YAML Basics

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: curl-pod
spec:
  containers:
  - name: curl-container
    image: curlimages/curl:latest
    command:
    - sleep
    - "3600"
```


Python

```python
pod = {
    "apiVersion": "v1",
    "kind": "Pod",
    "metadata": {
        "name": "curl-pod"
    },
    "spec": {
        "containers": [
            {
                "name": "curl-container",
                "image": "curlimages/curl:latest",
                "command": [
                    "sleep",
                    "3600"
                ]
            }
        ]
    }
}
```

DOCKER BASICS

For those who are new to Docker:

```
docker run -d -p 88:80 --name webserver nginx
docker ps
curl localhost:88
```

Let's play a game:

```
docker run -it dyego/snake-game
```

Starting Minikube

Start Minikube from scratch:

```
minikube delete
minikube start
```

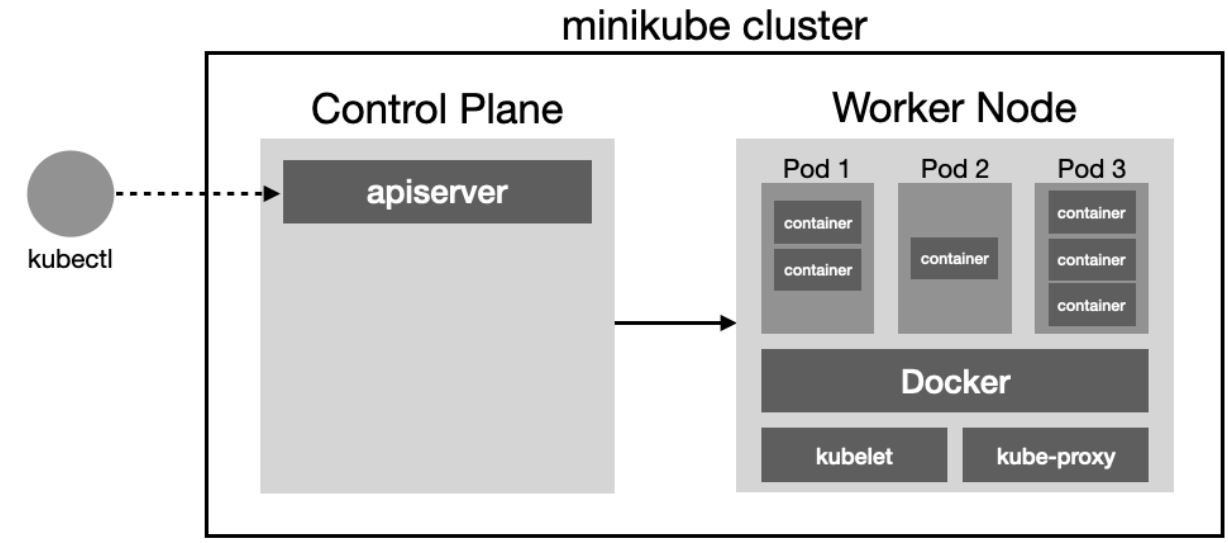
Shows the drivers used by Minikube

```
minikube profile list
```

minikube is using Docker for both the VM driver as well as the Runtime.

View the status of Minikube

```
minikube status
```



The Minikube cluster contains both the control plane and the worker node in a single node (machine)

Stop Minikube

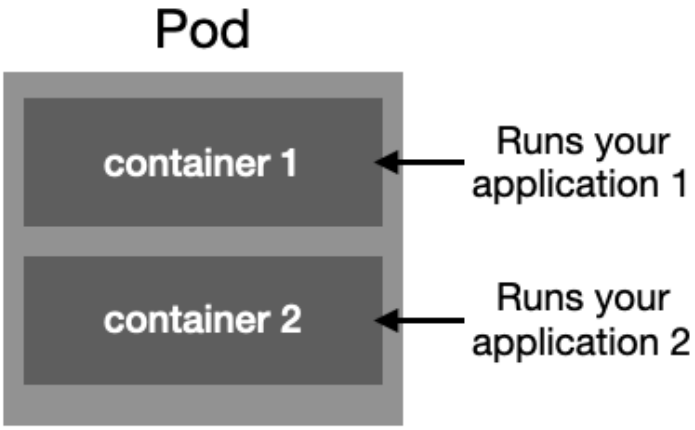
```
minikube stop
minikube delete
```

Starting with multiple Nodes

```
minikube start --nodes 3
minikube profile list
kubectl get nodes
```

Pods

A pod is the smallest deployable and manageable unit in Kubernetes. It represents a single instance of a running process in a cluster and can contain one or more containers (usually Docker containers) that share the same network namespace and storage volumes



nginxpod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx       # pod name
  labels:
    name: nginx     # key/value pair
spec:
  containers:
  - name: nginx     # container name
    image: nginx
    ports:
      - containerPort: 80
    imagePullPolicy: Always
```

Runs the nginx container, listens at port 80

Pod name is usually used in commands like: ***kubectl get pod nginx****, or* ***kubectl logs nginx***

Labels are optional, but are useful for selecting pods or grouping pods, such as:

```
selector:
  name: nginx
```

Apply the configuration to the cluster:

```
kubectl apply -f nginxpod.yaml
```

List all the pods:

```
kubectl get pods
```

Execute a command in a pod:

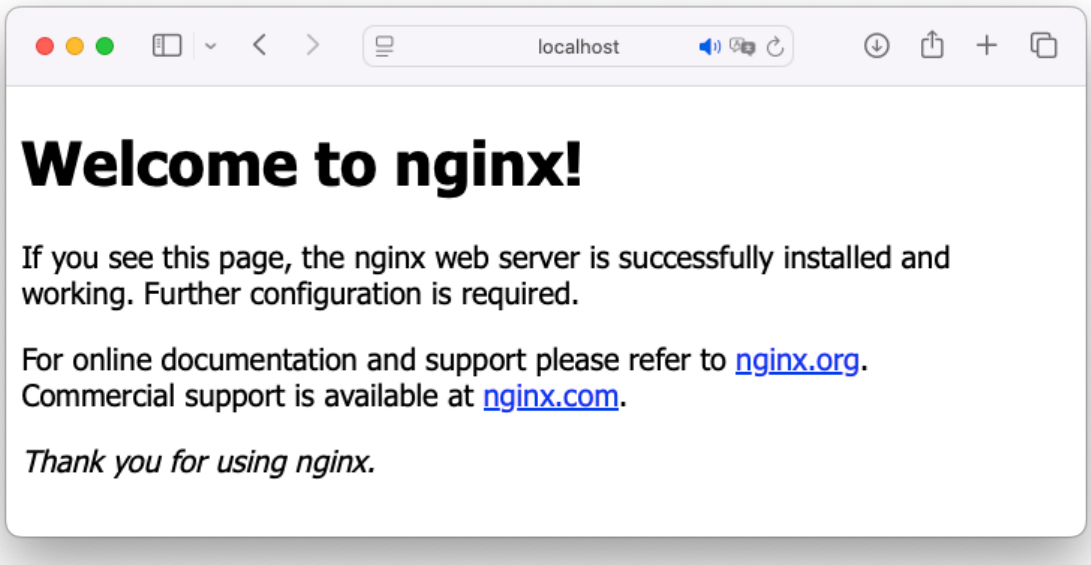
```
kubectl exec -it nginx -- bash
```

```
root@nginx:/# curl localhost
```

Port forwarding:

```
kubectl port-forward nginx 8080:80
```

http://localhost:8080

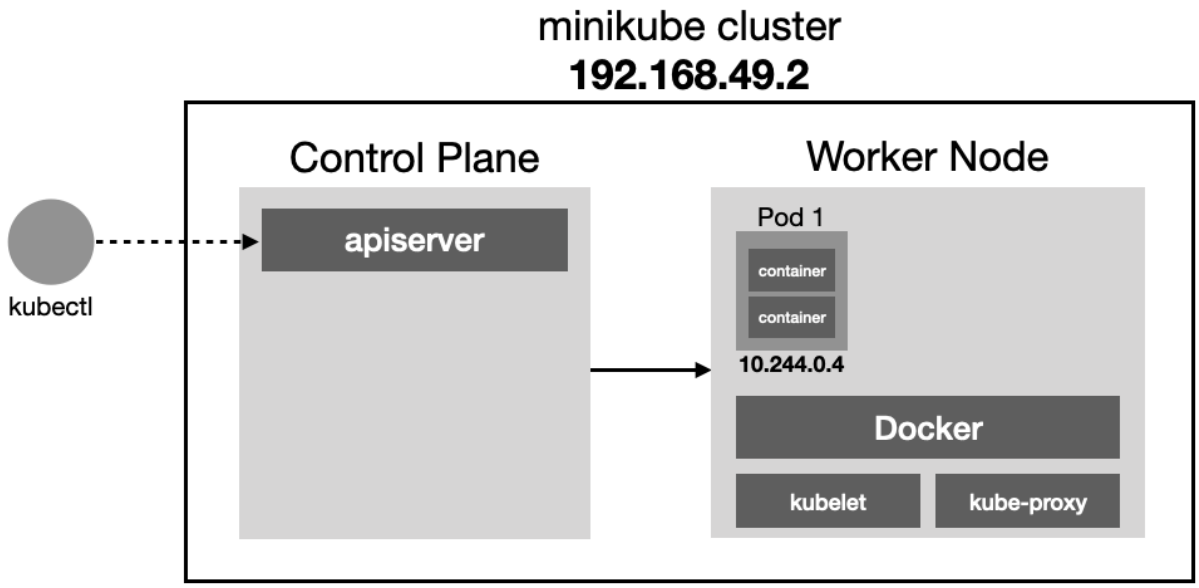


Getting the IP address of the Minikube cluster:

```
minikube ip
```

Viewing the IP addresses of each pod in the cluster

```
kubectl get pods -o wide
```



Delete a pod

```
kubectl delete pod nginx
```

OR

```
kubectl delete -f nginxpod.yaml
```

Deployments

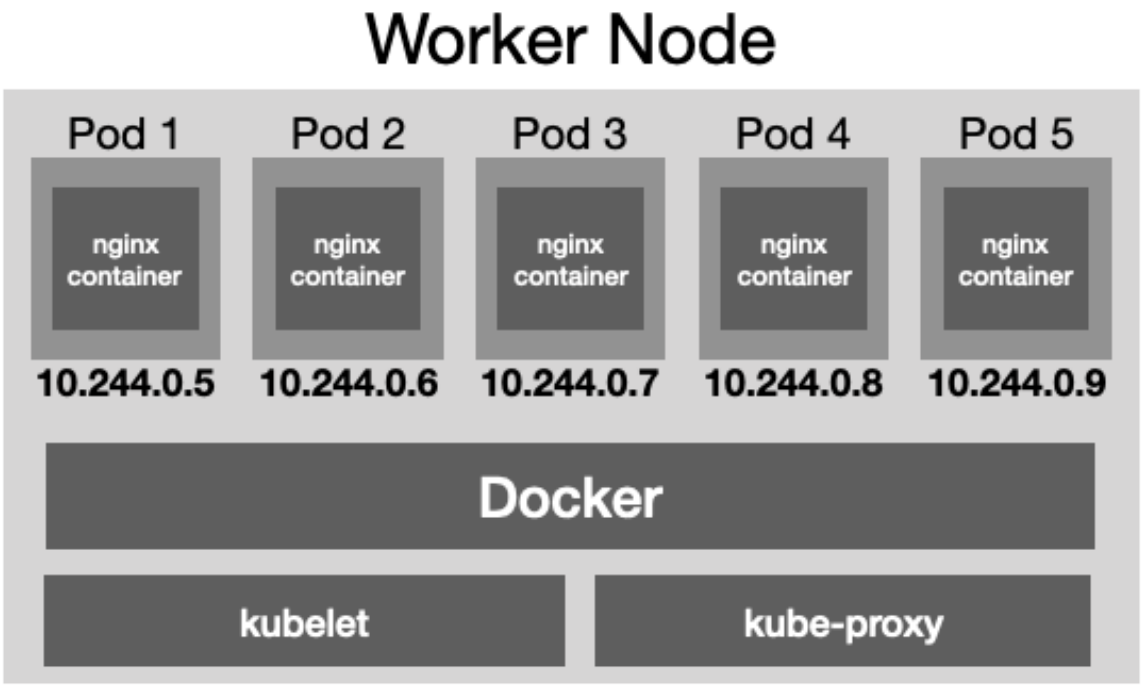
*A **deployment** in Kubernetes is a resource object used to manage the lifecycle of pods. It provides declarative updates to applications, such as rolling out new features or changes, updating existing ones, and rolling back to previous versions if necessary.*

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver-deployment
  labels:
    app: webserver
spec:
  replicas: 5
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "150Mi"
          limits:
            memory: "300Mi"
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 3
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
```

```
          periodSeconds: 10
```

```
kubectl apply -f deployment.yaml
kubectl get deployments
kubectl get pods
kubectl get pods -o wide
```



Delete a particular pod in the deployment

```
kubectl delete pod webserver-deployment-55f8f4f47c-5tn7b
```

Observe that the deleted pod will be created again automatically:

```
kubectl get pods -o wide
```

Scale now the number of pods in the deployment

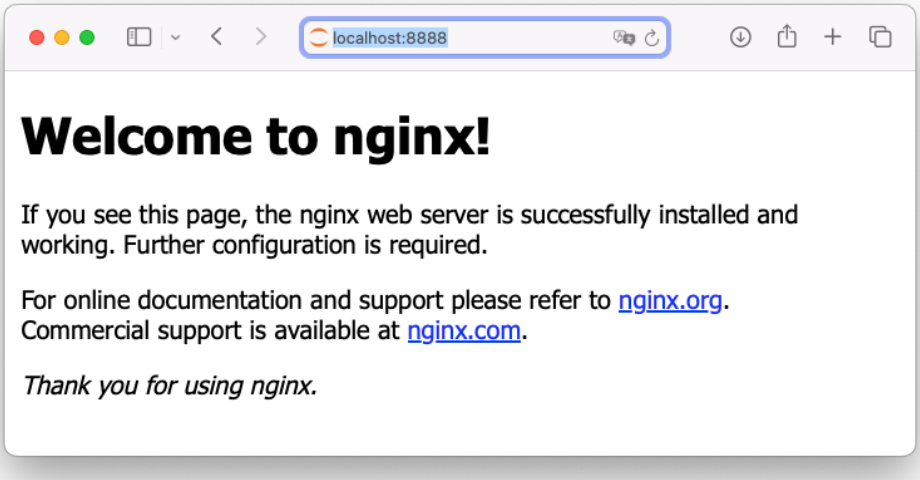
```
kubectl scale deployment webserver-deployment --replicas=3
```

```
kubectl get pods -o wide
```

Create port forwarding to a particular pod:

```
kubectl port-forward webserver-deployment-55f8f4f47c-62v66 8888:80
```

http://localhost:8888



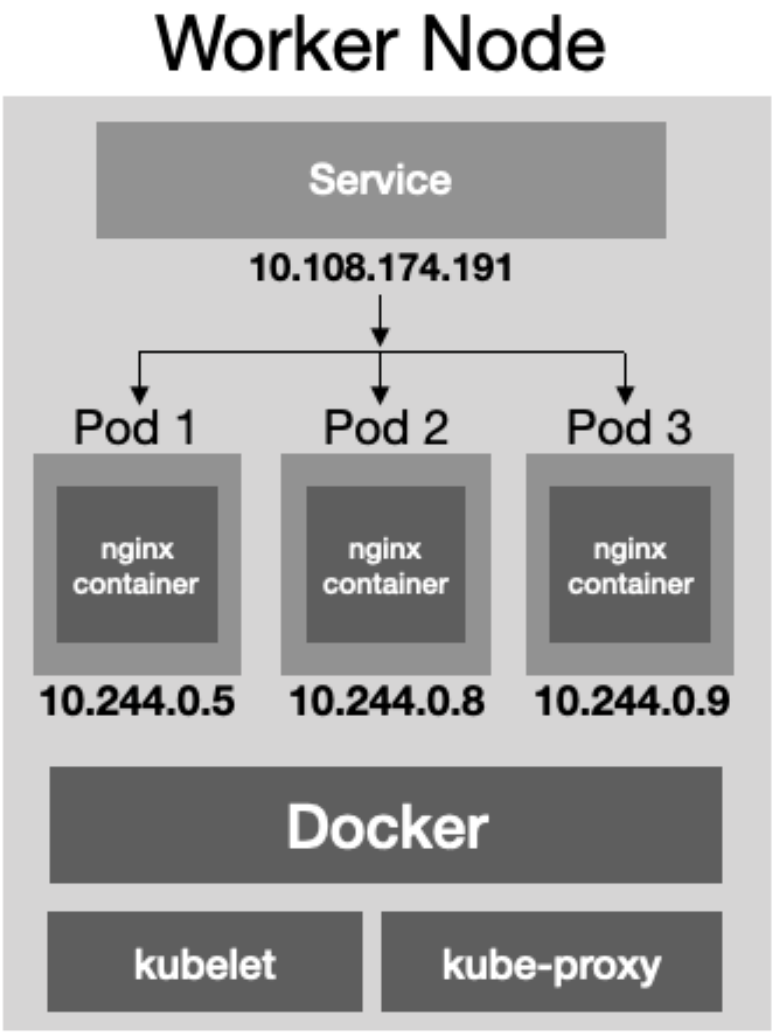
Services

The aim of a Service is to allow stable communication between pods — regardless of nodes. ***Cross-node communications****.*

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
  labels:
    svc: web-service
spec:
  selector:
    app: webserver
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

```
kubectl apply -f service.yaml
kubectl get services
kubectl describe service web-service
```

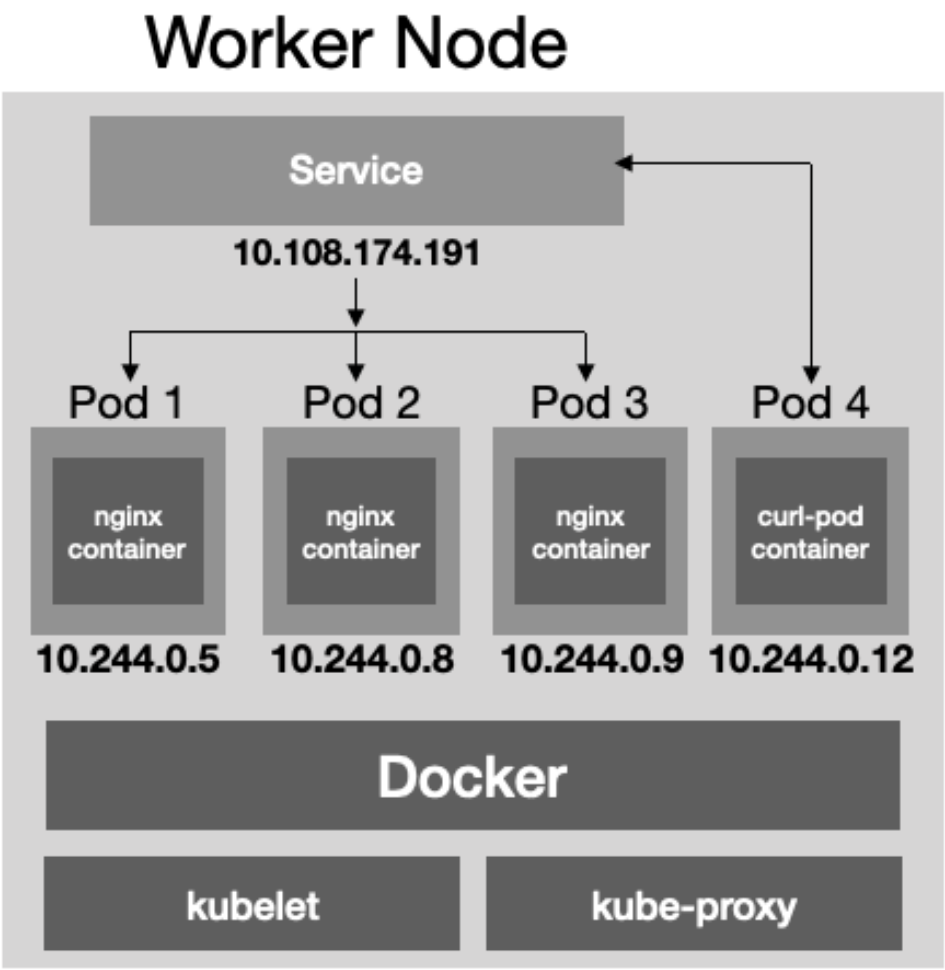


Creating another Pod to add to the node:

curlpod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: curl-pod
spec:
  containers:
  - name: curl-container
    image: curlimages/curl:latest
    command:
    - sleep
    - "3600"  # Sleep to keep the
              # container running
```

```
kubectl apply -f curlpod.yaml
kubectl get pods -o wide
```



Use the curl-pod to access the nginx server through the service:

```
kubectl exec -it curl-pod  -- curl web-service
```

OR

```
kubectl exec -it curl-pod  -- curl 10.104.172.185
```

(the ip address of the service)

Use port forwarding to allow the local computer to access the service:

```
kubectl port-forward service/web-service 8080:80
curl localhost:8080
```

Note that port forwarding is usually for debugging. The right way is to use a NodePort.

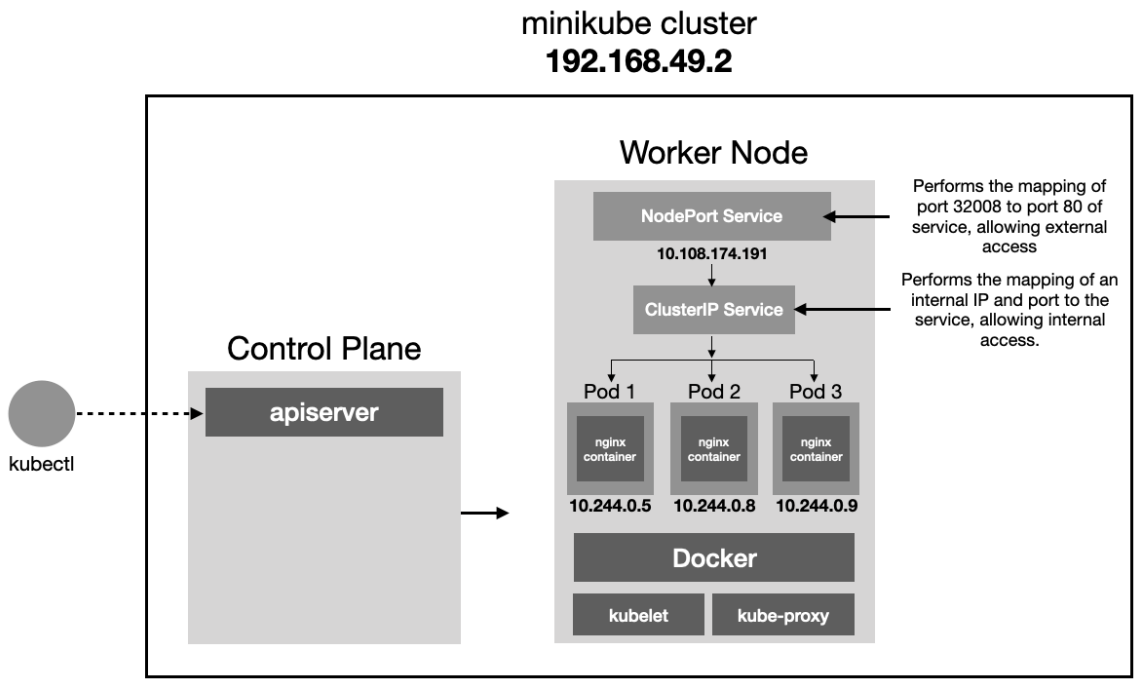
NodePort

A NodePort is a type of service. Its main use is to expose a Service outside the cluster on a port.

service_nodeport.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
  labels:
    svc: web-service
spec:
  type: NodePort
  selector:
    app: webserver
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 32008
```

```
kubectl apply -f service_nodeport.yaml
kubectl get service web-service
```

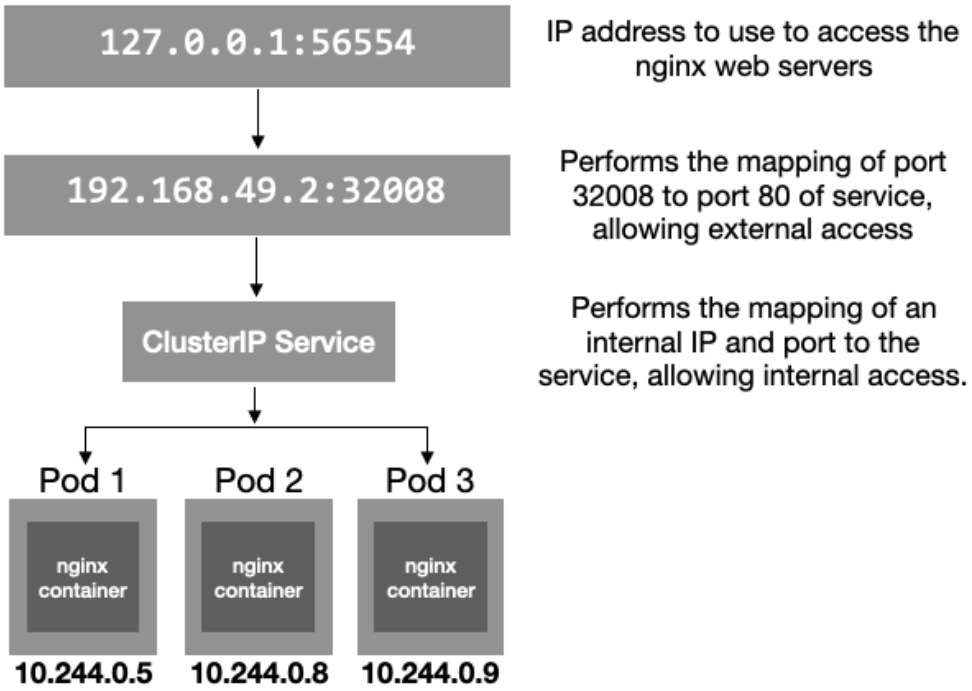


By right, you should be able to access the web server using http://192.168.49.2:32008. But if you're using Docker driver, NodePort may not be exposed properly by default.

Using the following, this command works around any IP/port mess by:

Looking up the correct IP + port

```
minikube service web-service
|-----------|-------------|-------------|---------------------------|
| NAMESPACE |    NAME     | TARGET PORT |            URL            |
|-----------|-------------|-------------|---------------------------|
| default   | web-service |          80 | http://192.168.49.2:32008 |
|-----------|-------------|-------------|---------------------------|
  Starting tunnel for service web-service.
|-----------|-------------|-------------|------------------------|
| NAMESPACE |    NAME     | TARGET PORT |          URL           |
|-----------|-------------|-------------|------------------------|
| default   | web-service |             | http://127.0.0.1:62531 |
|-----------|-------------|-------------|------------------------|
```



Each pod will take turn to return the HTML.

*On the VM, you can now use **curl http://192.168.58.2:32008***

ConfigMaps

A ConfigMap in Kubernetes is an API object used to store non-confidential configuration data in key-value pairs. ConfigMaps allow you to decouple configuration artifacts from container images, enabling you to keep application configuration separate from your containerized application code. They are often used to store configuration data such as application settings, environment variables, command-line arguments, and configuration files. It is commonly used when deploying your database servers in Kubernetes.

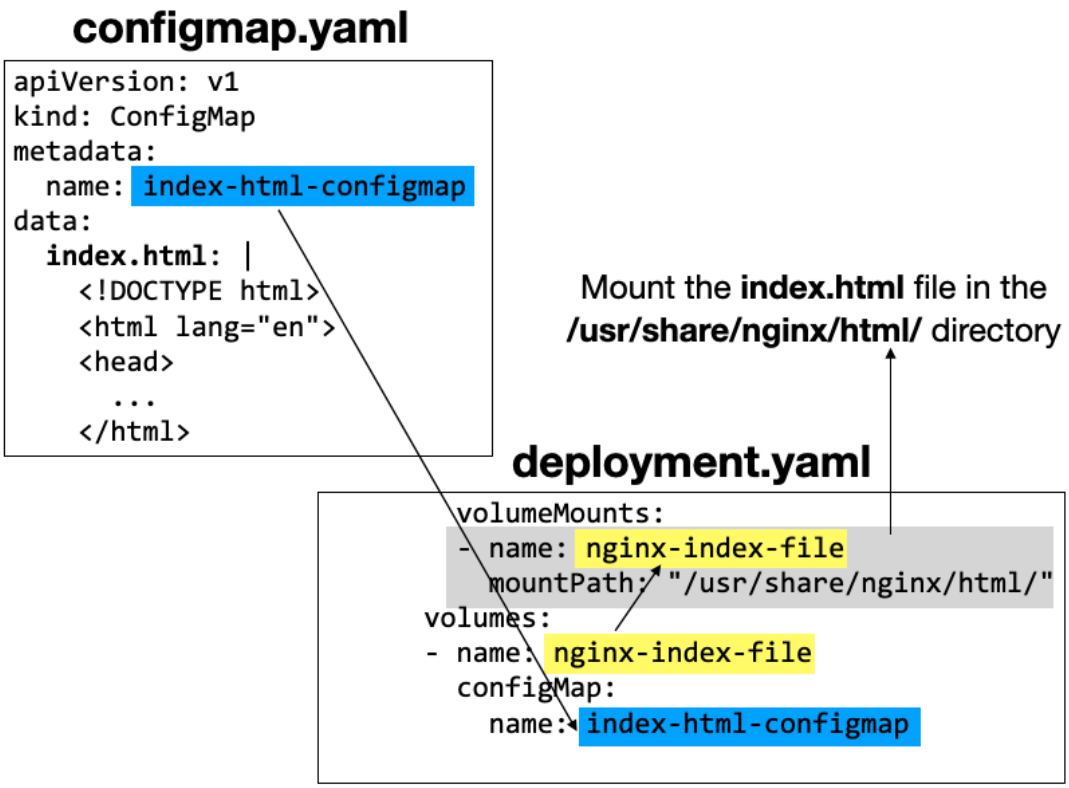
configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: index-html-configmap
data:
  index.html: |
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport"
        content="width=device-width,
                 initial-scale=1.0">
      <title>Hello Kubernetes</title>
      <style>
        .tray {
          display: inline-block;
          background-color: LightCoral;
          color: white;
          padding: 10px;
          border-radius: 5px;
        }
      </style>
    </head>
    <body>
      <center>
        <h1 class="tray">Hello,
Kubernetes!</h1>
      </center>
    </body>
    </html>
```

```
kubectl apply -f configmap.yaml
```

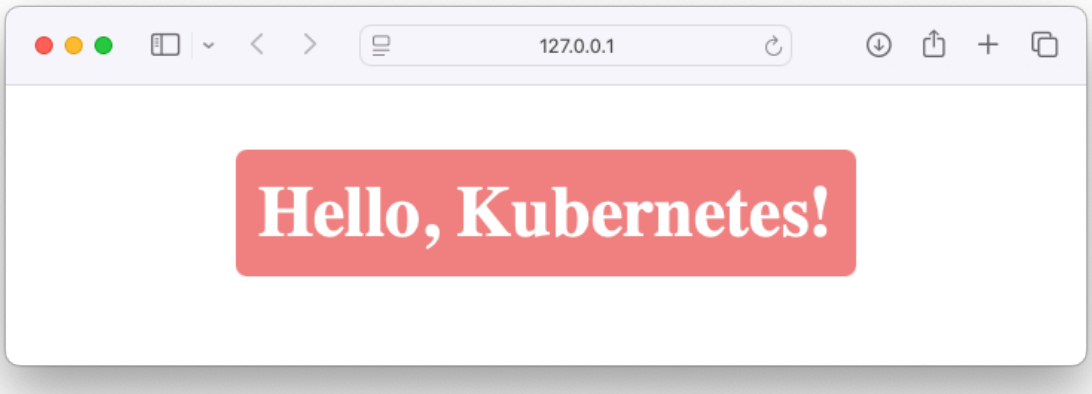
deployment.yaml

```
...
...
        volumeMounts:
        - name: nginx-index-file
          mountPath: "/usr/share/nginx/html/"
      volumes:
      - name: nginx-index-file
        configMap:
          name: index-html-configmap
```



If your ConfigMap contains a file named **index.html**, and you mount it to **/usr/share/nginx/html/**, nginx will automatically serve that file when someone hits the root URL (/).

```
kubectl apply -f deployment.yaml
kubectl port-forward service/web-service 8080:80
```



Ingress

Ingress is a Kubernetes resource that manages external access to services inside a cluster, typically over HTTP or HTTPS.

```
minikube addons enable ingress
```

ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webservice-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-
target: /$1
spec:
  rules:
  - host: "ws.localdomain"
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: web-service
            port:
              number: 80
```

```
kubectl apply -f ingress.yaml
kubectl get pods -n ingress-nginx
```

```
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-qrzm4        0/1     Completed   0          3m2s
ingress-nginx-admission-patch-pcbg6         0/1     Completed   1          3m2s
ingress-nginx-controller-77669ff58-vfxzm    1/1     Running     0          3m2s
```

```
minikube service list
```

```
|---------------|-------------------------------------|--------------|-----------------
----------|
|   NAMESPACE   |                NAME                 | TARGET PORT  |            URL
|
|---------------|-------------------------------------|--------------|-----------------
----------|
| default       | kubernetes                          | No node port |
| default       | web-service                         | No node port |
| ingress-nginx | ingress-nginx-controller            | http/80      |
http://192.168.58.2:32486 |
|               |                                     | https/443    |
http://192.168.58.2:30935 |
```

```
| ingress-nginx | ingress-nginx-controller-admission | No node port |
| kube-system   | kube-dns                           | No node port |
|---------------|------------------------------------|--------------|-----------------
----------|
```

Access it locally:

```
curl -H 'Host: ws.localdomain' http://192.168.58.2:32486
```

Making it available on the web:

```
[student@ip-172-30-1-100 ~]$ minikube tunnel --bind-address=172.30.1.100
```

In another Terminal:

```
nano ingress.yaml
```

ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webservice-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-
target: /$1
spec:
  rules:
  - host:
"myweb.msshanghai20251.neueda.com"
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: web-service
            port:
              number: 80
```

```
kubectl apply -f ingress.yaml
kubectl get ingress
```

*Use a browser: http://**myweb.msshanghai20251.neueda.com***

Using MySQL

mysql.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        # image: mysql:latest
        image: mysql:5.7.39-oracle
        env:
          - name: MYSQL_ROOT_PASSWORD
            value: topsecret
        ports:
          - containerPort: 3306
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-nodeport
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
      nodePort: 30001   # 30000 to 32767
  type: NodePort
```

```
kubectl apply -f mysql.yaml
sudo yum install mysql
minikube service list
```

```
|-------------|-----------------------------------|-------------|---------------------------|
|  NAMESPACE  |               NAME                | TARGET PORT |            URL            |
```

```
|--------------|-----------------------------------|--------------|--------------------------|
| default      | kubernetes                        | No node port |                          |
| default      | mysql-nodeport                    |         3306 | http://192.168.58.2:30001 |
| default      | web-service                       | No node port |                          |
| ingress-nginx | ingress-nginx-controller          | http/80      | http://192.168.58.2:32486 |
|              |                                   | https/443    | http://192.168.58.2:30935 |
| ingress-nginx | ingress-nginx-controller-admission | No node port |                          |
| kube-system  | kube-dns                          | No node port |                          |
|--------------|-----------------------------------|--------------|--------------------------|
```

GET THE IP address

```
mysql -h 192.168.58.2 -P30001  -u root -p
```

*Password is **topsecret***

Using MySQL 8

Use version 5

image: mysql:5.7.39-oracle

Instead of:

 # image: mysql:8

As version 8 will have authentication error.

To fix this, use ConfigMap:

Add the following lines to mysql.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
       app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:8.0
        # image: mysql:5.7.39-oracle
        env:
          - name: MYSQL_ROOT_PASSWORD
            value: topsecret
        ports:
          - containerPort: 3306
        volumeMounts:
        - name: mysql-config-volume                          # specify volume name
```

```
          mountPath: /etc/mysql/conf.d/default_auth.cnf  # path to mount file
          subPath: default_auth                          # name of config
      volumes:
      - name: mysql-config-volume                        # volume name
        configMap:
          name: mysql-config                             # name of ConfigMap
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-nodeport
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
      nodePort: 30001   # 30000 to 32767
  type: NodePort
```

```
$ nano configmap_mysql.yaml
```

configmap_mysql.yaml

```
# config-map.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  # name of ConfigMap. This will be referred from volume definition
  name: mysql-config
  labels:
    app: mysql
data:
  # default_auth is the name of config. This will be referred from volume mount
definition
  default_auth: |
    [mysqld]
    default_authentication_plugin=mysql_native_password
```

```
$ kubectl apply -f configmap_mysql.yaml
$ kubectl apply -f mysql.yaml
$ mysql -h 192.168.58.2 -P30001  -u root -p
```

Project – Creating a REST API Pod

What we will do in this project is to

- Create a Docker image containing our REST API
- Publish the image to the Docker hub
- Use the image and run it as a Pod
- Deploy the pod using NodePort

```
sudo yum -y install python-pip
pip install flask
```

```
mkdir RESTAPI
cd RESTAPI
nano app.py
```

app.py

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def hello():
    return jsonify({'message': 'Hello,
this is a REST API!'})

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

```
python app.py
```

On another Terminal

```
curl localhost:5000
```

Packaging the Docker Image

Back in the original terminal, stop the REST API.

```
nano requirements.txt
```

requirements.txt

```
Flask==2.0.2
Werkzeug==2.0.2
```

```
nano Dockerfile
```

Dockerfile

```
FROM python:3.9

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r
requirements.txt

COPY . .

CMD ["python", "app.py"]
```

```
docker build -t my-flask-app .
docker run -p 5000:5000 my-flask-app
```

Publishing Image to Docker Hub

Go to https://hub.docker.com and sign up for an account.

```
docker login -u weimenglee
docker tag my-flask-app:latest weimenglee/my-flask-app:latest
docker push weimenglee/my-flask-app:latest
```

View it on: https://hub.docker.com/r/**weimenglee**/my-flask-app

Important *- You need to set the image to* ***public****:*

https://hub.docker.com/repository/docker/weimenglee/my-flask-app/settings

Creating a Pod

```
nano flask-app-pod.yaml
```

flask-app-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: flask-app-pod
spec:
  containers:
    - name: flask-app-container
```

```
      image: weimenglee/my-flask-
app:latest
      ports:
        - containerPort: 5000
```

```
kubectl apply -f flask-app-pod.yaml
kubectl get pods
kubectl port-forward flask-app-pod 5000:5000
```

To view error generated by Pod: kubectl describe pod flask-app-pod

```
curl localhost:5000
```

Deploying Using NodePort

```
nano flask-app-pod.yaml
```

flask-app-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: flask-app-pod
  labels:
    app: flask-app-pod
spec:
  containers:
    - name: flask-app-container
      image: weimenglee/my-flask-
app:latest
      ports:
        - containerPort: 5000
---
apiVersion: v1
kind: Service
metadata:
  name: flask-app-service
spec:
  type: NodePort
  selector:
    app: flask-app-pod
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
      nodePort: 30100
# specify the NodePort you want to use
#   (range: 30000-32767)
```

```
kubectl apply -f flask-app-pod.yaml
kubectl get svc
minikube service flask-app-service
```

```
curl http://192.168.58.2:30100
```