

## Overview: Containerized App Deployment on Kubernetes

In this project, we built a complete, end-to-end Kubernetes deployment pipeline for a minimal web application, running on a local multi-node Kubernetes cluster using KIND (Kubernetes IN Docker).

Workflow – Create a Flask app > Dockerize the app > Store it in Docker registry > Create a Kubernetes deployment > Deploy the app > Expose the app using a NodePort-type service > Drain a node with 0 downtime > Uncordon the node > Validate the app.

Applications/Packages Installed:

- Docker
- Kubernetes
- KIND

Aliases Used:

1. alias kgp='kubectl get pods'
2. alias kgs='kubectl get svc'
3. alias kga='kubectl get all'
4. alias kaf='kubectl apply -f'
5. alias kdf='kubectl delete -f'
6. alias kctx='kubectl config current-context'
7. alias kctxs='kubectl config get-contexts'
8. alias kns='kubectl config set-context --current --namespace'
9. alias kgn='kubectl get nodes'

Project Directory structure

```
akhilrao@Akhils-MacBook-Air ln-app % tree
.
├── app.py
├── Dockerfile
├── k8s
│   ├── deployment.yaml
│   ├── kind-2node-cluster.yaml
│   ├── kind-config.yaml
│   └── service.yaml
├── print.sh
└── requirements.txt

2 directories, 8 files
```

**Create a Simple App:** We start by creating a minimal Flask app (app.py) that returns a simple message on the root route (/). This application will later be containerized and deployed to Kubernetes.

Created a basic Python Flask app that listens on port 5000 and returns a simple response.

```
File: app.py
-----
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return "🌈 Hello from your Kubernetes App!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
-----
```

```
File: requirements.txt
-----
flask
-----
```

### Create Docker Image:

A Dockerfile is written to containerize the Flask app. We build the Docker image using the docker build command and tag it with our Docker Hub username.

Created a basic Python Flask app that listens on port 5000 and returns a simple response

```
File: Dockerfile
-----
FROM python:3.10-slim
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
-----
```

## Push Docker Image to Docker Hub:

The image is pushed to Docker Hub so it can be pulled by the Kubernetes cluster later.

```
akhilrao@Akhils-MacBook-Air ln-app % vi requirements.txt
akhilrao@Akhils-MacBook-Air ln-app % vi Dockerfile
akhilrao@Akhils-MacBook-Air ln-app % docker build -t akhilrao199/flask-k8s-app:v1 .

[+] Building 20.8s (10/10) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 154B                                0.0s
=> [internal] load metadata for docker.io/library/python:3.10-slim 4.3s
=> [auth] library/python:pull token for registry-1.docker.io      0.0s
=> [internal] load .dockerignore                                   0.0s
=> => transferring context: 2B                                       0.0s
=> [1/4] FROM docker.io/library/python:3.10-slim@sha256:49454d2bf78a48f 13.7s
=> => resolve docker.io/library/python:3.10-slim@sha256:49454d2bf78a48f2 0.0s
=> => sha256:d0ac4bafb0af2562197fc84adfdcf99c11c69f5bd72f4db 248B / 248B 0.6s
=> => sha256:3937e61e7b960be410c9643954b0f8d2b1c2a9f05 15.58MB / 15.58MB 8.1s
=> => sha256:8a45c7e905d6f25747fd1b9286ccaf78e53af421e8 3.33MB / 3.33MB 3.1s
=> => sha256:b16f1b16678093d11ecfece1004207a40f9bc1b7 28.07MB / 28.07MB 12.9s
=> => extracting sha256:b16f1b16678093d11ecfece1004207a40f9bc1b7d9d1d16a 0.5s
=> => extracting sha256:8a45c7e905d6f25747fd1b9286ccaf78e53af421e86800b 0.1s
=> => extracting sha256:3937e61e7b960be410c9643954b0f8d2b1c2a9f055c59e88 0.2s
=> => extracting sha256:d0ac4bafb0af2562197fc84adfdcf99c11c69f5bd72f4dbd 0.0s
=> [internal] load build context                                   0.0s
=> => transferring context: 434B                                       0.0s
=> [2/4] WORKDIR /app                                             0.2s
=> [3/4] COPY . .                                                 0.0s
=> [4/4] RUN pip install -r requirements.txt                      1.9s
=> exporting to image                                             0.5s
=> => exporting layers                                               0.4s
=> => exporting manifest sha256:a6d256402b59528718c934af0312e9ae49c3db60 0.0s
=> => exporting config sha256:47756f867ae6167f0da1d75f89217b17ee4247b952 0.0s
=> => exporting attestation manifest sha256:1edceedcc3426f48d0f43e4a14fe 0.0s
=> => exporting manifest list sha256:b0da3924abca8d027aeff59d97414e9f380 0.0s
=> => naming to docker.io/akhilrao199/flask-k8s-app:v1             0.0s
=> => unpacking to docker.io/akhilrao199/flask-k8s-app:v1         0.1s
```

```
akhilrao@Akhils-MacBook-Air ln-app % docker push akhilrao199/flask-k8s-app:v1
The push refers to repository [docker.io/akhilrao199/flask-k8s-app]
0007d20ccbb2: Pushed
d0ac4bafb0af: Pushed
b16f1b166780: Pushed
8a45c7e905d6: Pushed
685636866349: Pushed
514619c0e36f: Pushed
4c7ba2634b78: Pushed
3937e61e7b96: Pushed
v1: digest: sha256:b0da3924abca8d027aeff59d97414e9f380a578a065d35df5cf4ab204c8f0cf7 size: 856
```

**Kind** (Kubernetes IN Docker): KIND simulates a real Kubernetes environment on your machine without the need for cloud infrastructure or heavy VMs. It creates Kubernetes nodes as Docker containers, allowing you to spin up multi-node clusters quickly and easily.

A custom kind-config.yaml was created with 2 nodes — one control plane and one worker. This setup allows us to test multi-node scheduling

```
File: kind-2node-cluster.yaml

kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
    extraPortMappings:
      - containerPort: 30080
        hostPort: 30080
  - role: worker
  - role: worker
```

```
kind create cluster --name flask-cluster --config kind-2node-cluster.yaml
```

```
akhilrao@Akhils-MacBook-Air k8s % kind create cluster --name flask-cluster --config kind-2node-cluster.yaml
Creating cluster "flask-cluster" ...
✓ Ensuring node image (kindest/node:v1.32.2)
✓ Preparing nodes
✓ Writing configuration
✓ Starting control-plane
✓ Installing CNI
✓ Installing StorageClass
✓ Joining worker nodes
Set kubectl context to "kind-flask-cluster"
You can now use your cluster with:

kubectl cluster-info --context kind-flask-cluster

Have a nice day! 🙌
```



**Create Deployment YAML:** A Kubernetes deployment.yaml file is created with replicas: 4, pod labels and Docker image pulled from registry.

```
File: deployment.yaml
-----
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app
spec:
  replicas: 4
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: "kubernetes.io/hostname"
          whenUnsatisfiable: ScheduleAnyway
          labelSelector:
            matchLabels:
              app: flask-app
      containers:
        - name: flask-container
          image: akhilrao199/flask-k8s-app:v1
          ports:
            - containerPort: 5000
```

```
Kubectl apply -f deployment.yaml
```

```
akhilrao@Akhils-MacBook-Air k8s % kubectl apply -f deployment.yaml
deployment.apps/flask-app created
```

### Create a Service to Expose the App:

We define a service.yaml file to expose the app using NodePort or Loadbalancer. In this project we are using Kind to provision nodes, so NodePort is preferred.

```
-----
File: service.yaml
-----
apiVersion: v1
kind: Service
metadata:
  name: flask-service
spec:
  selector:
    app: flask-app
  type: NodePort
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
      nodePort: 30080
-----
```

```
Kubectl apply -f service.yaml
```

```
[akhilrao@Akhils-MacBook-Air k8s % kubectl apply -f service.yaml
service/flask-service created
```

```
Kubectl get pods -o wide
Kubectl get nodes
```

```
akhilrao@Akhils-MacBook-Air k8s % kubectl get pods -o wide
kubectl get nodes
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE                                NOMINATED NODE   READINESS GATES
flask-app-5867bff4b7-4xfgw          0/1     ContainerCreating   0       7s    <none>          flask-cluster-worker              <none>           <none>
flask-app-5867bff4b7-8ks19          0/1     ContainerCreating   0       7s    <none>          flask-cluster-worker2             <none>           <none>
flask-app-5867bff4b7-8s4hq          0/1     ContainerCreating   0       7s    <none>          flask-cluster-worker2             <none>           <none>
flask-app-5867bff4b7-rp8hp          0/1     ContainerCreating   0       7s    <none>          flask-cluster-worker              <none>           <none>
NAME                                STATUS   ROLES    AGE   VERSION
flask-cluster-control-plane         Ready   control-plane   6m7s   v1.32.2
flask-cluster-worker                Ready   <none>         5m57s   v1.32.2
flask-cluster-worker2               Ready   <none>         5m57s   v1.32.2
```

The deployment is scaled to 4 replicas. Using kubectl get pods -o wide, we confirm that 2 pods are scheduled per node (even distribution).

## Node Maintenance:

We simulate maintenance, Kubernetes upgrades by cordoning and draining a Node. This safely evicts all pods from a node, which are rescheduled on another node in the cluster, if no nodes are available, there might be a chance of your application downtime .

```
kubectl cordon flask-cluster-worker
kubectl drain flask-cluster-worker --ignore-daemonsets --delete-emptydir-data
```

```
akhilrao@Akhils-MacBook-Air k8s % kubectl drain flask-cluster-worker --ignore-daemonsets --delete-emptydir-data
node/flask-cluster-worker cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/kindnet-r7819, kube-system/kube-proxy-77rjd
evicting pod default/flask-app-5867bff4b7-rp8hp
evicting pod default/flask-app-5867bff4b7-4xfcw
pod/flask-app-5867bff4b7-rp8hp evicted
pod/flask-app-5867bff4b7-4xfcw evicted
node/flask-cluster-worker drained
```

```
akhilrao@Akhils-MacBook-Air k8s % kgn
```

NAME	STATUS	ROLES	AGE	VERSION
flask-cluster-control-plane	Ready	control-plane	10m	v1.32.2
flask-cluster-worker	Ready,SchedulingDisabled	<none>	10m	v1.32.2
flask-cluster-worker2	Ready	<none>	10m	v1.32.2

Validate if the 2 pods from Node1 moved to Node2 and if draining is successful:

Kubectl get pods -o wide

```
akhilrao@Akhils-MacBook-Air k8s % kubectl get pods -o wide
kubectl get nodes
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GAT
flask-app-5867bff4b7-8ks19	1/1	Running	0	5m8s	10.244.1.3	flask-cluster-worker2	<none>	<none>
flask-app-5867bff4b7-8s4hq	1/1	Running	0	5m8s	10.244.1.2	flask-cluster-worker2	<none>	<none>
flask-app-5867bff4b7-rsd56	1/1	Running	0	97s	10.244.1.5	flask-cluster-worker2	<none>	<none>
flask-app-5867bff4b7-x9twm	1/1	Running	0	97s	10.244.1.4	flask-cluster-worker2	<none>	<none>

  

NAME	STATUS	ROLES	AGE	VERSION
flask-cluster-control-plane	Ready	control-plane	11m	v1.32.2
flask-cluster-worker	Ready	<none>	10m	v1.32.2
flask-cluster-worker2	Ready	<none>	10m	v1.32.2

Uncordon the Node and place the pods back:

```
akhilrao@Akhils-MacBook-Air k8s % kubectl uncordon flask-cluster-worker
node/flask-cluster-worker uncordoned
```

Kubectl get nodes

```
akhilrao@Akhils-MacBook-Air ln-app % kgn
```

NAME	STATUS	ROLES	AGE	VERSION
flask-cluster-control-plane	Ready	control-plane	121m	v1.32.2
flask-cluster-worker	Ready	<none>	121m	v1.32.2
flask-cluster-worker2	Ready	<none>	121m	v1.32.2

You have multiple ways to get back the pods from Node2 to Node1.

**Rolling Restart pods:** As we have set the app's replicas as 4, when we hit the below command, it schedules the pods according to the available nodes

```
kubectl rollout restart deployment flask-app
```

```
akhilrao@Akhils-MacBook-Air k8s % kubectl rollout restart deployment flask-app
deployment.apps/flask-app restarted
akhilrao@Akhils-MacBook-Air k8s % kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS
flask-app-58b9d7ccb6-6kncj	1/1	Terminating	0	89s	10.244.1.6	flask-cluster-worker2	<none>	<none>
flask-app-58b9d7ccb6-skhs	1/1	Terminating	0	89s	10.244.2.6	flask-cluster-worker	<none>	<none>
flask-app-58b9d7ccb6-vhtxx	1/1	Terminating	0	90s	10.244.2.5	flask-cluster-worker	<none>	<none>
flask-app-58b9d7ccb6-vkqsl	1/1	Terminating	0	90s	10.244.2.4	flask-cluster-worker	<none>	<none>
flask-app-795c55cb9-25rst	1/1	Running	0	3s	10.244.2.7	flask-cluster-worker	<none>	<none>
flask-app-795c55cb9-5xz6m	1/1	Running	0	4s	10.244.1.8	flask-cluster-worker2	<none>	<none>
flask-app-795c55cb9-7dtll	1/1	Running	0	4s	10.244.1.7	flask-cluster-worker2	<none>	<none>
flask-app-795c55cb9-bpz86	1/1	Running	0	3s	10.244.1.9	flask-cluster-worker2	<none>	<none>

**Scale the pods:** Scale-in the pods to 2 and scale-out back to 4 using command

```
kubectl scale deployment/flask-app --replicas=2
```

```
kubectl scale deployment/flask-app --replicas=4
```

**Delete the pods:** If you delete all the pods, you might have app downtime, you can carefully delete 2 pods in this case, so that replica set will create them back with balance in nodes as shown below.

```
Kubectl delete pod <pod-A> <pod-B>
```

```
akhilrao@Akhils-MacBook-Air k8s % kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE
flask-app-795c55cb9-jpqq4	1/1	Running	0	20m
flask-app-795c55cb9-khmt4	1/1	Running	0	20m
flask-app-795c55cb9-l7tkn	1/1	Running	0	20m
flask-app-795c55cb9-srhv4	1/1	Running	0	20m

```
akhilrao@Akhils-MacBook-Air k8s % kubectl delete pod flask-app-795c55cb9-jpqq4 flask-app-795c55cb9-srhv4
pod "flask-app-795c55cb9-jpqq4" deleted
pod "flask-app-795c55cb9-srhv4" deleted
```

```
Kubectl get pods -o wide
```

```
akhilrao@Akhils-MacBook-Air k8s % kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS
flask-app-795c55cb9-5m7b6	1/1	Running	0	32s	10.244.2.11	flask-cluster-worker	<none>	<none>
flask-app-795c55cb9-9kp4q	1/1	Running	0	32s	10.244.2.10	flask-cluster-worker	<none>	<none>
flask-app-795c55cb9-khmt4	1/1	Running	0	22m	10.244.1.11	flask-cluster-worker2	<none>	<none>
flask-app-795c55cb9-l7tkn	1/1	Running	0	22m	10.244.1.10	flask-cluster-worker2	<none>	<none>

Validate the application:





**Destroy all infrastructure that we created:**

```
kubectl delete -f deployment.yaml  
kubectl delete -f service.yaml  
kind delete cluster
```