# Harman Take Home Test – Solutions

## Task 1

Create a deployment running nginx version 1.16.1 that will run in 2 pods that uses /var/www as its root dir. Nginx should run on port 8090  
a. Create a separate pod that mounts a volume. The volume should have a dir /var/www and contain a file "Hello World" html. This Pod should load before Nginx ( you need to prove this)  
b. Create service Load Balancer for Nginx  
c. Make  Nginx  accessible via the internet on port 80  
d. Scale this to 4 pods.  
e. Create rolling update to 1.19.4  
f. Check the status of the upgrade  
g. How do you do this in a way that you can see history of what happened?  
h. How would you do a Canary Upgrade?  
i. Undo the upgrade  
j. Scale down to 2 pods  
  
1) Do not use Helm  
2) Submit all yaml and config files  
3) Submit all kubectl commands  
4) Document your work in a way that anyone could repeat the results using your documentation

## Task 2

Write a very simple python program that would expose itself to port 80 with HELLO HARMAN message. Then, write yaml manifest for it, deploy it to K8S and expose it using nginx ingress.​

## Task 1 – Solutions

#vi demo-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.16.2

volumeMounts:

- name: workdir

mountPath: /var/www

ports:

- containerPort: 8090

#kubectl apply demo-deployment.yaml

1. #vi demo-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: init-demo

spec:

initContainers:

- name: install

image: busybox

command:

- wget

- "-O"

- "/work-dir/index.html"

- http://info.cern.ch

volumeMounts:

- name: workdir

mountPath: "/work-dir"

As this pod contains an init container, it will load prior to the previous container, provided we execute this yaml file before the previous one, or combine both of these yaml files together.

#kubectl apply demo-pod.yaml

1. #vi demo-service.yaml

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 9376

clusterIP: 10.0.171.239

type: LoadBalancer

status:

loadBalancer:

ingress:

- ip: 0.0.0.0/0

# kubectl apply demo-service.yaml

1. This will create a LoadBalancer type of service which will be accessible from the internet as the ingress is 0.0.0.0/0
2. #kubectl scale deployment.v1.apps/nginx-deployment –replicas=4
3. #kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.19.1
4. #kubectl rollout status deployment/nginx-deployment
5. History is not enabled by default. We need to use the --record=true option on kubectl create or kubectl apply to enable it.
6. If we want to roll out releases to a subset of users or servers using the Deployment, we can create multiple Deployments, one for each release, following the canary pattern. We can use multiple labels to distinguish deployments of different releases or configurations of the same component. It is common practice to deploy a canary of a new application release (specified via image tag in the pod template) side by side with the previous release so that the new release can receive live production traffic before fully rolling it out.

For instance, we can use a track label to differentiate different releases.

The primary, stable release would have a track label with value as stable

name: frontend

replicas: 3

...

labels:

app: guestbook

tier: frontend

track: stable

...

image: gb-frontend:v3

and then you can create a new release of the guestbook frontend that carries the track label with different value (i.e. canary), so that two sets of pods would not overlap

name: frontend-canary

replicas: 1

...

labels:

app: guestbook

tier: frontend

track: canary

...

image: gb-frontend:v4

1. #kubectl rollout undo deployment.v1.apps/nginx-deployment
2. #kubectl scale deployment.v1.apps/nginx-deployment –replicas=2

## Task 2 – Solutions

The Source Code: helloharman.py

# This program prints Hello, world!

print('Hello Harman')

The Dockerfile to build the image:

FROM python:3

ADD helloworld.py /

RUN pip install flask

RUN pip install flask\_restful

EXPOSE 80

CMD ["python", "./helloworld.py"]

To build the image:

#docker build –tag my-python-app .

To push the image to Docker Hub

#docker push tag my-python-app:latest

#docker image push

To create the deployment:

#kubectl create deployment

To deploy the image:

#vi python-deploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: python-deployment

labels:

app: my-python-app

spec:

replicas: 3

selector:

matchLabels:

app: my-python-app

template:

metadata:

labels:

app: my-python-app

spec:

containers:

- name: my-python-app

image: my-python-app:latest

ports:

- containerPort: 80

#kubectl apply –f python-deploy.yaml

To expose the deployment through an Ngnix Ingress:

#vi python-ingress.yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: python-ingress

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /$1

spec:

rules:

- host: python-deploy.info

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: web

port:

number: 8080

#kubectl apply –f python-ingress.yaml