Assignment - 4
Docker Desktop and Playground

Name	SHANMUGARAJAN M
College Roll No	113119UG04097
Date	22 October 2022
Team ID	PNT2022TMID22461
Project Name	Skill / Job Recommender Application

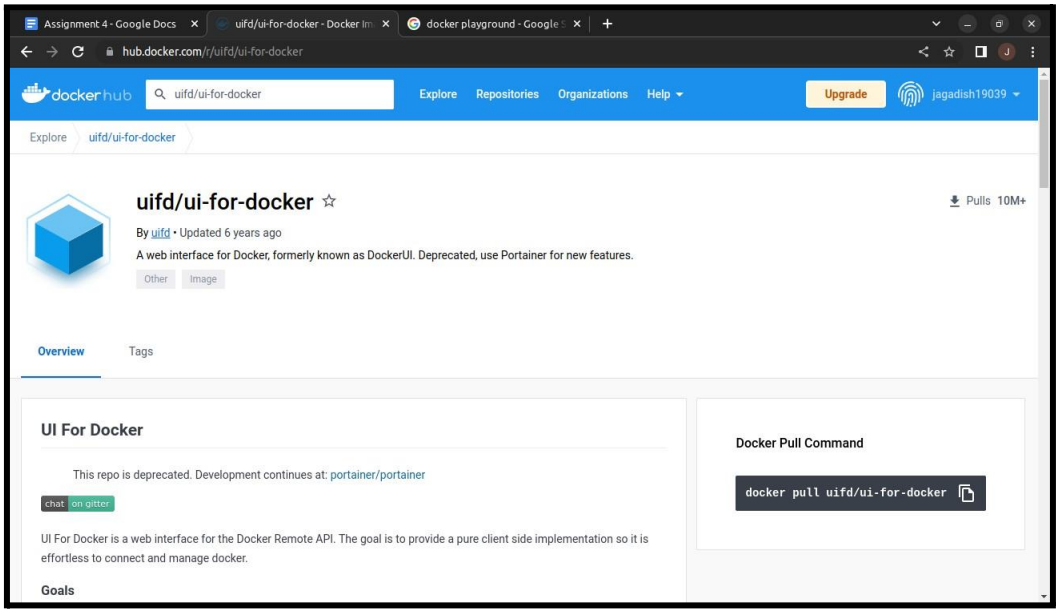
Question

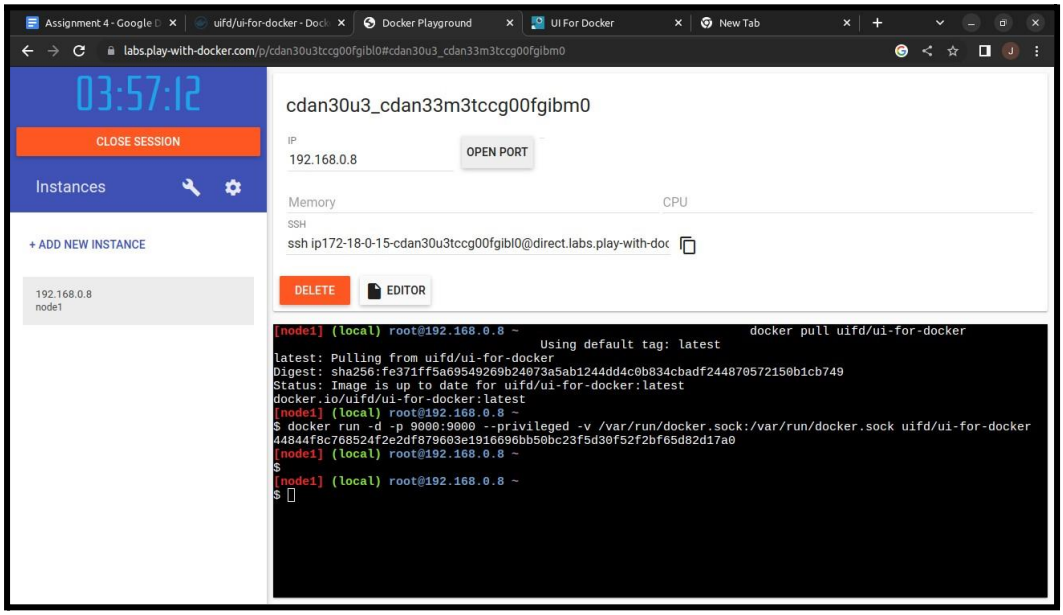
1. Pull an Image from docker hub and run it in Docker Playground
2. Create a docker file for the jobportal application and deploy it in Docker desktop application
3. Create a IBM container registry and deploy helloworld app or jobportal app
4. Create a Kubernetes cluster in IBM cloud and deploy helloworld image or jobportal image and also expose the same app to run in nodeport

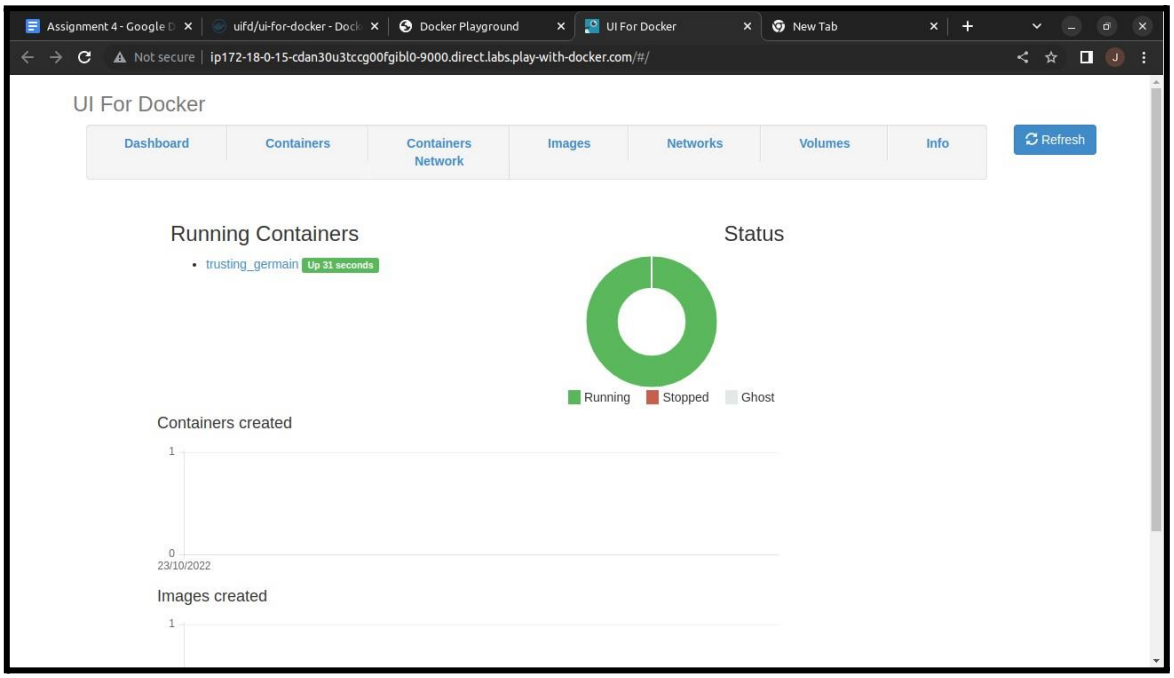
Solutions

1. Pull an Image from docker hub and run it in Docker Playground

a. Pull an image *uifd/ui-for-docker* from the docker hub
b. This image is used for viewing and managing the docker engine
c. Use **`docker pull image_name`** and **`docker run -it image_name`** commands to run the above image in the Docker Playground


Assignment 4 - Google Docs
uifd/ui-for-docker - Docker Im
docker playground - Google
hub.docker.com/r/uifd/ui-for-docker
dockerhub
uifd/ui-for-docker
Explore
Repositories
Organizations
Help
Upgrade
jagadish19039
Explore
uifd/ui-for-docker
uifd/ui-for-docker
By uifd • Updated 6 years ago
A web interface for Docker, formerly known as DockerUI. Deprecated, use Portainer for new features.
Other
Image
Pulls 10M+
Overview
Tags
UI For Docker
This repo is deprecated. Development continues at: portainer/portainer
chat on gitter
UI For Docker is a web interface for the Docker Remote API. The goal is to provide a pure client side implementation so it is effortless to connect and manage docker.
Goals
Docker Pull Command
docker pull uifd/ui-for-docker


Assignment 4 - Google D
uifd/ui-for-docker - Dock
Docker Playground
UI For Docker
New Tab
labs.play-with-docker.com/p/cdan30u3tccg00fgibl0#cdan30u3_cdan33m3tccg00fgibm0
03:57:12
CLOSE SESSION
Instances
+ ADD NEW INSTANCE
192.168.0.8
node1
cdan30u3_cdan33m3tccg00fgibm0
IP
192.168.0.8
OPEN PORT
Memory
CPU
SSH
ssh ip172-18-0-15-cdan30u3tccg00fgibl0@direct.labs.play-with-doc
DELETE
EDITOR
[node1] (local) root@192.168.0.8 ~
docker pull uifd/ui-for-docker
Using default tag: latest
latest: Pulling from uifd/ui-for-docker
Digest: sha256:fe371ff5a69549269b24073a5ab1244dd4c0b834cbadf244870572150b1cb749
Status: Image is up to date for uifd/ui-for-docker:latest
docker.io/uifd/ui-for-docker:latest
[node1] (local) root@192.168.0.8 ~
$ docker run -d -p 9000:9000 --privileged -v /var/run/docker.sock:/var/run/docker.sock uifd/ui-for-docker
44844f8c768524f2e2df879603e1916696bbb50bc23f5d30f52f2bf65d82d17a0
[node1] (local) root@192.168.0.8 ~
$
[node1] (local) root@192.168.0.8 ~
$



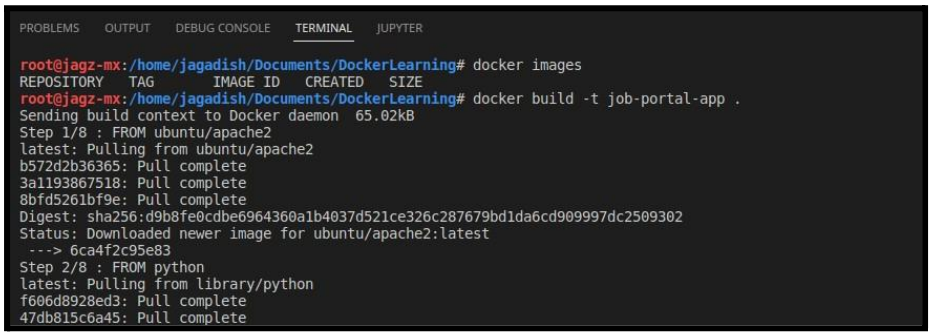
2. Create a docker file for the jobportal application and deploy it in Docker desktop application

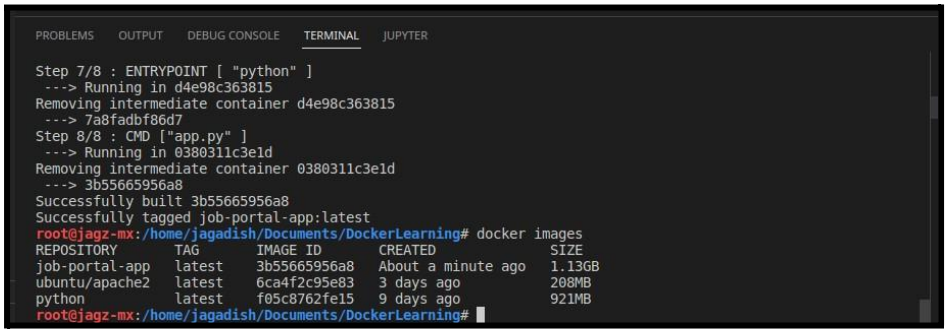
a. Create a docker file for build and deploy flask app.
b. Use **`docker build -t image_name .`** in the current directory to start building the docker image and deploy in our local docker
c. Use **`docker run -p 5000:5000 image_name`** to run in local system

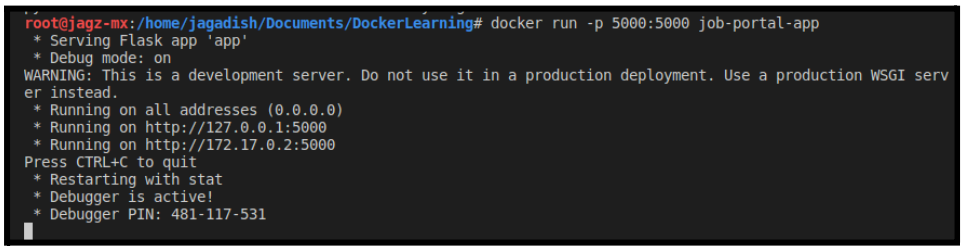
Dockerfile

```
FROM ubuntu/apache2
FROM python
COPY ./requirements.txt /flaskApp/requirements.txt
WORKDIR /flaskApp
RUN pip install -r requirements.txt
COPY . /flaskApp
ENTRYPOINT [ "python" ]
CMD ["app.py" ]
```

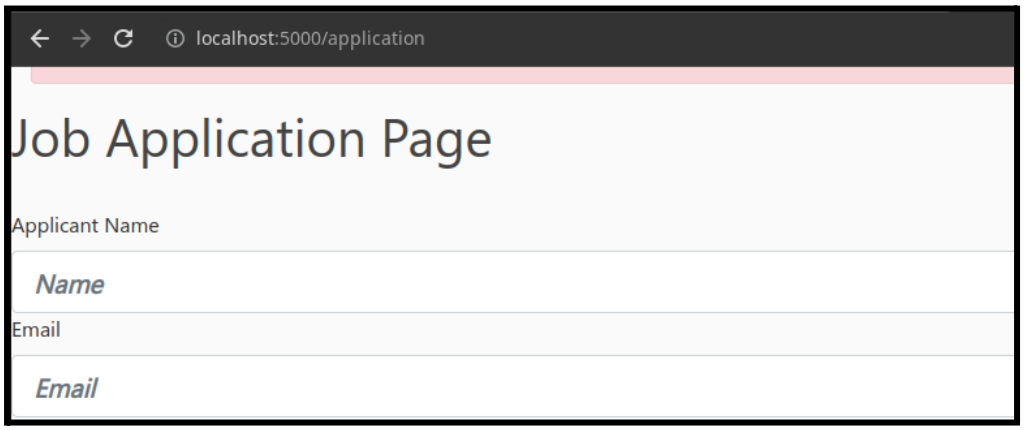
Steps Involved





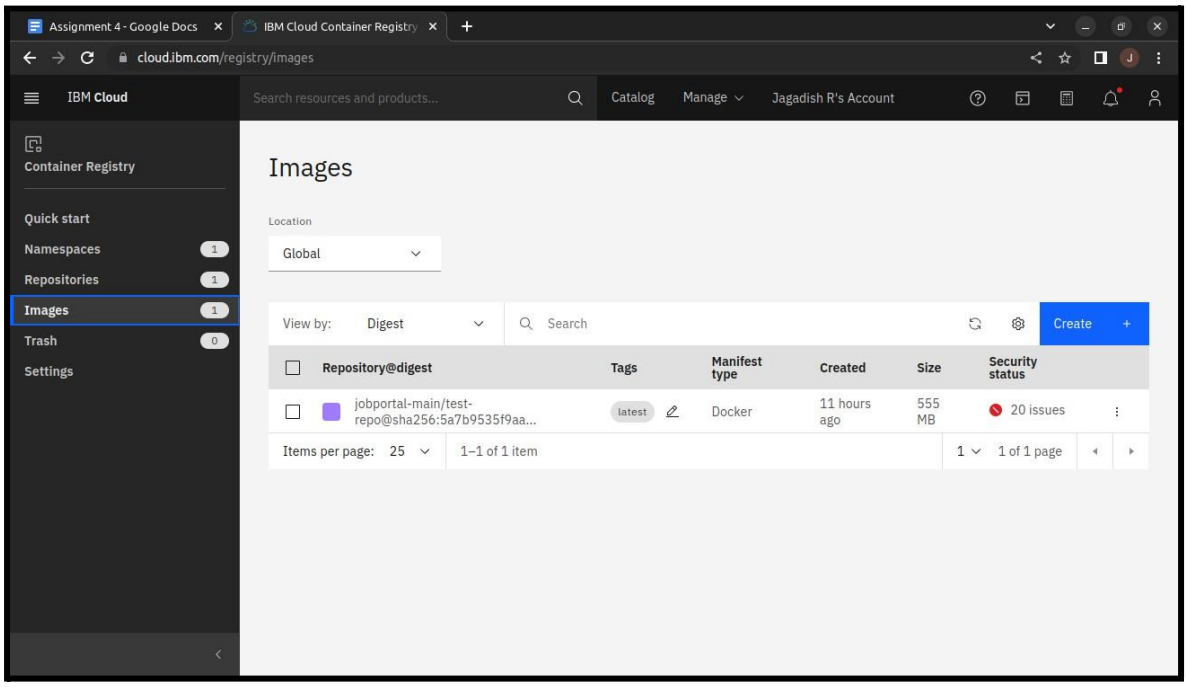


Run locally using docker



3. Create a IBM container registry and deploy helloworld app or jobportal app

a. Log into IBM cloud
b. Create a **container registry**
c. Using IBM Cloud CLI, install the **container registry plugin** in our system
d. Push our docker image into the created container registry using **docker push**
e. So, our job portal app is deployed in the IBM container registry



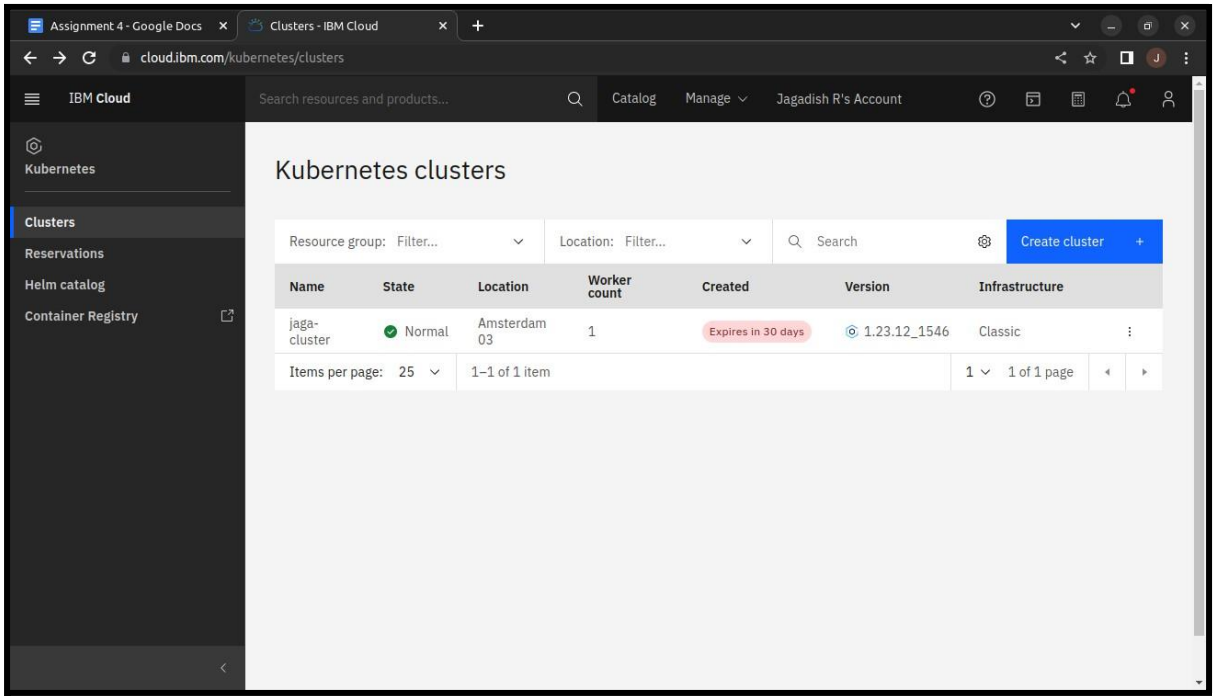
4. Create a Kubernetes cluster in IBM cloud and deploy helloworld image or jobportal image and also expose the same app to run in nodeport

a. Log into IBM cloud
b. Create a **kubernete**
c. Using IBM Cloud CLI, install the **ks plugin** in our system
d. Create **a cluster** in the kubernetes
e. Now, go to the **kubernetes dashboard** where we need to create a service based on a yml file (given below)
f. In that file, we have to mention *which image we are going to use* and the *app name*
g. Take the **public IP address** and **Nodeport** since we exposed the *flask app in nodeport*
h. Finally, we got the **url address** where our flask app is hosted

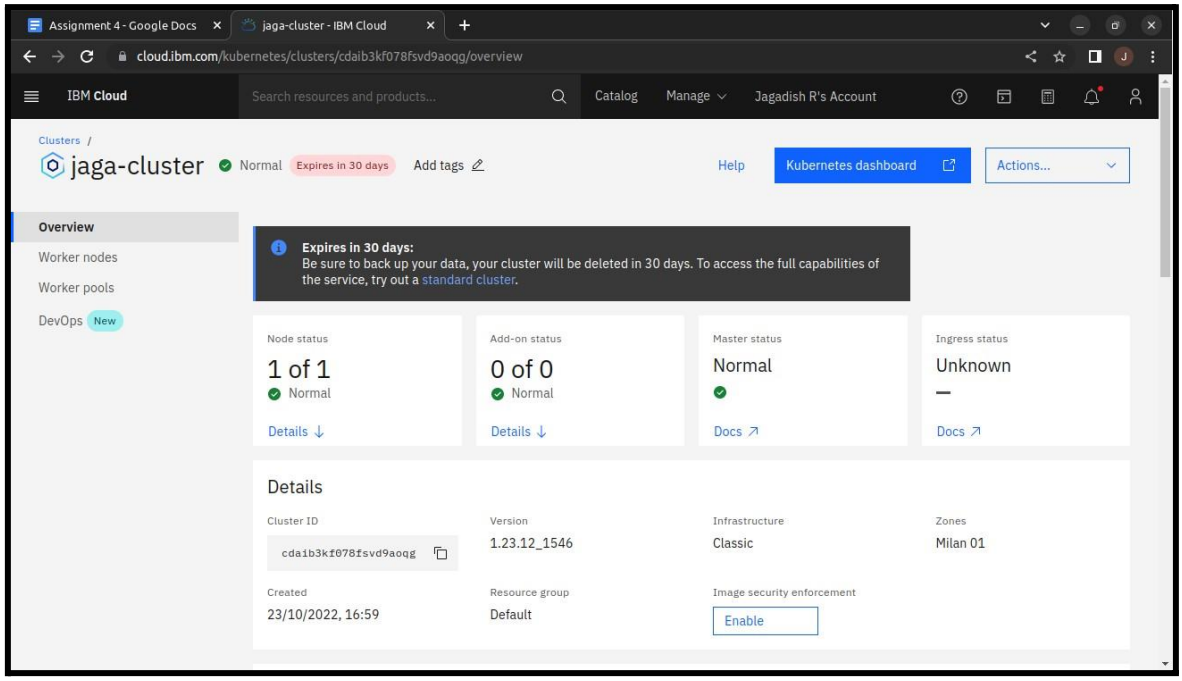
job-portal-app.yml

```
apiVersion: v1
kind: Service
metadata:
  name: job-portal-app
spec:
  selector:
    app: job-portal-app
  ports:
  - port: 5000
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: job-portal-app
  labels:
    app: job-portal-app
spec:
  selector:
    matchLabels:
      app: job-portal-app
  replicas: 1
  template:
    metadata:
      labels:
        app: job-portal-app
    spec:
      containers:
      - name: job-portal-app
        image: image_name
        ports:
        - containerPort: 5000
        env:
        - name: DISABLE_WEB_APP
          value: "false"
```

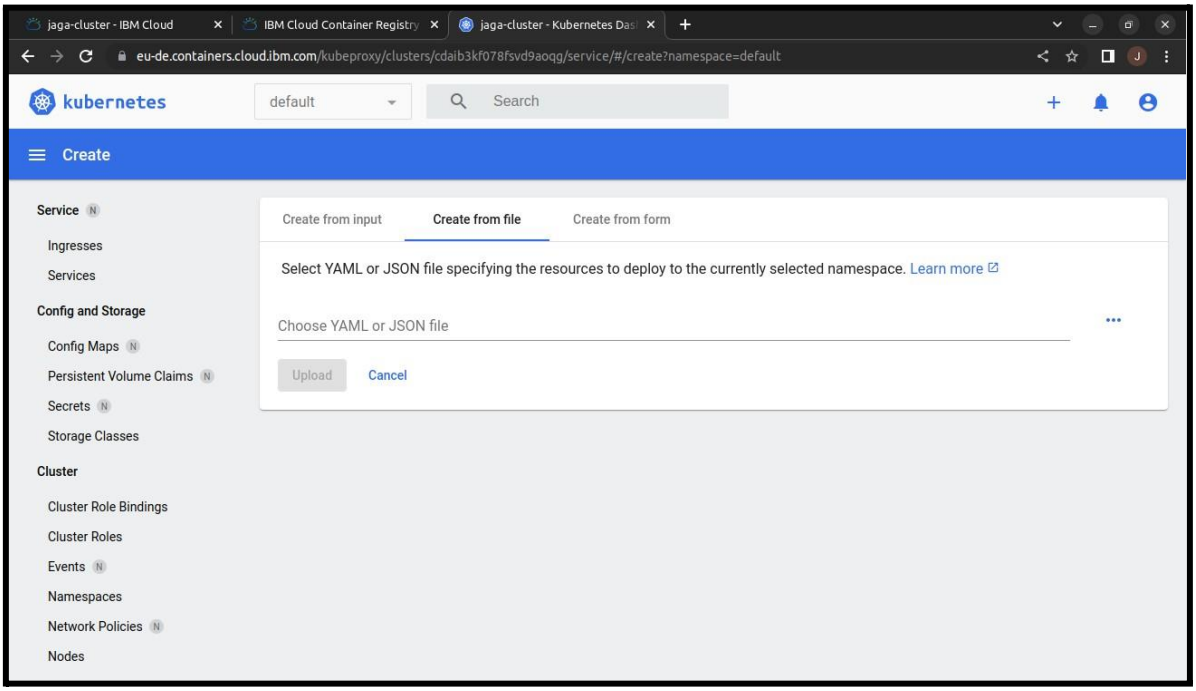
Cluster creation



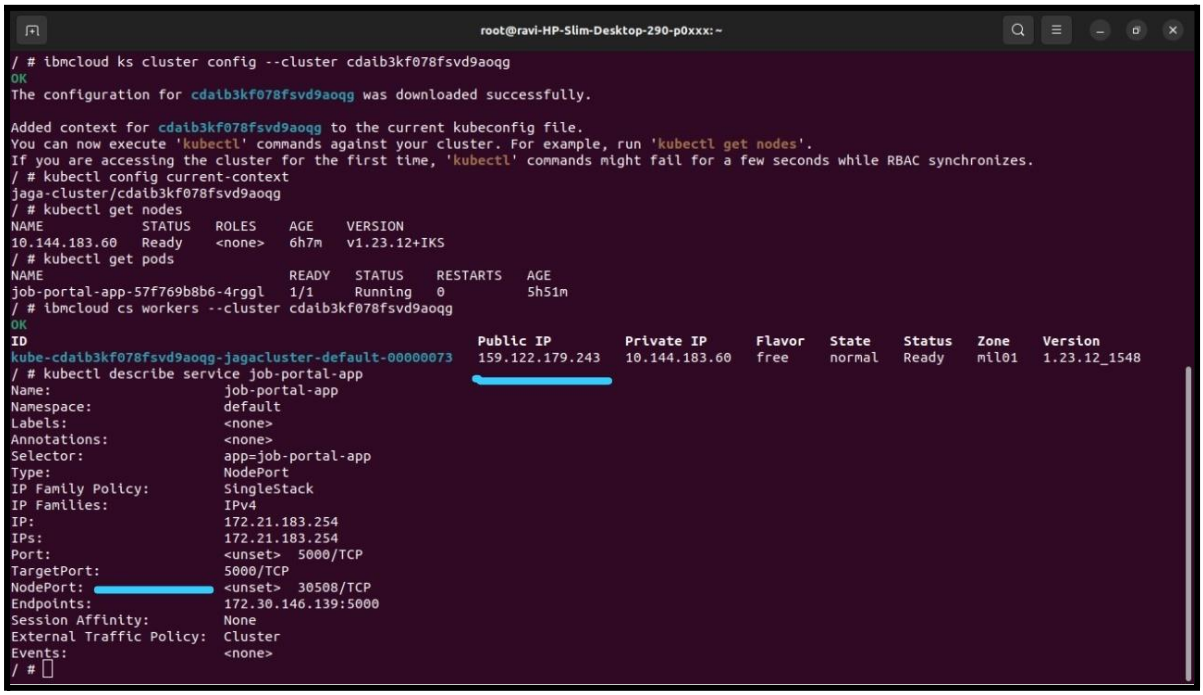
Configuring the cluster



Creating a service based on the yml file



Procedure to find the exposed url



Run our flask app in the IBM kubernetes