CONTAINMENT ZONE ALERTING APPLICATION
ASSIGNMENT 4:

Questions:

1. **Pull an Image from docker hub and run it in docker playground.**

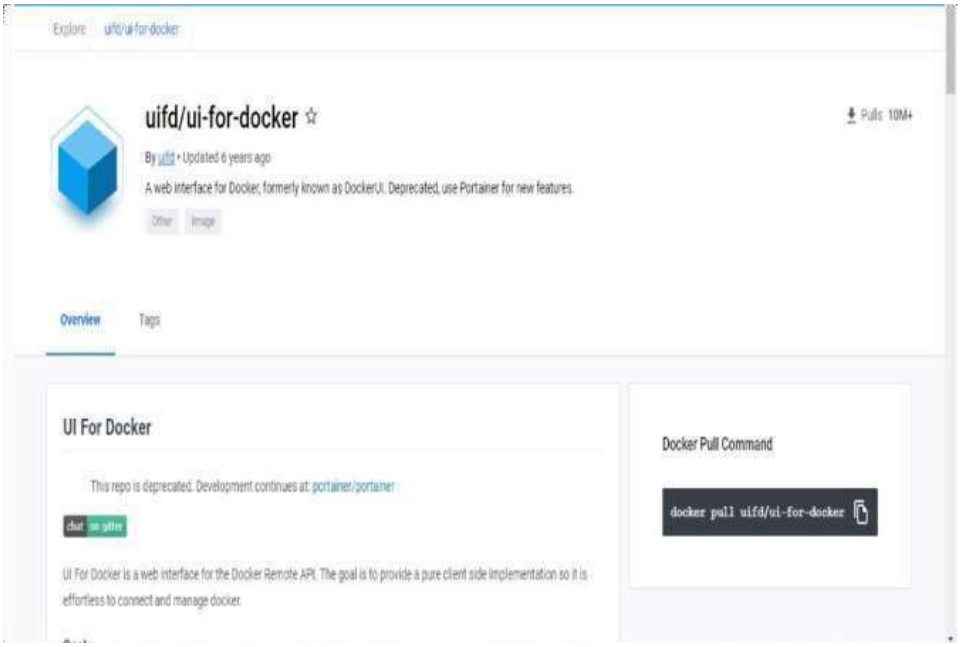
2. **Create a dockerfile for the job portal / flask application and deploy it inDocker desktop application.**

3. **Create an IBM container registry and push a docker image of a flaskapplication or job portal app.**

4. **Create a Kubernetes cluster in IBM cloud and deploy flask applicationimage or job portal image and also expose the same app to run in nodeport.**

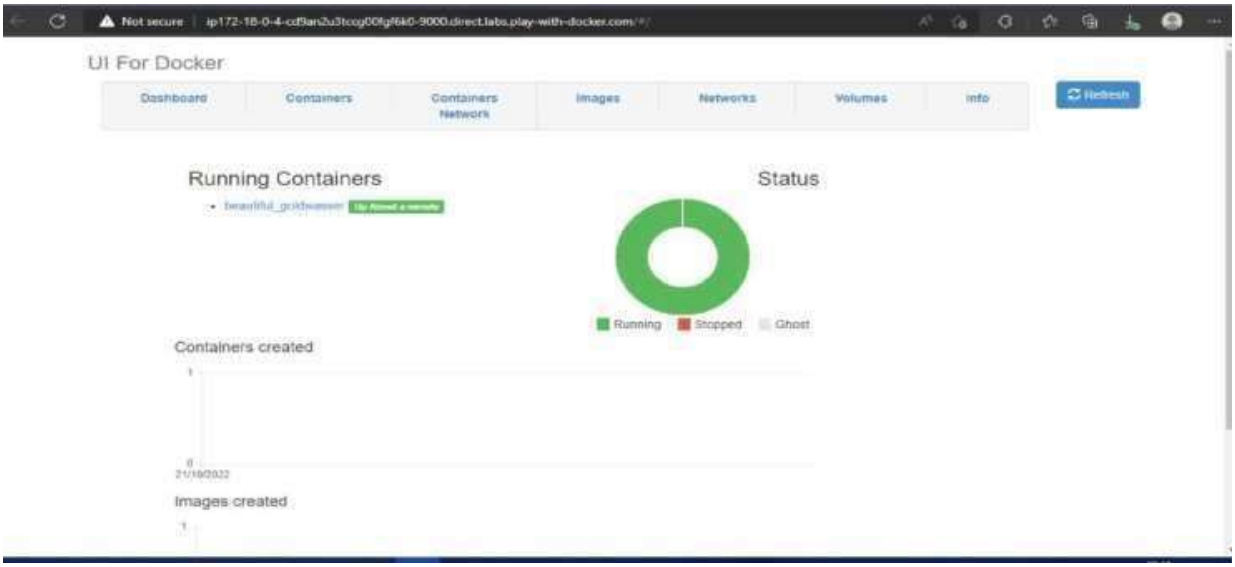
Answers:

1. **Pull an Image from docker hub and run it in docker playground.**

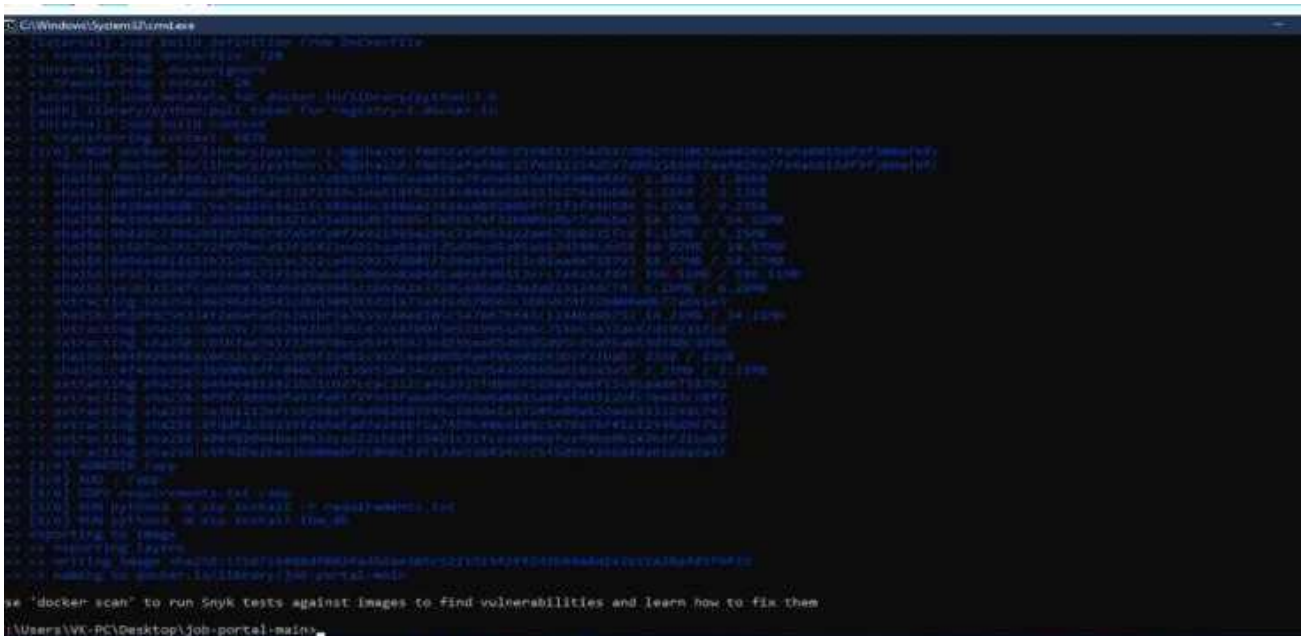


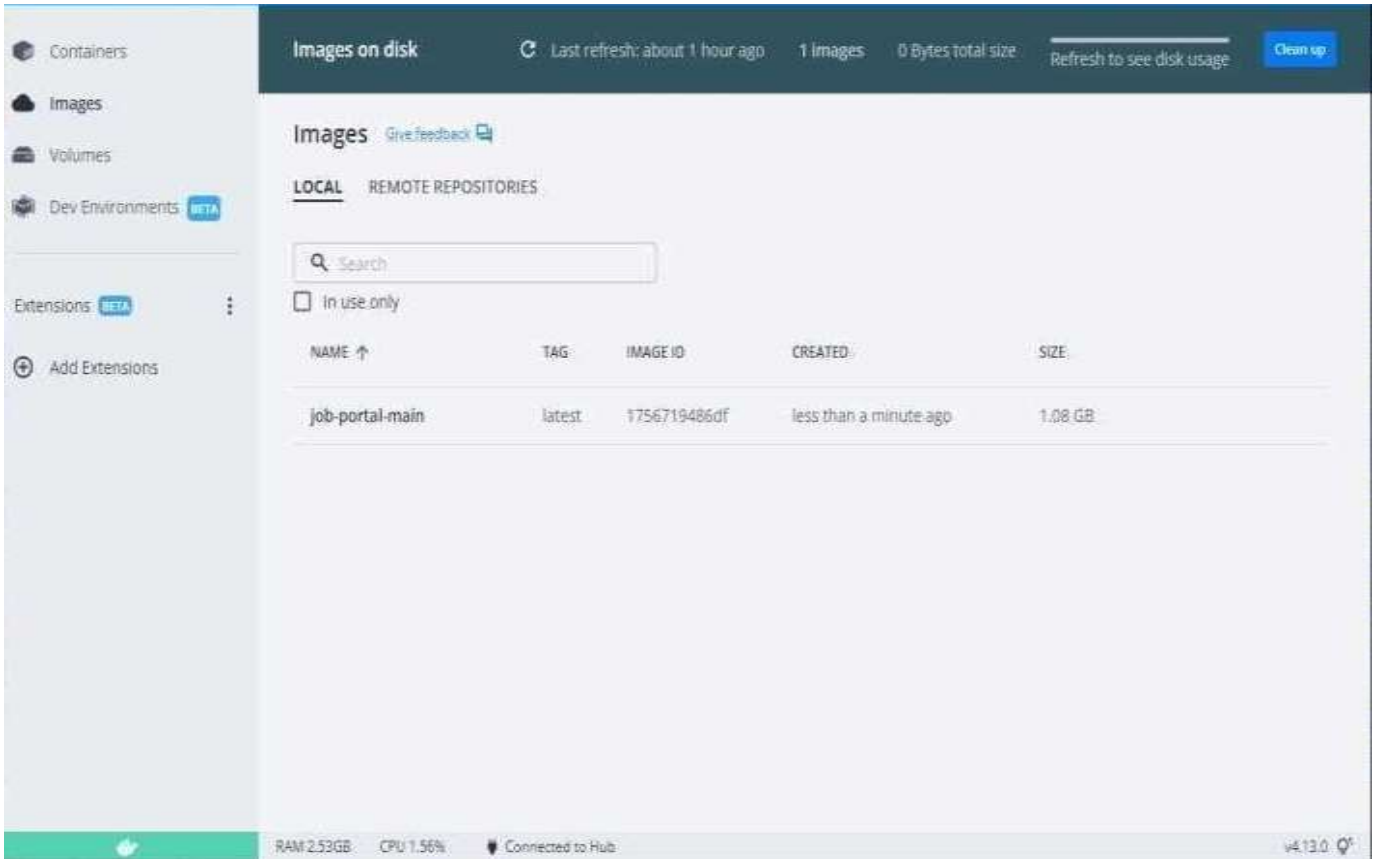
cd9an2u3_cd9av060qau0008hbjs0
192.168.0.13
OPEN PORT
Memory
CPU
ssh ip172-18-0-4-cd9an2u3tccg00fgf6k0@direct.labs.play-w
DELETE
EDITOR
This is a sandbox environment. Using personal credentials
is HIGHLY! discouraged. Any consequences of doing so are
completely the user's responsibilites.
The PWD team.
[node1] (local) root@192.168.0.13 ~
$ docker pull uifd/ui-for-docker
Using default tag: latest
latest: Pulling from uifd/ui-for-docker
841154d080c8: Pull complete
Digest: sha256:fe371ff5a69549269b24073a9ab1244dd4c0b834cbedf244870572150b1cb749
Status: Downloaded newer image for uifd/ui-for-docker:latest
docker.io/uifd/ui-for-docker:latest
[node1] (local) root@192.168.0.13 ~
$ docker run -d -p 9000:9000 --privileged -v /var/run/docker.sock:/var/run/docker.sock uifd/ui-for-docker
c590dd163101ee795bdcea0cb1ddd98f6fc549cb5f24dacb9ff7c1931923fc0d
[node1] (local) root@192.168.0.13 ~
$

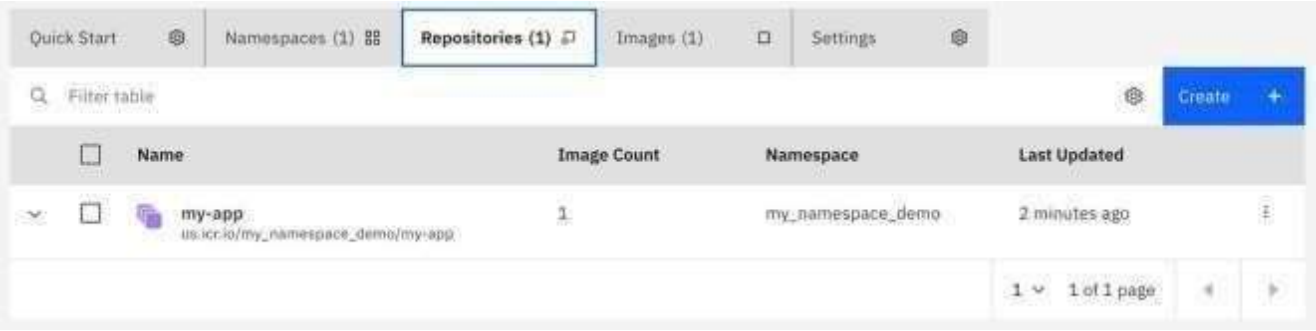
Not secure
ip172-18-0-4-cd9an2u3tccg00fgf6k0-9000.direct.labs.play-with-docker.com
UI For Docker
Dashboard
Containers
Containers Network
Images
Networks
Volumes
Info
Refresh
UI For Docker
The UI for Docker container engine
Learn more.
Running Containers
Status


Not secure
UI For Docker
Dashboard
Containers
Containers Network
Images
Networks
Volumes
Info
Refresh
Running Containers
Status
Running
Stopped
Ghost
Containers created
Images created

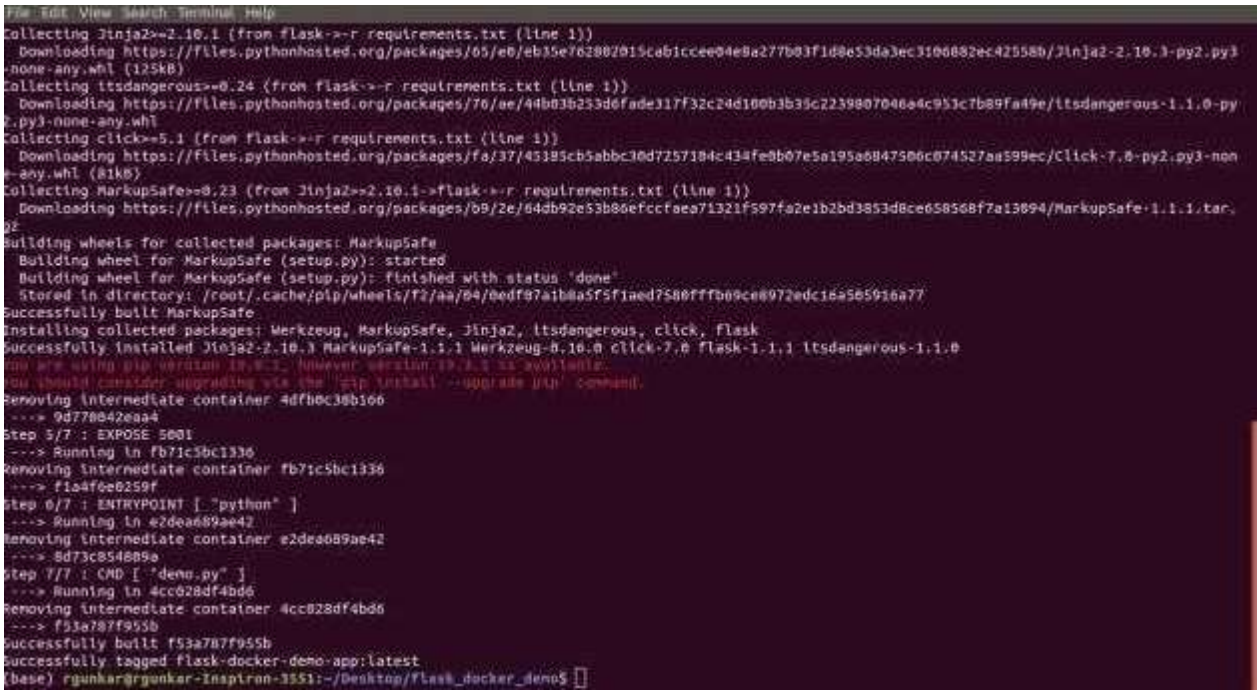
2. Create a dockerfile for the job portal / flask application and deploy it inDocker desktop application.







3. Create an IBM container registry and push a docker image of a flaskapplication or job portal app.



2. Change directory to Lab 1:

```
cd "Lab 1"
```

3. Log in to the IBM Cloud CLI:

```
ibmcloud login
```

To specify an IBM Cloud region, include the API endpoint.

Note: If you have a federated ID, use `ibmcloud login --sso` to log in to the IBM Cloud CLI. You know you have a federated ID when the login fails without the `-- sso` and succeeds with the `-- sso` option.

4. In order to upload images to the IBM Cloud Container Registry, you first need to create a namespace with the following command:

```
ibmcloud cr namespace-add <my_namespace>
```

5. Build the container image with a `1` tag and push the image to the IBM Cloud Registry: `ibmcloud cr build --tag us.icr.io/<my_namespace>/hello-world:1 .`

6. Verify the image is built:

```
ibmcloud cr images
```

7. If you created your cluster at the beginning of this, make sure it's ready for use.

Create a

1 . Push an image to IBM Cloud Container Registry

To push an image, we first must have an image to push. We have prepared several

`Dockerfile`s in this repository that will create the images. We will be running the images, and creang new ones, in the later labs.

This lab uses the Container Registry built in to IBM Cloud, but the image can be created and uploaded to any standard Docker registry to which your cluster has acces

Kubernetes cluster in IBM cloud and deploy flask pplication image or job portal image and also expose the same app.

Run ibmcloud ks clusters and make sure that your cluster is in "Normal"state.

Use ibmcloud ks workers --cluster <yourclustername> , and make surethat all workers are in "Normal" state with "Ready" status.

Make a note of the public IP of the worker.

You are now ready to use Kubernetes to deploy the hello-world application.

2. Deploy your application
   1. Run ibmcloud ks cluster config --cluster <yourclustername> .

Start by running your image as a deployment:
kubectl create deployment hello-world-deployment

When you're all done, you can either use this deployment in the next lab of this course , or you can remove the deployment and thus stop taking the course.

1. To remove the deployment and service, use `kubectl delete all -l app=hello-world-deployment .`

-- image=us.icr.io/<my_namespace>/hello-world:1

This action will take a bit of time. To check the status of your deployment, you can use kubectl get pods .

You should see output similar to the following:

=> kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
helloworld-562211614-0g2kd	0/1	ContainerCreating	0	1m

2. Once the status reads `Running` , expose that deployment as a service, accessed through the IP of the worker nodes. The example for this course listens on port 8080 . Run:

```
kubectl expose deployment/hello-world-deployment --type=NodePort --port=8080
-- name=hello-world-service --target-port=8080
```

3. To find the port used on that worker node, examine your new service: `kubectl describe service hello-world-service` Take note of the "NodePort:" line as `< nodeport>`

4. Run `ibmcloud ks worker ls --cluster <name-of-cluster>` , and note the public IP as `< public-IP>` .

5. You can now access your container/service using `curl <public-IP>:<nodeport>` (or your favorite web browser). If you see, "Hello world! Your app is up and running in a cluster!" you're done!