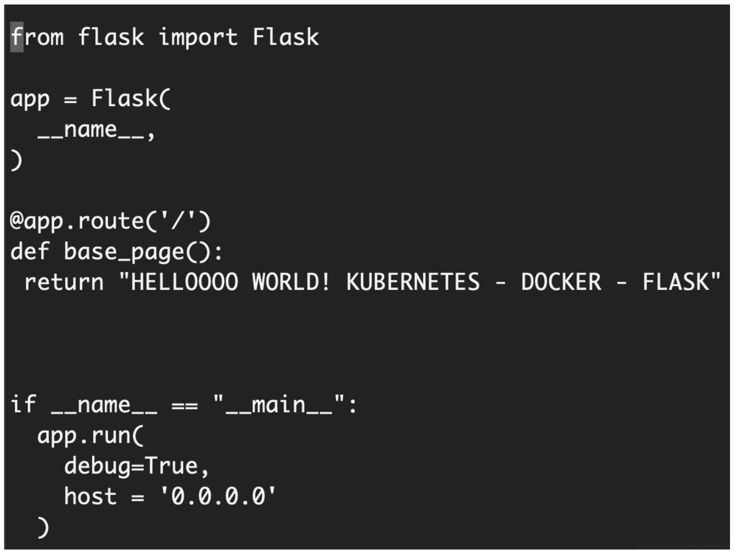
Kubernetes and Docker Project (3 Tasks):

Containerizing and Deploying a Flask App and multiservice Go App using tools like DockerHub, K3d and Yml files.

Task 1: Dockerize your flask app and create an image you can push up to DockerHub.

Use this link to help you create your flask app in a container:
https://runnable.com/docker/python/dockerize-your-flask-application

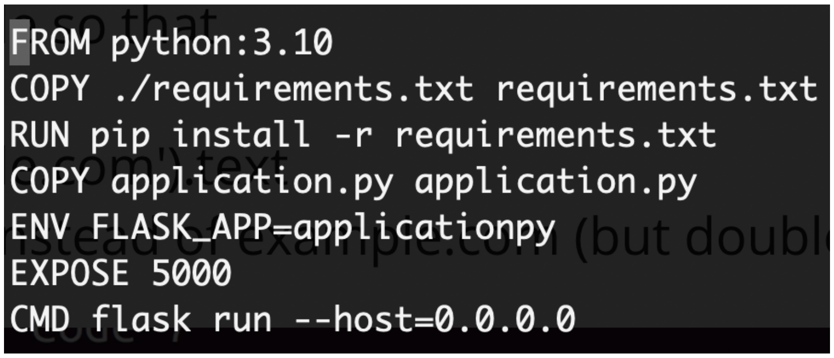
1. First Step is to Ensure your Flask App can run locally



```python
from flask import Flask

app = Flask(
  __name__,
)

@app.route('/')
def base_page():
 return "HELLOOOO WORLD! KUBERNETES - DOCKER - FLASK"



if __name__ == "__main__":
  app.run(
    debug=True,
    host = '0.0.0.0'
  )
```

2. You should have your app (application.py) in a folder along with a requirements.txt file and Dockerfile:
   a. requirements.txt
      i. Flask == 2.0.2
      ii. Only line in the sample app requirements
      iii. Can create requirements file using: **pip freeze > requirements. txt**
   b. Dockerfile
      i. This file describes the image that will be created when it is used. It will complete the following: Install python, bring

over the Req file to the image, install the req file on it, bring over the app file to the image, run the app using Flask and expose port 5000.



```
FROM python:3.10
COPY ./requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY application.py application.py
ENV FLASK_APP=applicationpy
EXPOSE 5000
CMD flask run --host=0.0.0.0
```

3. Build your Image from the Dockerfile and give it a name with -t.
    a. docker build -t kube-task-image .
    b. docker build -t <imageNametaghere> <path to Dockerfile>
4. We now need to run a container of this image. (just to check app works at this level)
    a. the app is running on 5000 but we want to use port 8080 with the container
    b. docker run -ti -p 8080:5000 <image name>
    c. To turn off container after app runs, docker kill <containerId>
    d. Could also turn back on with: docker start <containerId>
5. Finally lets Push the Image to DockerHub
    a. First tag your image with a Dockerhub name using your DH Username. docker tag <localimagename> fordonez20/<newnameforimageondockerhub>
    b. docker push fordonez20/<newtagname>

Task 2: Now Deploy your Flask app with Kubernetes.

This app will be running on a Kubernetes Pod with port 5000 exposed. To access the app there, first an external user will first be sent to port 8081 which is mapped to port 8080 by a Loadbalancer. We will create a Yaml file that will describe a Deployment (K8 Cluster) and a service. This service will connect to our loadbalancer on port 8080 and the Target Port for the Service will be 5000, where the Flask app is exposed.

Traffic Flow: Localhost:8081 → Loadbalancer 8081:8080 → K3d Cluster 8080:5000 → Pod (Flask App on 5000)

1. Create your cluster with a Load Balancer.
   `k3d cluster create cluster-kube-task-2 -p "8081:8080@loadbalancer"`

2. Create a deployment yaml file for your flask app.
   a. Yaml file contents: names must be lowercase
   b. Targetport and containerport need to say 5000
   c. Containerport opens 5000, target targets
   d. LoadBalancer (capital B)
   e. YML Contents:

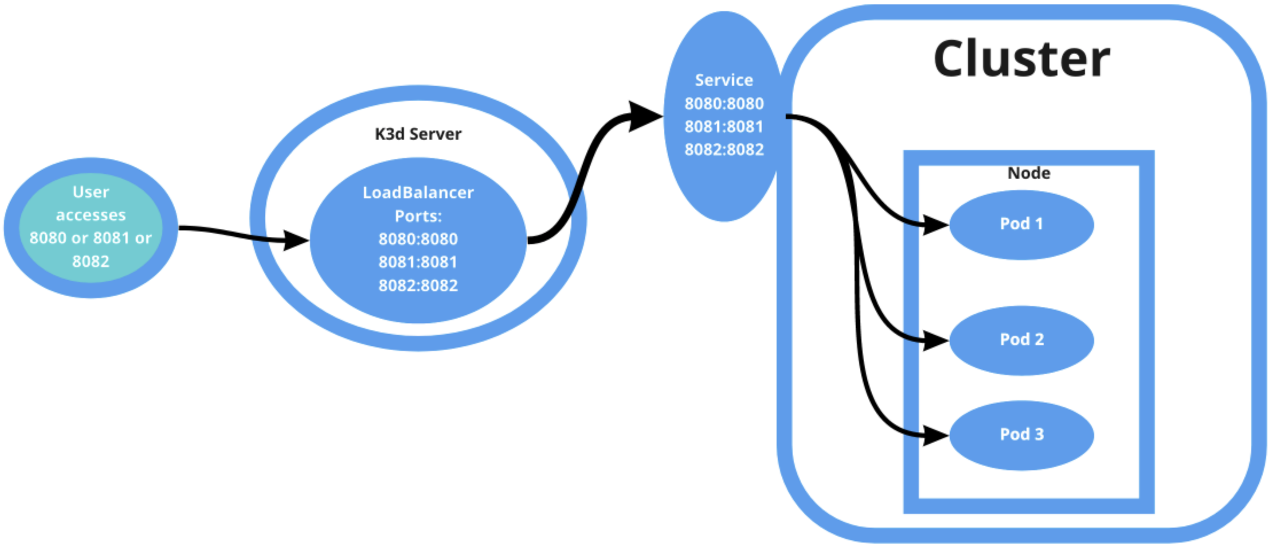
```
# Kube-Task-2 application- just the deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-task-2-deployment
spec:
  selector:
    matchLabels:
      app: kube-task-2
  replicas: 1 # tells deployment to run 1 pods matching the template
  template: # create pods using pod definition in this template
    metadata:
      labels:
        app: kube-task-2
    spec:
      containers:
      - name: kube-task-2-container
        image: fordonez20/kube-task-image-v2-dh:latest
        ports:
        - containerPort: 5000

---

apiVersion: v1
kind: Service
metadata:
  name: kube-task-2
spec:
  type: LoadBalancer
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 5000
  selector:
    app: kube-task-2
```

3. Run the yml file which will use your DockerHub Image:
   This will Create a Deployment and Service(type loadbalancer)
   a. `kubectl create -f <file-name.yml>`
4. Now you can check that your app is running at localhost:8081

Task 3: Deploying An Application with Kubernetes and 3 Pods

Our goal now is to use Kubernetes to deploy a given "Wishlist application" (written in Go), which mimics an app using 3 microservices.

- Port mapping needs to be addressed in an architecture where there are 3 containers in a cluster. What does this look like in kubernetes?
- When we create the K8s cluster and its associated loadbalancer, we can make additional port maps that will correspond to the pods running each microservice.


User accesses 8080 or 8081 or 8082
K3d Server
LoadBalancer Ports: 8080:8080 8081:8081 8082:8082
Service 8080:8080 8081:8081 8082:8082
Cluster
Node
Pod 1
Pod 2
Pod 3

1. Make sure you have a DockerHub Image of your application. This app requires 3 separate images, one for each microservice. This means there were 3 different Dockerfiles used to create all images. Then each image needed to be pushed up to DockerHub.
   a. docker build -t <imageNametaghere> <path to Dockerfile>
   b. docker tag <localimagename> fordonez20/<newtagnameforimageondockerhub>
   c. docker push fordonez20/<newtagname>

2. Create your Cluster and Loadbalancer using the following command
    a. k3d cluster create wishlist-cluster -p 8080:8080@loadbalancer -p 8081:8081@loadbalancer -p 8082:8082@loadbalancer

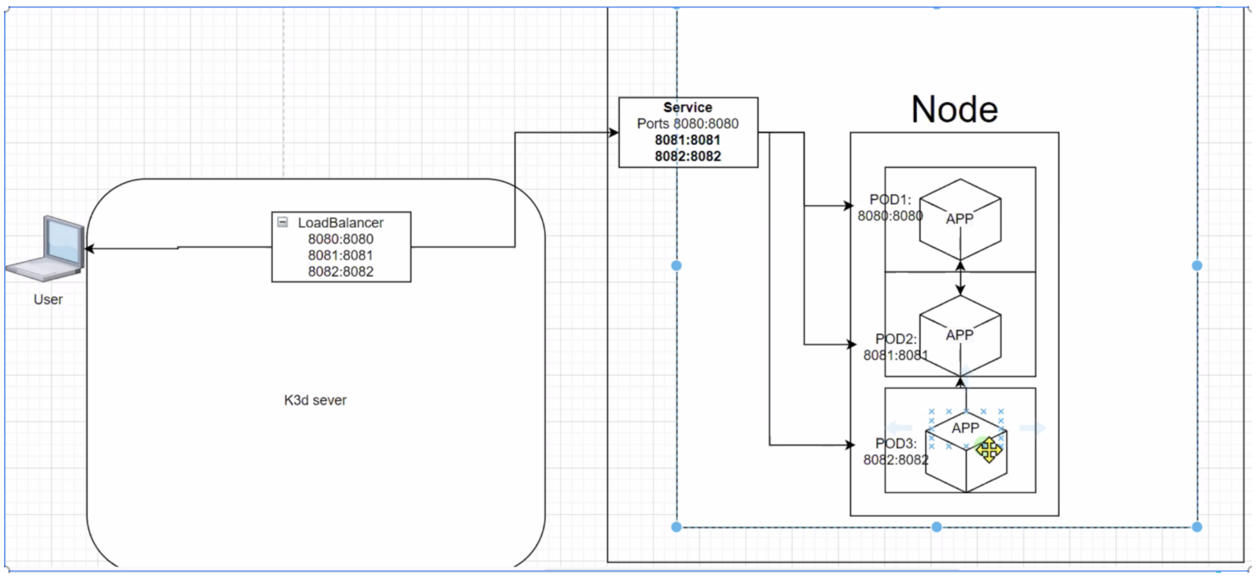
3. Now you need to make your yaml file. This will describe the 3 pods under a Deployment and it will describe the Service between your loadbalancer and the pods.

```
# Wishlist deployment yaml
kind: Deployment
apiVersion: apps/v1
metadata:
 name: wishlist-deployment
 labels:
   app: wishlist
spec:
 replicas: 3 #We always want more than 1 replica for HA
 selector:
   matchLabels:
     app: wishlist
 template:
   metadata:
     labels:
       app: wishlist
   spec:
     containers:
     - name: wishlist #1st container
       image: karthequian/wishlist:1.0 #Dockerhub image
```

```yaml
       ports:
       - containerPort: 8080 #Exposes the port 8080 of the container
       env:
       - name: PORT #Env variable key passed to container that is read by app
         value: "8080" # Value of the env port.
     - name: catalog #2nd container
       image: karthequian/wishlist-catalog:1.0
       ports:
       - containerPort: 8081
       env:
       - name: PORT
         value: "8081"
     - name: auth #3rd container
       image: karthequian/wishlist-auth:1.0
       ports:
       - containerPort: 8082
       env:
       - name: PORT
         value: "8082"
---
kind: Service
apiVersion: v1
metadata:
 name: wishlist-service
 namespace: default
spec:
 type: NodePort
 selector:
   app: wishlist
 ports:
```

```
- name: wishlist-port

  protocol: TCP

  port: 8080

  targetPort: 8080

- name: wishlist-auth-port

  protocol: TCP

  port: 8081

  targetPort: 8081

- name: wishlist-catalog-port

  protocol: TCP

  port: 8082

  targetPort: 8082
```

4. Build your pods with the following command:
   a. kubectl create -f <file-name.yaml>
5. Your application should now be completely deployed following the System Architecture below.

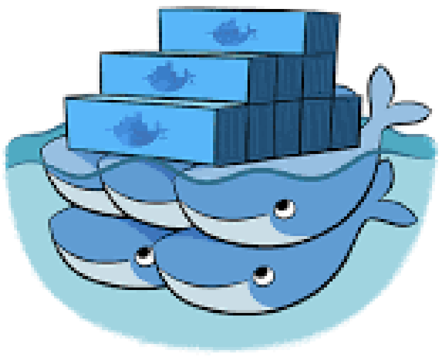


3x Dockerfiles used in Task 3:

```
#Dockerfile.list

FROM golang

MAINTAINER Karthik Gaekwad


RUN go get github.com/prometheus/client_golang/prometheus

RUN go get github.com/prometheus/client_golang/prometheus/promhttp

RUN go get github.com/gorilla/mux

COPY . /go/src/github.com/karthequian/wishlist

WORKDIR /go/src/github.com/karthequian/wishlist/list

RUN go get && go build -o /bin/list

ENTRYPOINT ["/bin/list"]
```

```
#Dockerfile.auth

FROM golang
MAINTAINER Karthik Gaekwad

RUN go get github.com/prometheus/client_golang/prometheus
RUN go get github.com/prometheus/client_golang/prometheus/promhttp
COPY . /go/src/github.com/karthequian/wishlist
WORKDIR /go/src/github.com/karthequian/wishlist/auth
RUN go get && go build -o /bin/auth

ENTRYPOINT ["/bin/auth"]
```

```
#Dockerfile.catalog
FROM golang
MAINTAINER Karthik Gaekwad

RUN go get github.com/prometheus/client_golang/prometheus
RUN go get github.com/prometheus/client_golang/prometheus/promhttp
RUN go get github.com/gorilla/mux
COPY . /go/src/github.com/karthequian/wishlist
WORKDIR /go/src/github.com/karthequian/wishlist/products
RUN go get && go build -o /bin/products

ENTRYPOINT ["/bin/products"]
```



kubernetes