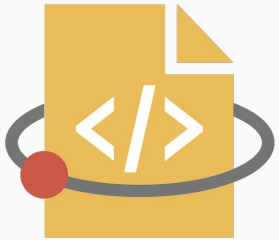
Understanding Kubernetes Core Concepts

Iqbal Syamil - Telkom University

Muhammad Iqbal Syamil Ayasy

- Sophomore student of Informatics @Telkom University
- Co-organizer @Kubernetes and Cloud Native Bandung
- Pengisi suara @dotMap(podcast)

CLOUD NATIVE
COMPUTING FOUNDATION



Season of Docs

https://github.com/cncf/mentoring

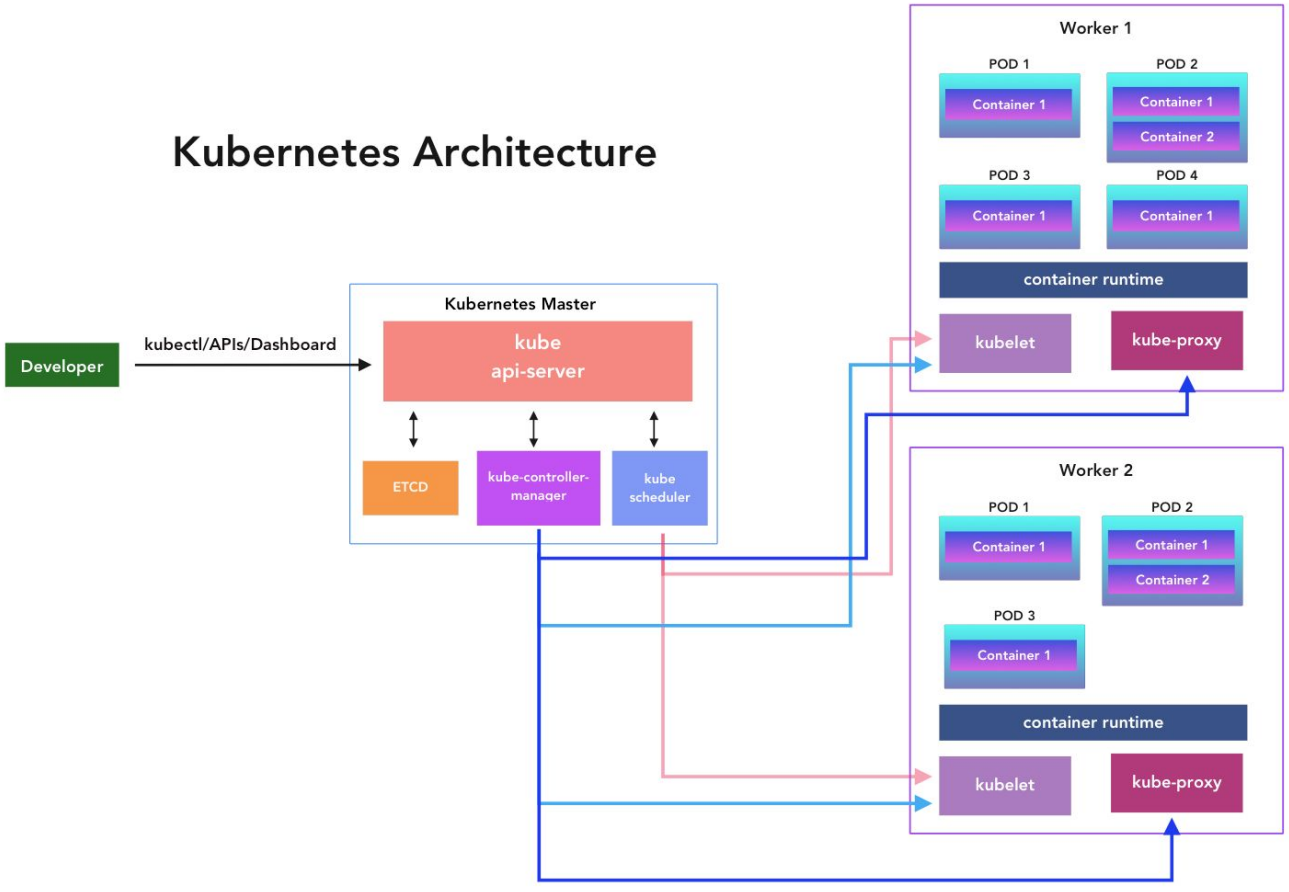
Outline

- Kubernetes Architecture
- Reconciliation
- Kubernetes object
    - Pods
    - Replica set
    - Service
    - Deployment
        - Deployment strategy
- Demo

Kubernetes

Unsplash/johantaljaard

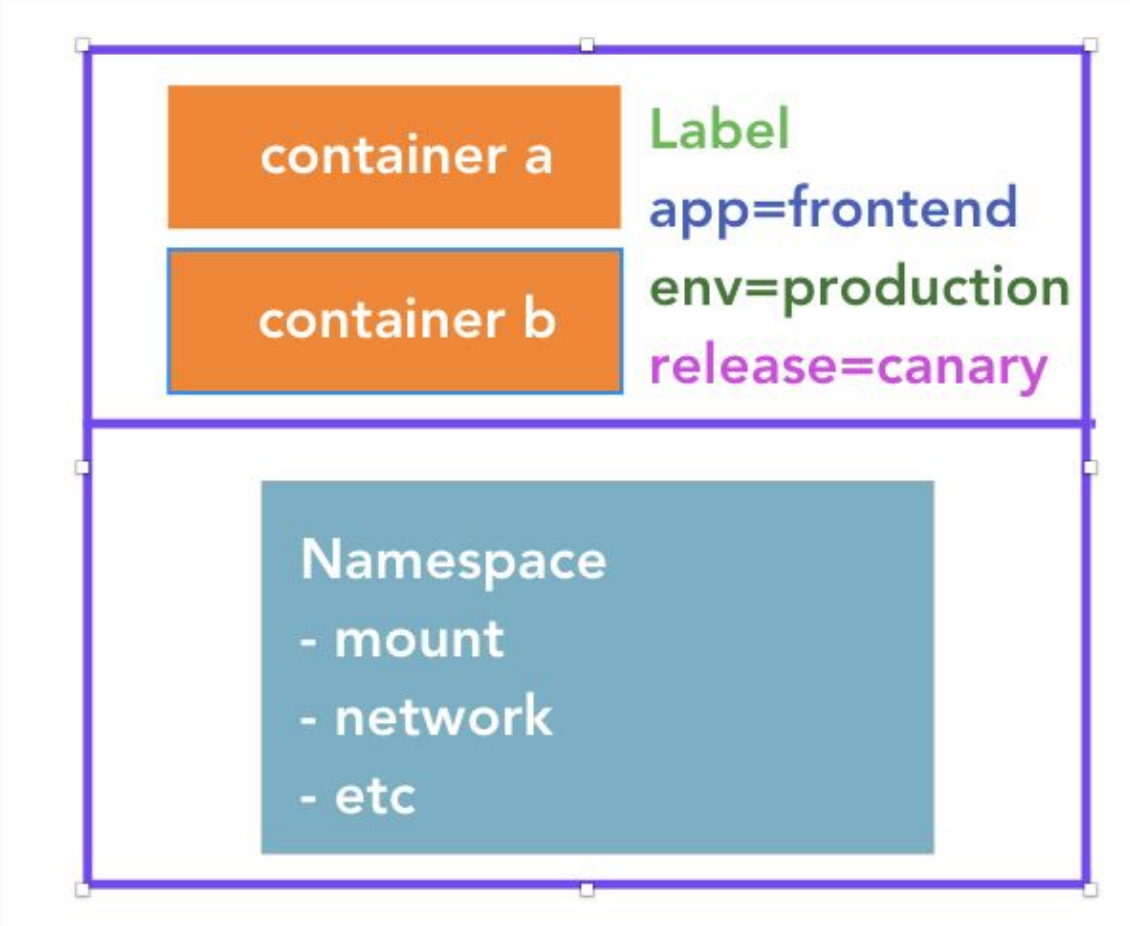
Kubernetes Architecture

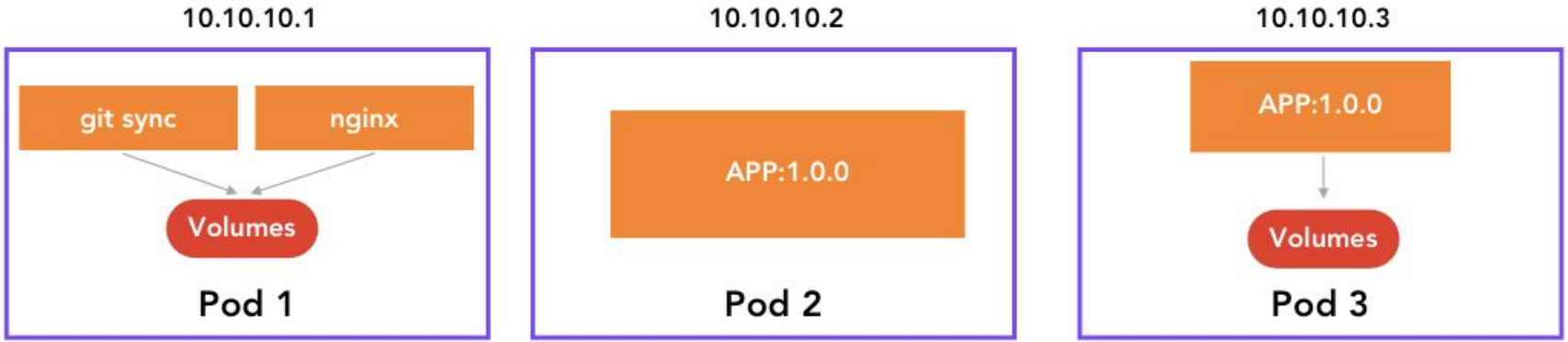


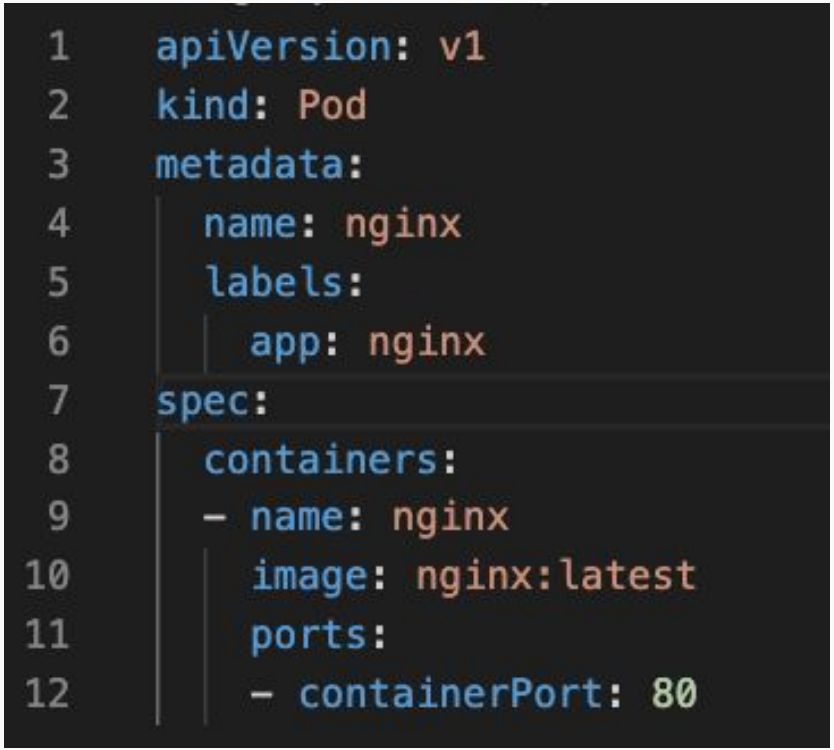
Kubernetes Objects

Pods

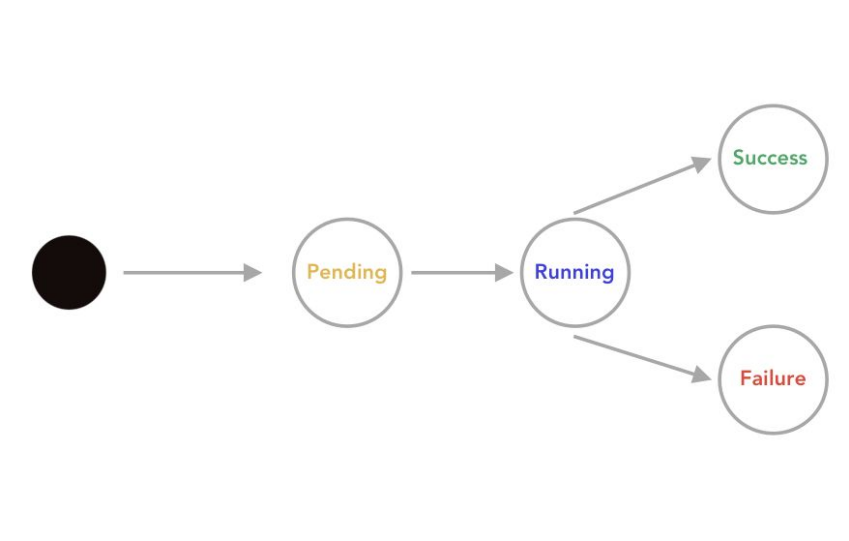
Pod is an Atomic unit on the Kubernetes platform.




10.10.10.1
git sync
nginx
Volumes
Pod 1
10.10.10.2
APP:1.0.0
Pod 2
10.10.10.3
APP:1.0.0
Volumes
Pod 3



Kubernetes Pod Example YAML

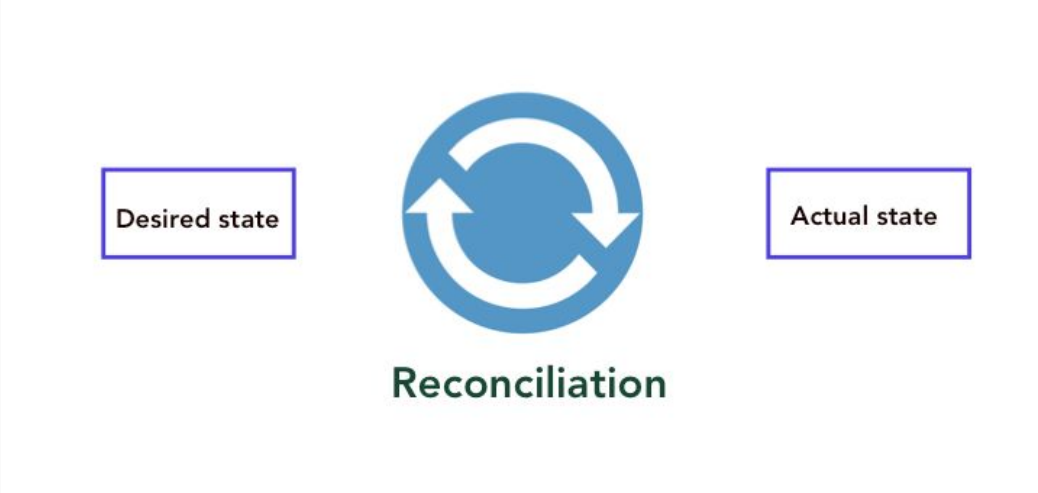


Pods Lifecycle

- Pending
- Running
- Succeeded
- Failled
- Unknown

Reconciliation

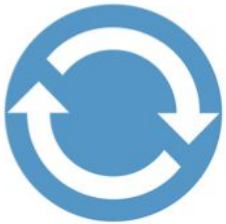
Reconciliation - the action of making one view or belief compatible with another.



https://dev.to/adipolak/kubernetes-self-healing-reconciliation-loop-4aj5

Creating a pod

k8s master



Reconciliation

worker

Creating a pod

kind: pod
name: foo
type: oval
color: red

kubectl create -f oval.yaml

k8s master

Reconciliation

worker

Creating a pod

make pod “foo”
- color=red
- shape= oval

k8s master

worker

Reconciliation

Creating a pod

k8s master

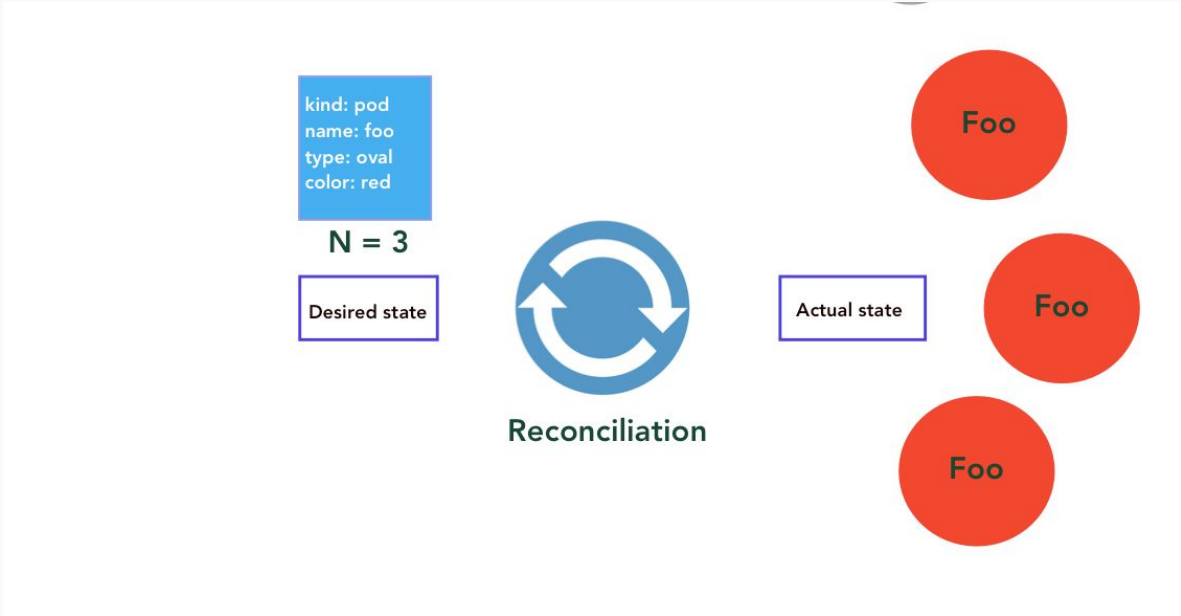
Reconciliation

worker

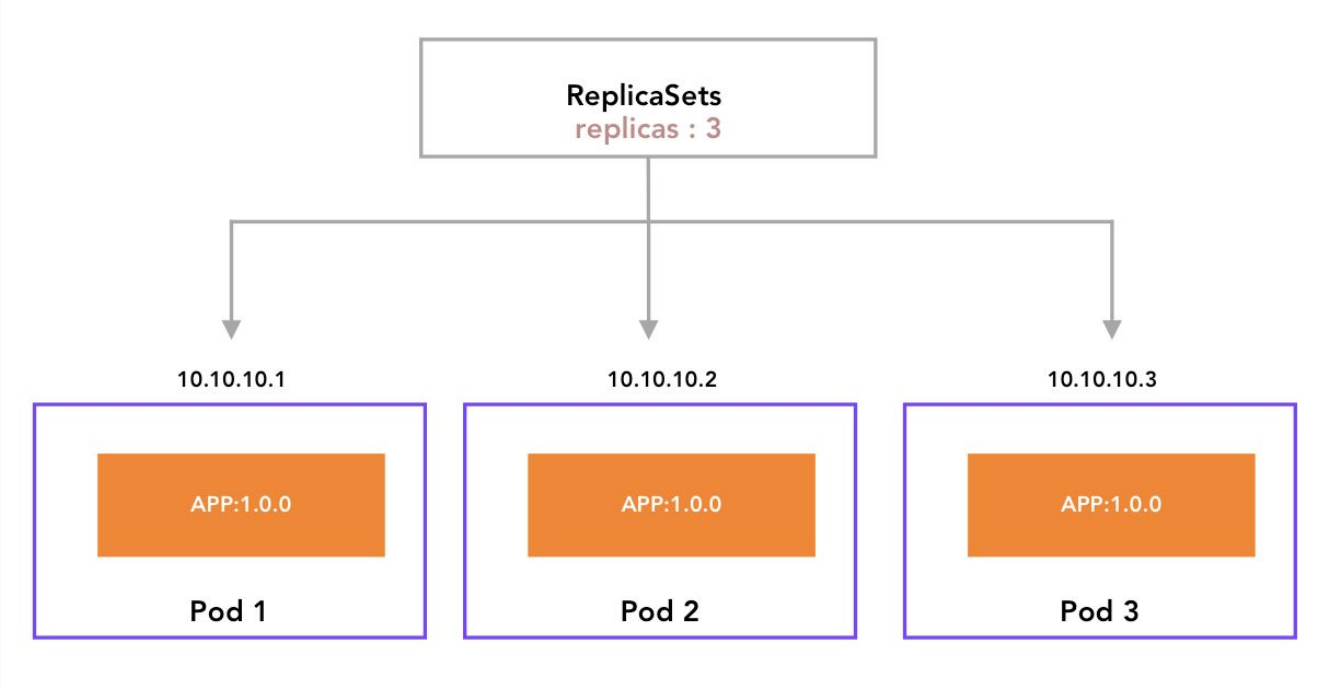
Foo

Replica Set

Replica Set ensures how many **replica of pod** should be running



Example ReplicaSets

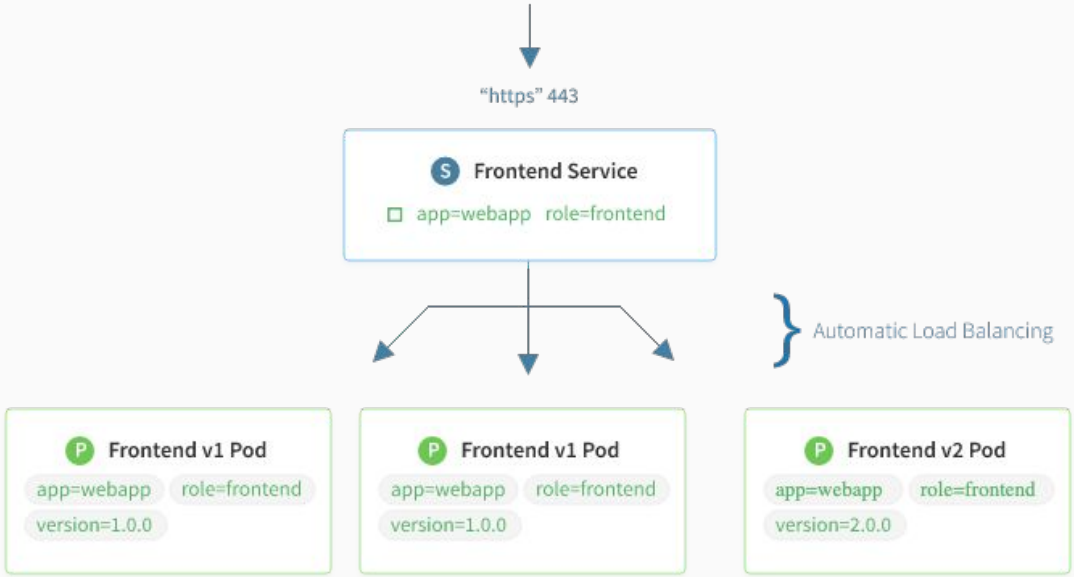


```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
  labels:
    app: nginx
    visualize: "true"
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        visualize: "true"
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Kubernetes ReplicaSet Example YAML

Service

A Kubernetes Service is an abstraction layer which defines a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those Pods



Kubernetes Service Example YAML

```yaml
kind: Service
apiVersion: v1

metadata:
  name: hostname-service

spec:
  type: NodePort
  selector:
    app: echo-hostname
  ports:
    - nodePort: 30163
      port: 8080
      targetPort: 80
```

Make the service available to network requests from external clients

Forward requests to pods with label of this value

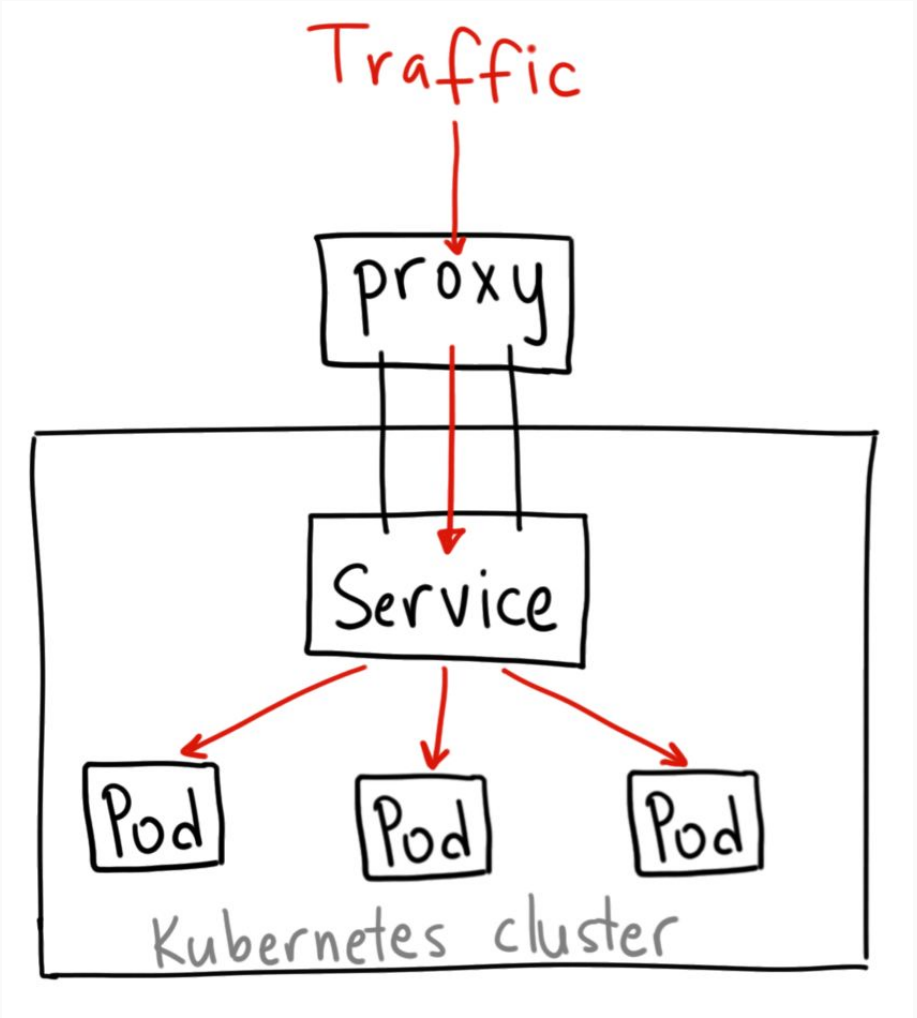
`nodePort`
access service via this external port number

`port`
port number exposed internally in cluster

`targetPort`
port that containers are listening on

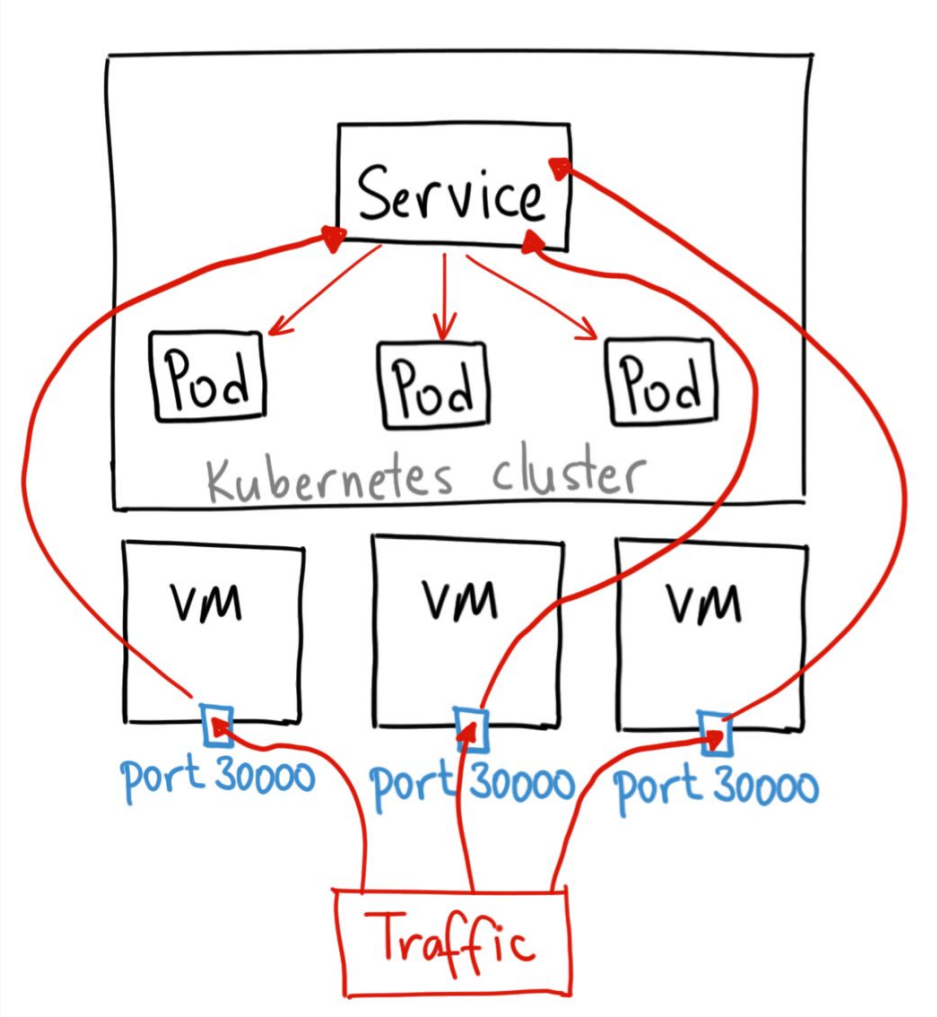
type:ClusterIP

A CluterIP Exposes the Service on an internal IP in the cluster



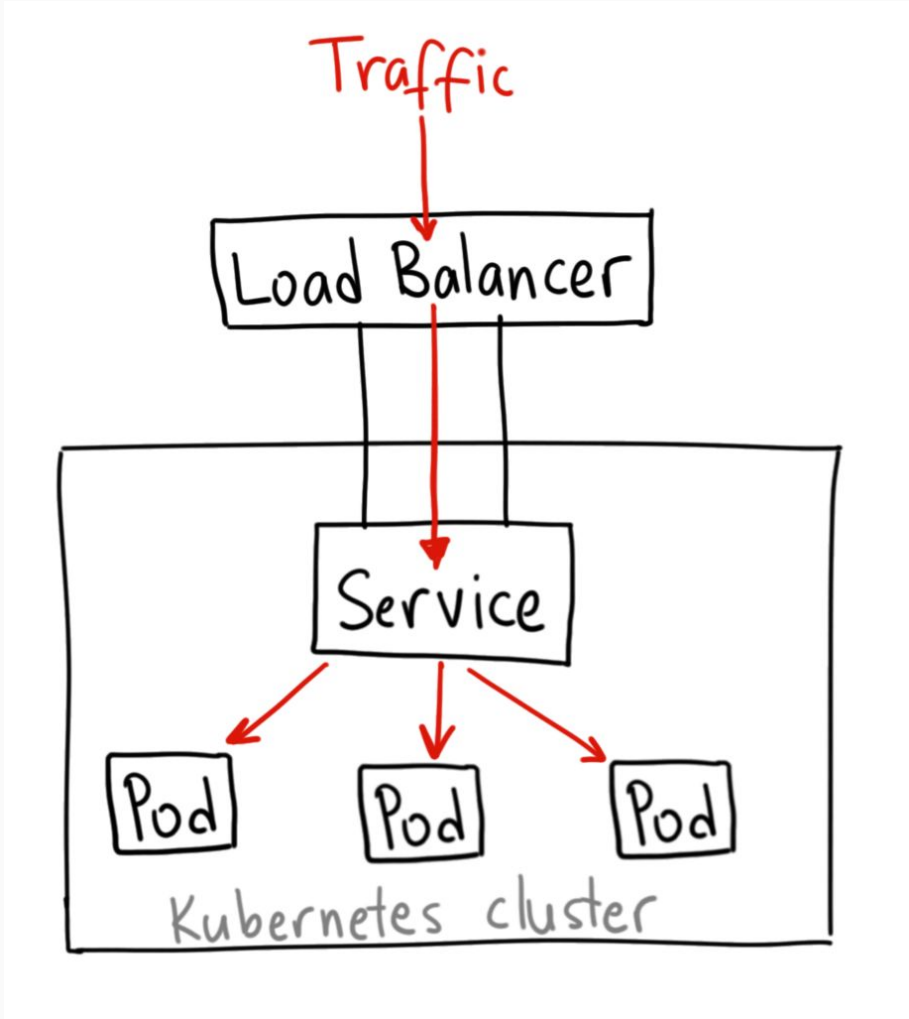
type:NodePort

A NodePort service opens a specific port on all the Nodes (the VMs), and any traffic that is sent to this port is forwarded to the service.



type:LoadBalancer

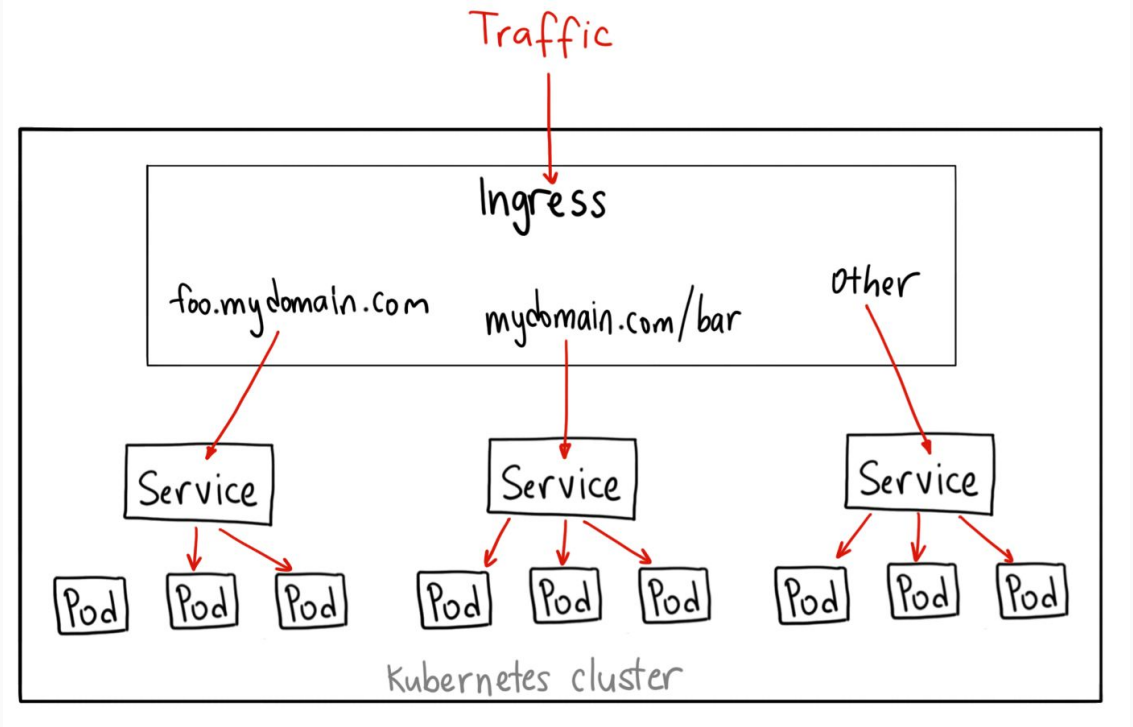
The service becomes accessible externally through a cloud provider's load balancer functionality. And All traffic on the port you specify will be forwarded to the service.



Ingress

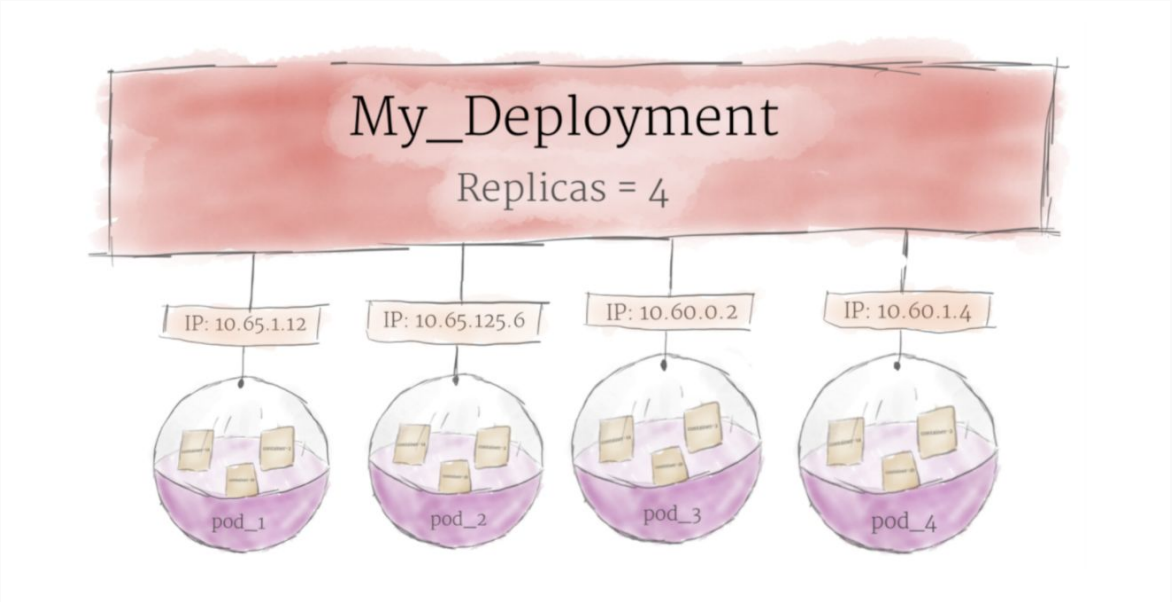
Ingress is actually **NOT** a type of service .

Instead, it sits in front of multiple services and act as a “smart router” or entrypoint into your cluster.



Deployment

A deployment is an object in Kubernetes that lets you manage a set of identical pods with **deployment strategy**

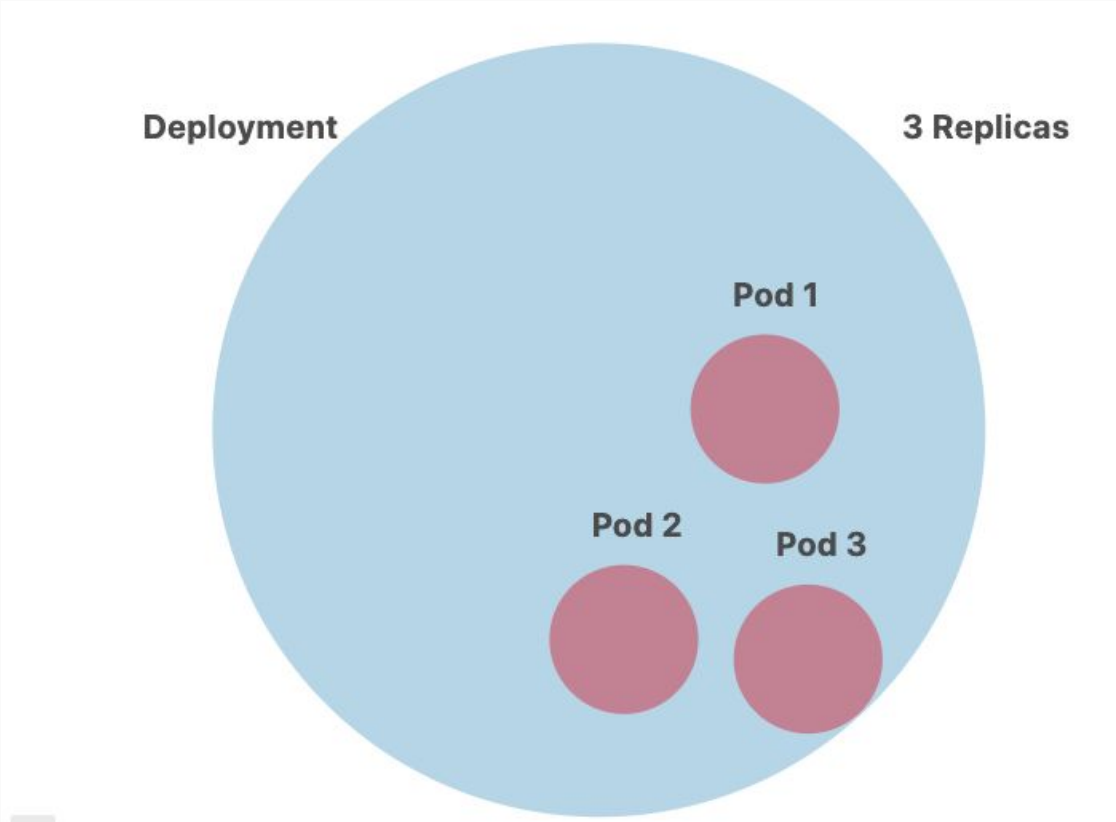


ReplicaSet vs Deployment

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: example-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      name: app
  template:
    metadata:
      labels:
        name: app
    spec:
      containers:
      - name: app
        image: learnk8s/hello:1.0.0
```

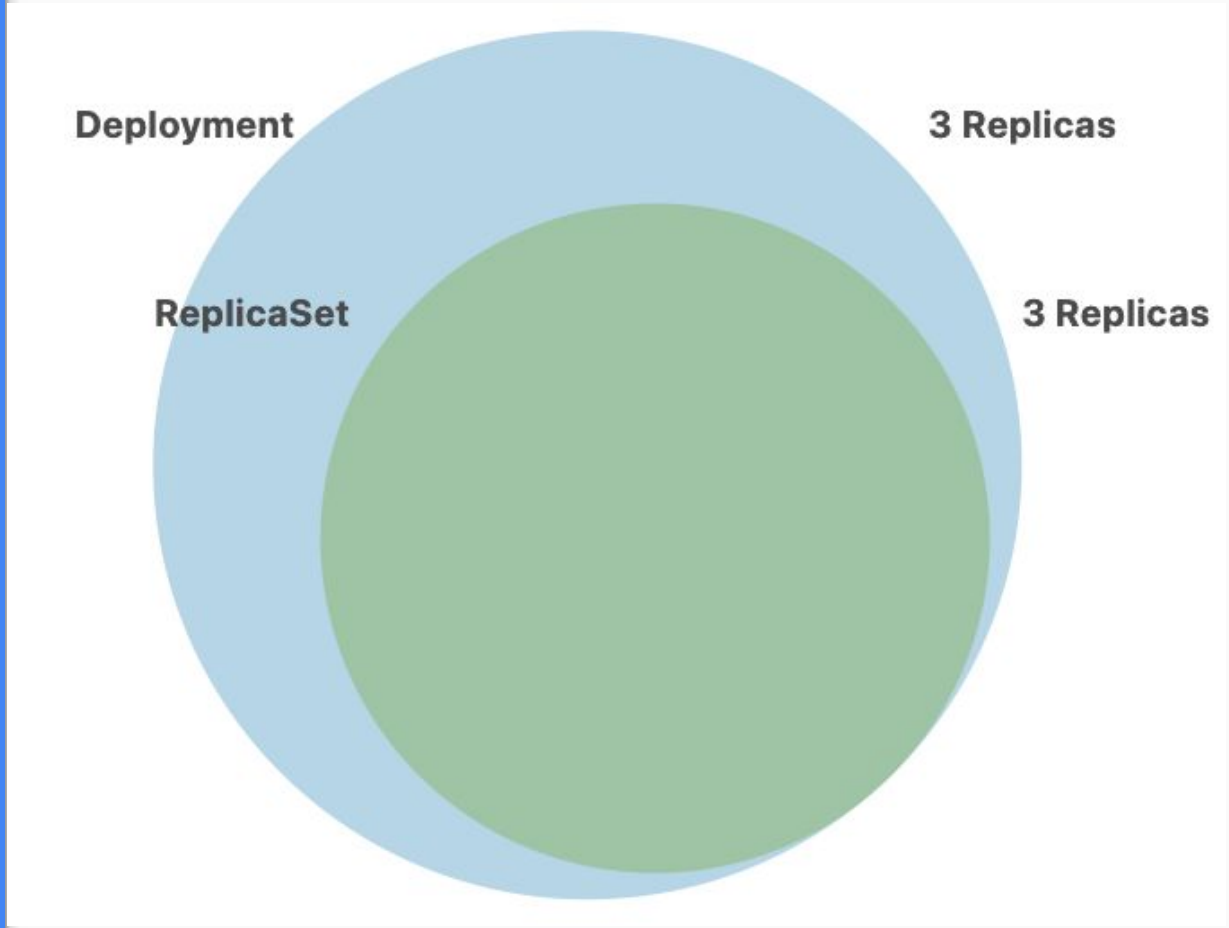
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      name: app
  template:
    metadata:
      labels:
        name: app
    spec:
      containers:
      - name: app
        image: learnk8s/hello:1.0.0
```

You might be tempted to think that Deployments are in charge of creating Pods.

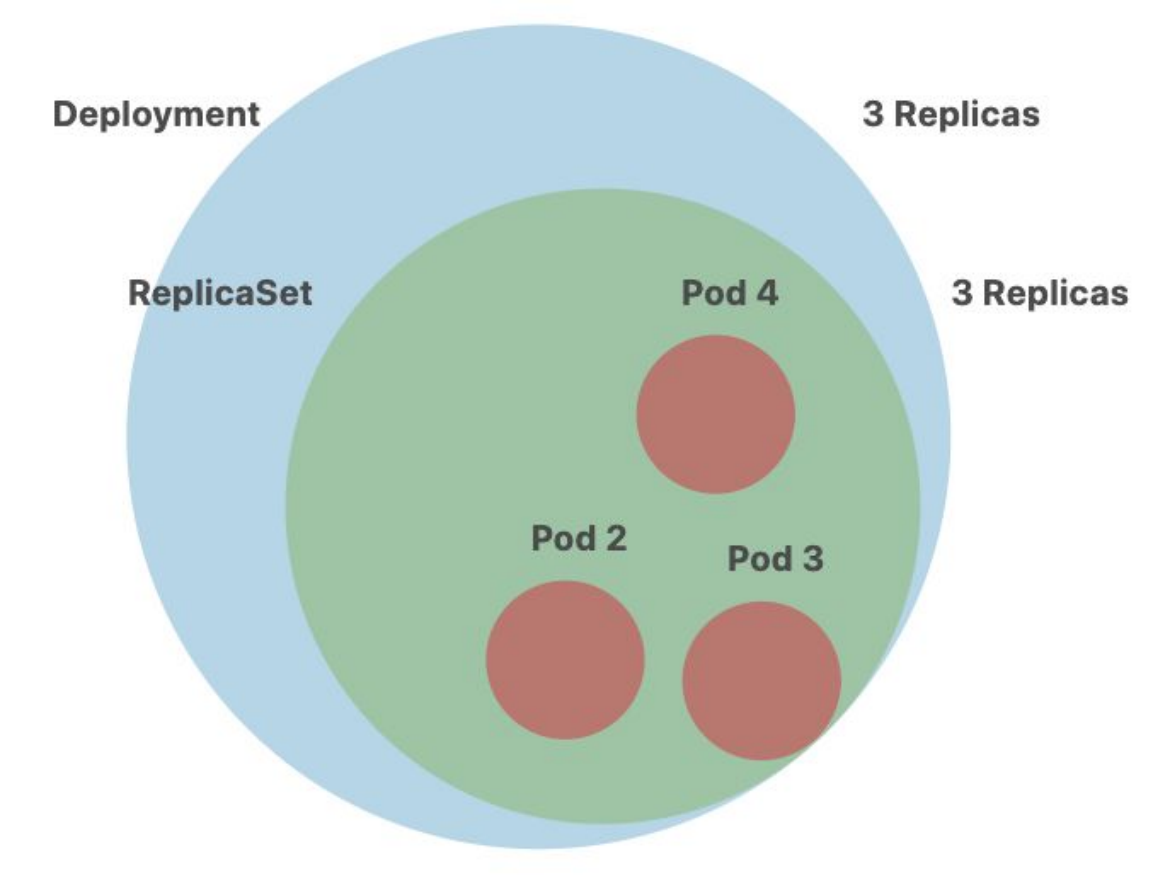


The Deployment **doesn't** create Pods. It creates another object called ReplicaSet that create pods instead.

The Deployment passes the spec (which includes the replicas) to the ReplicaSet.

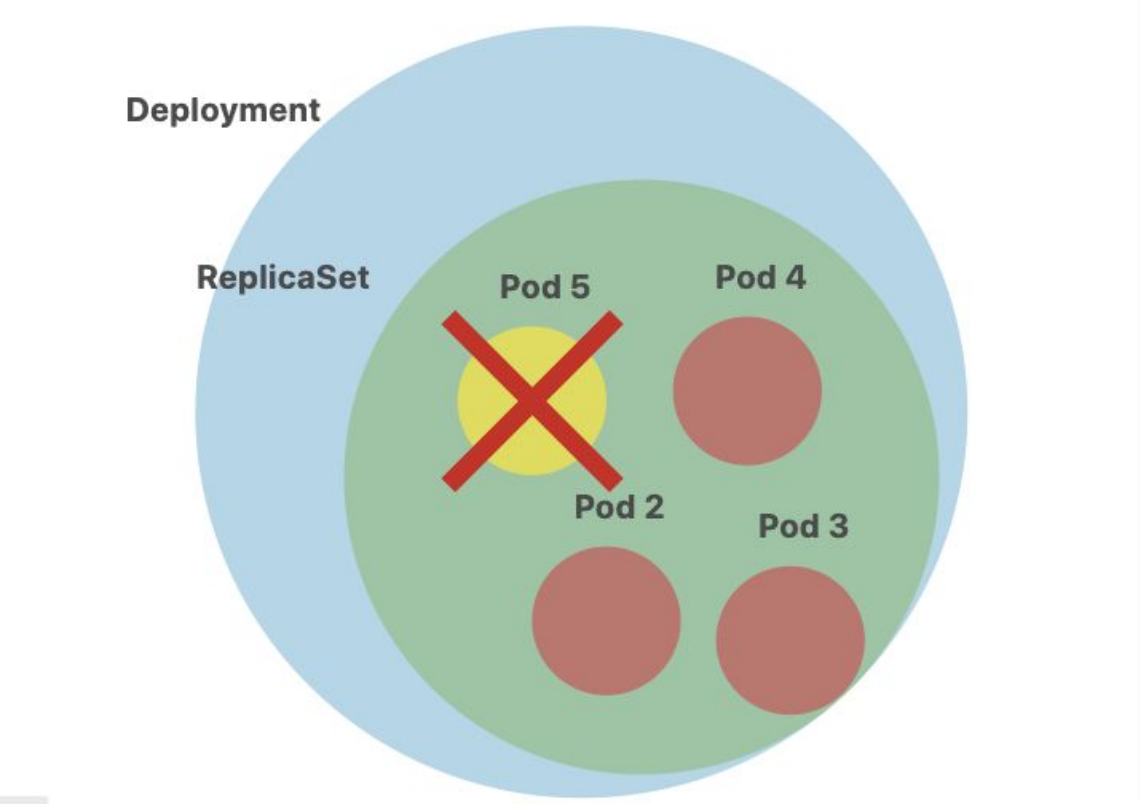


The ReplicaSet is in charge of creating the Pods and watching over them.



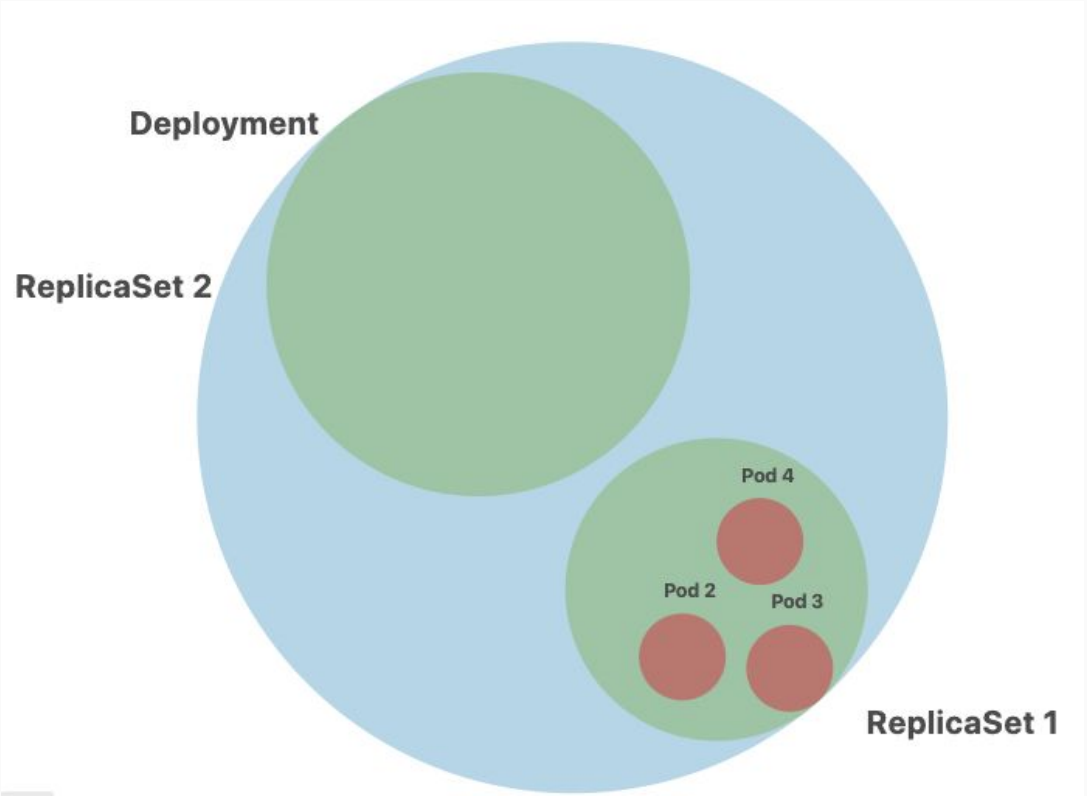
Can we have **two** Pods in the same ReplicaSet?

ReplicaSets can only contain a single type of Pod. You can't use two different Docker images.

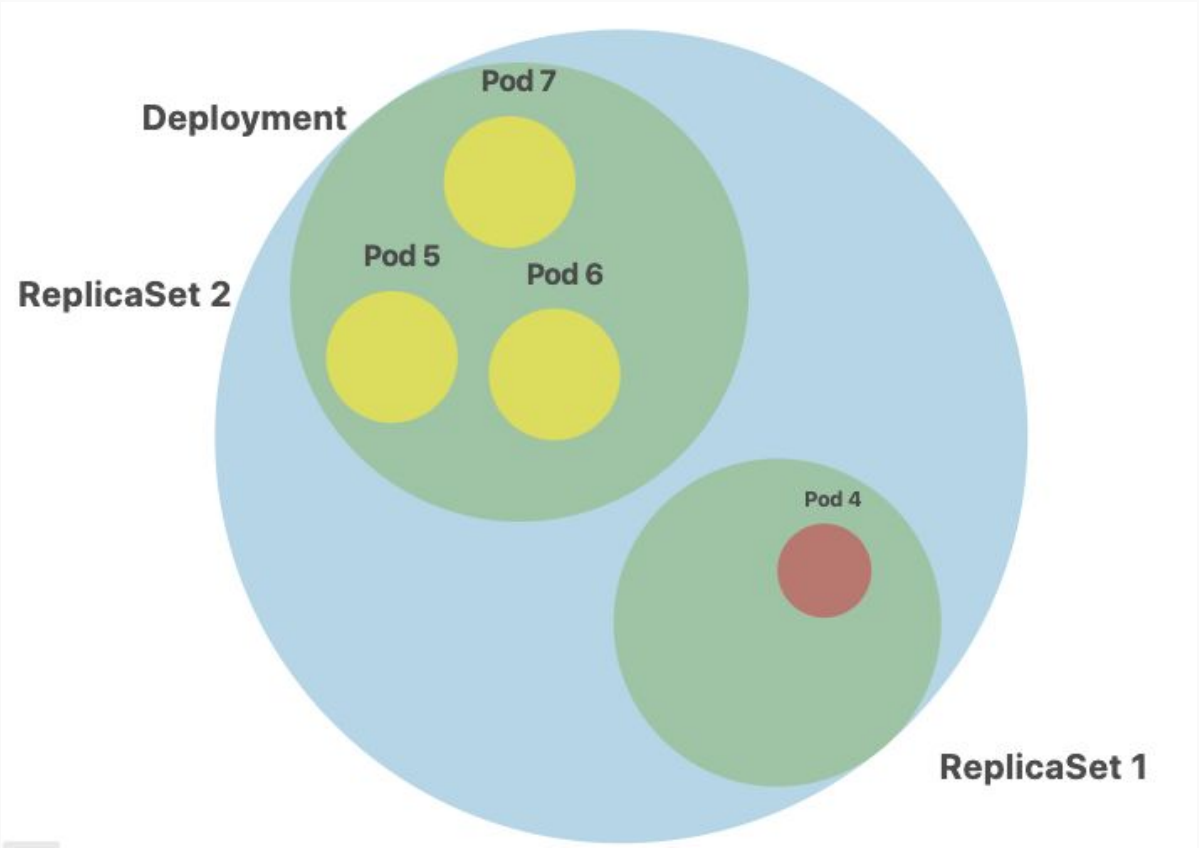


How can we deploy **two versions** of the app simultaneously?

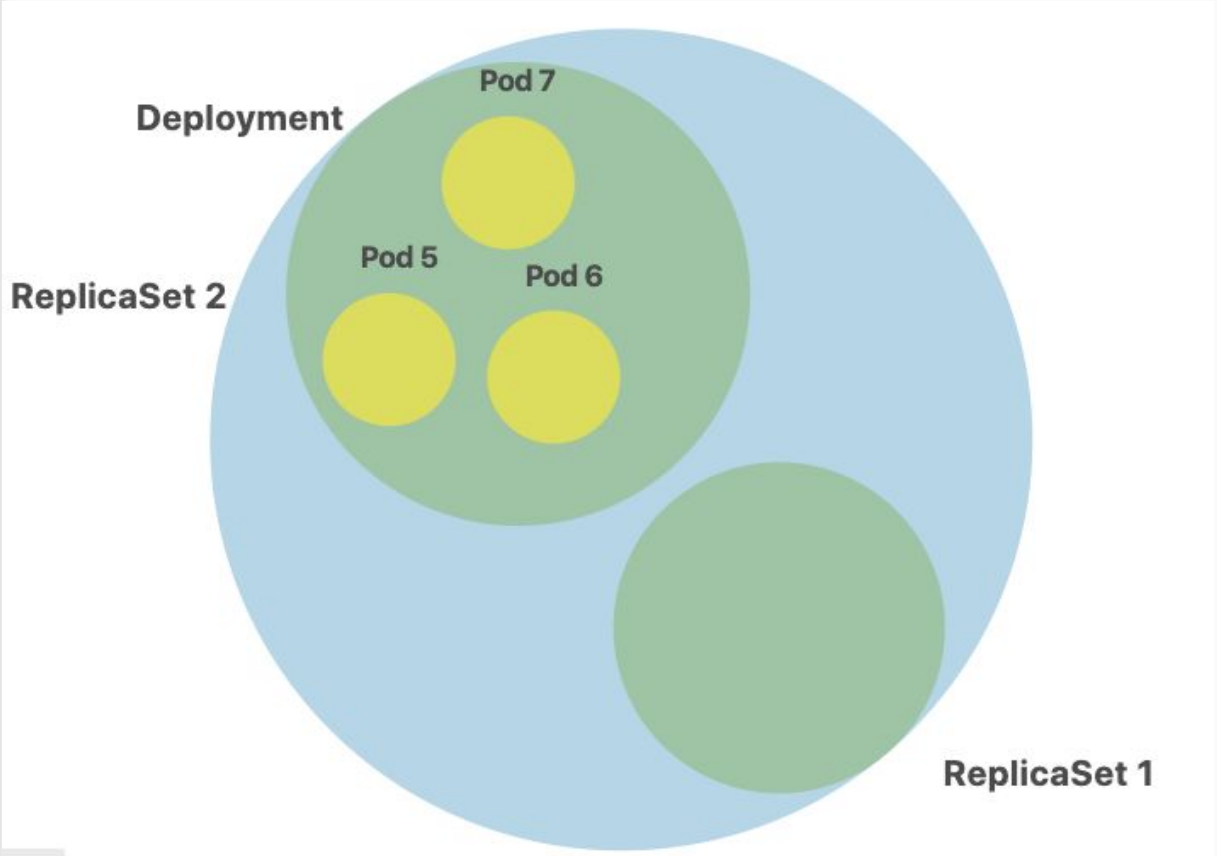
The Deployment knows that you can't have different Pods in the same ReplicaSet. So it creates another ReplicaSet.



It increases the number of replicas of the current ReplicaSet. And then it proceeds to decrease the replicas count in the previous ReplicaSet.



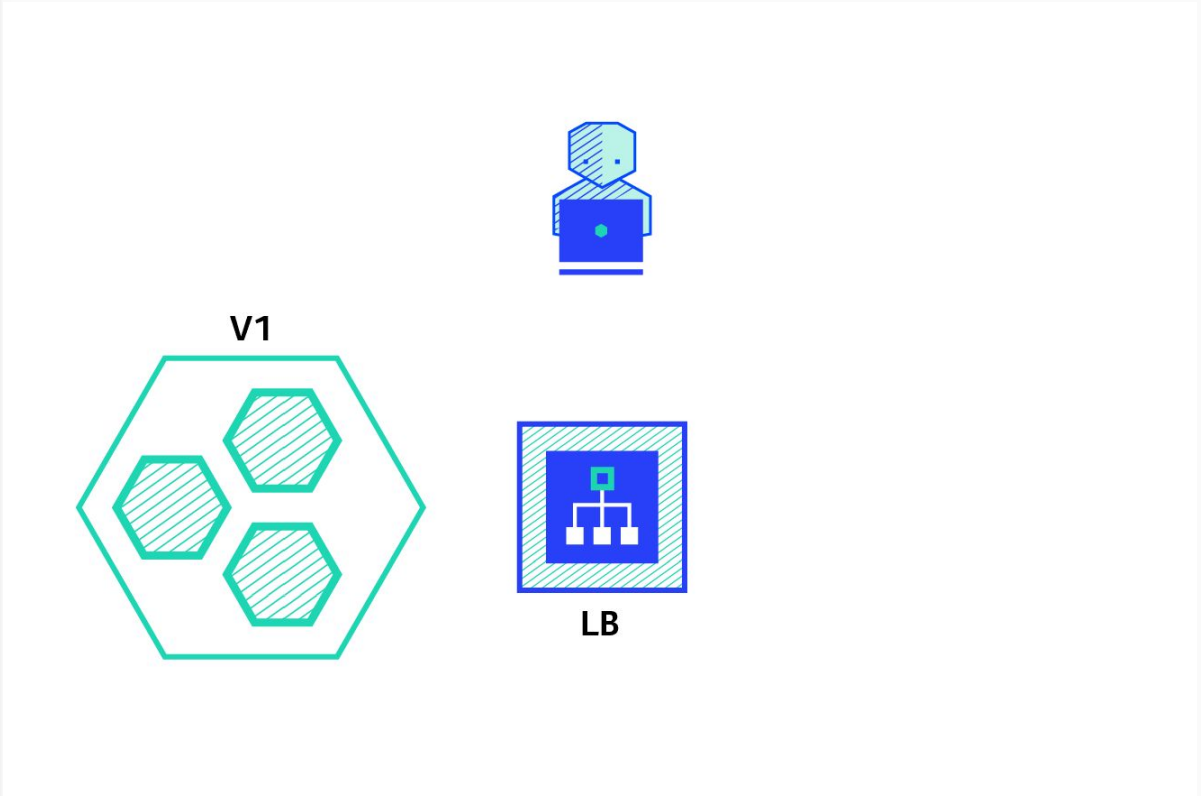
After the rolling update is completed, **the previous ReplicaSet is not deleted**.



Deployment Strategy

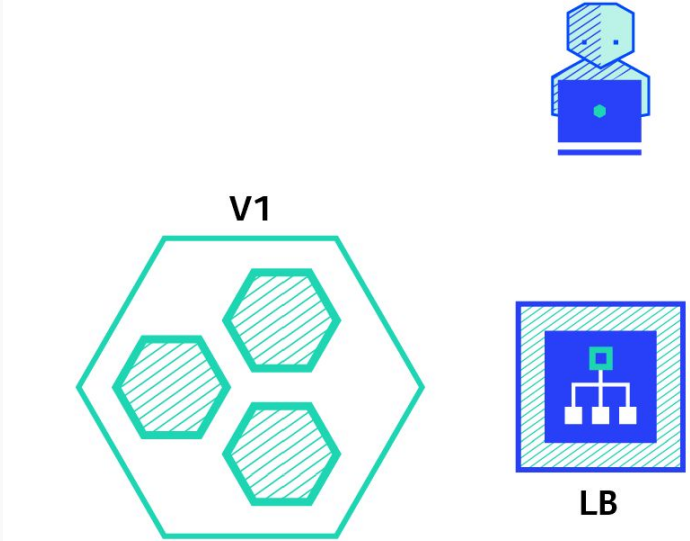
Recreate / Highlander

Version A is **terminated** then version B is **rolled out**.



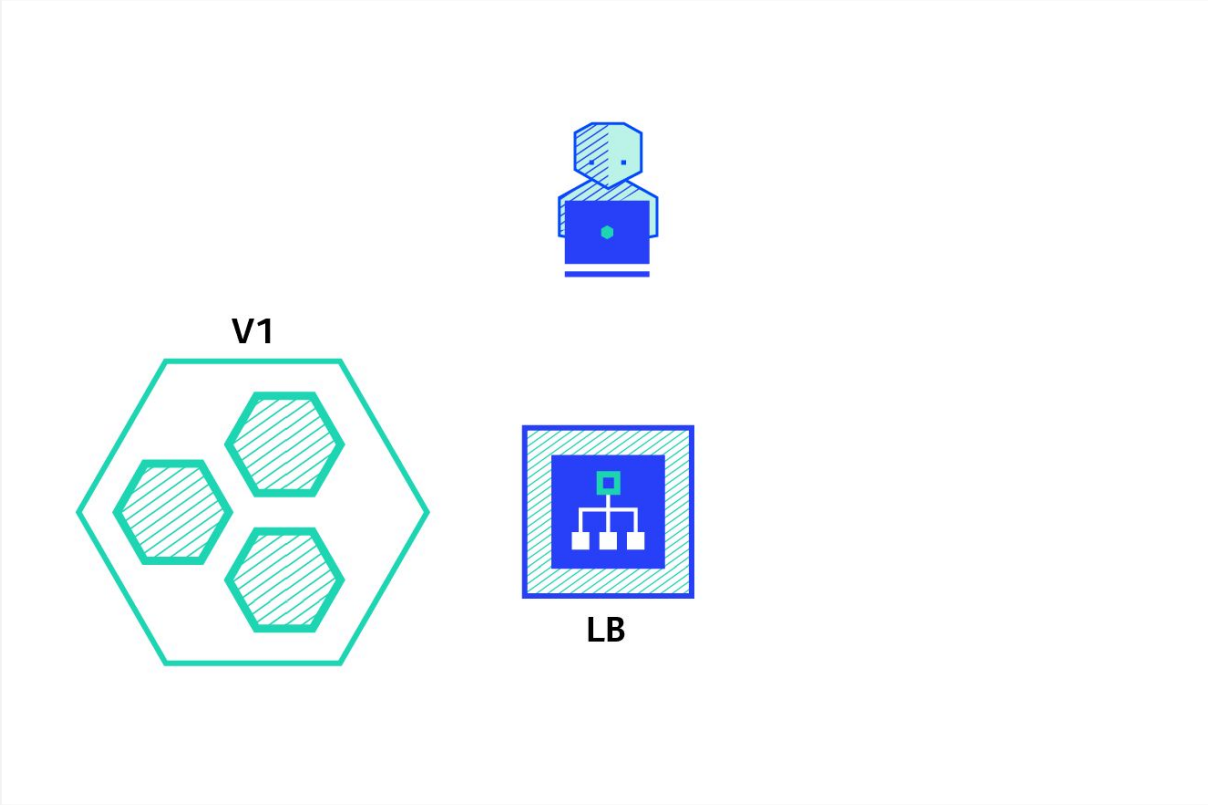
Rolling-Update

Version B is slowly **rolled out** and **replacing** version A.



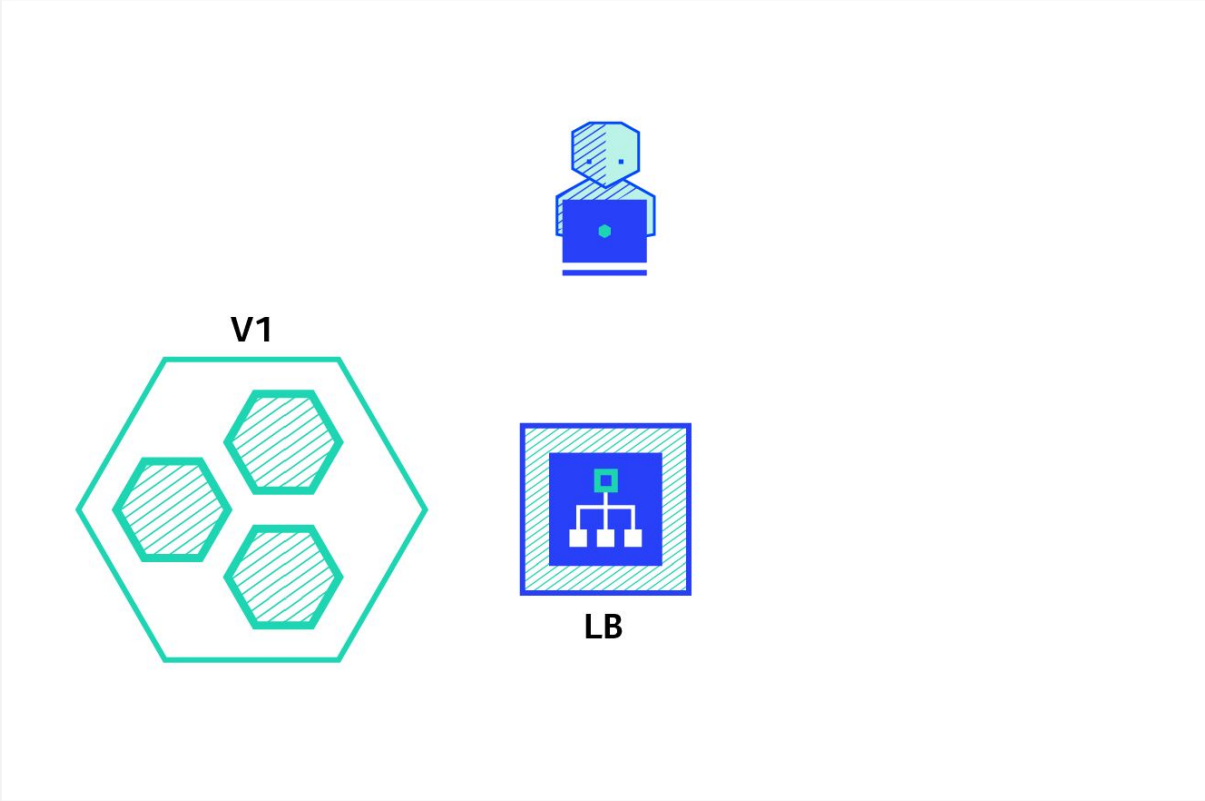
Blue/Green a.k.a Red/Black

Version B is released **alongside** version A, then the traffic is **switched** to version B



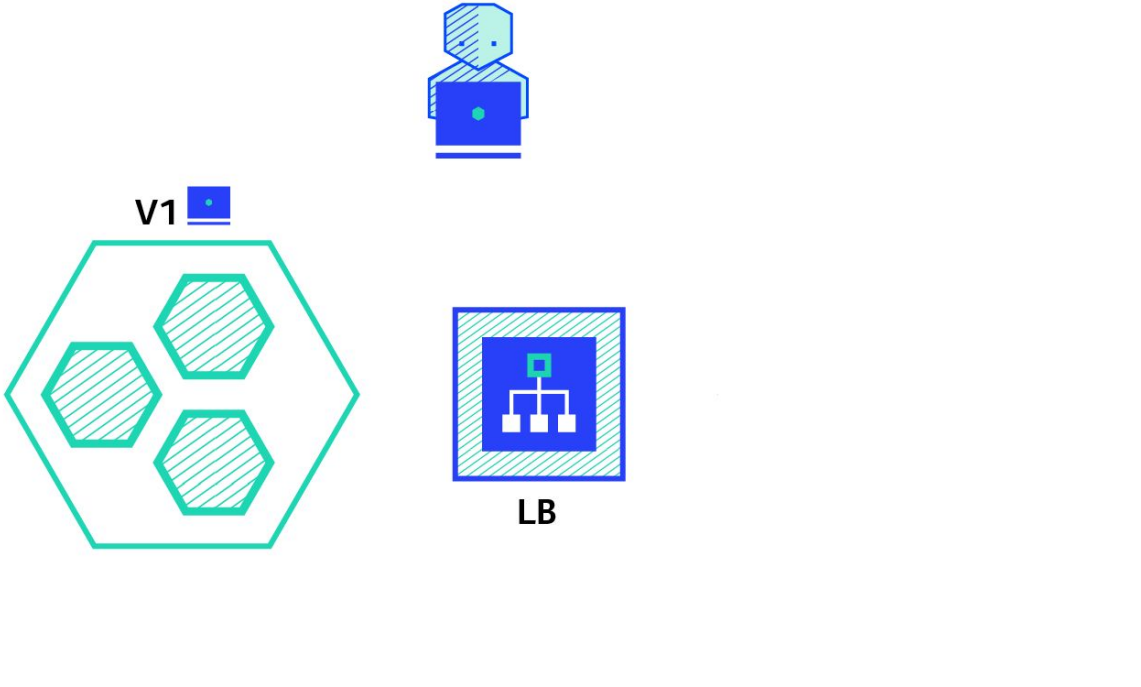
Canary

Version B is released to a **subset of users**, then proceed to a **full rollout**.



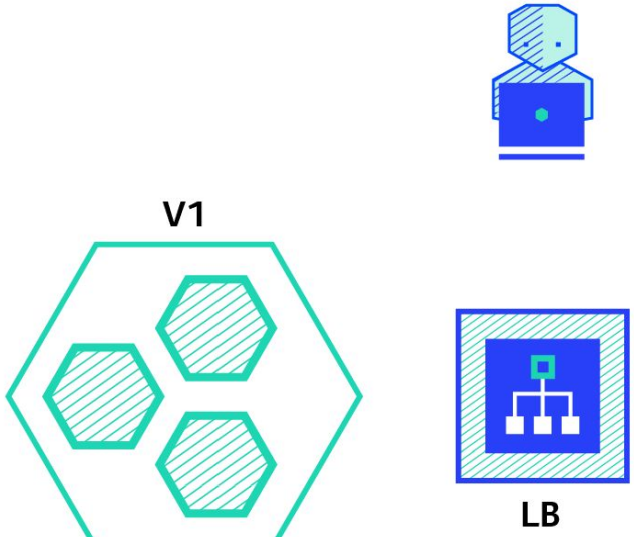
A/B Testing

Version B is **released** to a subset of users under **specific condition**



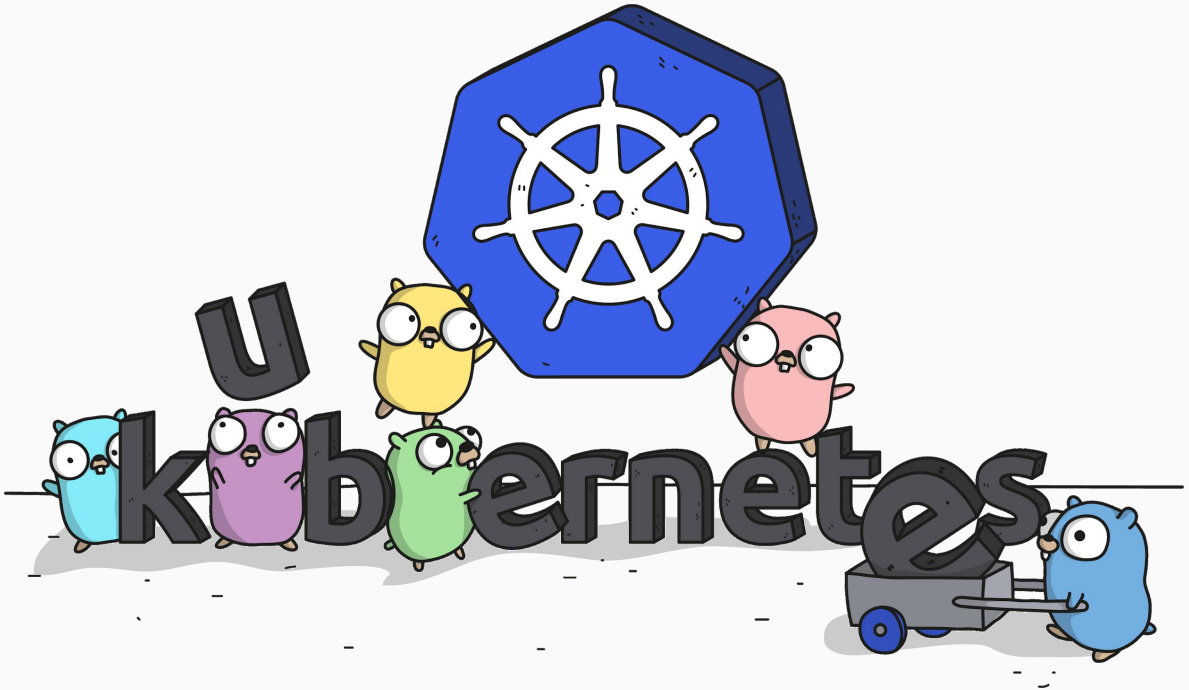
Shadow

Version B receives **real-world traffic** alongside version A and **doesn’t impact** the response.



Demo Time

Thanks



P.S. I'm looking for a summer internship :D

Contact : telegram.me/iqbalsyamil