Homework Assignment 2 Report
SWE-645, Spring 2023 **04/10/2023**

Name : Ashrit Mullur Ravi
GMU ID : G01391973

Name : Vaishnavi Gururaj
GMU ID : G01399296

Name : Harsha Masandrapalya Vanarajaiah
GMU ID : G01396731

The GitHub link for the Project is https://github.com/Ashrit26/swe645_project_2

Part 1:
Containerizing the Web application that we had developed in Homework 1 – Part 2, using Docker container technology.

First, we installed the docker desktop in our machine. Then, we created an account on https://hub.docker.com/.
We then created a file called Dockerfile in Eclipse after which using the warfile which was built in the Homework-1, we built the docker image using the following command.
“sudo docker build –platform linux/amd64 -t -pstudentsurvey645:0.1 .”

Then, we used the following command to verify the image:
“sudo docker run -it -p 8182:8080 studentsurvey645:0.1”
This enabled us to access the webpage through the localhost URL which is as follows:.
http://localhost:8182/swe645-assignemnt1

After ensuring the image was accurate, we pushed the image to docker hub using our credentials.

We then verified our image on the hub which was accessible from the internet.

Part 2:
Deploying the containerized application on the open-source container orchestration platform Kubernetes to enable scalability and resiliency of the application.

We created an AWS EC2 instance with Ubuntu Server 22.04 LTS (HVM), SSD Volume Type AMI. While creating the instance, we enabled public security and downloaded the private key (pem file (swe645assignment2.pem)) to access the server.
Under Instance Type we chose t2.large. We tried to run it multiple times on other types under free tiers like medium but since Rancher didn’t start, we used the large type.

We then executed this command to ensure that our key was not publicly viewable “chmod 400 swe645assignment2.pem”

For connecting to the AWS EC2 machine, the following linux command was used:
ssh -i "swe645assignment2.pem" ubuntu@ec2-3-223-138-39.compute-1.amazonaws.com

Then update using following command:
“sudo apt-get update”

We installed docker using following command:
“sudo apt install docker.io”
After installing docker, we enabled the rancher image using the following command:
“sudo docker run --privileged=true -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher”

We could then open the Public IPv4 address from the instance in the browser to see the dashboard of rancher.
https://3.223.138.39/dashboard/home

The credentials of Rancher dashboard are:
Username: admin
Password: admin@123456

For security purposes, we then created our own password and logged in as an admin user. After this, we created the custom cluster with our new cluster name. This cluster hosted our etcd, control panel, and worker nodes.
Once the cluster was active, it was ready for deployment.

Then under deployment, we created two deployments namely NodePort and LoadBalancer with their Private port as 8080. Also added our container image which we had pushed to the Docker Hub in part -1.
After the deploying, the pods become active in some time.

Clicking on the nodeport services let us view our survey page. It renders to an address on the browser, we edited it by adding our war file name.
https://3.223.138.39/k8s/clusters/c-m-nj4fw7dj/api/v1/namespaces/default/services/http:project2nodeport:8080/proxy/swe645-assignment1/

We then downloaded and saved the KubeConfig file (project2.yaml) for the cluster.

Part 3:
Establishing a CI/CD pipeline that includes a Git source code repository at GitHub, and Jenkins for automated build and for the automated deployment of our application on Kubernetes

We created an AWS EC2 instance for Jenkins with Ubuntu Server 22.04 LTS (HVM), SSD Volume Type AMI. While creating the instance, we enabled public security and downloaded the private key (pem file (swe645assignment2.pem)) to access the server.

Under Instance Type we chose t2.large. We tried to run it multiple times on other types under free tiers like medium but since Rancher didn’t start, we used the large type.

We then executed this command to ensure that our key was not publicly viewable
“chmod 400 swe645assignment2.pem”

For connecting to the AWS EC2 machine, the following linux command was used:
ssh -i "swe645assignment2.pem" ubuntu@ec2-34-206-216-101.compute-1.amazonaws.com

Then update using following command:
“sudo apt-get update”

We installed docker using following command:
“sudo apt install docker.io”

Then we followed the instructions provided in this link https://pkg.jenkins.io/debian/ to install jenkins on our instance and configured it with necessary requirements

We used the Public IPv4 address from the instance to view the dashboard of Jenkins in our browser.
http://34.206.216.101:8080/

The Credentials of Jenkins Dashboard are:
Username: project2
Password: admin@123456

We created a Jenkins pipeline and installed necessary plugins as required. (CloudBees docker & Rancher).
(https://docs.cloudbees.com/docs/admin-resources/latest/plugins/docker-workflow/
https://plugins.jenkins.io/rancher/)

We configured the pipeline with the source as our GitHub repo in such a way that it polls the SCM every minute (*/1****) and provided access to JenkinsFile stored in our GitHub repository.

A JenkinsFile was created and pushed, which includes several stages such as build, push to Docker Hub and deployment on Rancher single node and Rancher.

We also had to add the credentials of both GitHub and DockerHub under manage credentials section.

After setting up the pipeline successfully, any changes made to the code and pushed to GitHub triggered the SCM polling process, which in turn initiated a new build. Once all the necessary steps were completed, the newly built code was pushed to Docker Hub, and the image was updated in Rancher deployments, thereby hosting the latest build.

Issues faced while doing the Assignment with their solution

In part-1 of the assignment,
Containerization is a complex process that required a good understanding of the technology. The process involved creating a Dockerfile, building the Docker image, and testing the containerized application to ensure that it worked as expected. Some of the challenges that we faced during this process include dependency management, configuring the application to run within a container, and ensuring that the containerized application is secure. Since we MacBook Air with an M1 chip, our image was built with a different platform.
So we ran the following command to build it in the Linux platform with an amd64 processor.
“sudo docker build –platform linux/amd64 -t -pstudentsurvey645:0.1 .”

Then in part-2 of the assignment,
Kubernetes is a complex container orchestration platform that requires a good understanding of the technology. Some of the challenges we faced while deploying the containerized application on Kubernetes included setting up the Kubernetes cluster, configuring the Kubernetes deployment manifest file, and configuring the Kubernetes service. Each time while creating the cluster gave a different error, like the cluster was not getting active even after 30 minutes stating “provisioning” under the cluster section. Then we used two instances, one for rancher and one for worker nodes so the load is not on one instance. Also, we faced network issues during the cluster creation.
We created multiple instances in case there was any issue while creating the instances.
Also, we came up with a solution like making the IP address elastic. As we had only 4 hours of buffer time, the instances once restarted came up with different Public IPv4 address every time. So under Network & Security tab, we allocated elastic IP to all the instances created.

Then in part-3 of the assignment,
Setting up a CI/CD pipeline requires knowledge of the tools and technologies involved, such as Git, Jenkins, and Kubernetes. Some of the challenges that we faced included, setting up Jenkins to automate the build and deployment process, and configuring the Jenkins pipeline to deploy the application on Kubernetes.
Jenkins pipelines are typically configured using a declarative syntax, which was initially difficult to get right. Even a small mistake in the configuration caused the pipeline to fail.
The CI/CD pipeline needs to integrate with other tools and systems, such as version control systems (here we have used GitHub), testing frameworks, and deployment tools. Any issues with the integration can cause problems with the pipeline.
Jenkins relies heavily on plugins to extend its functionality. Plugins were not installed correctly in the initial setup and some were incompatible with each other, these caused problems with the pipeline.
We faced an error in the ownership of the directory which was not owned by the root user and group, which led to our issues with certain applications that rely on correct file permissions. To fix this issue, we used the following command to change the ownership of the directory to root:
“sudo chown root:root /path/to/directory”