

SWE 645 Homework / Assignment 2

Author of document: Qasim Shahid

Date Due: 3/10/2025

GitHub Repository: <https://github.com/qasimshahid/qshahid-swe645-asst2/>

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

In this homework assignment, we are tasked to build a CI/CD pipeline using a very simple HTML page. We containerize a simple survey HTML page 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 docker image defined in the Dockerfile using docker build
3. 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)
4. 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 30001 mapped to the container's port 80, allowing external access at <http://<public-cluster-ip>:30001>.

Please make sure you have a Docker Hub account, an AWS Learner Lab 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.

AWS EC2 Setup

This assignment makes extensive use of AWS. Please use the Learner Lab provided by the professor as this gives you a \$50 budget to work with.

- 2 EC2 machines:
 - 1 t2.large machine, which will host Rancher
 - 1 t2.micro 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.

Elastic IP addresses (2) ⌂ Actions ▾ Allocate Elastic IP address

Allocated IPv4 address ▾	Type ▾	Allocation ID ▾	Reverse DNS record ▲	Associated instance ID ▾
18.210.159.87	Public IP	eipalloc-08d74dbe9f1a00b95	–	i-0a5aa0493b23a8efa 🔗
3.210.232.198	Public IP	eipalloc-0b5c023d6ce260f6d	–	i-009d77b1829bfd303 🔗

-
- Please use the AWS Learner Lab and add this setting in “Advanced” when creating your instances:

▼ **Advanced details** Info

Domain join directory Info

Select ↕ 🔄 Create new directory 🔗

IAM instance profile Info

Select ↕ 🔄 Create new IAM profile 🔗

Select ✓

EMR_EC2_DefaultRole
arn:aws:iam::728762828036:instance-profile/EMR_EC2_DefaultRole

LabInstanceProfile
arn:aws:iam::728762828036:instance-profile/LabInstanceProfile

-
- Please proceed without a key, as we will be using Instance Connect.

▼ **Key pair (login)** Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the

Key pair name - required

Select ↕ 🔄 Create new key pa

Proceed without a key pair (Not recommended) Default value

vockey
Type: rsa

- Please create the following security group and apply it to both the Rancher instance and the Jenkins instance. Inbound rules image is shown first, and then outbound rules.

Details

Security group name
[launch-wizard-1](#)

Security group ID
[sg-0c8d28e561c50c1ae](#)

Description
[launch-wizard-1](#) created 2025-03-08T21:34:11.068Z

VPC ID
[vpc-0acbab5dcda107738](#)

Owner
728762828036

Inbound rules count
5 Permission entries

Outbound rules count
1 Permission entry

[Inbound rules](#)
[Outbound rules](#)
[Sharing - new](#)
[VPC associations - new](#)
[Tags](#)

Inbound rules (5)
[Manage tags](#)
[Edit inbound rules](#)

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
	sgr-0549b3640fb9c0d21	IPv4	SSH	TCP	22	0.0.0.0/0	-
	sgr-097f573f8b6e9dcb	IPv4	HTTP	TCP	80	0.0.0.0/0	-
	sgr-05f8c367cd4a26be	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
	sgr-0895faf11d28d11e1	IPv4	Custom TCP	TCP	30000 - 40000	0.0.0.0/0	-
	sgr-0668445d96ce81f0	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-

Details

Security group name
[launch-wizard-1](#)

Security group ID
[sg-0c8d28e561c50c1ae](#)

Description
[launch-wizard-1](#) created 2025-03-08T21:34:11.068Z

VPC ID
[vpc-0acbab5dcda107738](#)

Owner
728762828036

Inbound rules count
5 Permission entries

Outbound rules count
1 Permission entry

[Inbound rules](#)
[Outbound rules](#)
[Sharing - new](#)
[VPC associations - new](#)
[Tags](#)

Outbound rules (1)
[Manage tags](#)
[Edit outbound rules](#)

<input type="checkbox"/>	Name	Security group rule ID	IP version	Type	Protocol	Port range	Destination	Description
<input type="checkbox"/>	-	sgr-05ec72ce919b34ed2	IPv4	All traffic	All	All	0.0.0.0/0	-

-
- The important thing here is to make sure that we can use ports 8080 (this is where we will access Jenkins) and ports 30000-40000 (specifically, we will be deploying to port 30001 with our NodePort service, so this needs to be accessible). Port 80 also needs to be accessible since this is how we will access Rancher. Overall, just copy my settings and that should be sufficient.

- 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:

EC2 Instance Connect | Session Manager | SSH client | EC2 serial console

Instance ID
i-05649b9f3d5a98b81 (ex)

Connection Type

☒ Connect using EC2 Instance Connect
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

☐ Connect using EC2 Instance Connect Endpoint
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

☒ Public IPv4 address
52.55.20.9

☐ IPv6 address

Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ubuntu.

ubuntu

Note: In most cases, the default username, ubuntu, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel Connect

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
```

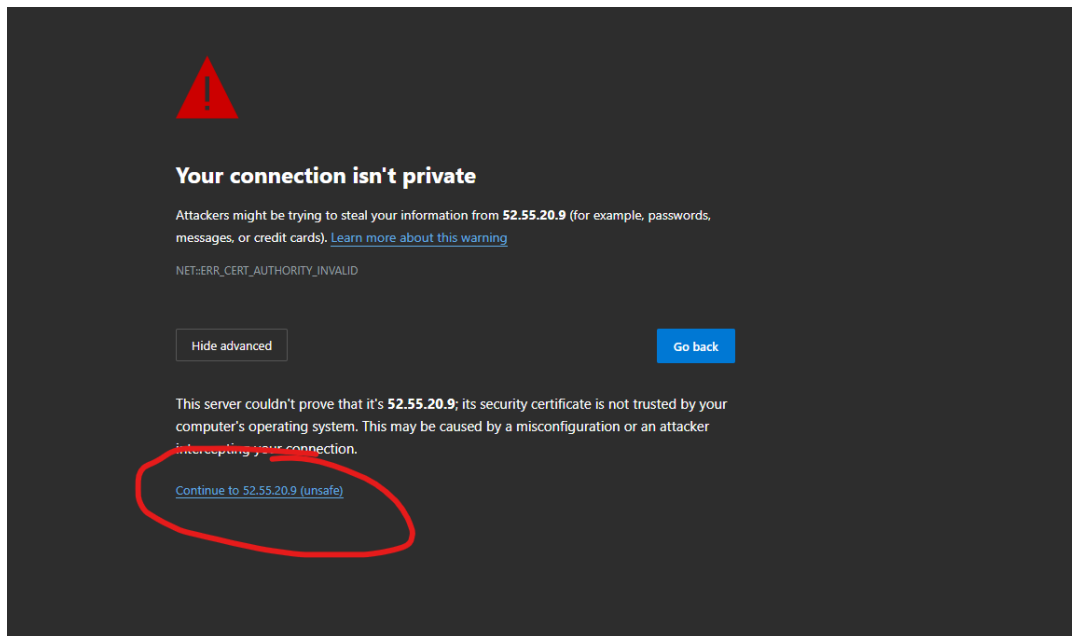
After that, go to the following link to access Rancher:

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

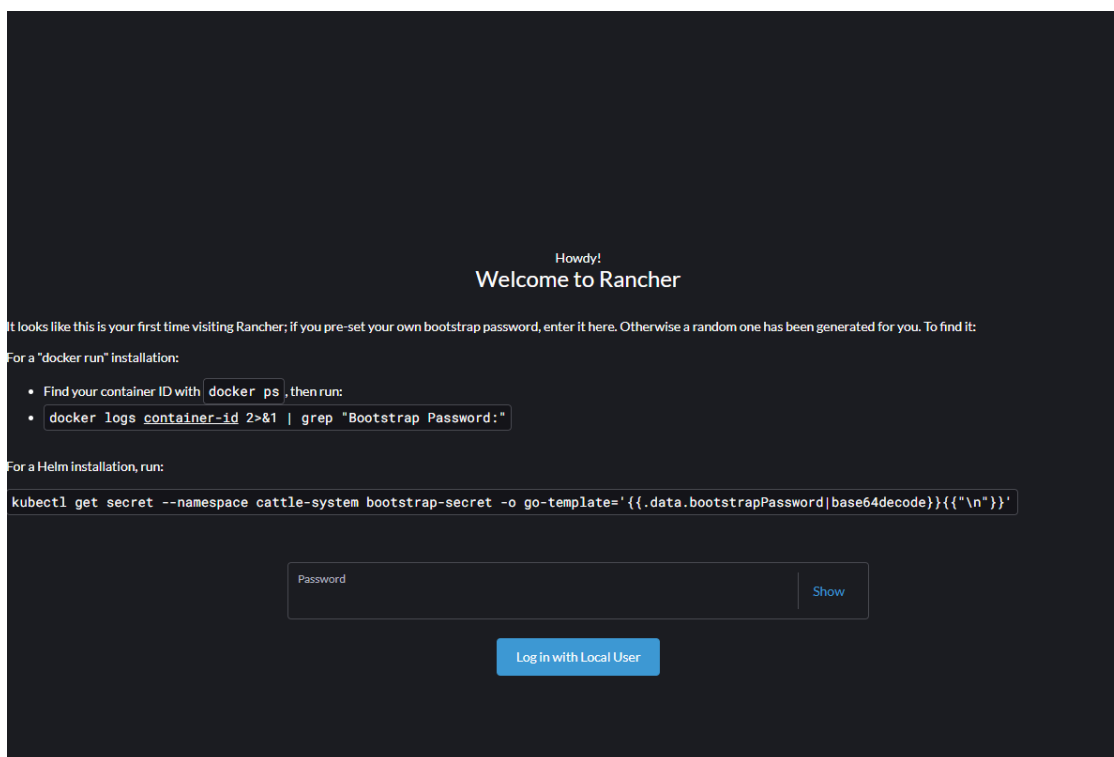
For example,

`http://52.55.20.9:80`

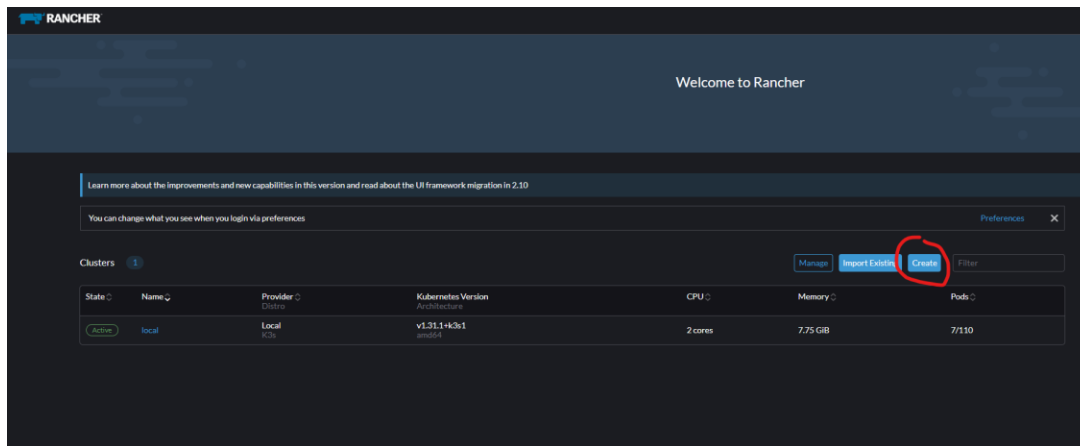
It will tell you the site is not secure, but just proceed past the warnings.



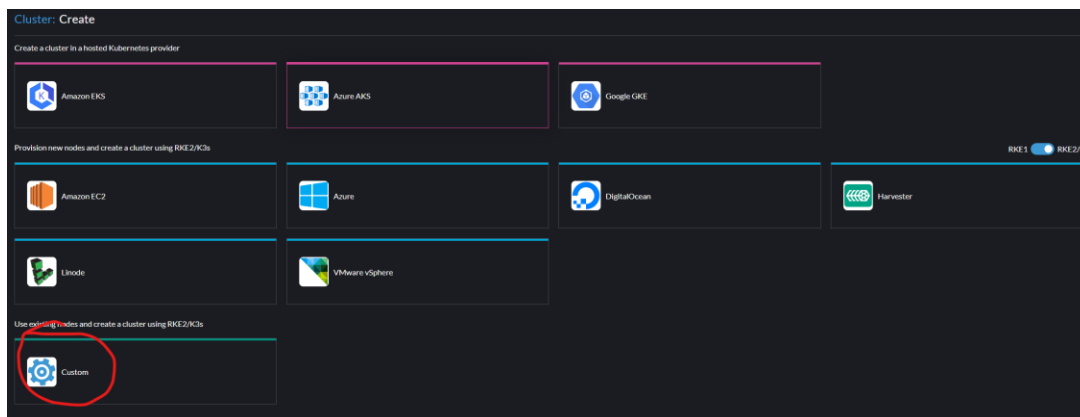
After that, you will be met by this UI and login page from Rancher.



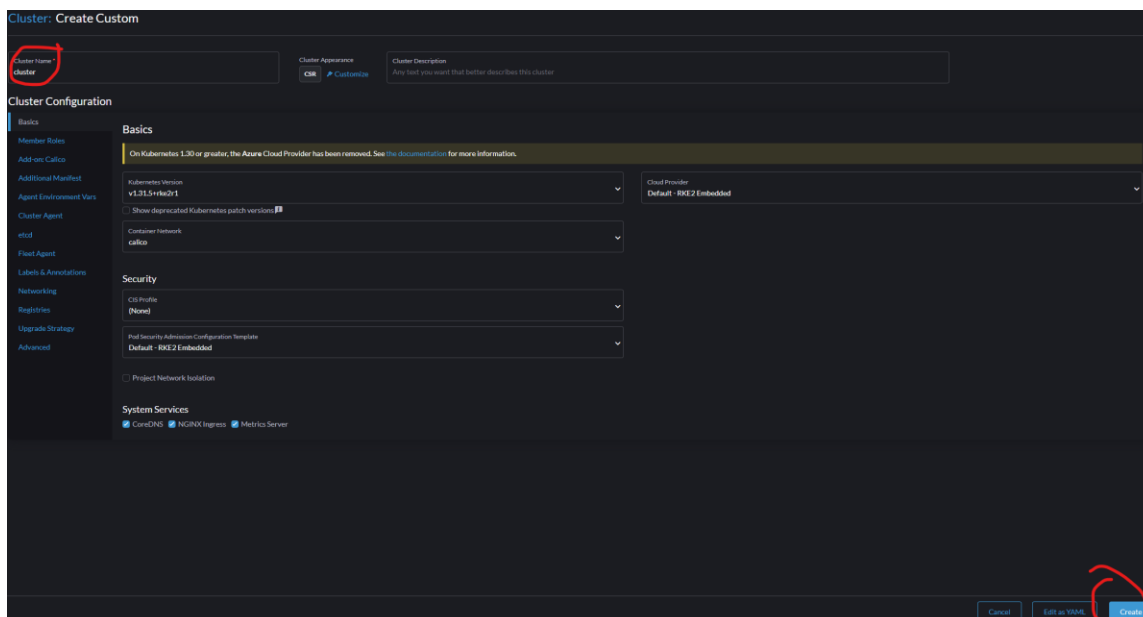
Follow the instructions and login to Rancher with the bootstrap password. Create your own password so that you remember the login. The username is always “admin”. For me, I always use “temppass1234”



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



You don't need to do much. Just name it "cluster" and press create.



Cluster: cluster
Updating
Namespaces: fleet-default
Age: just now
Exit

waiting for at least one control plane, etcd, and worker node to be registered

Provisioner: RKE2

Machines
Provisioning Log
Registration
Snapshots
Conditions
Recent Events
Related Resources

Step 1

Node Role

Choose what roles the node will have in the cluster. The cluster needs to have at least one node with each role.

☒ etcd
☒ Control Plane
☒ Worker

Show Advanced

Step 2

Registration Command

Run this command on each of the existing Linux machines you want to register.

```
curl --insecure -fL https://54.80.152.86/system-agent-install.sh | sudo sh -s -- --server https://54.80.152.86 --label 'cattle.io/os=linux' --token 4pgf8ppjlqzszr6ljfgr4bs2wzbrzgf0k56vtv42kdxrt5blsqcz --ca-checksum c3888f56a79f38f52b54777bf722338547b644658d8e3c8ca9ef46e7cae9f9 --etcd --controlplane --worker
```

☒ Insecure: Select this to skip TLS verification if your server has a self-signed certificate.

Run this command in PowerShell on each of the existing Windows machines you want to register. Windows nodes can only be workers.

The cluster must be up and running with Linux etcd, control plane, and worker nodes before the registration command for adding Windows workers will display.

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.

```
root@ip-172-31-46-212:/home/ubuntu# curl --insecure -fL https://54.80.152.86/system-agent-install.sh | sudo sh -s -- --server https://54.80.152.86 --label 'cattle.io/os=linux' --token 4pgf8ppjlqzszr6ljfgr4bs2wzbrzgf0k56vtv42kdxrt5blsqcz --ca-checksum c3888f56a79f38f52b54777bf722338547b644658d8e3c8ca9ef46e7cae9f9 --etcd --controlplane --worker
% Total    % Received % Xferd Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 33662  0 33662  0  0 3446k    0 --:--:-- --:--:-- --:--:-- 3652k
[INFO] Label: cattle.io/os=linux
[INFO] Role requested: etcd
[INFO] Role requested: controlplane
[INFO] Role requested: worker
[INFO] CA strict verification is set to true
[INFO] Using default agent configuration directory /etc/rancher/agent
[INFO] Using default agent var directory /var/lib/rancher/agent
[INFO] Successfully downloaded CA certificate
[INFO] Value from https://54.80.152.86/cacerts is an x509 certificate
[INFO] Successfully tested Rancher connection
[INFO] Downloading rancher-system-agent binary from https://54.80.152.86/assets/rancher-system-agent-amd64
[INFO] Successfully downloaded the rancher-system-agent binary.
[INFO] Downloading rancher-system-agent-uninstall.sh script from https://54.80.152.86/assets/system-agent-uninstall.sh
[INFO] Successfully downloaded the rancher-system-agent-uninstall.sh script.
[INFO] Generating Cattle ID
[INFO] Successfully downloaded Rancher connection information
[INFO] systemd: Creating service file
[INFO] Creating environment file /etc/systemd/system/rancher-system-agent.env
[INFO] Enabling rancher-system-agent.service
Created symlink /etc/systemd/system/multi-user.target.wants/rancher-system-agent.service → /etc/systemd/system/rancher-system-agent.service.
[INFO] Starting/restarting rancher-system-agent.service
root@ip-172-31-46-212:/home/ubuntu#
```

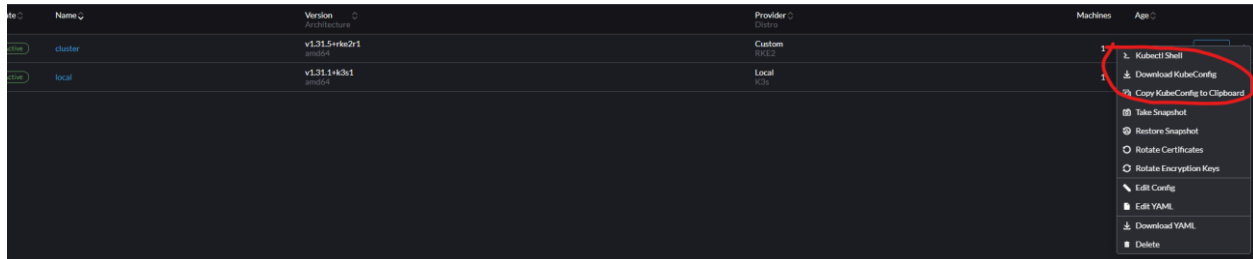
Wait about 5 minutes at this point for everything to be set up properly. During this time, Rancher will show that the cluster is updating. Once 5 minutes has elapsed, you should now see that your new Kubernetes cluster is active.

<input type="checkbox"/> State	Name	Version Architecture
<input checked="" type="checkbox"/> Active	cluster	v1.31.5+rke2r1 amd64
<input checked="" type="checkbox"/> Active	local	v1.31.1+k3s1 amd64

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.



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.

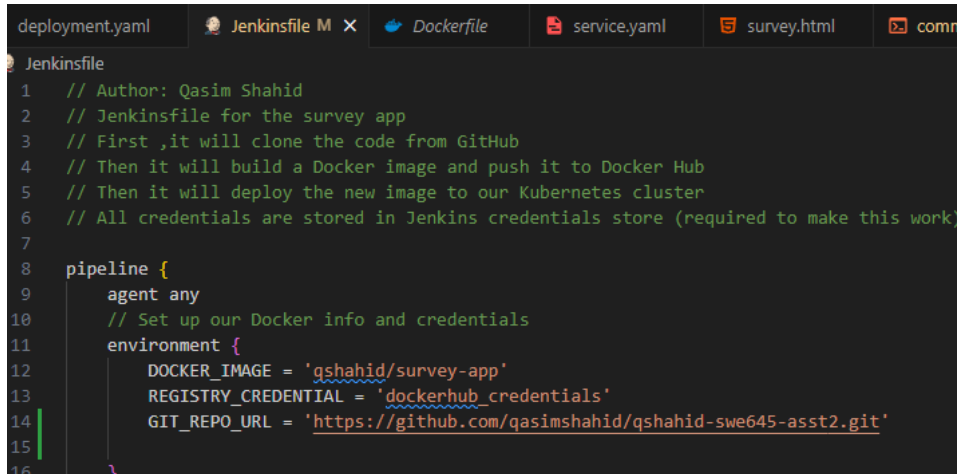
Congratulations, at this point, you are done with everything regarding the Rancher instance. Rancher has been setup, and you can access it using the elastic IP we assigned to the instance at `http://{ec2-public-ip}:80`

GitHub Repository Setup

GitHub Repository: <https://github.com/qasimshahid/qshahid-swe645-asst2/>

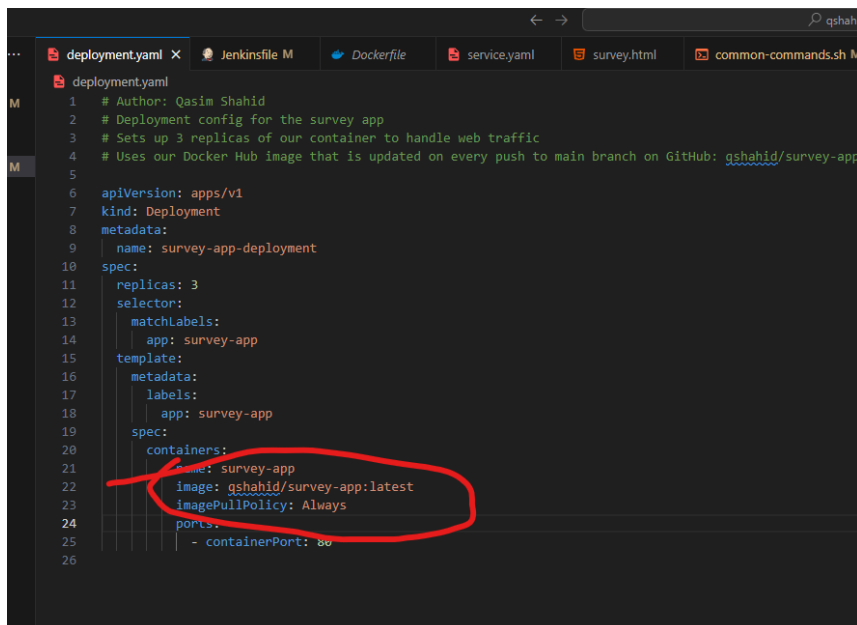
Please fork this repository or copy all the files from it and add them to a new repository that you created. This has pretty much everything you need, but you do need to make some changes.

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.



```
deployment.yaml | Jenkinsfile M X | Dockerfile | service.yaml | survey.html | common-commands.sh M
Jenkinsfile
1 // Author: Qasim Shahid
2 // Jenkinsfile for the survey app
3 // First ,it will clone the code from GitHub
4 // Then it will build a Docker image and push it to Docker Hub
5 // Then it will deploy the new image to our Kubernetes cluster
6 // All credentials are stored in Jenkins credentials store (required to make this work)
7
8 pipeline {
9   agent any
10  // Set up our Docker info and credentials
11  environment {
12    DOCKER_IMAGE = 'qshahid/survey-app'
13    REGISTRY_CREDENTIAL = 'dockerhub_credentials'
14    GIT_REPO_URL = 'https://github.com/qasimshahid/qshahid-swe645-asst2.git'
15  }
16 }
```

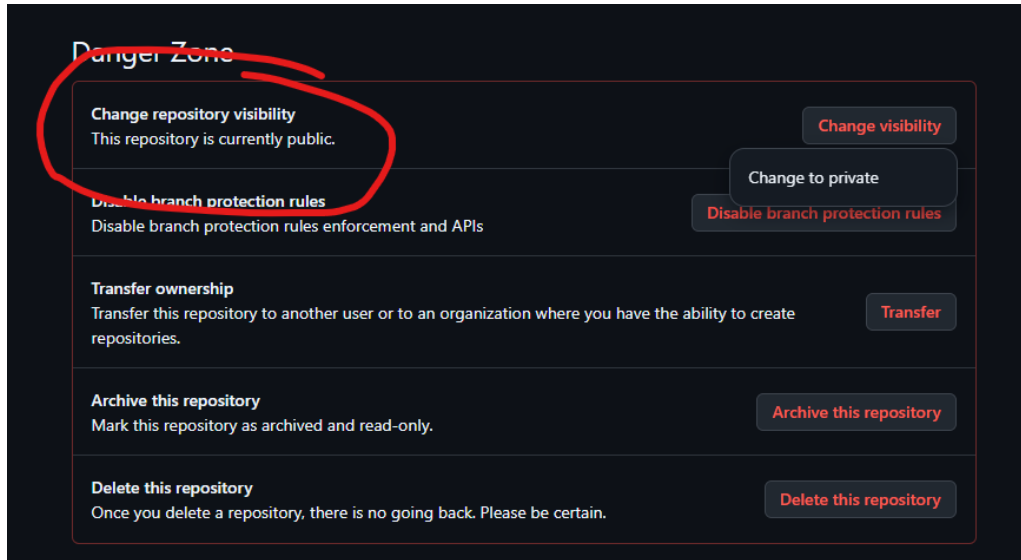
Don't worry about dockerhub_credentials value, we will do that together later on.



```
deployment.yaml | Jenkinsfile M | Dockerfile | service.yaml | survey.html | common-commands.sh M
deployment.yaml
1 # Author: Qasim Shahid
2 # Deployment config for the survey app
3 # Sets up 3 replicas of our container to handle web traffic
4 # Uses our Docker Hub image that is updated on every push to main branch on GitHub: qshahid/survey-app
5
6 apiVersion: apps/v1
7 kind: Deployment
8 metadata:
9   name: survey-app-deployment
10 spec:
11   replicas: 3
12   selector:
13     matchLabels:
14       app: survey-app
15   template:
16     metadata:
17       labels:
18         app: survey-app
19     spec:
20       containers:
21       - name: survey-app
22         image: qshahid/survey-app:latest
23         imagePullPolicy: Always
24         ports:
25         - containerPort: 80
26
```

Change this value in the deployment.yaml file to be whatever your Docker Hub username is plus the name you give to the image.

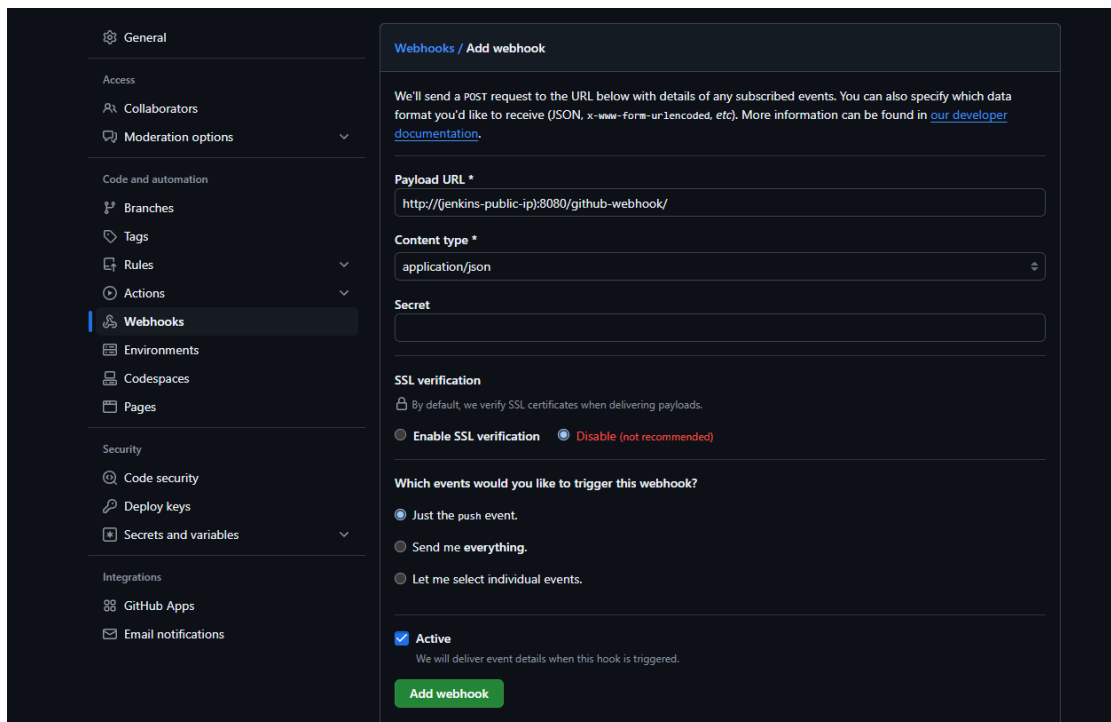
After you do all that, make sure the GitHub repository is public by going into the repository settings:



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

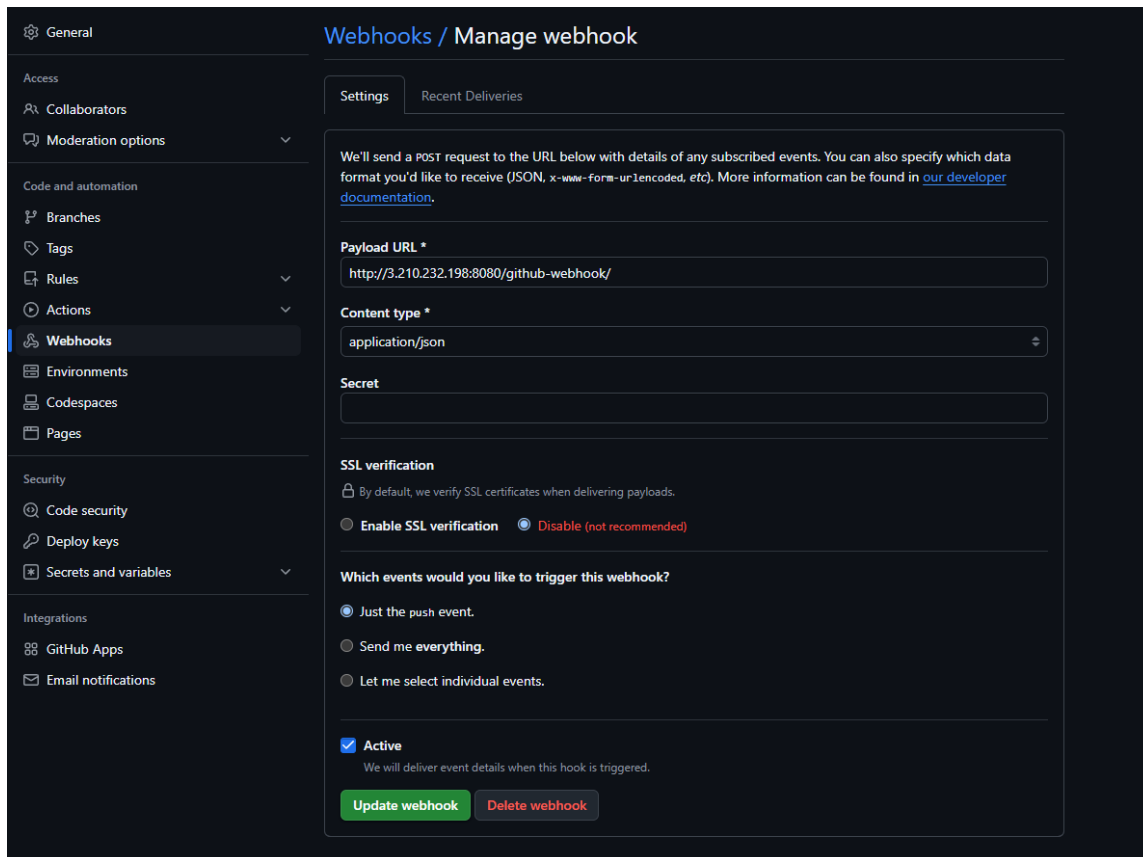
`http://{jenkins-public-ip}:8080/github-webhook/`

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



Add this webhook.

Example:



The screenshot shows the GitHub 'Manage webhook' interface. On the left is a sidebar with navigation links: General, Access (Collaborators, Moderation options), Code and automation (Branches, Tags, Rules, Actions), Webhooks (selected), Environments, Codespaces, Pages, Security (Code security, Deploy keys, Secrets and variables), and Integrations (GitHub Apps, Email notifications). The main panel is titled 'Webhooks / Manage webhook' and has two tabs: 'Settings' (active) and 'Recent Deliveries'. The 'Settings' tab contains the following fields: a descriptive paragraph about the webhook, a 'Payload URL' field with the value 'http://3.210.232.198:8080/github-webhook/', a 'Content type' dropdown menu set to 'application/json', a 'Secret' text field, an 'SSL verification' section with 'Disable (not recommended)' selected, and a 'Which events would you like to trigger this webhook?' section with 'Just the push event.' selected. At the bottom, the 'Active' checkbox is checked, and there are 'Update webhook' and 'Delete webhook' buttons.

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 settings.

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.

This machine uses t2.micro as that is all we need. Remember to use 30 gb of storage and use a security group that has an inbound rule to accept all traffic for port 8080. This is all covered earlier, and you should have already completed this.

Please connect to the instance using Instance Connect:

EC2 Instance Connect | Session Manager | SSH client | EC2 serial console

Instance ID
i-0910c6ae322c31abd (jenk)

Connection Type

☒ Connect using EC2 Instance Connect
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.

☐ Connect using EC2 Instance Connect Endpoint
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

☒ Public IPv4 address
54.87.102.0

☐ IPv6 address
-

Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ubuntu.

Q ubuntu X

Note: In most cases, the default username, ubuntu, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel Connect

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
```

(This command is super important, make sure to execute this one or else builds might fail.)

```
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:

```
root@ip-172-31-91-46:/home/ubuntu# ls
get-docker.sh
root@ip-172-31-91-46:/home/ubuntu# sudo usermod -a -G docker jenkins
root@ip-172-31-91-46:/home/ubuntu# sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
root@ip-172-31-91-46:/home/ubuntu# sudo systemctl start jenkins
root@ip-172-31-91-46:/home/ubuntu# sudo systemctl status jenkins
* jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
   Active: active (running) since Mon 2025-03-10 11:28:15 UTC; 26s ago
     Main PID: 3172 (java)
       Tasks: 43 (limit: 1129)
      Memory: 276.3M (peak: 330.0M)
         CPU: 15.35s
    CGroup: /system.slice/jenkins.service
            └─172 /usr/bin/java -Djava.net.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

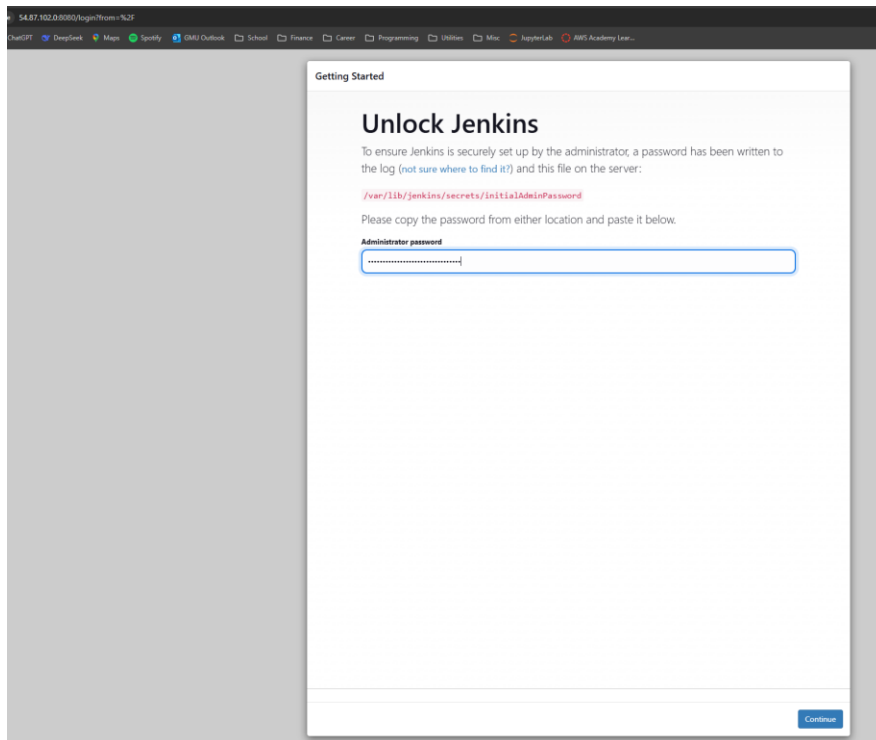
Mar 10 11:28:09 ip-172-31-91-46 jenkins[3172]: See https://www.jenkins.io/doc/book/installing/linux/
Mar 10 11:28:09 ip-172-31-91-46 jenkins[3172]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Mar 10 11:28:09 ip-172-31-91-46 jenkins[3172]: *****
Mar 10 11:28:09 ip-172-31-91-46 jenkins[3172]: *****
Mar 10 11:28:09 ip-172-31-91-46 jenkins[3172]: *****
Mar 10 11:28:15 ip-172-31-91-46 jenkins[3172]: 2025-03-10 11:28:15.211+0000 [id=36] INFO Jenkins.InitReactorRunner$1$onAttained: Completed initialization
Mar 10 11:28:15 ip-172-31-91-46 jenkins[3172]: 2025-03-10 11:28:15.240+0000 [id=37] INFO Hudson.lifecycle.Lifecycle$onReady: Jenkins is fully up and running
Mar 10 11:28:15 ip-172-31-91-46 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Mar 10 11:28:15 ip-172-31-91-46 jenkins[3172]: 2025-03-10 11:28:15.531+0000 [id=46] INFO h.m.DownloadService$Downloadable$load: Obtained the updated data file for Hudson.tasks.Maven.MavenInstaller
Mar 10 11:28:15 ip-172-31-91-46 jenkins[3172]: 2025-03-10 11:28:15.534+0000 [id=46] INFO Hudson.util.Retrier$Start: Performed the action check updates server successfully at the attempt #1
root@ip-172-31-91-46:/home/ubuntu#
```

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>

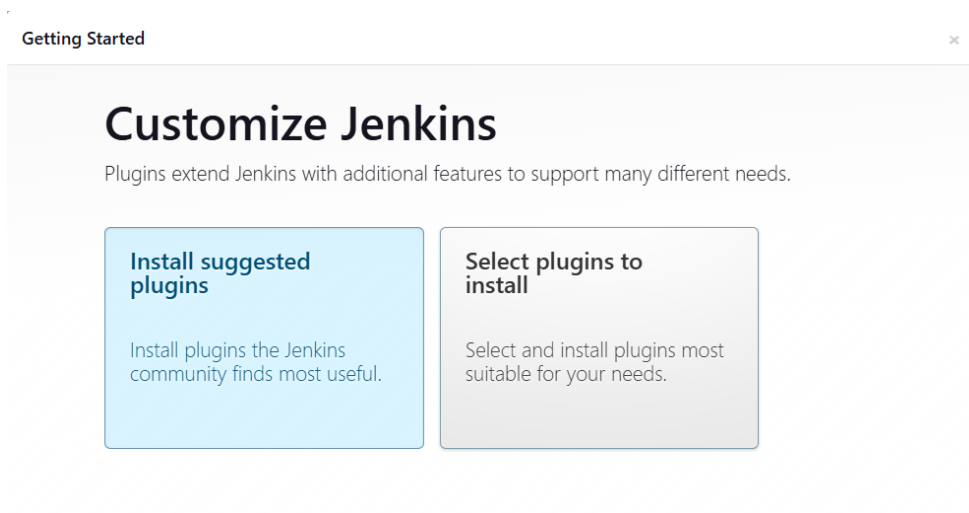
For example:

<http://54.87.102.0:8080/>



Go to 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.



Getting Started

Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

Jenkins 2.492.2 [Skip and continue as admin](#) [Save and Continue](#)

Then, create a username and password. I use “admin” and “temppass1234” 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:

Jenkins

Dashboard >

+ New Item

- Build History
- Manage Jenkins**
- My Jenkins

Build Queue ▼
No builds in the queue.

Build Executor Status 0/2 ▼

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

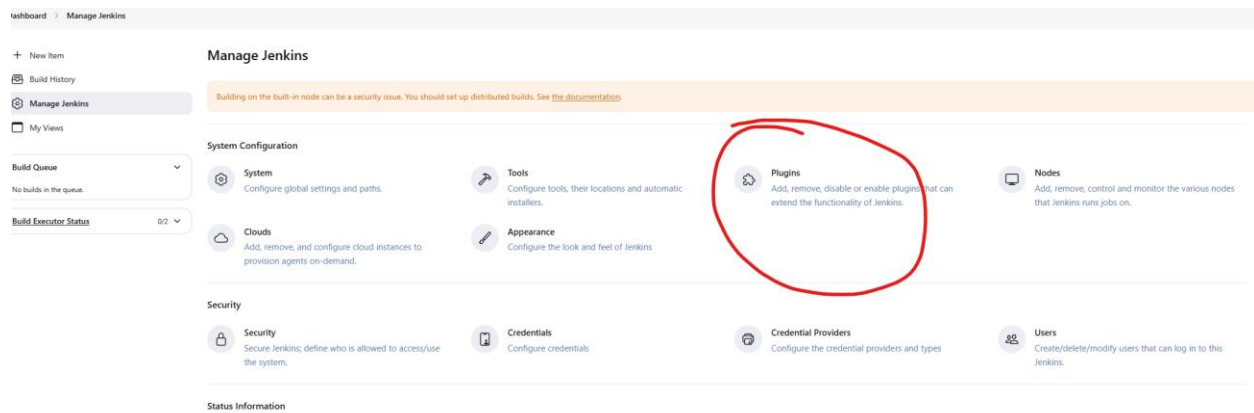
Start building your software project

Create a job +

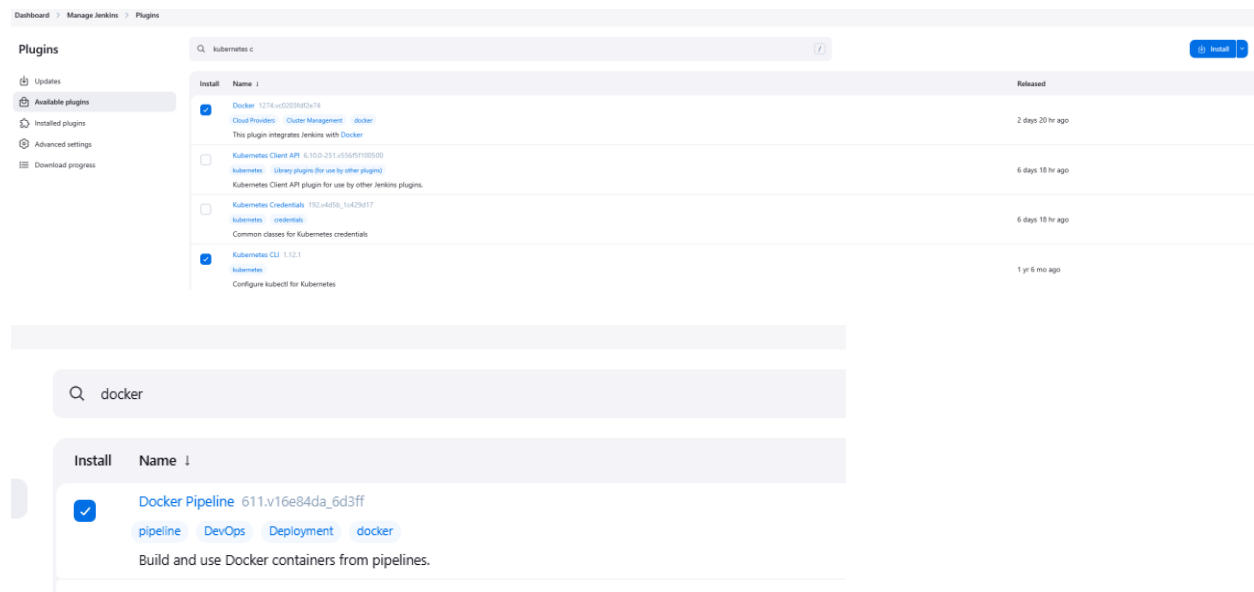
Set up a distributed build

- Set up an agent 📄
- Configure a cloud ☁
- Learn more about distributed builds ?

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

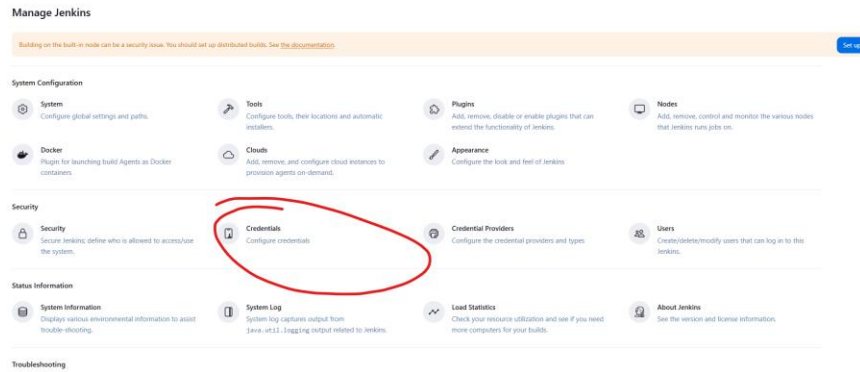


We need to install three plugins:
Docker plugin
Kubernetes CLI plugin
Docker Pipeline plugin



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:



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.



Make it a “Secret file”

Copy these settings:

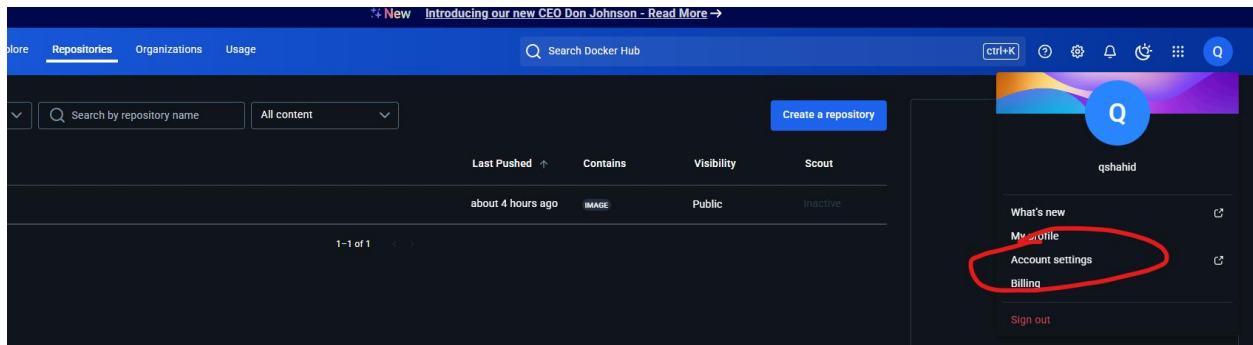
ID - kubeconfig_credentials

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.

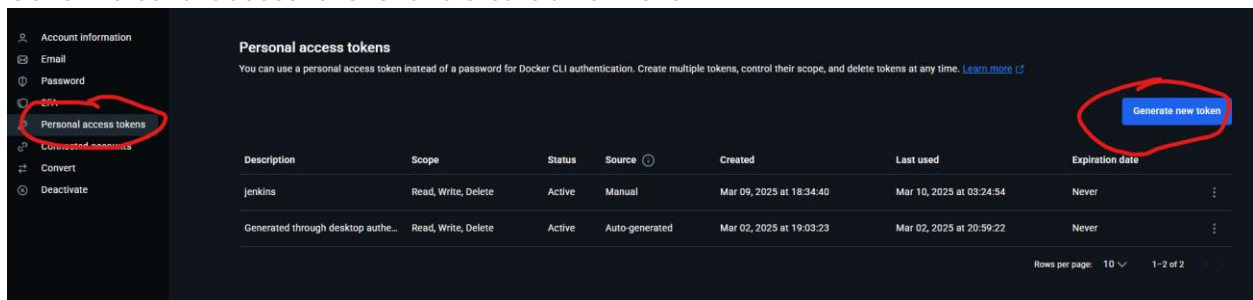
This will allow us to deploy to our cluster even though we are not on the same instance. Jenkins can use Kubernetes CLI with this file to connect to our cluster on the other EC2 instance.

Second, lets add our Docker Hub credentials:

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

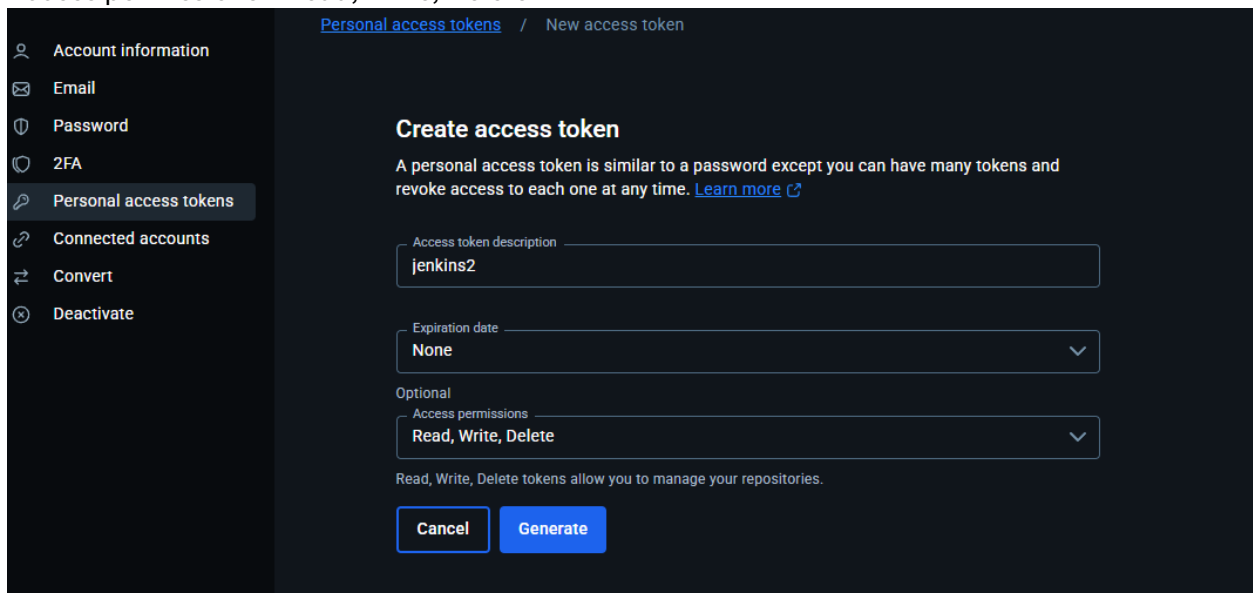


Go to “Personal access tokens” and create a new token.

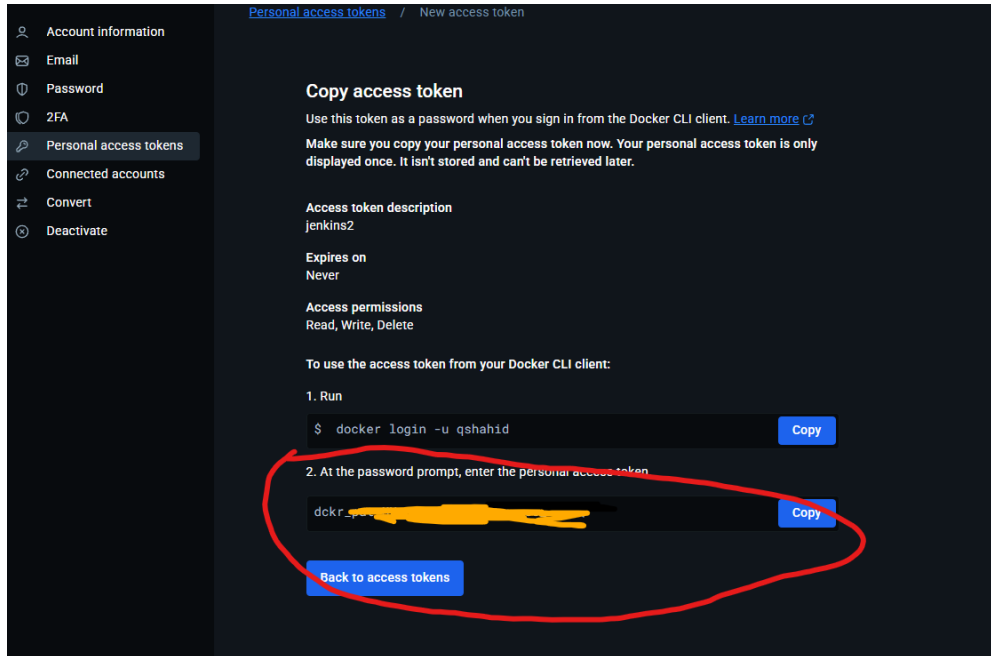


Copy these settings:

Access permissions – Read, Write, Delete



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.



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:

New credentials

Kind
Username with password

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

Username ?
qshahid

☐ Treat username as secret ?

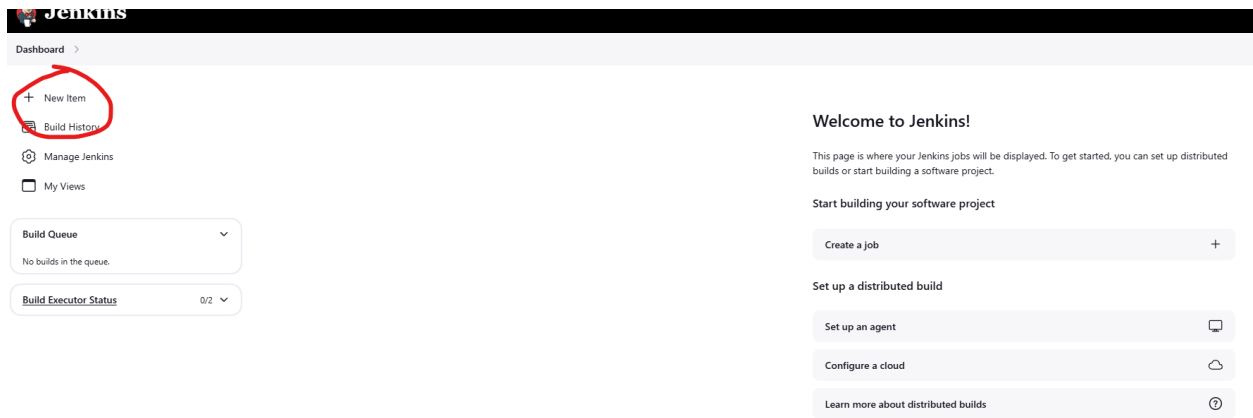
Password ?
.....

ID ?
dockerhub_credentials

Description ?

[Create](#)

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



Create a new pipeline.

New Item

Enter an item name

pipeline

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.



Organization Folder

Creates a set of multibranch project subfolders by scanning for repositories.

OK

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

The screenshot shows the Jenkins configuration interface. In the 'Triggers' section, the checkbox for 'GitHub hook trigger for GITScm polling' is selected and circled in red. In the 'Pipeline' section, the 'Definition' dropdown is set to 'Pipeline script from SCM' and is circled in red. Below this, the 'SCM' dropdown is set to 'Git'. The 'Repository URL' field contains 'https://github.com/qasimshahid/qshahid-swe645-asst2/' and is circled in red. The 'Branches to build' section has a dropdown set to '*/main', which is also circled in red. At the bottom, there are 'Save' and 'Apply' buttons.

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.

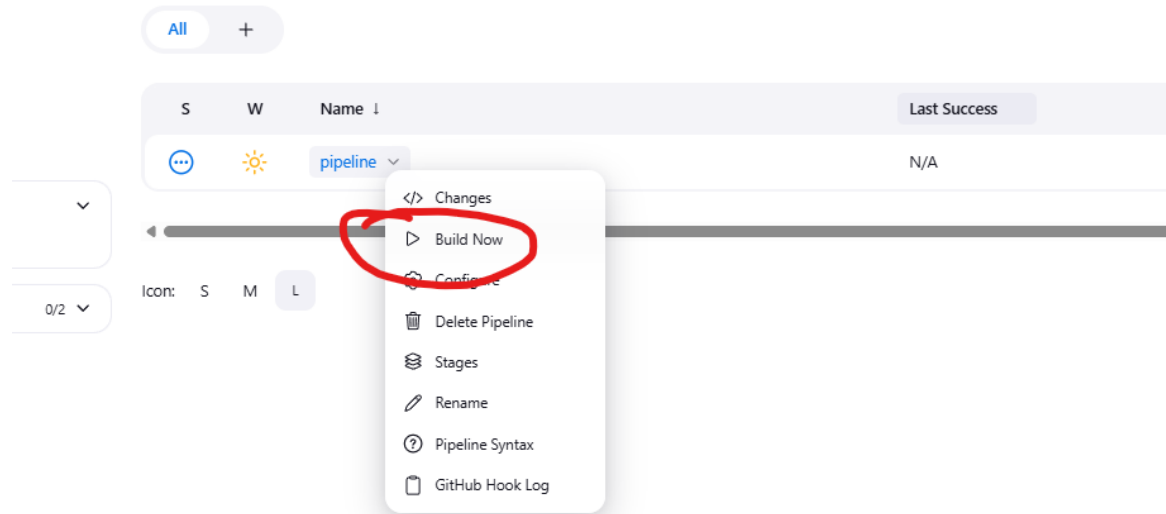
Congratulations, we are ready to do our first build. Everything should be setup properly now.

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:



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.

You can make changes to the survey.html, just as adding a timestamp, to do this.

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:

```
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withDockerRegistry
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy to Kubernetes)
[Pipeline] script
[Pipeline] {
[Pipeline] withKubeConfig
[Pipeline] {
[Pipeline] sh
+ kubectl apply -f deployment.yaml
deployment.apps/survey-app-deployment created
[Pipeline] sh
+ kubectl apply -f service.yaml
service/survey-app-service created
[Pipeline] sh
+ kubectl rollout restart deployment/survey-app-deployment
deployment.apps/survey-app-deployment restarted
[Pipeline] }
[kubernetes-cli] kubectl configuration cleaned up
[Pipeline] // withKubeConfig
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] cleansws
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] done
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Status

Changes

Build Now

Configure

Delete Pipeline

Stages

Rename

Pipeline Syntax

GitHub Hook Log

pipeline

Permalinks

- Last build (#4), 1 min 47 sec ago
- Last stable build (#4), 1 min 47 sec ago
- Last successful build (#4), 1 min 47 sec ago
- Last failed build (#3), 3 min 57 sec ago
- Last unsuccessful build (#3), 3 min 57 sec ago
- Last completed build (#4), 1 min 47 sec ago

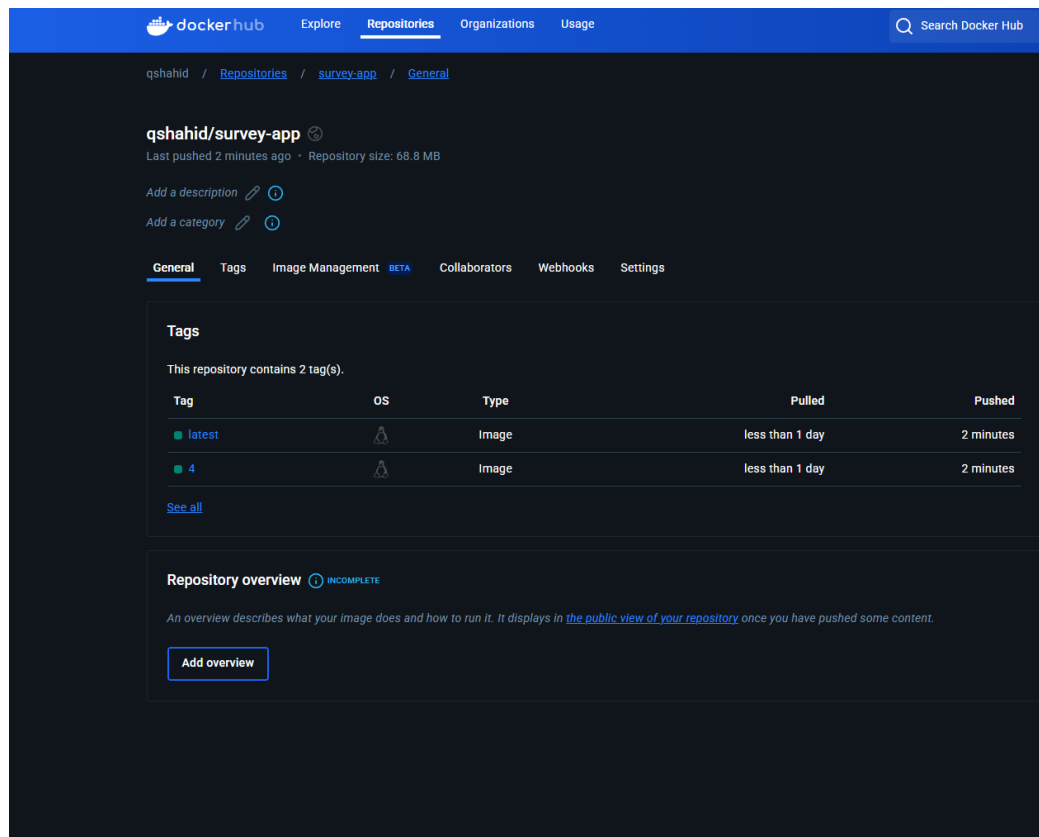
Builds

Filter

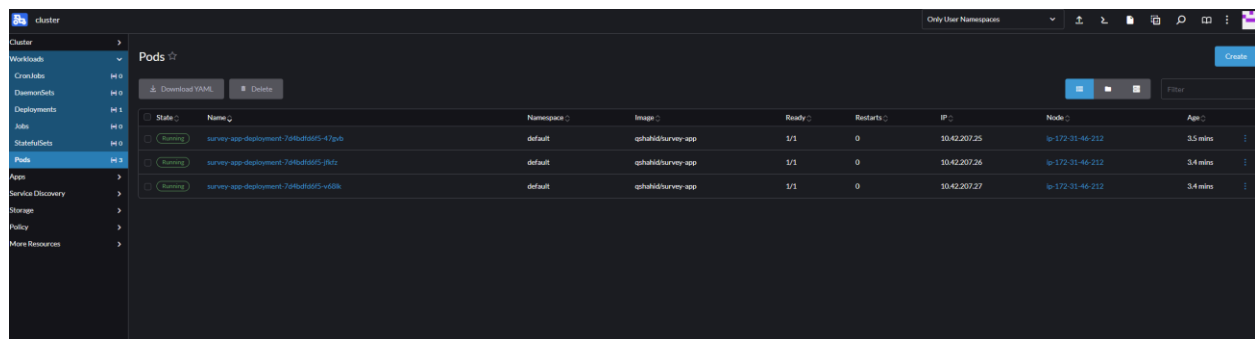
Today

#4 12:37 PM

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.)



New image successfully pushed to Docker Hub under my account.



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

You can access these at the Rancher Instance's public elastic IP. The NodePort service I used hosts the website on the 30001 port. So you can go to:

<http://<public-cluster-ip>:30001>

For example:

<http://54.80.152.86:30001>

Net secure 54.80.152.86:30001

Student Survey Form 3/10/2025

First Name: Last Name:

Street Address:

City: State: ZIP:

Telephone: Email:

Date of Survey:

What did you like most about the campus?

☐ Students

☐ Location

☐ Campus

☐ Atmosphere

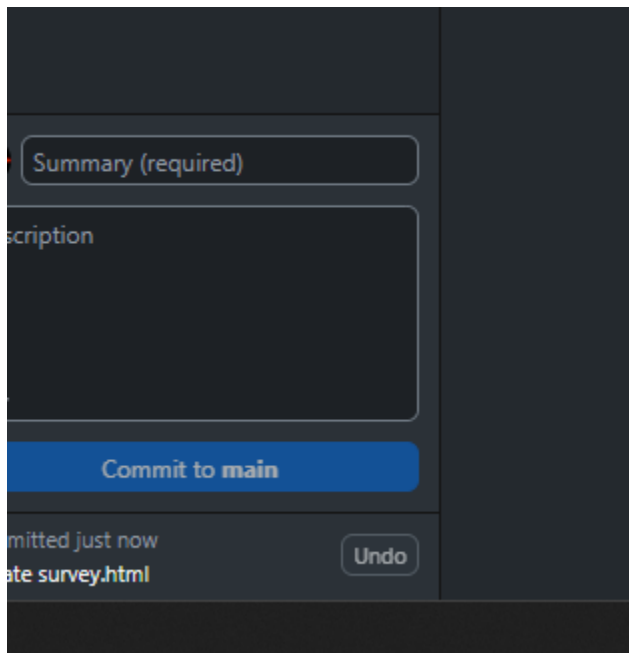
☐ Dorm Rooms

☐ Sports

How did you become interested in the university?

Here you can see my site.

I'll now push a small change to GitHub, let's see what happens.



Dashboard > pipeline > GitHub Hook Log

Status

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline

📅 Stages

✎ Rename

❓ Pipeline Syntax

📄 GitHub Hook Log

Builds

Filter

Today

#5 12:45 PM

#4 12:37 PM

#3 12:35 PM

#2 12:32 PM

#1 12:27 PM

Last GitHub Push

```
Started on Mar 10, 2025, 12:44:57 PM
Started by event from 140.82.115.60 on http://54.87.102.0:8080/github-webhook/ on Mon Mar 10 12:44:56 UTC 2025
Using strategy: Default
[poll] Last Built Revision: Revision e594667ee5e9ae8b69e2a33f7ae5c43504357487 (refs/remotes/origin/main)
The recommended git tool is: NONE
No credentials specified
> git --version # timeout=10
> git --version # 'git version 2.43.0'
> git ls-remote -h -- https://github.com/qasimshahid/qshahid-swe645-asst2/ # timeout=10
Found 1 remote heads on https://github.com/qasimshahid/qshahid-swe645-asst2/
[poll] Latest remote head revision on refs/heads/main is: 6fd8699394088e3876d09c12db71e2991fae8542
Using strategy: Default
The recommended git tool is: NONE
No credentials specified
> git --version # timeout=10
> git --version # 'git version 2.43.0'
> git ls-remote -h -- https://github.com/qasimshahid/qshahid-swe645-asst2.git # timeout=10
Found 1 remote heads on https://github.com/qasimshahid/qshahid-swe645-asst2.git
[poll] Latest remote head revision on refs/heads/main is: 6fd8699394088e3876d09c12db71e2991fae8542
Done. Took 0.65 sec
Changes found
```

Student Survey Form 3/10/2025 9 AM

First Name:		Last Name:	
<input type="text"/>		<input type="text"/>	
Street Address:			
<input type="text"/>			
City:	State:	ZIP:	
<input type="text"/>	<input type="text"/>	<input type="text"/>	
Telephone:		Email:	
<input type="text"/>		<input type="text"/>	

Ignore the odd timing, I did this at 9 AM on 3/10/2025. You can see, pushing to main branch on my GitHub repository uses the webhook and triggers the build on the Jenkins server automatically.

That's it! Thanks for reading.