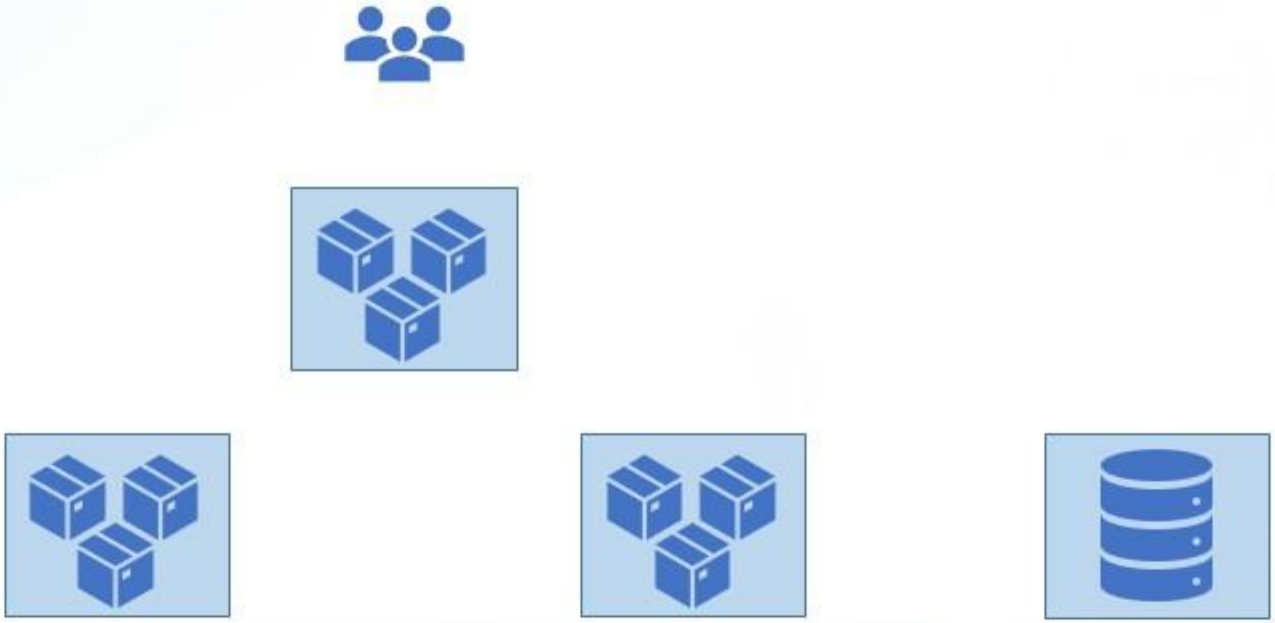
SkillAssure

Kubernetes Services

Services

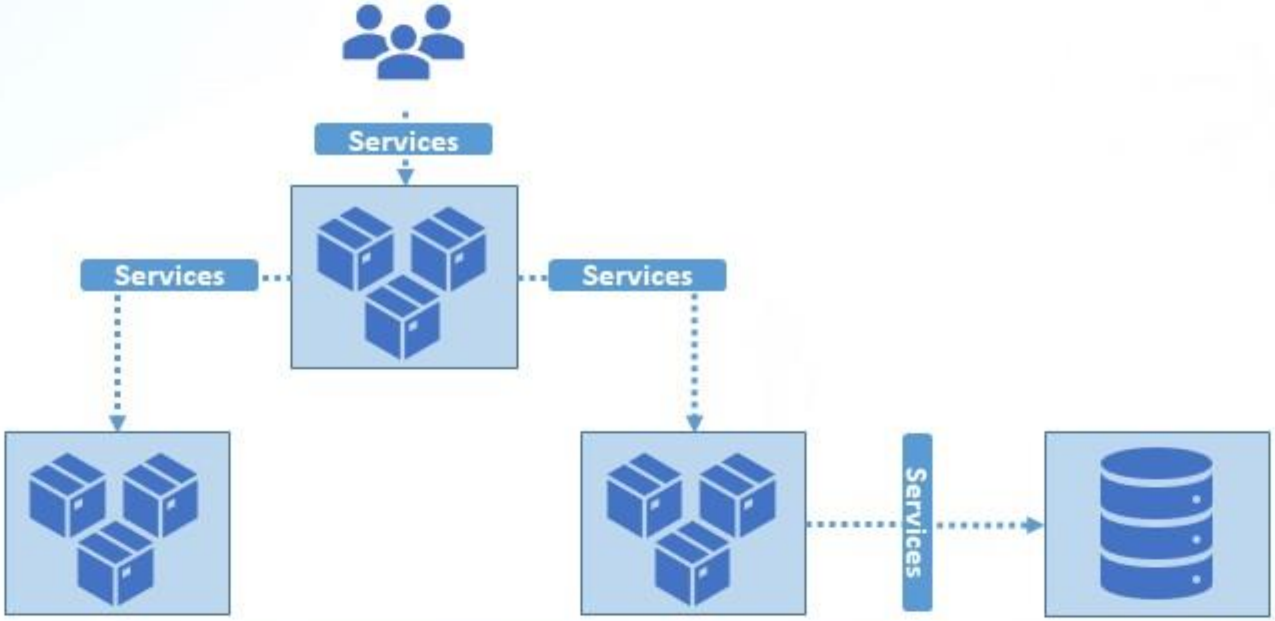
- Kubernetes services enable communication between various components within and outside of the application.
- Kubernetes services connects our application with other applications or users.
- For example our application has groups of pods running various applications such as
  - Frontend Application for serving load to users
  - Backend Application for processes
  - Any other app for connecting to an external data source.



SkillAssure

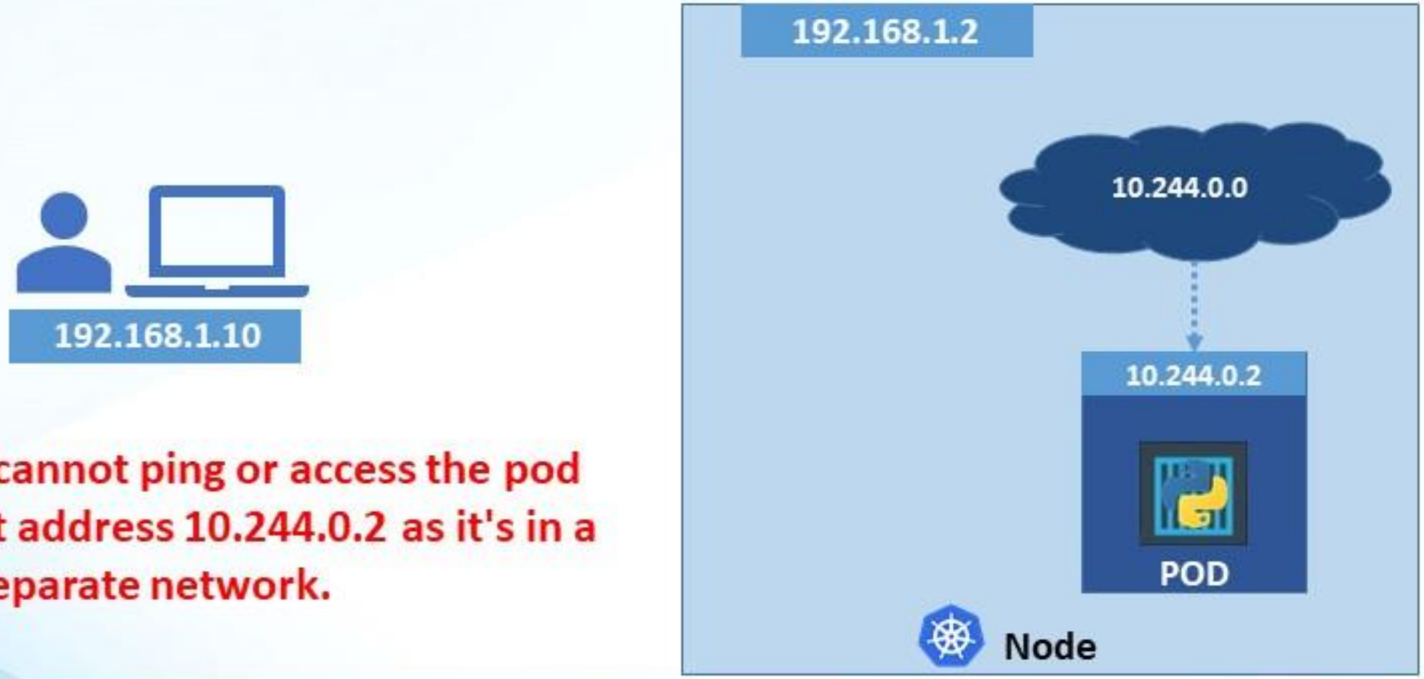
Services

- Services enable connectivity between these groups of pods
- Services enable:
  - Frontend application to be made available to end users
  - Communication between backend and frontend pods
  - Establishing connectivity to an external data source



Services

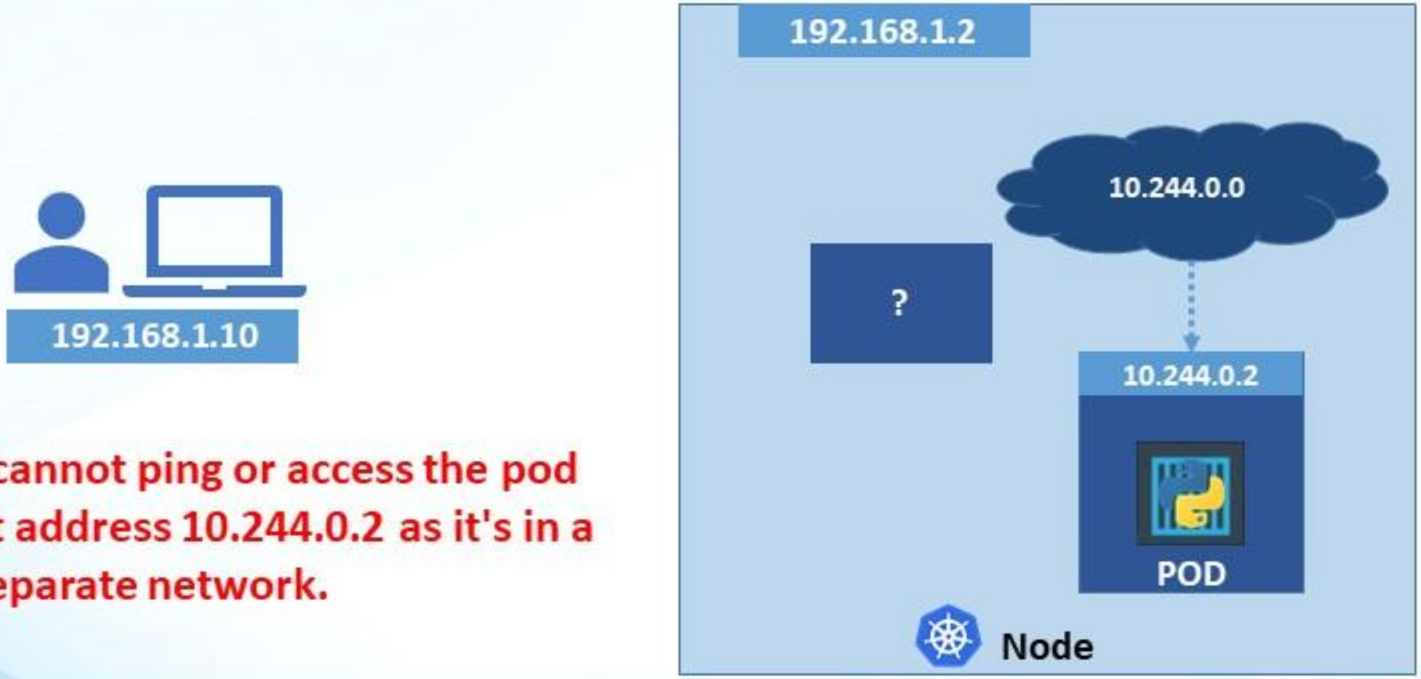
- Let's look at the existing setup
  - The Kubernetes node has an IP address, i.e. 192.168.1.2
  - My laptop is on the same network as well & it has an IP address 192.168.1.10
  - The internal pod network is in the range 10.244.0.0
  - POD has an IP 10.244.0.2 hosting a web page
- How do I access the web page as an external user ?



I cannot ping or access the pod at address 10.244.0.2 as it's in a separate network.

Services

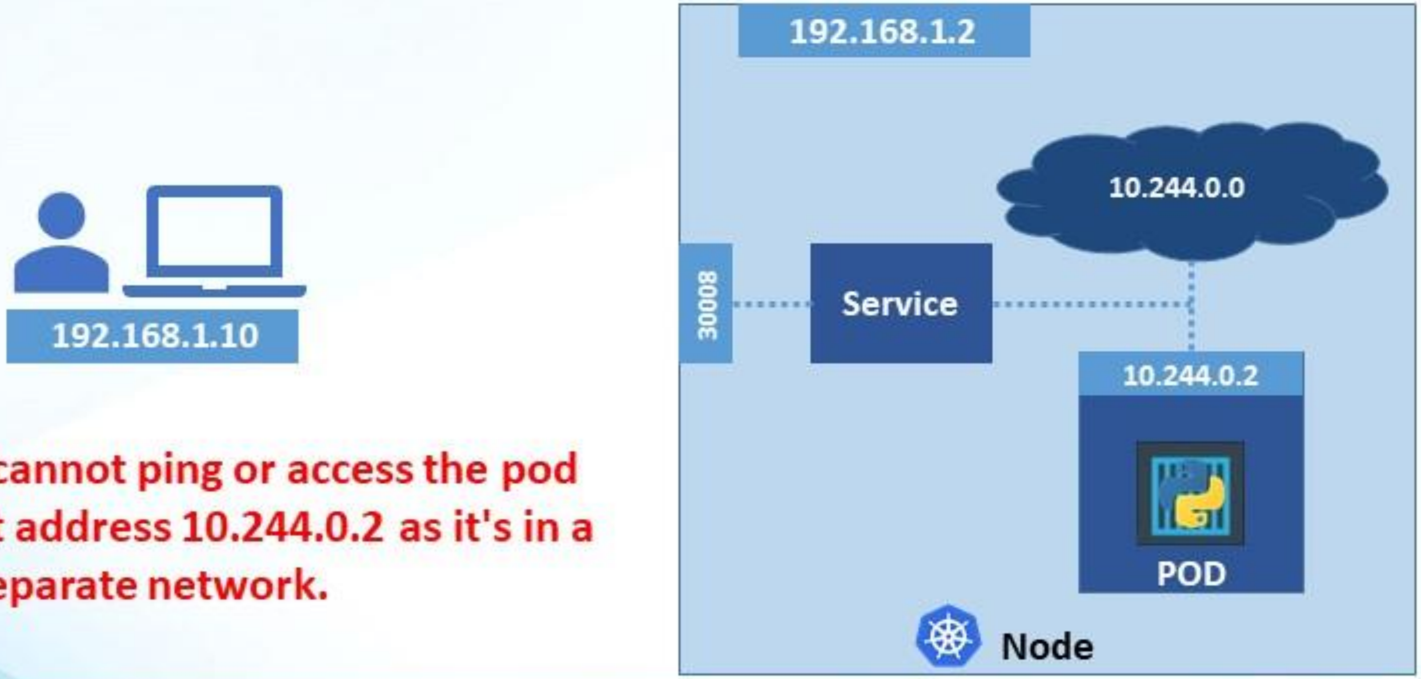
- I want to access the web page simply by accessing the IP of Kubernetes node.
- We need something in the middle to help us route our requests.
- The request from our laptop through the node to the pod running the web container.



I cannot ping or access the pod at address 10.244.0.2 as it's in a separate network.

Services

- This is where we use Kubernetes service.
- Kubernetes service is an object just like pods, replicaset or deployments
- One of its use case is to listen to a port on the node and forward request on that port to a port on the pod.
- This type of service is known as a **NodePort Service**



I cannot ping or access the pod at address 10.244.0.2 as it's in a separate network.

Services Types

NodePort

Service makes an internal POD accessible on a port on the Node

ClusterIP

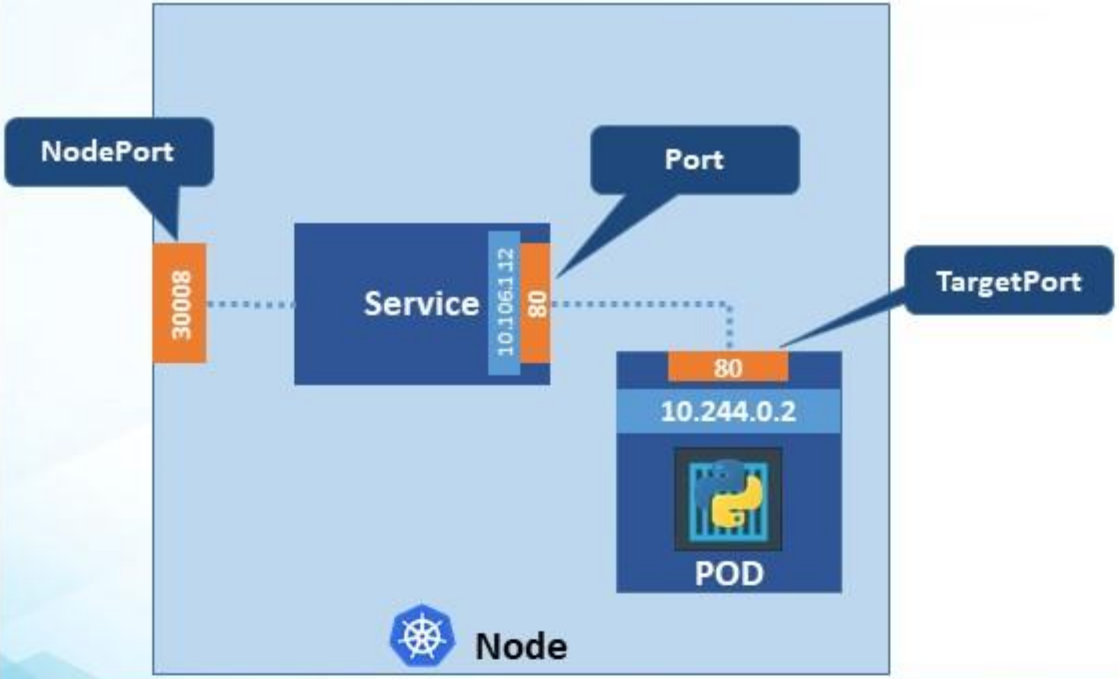
Service creates a virtual IP inside the cluster to enable communication between different services (e.g. Set of front-end & back-end servers)

LoadBalancer

Service provisions a load balancer for our application in supported cloud providers, e.g. to distribute load across the different web servers in our front-end tier.

Services - NodePort

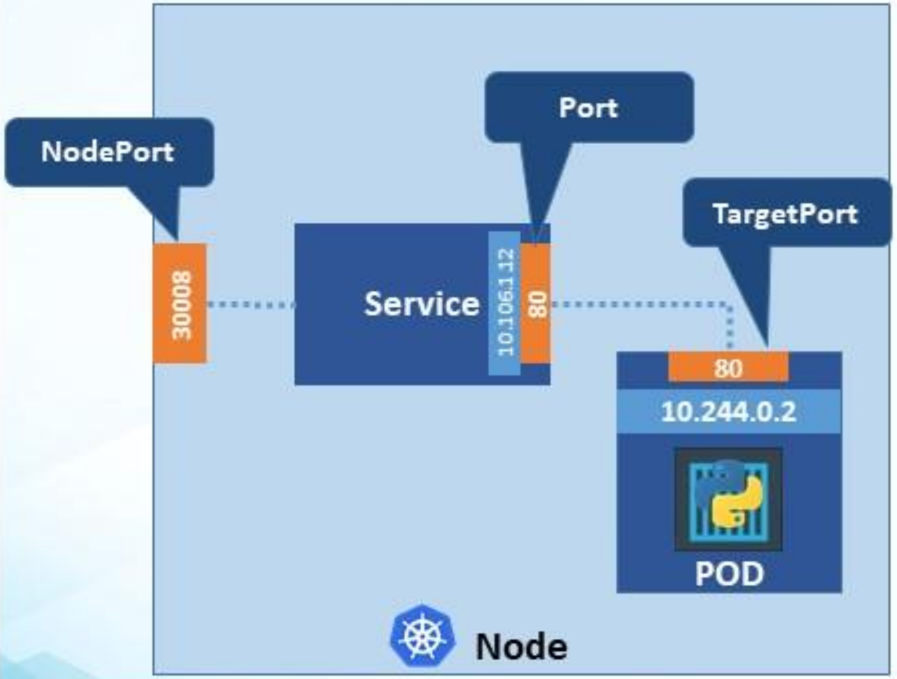
- Let's take a closer look at Node Port Service
- There are 3 ports involved
  - **Port on the Pod - TargetPort**
  - **Port on the Service - Port**
  - **Port on the Node - NodePort**



- **Target Port –** Where web server runs and where service forwards requests.
- **Port –** Port on service. Service has its own IP address, called as Cluster IP of Service.
- **NodePort -** Used to access the web server externally. NodePorts valid range is from 30000 to 32767

Services - NodePort

- To create a Service we will use the definition file
- The high-level structure of the file remains the same
  - **apiVersion**
  - **kind**
  - **metadata**
  - **spec**

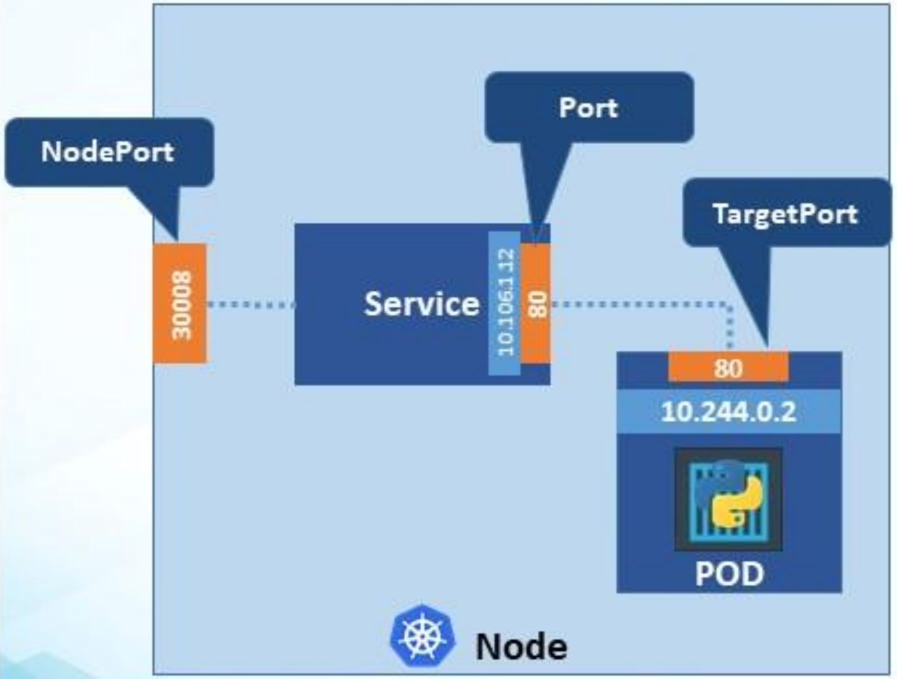


service-definition.yml

```
apiVersion:
kind:
metadata:



spec:
```

Services - NodePort

- apiVersion – v1
- kind - Service
- metadata – It can have name and labels
- spec – Here we define the actual service

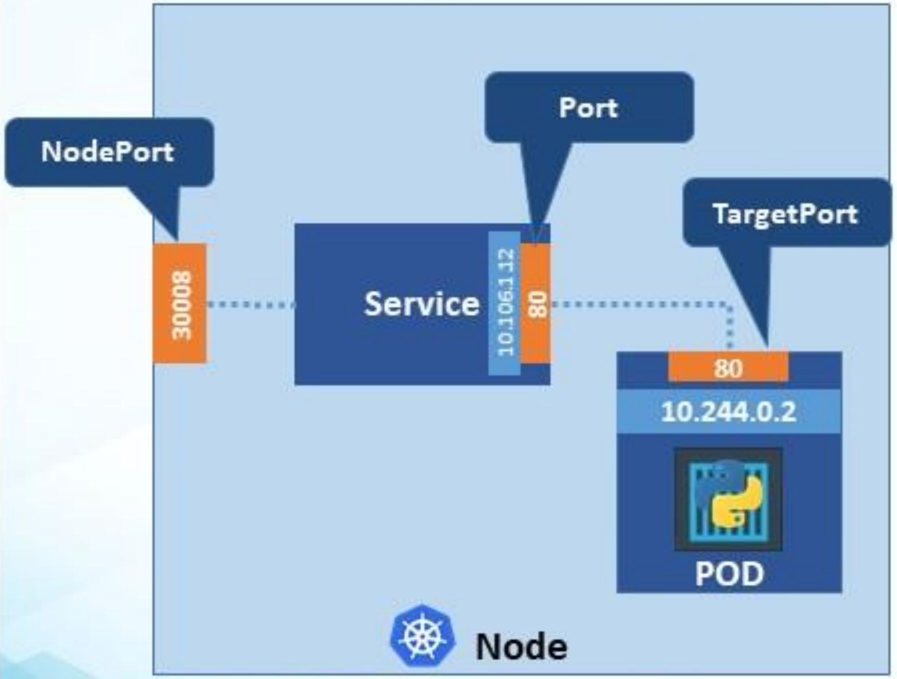


service-definition.yml

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

spec:
```

Services - NodePort

- Spec has
  - **type** - Refers to the type of service we are creating, e.g. nodePort, clusterIP, loadBalancer
  - **ports** – Specify different ports used in the service

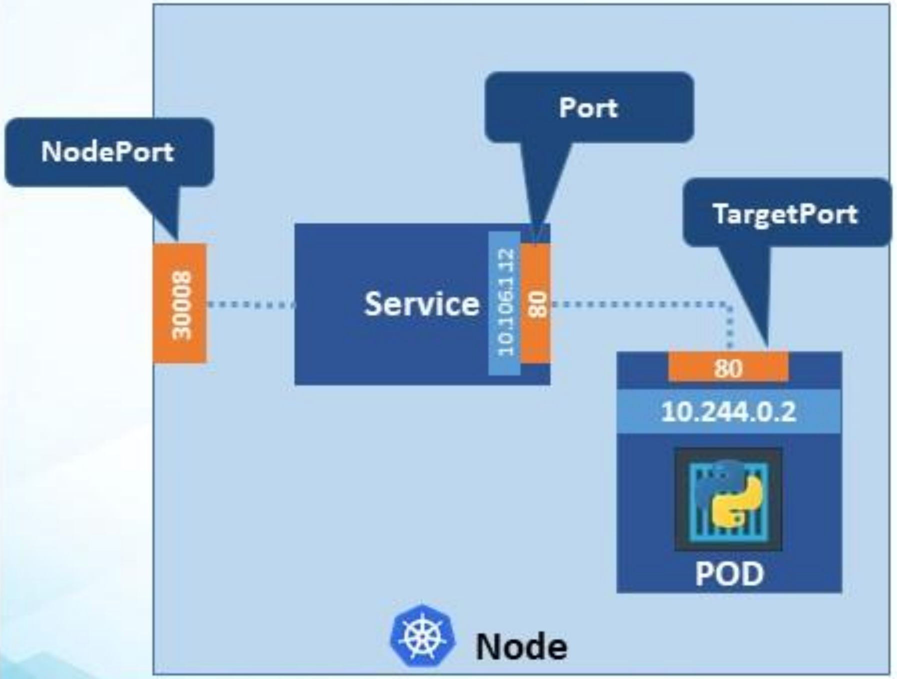


service-definition.yml

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

spec:
  type: NodePort
  ports:
```

Services - NodePort

- Ports has
  - **targetPort**
  - **port**
  - **nodePort**

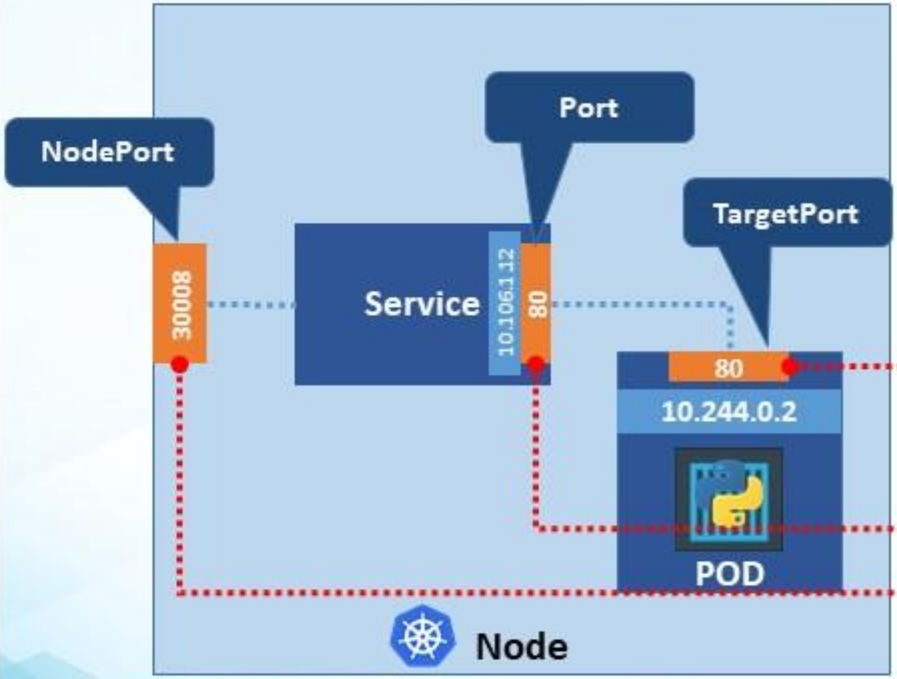


service-definition.yml

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

spec:
  type: NodePort
  ports:
    - targetPort:
      port:
      nodePort:
```

Services - NodePort

- Port is the only mandatory field.
- If you don't provide
  - **targetPort** - it assumes to be the same as port.
  - **nodePort** – it takes a free port automatically (between 30000-32767)

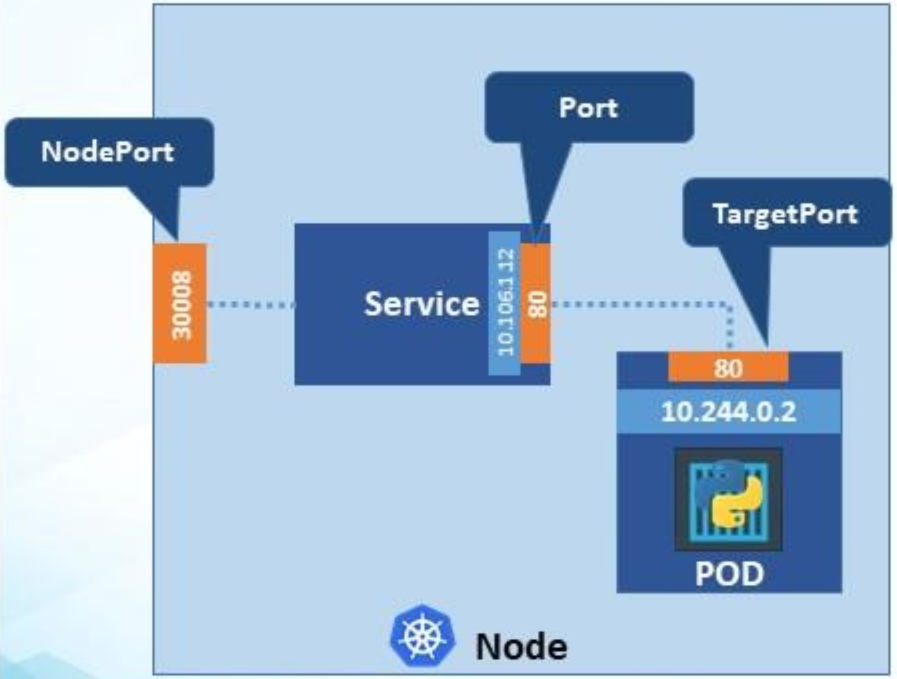


service-definition.yml

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

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

Services - NodePort

- Ports is an array.
- We can have multiple port mappings within a single service.

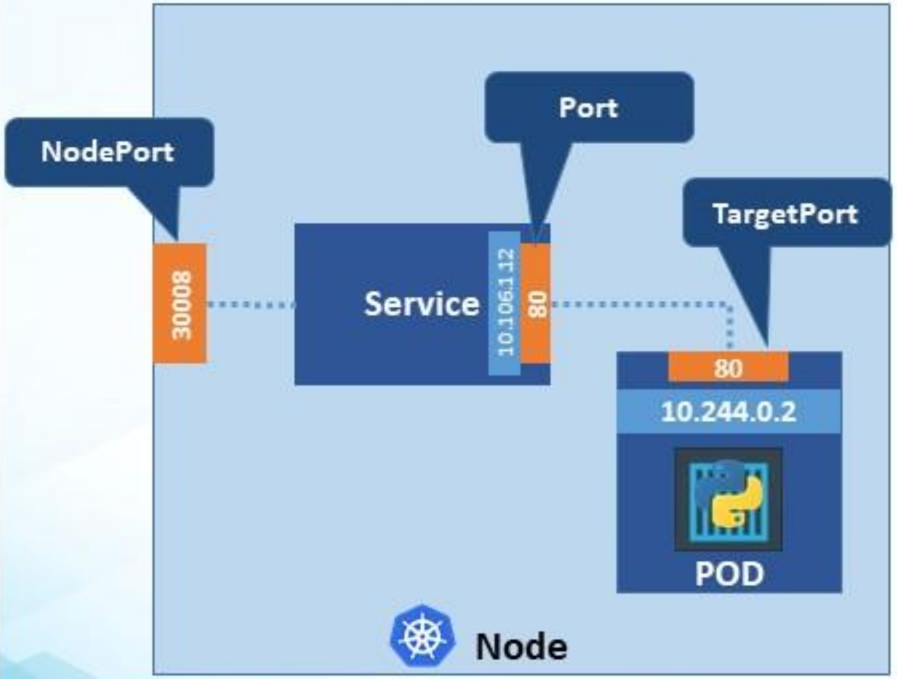


service-definition.yml

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

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

Services - NodePort

- **Something Missing?**
- Our definition file does not connect the service to the pod.
- We specified the targetPort but we didn't mention on which pod.

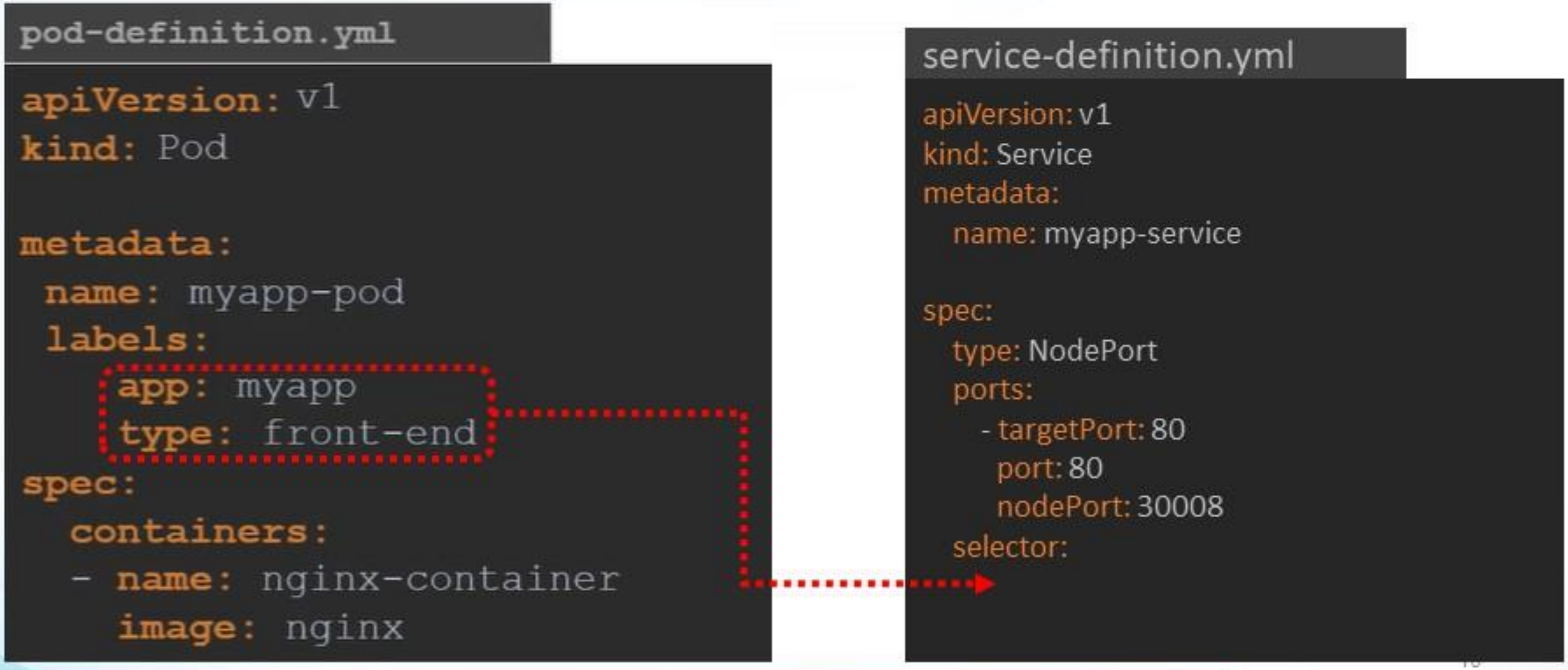


service-definition.yml

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

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

Services - NodePort

- We know that the pod was created with a label.
- We will use labels and selectors to link the service to the pod
- We have a new property in the specs section and that is called selector.



pod-definition.yml

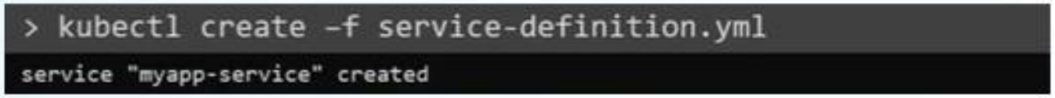
```
apiVersion: v1
kind: Pod

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

service-definition.yml

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

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

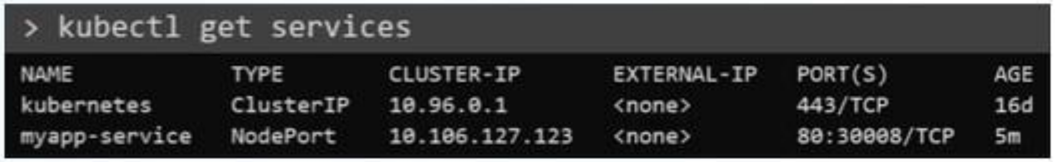
Services - NodePort

- Create a service



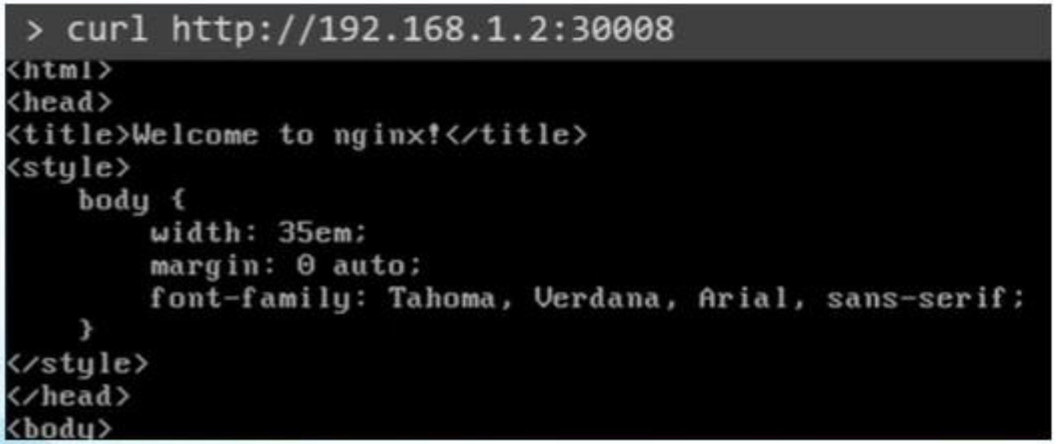
```
> kubectl create –f service-definition.yml
service "myapp-service" created
```

- See the created service

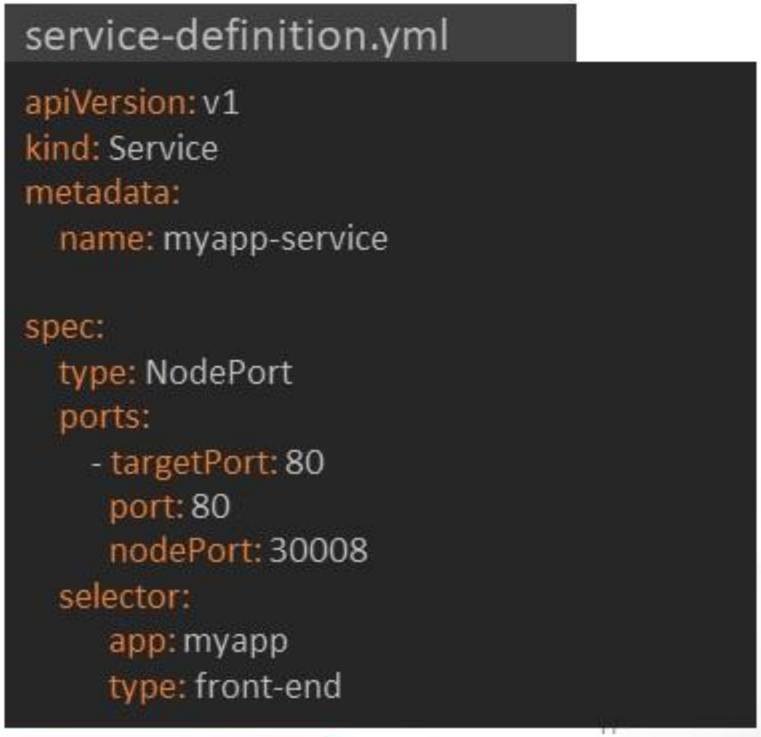


```
> kubectl get services
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP        16d
myapp-service  NodePort    10.106.127.123   <none>        80:30008/TCP   5m
```

- Access the web service using curl or a web browser



```
> 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-definition.yml

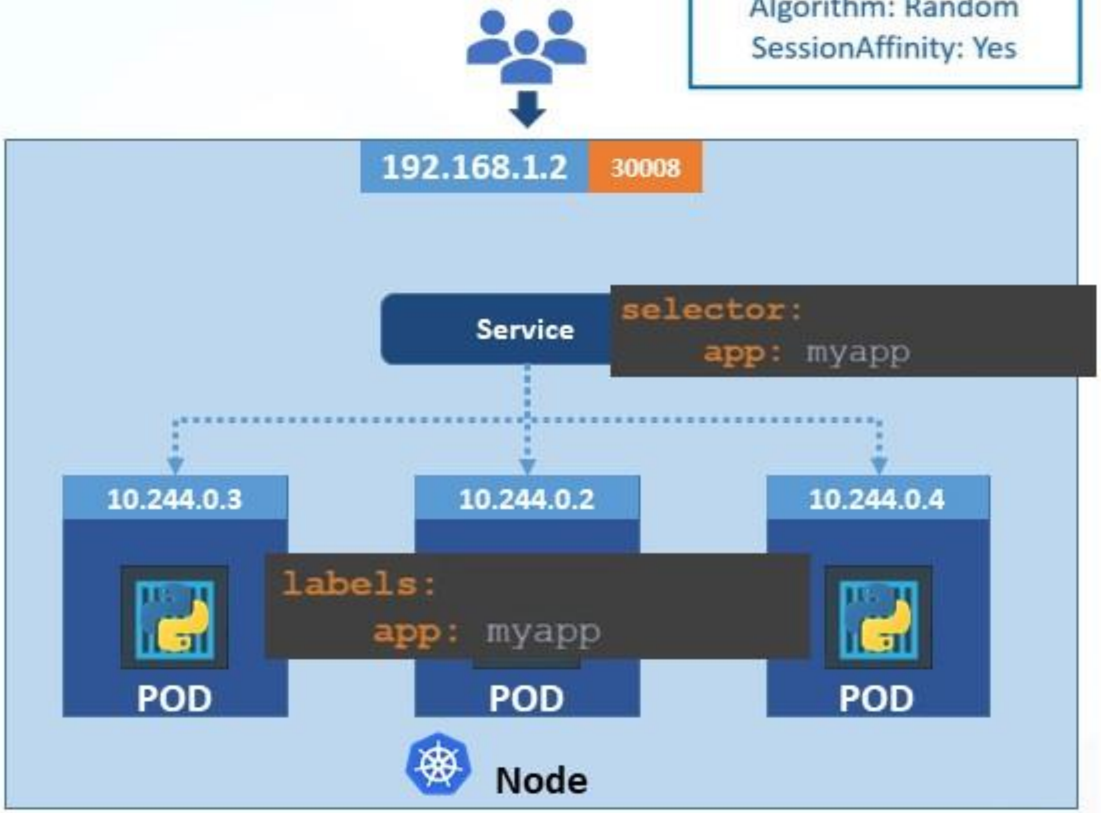
```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service

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

Services - NodePort

- We saw how to map a single Pod to a Service
- We can have multiple Pods running our web application
- How does service handle multiple Pods?

- They all have the same labels i.e. app: myapp
- The same label is used as a selector during creation of the service.
- So when the service is created it looks for Pods with the label and finds all 3.
- The service automatically selects all 3 Pods as endpoints to forward the external requests coming from users.
- We don't have to do any additional configuration for this.
- It uses **random algorithm** to balance the load

Algorithm: Random
SessionAffinity: Yes

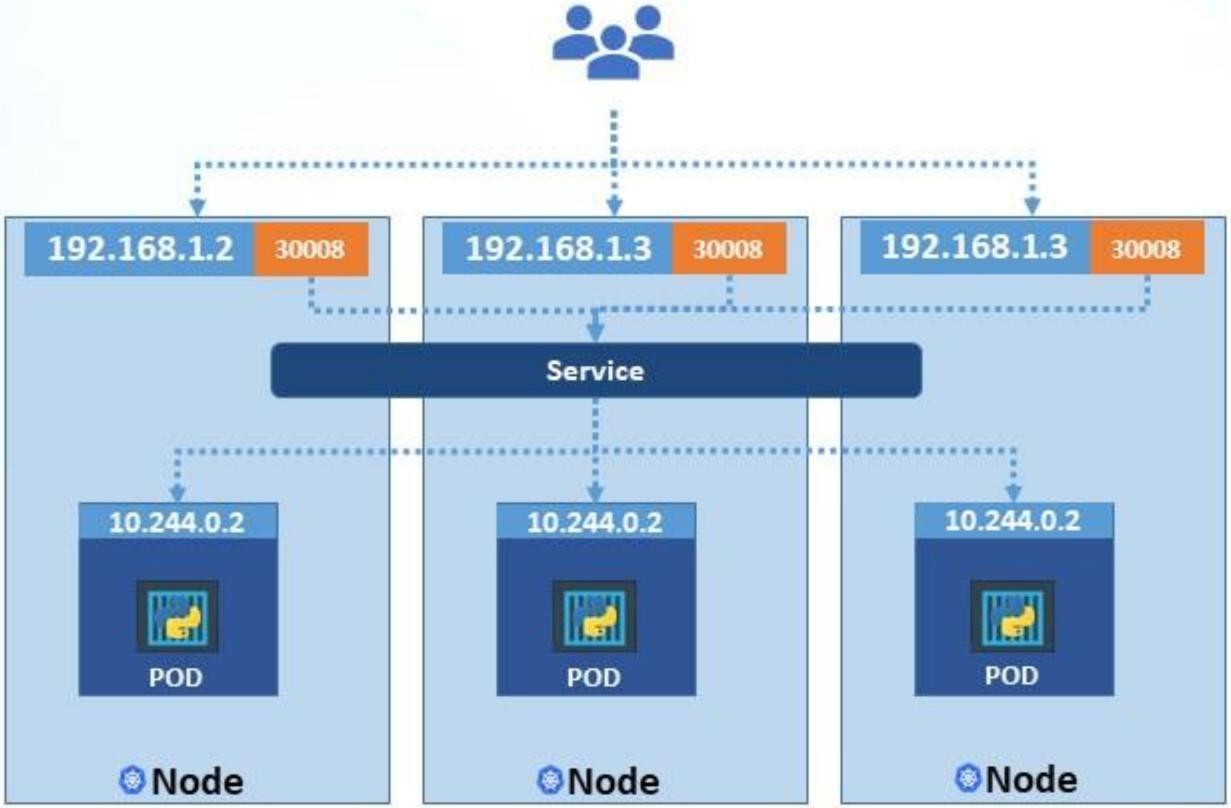


```
selector:
    app: myapp
```

```
labels:
    app: myapp
```

Services - NodePort

- What happens when the pods are distributed across multiple nodes?
- Kubernetes automatically creates a service that spans across all the nodes
- It maps the target port to the same node port on all the nodes in the cluster.
- We can access the application using the IP of any node in the cluster

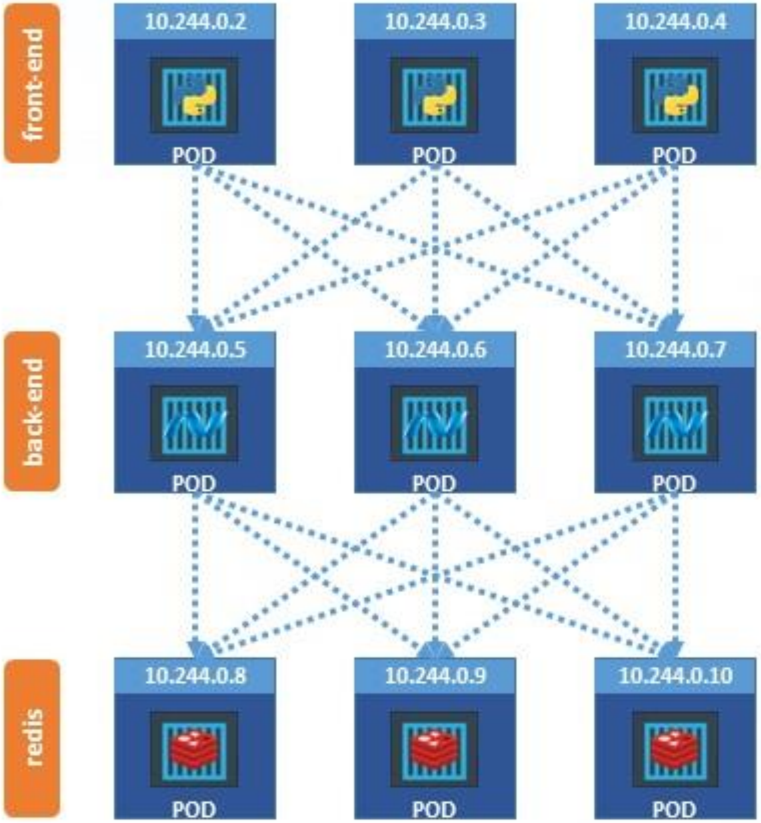


Demo – NodePort

Services – ClusterIP

ClusterIP

- A full stack web application may have multiple tiers.
  - front-end
  - back-end
  - redis
  - db
- The web front-end server needs to communicate to the back-end servers
- The back-end servers need to communicate to the database as well as the redis services etc.
- What is the right way to establish connectivity between these services or tiers of my application?



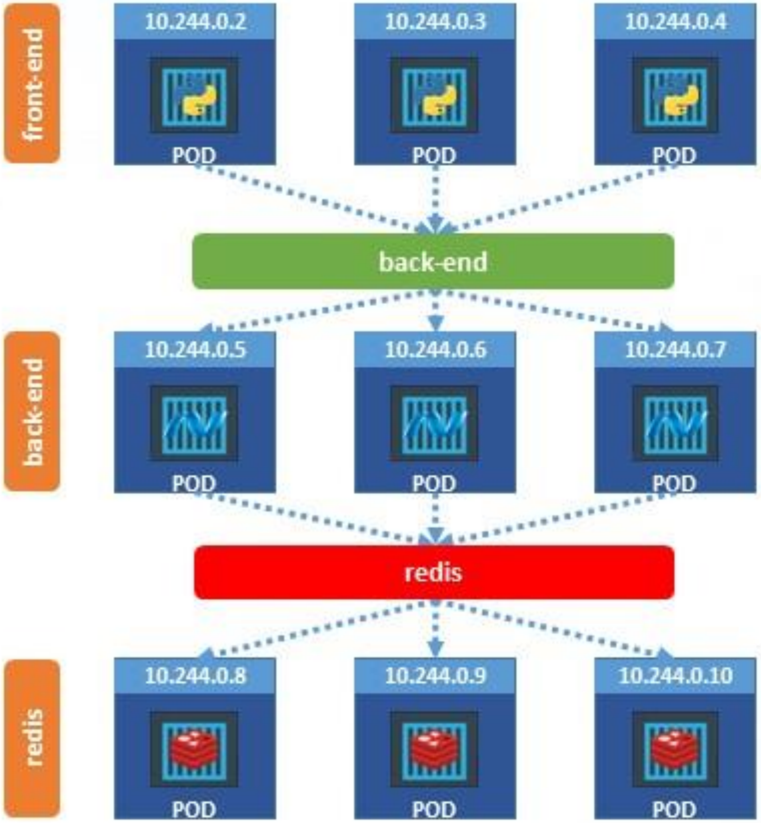
ClusterIP

- Pods has static IP addresses and cannot be used for internal communication

- Kubernetes service help us group the pods together
- It provide a single interface to access the pods in a group.
  - e.g. Service created for the back-end pods groups all the back-end pods together

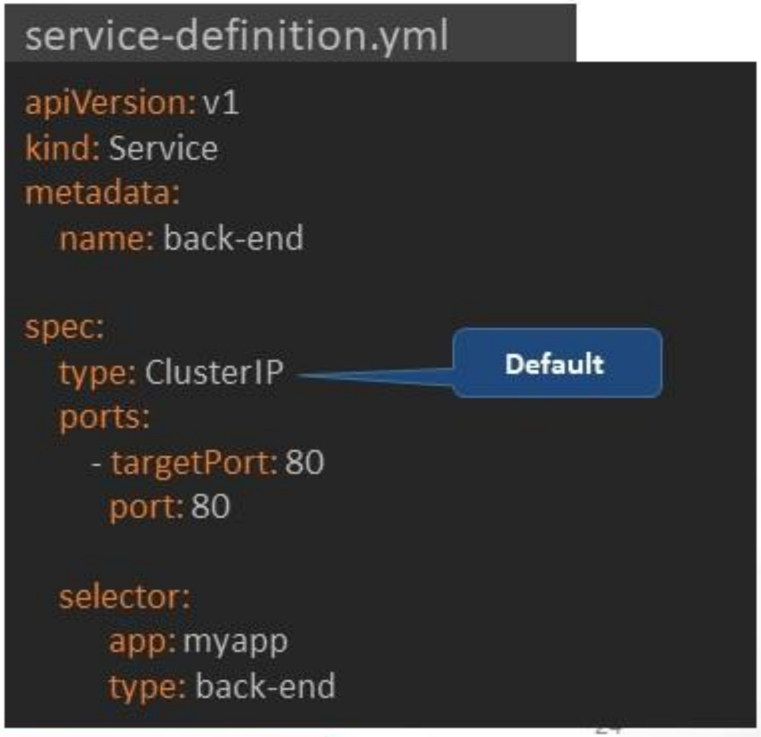
- Each service gets an IP and name assigned to it inside the cluster

- This type of service is known as **Cluster IP**



Services - ClusterIP

- We will use a definition file to create the service.
- Start with the default template
- Under “spec” mention type as ClusterIP
  - ClusterIP is the default type of service. Kubernetes will consider service type as ClusterIP If we do not specify it.
- Specify the targetPort & port
- Link the service to a set of pods using selector

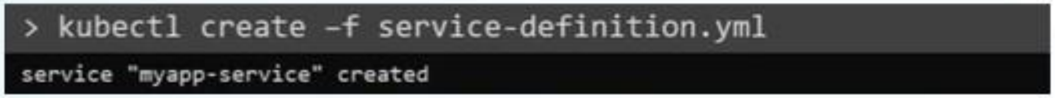


service-definition.yml

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

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

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

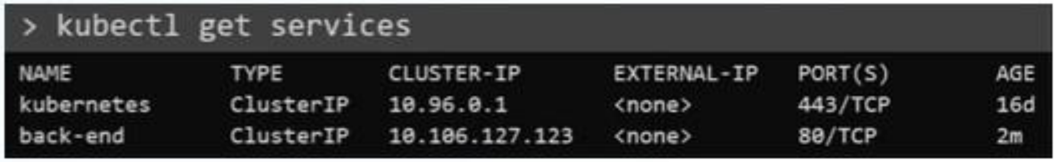
Services - ClusterIP

- Create a service



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

- See the created service



```
> 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
```

- Service can be accessed by other Pods using the cluster IP or the service name.

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
```

Service with YAML

Exercise 30

<u>Introduction</u>: Let us start with Services! Given a service-definition.yml file.

Instruction: Add all the root level properties to it. Note: Only add the properties, not any values.

service-definition.yml

Exercise 30 - Solution

Introduction: Let us start with Services! Given a service-definition.yml file.

Instruction: Add all the root level properties to it. Note: Only add the properties, not any values.

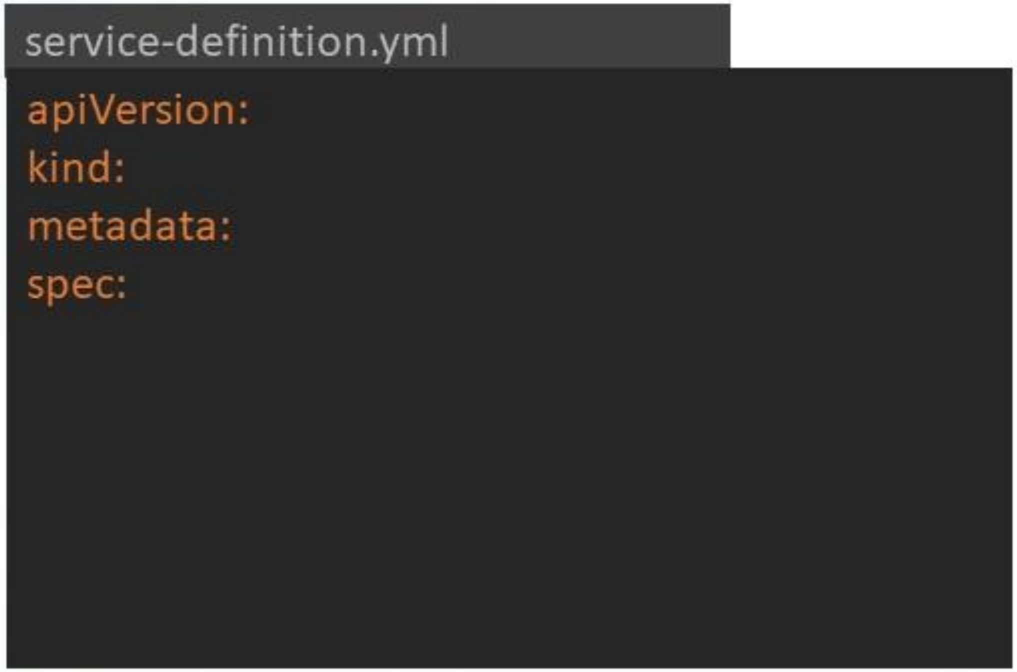
service-definition.yml

```yaml
apiVersion:
kind:
metadata:
spec:
```

Exercise 31

Introduction: Let us now add values for Service. Service is under apiVersion v1

Instruction: Update values for apiversion and kind.



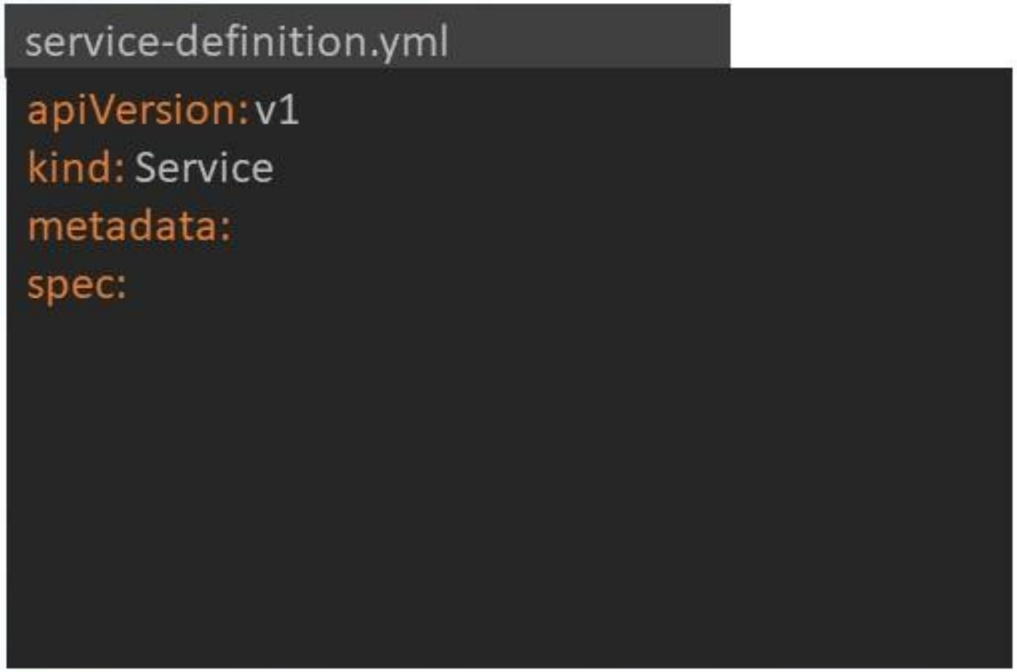
service-definition.yml

```yaml
apiVersion:
kind:
metadata:
spec:
```

Exercise 31 - Solution

Introduction: Let us now add values for Service. Service is under apiVersion v1

Instruction: Update values for apiVersion and kind.



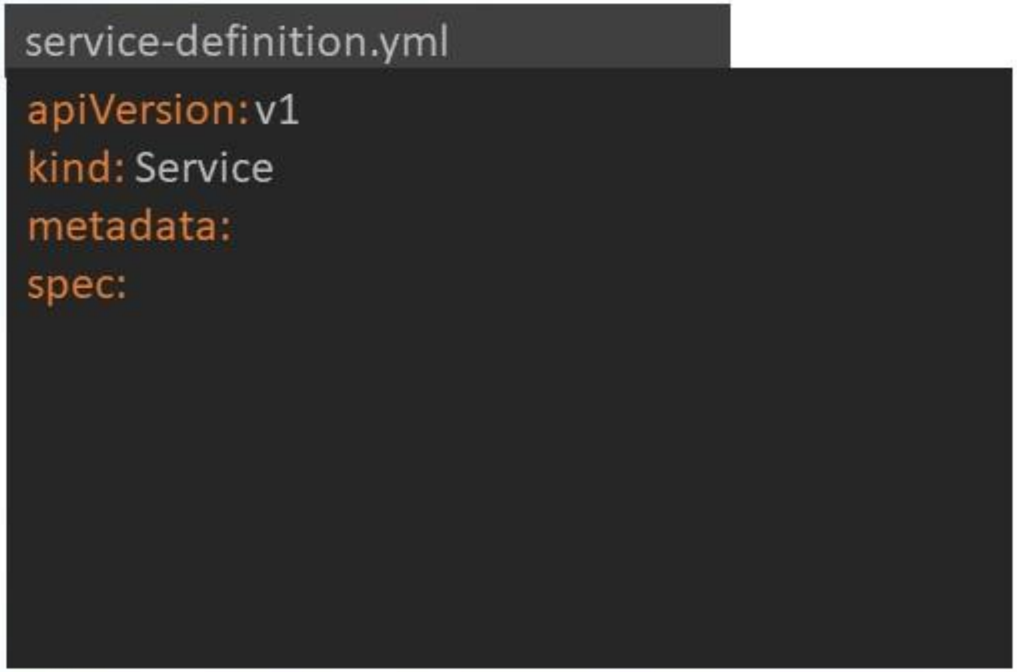
service-definition.yml

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

Exercise 32

Introduction: Let us now add values for metadata

Instruction: Add a name for the service = **frontend** and a label = **app=>myapp**



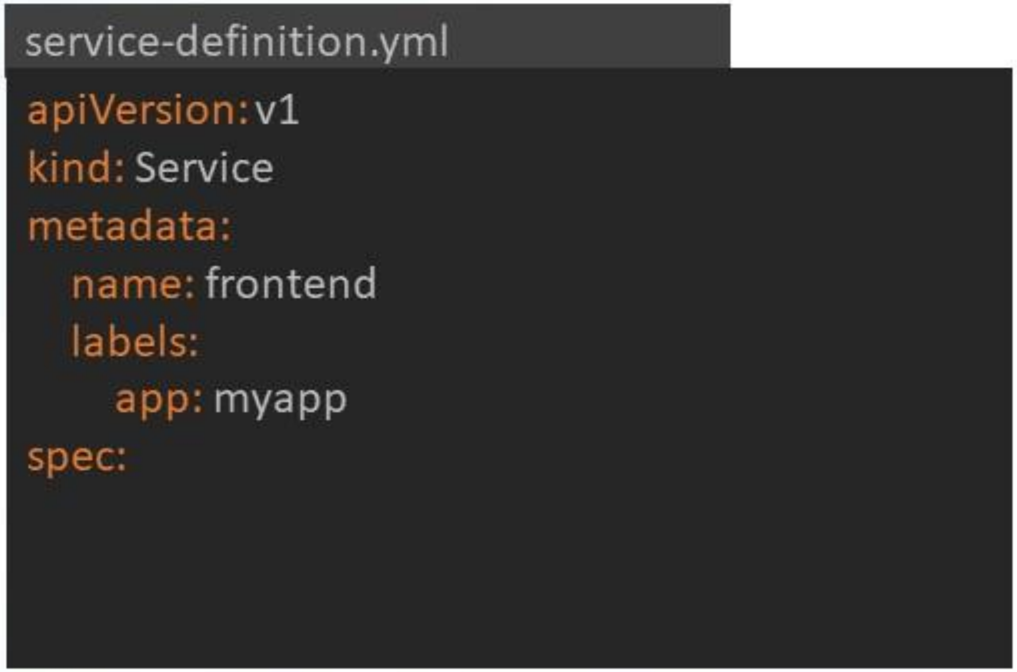
service-definition.yml

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

Exercise 32 - Solution

Introduction: Let us now add values for metadata

Instruction: Add a name for the service = **frontend** and a label = **app=>myapp**



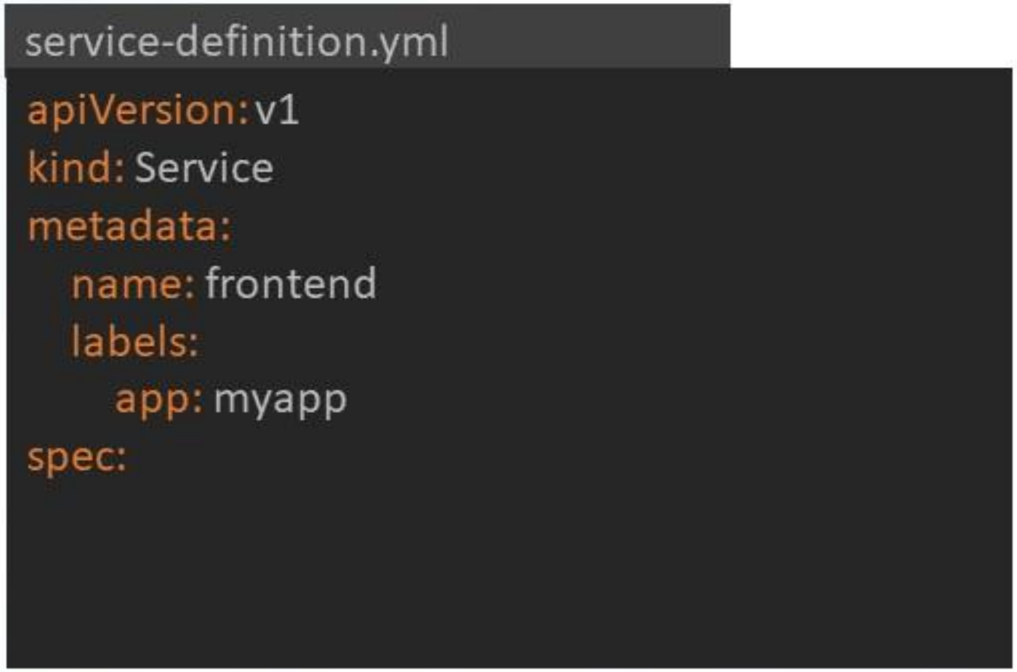
service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
```

Exercise 33

Introduction: Let us now add value for spec section. The spec section for Services have type, selectors and ports

Instruction: Add properties under spec section – **type, selectors and ports**. Do not add any value for them



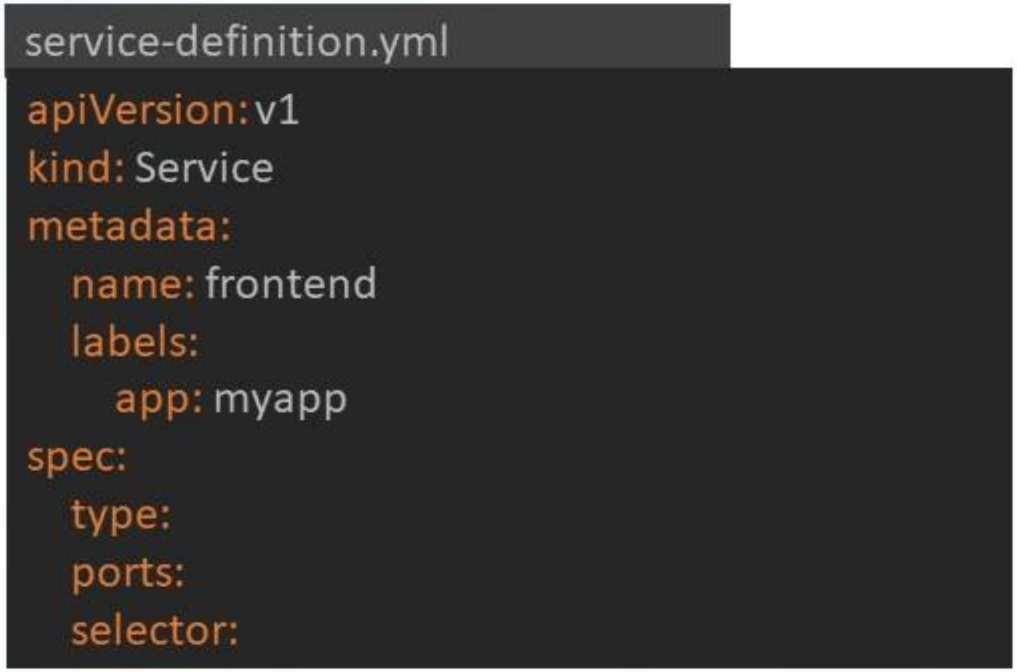
service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
```

Exercise 33 - Solution

Introduction: Let us now add value for spec section. The spec section for Services have type, selectors and ports

Instruction: Add properties under spec section – **type, selectors and ports**. Do not add any value for them



service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
  type:
  ports:
  selector:
```

Exercise 34

Introduction: Let us now add value for ports. Port is an Array/List. Each item in the list has a set of properties – port and targetPort

Instruction: Create an Array/List item under **ports**. Add a dictionary with properties **port** and **targetPort**. Set values for both to port 80.

service-definition.yml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
  type:
  ports:
  selector:
```

Exercise 34 - Solution

<u>**Introduction**</u>: Let us now add value for ports. Port is an Array/List. Each item in the list has a set of properties – port and targetPort

Instruction: Create an Array/List item under **ports**. Add a dictionary with properties **port** and **targetPort**. Set values for both to port 80.

service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
  type:
  ports:
    - port: 80
      targetPort: 80
  selector:
```

Exercise 35

Introduction: Let us now add value for type. Since we are creating a frontend service for enabling external access to users, we will set it to NodePort

Instruction: Set value for **type** to **NodePort**

service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
  type:
  ports:
    - port: 80
      targetPort: 80
  selector:
```

Exercise 35 - Solution

Introduction: Let us now add value for type. Since we are creating a frontend service for enabling external access to users, we will set it to NodePort

Instruction: Set value for **type** to **NodePort**

service-definition.yml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
  selector:
```

Exercise 36

Introduction: Let us now add value for selector. We need to link the Service to the Pods Created by the deployment

Instruction: Given the deployment-definition.yml file we created in the previous section. Copy the appropriate labels and paste it under selector section-definition.yml file

deployment-definition.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: mywebsite
    tier: frontend
spec:
  replicas: 4
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
    spec:
      containers:
        - name: nginx
          image: nginx
  selector:
    matchLabels:
      app: myapp
```

service-definition.yml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
  selector:
```

Exercise 36 - Solution

Introduction: Let us now add value for selector. We need to link the Service to the Pods Created by the deployment

Instruction: Given the deployment-definition.yml file we created in the previous section. Copy the appropriate labels and paste it under selector section-definition.yml file

deployment-definition.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: mywebsite
    tier: frontend
spec:
  replicas: 4
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
    spec:
      containers:
        - name: nginx
          image: nginx
  selector:
    matchLabels:
      app: myapp
```

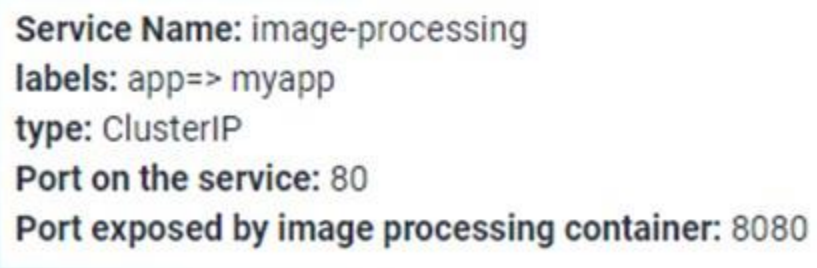
service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: myapp
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: myapp
```

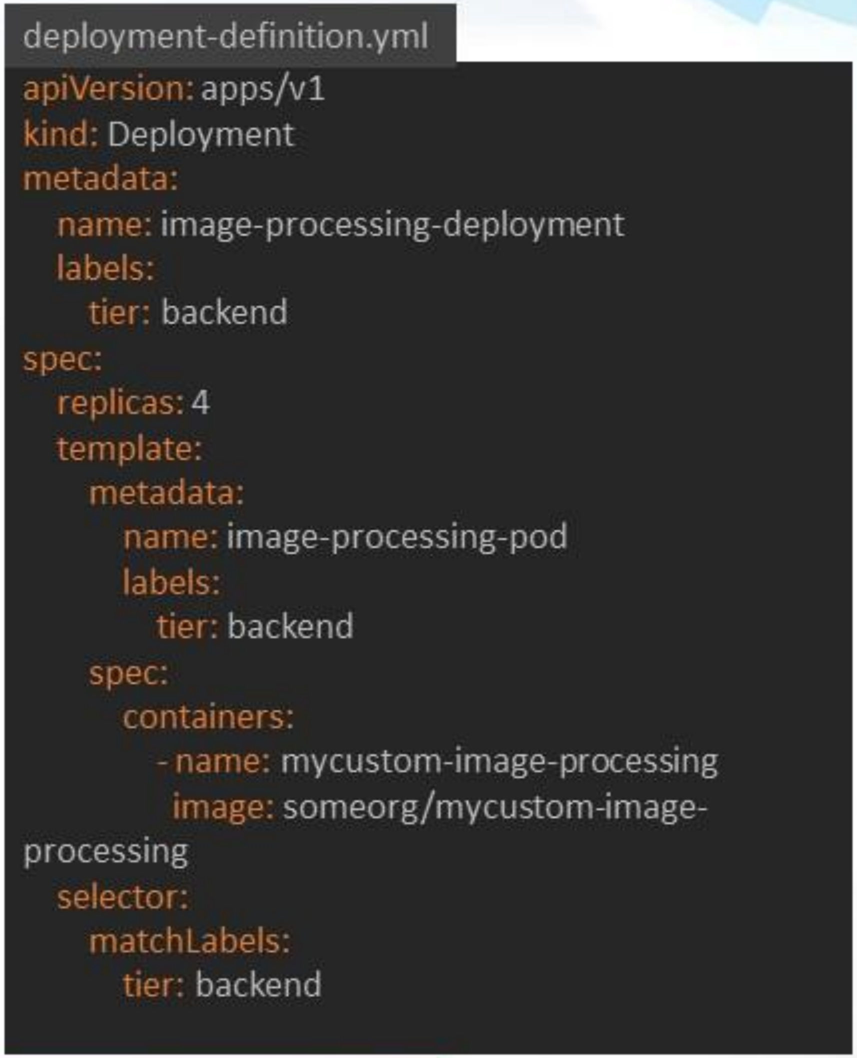
Exercise 37

Introduction: Create a service-definition.yml file from scratch. You are tasked to create a service to enable the frontend pods to access a backend set of pods

Instruction: Use the information provided in the table below to create a backend service definition file. Refer to the provided deployment-definition file for information regarding the PODs



Service Name: image-processing
labels: app=> myapp
type: ClusterIP
Port on the service: 80
Port exposed by image processing container: 8080



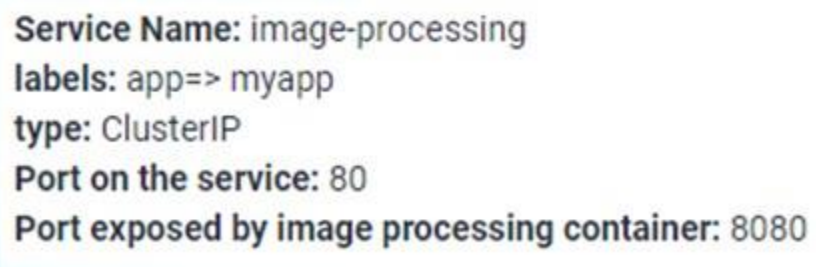
deployment-definition.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
   name: image-processing-deployment
   labels:
      tier: backend
spec:
   replicas: 4
   template:
      metadata:
         name: image-processing-pod
         labels:
            tier: backend
      spec:
         containers:
            - name: mycustom-image-processing
              image: someorg/mycustom-image-
processing
   selector:
      matchLabels:
         tier: backend
```

Exercise 37 - Solution

Introduction: Create a service-definition.yml file from scratch. You are tasked to create a service to enable the frontend pods to access a backend set of pods

Instruction: Use the information provided in the table below to create a backend service definition file. Refer to the provided deployment-definition file for information regarding the PODs



Service Name: image-processing
labels: app=> myapp
type: ClusterIP
Port on the service: 80
Port exposed by image processing container: 8080

service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: image-processing
  labels:
    app: myapp
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 8080
  selector:
    tier: backend
```

Exercise

Thank You