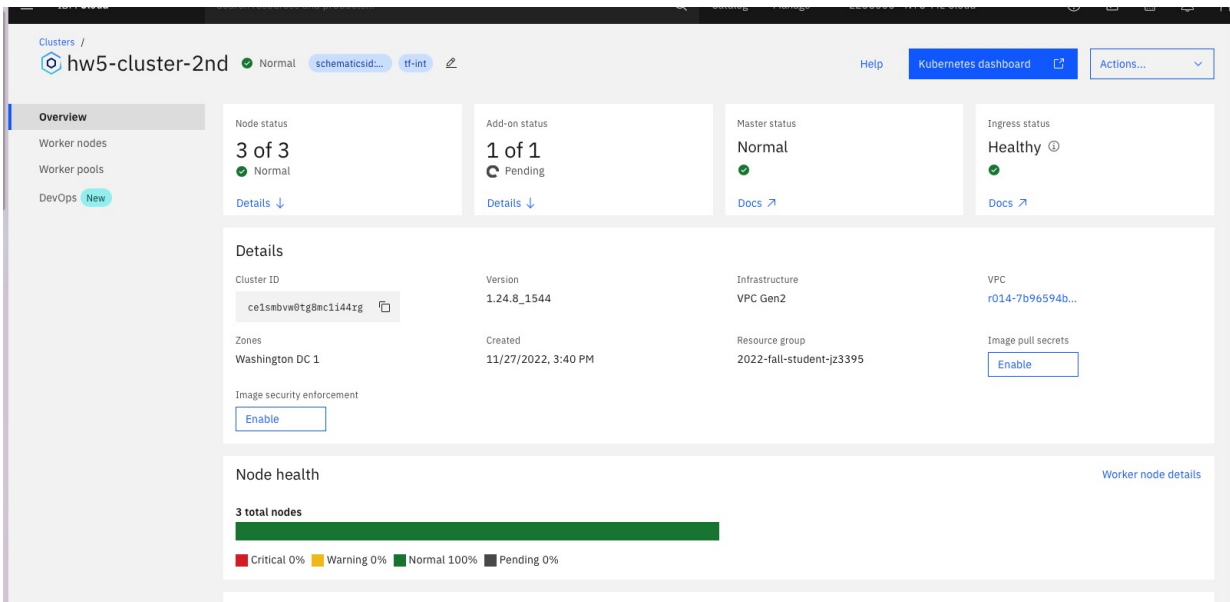


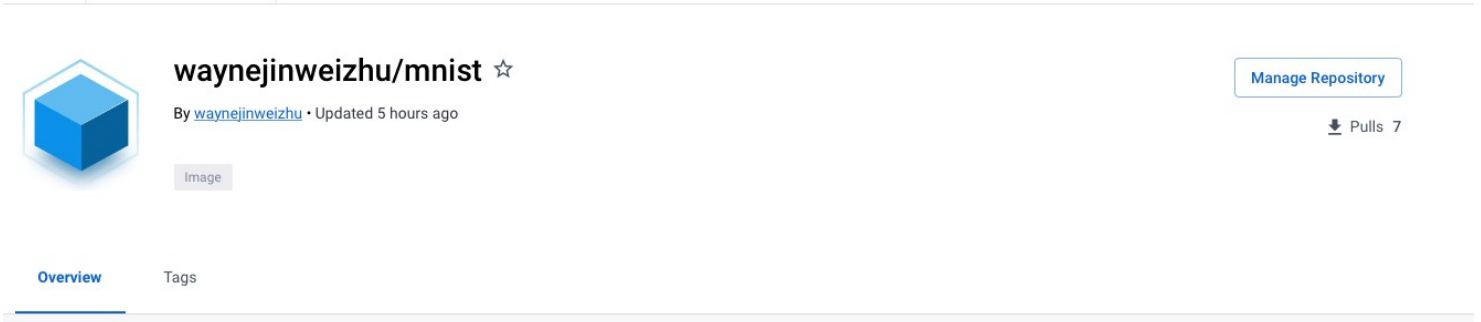
Steps to launch the k8s cluster apps:

1. I followed the instruction outlined in slide 9 to initialize a VPC and a K8s cluster on IBM Cloud.



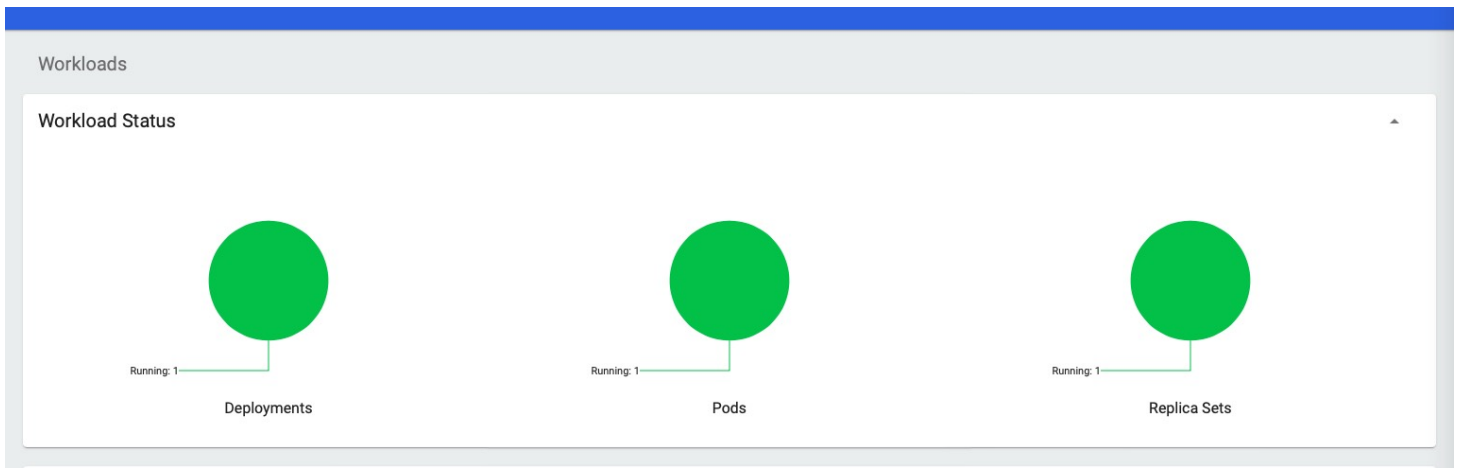
2. I modified the Dockerfile of HW3 to train the model locally in a docker container on my VM as in HW3.

3. I push that image, which takes up 5GB, to Docker Hub.



4. Upon encountering Ingress status warning and PullImageOff error during deployment, I had to restart my k8s cluster in 1 zone and attach my subnet to a public gateway.

5. Then, I deploy my mnist container on k8s using the cloud interface, so it saves me the time to handwrite a k8s manifest yaml file.



7. I couldn't find the model.save output mnist_cnn.pt for a long time. Eventually, I realized it's in the container's directory, and once the container finishes running it's gone. So I let my python program sleep while I docker copy the file into my VM. This approach works.

8. I create a separate directory called mnistinference, revised the dockerfile, added flask process and deployed it on my local VM. The user will see this:

The screenshot shows a web browser window with the address bar displaying '127.0.0.1'. Below the address bar, there is a text input field with the placeholder text 'Enter Any Input to start inference' and a 'Login' button. The input field contains the text 'firstname'.

9. Once you enter anything, the app will load the mnist_cnn.pt and conduct the rest of the inference.

```
10.0.2.2 - - [28/Nov/2022 22:38:20] "GET / HTTP/1.1" 200 -
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/MNIST/raw/train-images-idx3-ubyte.gz
100.0%
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/MNIST/raw/train-labels-idx1-ubyte.gz
100.0%
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw/t10k-images-idx3-ubyte.gz
100.0%
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100.0%
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/MNIST/raw

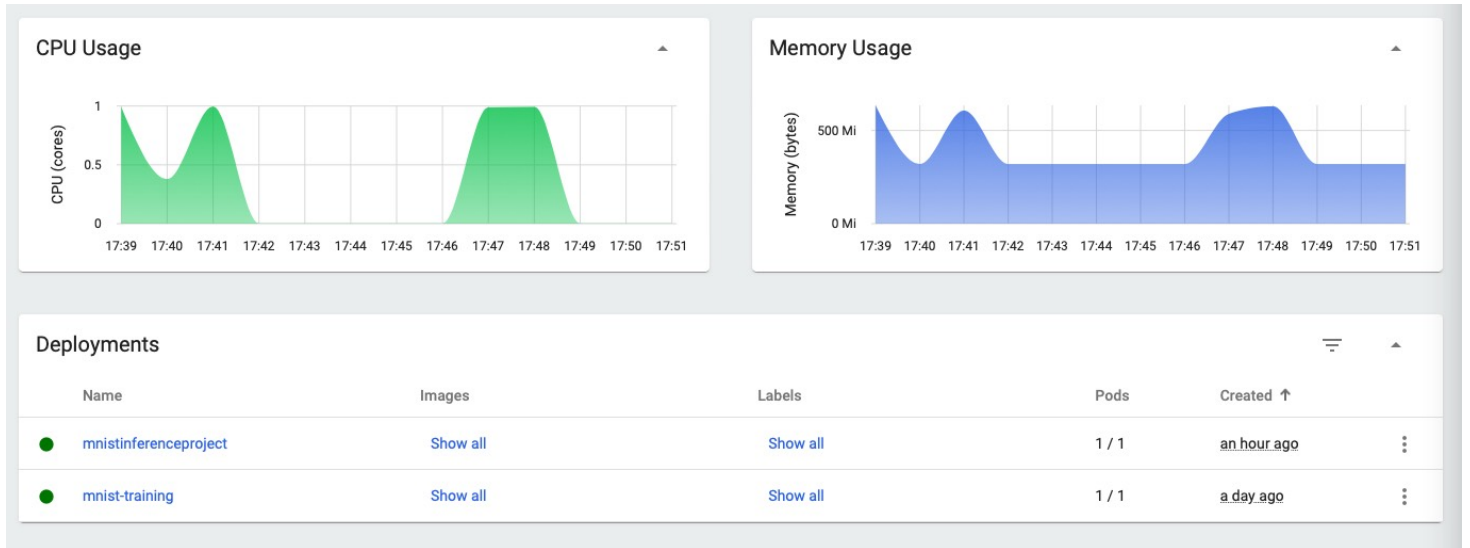
Test set: Average loss: 0.0553, Accuracy: 9823/10000 (98%)

Test set: Average loss: 0.0553, Accuracy: 9823/10000 (98%)

Test set: Average loss: 0.0553, Accuracy: 9823/10000 (98%)
```

10. Then, upon the successful test locally, I pushed the image to docker hub.

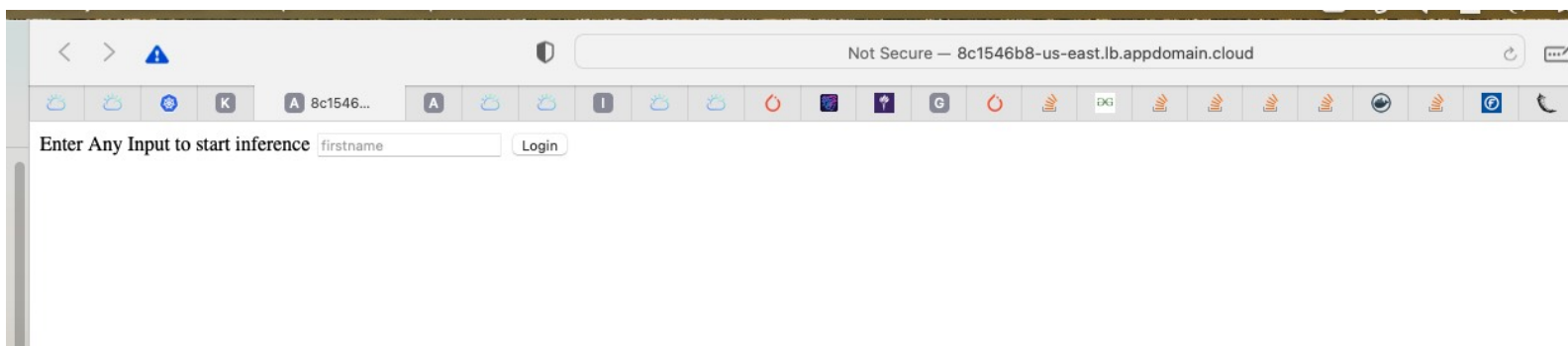
11. Then, I deploy the deployment and service on k8s dashboard.



12. Then I was able to repeat my local webpage inference process using the external URL given by k8s: <http://8c1546b8-us-east.lb.appdomain.cloud:8002>

Services

Name	Labels	Type	Cluster IP	Internal Endpoints	External Endpoints	Created ↑
<div><div></div><div>mnistinferenceproject</div></div>	<div>Show all</div>	LoadBalancer	172.21.144.86	mnistinferenceproject:8002 TCP mnistinferenceproject:31137 TCP	<div>8c1546b8-us-east.lb.appdomain.c</div> <div>an hour ago</div> <div></div>	



13. Then, upon inputting sth on the website, I can see that some unique log that is not seen anywhere else is generated on the k8s replicaset log. I am not sure why this output is different from the output when I run it on my local VM docker. But I didn't print any output explicitly so this might be why. But you can clearly see that 0.3% 0.7%.... reflects loading the model. Also note that only the first POST request will generate inference response. U need to restart afterward to do this again.

Logs from mnistinference... ▾ in mnistinference... ▾



```
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8002
* Running on http://172.17.61.134:8002
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 139-250-483
10.241.0.5 - - [28/Nov/2022 22:35:58] "GET / HTTP/1.1" 200 -
0.3% 0.7% 1.0% 1.3% 1.7% 2.0% 2.3% 2.6% 3.0% 3.3% 3.6% 4.0% 4.3% 4.6% 5.0% 5.3% 5.6% 6.0% 6.3% 6.6% 6.9% 7.3% 7.6% 7.9% 8.3% 8.6% 8.9% 9.3% 9.6% 9.9% 10.2%
100.0%
2.0% 4.0% 6.0% 7.9% 9.9% 11.9% 13.9% 15.9% 17.9% 19.9% 21.9% 23.8% 25.8% 27.8% 29.8% 31.8% 33.8% 35.8% 37.8% 39.7% 41.7% 43.7% 45.7% 47.7% 49.7% 51.7% 53.7%
100.0%
10.241.0.4 - - [28/Nov/2022 22:37:29] "GET / HTTP/1.1" 200 -
10.241.0.4 - - [28/Nov/2022 22:39:24] "GET / HTTP/1.1" 200 -
10.241.0.6 - - [28/Nov/2022 22:39:42] "POST / HTTP/1.1" 200 -
10.241.0.5 - - [28/Nov/2022 22:41:53] "GET / HTTP/1.1" 200 -
10.241.0.6 - - [28/Nov/2022 22:42:16] "GET / HTTP/1.1" 200 -
10.241.0.5 - - [28/Nov/2022 22:56:36] "GET / HTTP/1.1" 200 -
10.241.0.6 - - [28/Nov/2022 22:56:38] "GET / HTTP/1.1" 200 -
10.241.0.4 - - [28/Nov/2022 22:56:45] "GET / HTTP/1.1" 200 -
10.241.0.5 - - [28/Nov/2022 22:58:07] "GET / HTTP/1.1" 200 -
```

Controller usage:

For training, I used only 1 replicaset, 1 replicaset controller and a deployment. A deployment is for k8s. This is not a high security application, so 1 replication save resources.

For inference, I used only 1 replicaset, 1 replicaset controller, a deployment and a service. A deployment is for k8s, while a service is for the web interface. This is not a high security application, so 1 replication save resources.

Overall: This project takes me 20 hours. I hope the instruction could be more complete so it wouldn't be that time consuming. But overall, I learned a lot about k8s.

