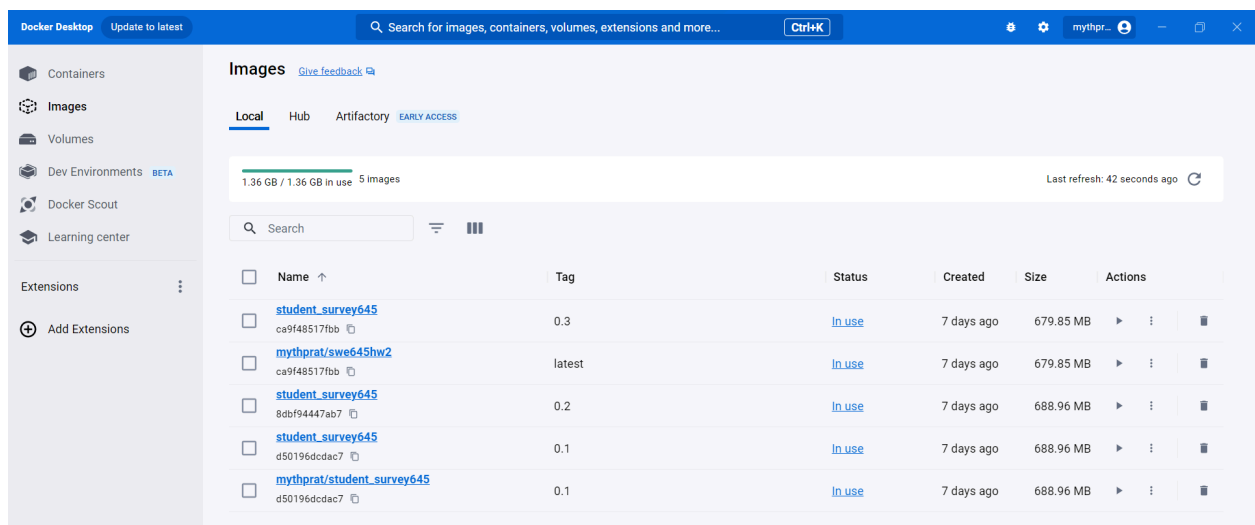
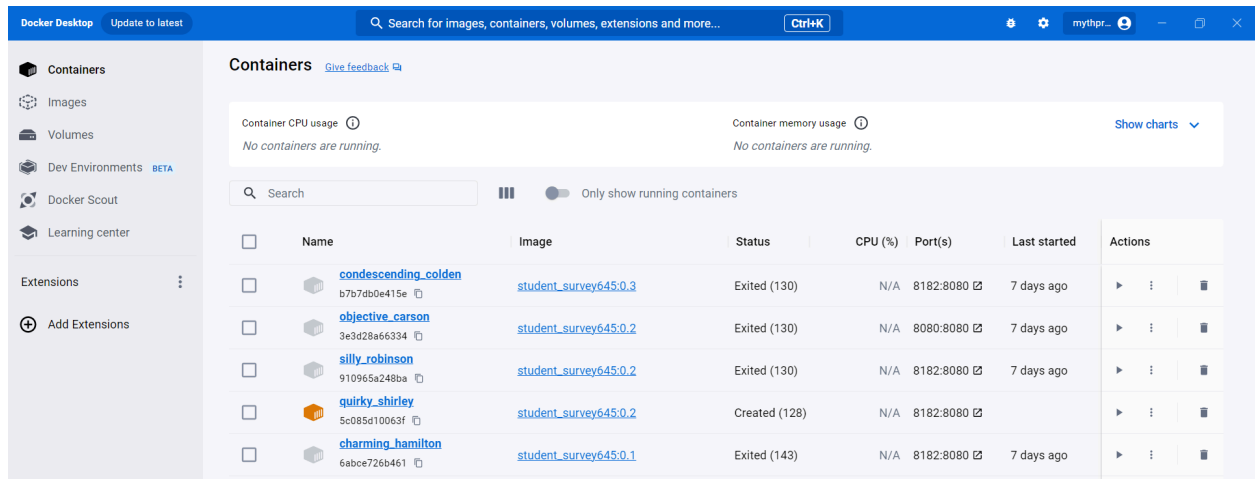
EXPOSE 8080

CMD ["catalina.sh","run"]

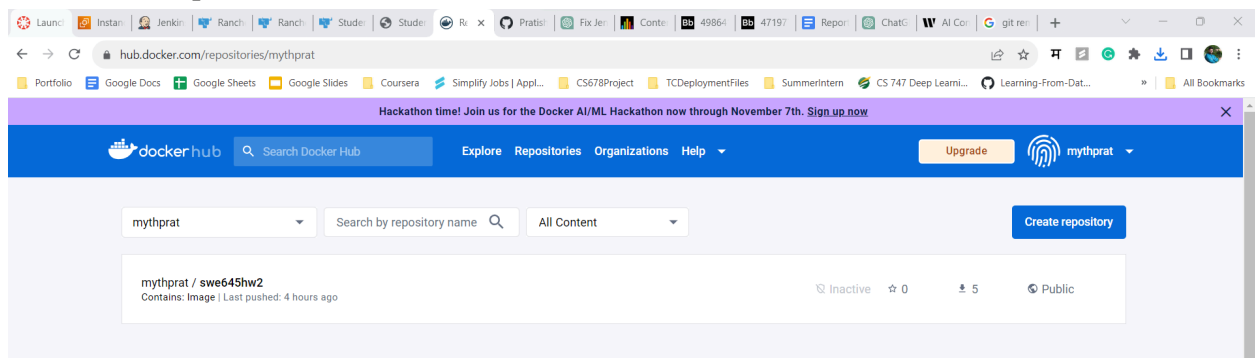
This command copies the war file into the tomcat webapps folder and then asks it to expose in port 8080 and then run tomcat.

5. After this we create a docker image from the created Dockerfile above using this command :
docker build --tag student_survey645:0.1
6. After creating the command we verified it using **docker run -it -p 8182:8080 student_survey645:0.1**.
We can see the result on our local system in our web browser.
7. Now after this we have to export this docker image into a docker hub account so that we can run this similarly on EC-2 instance this image will be used in the Rancher Deployment step.
8. Login to DockerHub using the command: `docker login -u . h.`
9. Change the name of your Docker image to follow the format /: using the docker tag command.
For example: **docker tag student_survey645:0.1 mythprat/swe645hw2:0.1**
10. Verified that our Docker image is pushed to DockerHub and accessible from the internet. This image will be used in the Rancher Deployment step.

Docker Desktop Images



DockerHub Repo



3. Installing docker and Rancher (to setup kubernetes cluster) on AWS EC2:

1. Firstly we have created two AWS EC2 instances from the learner lab provided.

2. Instances were created mostly with default settings and selections were made with AMI of Ubuntu, instance type of t2.large for Kubernetes and t2.medium for Rancher and allowing http, https options.
3. Config storage was increased to 30 Gib.
4. Two elastic IPs were created and associated with both of the EC2 instances.
5. Once the instances were up and running we connected both instances and we made a regular update check using the cmd **sudo apt-get update** after logging in as root user.
6. Then we installed docker on both instances using cmd **sudo apt install docker.io**.
7. Then we installed rancher in only the first instance i.e. Rancher instance using cmd **sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher**.
8. We then opened the public IPv4 of the instance to open up the rancher portal.
9. For the password of the rancher we needed to generate the password from the instance cmd line using the container ID.
10. Container ID was fetched using **sudo docker ps**.
11. Then we used the container ID to generate the password using **sudo docker logs container ID | grep "Bootstrap Password:"**.
12. We used the login credentials to log in to the rancher portal.
13. After logging in we were prompted to change the password and we set up our own password.
14. Nextlty we created a custom kubernetes cluster by checking in with the three nodes.
15. A cmd was generated that was needed to be copied into the other EC2 instance.
16. After a few minutes we get the message of the cluster as active.
17. Click on the active cluster and click on deployments to create a new deployment object with 3 replicas and create it as a nodeport.
18. Provide the image of the docker hub that we pushed.
19. Create one more deployment for load balancer.
20. Once the deployments are active we will be able to see our application by clicking on the services section under the specific deployment and on NodePort 8080 link.
21. We append our war file name at the end of the URL to access our application.

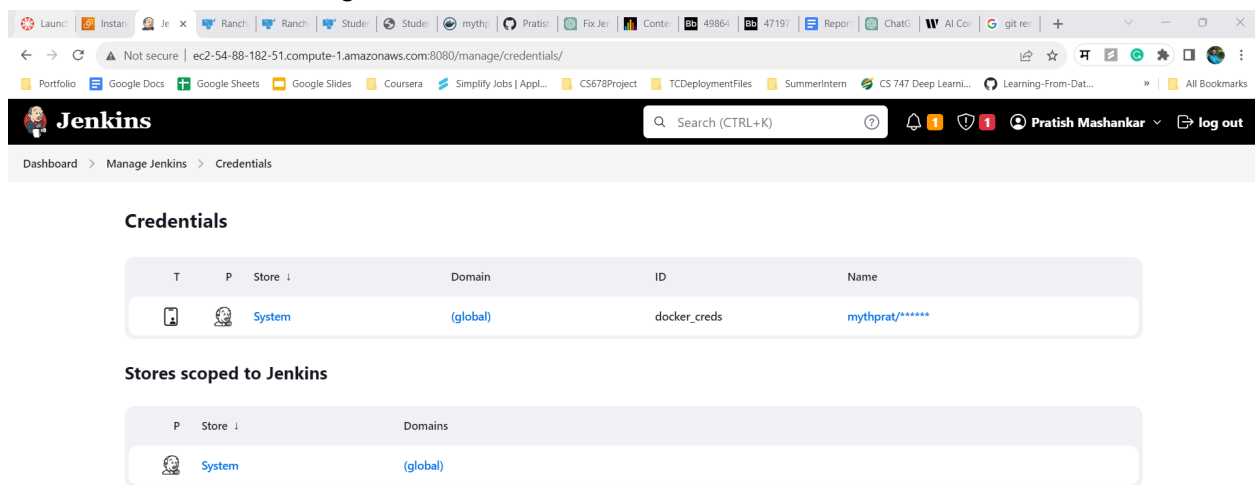
4. Steps to Install and setting-up Jenkins:

1. Create EC2 instances with the same process as we did with the last sections along with assigning it with an elastic IP.
2. Once connected, install docker in the instance as we installed it on previous instances.
3. Install Jenkins.
4. Once installed check the status by using the cmd **systemctl status jenkins** and opening a browser on public ip:8080.
5. Install Kubectl and login as jenkins user using the cmd
Sudo apt install snapd
Sudo snap install kubectl --classic
Sudo su jenkins

- Now login to your rancher cluster and copy the information of the kubeconfig file and paste it in /home/Jenkins/.kube/config file, where the kube folder was created jenkins home on EC2 console.
- Copy the content of the Kubeconfig from the rancher to the file in /home/Jenkins/.kube/config file in vi editor.
- Verify the running of kubectl using the cmd **kubectl config current-context**.

5. Create Credentials for Git and Docker:

- Go to manage credentials section which is present inside the manage jenkins section of the dashboard.
- Select system under stores scopes to jenkins.
- Click on global credentials.
- Add our docker and github credentials.



6. Building Pipeline:

- We use Jenkins to implement our pipeline.
- To implement a pipeline in Jenkins UI select 'New items' and then select pipeline item.
- While creating the pipeline we select cron job so that it checks every minute as a build trigger - this update every time when we make any change in our github repo
- Then we provide our Git repository url and credentials to connect with our Git repository.
- We also need to provide information to Jenkins where the Jenkinsfiles are located. In our case it's going to be the root directory of the Git repository.
- Leave the rest of the setting options as default and Save.


Setup Images Given Below:

Dashboard > All >

Enter an item name

TestExample

» Required field

**Freestyle project**

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

☒ **GitHub project**

Project url ?

https://github.com/PratishMashankar/cs645.git/

Advanced ▾

- ☐ Pipeline speed/durability override ?
- ☐ Preserve stashes from completed builds ?
- ☐ This project is parameterized ?
- ☐ Throttle builds ?

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build whenever a SNAPSHOT dependency is built ?
- ☐ GitHub hook trigger for GITScm polling ?
- ☒ Poll SCM ?

Schedule ?

* * * * *

⚠ Do you really mean "every minute" when you say "* * * * *"? Perhaps you meant "H * * * *" to poll once per hour

Would last have run at Tuesday, October 17, 2023 at 3:02:00 AM Coordinated Universal Time; would next run at Tuesday, October 17, 2023 at 3:02:00 AM Coordinated Universal Time.

- ☐ Ignore post-commit hooks ?

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/PratishMashankar/cs645.git

Credentials ?

- none -

+ Add

Advanced

7. Required Plugins and writing Jenkins file:

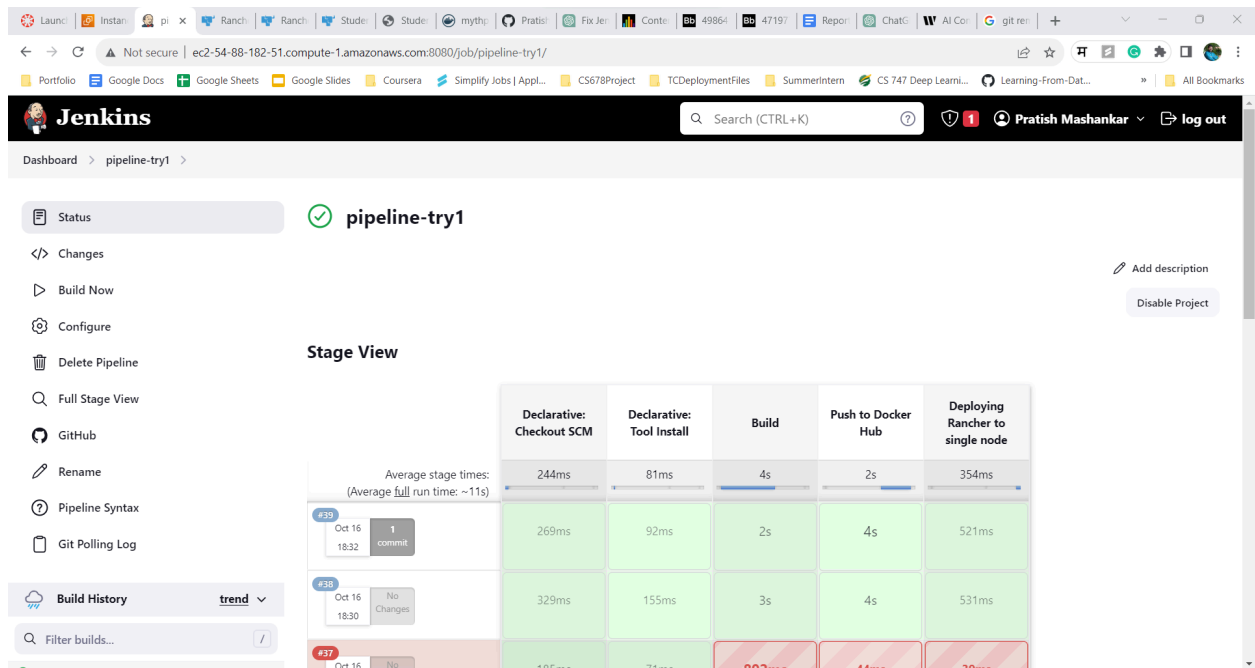
1. In jenkins navigate to configure and manage plugins and install the required plugins for docker, github and safe restart.
2. Under Manage Jenkins we can configure DockerHub credentials as environment variables in the Jenkinsfile, allowing secure access to DockerHub within the Jenkins pipeline processes.
3. Create Jenkinsfile in the same folder that has docker in it.

```
kubernetes-cluster1.yaml  Jenkinsfile
1  @NonCPS
2  def generateTag() {
3      return "build-" + new Date().format("yyyyMMdd-HHmmss")
4  }
5
6  pipeline {
7      environment {
8          registry = "mythprat/swe64shw2"
9          registryCredential = "docker_creds"
10     }
11     agent any
12
13     // tools {
14     //     maven 'Maven' // Replace with the desired Maven version
15     // }
16
17     stages{
18         stage('Build') {
19             steps {
20                 script {
21                     // sh 'mvn clean install'
22                     // sh 'mvn clean package'
23                     // sh 'mvn war:war'
24                     checkout scm
25                     sh 'rm -rf *.war'
26                     sh 'jar -cvf student_survey-1.war -C src/main/webapp/ .'
27
28                     sh 'echo ${BUILD_TIMESTAMP}'
29                     tag = generateTag()
30                     docker.withRegistry('', registryCredential){
31                         def customImage = docker.build("${registry}/${tag}")
32                     }
33                 }
34             }
35         }
36
37         stage('Push to Docker Hub') {
38             steps {
39                 script {
40                     sh 'echo ${BUILD_TIMESTAMP}'
41                     docker.withRegistry('', registryCredential) {
42                         def image = docker.build("${registry}/${tag}")
43                         image.push()
44                     }
45                 }
46             }
47         }
48     }
49
50     stage('Deploying Rancher to single node') {
51         steps {
52             // ...
53         }
54     }
55 }
```

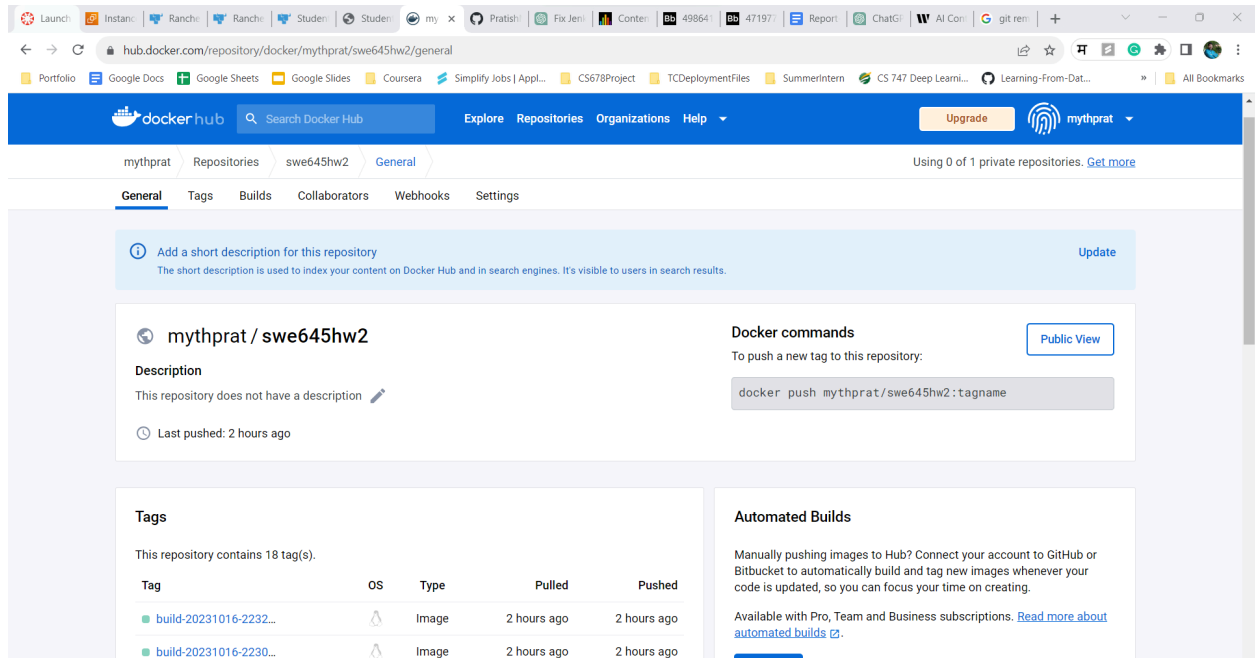
8. Running the Pipeline

1. We made a small change in the heading in our survey_form.html by adding “changed with Jenkins” in blue
2. Push the changes to the GitHub
3. The pipeline triggered the building of .war file, the creation of Docker image, pushing it to DockerHub, and deploying it to Rancher.
4. The changes are reflected in the pipeline

8.1. CI/CD Pipeline success:



8.2. Image is built in DockerHub



The screenshot shows the Docker Hub repository page for `mythprat/swe645hw2`. The page includes a search bar, navigation tabs (General, Tags, Builds, Collaborators, Webhooks, Settings), and a description section. The description states: "This repository does not have a description". The "Tags" section shows two tags: `build-20231016-2232...` and `build-20231016-2230...`, both of type "Image" and pushed 2 hours ago. The "Docker commands" section shows the command `docker push mythprat/swe645hw2:tagname`. The "Automated Builds" section mentions connecting to GitHub or Bitbucket for automated builds.

mythprat / swe645hw2

Description

This repository does not have a description

Last pushed: 2 hours ago

Docker commands

To push a new tag to this repository:

```
docker push mythprat/swe645hw2:tagname
```

Tags

This repository contains 18 tag(s).

Tag	OS	Type	Pulled	Pushed
build-20231016-2232...	linux	Image	2 hours ago	2 hours ago
build-20231016-2230...	linux	Image	2 hours ago	2 hours ago

Automated Builds

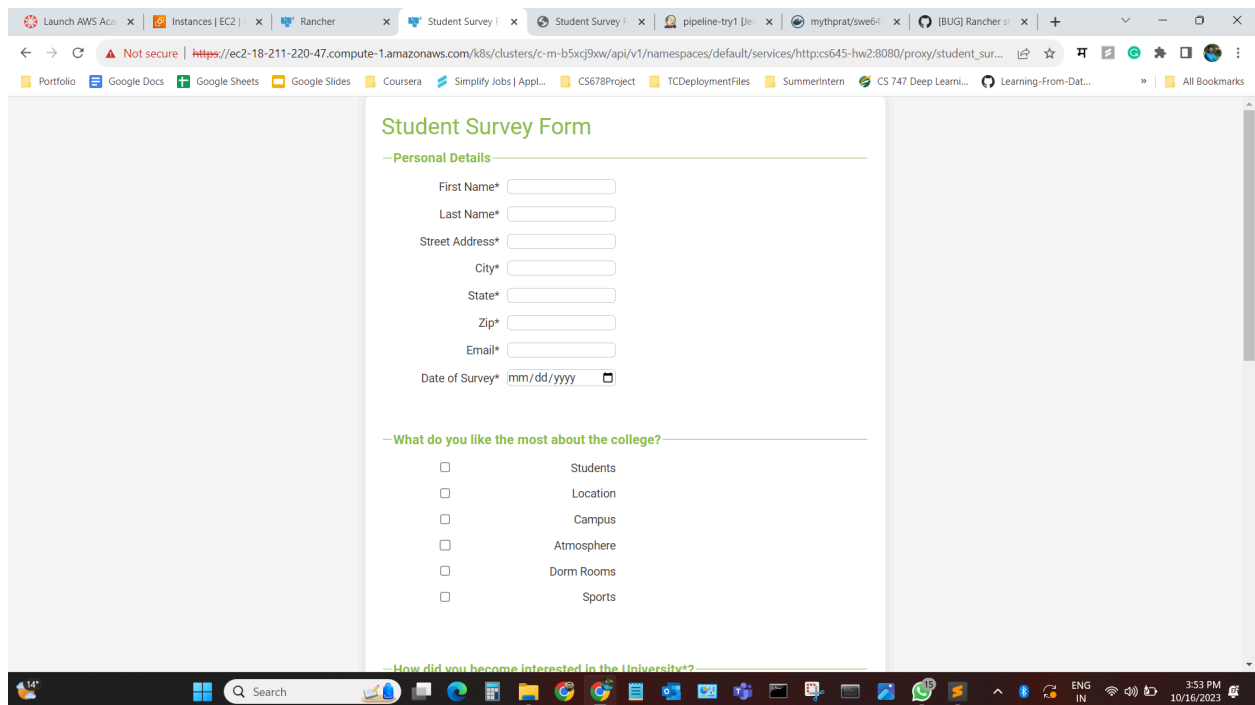
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#).

8.3. Node-port deployment EC2

survey_form is running on our cluster node-port on the EC2 server and changes are being reflected

Before:



The screenshot shows a web browser displaying the "Student Survey Form". The form is titled "Student Survey Form" and has two main sections: "Personal Details" and "What do you like the most about the college?". The "Personal Details" section includes fields for First Name, Last Name, Street Address, City, State, Zip, Email, and Date of Survey. The "What do you like the most about the college?" section includes checkboxes for Students, Location, Campus, Atmosphere, Dorm Rooms, and Sports. The form is running on an EC2 instance, as indicated by the browser's address bar showing the URL `https://ec2-18-211-220-47.compute-1.amazonaws.com/k8s/clusters/c-m-b5xcj9xw/api/v1/namespaces/default/services/http:cs645-hw2:8080/proxy/student_sur...`.

Student Survey Form

Personal Details

First Name*

Last Name*

Street Address*

City*

State*

Zip*

Email*

Date of Survey* mm/dd/yyyy

What do you like the most about the college?

☐ Students

☐ Location

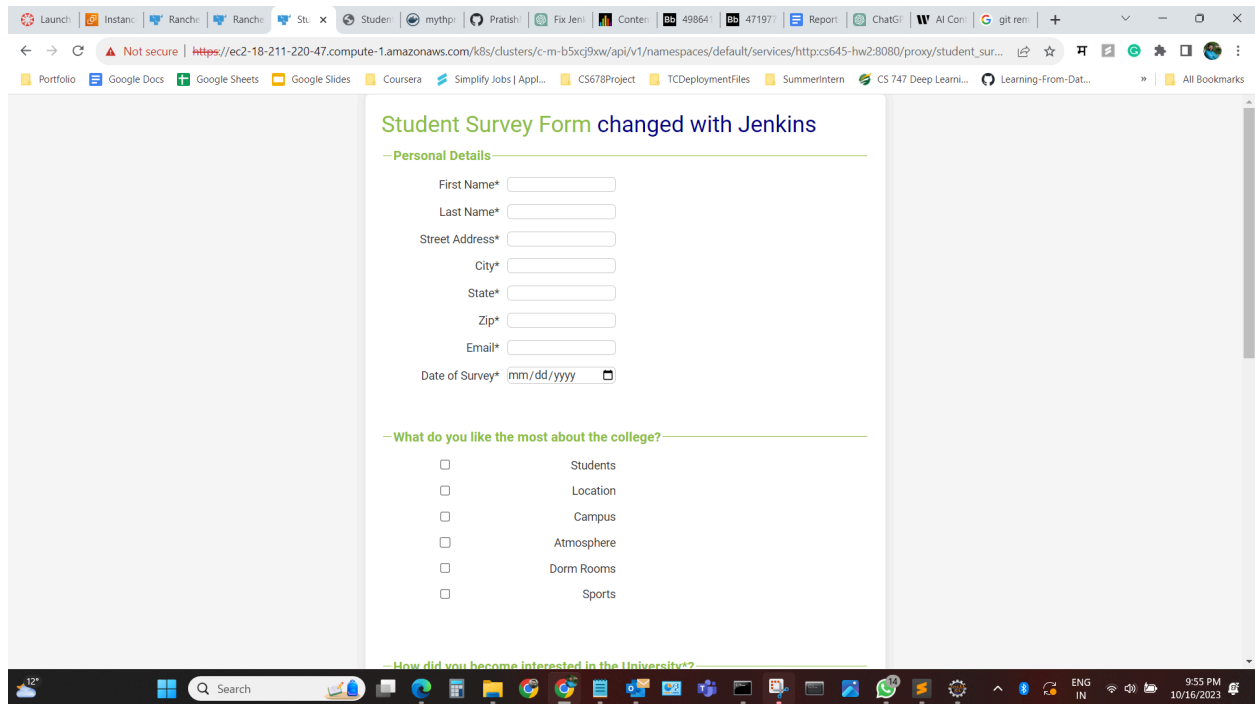
☐ Campus

☐ Atmosphere

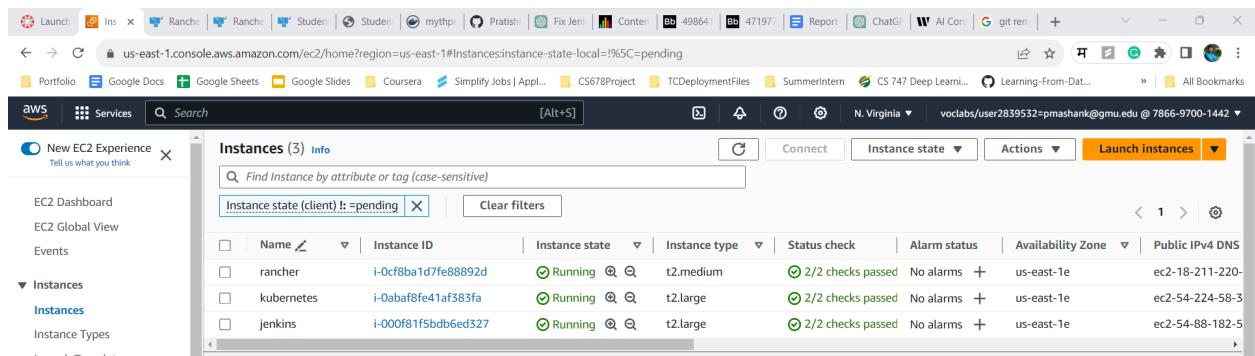
☐ Dorm Rooms

☐ Sports

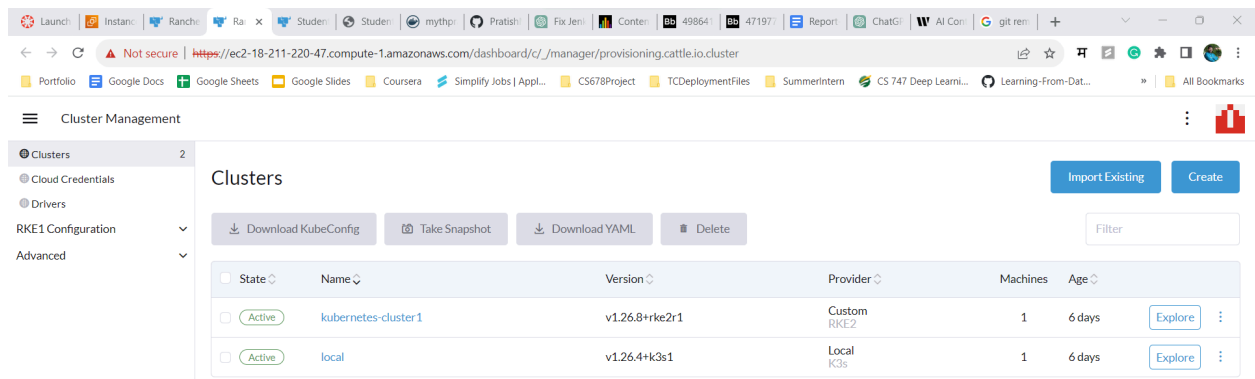
After GitHub Commit:



8.4. EC2 Instances



8.5. Cluster



8.6. Deployments

Cluster: kubernetes-cluster1

Deployments

State: Active

State	Name	Image	Ready	Up To Date	Available	Restarts	Age	Health
Active	cs645-hw2	mythprat/swe645hw2:build-20231016-223235	3/3	3	3	3	6 days	

8.7. Pods

Cluster: kubernetes-cluster1

Deployment: cs645-hw2 (Active)

Namespace: default | Age: 6 days | Pod Restarts: 3

Image: mythprat/swe645hw2:build-20231016-223235 | Ready: 3/3 | Up-to-date: 3 | Available: 3

Endpoints: 31819/TCP

Annotations: Show 1 annotation

Pods by State

Scale: 3

3 Running

State	Name	Image	Ready	Restarts	IP	Node	Age
Running	cs645-hw2-57867d846c-cnvr	mythprat/swe645hw2:build-20231016-223235	1/1	1 (9m14s ago)	10.42.32.65	ip-172-31-59-106	3.2 hours
Running	cs645-hw2-57867d846c-vnqnk	mythprat/swe645hw2:build-20231016-223235	1/1	1 (9m14s ago)	10.42.32.72	ip-172-31-59-106	3.2 hours
Running	cs645-hw2-57867d846c-zx9ln	mythprat/swe645hw2:build-20231016-223235	1/1	1 (9m14s ago)	10.42.32.124	ip-172-31-59-106	3.2 hours

Work Distribution

The team, consisting of Pratish Mashankar, Sanjeevkumar Hanumantlal Sharma, Teja Sajja, and Sai Ram Dasarapu, collectively worked on implementing a CI/CD pipeline using Kubernetes, Rancher, and Jenkins. All team members were actively present and involved throughout the entire process, observing and attempting implementation on their individual systems. Pratish and Sanjeev primarily focused on Jenkins deployment, leading the way to establish the final CI/CD pipeline, while also overseeing the overall implementation. Teja and Sai Ram played a significant role in configuring Docker Desktop, Kubernetes, and Rancher, laying the groundwork for the infrastructure components

essential to the project. Their combined efforts and collaboration contributed to the successful implementation of the CI/CD pipeline. Pratish provided images from his system, for documentation and reference purposes.

Conclusion and Lessons

After completion of the assignment we were able to build a highly efficient, containerized, and scalable application using technologies like docker which was used to containerize our application ensuring consistency across various platforms. For container orchestration, we have leveraged the services of the Kubernetes cluster management system and rancher platform to enable seamless deployment and scaling with guaranteed resilience and availability. Finally, the implementation of the CI/CD pipeline using Jenkins streamlined the development process, automatic building, testing, and deployment tasks.

References

- <https://www.rancher.com/>
- <https://www.docker.com>
- <https://kubernetes.io>
- <https://www.jenkins.io/doc/>
- <https://github.com/>
- <https://about.gitlab.com/topics/ci-cd/>
- https://docs.aws.amazon.com/whitepapers/latest/cicd_for_5g_networks_on_aws/cicd-on-aws.html