



Week 11 : SOFTWARE DEVELOPMENT TOOLS AND ENVIRONMENTS

Dockerfile

- Dockerfile is instructions to build Docker Image
 - How to run commands
 - Add files or directories
 - Create environment variables
 - What process to run when launching container
- Result from building Dockerfile is Docker Image

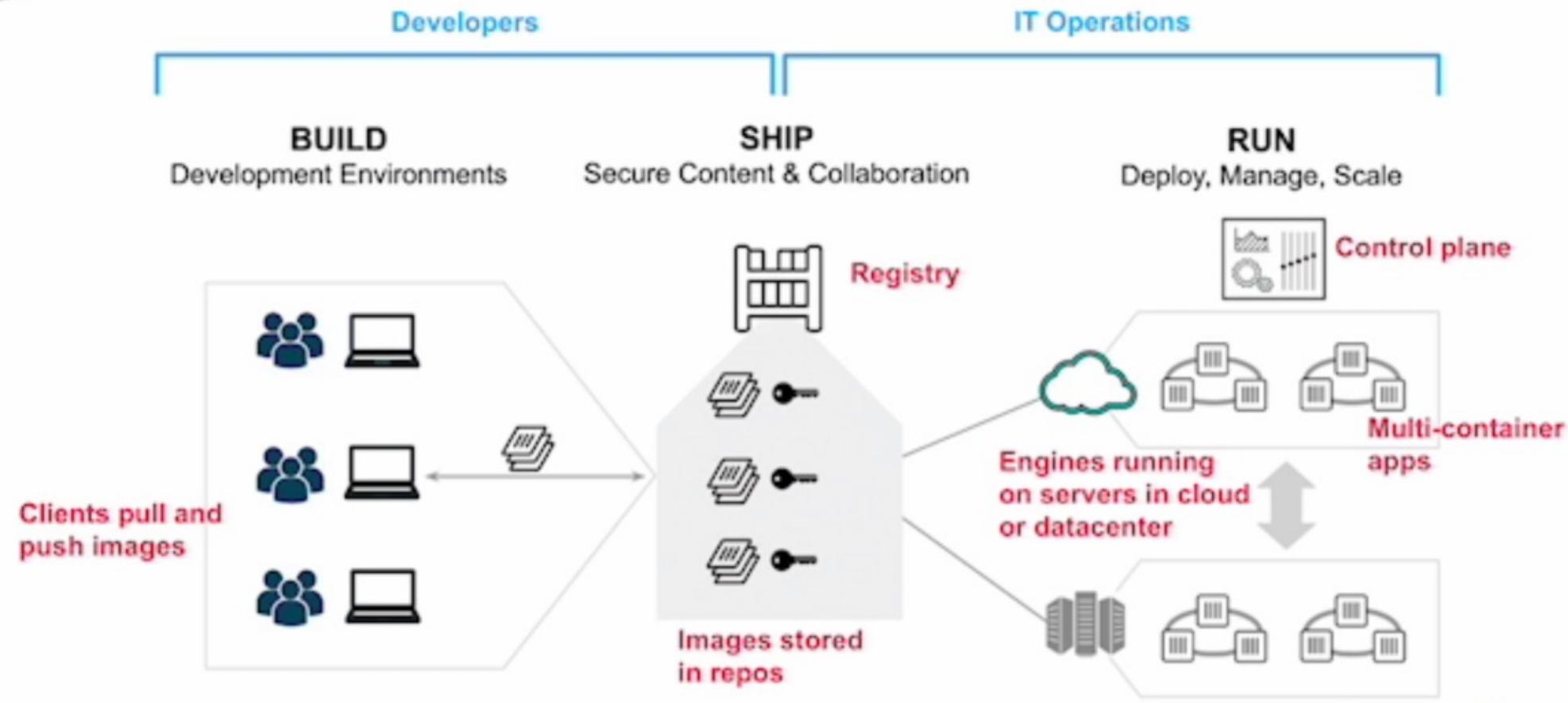


Dockerfile

Sample Dockerfile

```
FROM node:14.15.0-alpine3.12 ← OS + System Packages
COPY . /nodejs/.           ← Source Code
WORKDIR /nodejs
RUN npm install             ← Library Dependencies
ENV VERSION 1.0            ← Configuration
EXPOSE 8081
CMD ["node", "/nodejs/main.js"]
```

Docker and DevOps





Introduction to Kubernetes

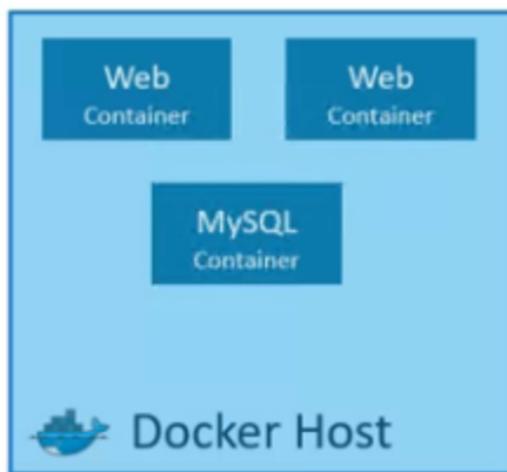
What is Kubernetes?

- **Kubernetes**, in Greek, means the Helmsman, or pilot of the ship, pilot of a ship of containers
- Kubernetes is a software written in **Go** for **automating deployment, scaling, and management** of containerized applications
- Focus on manage **applications**, not machines
- Open source, open API container **orchestrator**
- Supports **multiple cloud** and **bare-metal** environments
- Inspired and informed by 15 years of Google's experiences and internal systems

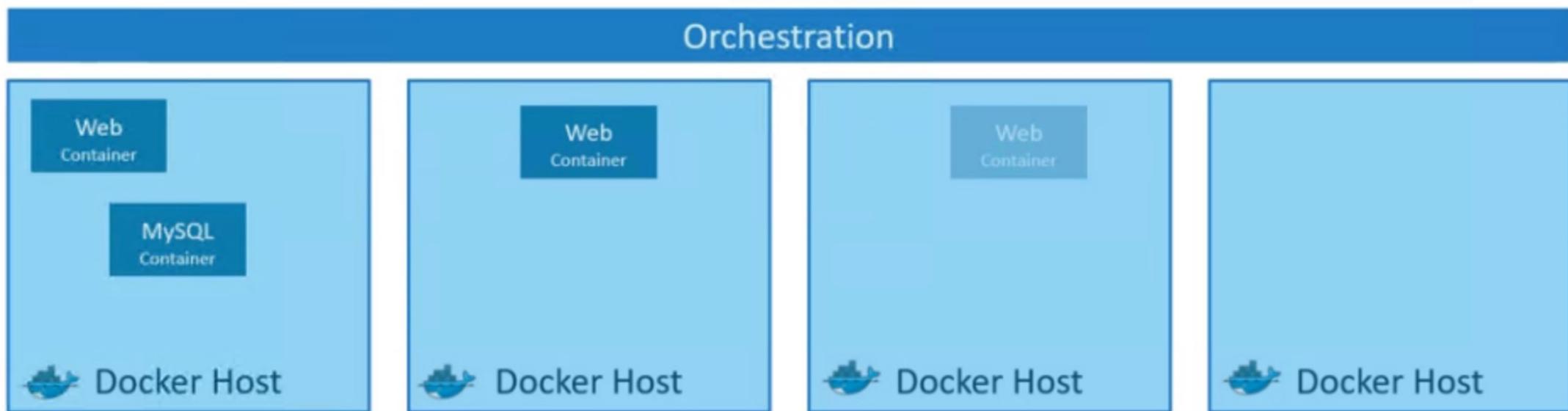


The Why's and What's of Kubernetes

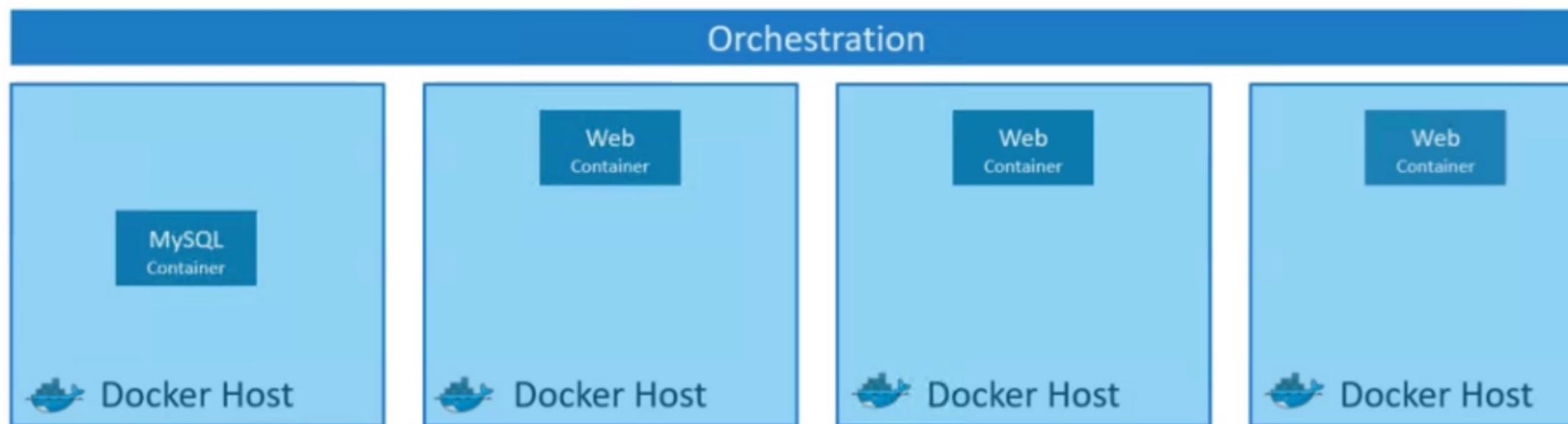
Container orchestration

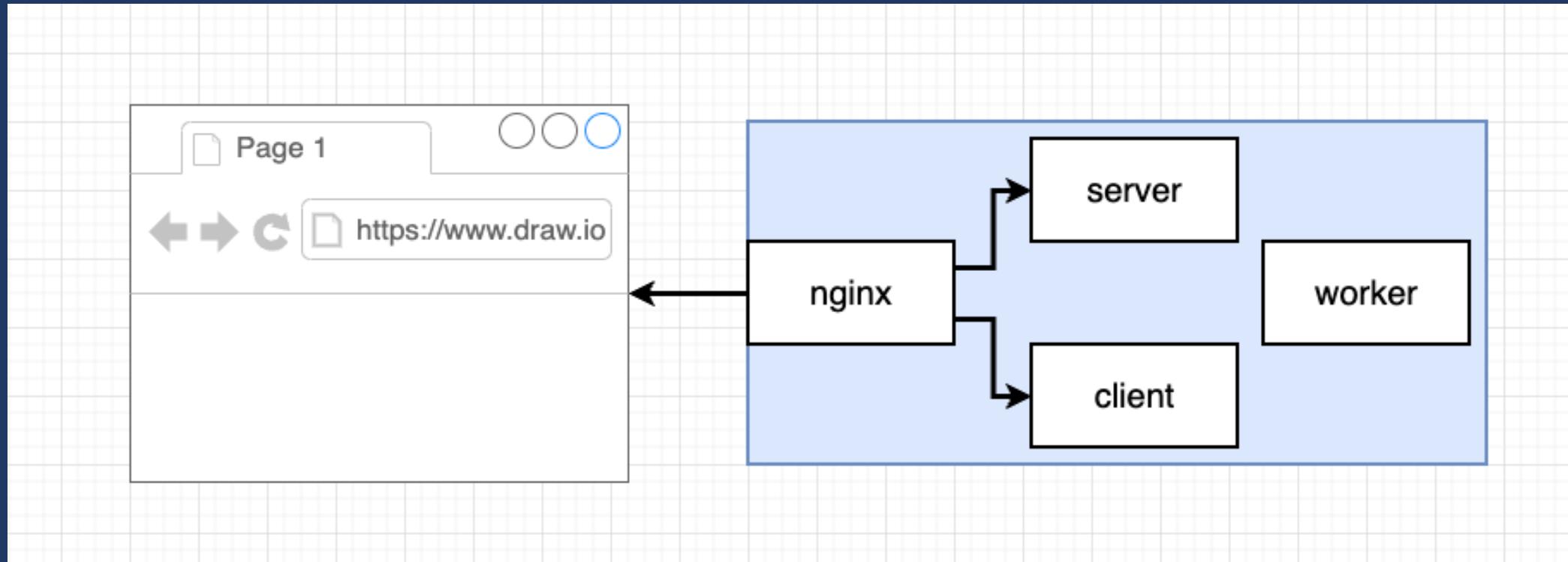


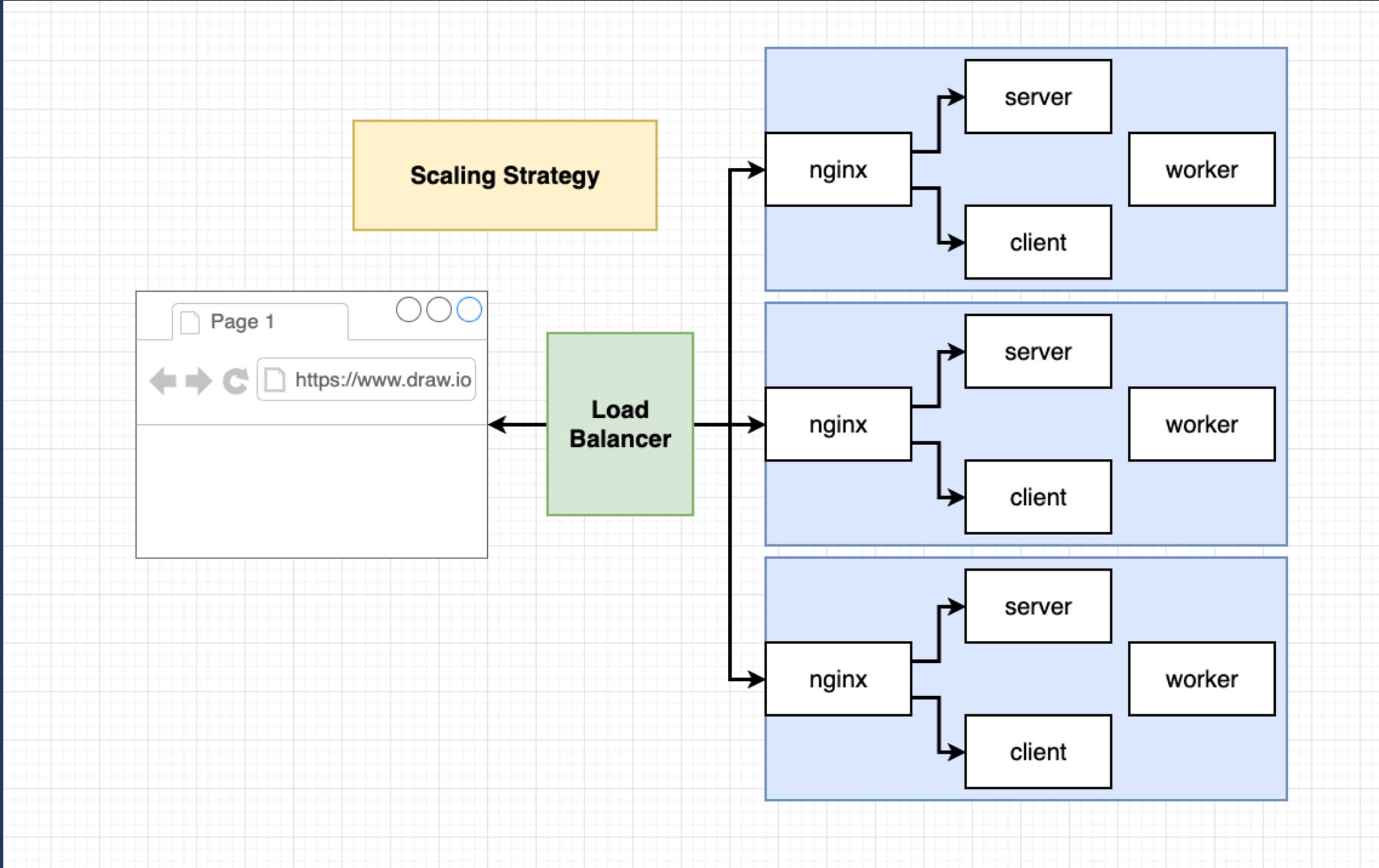
Container orchestration



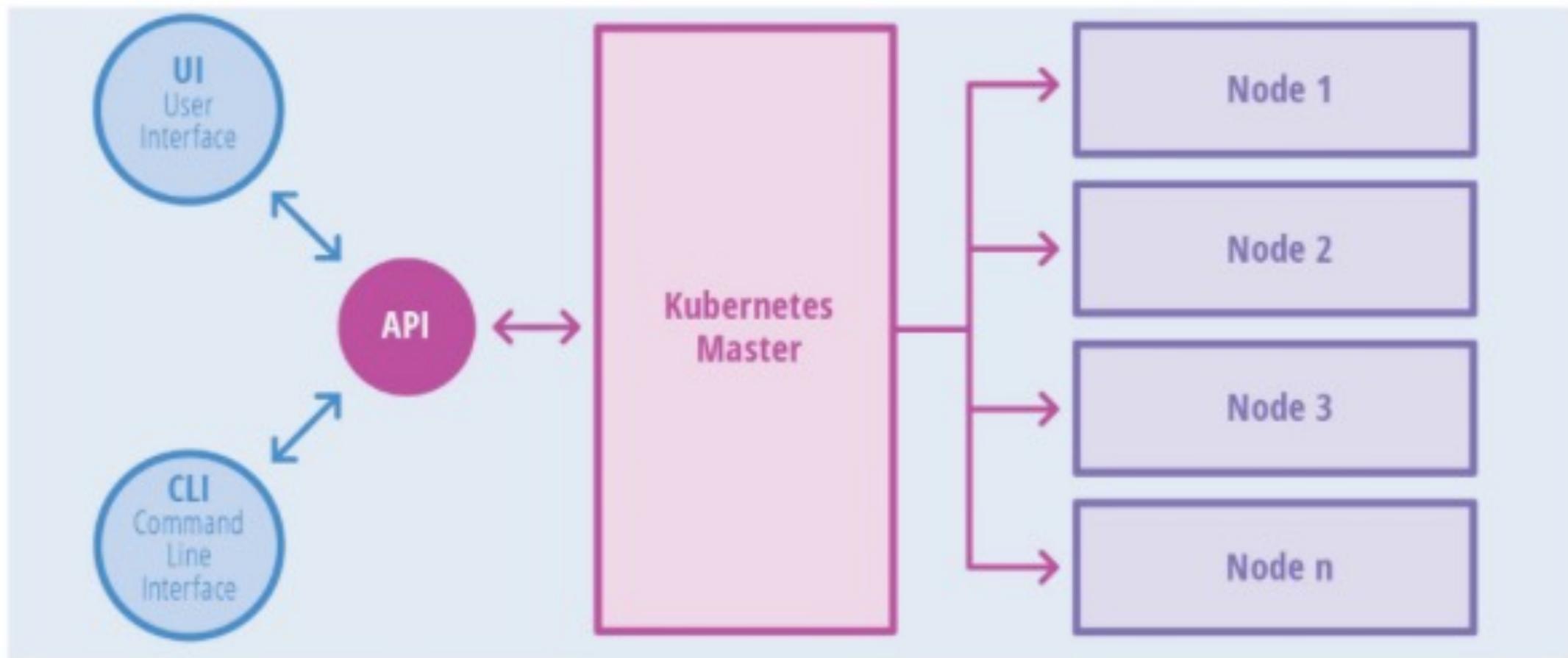
Container orchestration

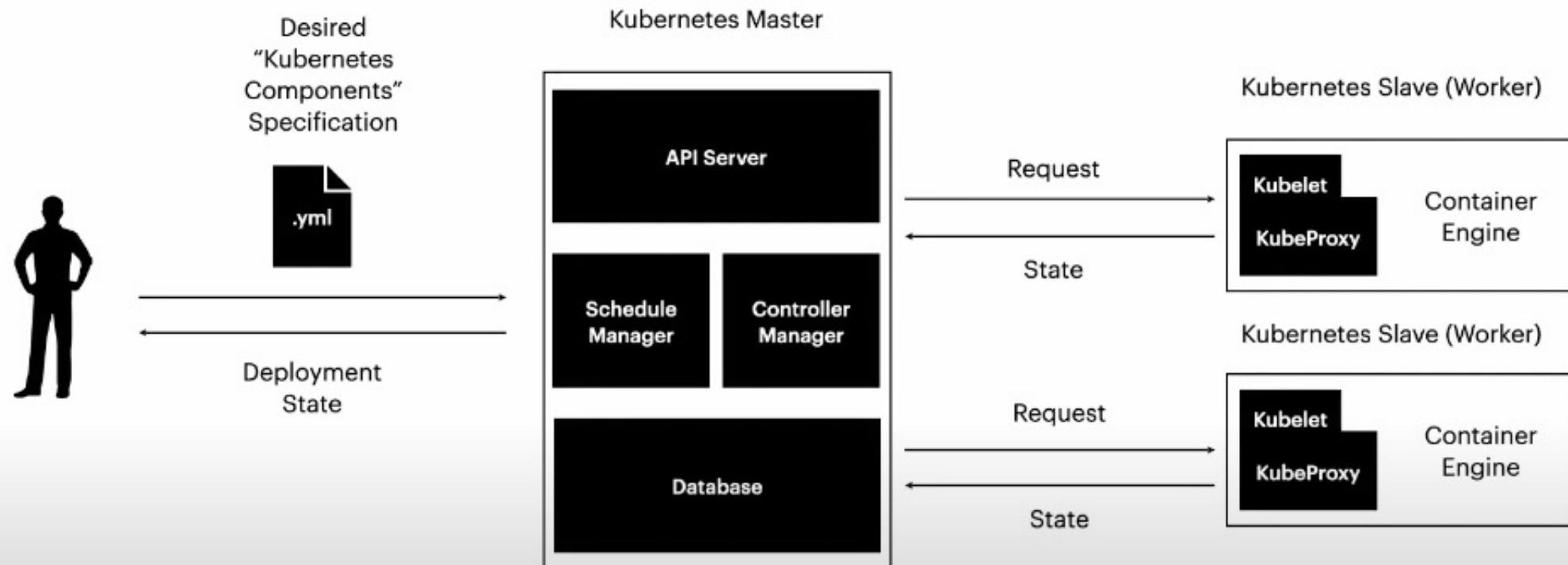


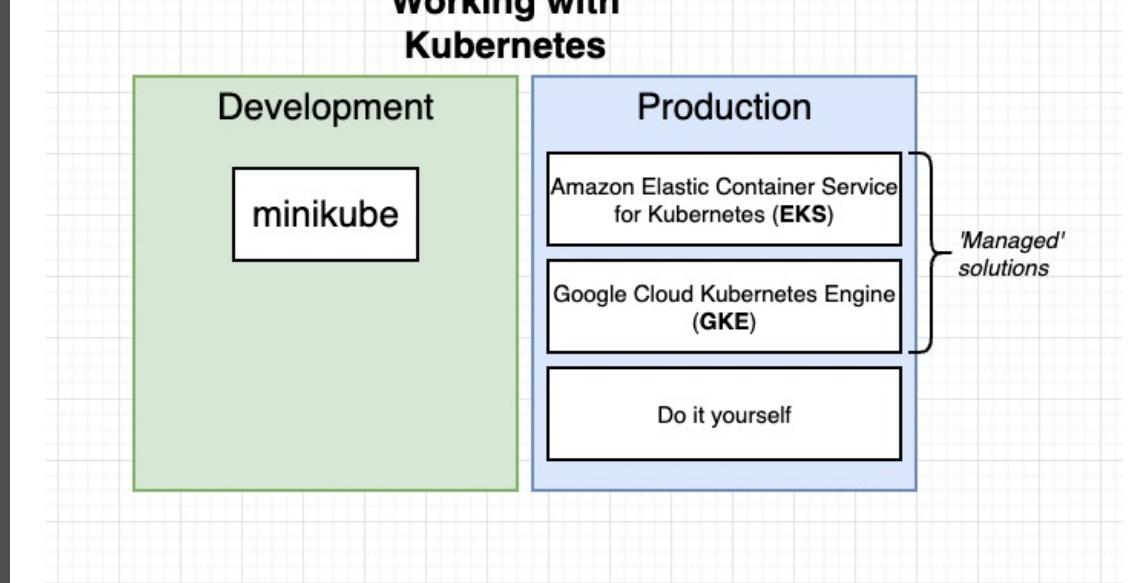
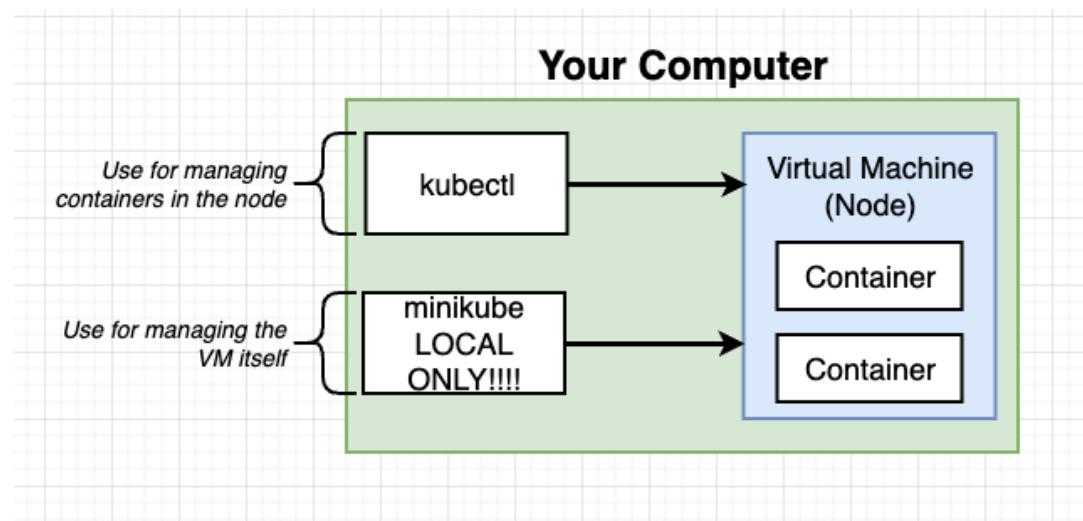




Kubernetes Architecture





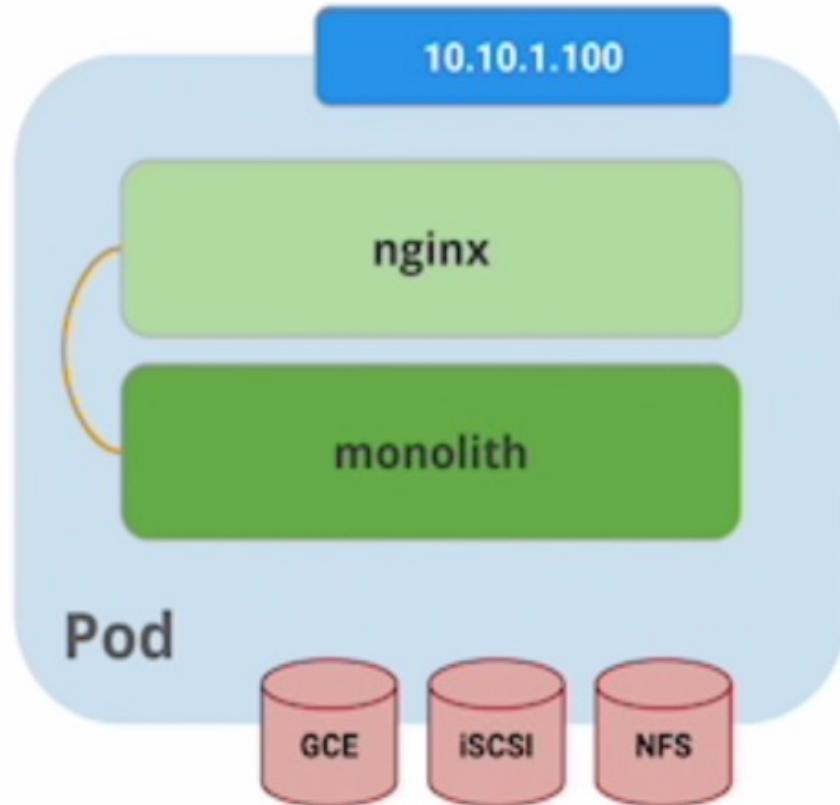
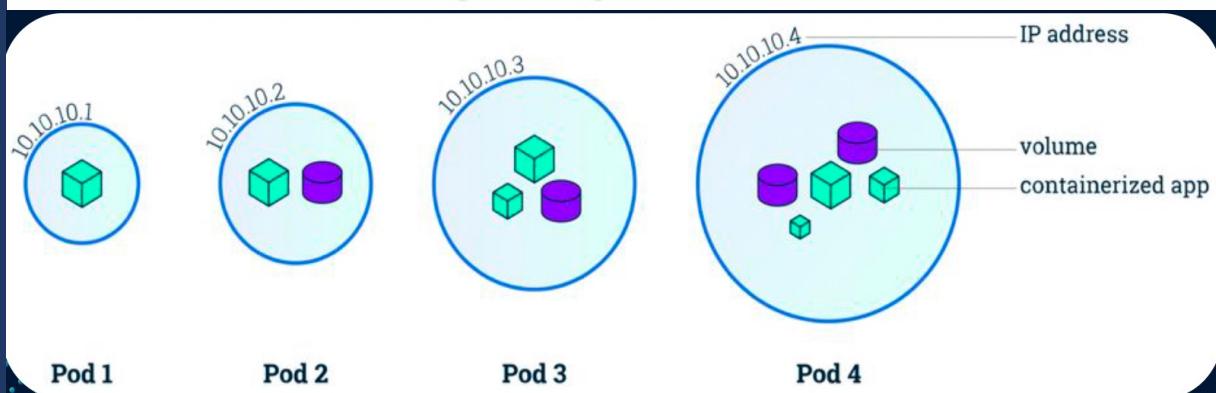


Concept of pods replica deployments service and namespace Rolling update

Pods

Logical Application

- One or more containers and volumes
- Shared namespaces
- One IP per pod



Deployment and replica

Drive current state towards desired state

app: hello
replicas: 1



Drive current state towards desired state

app: hello
replicas: 3



Deployments

Drive current state towards desired state

app: hello
replicas: 3



Deployments

Drive current state towards desired state

app: hello
replicas: 3

Dead 1 Node



Deployments

Drive current state towards desired state

app: hello
replicas: 3

Sol :Dead 1 Node



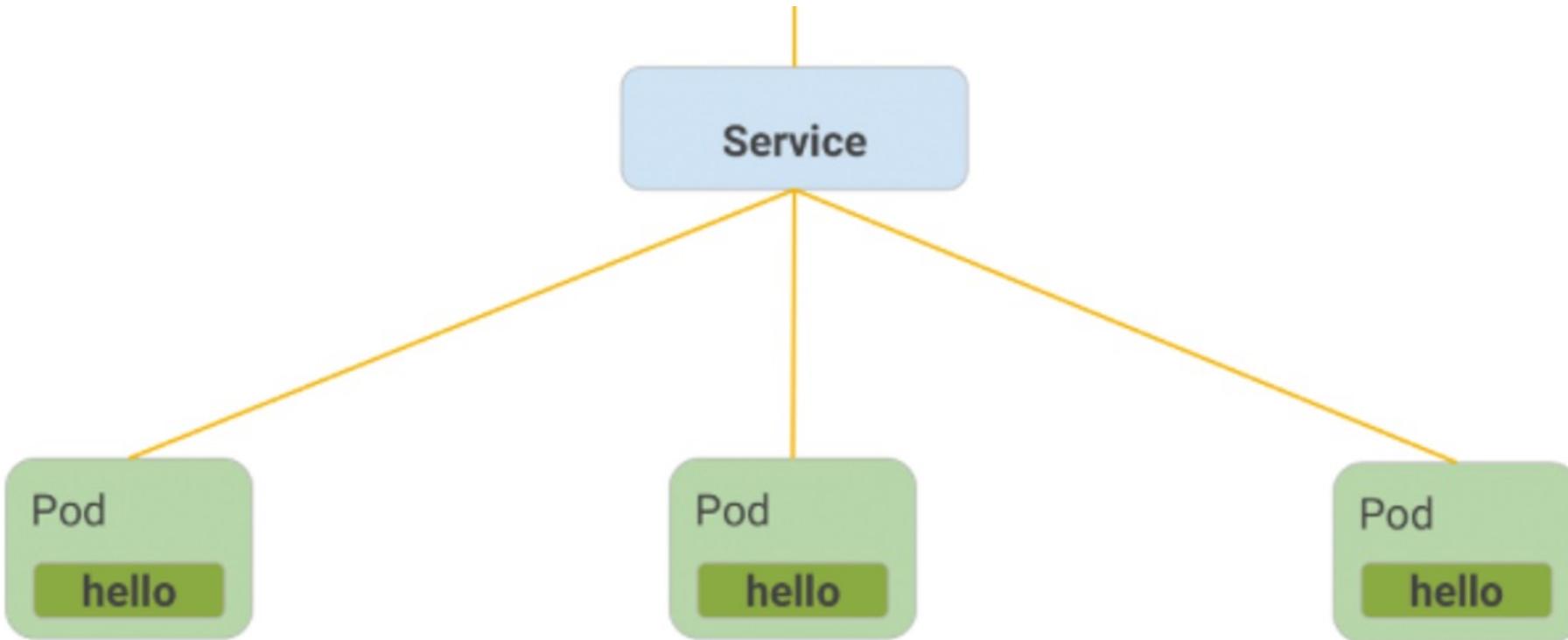
Deployments

Drive current state towards desired state

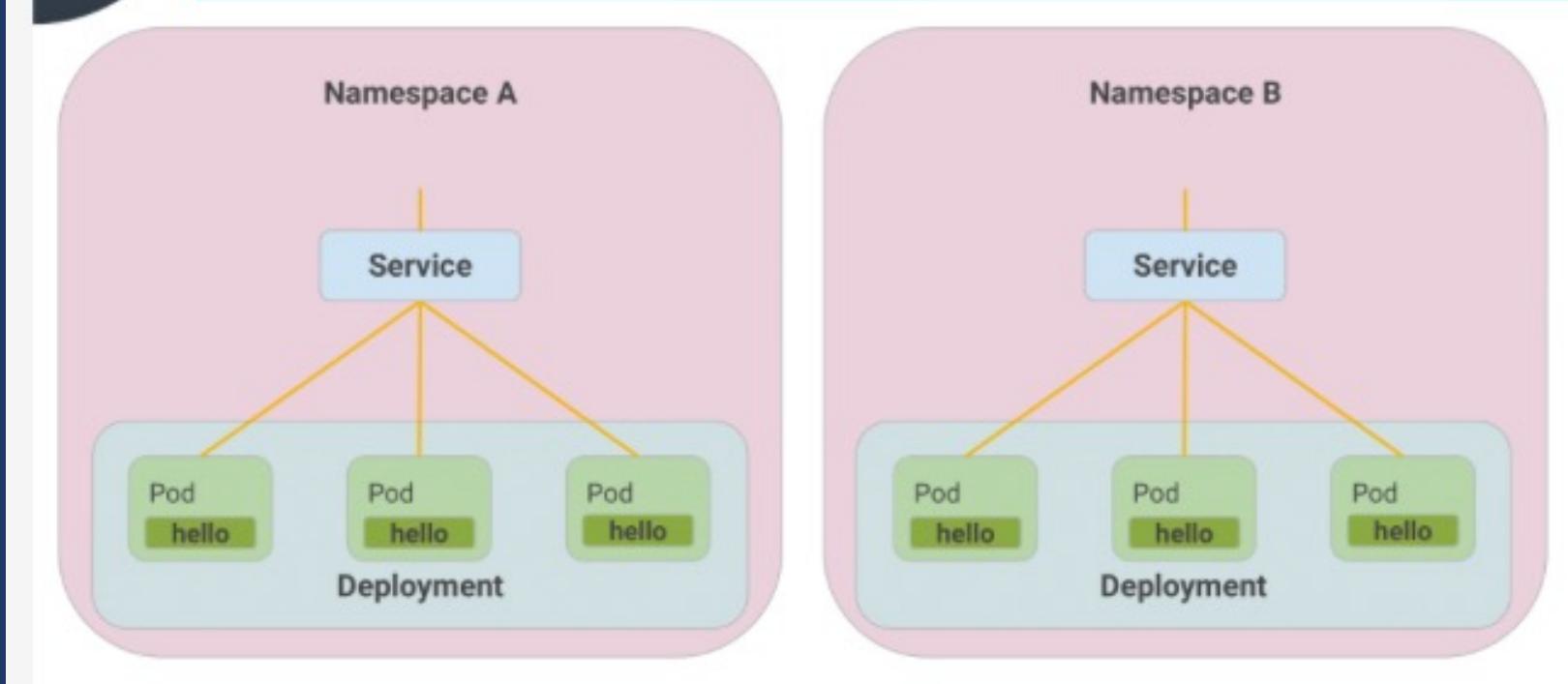
app: hello
replicas: 3



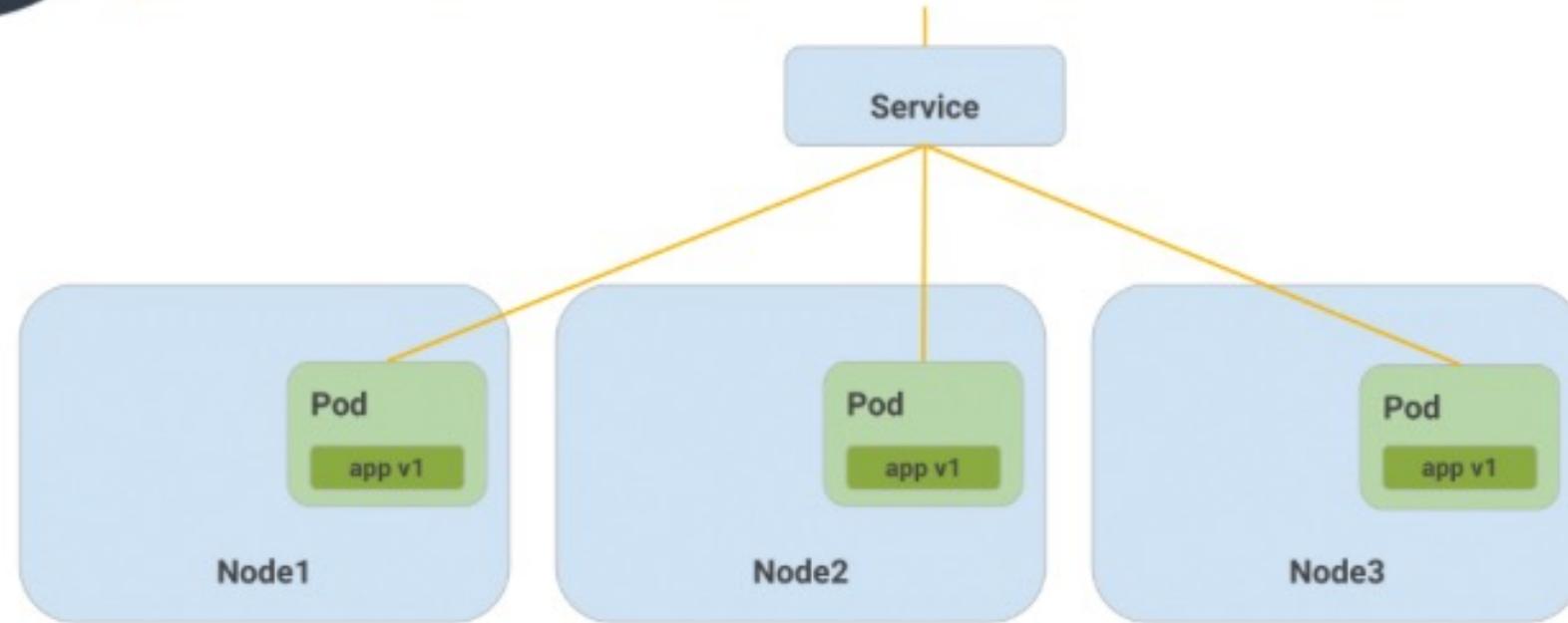
Service



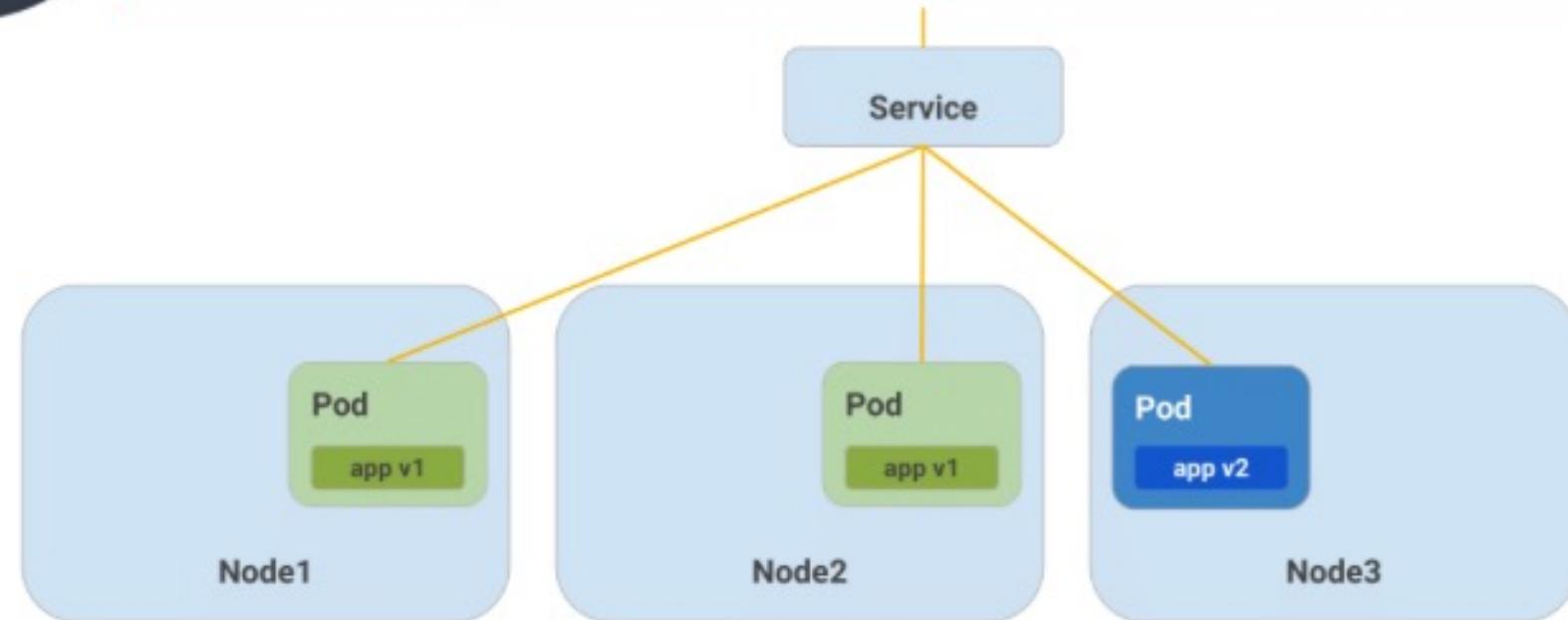
Namespace



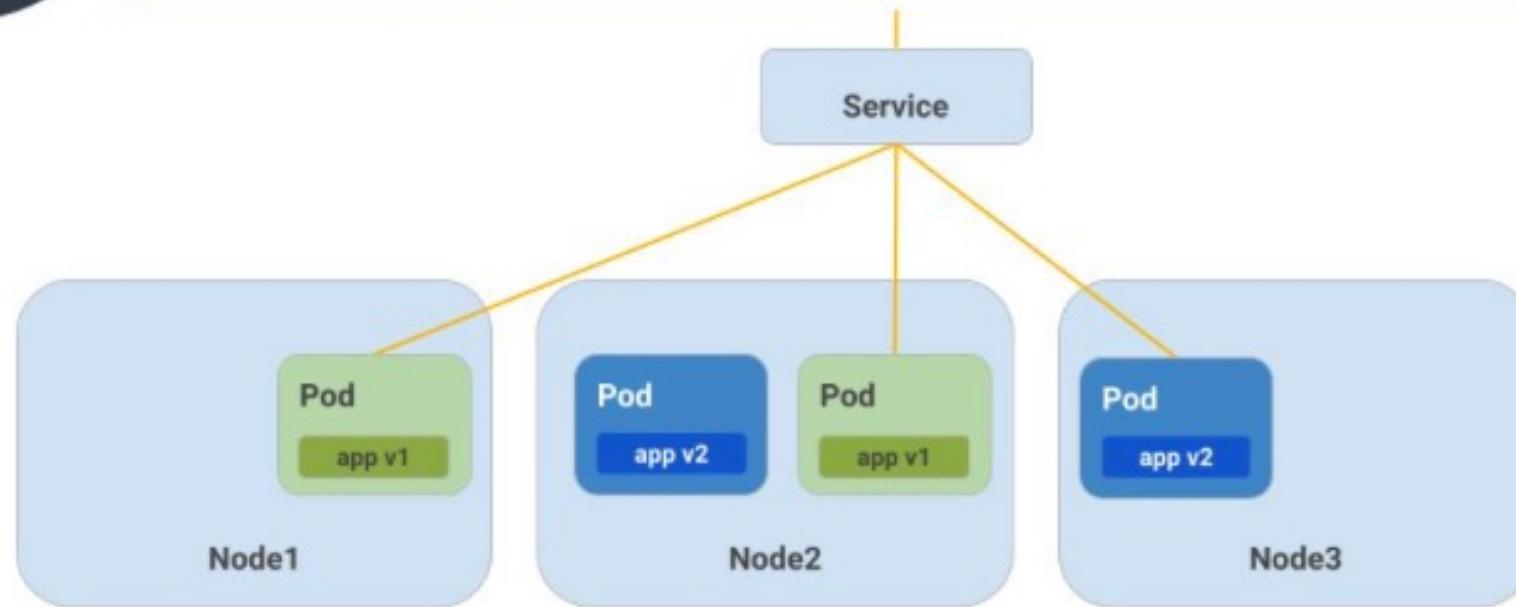
Rolling Update



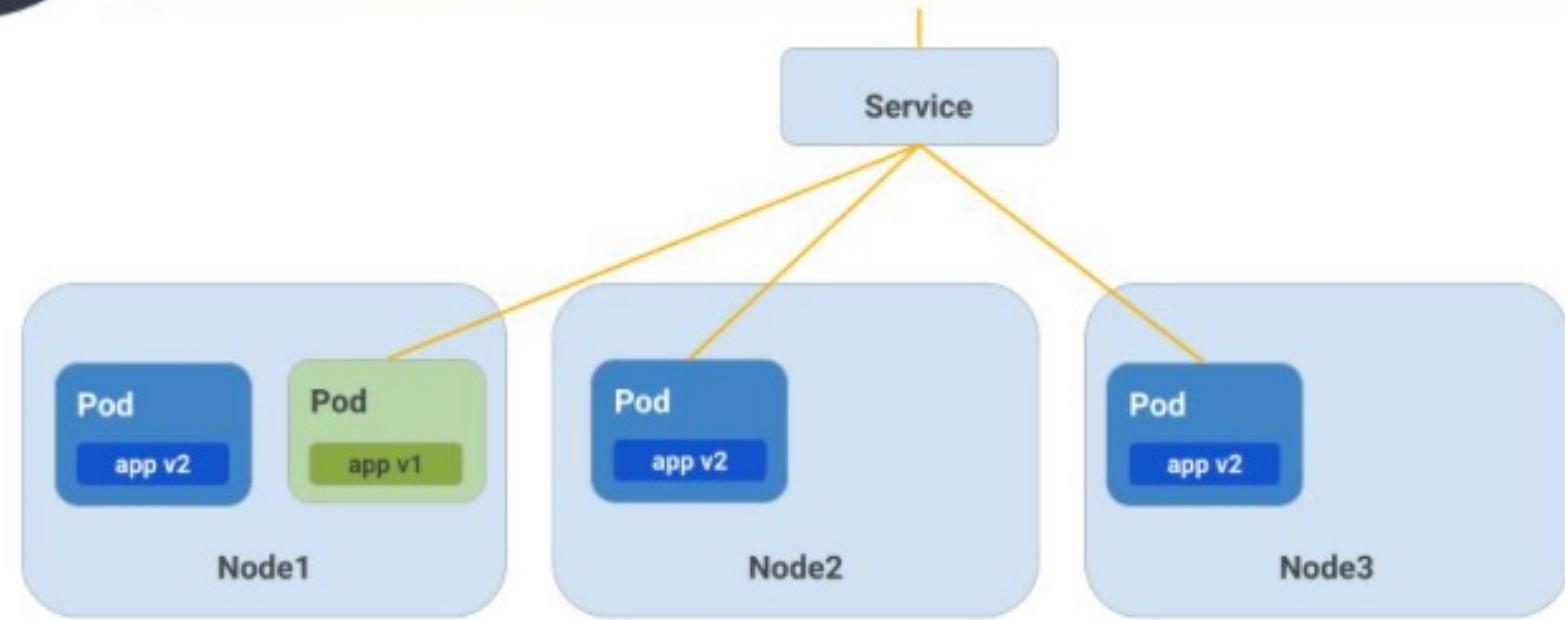
Rolling Update



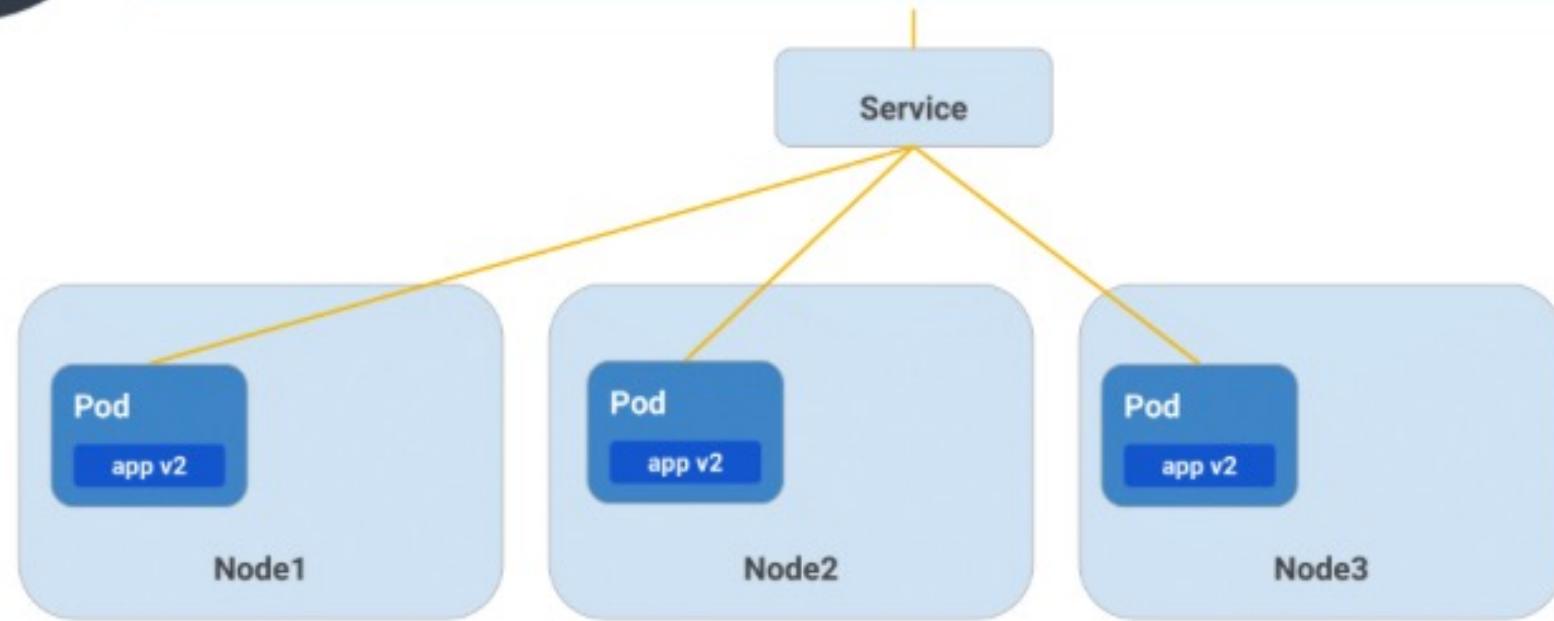
Rolling Update



Rolling Update



Rolling Update



Workflow for docker and kubernetes deployment:



Docker and Kubernetes Workflow



- Pod

<https://kubernetes.io/docs/concepts/workloads/pods/>

- Replicas set

<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>

- Deployment

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

Basic Command

Kubernetes Basic Command (get) :

kubectl get {object} –n {namespace}

kubectl get pod –n kube-system

kubectl get replicaset –n kube-system

kubectl get deployment –n kube-system

Kubernetes Basic Command (describe):

kubectl describe {object} {pod name} –n {namespace}

`kubectl describe pod {pod name} –n kube-system`

`kubectl describe replicaset {replicaset name} –n kube-system`

`kubectl describe deployment {deplotment name} –n kube-system`

Kubernetes Basic Command (log) :

kubectl logs {pod name} –n {namespace}

kubectl logs pod {pod name} –n kube-system

Kubernetes Basic Command (log) :

kubectl exec -it {pod name} –n {namespace} -- sh/bash

kubectl exec -it {pod name} –n {namespace} -- sh/bash

LAB basic-command

- <https://github.com/Tuchsanai/SOFTWARE-DEVELOPMENT-TOOLS-AND-ENVIRONMENTS/tree/main/Week11/basic-command>

Open a terminal window and run the following command to check the version of kubectl:

```
kubectl version
```

displays a table of all the nodes in the cluster

```
kubectl get node
```

display all the namespaces in the cluster, including their names, statuses

```
kubectl get namespace
```

Get information about resources in a specific namespace:

To get all pods across all namespaces in Kubernetes

```
kubectl get pods --all-namespaces
```

Get a list of all pods in the kube-system namespace:

```
kubectl get pods -n kube-system
```

Get a list of all services in the kube-system namespace:

```
kubectl get services -n kube-system
```

Get a list of all deployments in the kube-system namespace:

```
kubectl get deployments -n kube-system
```

Pod

YAML in Kubernetes

```
pod-definition.yml
```

```
apiVersion:  
kind:  
metadata:
```

```
spec:
```

YAML in Kubernetes

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myap

spec:
```

| Kind | Version |
|------------|---------|
| POD | v1 |
| Service | v1 |
| ReplicaSet | apps/v1 |
| Deployment | apps/v1 |
| | |

YAML in Kubernetes

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
```

| Kind | Version |
|------------|---------|
| POD | v1 |
| Service | v1 |
| ReplicaSet | apps/v1 |
| Deployment | apps/v1 |
| | |

YAML in Kubernetes

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

| Kind | Version |
|------------|---------|
| POD | v1 |
| Service | v1 |
| ReplicaSet | apps/v1 |
| Deployment | apps/v1 |
| | |

YAML in Kubernetes

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

| Kind | Version |
|------------|---------|
| POD | v1 |
| Service | v1 |
| ReplicaSet | apps/v1 |
| Deployment | apps/v1 |
| | |

```
kubectl create -f pod-definition.yml
```

LAB Pod

1 Create a new file called `nginx-pod.yaml` with the following content:

```
nano nginx-pod.yaml
```



2 Add the following content

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-lab
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

3. Save and close the file.

```
Ctrl+X, then Y, then Enter.
```

4. Create the Nginx pod

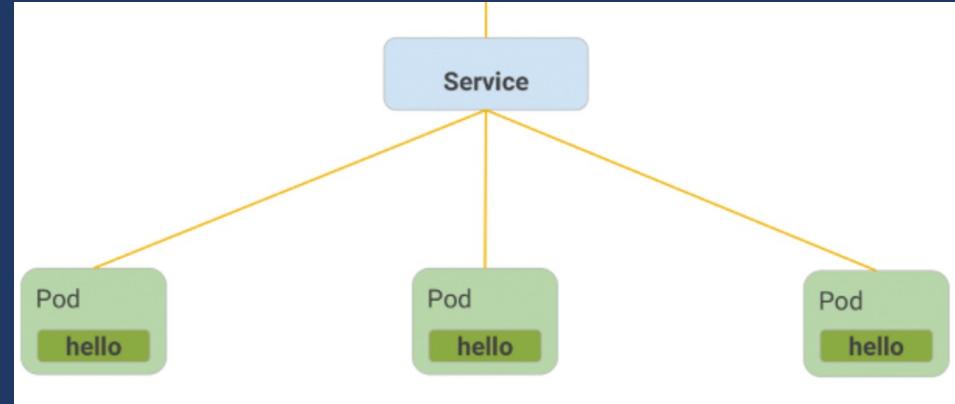
```
kubectl apply -f nginx-pod.yaml
```

5. Check the status of the pod

```
kubectl get pods
```

<https://github.com/it63070105/SOFTWARE-DEVELOPMENT-TOOLS-AND-ENVIRONMENTS/tree/main/Week11/Pod>

Service

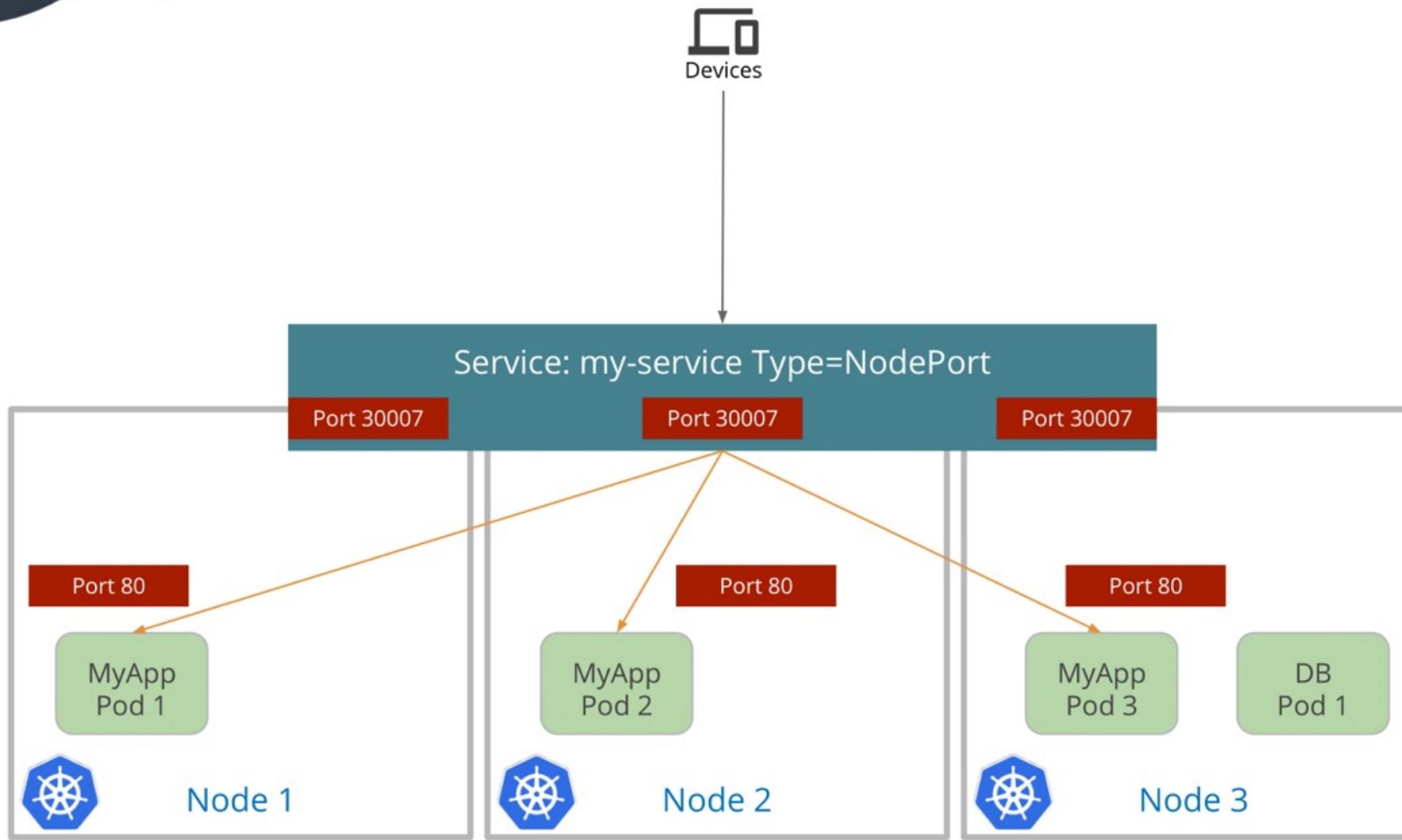


In Kubernetes, there are generally three types of services

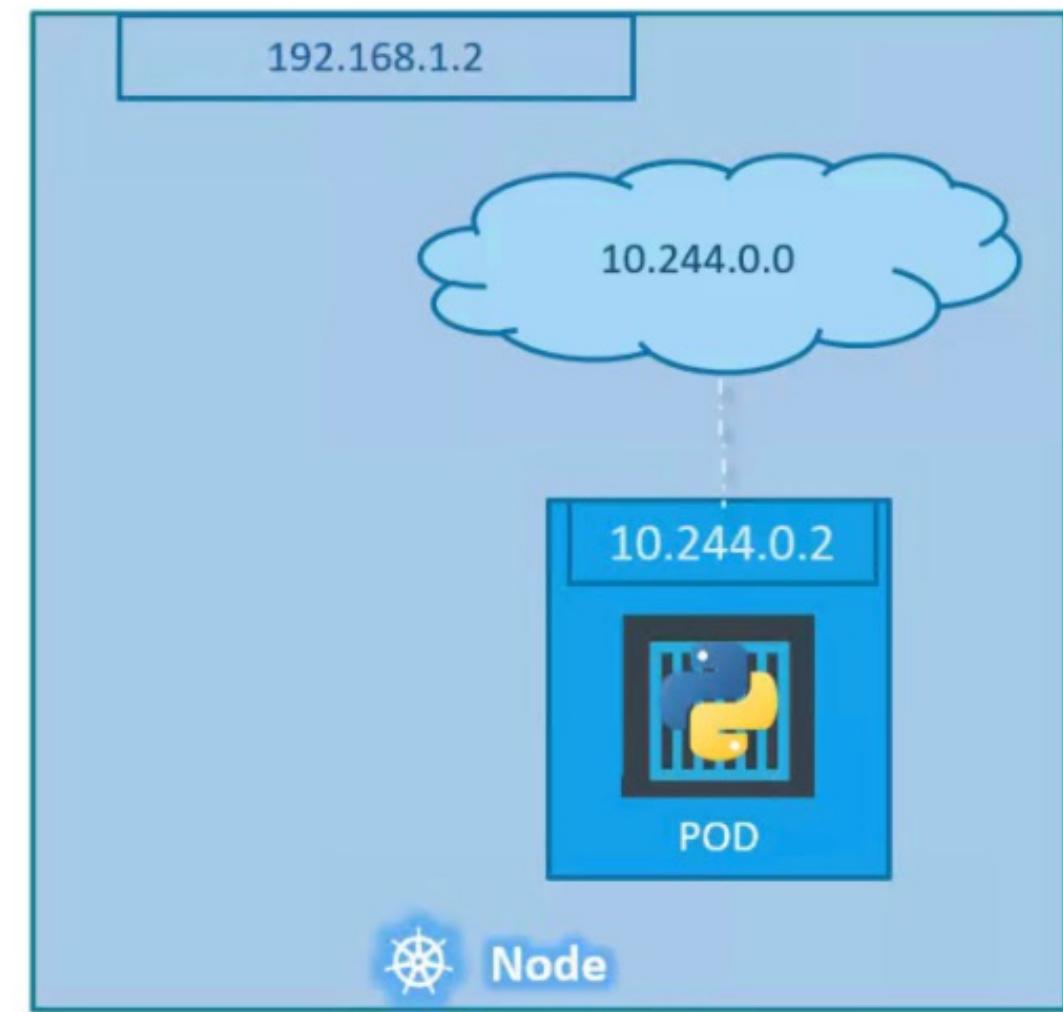
1. [NodePort](#): This service type allows the pods to be accessed from outside the cluster via a static port on each worker node. This is often used for development and testing purposes, but not recommended for production use as it can pose security risks.
2. [ClusterIP](#): This is the default service type in Kubernetes. It provides a stable IP address and DNS name to access a set of pods within the same Kubernetes cluster. This type of service is used for internal communication between components within the cluster.
3. [LoadBalancer](#): This service type is used to expose pods to the internet or other external networks by creating a load balancer in a public cloud provider, such as AWS or GCP. It provides a single entry point for traffic to reach the pods and distributes the traffic across multiple nodes.

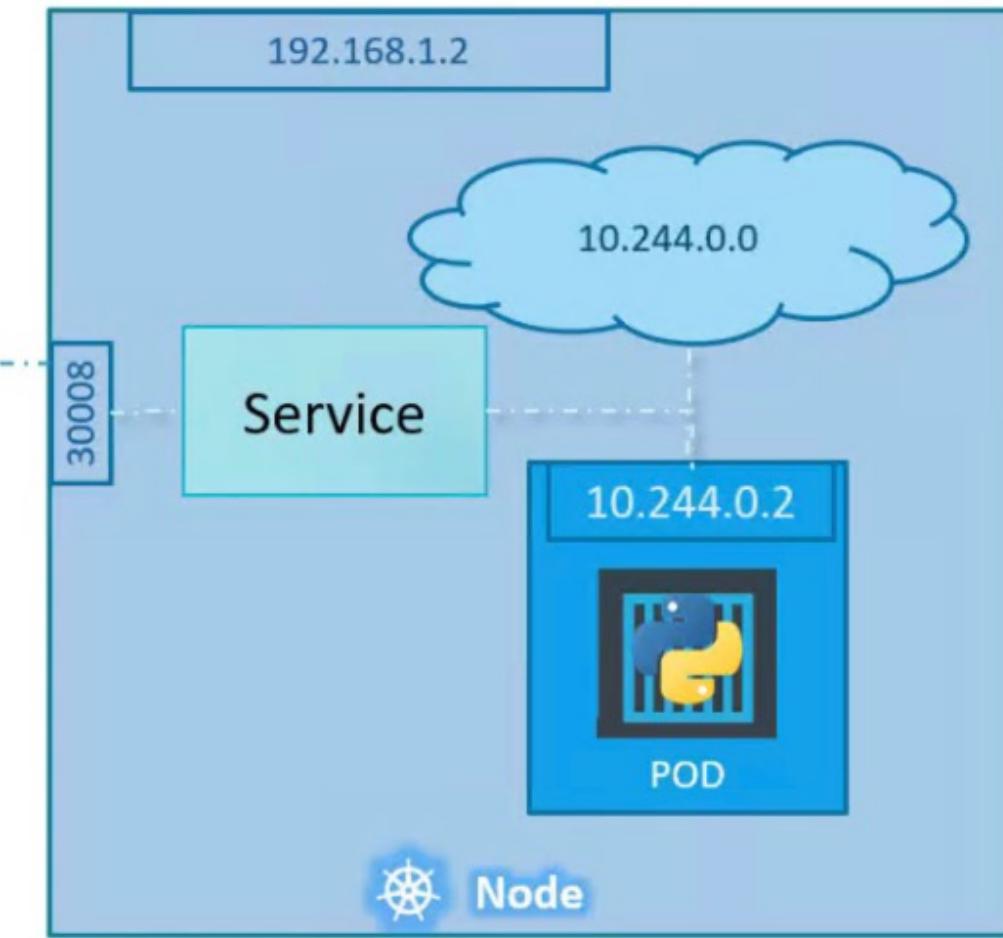
Node Pod

Service Type: Node Port

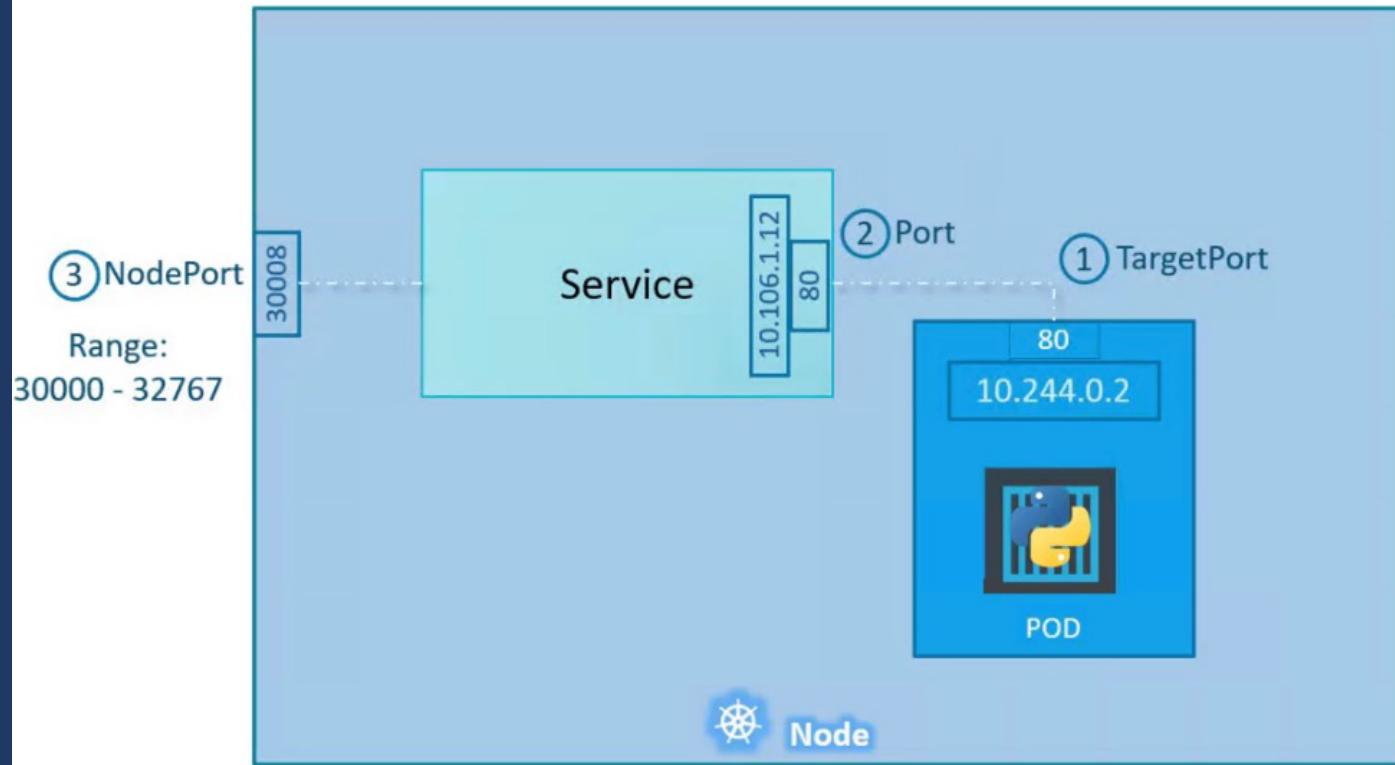


```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
    # By default and for
    # convenience, the
    `targetPort` is set to the
    same value as the `port`
    # field.
    - port: 80
      targetPort: 80
      # Optional field
      # By default and for
      # convenience, the Kubernetes
      control plane will allocate
      a port from a range
      (default: 30000-32767)
      nodePort: 30007
```





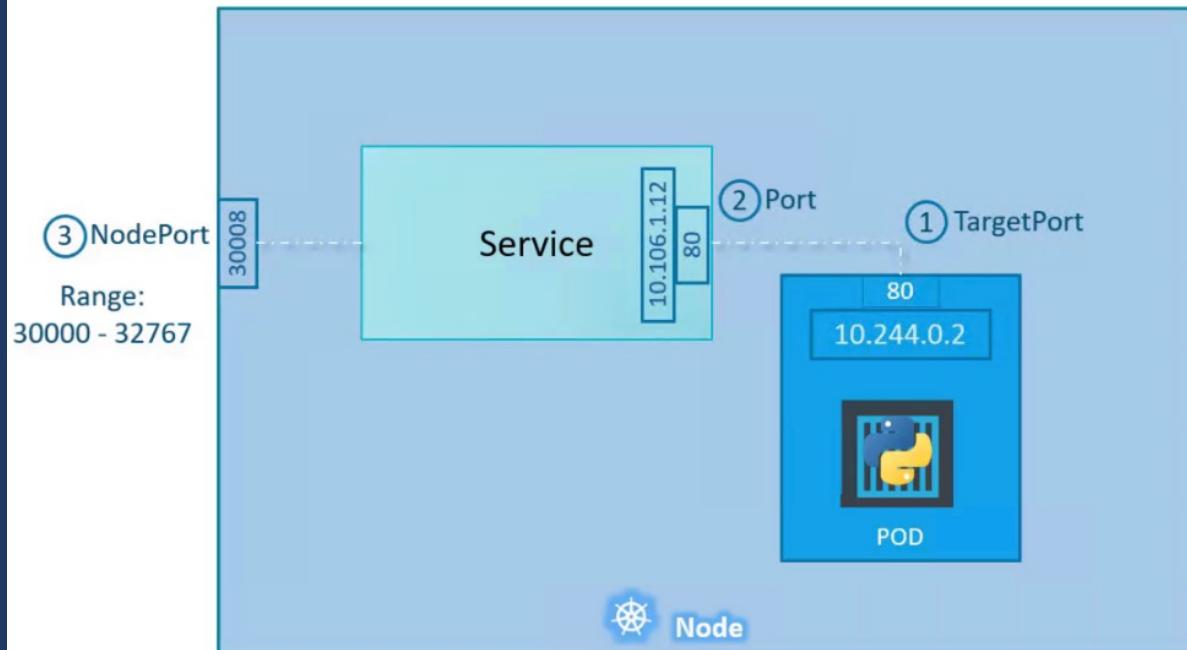
Service - NodePort



```
service-definition.yml
```

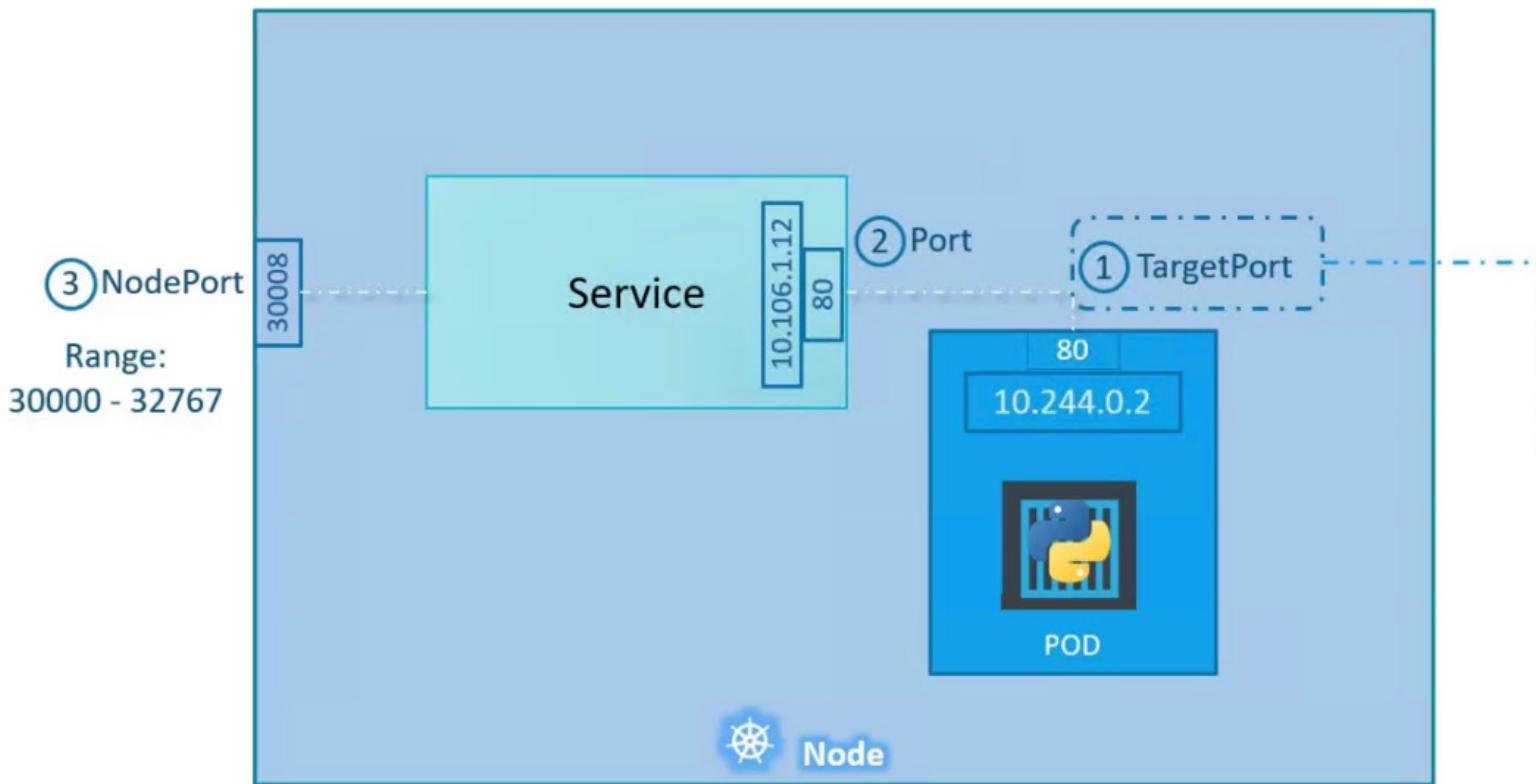
```
apiVersion: v1
kind: Service
metadata:
spec:
```

Service - NodePort



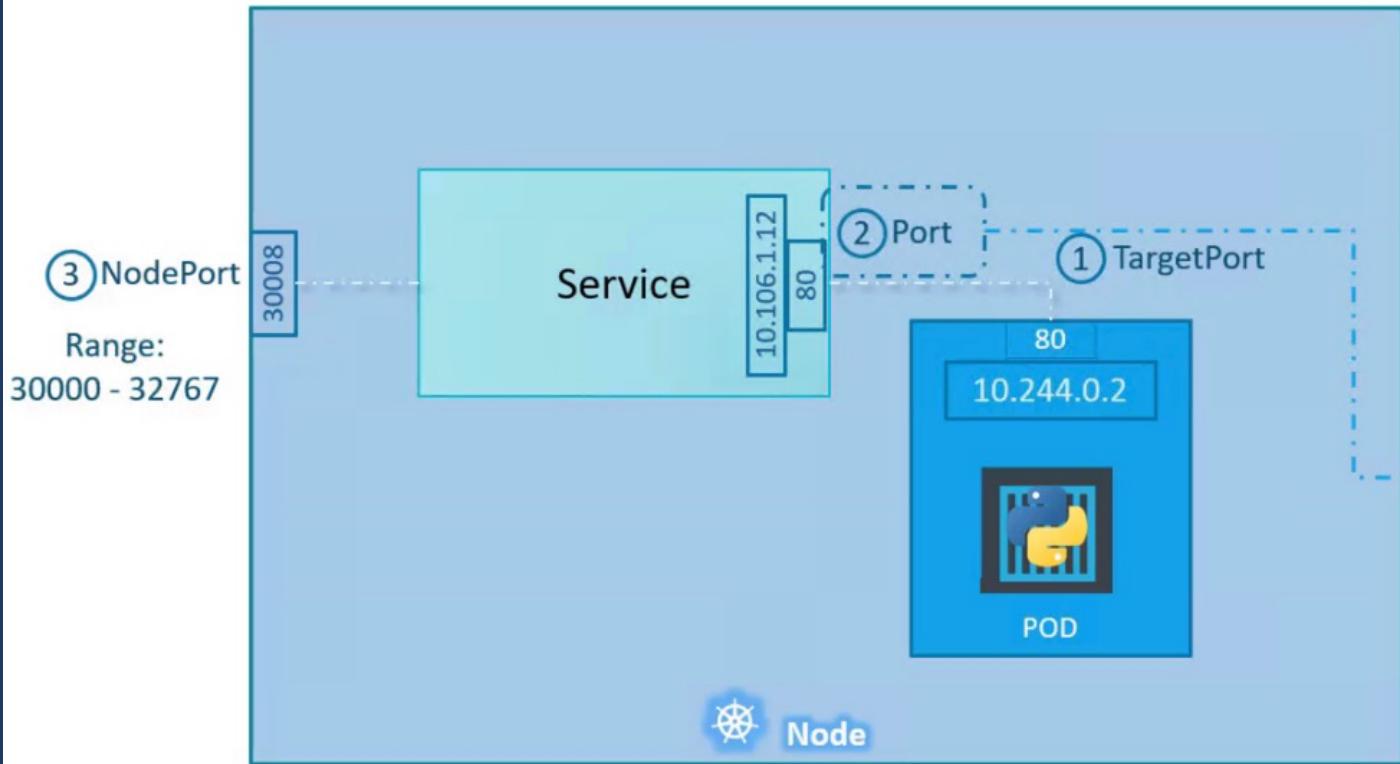
```
service-definition.yml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      *port: 80
      nodePort: 30008
```

Service - NodePort



```
service-definition.yml  
apiVersion: v1  
kind: Service  
metadata:  
  name: myapp-service  
spec:  
  type: NodePort  
  ports:  
    - targetPort: 80  
      port: 80  
      nodePort: 30008
```

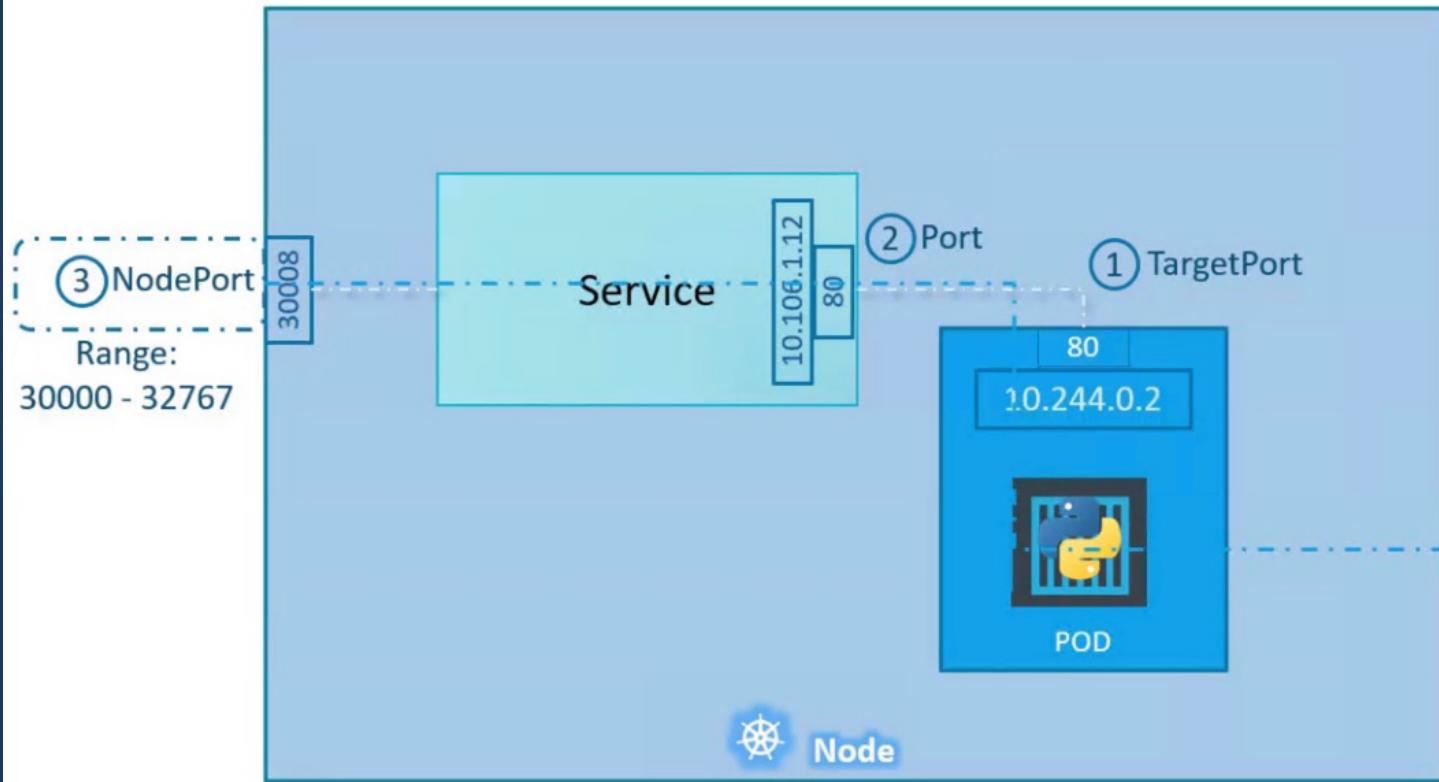
Service - NodePort



service-definition.yml

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
```

Service - NodePort



```
service-definition.yml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
```

Service - NodePort

```
service-definition.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service

spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
```

Service - NodePort

```
service-definition.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
```



```
pod-definition.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Service - NodePort

service-definition.yml

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Service - NodePort

```
service-definition.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service

spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

```
> kubectl create -f service-definition.yml
service "myapp-service" created
```

Service - NodePort

```
service-definition.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

```
> kubectl create -f service-definition.yml
service "myapp-service" created
```

```
> kubectl get services
```

| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|---------------|-----------|----------------|-------------|--------------|-----|
| kubernetes | ClusterIP | 10.96.0.1 | <none> | 443/TCP | 16d |
| myapp-service | NodePort | 10.106.127.123 | <none> | 80:30008/TCP | 5m |

Service - NodePort

```
service-definition.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008
  selector:
    app: myapp
    type: front-end
```

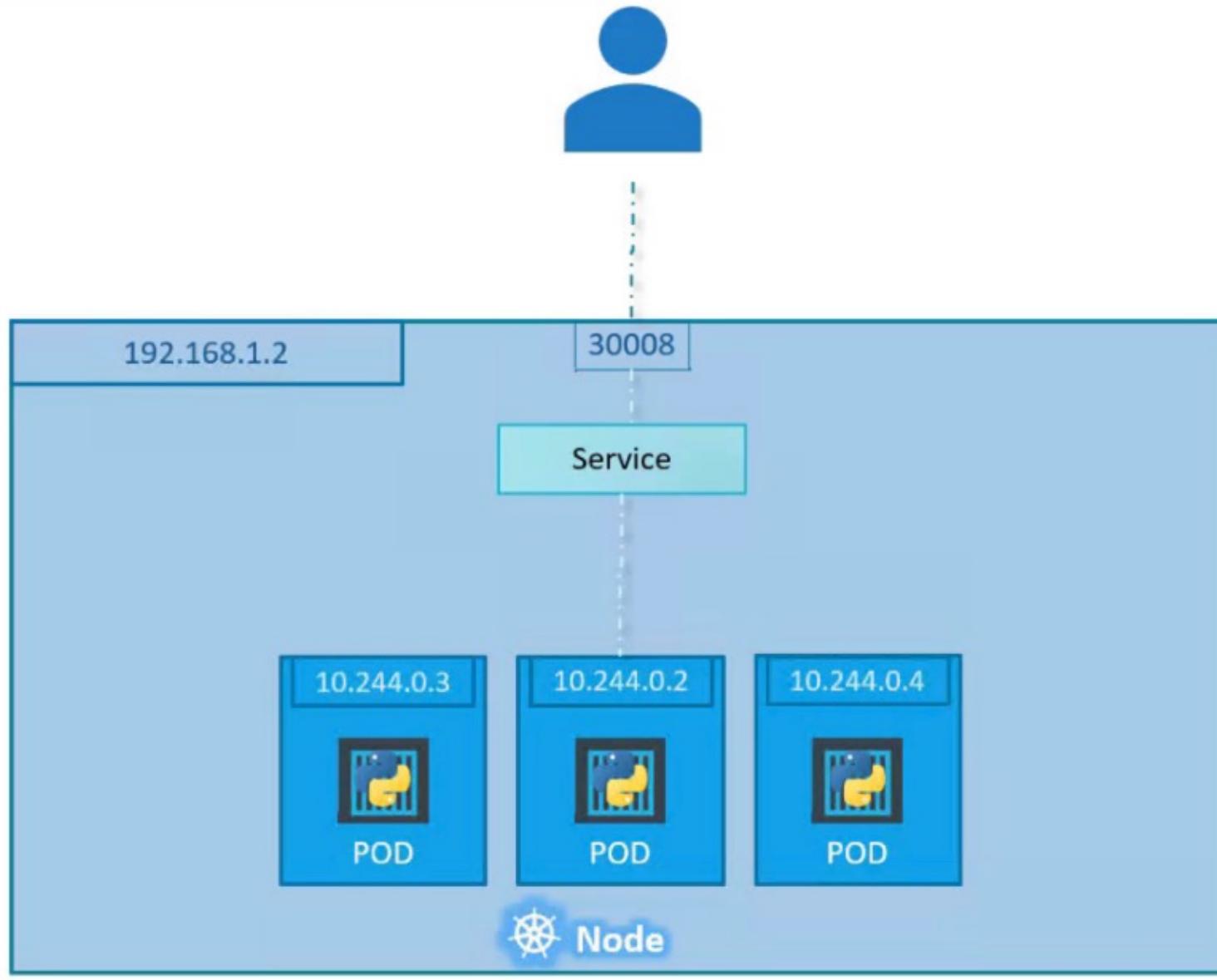
```
> kubectl create -f service-definition.yml
service "myapp-service" created
```

```
> kubectl get services
```

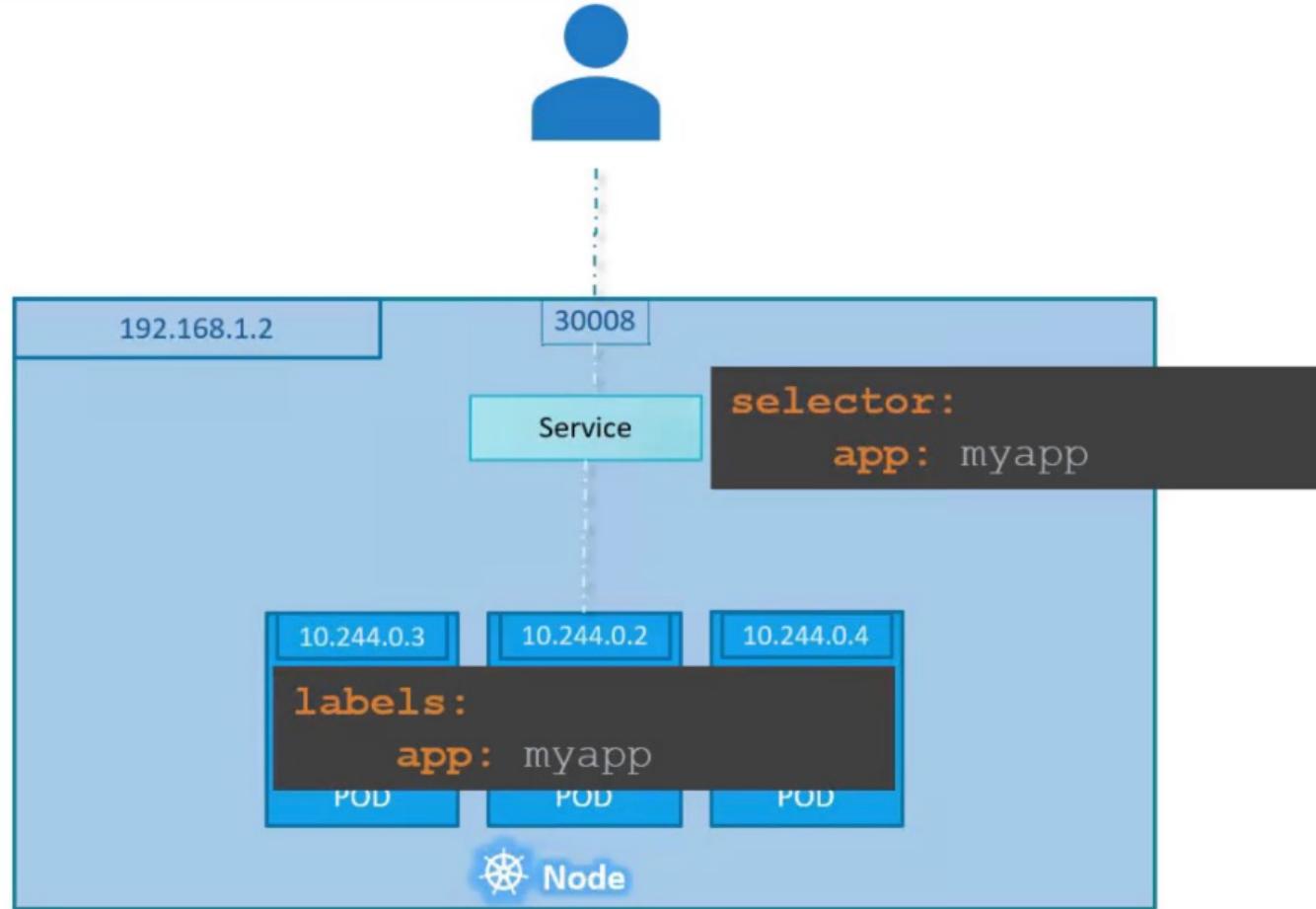
| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|---------------|-----------|----------------|-------------|--------------|-----|
| kubernetes | ClusterIP | 10.96.0.1 | <none> | 443/TCP | 16d |
| myapp-service | NodePort | 10.106.127.123 | <none> | 80:30008/TCP | 5m |

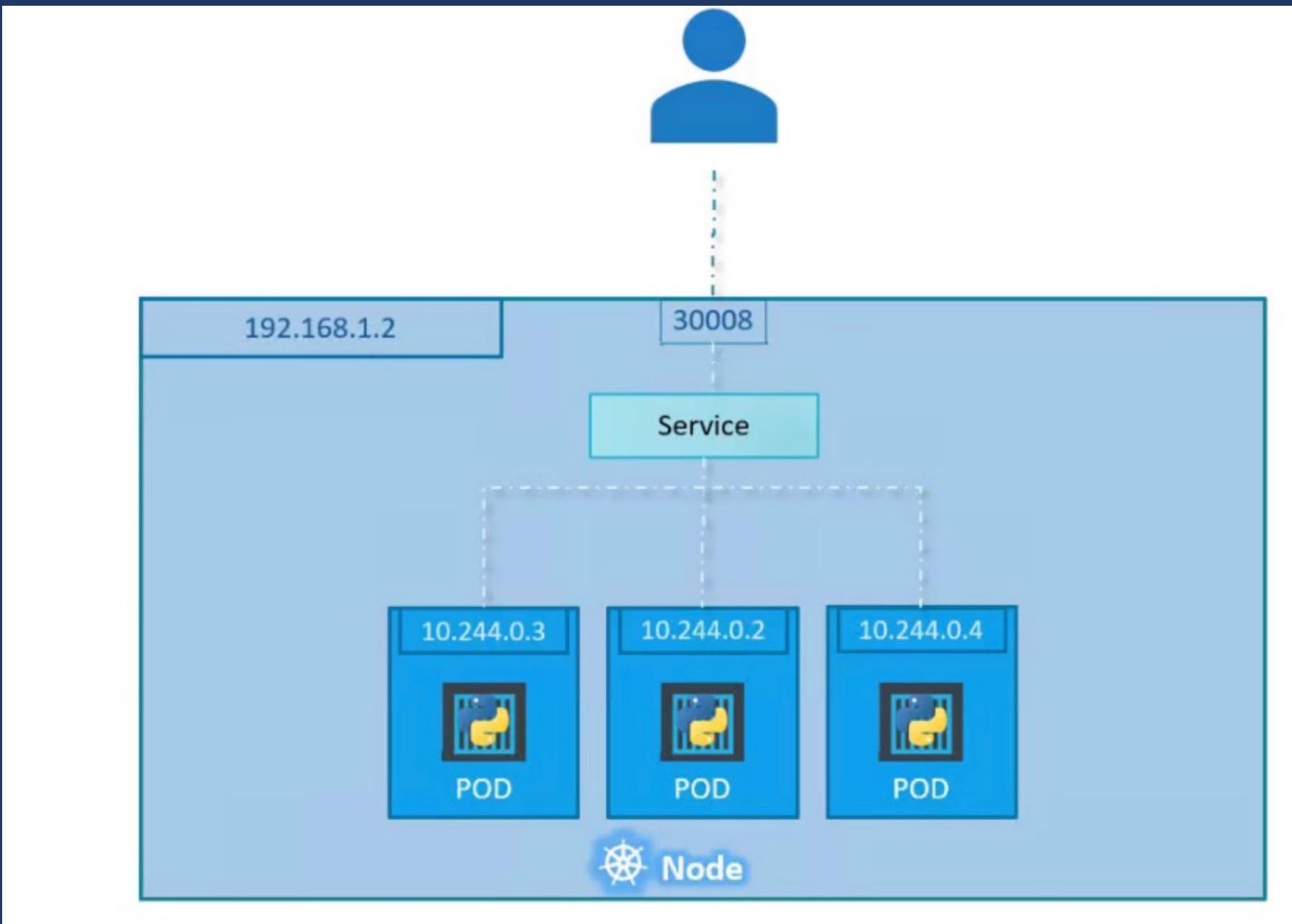
```
> curl http://192.168.1.2:30008
```

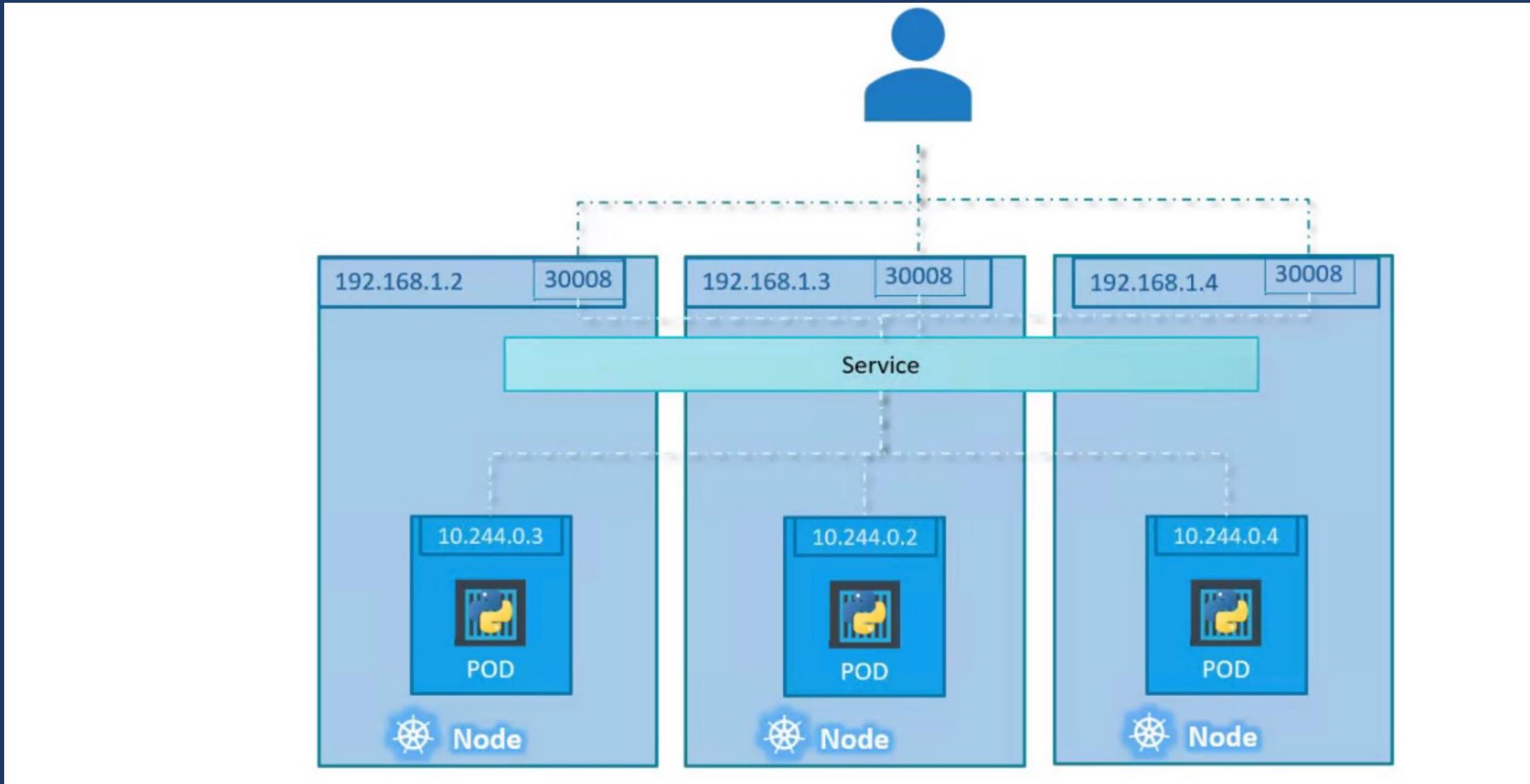
```
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
```

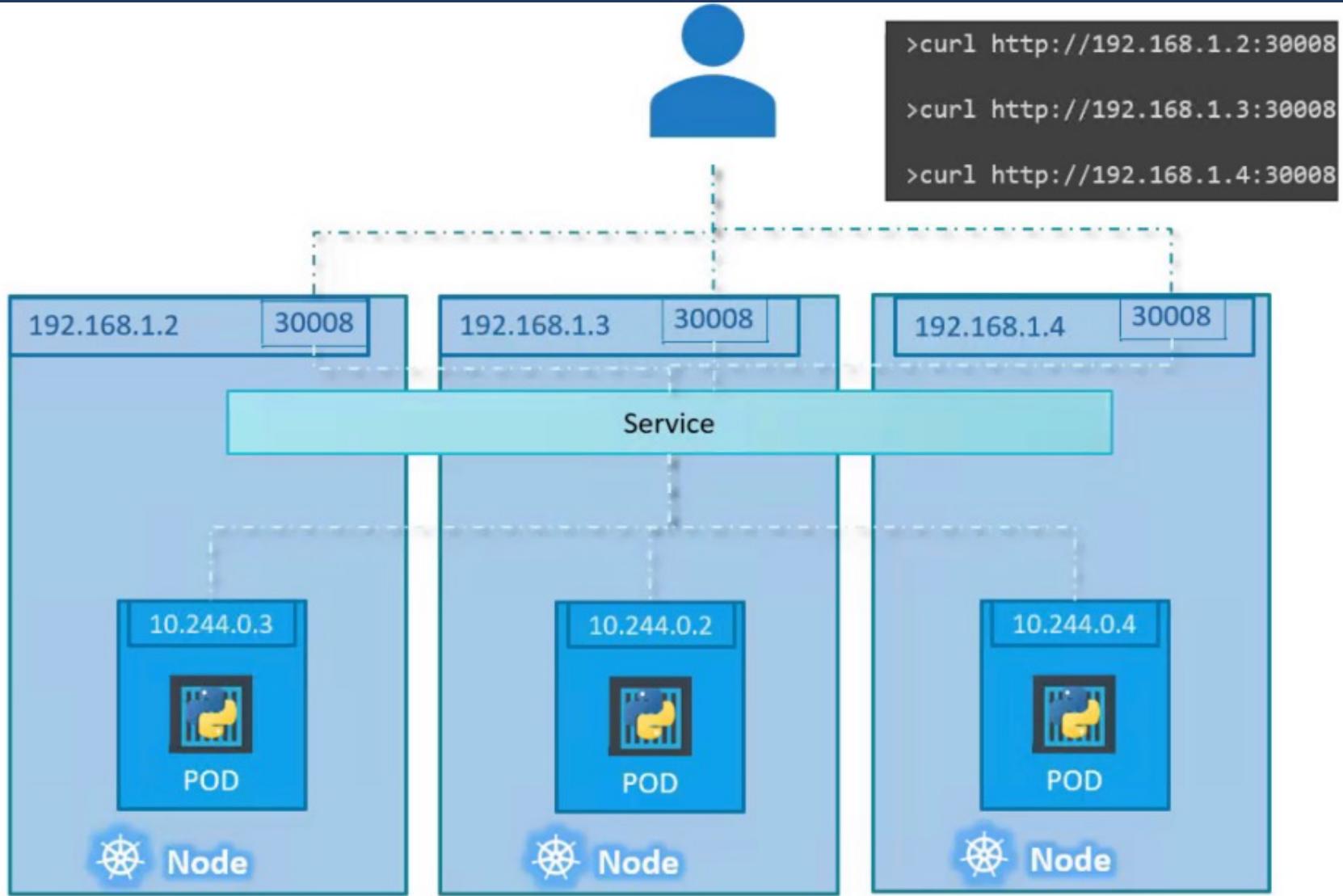


Service - NodePort





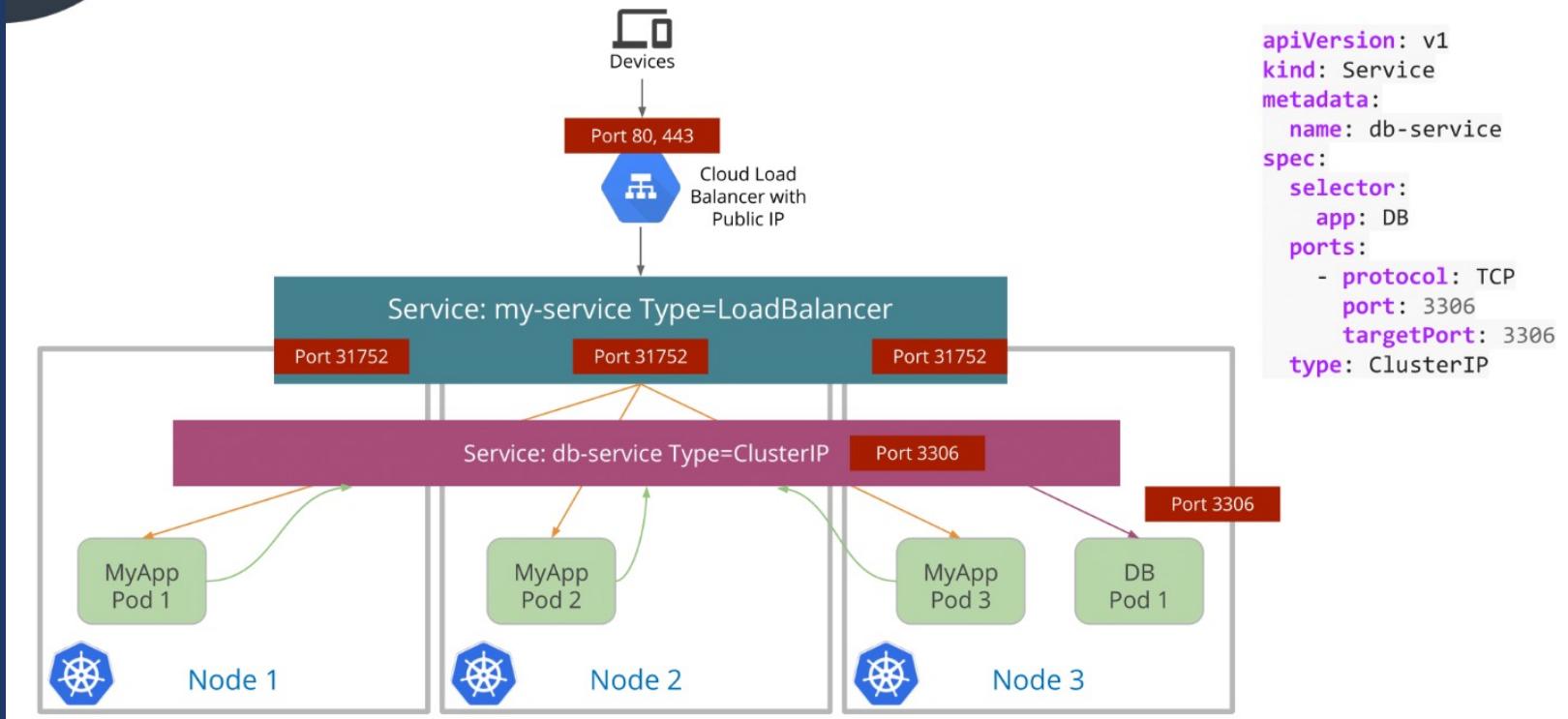


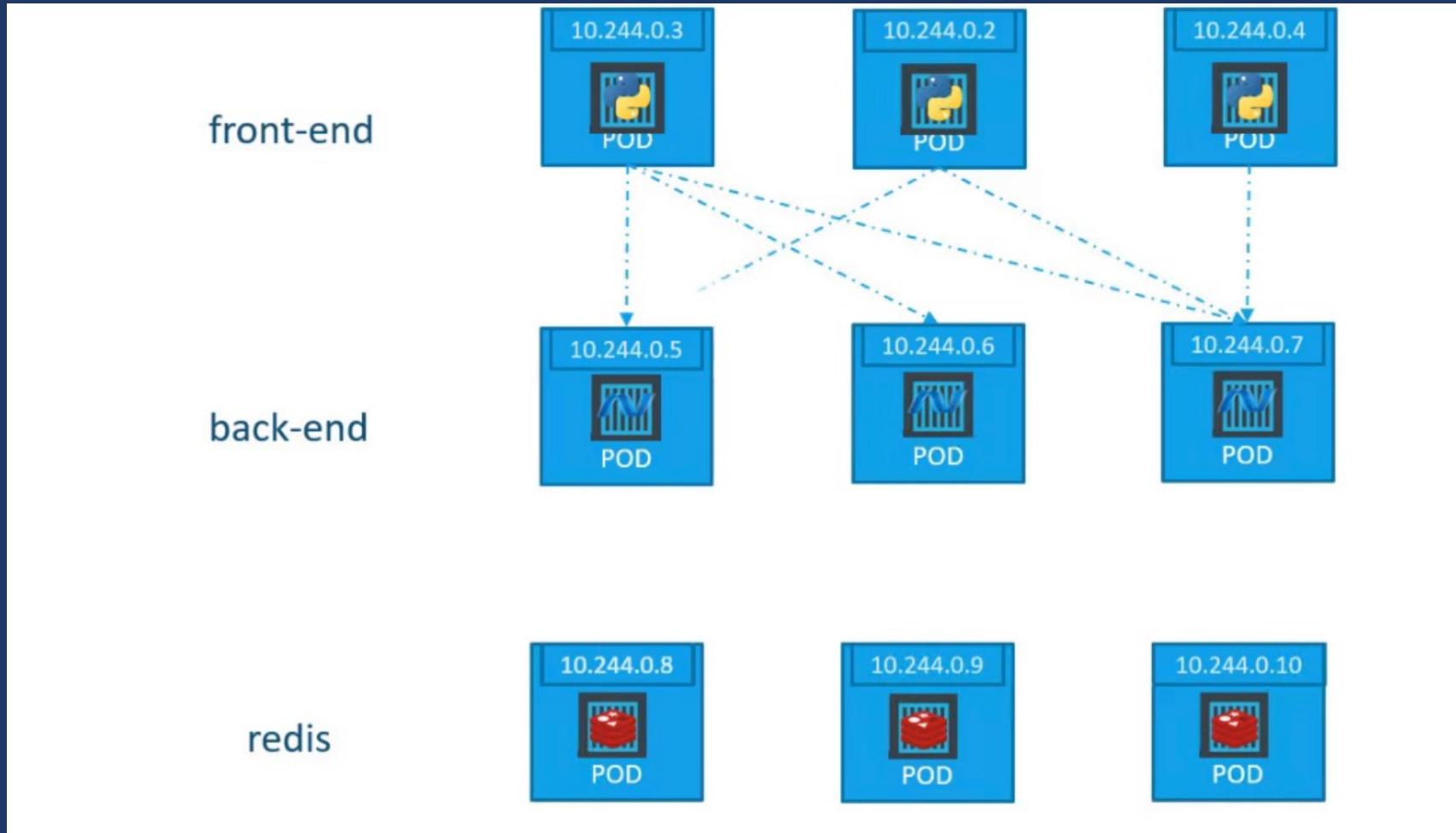


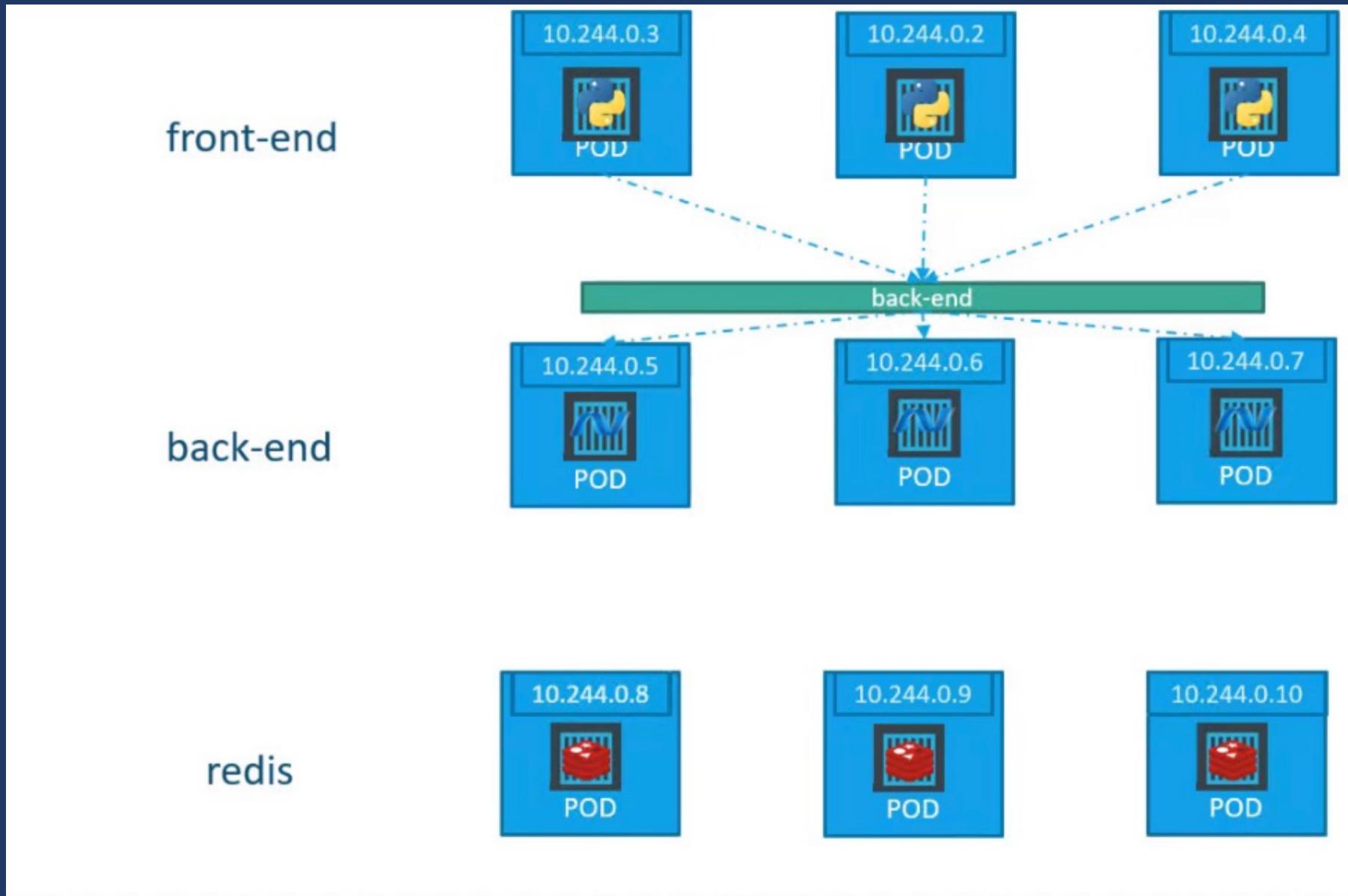
```
>curl http://192.168.1.2:30008  
>curl http://192.168.1.3:30008  
>curl http://192.168.1.4:30008
```

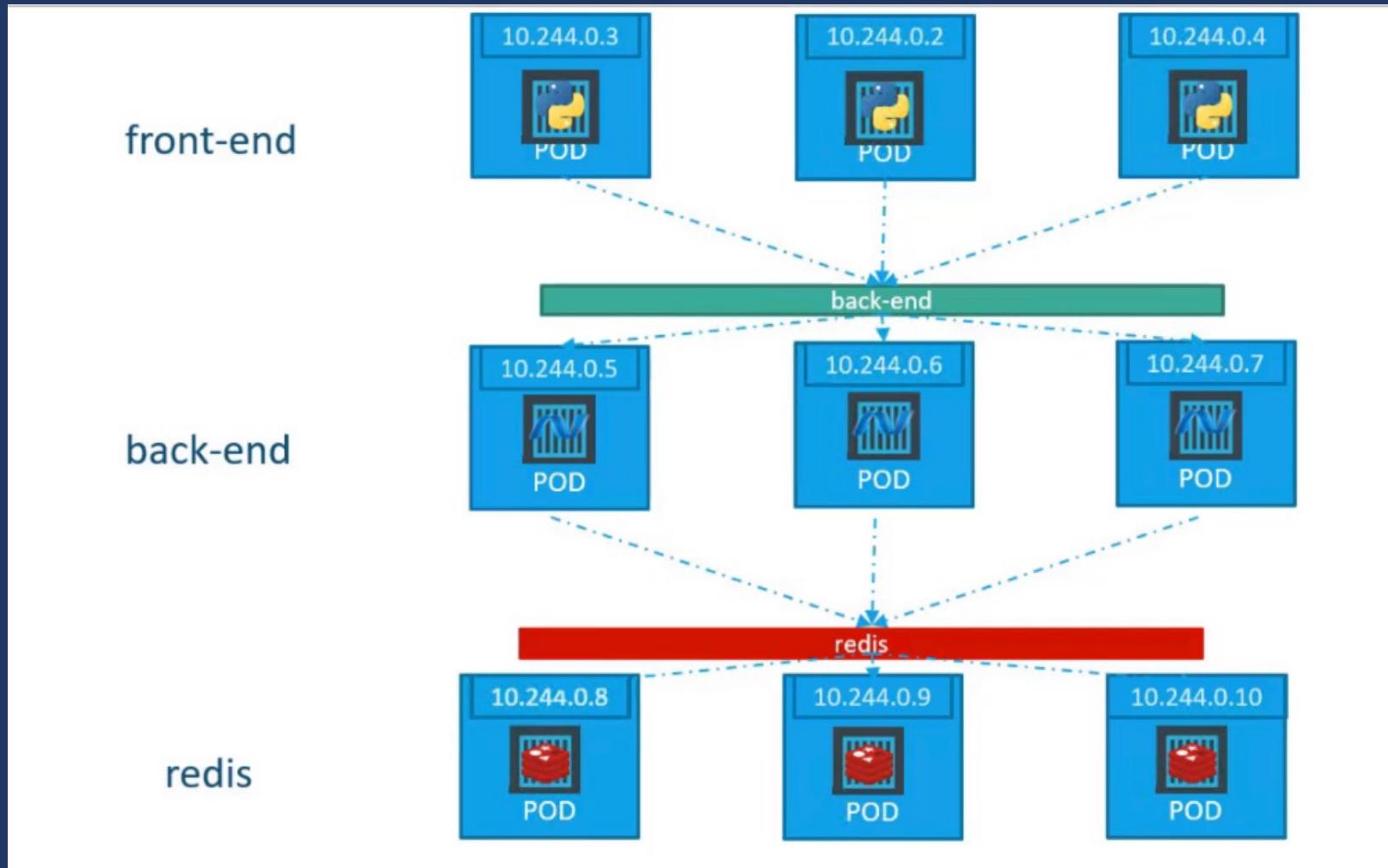
ClusterIP

Service Type: ClusterIP









```
service-definition.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: back-end

spec:
  type: ClusterIP
  ports:
    - targetPort: 80
      port: 80
```

```
service-definition.yml
apiVersion: v1
kind: Service
metadata:
  name: back-end
spec:
  type: ClusterIP
  ports:
    - targetPort: 80
      port: 80
  selector:
    app: myapp
    type: back-end
```

```
pod-definition.yml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
spec:
  containers:
    - name: nginx-container
      image: nginx
```

```
service-definition.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: back-end
spec:
  type: ClusterIP
  ports:
    - targetPort: 80
      port: 80

  selector:
    app: myapp
    type: back-end
```

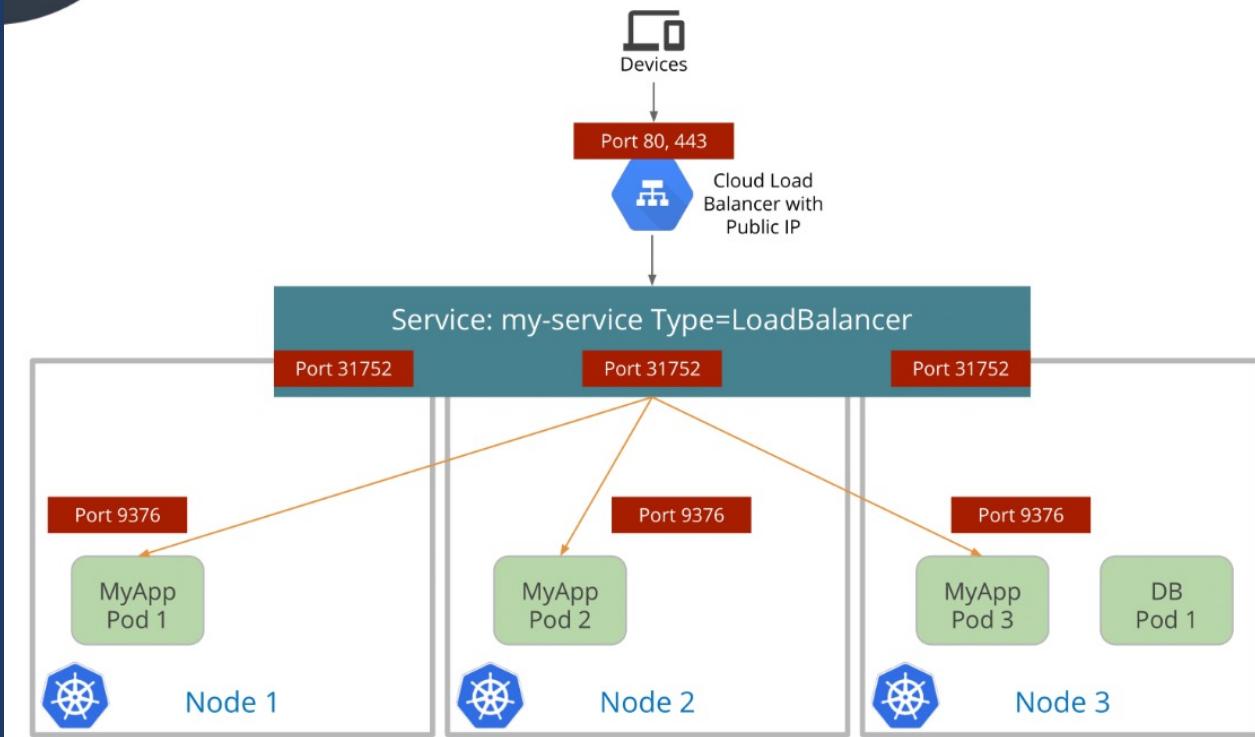
```
> kubectl create -f service-definition.yml
service "back-end" created
```

```
> kubectl get services
```

| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|------------|-----------|----------------|-------------|---------|-----|
| kubernetes | ClusterIP | 10.96.0.1 | <none> | 443/TCP | 16d |
| back-end | ClusterIP | 10.106.127.123 | <none> | 80/TCP | 2m |

LoadBalancer

Service Type: LoadBalancer

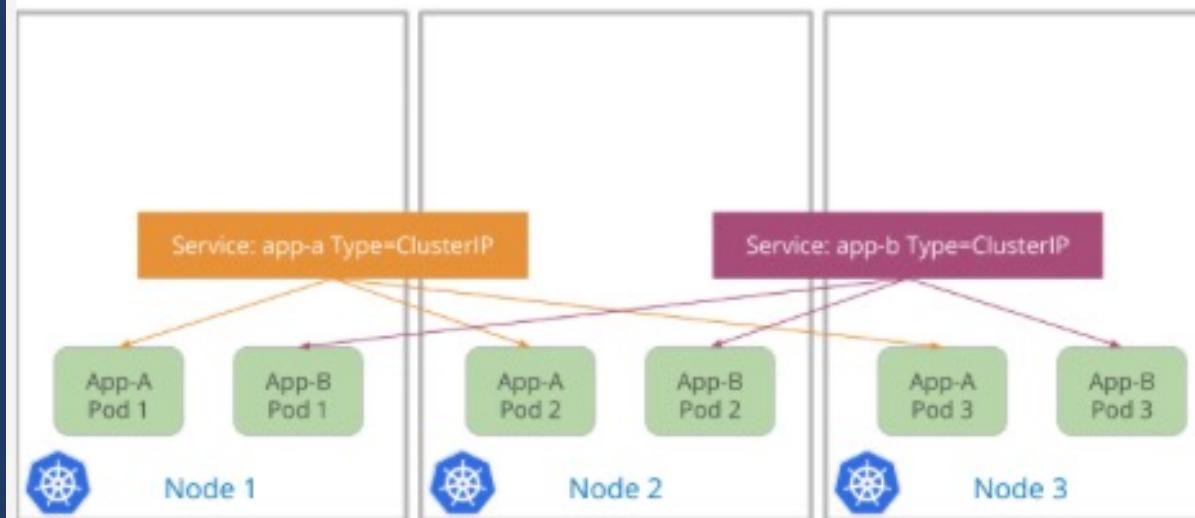


```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  type: LoadBalancer
```


Service Type: Ingress (1)

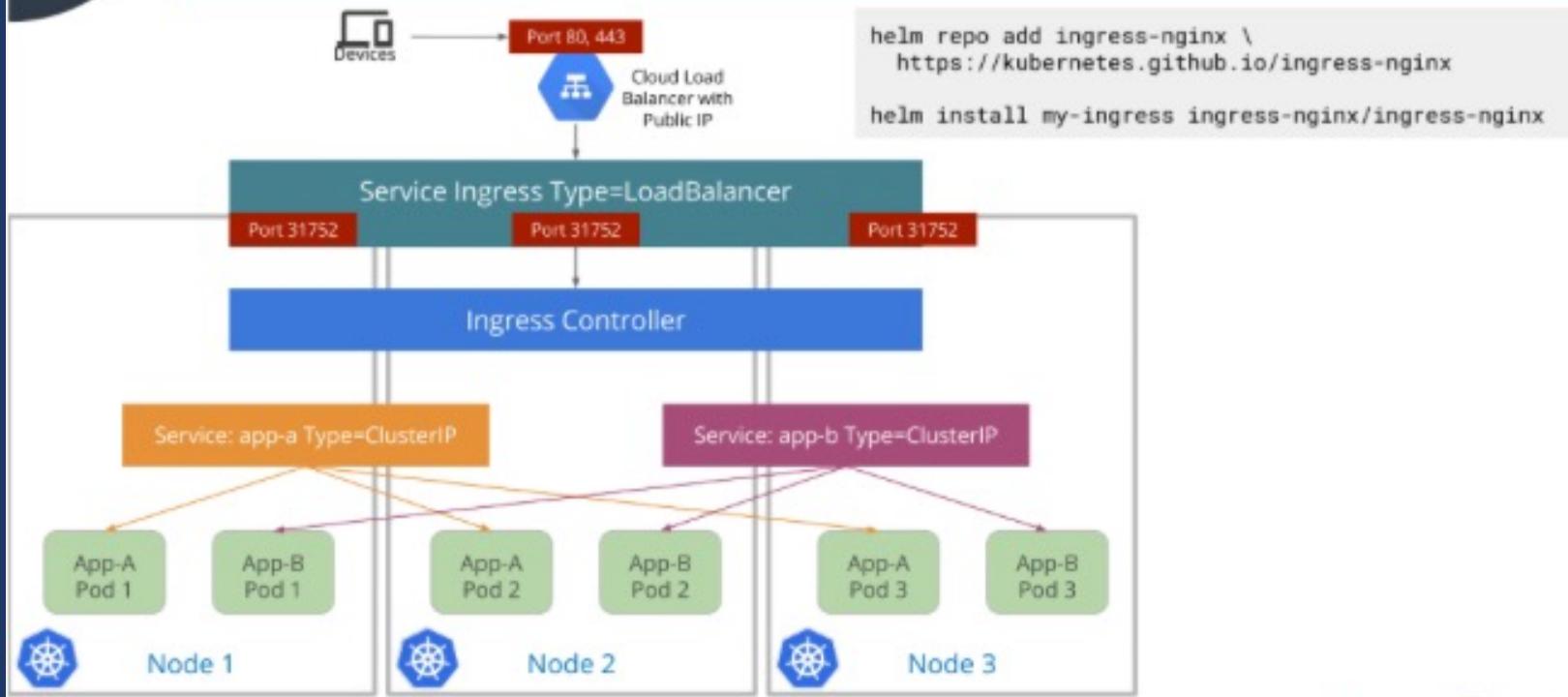


Devices

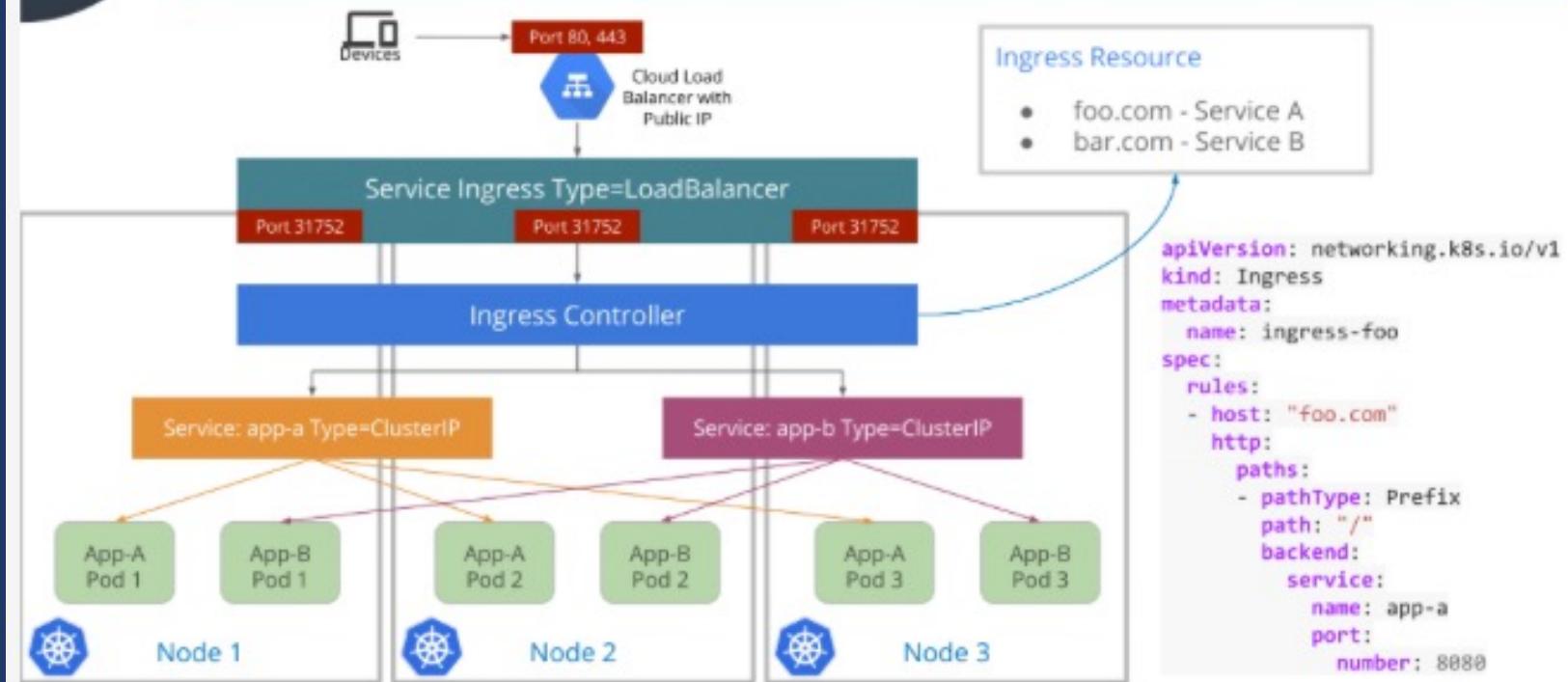


```
apiVersion: v1
kind: Service
metadata:
  name: app-a
spec:
  selector:
    app: App-A
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

Service Type: Ingress (2)



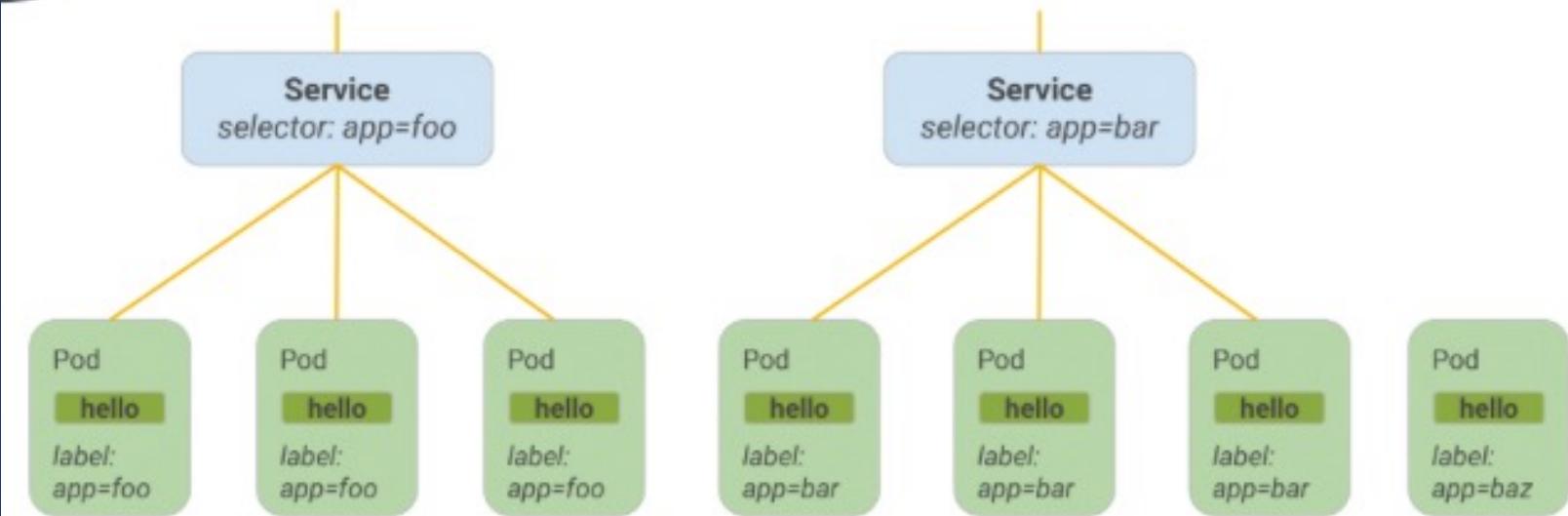
Service Type: Ingress (3)



Kubernetes Concept (5)



Labels and Selectors



Kubernetes Manifest File



Cloud Native Computing Foundation

Kubernetes Manifest File

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: default
spec:
  containers:
    - name: busybox
      image: busybox
      command:
        - sleep
        - "3600"
```

This is what called
Infrastructure as Code
(IaC)

YAML

- YAML is markup language. It is **human-readable** and easy to understand syntax.
- YAML has format for specific **configuration** information such as POD & Deployment definitions.
- Many software relate like **Kubernetes**, **SDN** and **OpenStack** etc.
- Similar **JSON language**
- Advantage to use YAML format
 - **Infrastructure as a code** (IAAC) - keep it on standard template.
 - **Stability** - keep it on version control, keep history change.
 - **Flexibility** - higher dynamic than use command line.

```
# Identity customer records
group_one:
  name: John Franky
  job: Developer
  skills:
    - Python
    - Ruby
    - customer
```

Manifest Structure

```
---  
apiVersion: v1      ← Define resource APIs  
kind: Pod          ← Define resource kind  
metadata:  
  name: static-web ← Define name of Pod  
  labels:           ← Define toggle label of Pod  
  role: myrole  
spec:              ← Define metadata like  
  containers:  
    - name: web  
      image: nginx  
      ports:  
        - name: web  
          containerPort: 80  
          protocol: TCP
```

Define resource APIs

Define resource kind

Define name of Pod

Define toggle label of Pod

Define metadata like
name, image and port of container(s)

