

SMART FASHION RECOMMENDATION APPLICATION

Assignment Number	4
Assignment Date	28 th October 2022
Student Name	Kishore.U
Student Roll Number	510919106010
Maximum marks	2 MARKS

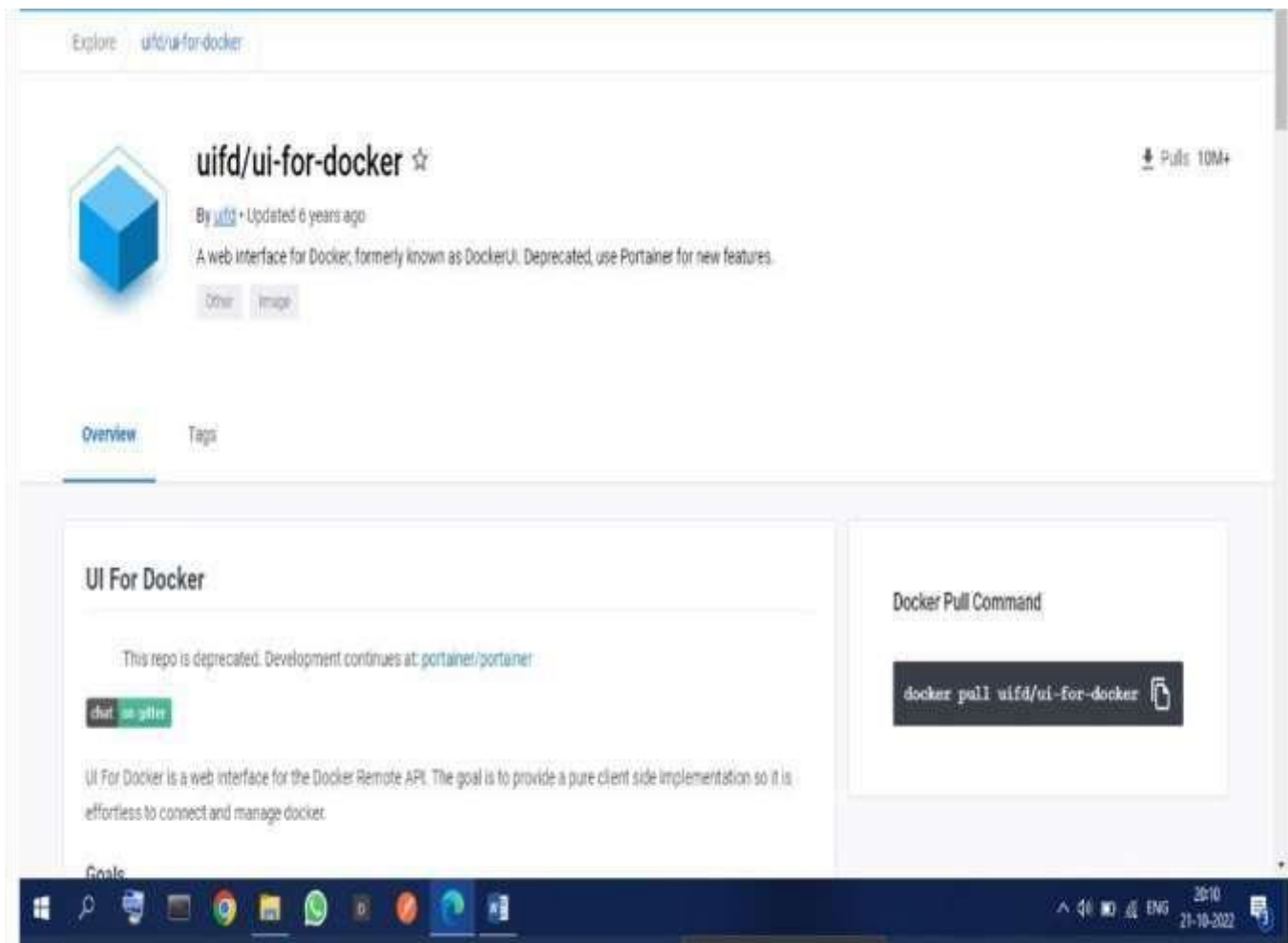
Question:

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 in Docker desktop application.
3. Create an IBM container registry and push a docker image of a flask application or job portal app.

4. Create a Kubernetes cluster in IBM cloud and deploy flask application image 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_cd9av060gau0008hbjs0

IP: 192.168.0.13 OPEN PORT

Memory CPU

SSH

ssh ip172-18-0-4-cd9an2u3tcog00fgf6k0@direct.labs.play-with-docker.com

DELETE EDITOR

```
This is a sandbox environment. Using personal credentials is HIGHLY discouraged. Any consequences of doing so are completely the user's responsibilities.

The FWD team.
=====
[local] (local) root@192.168.0.13 ~
# docker pull uifd/ui-for-docker
Using default tag: latest
latest: Pulling from uifd/ui-for-docker
041154d080c8: Pull complete
Digest: sha256:fe371ff5a69549269b24073a5ab1244dd4c0b834ebad244870572150b1eb749
Status: Downloaded newer image for uifd/ui-for-docker:latest
docker.io/uifd/ui-for-docker:latest
[local] (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
c590dd163101ee795bdcea0cb1dd498f6fc549cb5f24dadb9ff7c1931923fc0d
[local] (local) root@192.168.0.13 ~
```

UI For Docker

Dashboard Containers Containers Network Images Networks Volumes Info Refresh

UI For Docker

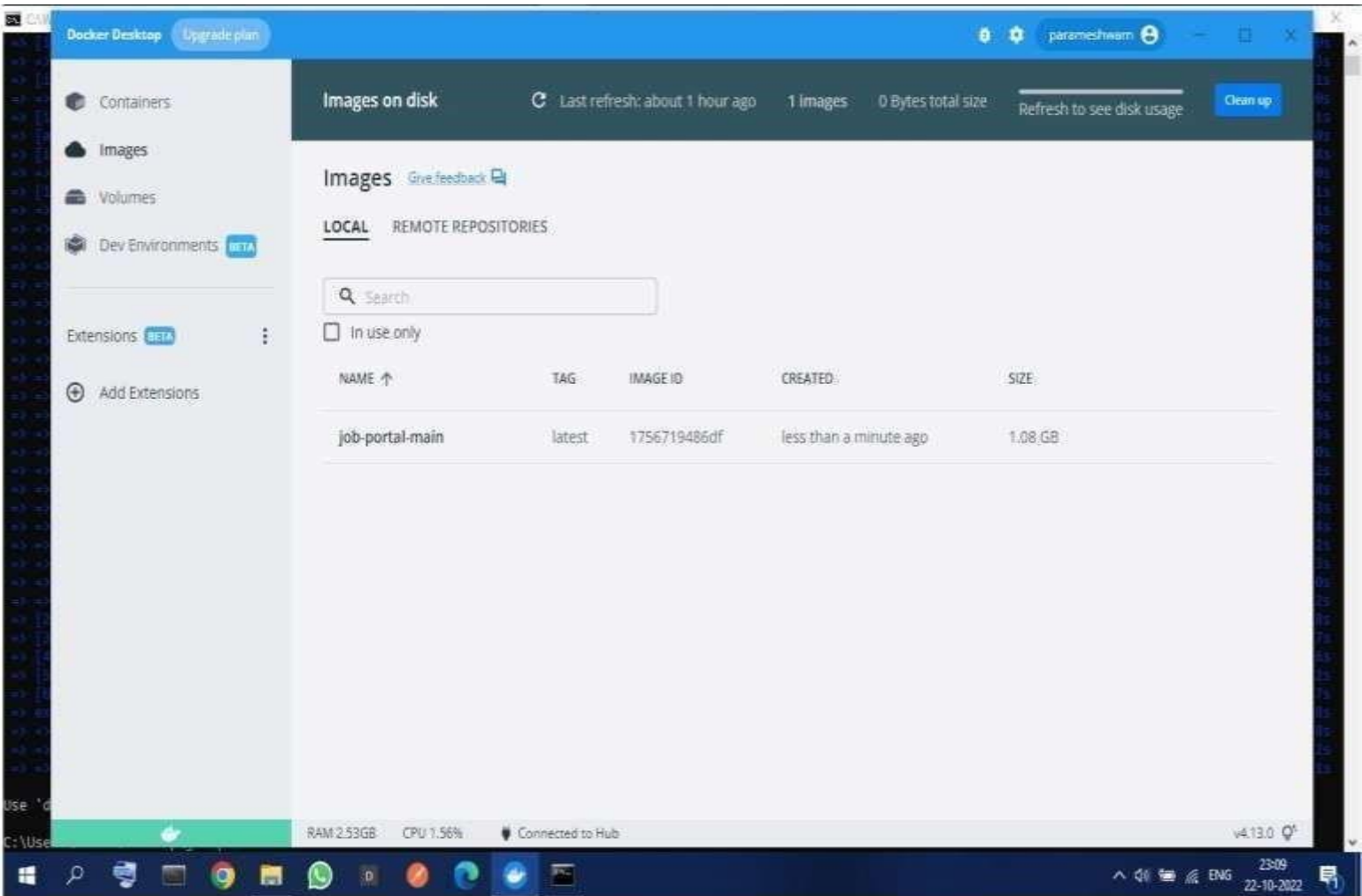
The UI for Docker container engine

Learn more

Running Containers

- beautiful_goldwasser Up About 2 minutes

Status



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



```
Activities Terminal * Sun 21:25 * rgunkar@rgunkar-Inspiron-3551: ~/Desktop/flask_docker_demo

File Edit View Search Terminal Help
Collecting Jinja2==2.10.1 (from flask->r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/85/e6/eb35e762867915cab1ccee04e8a277b03f1d8e53da3ec3106882ec42558b/Jinja2-2.10.3-py2.py3-none-any.whl (125kB)
Collecting itsdangerous==0.24 (from flask->r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/70/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2.py3-none-any.whl
Collecting click==5.1 (from flask->r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/fa/37/45185cb5abb310d7257104c434fe9b07e5a195a6847506c074527aa599ec/click-7.0-py2.py3-none-any.whl (81kB)
Collecting MarkupSafe==0.23 (from Jinja2==2.10.1->flask->r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/b9/2e/64db92e53b86efccfaea71321f597fa2e1b2bd3853d8ce658568f7a13694/MarkupSafe-1.1.1.tar.gz
Building wheels for collected packages: MarkupSafe
  Building wheel for MarkupSafe (setup.py): started
  Building wheel for MarkupSafe (setup.py): finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/f2/aa/84/0edf87a1b8a5f5f1aed7580fffb69ce0972edc16a505916a77
Successfully built MarkupSafe
Installing collected packages: Werkzeug, MarkupSafe, Jinja2, itsdangerous, click, flask
Successfully installed Jinja2-2.10.3 MarkupSafe-1.1.1 Werkzeug-0.10.0 click-7.0 flask-1.1.1 itsdangerous-1.1.0
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Removing intermediate container 4dfb6c38b166
----> 9d778042eaa4
Step 5/7 : EXPOSE 5001
----> Running in fb71c5bc1336
Removing intermediate container fb71c5bc1336
----> f1a4f6e6259f
Step 6/7 : ENTRYPOINT [ "python" ]
----> Running in e2dea689ae42
Removing intermediate container e2dea689ae42
----> 8d73c854889a
Step 7/7 : CMD [ "demo.py" ]
----> Running in 4cc028df4bd6
Removing intermediate container 4cc028df4bd6
----> f33a787f955b
Successfully built f33a787f955b
Successfully tagged flask-docker-demo:latest
(base) rgunkar@rgunkar-Inspiron-3551:~/Desktop/flask_docker_demo$
```

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.

4. **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>/helloworld: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 Kubernetes cluster in IBM cloud and deploy flask application image or job portal image and also expose the same app to run in nodeport.

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 creating 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 access

- Run `ibmcloud ks clusters` and make sure that your cluster is in "Normal" state.
- Use `ibmcloud ks workers --cluster <yourclustername>`, and make sure that 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 --
image=us.icr.io/<my_namespace>/hello-world:1
```

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 .
```

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!