Kubernetes Summary

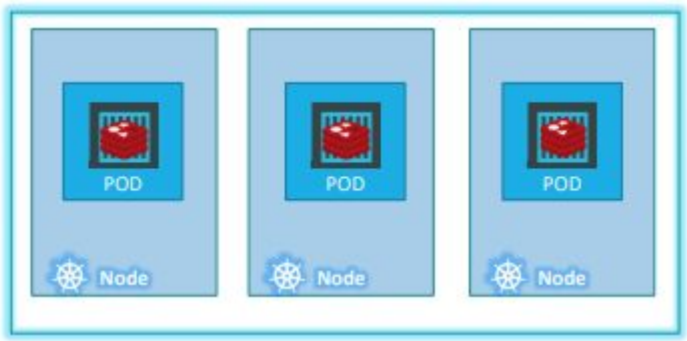
CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

- PODs
- ReplicaSets
- Deployment
- Namespaces
- Services
- Object Model
- Labels and Loose Coupling
- Ingress

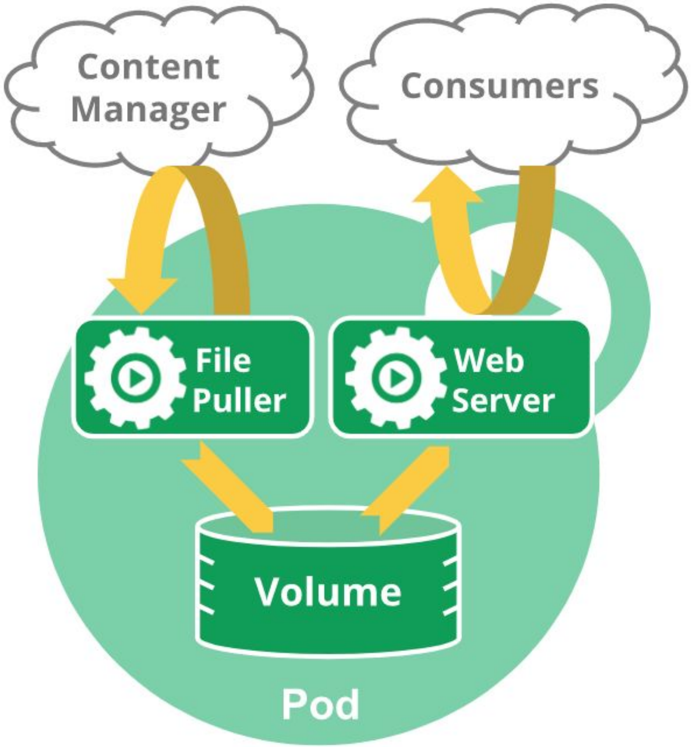
1

PODs

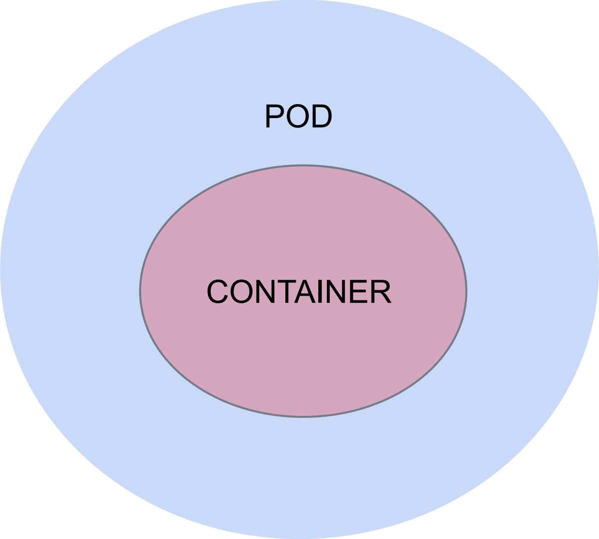


PODs

- The containers are encapsulated into a Kubernetes object known as PODs.
- A POD is a single instance of an application.
- A POD is the smallest object, that you can create in kubernetes.

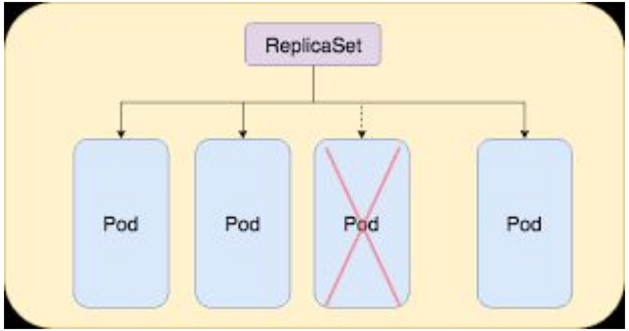


PODs



2

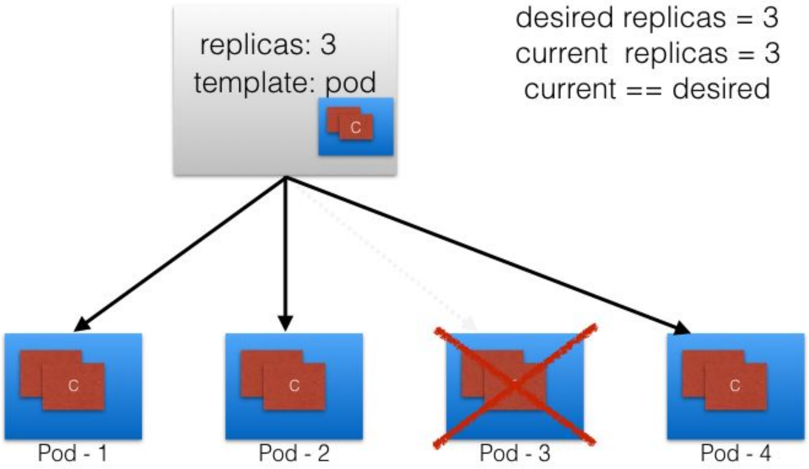
ReplicaSets



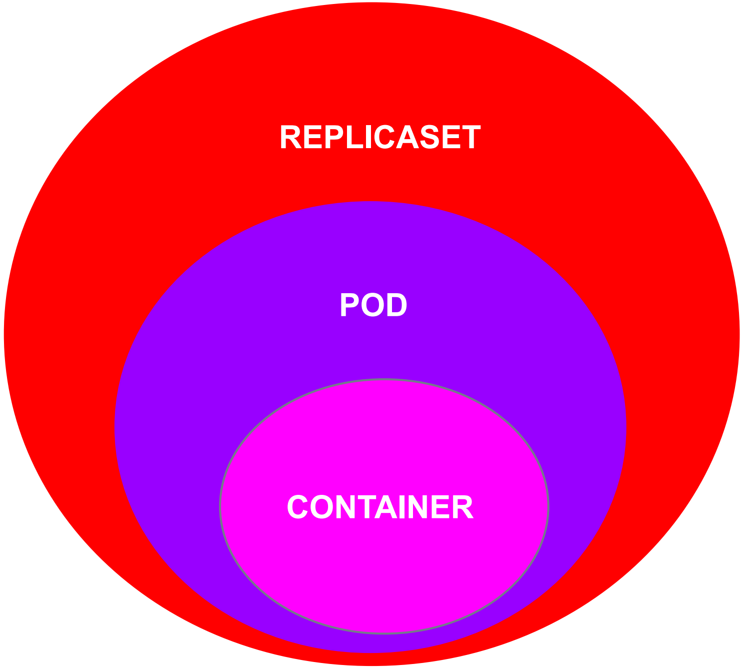
ReplicaSets

A **ReplicaSet's** purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

Replica Set

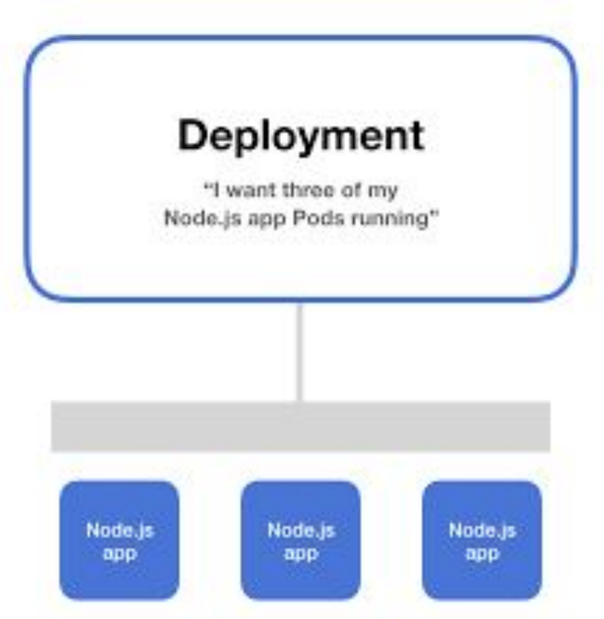


Replication Sets

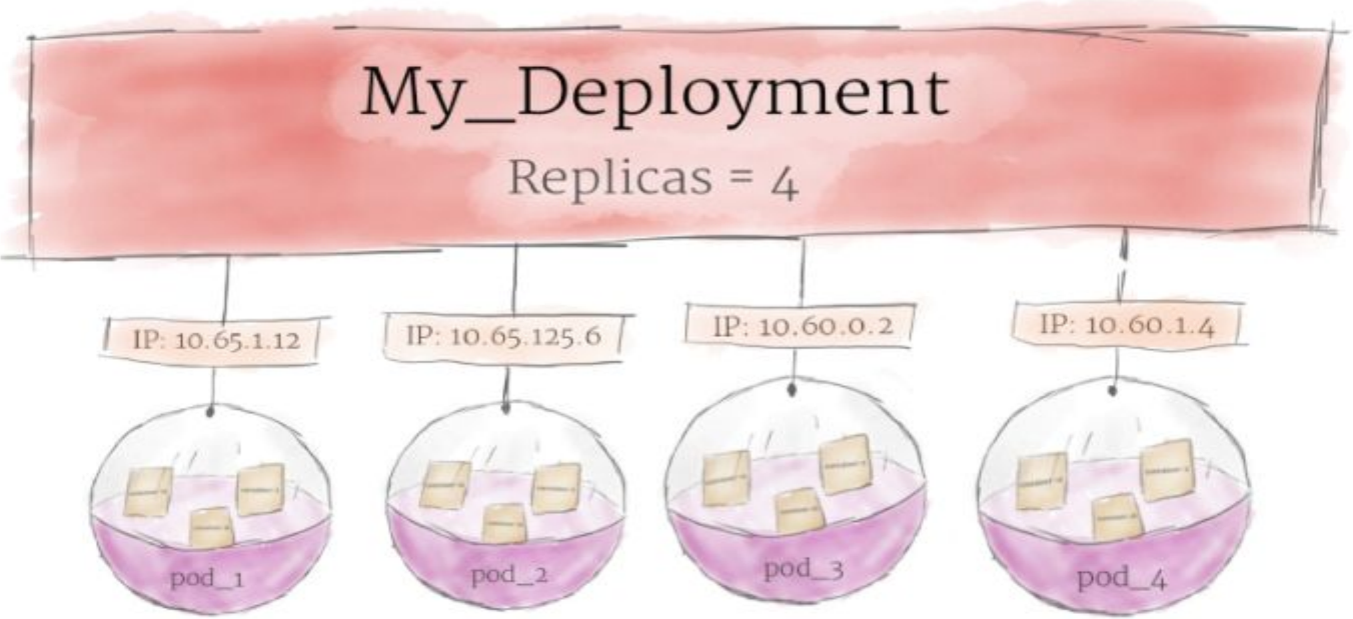


3

Deployment

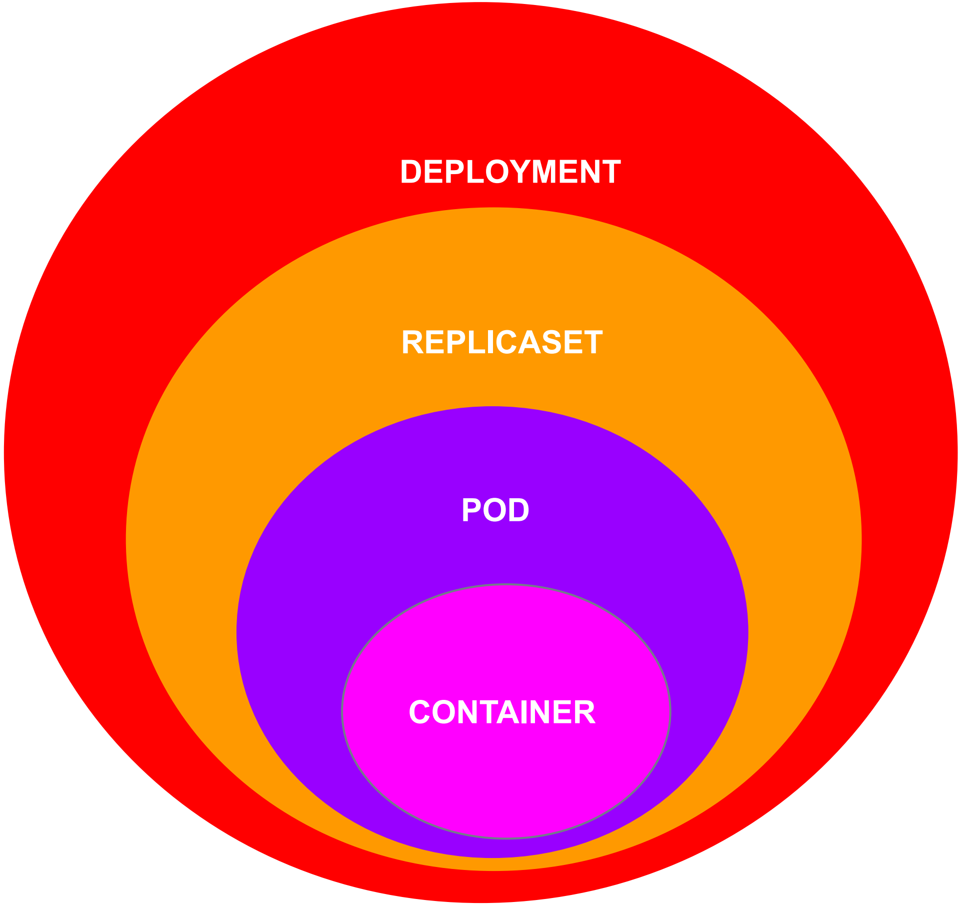


Deployment



Deployment is a method of converting images to containers and then allocating those images to pods in the Kubernetes cluster. A Deployment provides declarative updates for Pods and ReplicaSets.

Deployment

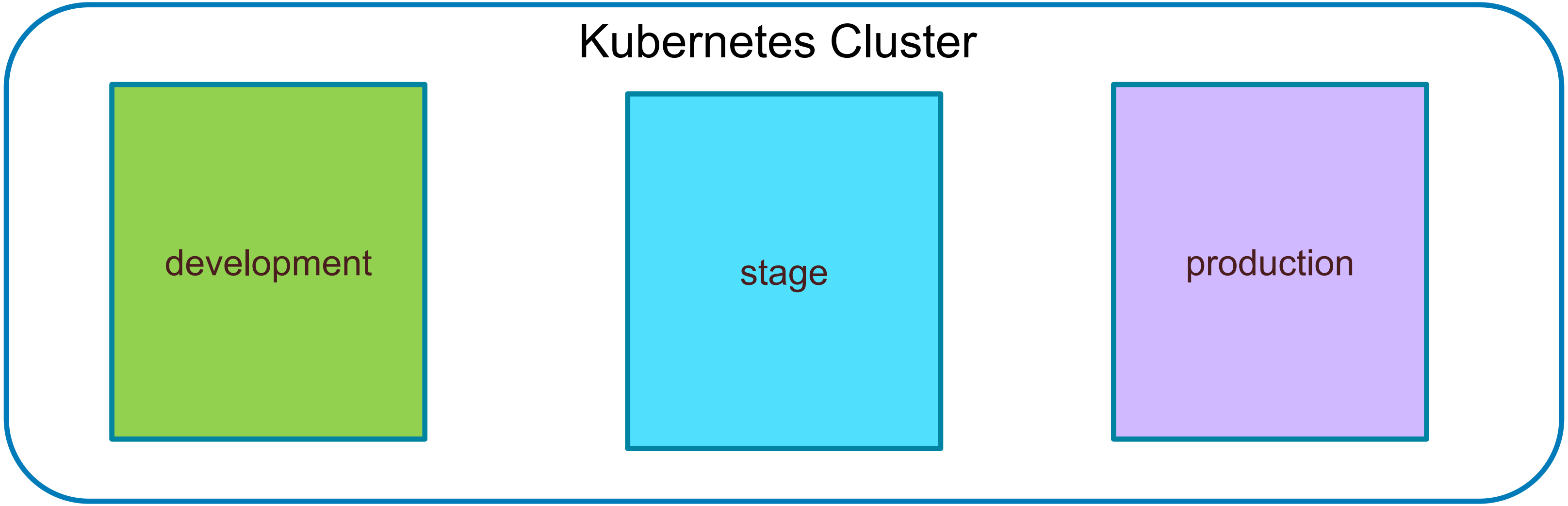


4

Namespaces

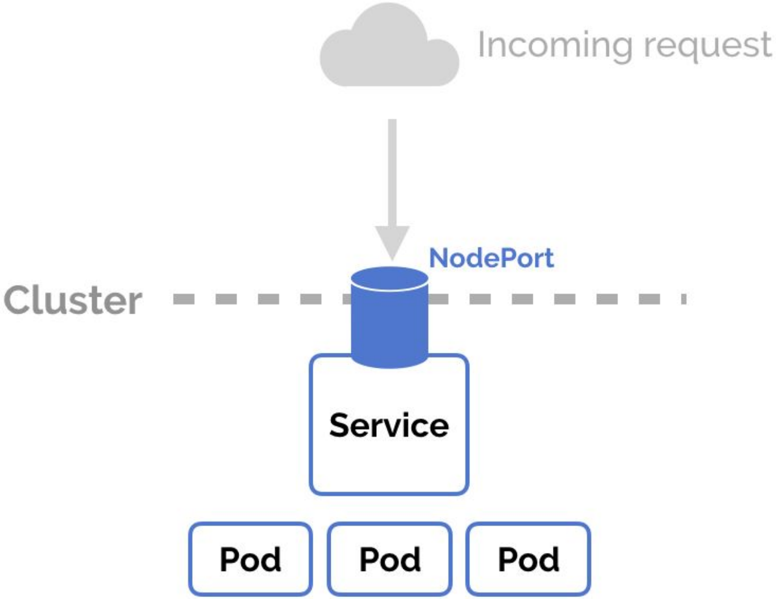
Namespaces

- Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called **namespaces**.
- The names of the resources/objects created inside a Namespace are unique, but not across Namespaces in the cluster.



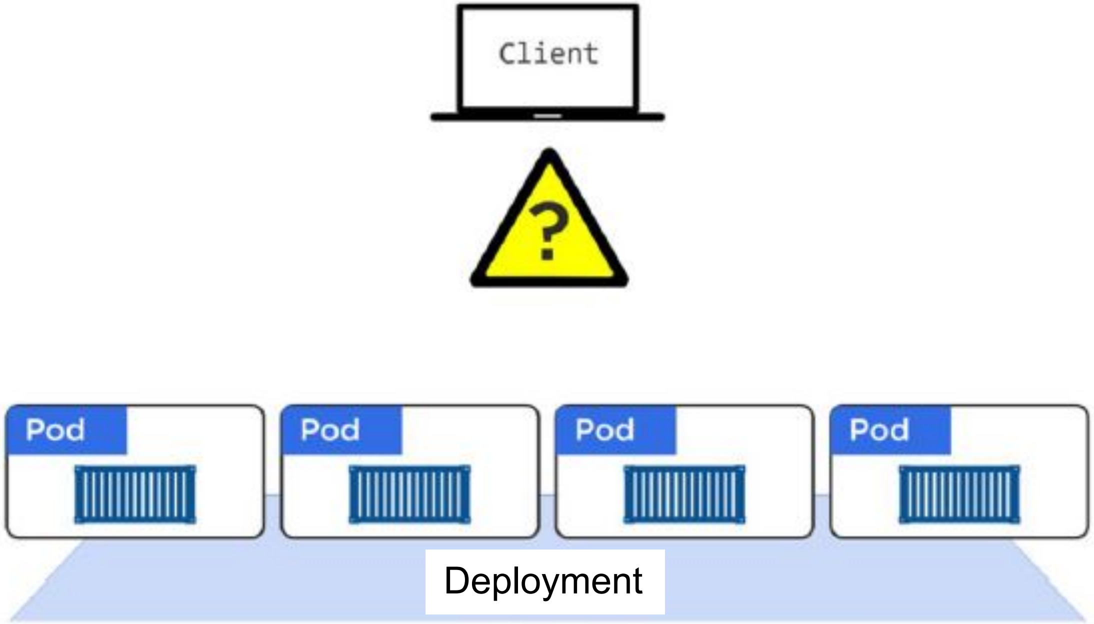
5

Services



Services

Each Kubernetes Pod gets its own IP address. But Kubernetes **Pods** are mortal. They are born and when they die, they are not resurrected. If you use a Deployment to run your app, it can create and destroy Pods dynamically. So, Pod IPs are unreliable.

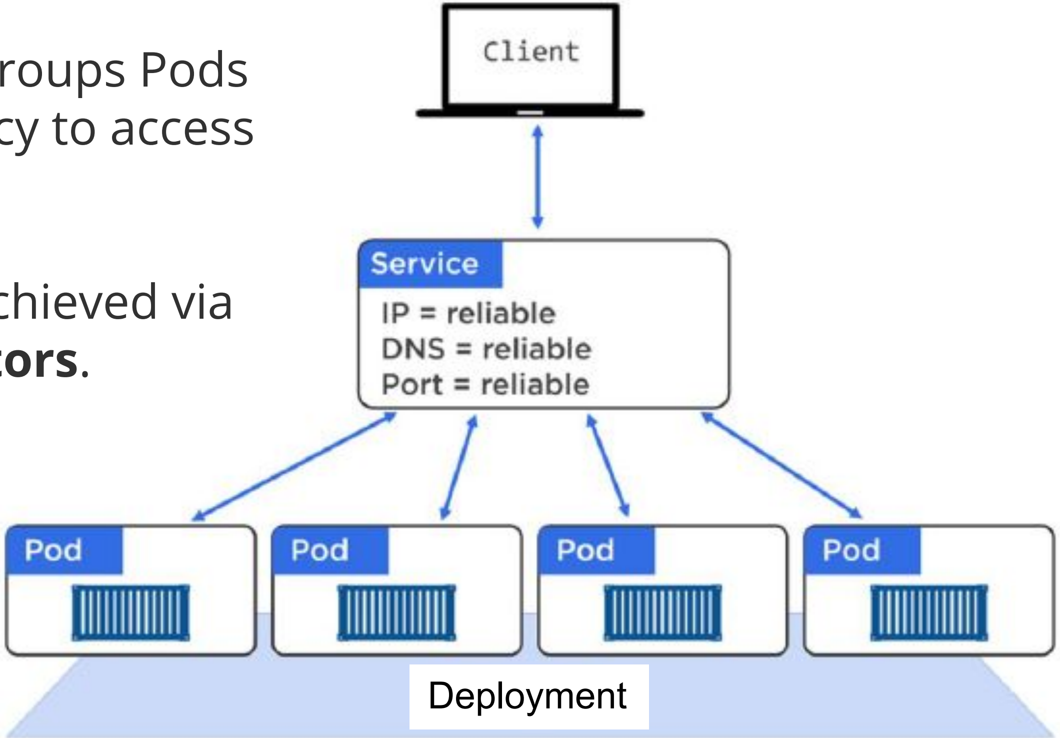


Services

The **Service** is associated with the Pods, and provides them with a stable IP, DNS and port.

Service logically groups Pods and defines a policy to access them.

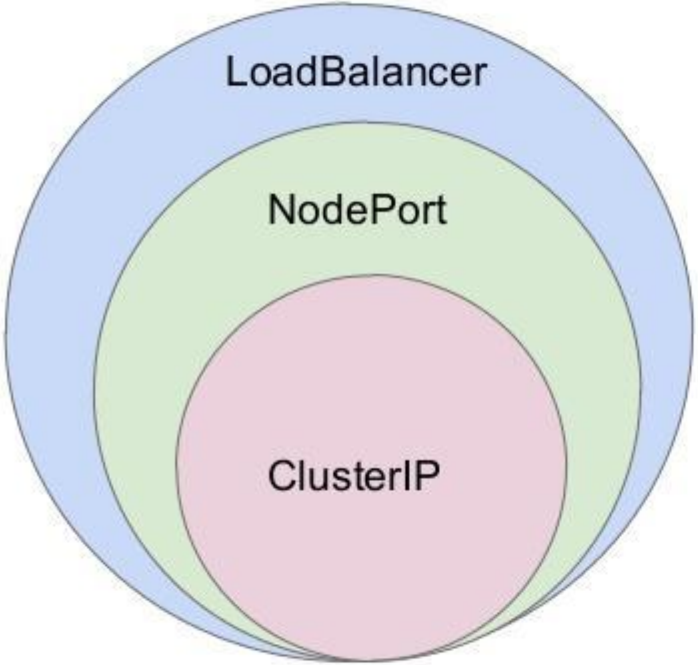
This grouping is achieved via **Labels** and **Selectors**.



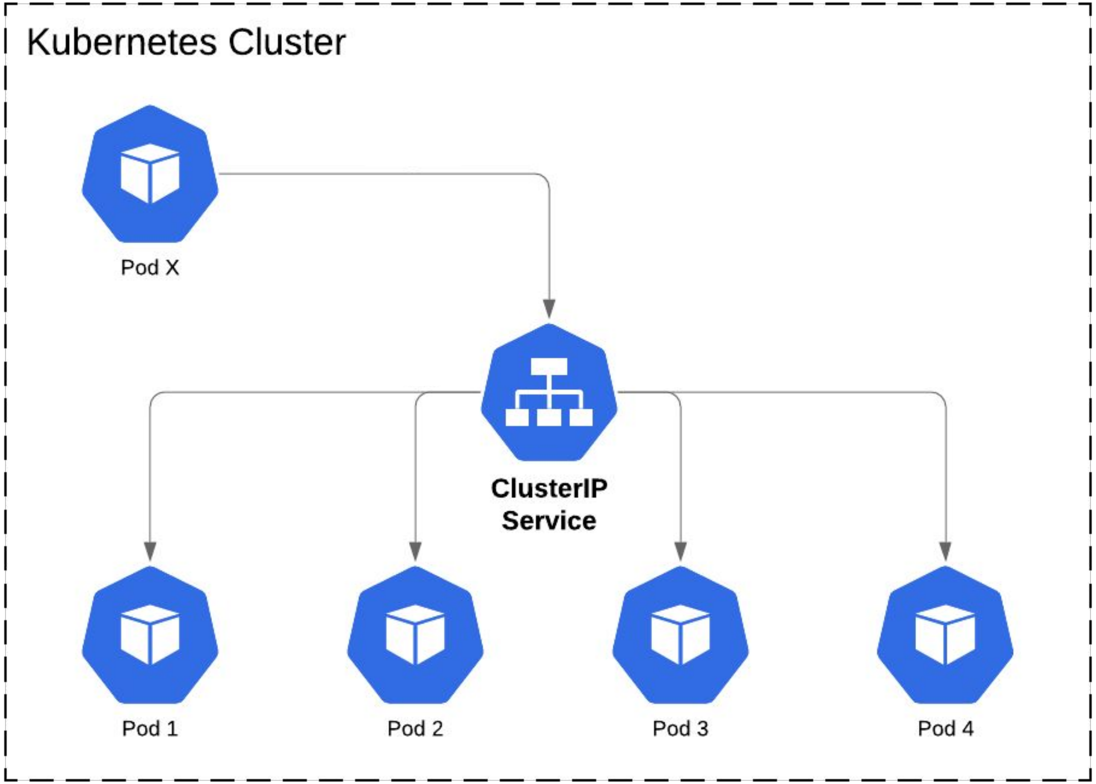
Service Types

There are 3 major service types:

- ClusterIP (default)
- NodePort
- LoadBalancer
- ExternalName



Service Types



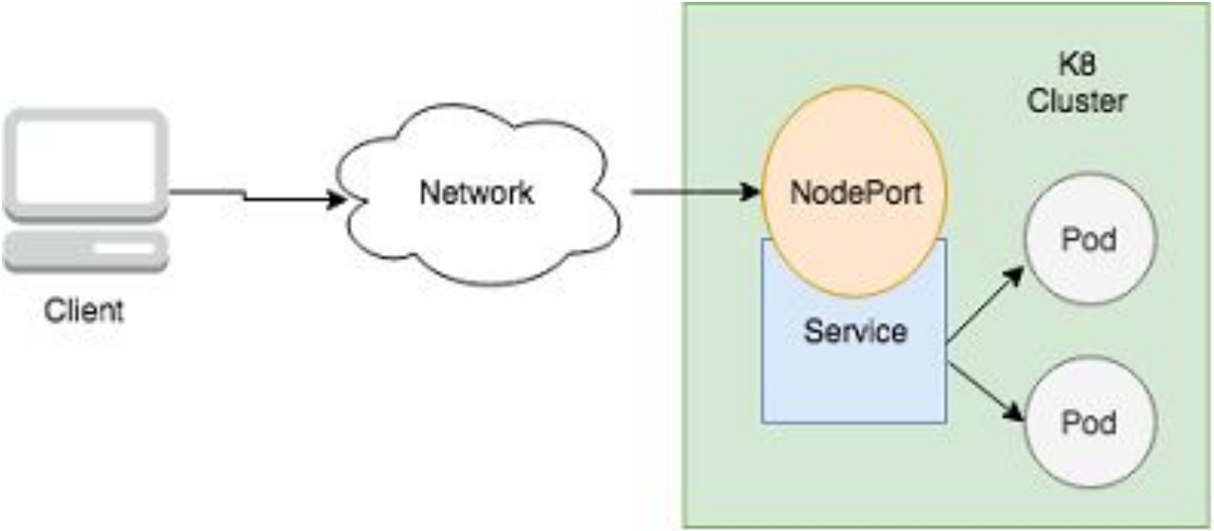
ClusterIP:
Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType.

Good for service of database & back-end apps.

Service Types

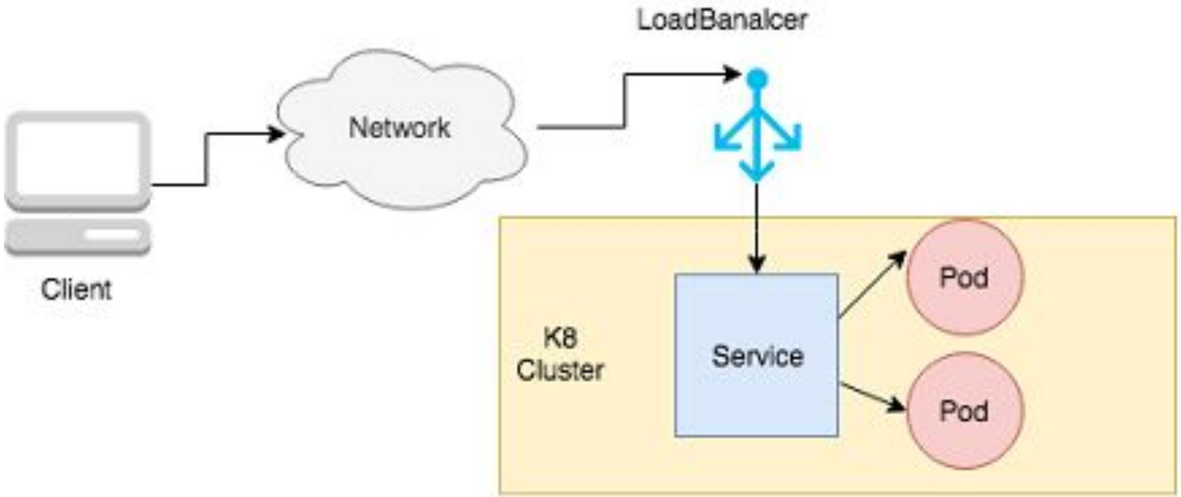
NodePort: Exposes the Service on each Node's IP at a static port (the NodePort). A ClusterIP Service, to which the NodePort Service **routes**, is automatically created. Port can either be **statically** defined, or **dynamically** taken from a range between 30000-32767.

With the **NodePort** *ServiceType*, in addition to a ClusterIP, a high-port is mapped to the respective Service, from all the worker nodes.

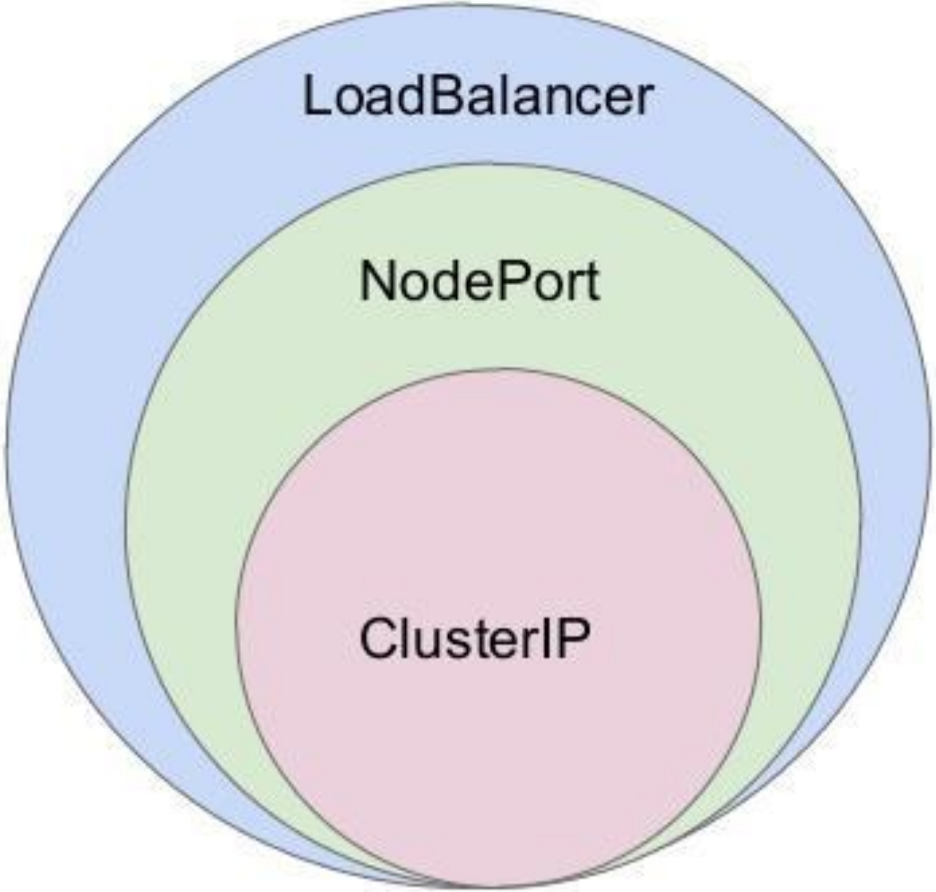


Service Types

LoadBalancer: Exposes the Service externally using a cloud provider's load balancer. The external load balancer routes to the automatically created NodePort and ClusterIP Services.



Service Types



6

Object Model

Object Model

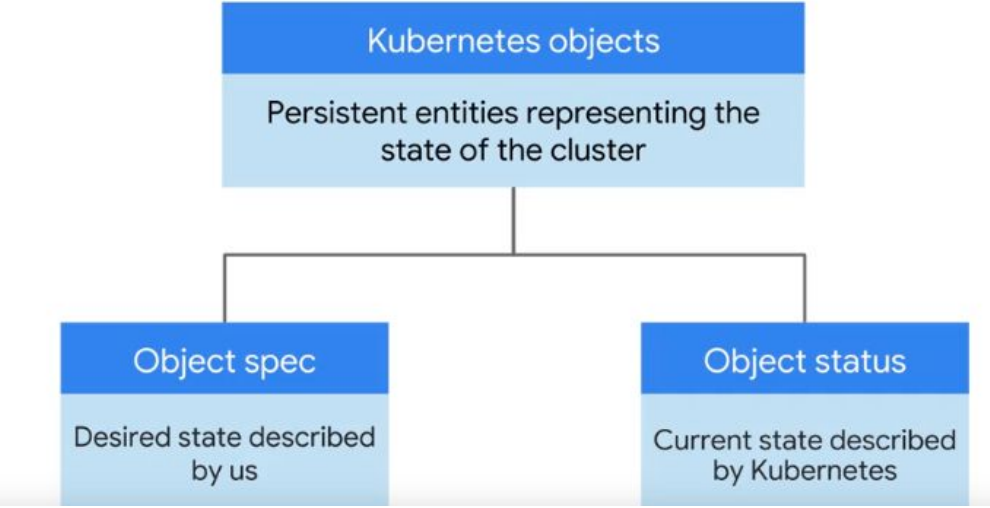
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

All objects must have **apiVersion, kind, metadata** and **spec** fields.

- **apiVersion:** Which version of the Kubernetes API you're using to create this object
- **kind:** What kind of object you want to create
- **metadata:** Data that helps uniquely identify the object, including a **name** string, **UID**, and optional **namespace**
- **spec:** What state you desire for the object

Object Model

- Once the Deployment object is created, the Kubernetes system attaches the **status** field to the object.
- **status** is managed by Kubernetes and describes the **actual state** of the object and its history.



Kubernetes objects

Persistent entities representing the state of the cluster

Object spec

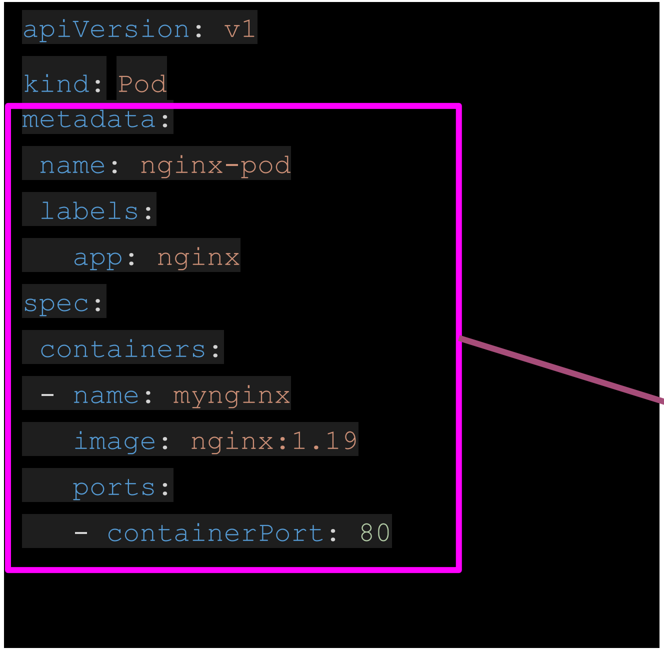
Desired state described by us

Object status

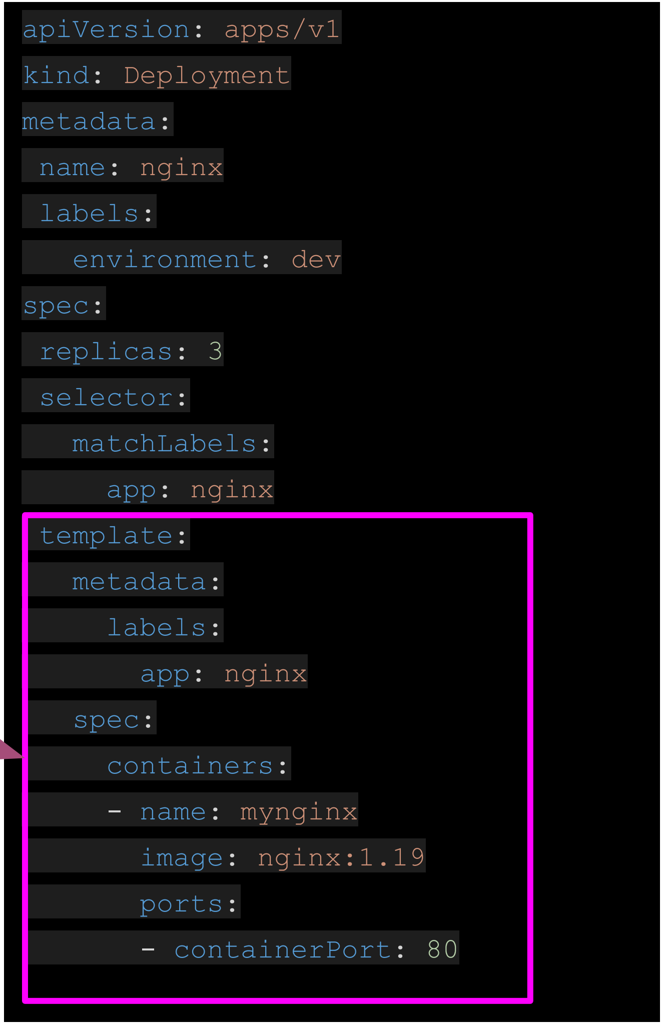
Current state described by Kubernetes

Object Model

Pod to Deployment



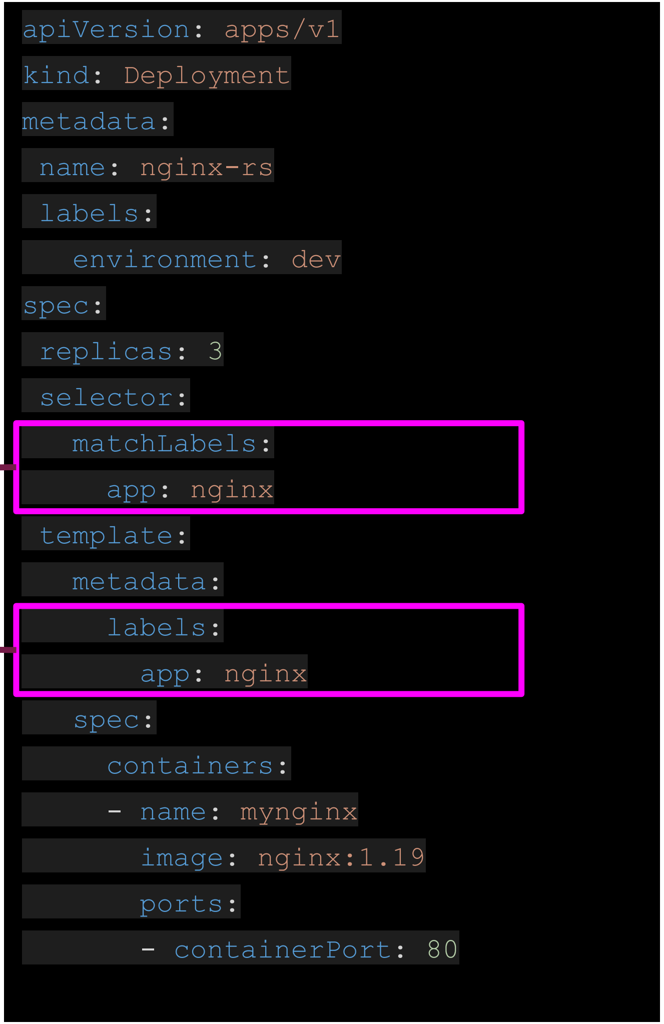
```yaml
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
 labels:
   app: nginx
spec:
 containers:
 - name: mynginx
   image: nginx:1.19
   ports:
   - containerPort: 80
```



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
 labels:
   environment: dev
spec:
 replicas: 3
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: mynginx
       image: nginx:1.19
       ports:
       - containerPort: 80
```

Object Model

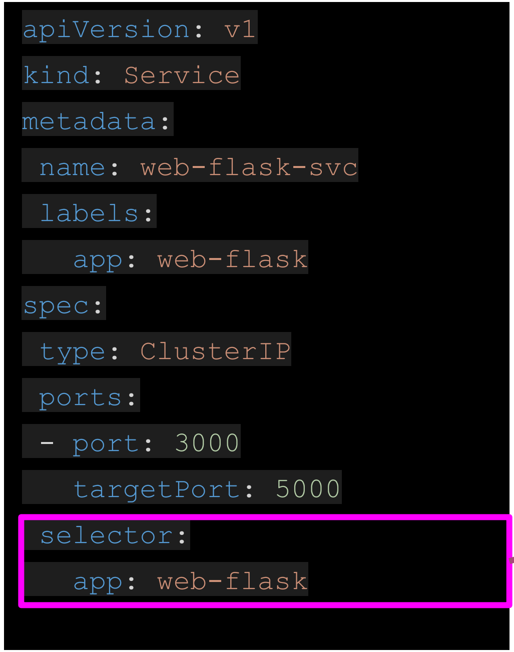
Pod Selector



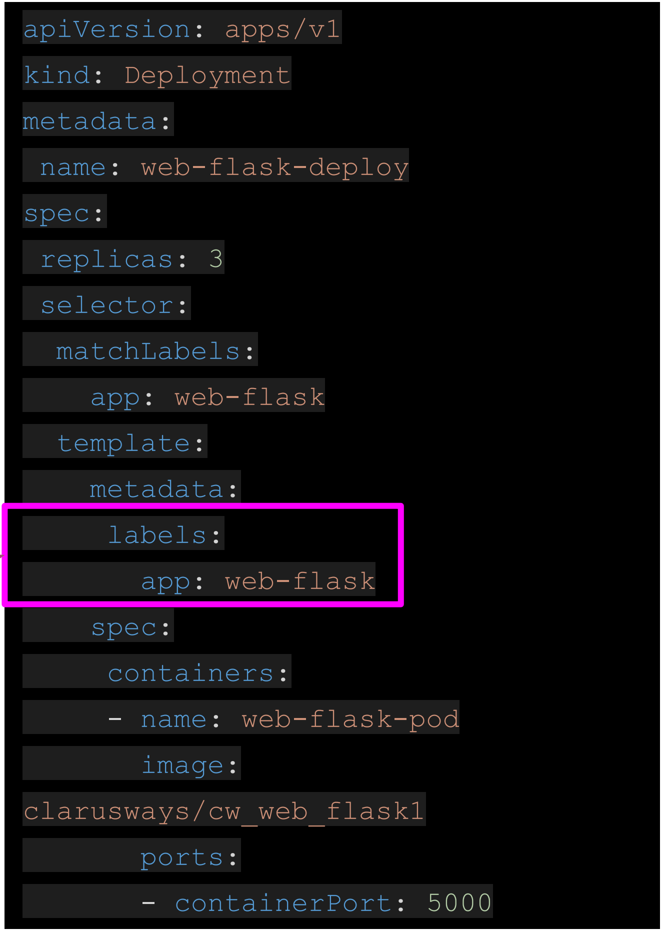
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-rs
 labels:
   environment: dev
spec:
 replicas: 3
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: mynginx
       image: nginx:1.19
       ports:
       - containerPort: 80
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

Object Model



```
apiVersion: v1
kind: Service
metadata:
 name: web-flask-svc
 labels:
   app: web-flask
spec:
 type: ClusterIP
 ports:
 - port: 3000
   targetPort: 5000
 selector:
   app: web-flask
```



```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: web-flask-deploy
spec:
 replicas: 3
 selector:
  matchLabels:
    app: web-flask
  template:
    metadata:
     labels:
       app: web-flask
    spec:
     containers:
     - name: web-flask-pod
       image:
clarusways/cw_web_flask1
       ports:
       - containerPort: 5000
```

7

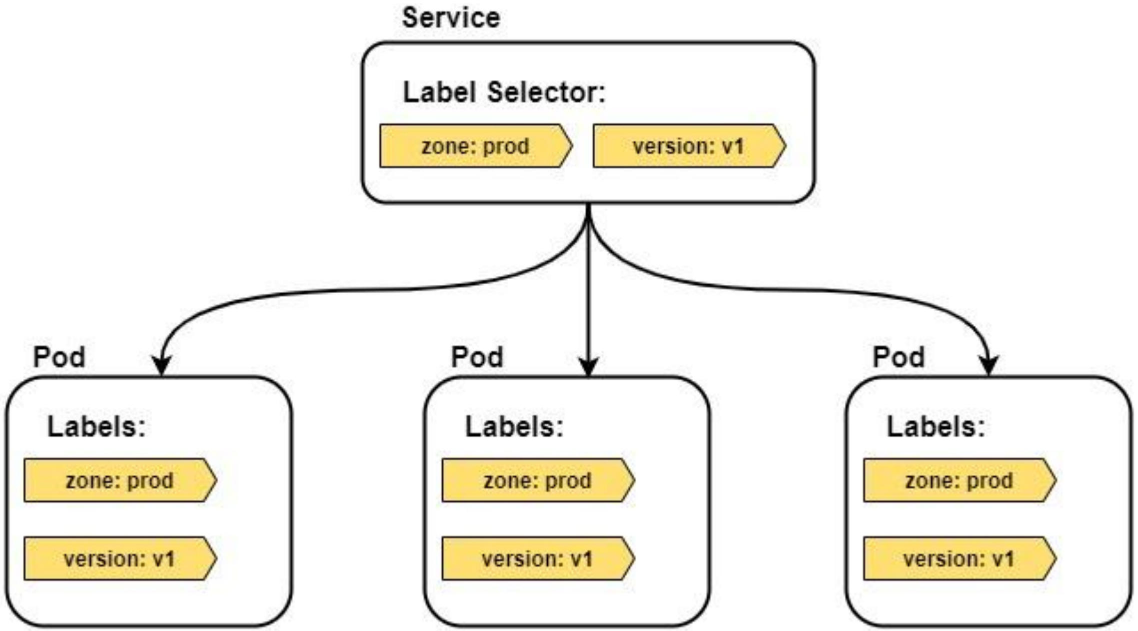
Labels and loose coupling

Labels and loose coupling

- Labels and Selectors use a **key/value** pair format.
- Pods and Services are loosely coupled via labels and label selectors.
- For a Service to match a set of Pods, and therefore provide stable networking and load-balance, it only needs to match some of the Pods labels.
- However, for a Pod to match a Service, the Pod must match all of the values in the Service's label selector.

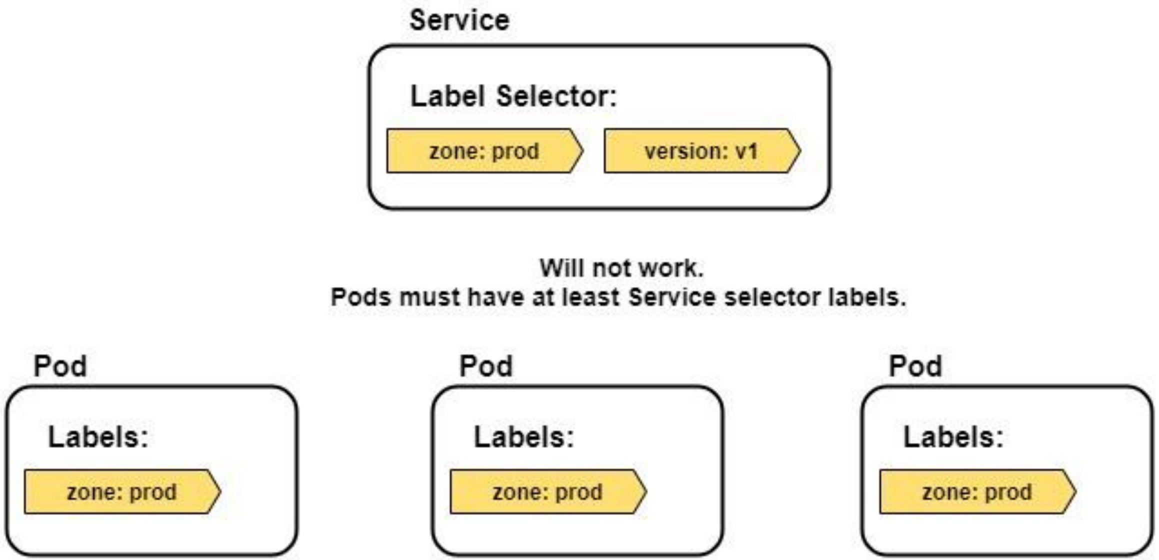
Labels and loose coupling

The figure below shows an example where 3 Pods are labelled as zone=prod and version=1, and the Service has a label selector that matches. This Service provides stable networking to all three Pods. It also provides simple load-balancing.



Labels and loose coupling

The figure below shows an example where the Service does not match any of the Pods. This is because the Service is selecting on two labels, but the Pods only have one of them. The logic behind this is a Boolean AND operation.



Service

Label Selector:

zone: prod

version: v1

Will not work.
Pods must have at least Service selector labels.

Pod

Labels:

zone: prod

Pod

Labels:

zone: prod

Pod

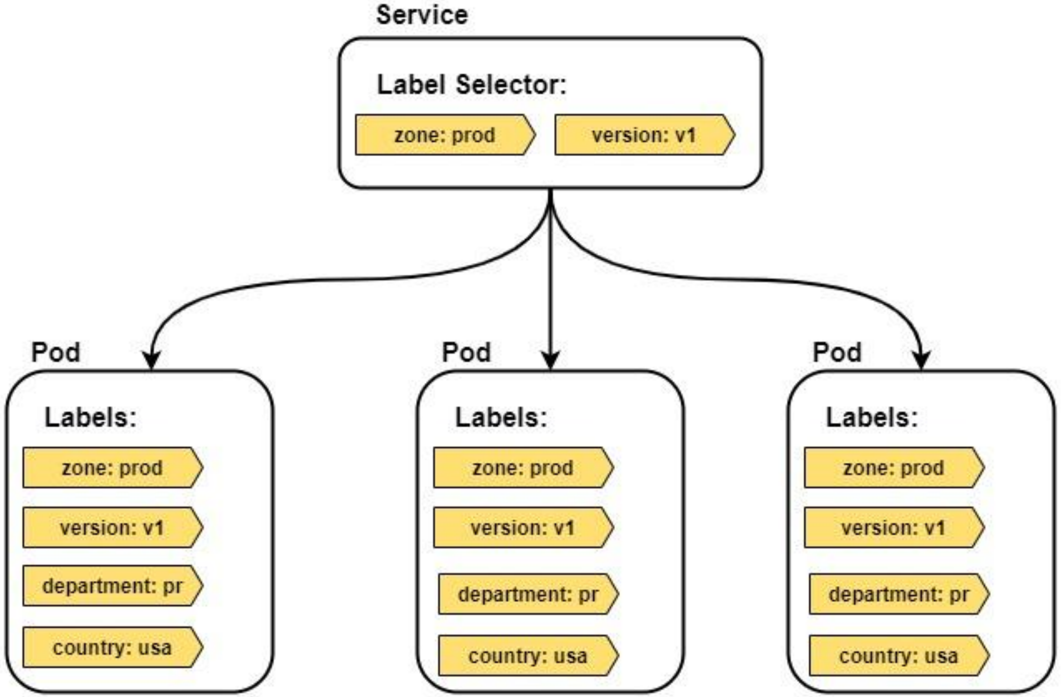
Labels:

zone: prod

CLARUSWAY©
WAY TO REINVENT YOURSELF

Labels and loose coupling

This figure shows an example that does work. It doesn’t matter that the Pods have additional labels that the Service is not selecting on.

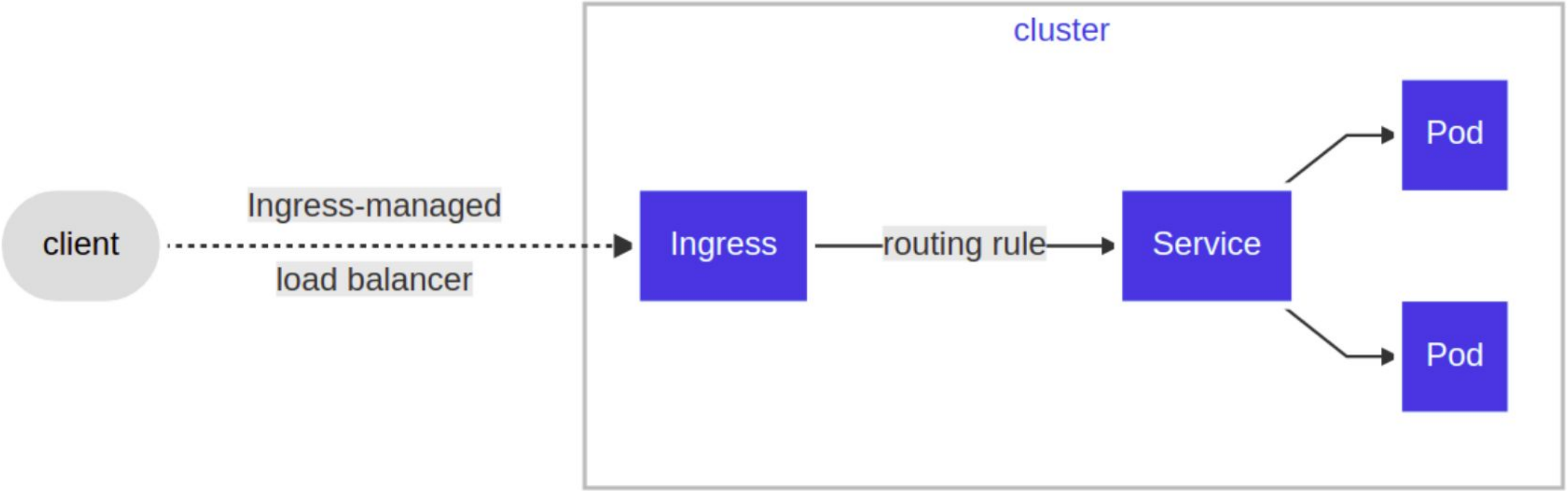


8

Ingress

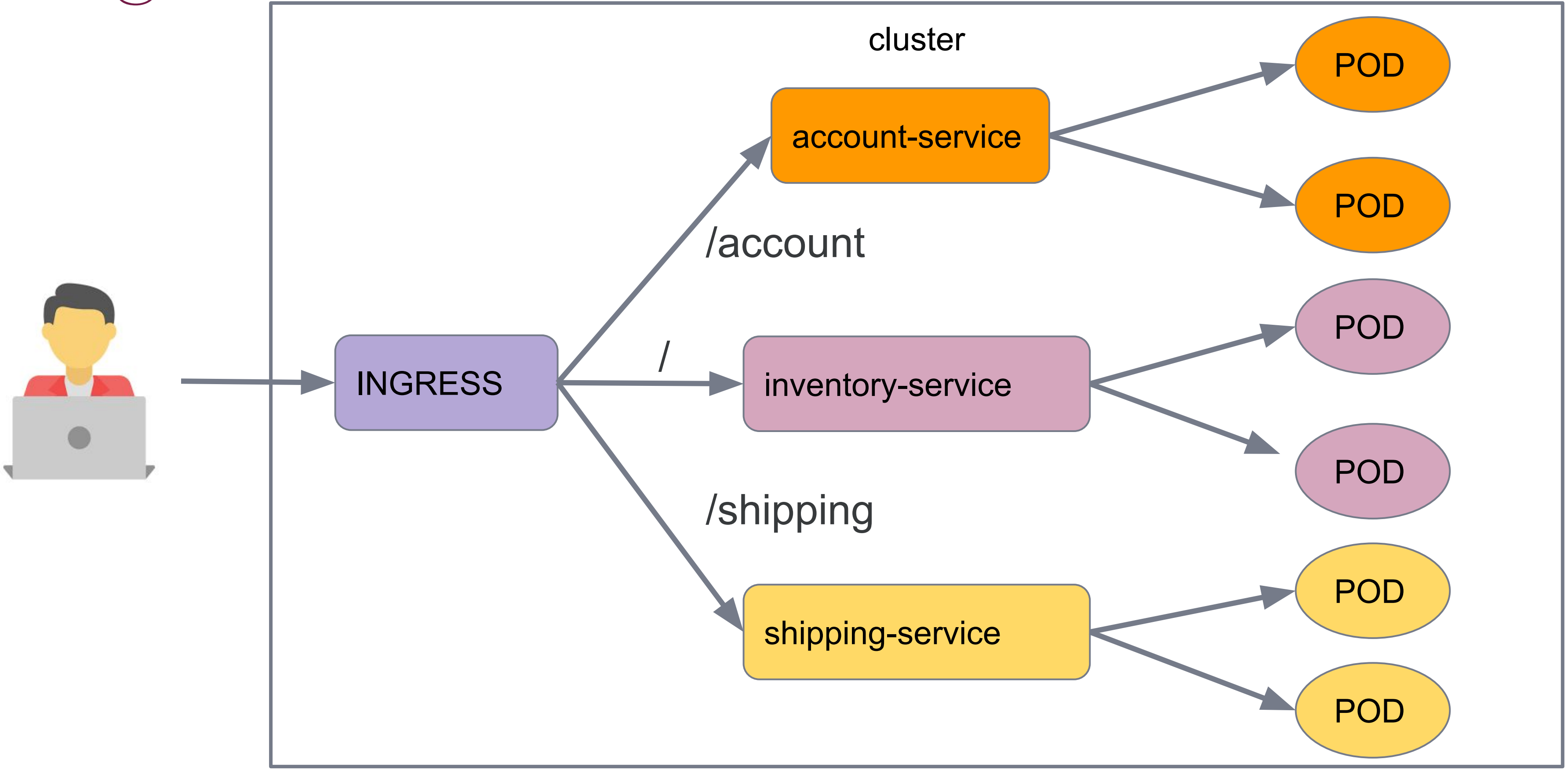
Ingress

"An Ingress is a collection of rules that allow inbound connections to reach the cluster Services."



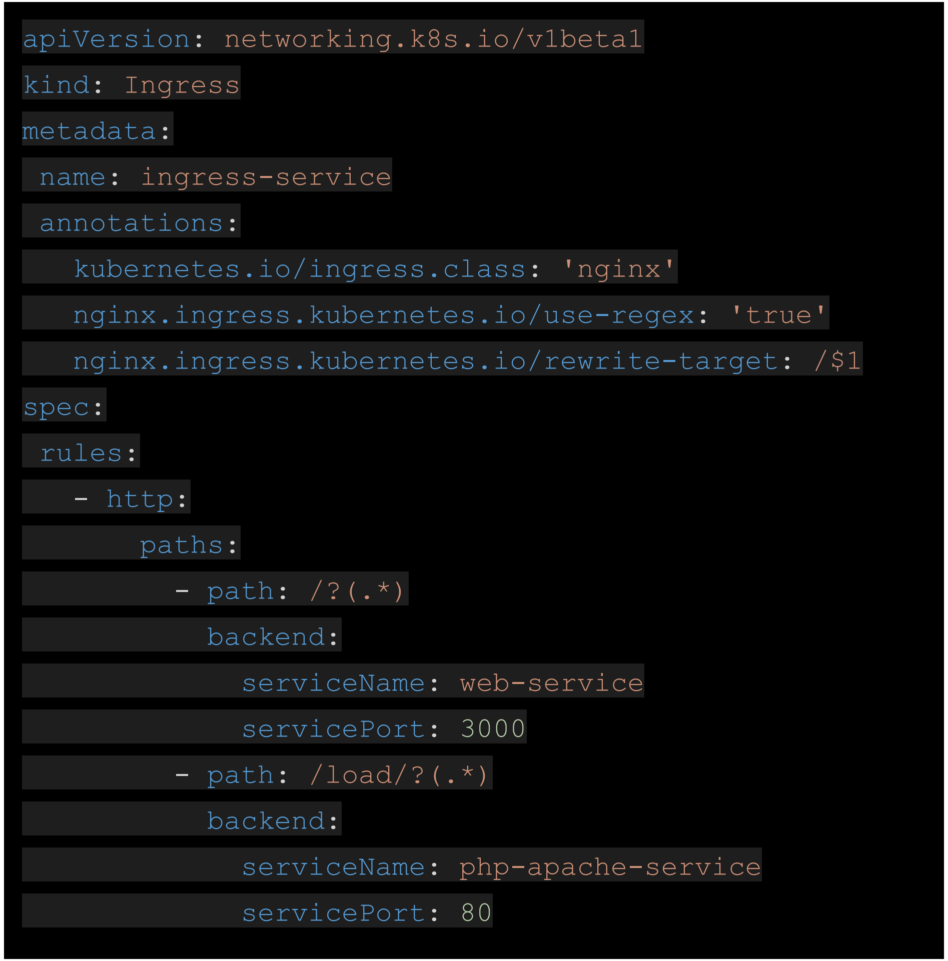
Ingress

www.clarus-commerce.com



Ingress

With Ingress, users do not connect directly to a Service. Users reach the Ingress endpoint, and, from there, the request is forwarded to the desired Service.



```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
 name: ingress-service
 annotations:
   kubernetes.io/ingress.class: 'nginx'
   nginx.ingress.kubernetes.io/use-regex: 'true'
   nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
 rules:
   - http:
       paths:
         - path: /?(.*)
           backend:
             serviceName: web-service
             servicePort: 3000
         - path: /load/?(.*)
           backend:
             serviceName: php-apache-service
             servicePort: 80
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

THANKS!

Any questions?

You can find me at:

- Joe@clarusway.com