**SWE 645 Homework / Assignment 3**

Author of document: Qasim Shahid

Date Due: 4/16/2025

Note: THIS DOCUMENT DOES NOT CONTAIN API TESTING. PLEASE WATCH THE VIDEO INCLUDED WITH THIS SUBMISSION TO SEE A FULL TEST USING POSTMAN. THIS DOCUMENT ONLY COVERS HOW TO SET THIS UP!

GitHub Repository: <https://github.com/qasimshahid/qshahid-swe645-asst3/>

Note: This repo is currently public but will be made private after this assignment is graded.

Another note: I ran out of budget on my AWS Learner Lab account, so this is being hosted on my own AWS account.

Website link (this will be available via Elastic IP and will be accessible since I will keep it up on my own AWS account until it is graded).

* <http://98.82.89.116:30007/api/surveys/version>
* (This is the link for an HTML page that shows the version of the app so that you can just see that it is up and running)

Other endpoints and what HTML method to use to access them

(Please replace any parameters in {} with your own specified parameters, such as ID of surveys)

* To get a survey by ID
  + GET
  + [http://98.82.89.116:30007/api/surveys/{id}](http://98.82.89.116:30007/api/surveys/%7bid%7d)
* To get all surveys
  + GET
  + <http://98.82.89.116:30007/api/surveys>
* To delete a survey by ID
  + DELETE
  + [http://98.82.89.116:30007/api/surveys/{id}](http://98.82.89.116:30007/api/surveys/%7bid%7d)
* To post a new survey
  + POST (Must send a valid survey in HTML Request Body via JSON, [please see the README.md of the GitHub repository for examples.](https://github.com/qasimshahid/qshahid-swe645-asst3/))
  + <http://98.82.89.116:30007/api/surveys>
* To update an existing survey by ID
  + PUT (Must send a valid survey in HTML Request Body via JSON, [please see the README.md of the GitHub repository for examples.](https://github.com/qasimshahid/qshahid-swe645-asst3/))
  + [http://98.82.89.116:30007/api/surveys/{id}](http://98.82.89.116:30007/api/surveys/%7bid%7d)

In this homework assignment, we are tasked to build a CI/CD pipeline using a very simple REST API for doing CRUD operations on survey data. We containerize the Spring Web REST API using Docker and then deploy it to a Kubernetes cluster, running at least 3 replicas. We use Rancher as a Kubernetes management platform to manage our cluster.

The Jenkins pipeline is triggered upon a push to the main branch of the GitHub repository. The pipeline does the following:

1. Get the most recent source code by cloning the repository
2. Build the jar file using the repository we cloned.
3. Build the docker image defined in the Dockerfile using docker build
4. Push the image we just built to Docker Hub (this requires an account, by default, the source code assumes mine is being used with username qshahid)
5. Deploys the latest container image for our app the Kubernetes cluster, creating a Deployment with 3 replicas (basically deploying the container to 3 pods). The app is exposed externally via a NodePort service, with port 30007 mapped to the container's port 8080, allowing external access at  
   http://<public-cluster-ip>:30007/api/surveys/version

Please make sure you have a Docker Hub account, an AWS account, and a GitHub account. Please also clone/fork my GitHub repository to your own account so you can configure the webhook that allows us to automatically build and deploy whenever we push to the main branch. Please also make this GitHub repository public as that is what I used. Continue reading to see all the other things that are required to make this work.

**AWS Security Group Setup**

Please duplicate / copy the following security group before doing anything since we will be assigning this security group to multiple things. This will make sure you are able to run without having port forwarding issues.

A screenshot of a computer

Description automatically generated

**AWS Database Setup**

Since we need to store our data somewhere, we need to create a free-tier MySQL database (or Amazon RDS) database. We need to make it publicly available so that our API can access it and we also need to create some credentials that we will remember, which will also be used in the deployment process so we can access the database.

Please also install MySQL Workbench on your computer so you can validate API operations:

<https://www.mysql.com/products/workbench/>

Go to Amazon RDS and create a new database. Copy these settings:

Use MySQL

A screenshot of a computer

Description automatically generated

Next, make sure to choose the free tier DB. Make a self-managed username and password. Note this down since we will need this to connect.

A screenshot of a computer

Description automatically generated

Then, there will be a bunch of options. Scroll down to connectivity and copy the following:

A screenshot of a computer

Description automatically generated

After this, click “Create database”

Now, open MySQL Workbench. Try to connect to your database using the IP, 3306 port, and your username and password that you set.

A screenshot of a computer

Description automatically generated

You should see this if you did everything correctly. Create the connection and connect to the DB. We need to execute one command in it.

A screenshot of a computer

Description automatically generated

Once connected, please go to the GitHub repository and go to the createDb.sql file and copy the command to create the database. This is the only SQL command we need to run.

“CREATE DATABASE survey\_db;”

A screenshot of a computer

Description automatically generated

Now, you are done with database setup. Note down the AWS DB link and your credentials. We will be injecting these credentials into our Jenkins instance by creating a new credential that uses a secret file. This will be “application-secret.properties” so that our API can connect to the database.

Create a file exactly like this but use your own DB link and credentials. Note that I specify the database I want to connect to, survey\_db, and you should do this as well. We created this database in the previous step.

A screen shot of a computer

Description automatically generated

Have it be the same name as well, application-secret.properties. Like I said, we will give this file to Jenkins.

Once the file has been created, save it somewhere where you will remember.

**AWS EC2 Setup**

This assignment makes extensive use of AWS. Please keep that in mind since this might cost you money.

* 2 EC2 machines:
  + 1 t2.large machine, which will host Rancher
  + 1 t2.large machine, which will host Jenkins
  + Please use Ubuntu on all of these machines and configure them to have 30 GB of storage.
  + From this point on, I will refer to them as the Rancher instance and the Jenkins instance.
* Elastic IPs
  + Please attach an elastic IP to both of these machines as it makes it easier to work with.
  + A screenshot of a computer

    Description automatically generated
* Please add this setting in “Advanced” when creating your instances (if you are using AWS Learner Lab, which I am not using):
  + A screenshot of a computer

    Description automatically generated
* Please proceed without a key, as we will be using Instance Connect.
  + A screenshot of a computer screen

    Description automatically generated
* Please create the following security group and apply it to both the Rancher instance and the Jenkins instance.
  + A screenshot of a computer

    Description automatically generated
  + Leave outbound rules as the default.
* After you do all this, you should be ready to proceed to the next step of setting up your EC2 machines. In particular, we need to install some dependencies.

For both machines, you need to execute the following commands. These will install some dependencies, such as Docker, kubectl, and generally just update the machines.

sudo su

sudo apt-get update -y

sudo apt-get upgrade -y

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

sudo snap install kubectl --classic

**Setup Rancher Instance and Cluster**

In order to get Rancher on your t2.large instance, do the following.

Login to the instance using instance connect:

A screenshot of a chat

Description automatically generated

Run all the commands mentioned at the end of the previous section.

sudo su

sudo apt-get update -y

sudo apt-get upgrade -y

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

sudo snap install kubectl –classic

Then, to use Rancher, run the following command:

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

After that, go to the following link to access Rancher:

http://{ec2-public-ip}:80

For example,

http://52.55.20.9:80

It will tell you the site is not secure, but just proceed past the warnings.

A screenshot of a computer

Description automatically generated

After that, you will be met by this UI and login page from Rancher.

A screenshot of a computer

Description automatically generated

Follow the instructions and login to Rancher with the bootstrap password. Create your own password so that you remember the login. The username is always “admin”. For me, I always use “temppass3$”

A screenshot of a computer

Description automatically generated

Let’s create our new cluster. Use RKE2/3 to create a custom cluster.

A screenshot of a computer

Description automatically generated

You don’t need to do much. Just name it “cluster” and press create.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Then, you need to register a control plane, etcd, and worker node. You can do these on separate machines, but let’s just do it here on the Rancher instance since this is a large machine and can handle it. Please select the “insecure” option of the registration command or this won’t work. Copy it and run it on your Rancher instance.

A computer screen with many small colored lines

Description automatically generated with medium confidence

Wait about 5 minutes at this point for everything to be set up properly. During this time, Rancher will show that the cluster is updating. Once 5 minutes has elapsed, you should now see that your new Kubernetes cluster is active.

A black rectangular object with white lines

Description automatically generated

We have our K8s cluster created, named “cluster”, and it has 1 worker node, 1 control plane, and 1 etcd node, all on the same machine as where our cluster is.

The last step that you should do is download the kubeconfig file to your computer, as we will need it here shortly for the Jenkins instance:

This can be done from the “Cluster Management” tab on the left.

A screen shot of a computer

Description automatically generated

This allows us to connect to our cluster from different machines and instances (like our Jenkins instance that we will make after this) using the kubeconfig file.

Congratulations, at this point, you are done with everything regarding the Rancher instance. Rancher has been setup, and you can access it using the elastic IP we assigned to the instance at   
http://{ec2-public-ip}:80

**GitHub Repository Setup**

GitHub Repository: <https://github.com/qasimshahid/qshahid-swe645-asst3/>

Please fork this repository or copy all the files from it and add them to a new repository that you created. This has pretty much everything you need, but you do need to make some changes.

First of all, please fix these settings to be your own. They should utilize the link of your GitHub repository and whatever your username is for Docker Hub and what your image should be called.

For example, here you can see I am using my Docker Hub username. Replace this with your own. Please look through all the files of the GitHub repository and replace it wherever applicable. You need to make sure it is all configured correctly for our pipeline to work without access issues.

A screenshot of a computer program

Description automatically generated

After you do all that, make sure the GitHub repository is public by going into the repository settings:

A screenshot of a computer

Description automatically generated

Please also at this point go into the webhook settings and create a new webhook that points to the following link:

http://{jenkins-public-ip}:8080/github-webhook/  
  
Replace {jenkins-public-ip} with the public elastic IP of your Jenkins instance and use the following settings:

A screenshot of a computer

Description automatically generated

Add this webhook.

Example:

A screenshot of a computer

Description automatically generated

This webhook will allow us to automatically build and deploy our application whenever we push to main branch on our GitHub repository. GitHub will recognize our push event and send a POST request to our Jenkins server, which will then kick off the pipeline.

Congratulations, at this point, you are done with setting up your GitHub repo. Push all the code changes and modifications you made and save all of these setting.

**Jenkins Instance Setup**

So, this is the other EC2 machine we created at the start, and this is where we will host our Jenkins server. This will allow us to automate our builds and make a pipeline that gets triggered whenever we push to the main branch of our GitHub repository.

Please connect to the instance using Instance Connect:

A screenshot of a computer

Description automatically generated

After that, please execute these commands if you haven’t already. These commands are required for both instances:

sudo su

sudo apt-get update -y

sudo apt-get upgrade -y

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

sudo snap install kubectl --classic

After that, please execute the following commands or follow the instructions from the official Jenkins site on how to download for Ubuntu (https://www.jenkins.io/doc/book/installing/linux/):

sudo apt install fontconfig openjdk-17-jre -y

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

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

sudo apt-get update

sudo apt-get install jenkins -y

We also need to install Maven so we can build our Spring application.

sudo apt install maven

sudo apt install openjdk-17-jdk

IMPORTANT: Installing Maven might install a different version of Java other than 17. We need to make sure that our machine will be using JDK 17 and not any other JDK version. Please check the java version using:

mvn -version

A computer screen shot of a black screen

Description automatically generated

Make sure it says version 17 or else our pipeline will NOT work.

If it doesn’t say version 17, use these commands to find the path to JDK 17 which we installed earlier and point your Java to that directory.

# IMPORTANT - MAKE SURE JENKINS INSTANCE HAS JAVA 17 SET AS THE DEFAULT JAVA

# VERSION

# You can check the default java version using the following command:

update-java-alternatives --list

# If you have multiple versions of Java installed, you can set the default version using the following command:

# This will set Java 17 as the default version. Make sure to replace the path with the correct # one for your system. This is what the path should be if you installed openjdk-17-jdk

sudo update-java-alternatives --set /usr/lib/jvm/java-1.17.0-openjdk-amd64

(This command is also SUPER IMPORTANT, make sure to execute this one or else builds might fail. If it does not work after your first build and you get some docker permission issues, execute it again)

sudo usermod -a -G docker jenkins

sudo systemctl enable jenkins

sudo systemctl start jenkins

sudo systemctl status jenkins

These commands will install Java and Jenkins. They will also make Docker accessible to Jenkins without needing to do sudo. The last command will also print a password, please store this as we will need it here in a second:

A computer screen with a red circle

Description automatically generated

Now, the server is up. To go to it, use the elastic public IP we created at the start and go to this link:

http://{jenkins-public-ip}:8080

For example:   
<http://54.87.102.0:8080/>

A screenshot of a computer

Description automatically generated

Go the link and Jenkins will ask you for the password from earlier. Paste it in.

Jenkins will now ask you what plugins you want. Select “Install suggested plugins.” We will add some more later on.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Then, create a username and password. I use “admin” and “temppass3$” with my GMU email. After this, keep clicking continue until the setup is complete.

We are now done with the initial setup. You should be on this screen:

A screenshot of a computer

Description automatically generated

Now, go to “Manage Jenkins” so that we can install some plugins.

A screenshot of a computer

Description automatically generated

We need to install three plugins:

Docker plugin

Kubernetes CLI plugin

Docker Pipeline plugin

A close-up of a white paper

Description automatically generated

A screenshot of a chat

Description automatically generated

Install these and wait for them to finish downloading. We need these to build our image and deploy to our cluster.

After these are complete, go back to “Manage Jenkins” but this time, go to Credentials:

A screenshot of a computer

Description automatically generated

We will now add our Docker Hub credentials and the kubeconfig file that you should have saved from the Rancher Instance setup.

First, the kubeconfig file:

Add a new credential to System -> Global credentials.

A screen shot of a computer

Description automatically generated

Make it a “Secret file”

Copy these settings:

ID - kubeconfig\_credentials

File – Select the .yaml file we downloaded earlier when setting up the Rancher Instance. This was the KubeConfig file of our cluster that we downloaded from Rancher.

A screenshot of a computer

Description automatically generated

This will allow us to deploy to our cluster even though we are not on the same instance. Jenkins can use Kubernetes CLI with this file to connect to our cluster on the other EC2 instance.

Second, lets add our Docker Hub credentials:

First, you need to go to Docker Hub and login to your account. Go to “Account settings.”

A screenshot of a computer

Description automatically generated

Go to “Personal access tokens” and create a new token.

A screenshot of a computer

Description automatically generated

Copy these settings:

Access permissions – Read, Write, Delete

A screenshot of a computer

Description automatically generated

Click generate. After that, it will give you a password. Copy this and save it somewhere as we will need it here soon. This will only show once, so be careful and make sure to save it somewhere.

A screenshot of a computer

Description automatically generated

Now, go back to Jenkins and create another credential:

Make it as a “Username with password”. Then, add your Docker Hub username and take the password you just copied from the Personal Access Token and paste it into the password field. Then, give it this id:

dockerhub\_credentials

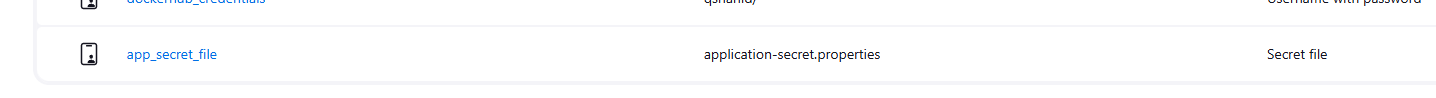
Save this credential. Here is an example:

A screenshot of a computer

Description automatically generated

Now, we also need to add our application-secret.properties file to Jenkins. Follow the same steps you did for the kubeconfig file.

Select your application-secret.properties as the file (this has your database connection details) and use the id “app\_secret\_file”.



Overall, you should have this setup in your credentials in Jenkins:

A screenshot of a computer

Description automatically generated

Finally, we are ready to create our pipeline. Go back to the Jenkins dashboard and create a new item:

A screenshot of a computer

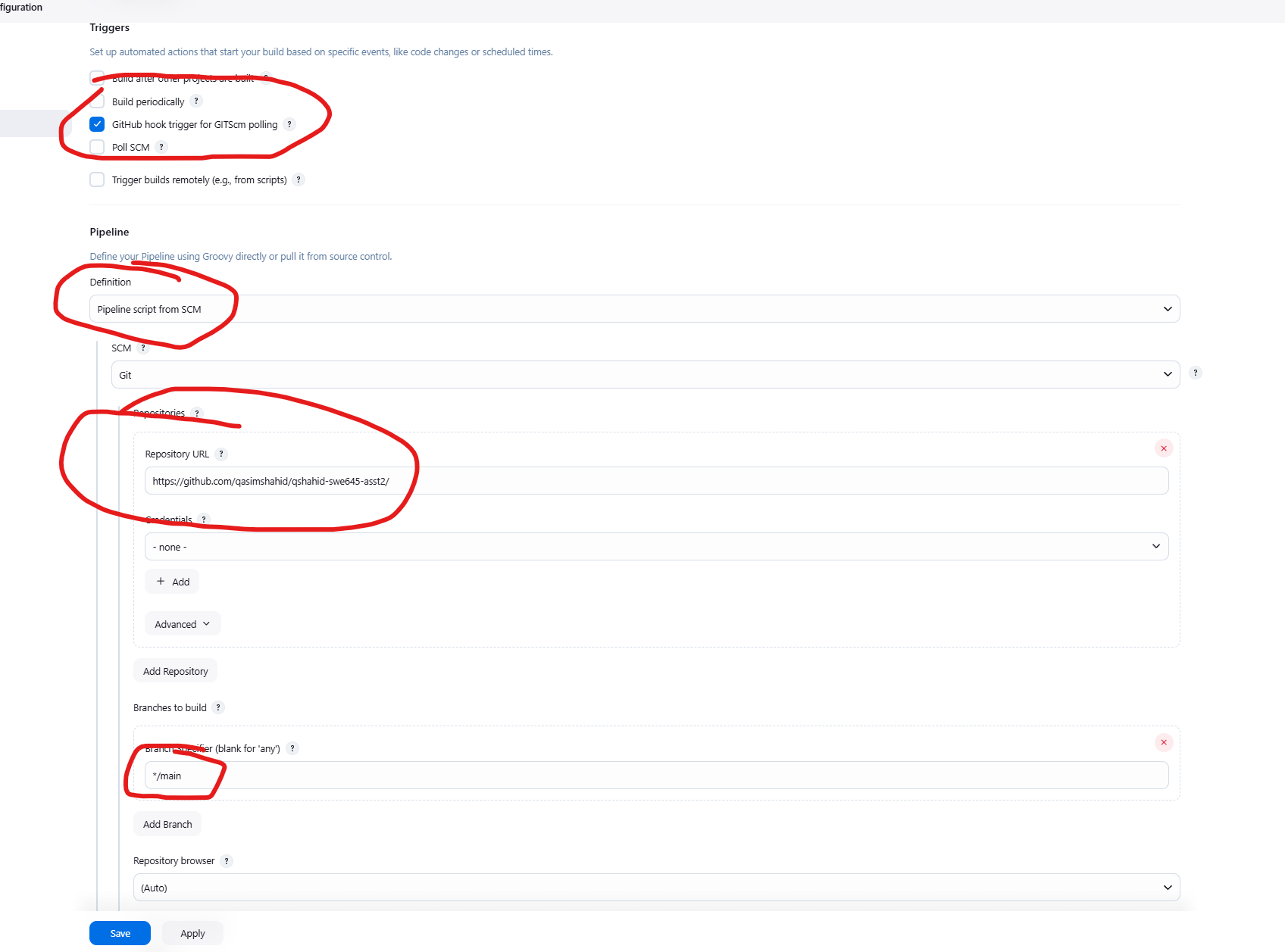
Description automatically generated

Create a new pipeline.

A screenshot of a computer

Description automatically generated

This will give you a bunch of options. We want to copy these settings:



Namely, do the following:

Select “GitHub hook trigger for GITScm polling” as a trigger

Pipeline definition should be from “Pipeline script from SCM”. This means it will use our Jenkinsfile in our GitHub repository. By default, the name of the file is Jenkinsfile, so we don’t need to change it.

Add the link to your GitHub repository.

Change the branch to “main”. This tells Jenkins to build the main branch. This is important because it is no longer named “master,” so make sure you enter “main” here.

After all of that, save the pipeline.

Congratulations, we are ready to do our first build. Everything should be setup properly now.

**Running the Application**

Assuming you followed all the rest of the steps, you should be ready to build and deploy the app to your cluster.

You can do these two ways.

Manually via Jenkins:

A screenshot of a computer

Description automatically generated

Alternatively, if you were able to correctly setup your webhook from your GitHub repository, then you can simply push updates to your main branch and the pipeline will be triggered automatically.

IF YOU HAVE ANY ISSUES WITH THE DOCKER BUILD STEP:

Execute this command and then restart the Jenkins server.

sudo usermod -a -G docker Jenkins

sudo reboot

Once you either push a change to GitHub or manually trigger the build through Jenkins, you should see this in the build output console:

A screenshot of a computer program

Description automatically generated

Here you can see our build succeeding (after a couple mistakes I made when recreating this, if you follow the guide, you should not have these.)

A white background with blue text and black text

Description automatically generated

New image successfully pushed to Docker Hub under my account.

A screenshot of a computer

Description automatically generated

In Rancher, you can see the 3 pods / replicas we created as a part of our deployment.

A screenshot of a computer

Description automatically generated

You can access these at the Rancher Instance’s public elastic IP. The NodePort service I used hosts the API on the 30007 port. So, you can go to:

http://<public\_instance\_ip>:30007/api/surveys/version

For example:

<http://98.82.89.116:30007/api/surveys/version>

Here you can see my API responding via some HTML when we ask for the version.

A screenshot of a web page

Description automatically generated

I’ll now push a small change to GitHub, let’s see what happens. We will change version to 2.0.0.0.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

Let’s go to <http://98.82.89.116:30007/api/surveys/version> again and see if our pipeline deployed our change.

A screenshot of a computer

Description automatically generated

You can see that it worked. We are now on version 2.0.0.0, which we just pushed. Our pipeline listened to the push event and automatically deployed the change to our cluster.

Successful upload to Docker Hub as well.

A screenshot of a computer

Description automatically generated

NOTE: I won’t be testing the API for doing CRUD operations, but this is covered in the demo\_video.mp4 file which will be included with the submission. In that, we go through all the CRUD operations on our APIs and make sure they are being persisted in our database.

References:

<https://docs.docker.com/security/for-developers/access-tokens/>

<https://kubernetes.io/>

<https://plugins.jenkins.io/pipeline-github/>

<https://www.cprime.com/resources/blog/how-to-integrate-jenkins-github/>

<https://www.rancher.com/>

<https://ranchermanager.docs.rancher.com/>

<https://plugins.jenkins.io/kubernetes-cli/>

<https://medium.com/@pratik.941/building-rest-api-using-spring-boot-a-comprehensive-guide-3e9b6d7a8951>

<https://start.spring.io/>

<https://askubuntu.com/questions/740757/switch-between-multiple-java-versions>

https://www.baeldung.com/executable-jar-with-maven#:~:text=To%20build%20a%20jar%2C%20we,%3A8080%2F%20in%20a%20browser.

Lessons Learned:

This assignment, I learned how to deploy REST APIs to a Kubernetes cluster. In the last assignment, we did something very similar for a very simple web server and HTML file, and this assignment, we did basically that but the backend. So combined, these two assignments have taught me how to almost deploy an entire website with a front and back end to Kubernetes. I also learned a lot about how to develop APIs using Spring, and also learned how dependencies are managed with Spring projects (Maven). Overall, I learned a lot about how to create and deploy an API which connects to a database, and then create an automated pipeline that builds and deploys it for me.