NOLSATU

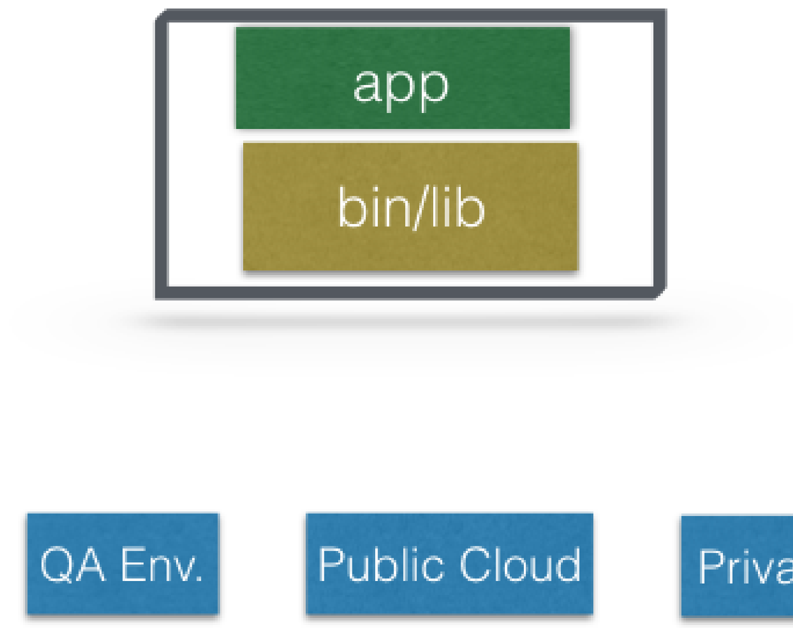
Kubernetes Administration (K9-ADM)

References

- Kubernetes Documentation: https://kubernetes.io/docs/home/
- Kubernetes Cookbook - Sébastien Goasguen, Michael Hausenblas, 2018
- Kubernetes in Actions - Marko Lukša, 2017

Container Orchestration

What Are Containers?



Application-centric way to deliver high-performing, scalable applications on the infrastructure of your choise.

What Is Container Orchestration?

Container Orchestrators are the tools which group hosts together to form a cluster, and help us fulfill the requirements mentioned below:

- Are fault-tolerant
- Can scale, and do this on-demand
- Use resources optimally
- Can discover other applications automatically, and communicate with each other
- Are accessible from the external world
- Can update/rollback without any downtime.

Container Orchestrators

docker SWARM

Apache MESOS™

Amazon ECS

kubernetes by Google

HashiCorp Nomad

Container Orchestrators (1)

- Docker Swarm, provided by Docker, Inc. It is part of Docker Engine.
- Kubernetes, started by Google, but now, it is a part of the Cloud Native Computing Foundation project.
- Mesos Marathon, one of the frameworks to run containers at scale on Apache Mesos.
- Amazon EC2 Container Service (ECS), a hosted service provided by AWS to run Docker containers at scale on its infrastructrue.
- Hashicorp Nomad, provided by HashiCorp.

Why Use Container Orchestrators?

- Bring multiple hosts together and make them part of a cluster
- Schedule containers to run on different hosts
- Help containers running on one host reach out to containers running on other hosts in the cluster
- Bind containers and storage
- Bind containers of similar type to a higher-level construct, like services, so we don't have to deal with individual containers
- Keep resource usage in-check, and optimize it when necessary
- Allow secure access to applications running inside containers.

Kubernetes

What Is Kubernetes?

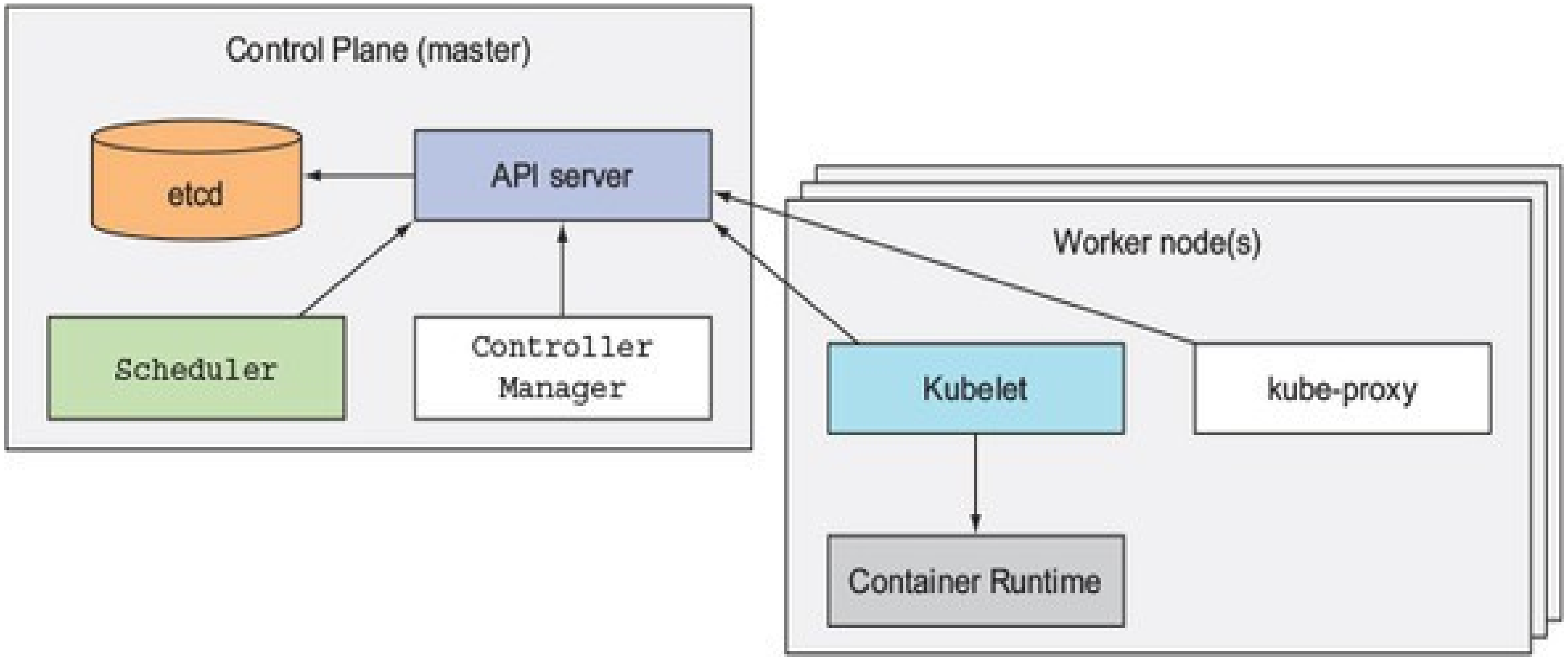
- "Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications."
- Kubernetes comes from the Greek word κυβερνήτης:, which means helmsman or ship pilot.
- People pronounce Kubernetes in a few different ways. Many pronounce it as Koo-ber-nay-tace, while others pronounce it more like Koo-ber-netties.
- Kubernetes is also referred to as k8s, as there are 8 characters between k and s.
- Kubernetes was started by Google and, with its v1.0 release in July 2015, Google donated it to the Cloud Native Computing Foundation (CNCF).

Kubernetes Features

- Automatic binpacking
- Self-healing
- Horizontal scaling
- Service discovery and Load balancing
- Automated rollouts and rollbacks
- Secrets and configuration management
- Storage orchestration
- Batch execution

Kubernetes Architecture

Kubernetes Architecture



Master Node Components

- **API Server**, which you and the other Control Plane components communicate with.
- **Scheduler**, which schedules your apps (assigns a worker node to each deployable component of your application).
- **Controller Manager**, which performs cluster-level functions, such as replicating components, keeping track of worker nodes, handling node failures, and so on.
- **Etcd**, a reliable distributed data store that persistently stores the cluster configuration.

Worker Node Components

- **Container Runtime**, Docker, rkt.
- **Kubelet**, which talks to the API server and manages containers on its node
- **Kube-proxy**, which load-balances network traffic between application components

Kubernetes Networking

Kubernetes using three different types of networks

- **Infrastructure Network**, The network your physical (or virtual) machines are connected to. Normally your production network, or a part of it
- **Service Network**: The (completely) virtual (rather fictional) network, which is used to assign IP addresses to Kubernetes Services, which you will be creating.
- **Pod Network**: This is the network, which is used by the pods. However it is not a simple network either, depending on what kubernetes network solution you are employing.

Container-to-Container Communication Inside a Pod

Inside a Pod, containers share the Network Namespaces, so that they can reach to each other via localhost.

Pod-to-Pod Communication Across Nodes

Pod-to-Pod communication across Hosts can be achieved via:

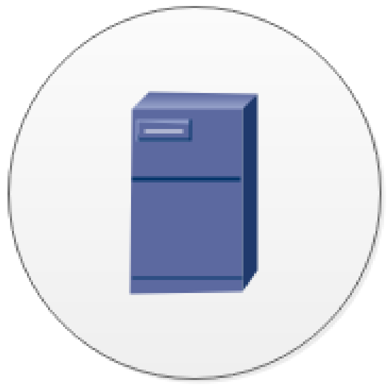
- Routable Pods and nodes, using the underlying physical infrastructure, like Google Container Engine
- Using Software Defined Networking, like Flannel, Weave, Calico, etc.

Communication Between the External World and Pods

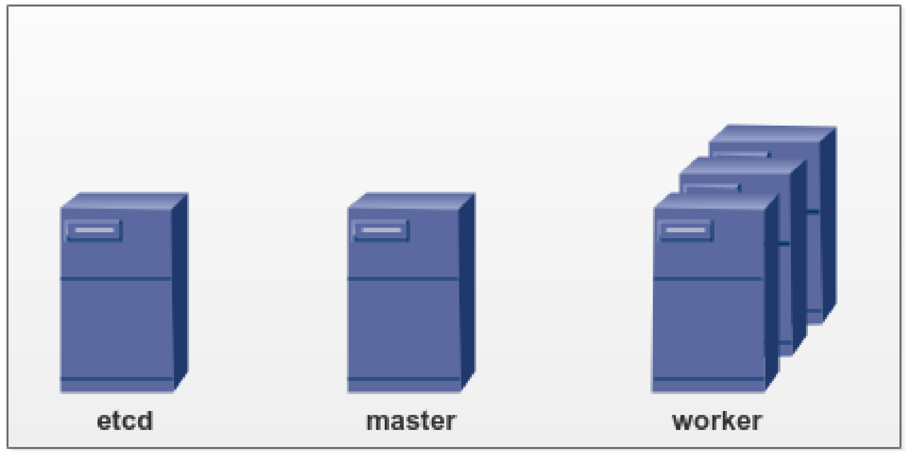
By exposing our services to the external world with kube-proxy, we can access our applications from outside the cluster.

Installing Kubernetes

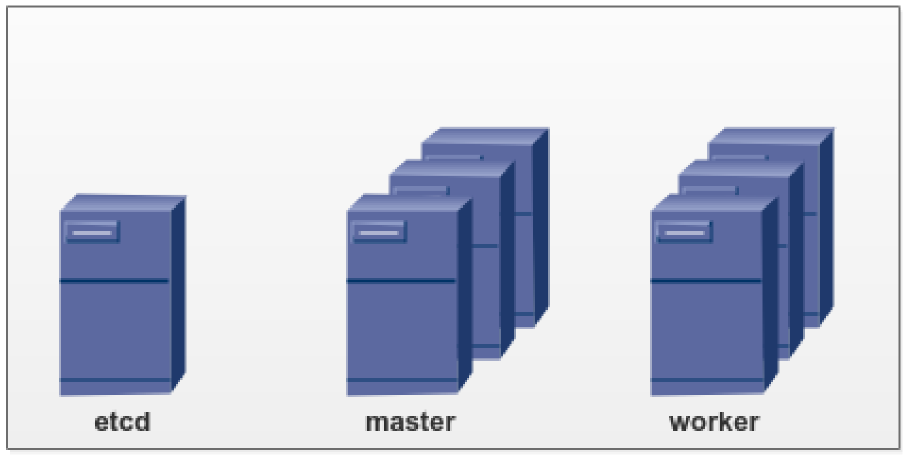
Kubernetes Configuration



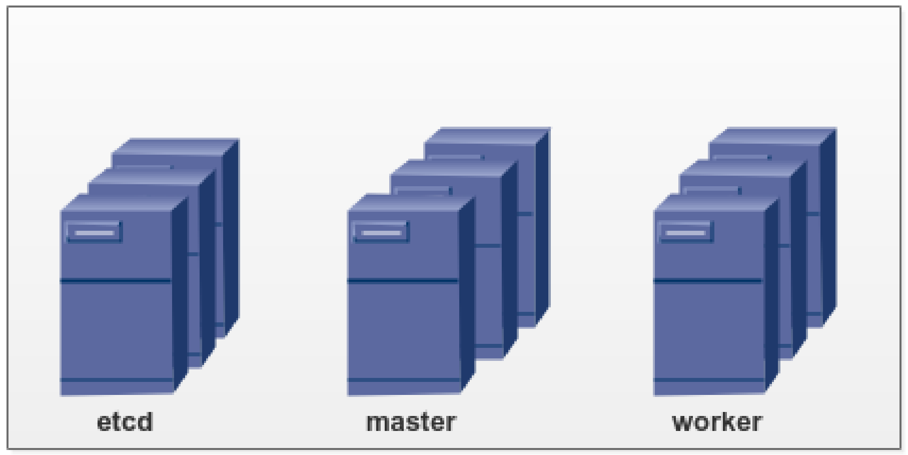
All in One
Kubernetes Cluster



single etcd, single master, multi worker
Kubernetes Cluster



single etcd, multi master, multi worker
Kubernetes Cluster



multi etcd, single master, multi worker
Kubernetes Cluster

Independent Solutions

- Running Kubernetes Locally via Minikube
- Bootstrapping Clusters with kubeadm
- Creating a Custom Cluster from Scratch

Hosted Solutions

- Google Container Engine
- Azure Container Service
- IBM Bluemix Container Service

On-Premise Solutions

- CoreOS
- CloudStack
- VMware vSphere
- VMware Photon Controller
- DCOS
- oVirt
- OpenStack
- rkt
- Mesos
- Ubuntu Juju
- Windows Server Containers

Accessing Kubernetes

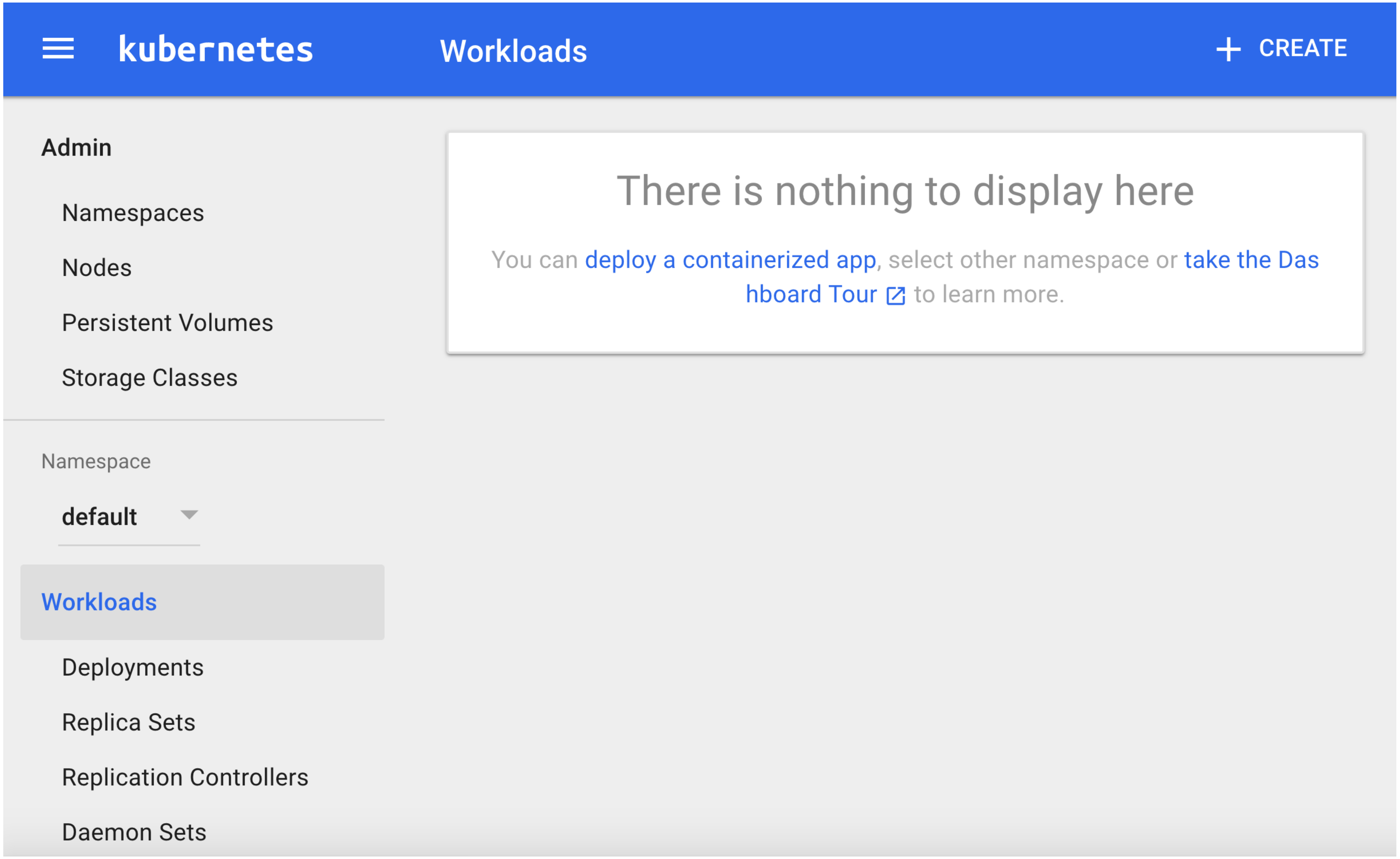
Access Kubernetes Cluster

- CLI: kubectl
- GUI: kubernetes-dashboard
- APIs

Kubectl Configuration File

- kubectl config view, menampilkan detil koneksi cluster
- kubectl cluster-info, menampilkan info cluster

Kubernetes Dashboard



There is nothing to display here

You can deploy a containerized app, select other namespace or take the Das hboard Tour to learn more.

Kubectl Proxy

- HTTP Proxy to Access the Kubernetes API
- runs kubectl in a mode where it acts as a reverse proxy
- Example: kubectl proxy --port=8080
- Exploring the Kubernetes API: curl http://localhost:8080/api/

APIs without Proxy

- Get the token
- Get the API Server endpoint
- Access the API Server using the curl command, as shown below

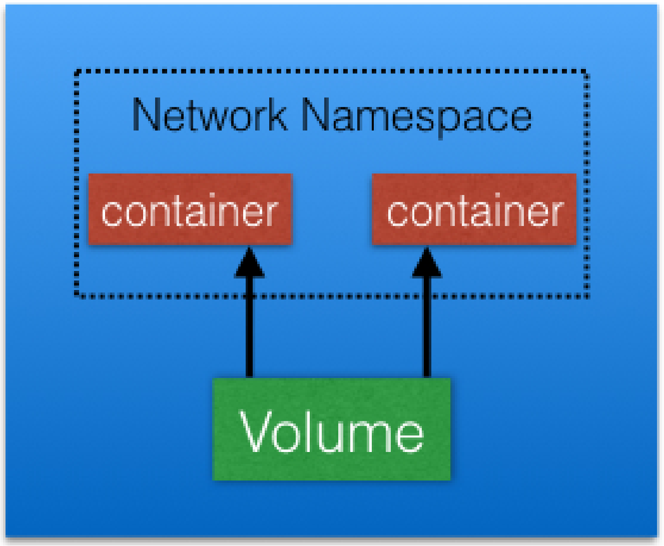
Kubernetes Building Blocks

Kubernetes Object Model

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

- apiVersion, API endpoint on API server which we want to connect to
- kind, object type
- metadata, basic information of object
- spec, desire state of deployment

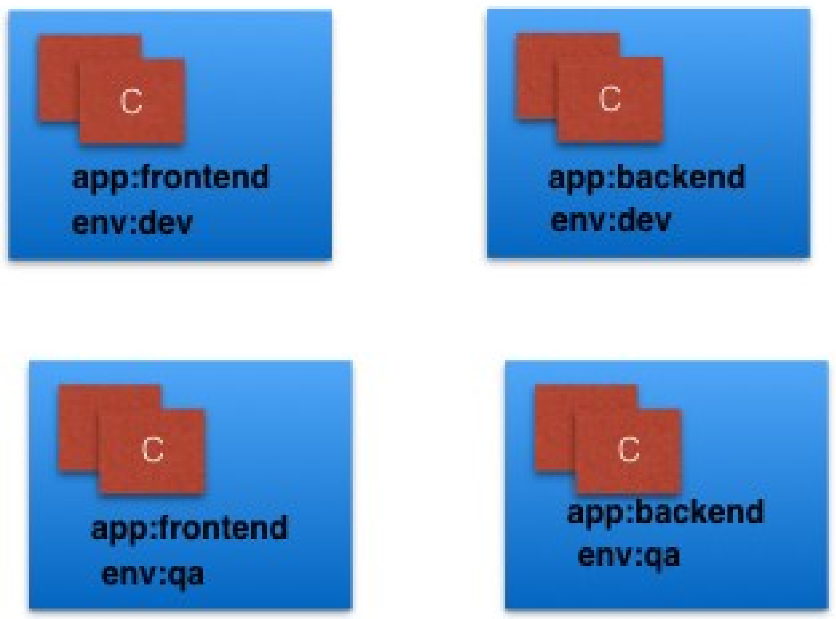
Pods



A Pod represents a single instance of the application. A Pod is a logical collection of one or more containers, which:

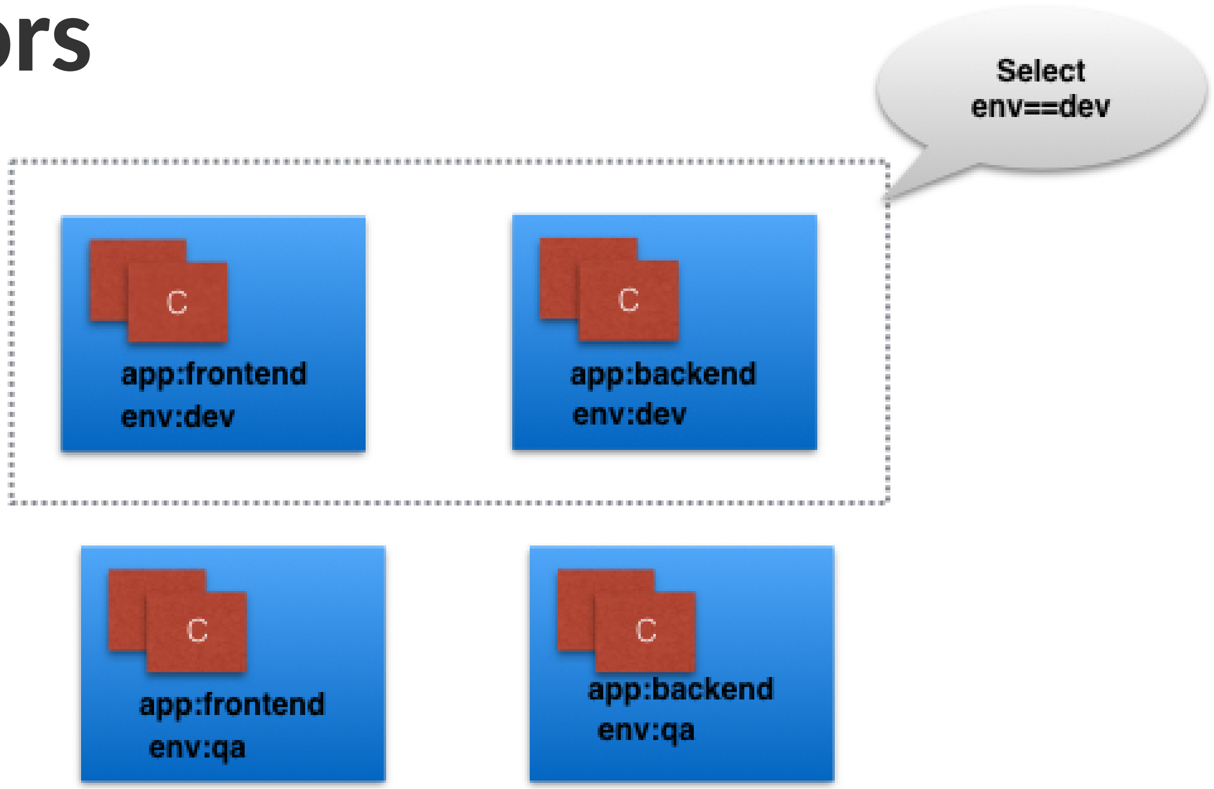
- Are scheduled together on the same host
- Share the same network namespace
- Mount the same external storage (Volumes)

Labels



Labels are key-value pairs that can be attached to any Kubernetes objects (e.g. Pods). Labels are used to organize and select a subset of objects, based on the requirements in place. Many objects can have the same label(s). Labels do not provide uniqueness to objects.

Label Selectors

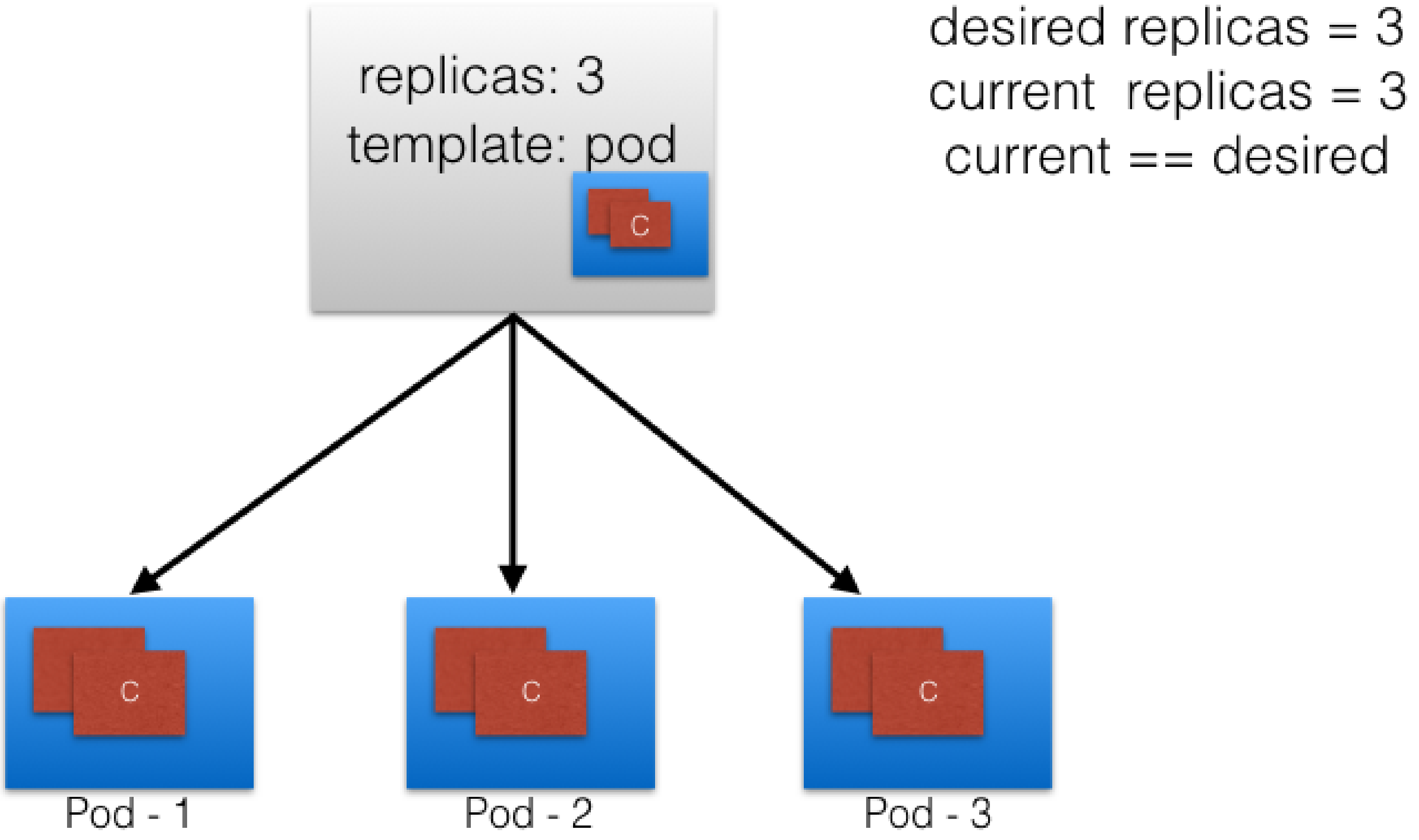


- Equality-Based Selectors allow filtering of objects based on label keys and values. With this type of Selectors, we can use the **=**, **==**, or **!=** operators.
- Set-Based Selectors allow filtering of objects based on a set of values. With this type of Selectors, we can use the **in**, **notin**, and **exist** operators.

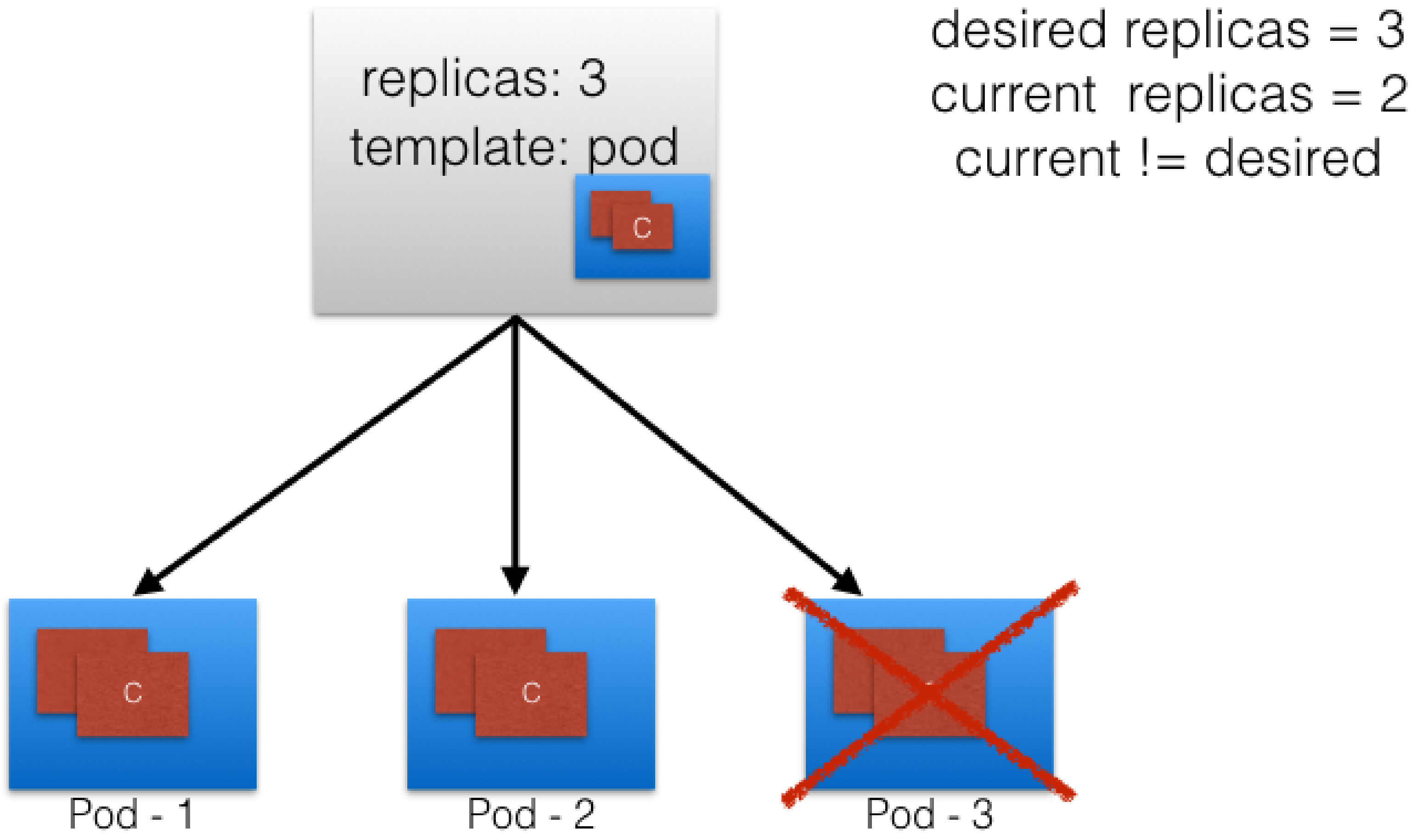
Replica Sets

A Replica Set (rs) is the next-generation Replication Controller. Replica Sets support both equality- and set-based Selectors, whereas ReplicationControllers only support equality-based Selectors.

Replica Set Operations (1)

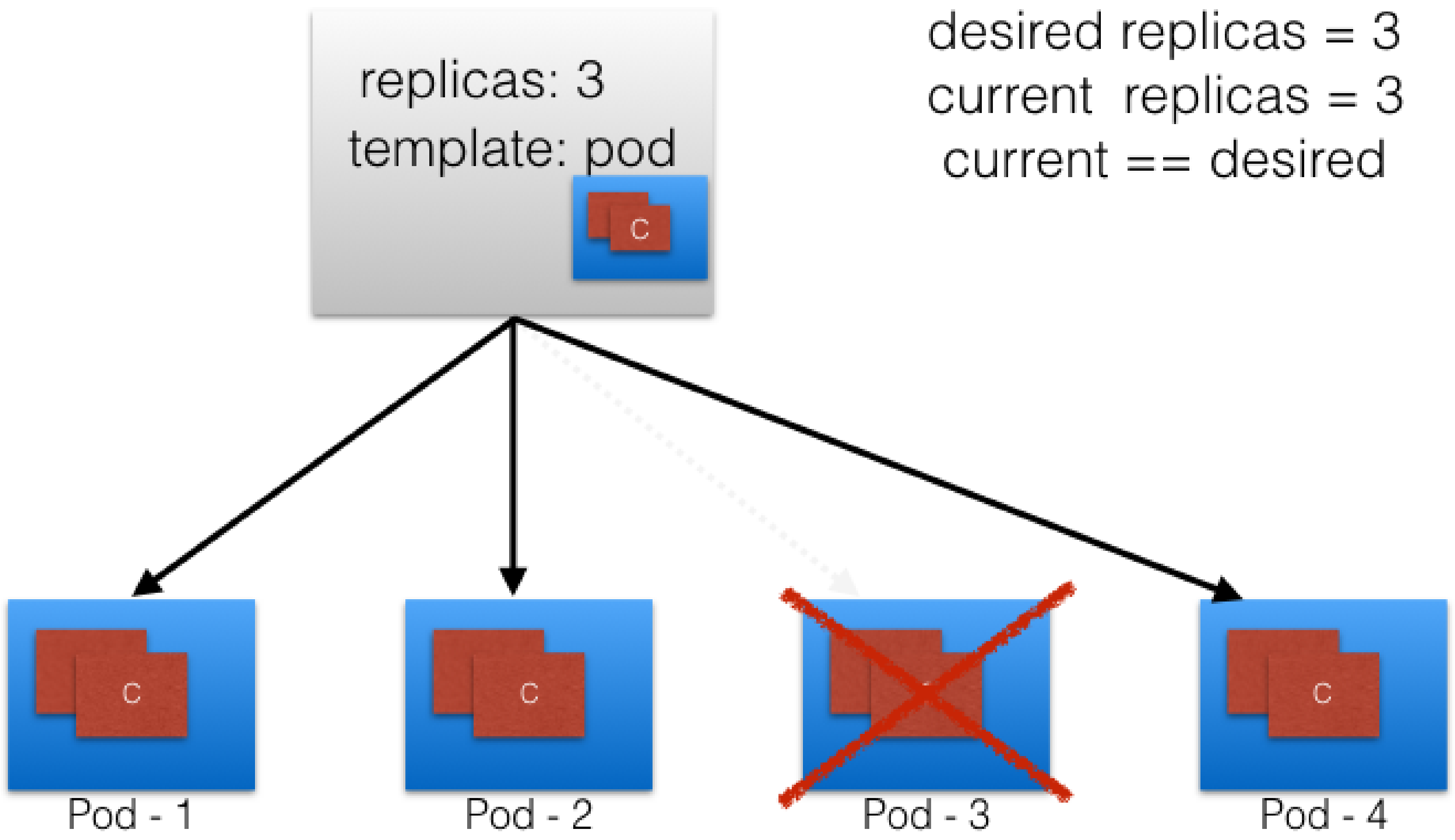


Replica Set Operations (2)



desired replicas = 3
current replicas = 2
current != desired

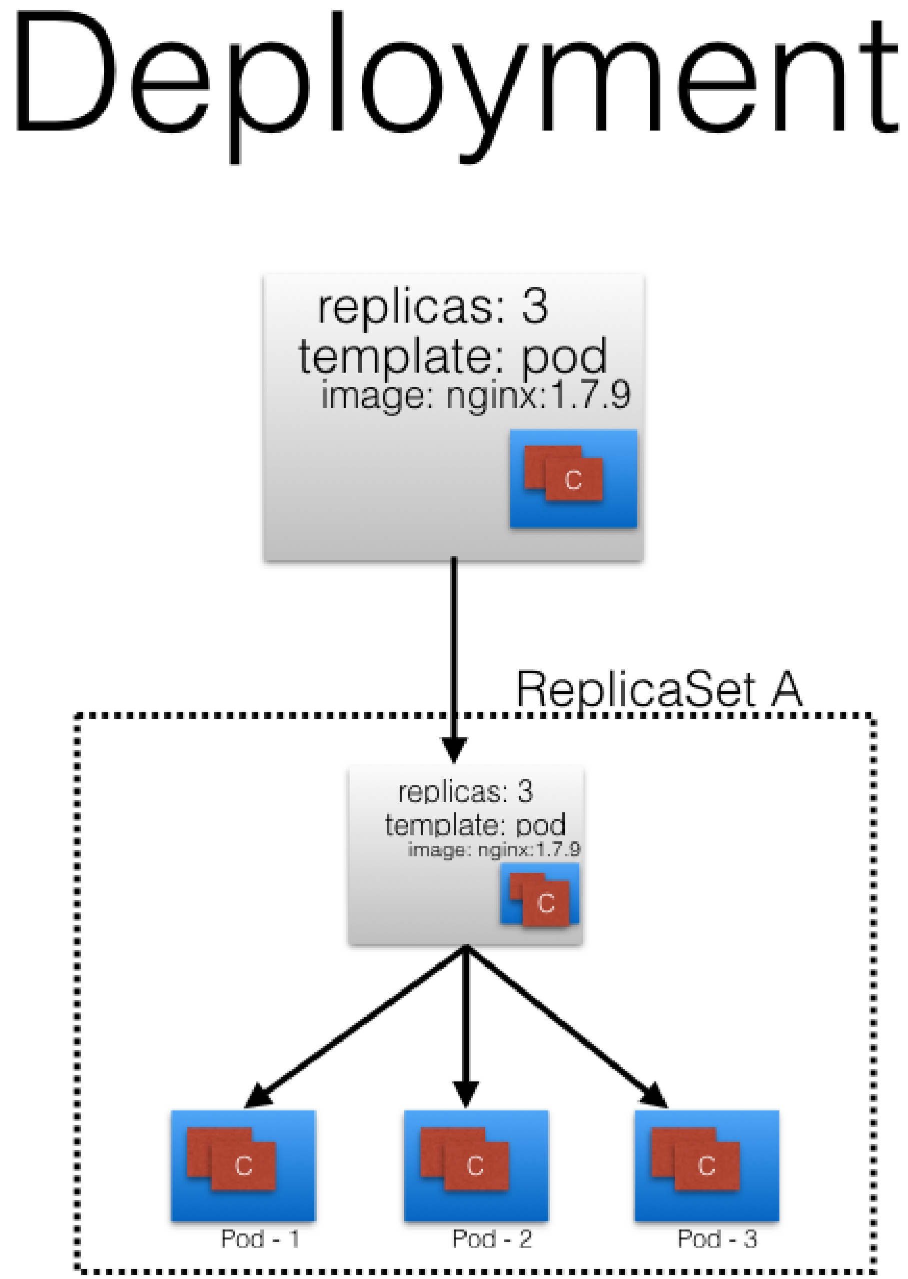
Replica Set Operations (3)



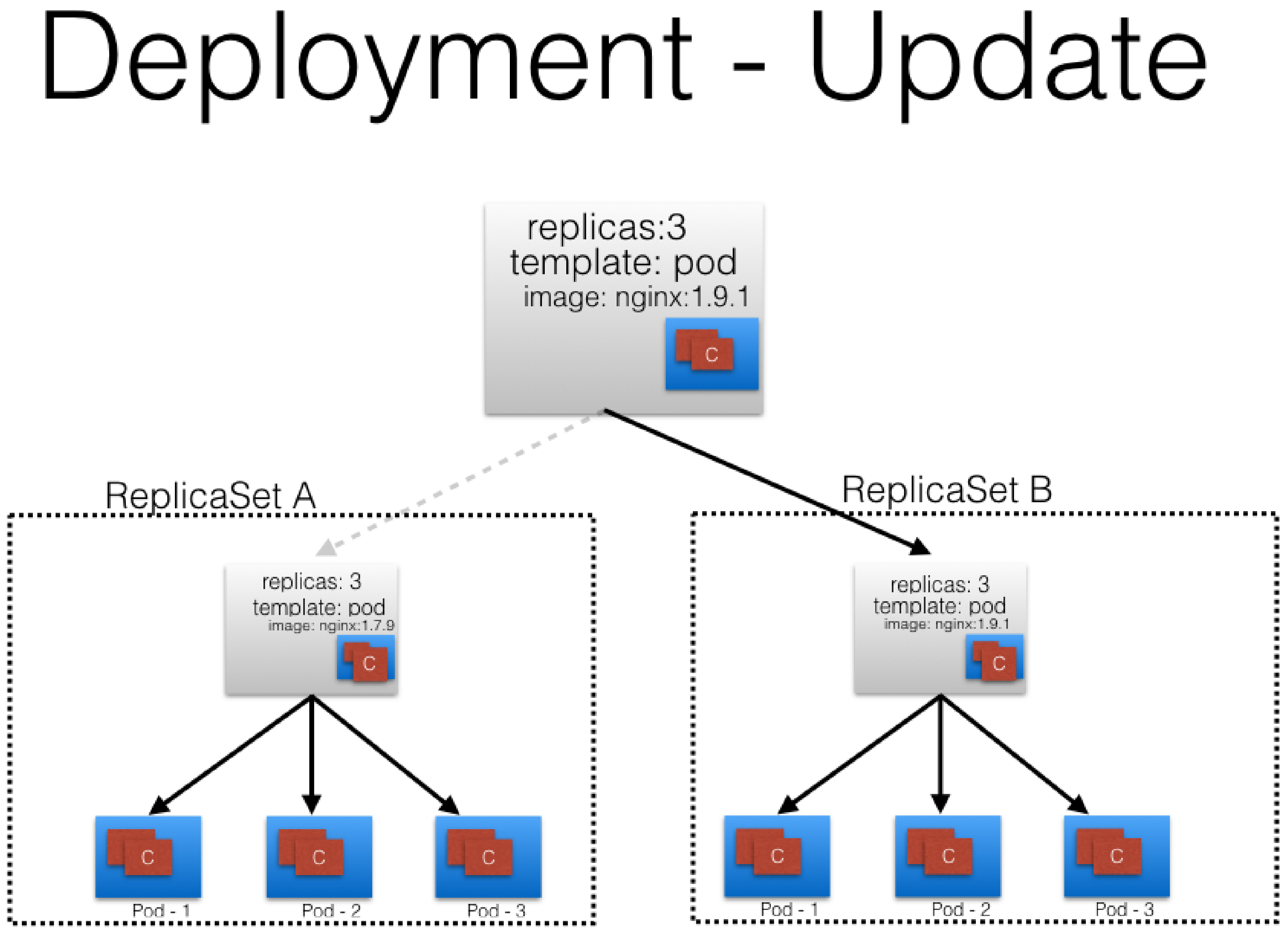
Deployments

Deployment objects provide declarative updates to Pods and ReplicaSets. The DeploymentController is part of the Master Node's Controller Manager, and it makes sure that the current state always matches the desired state.

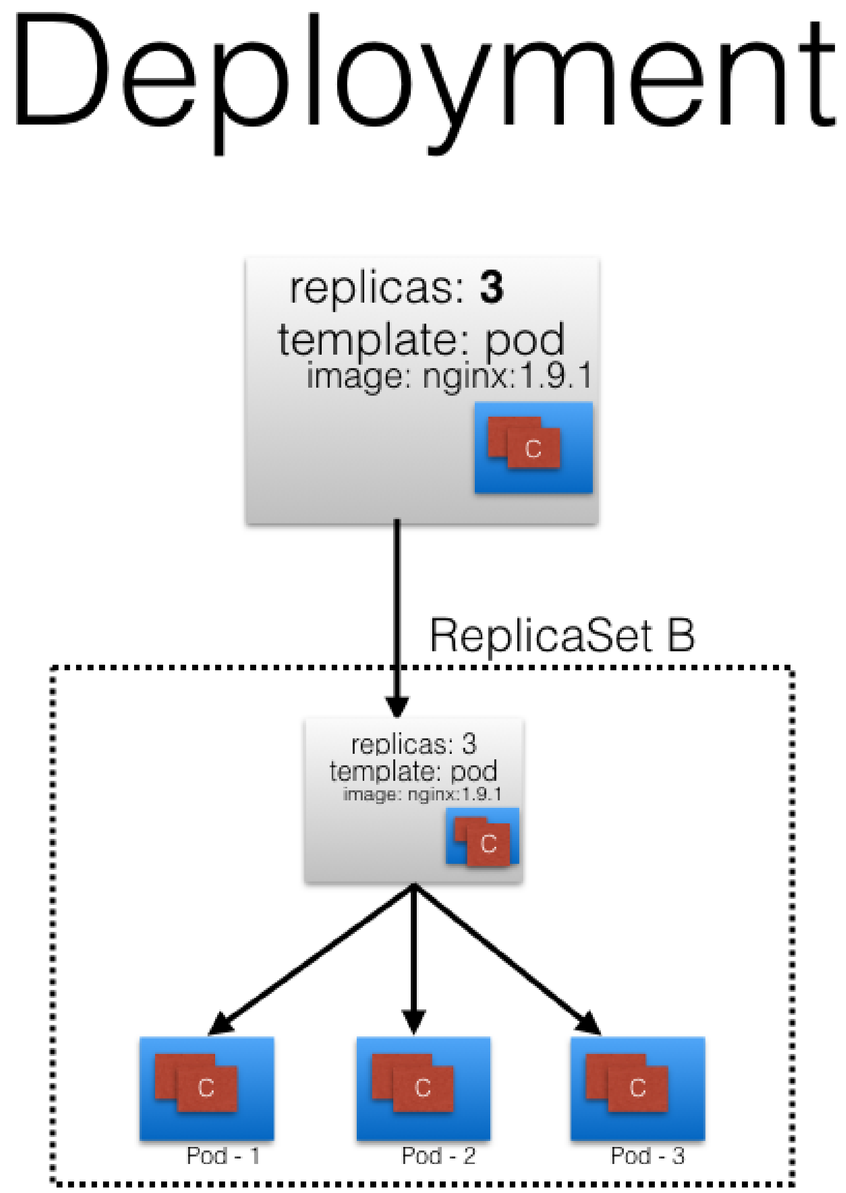
Deployment Operations (1)



Deployment Operations (2)



Deployment Operations (3)



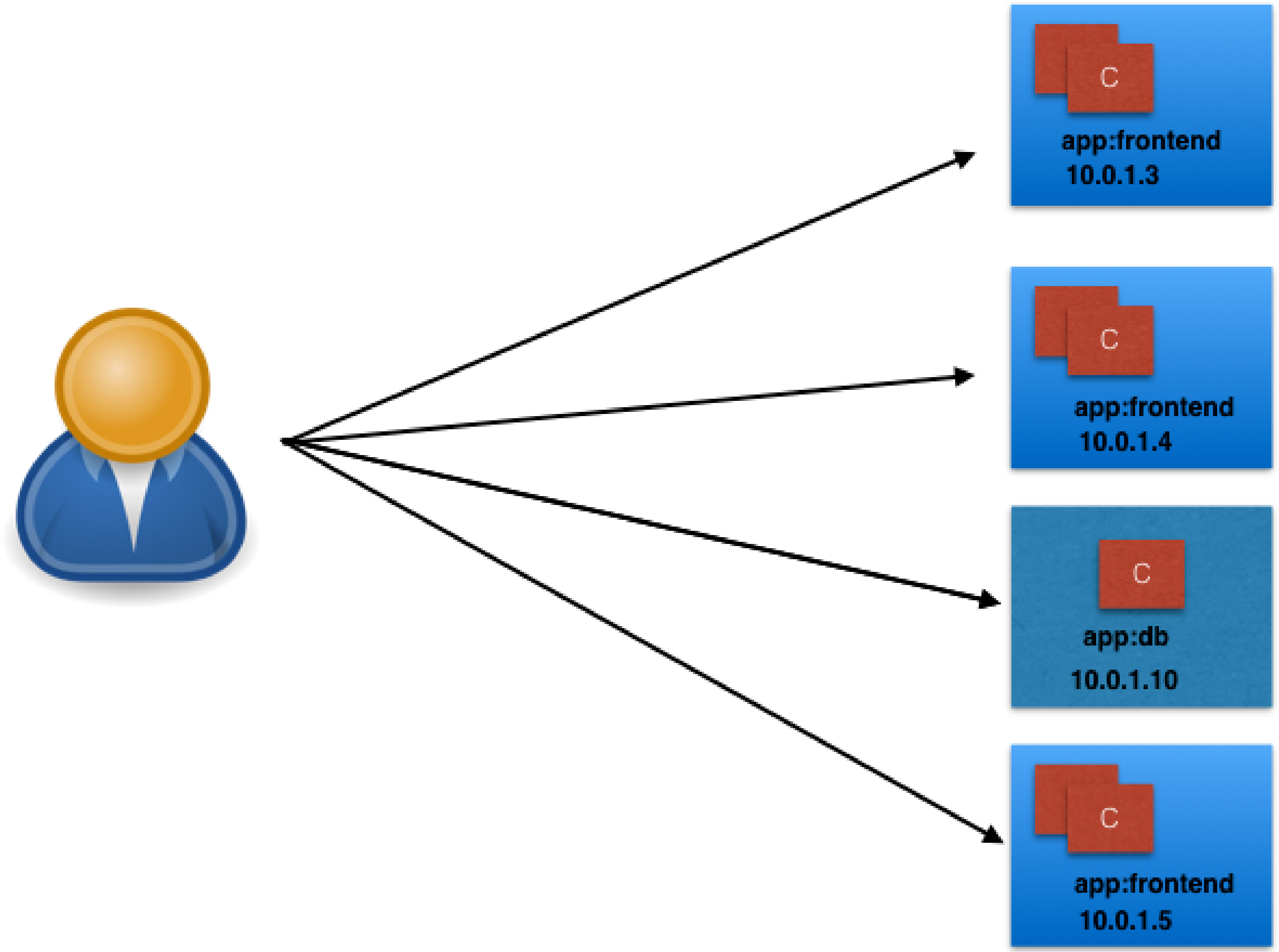
Namespaces

```
$ kubectl get namespaces
NAME           STATUS       AGE
default        Active       11h
kube-public    Active       11h
kube-system    Active       11h
```

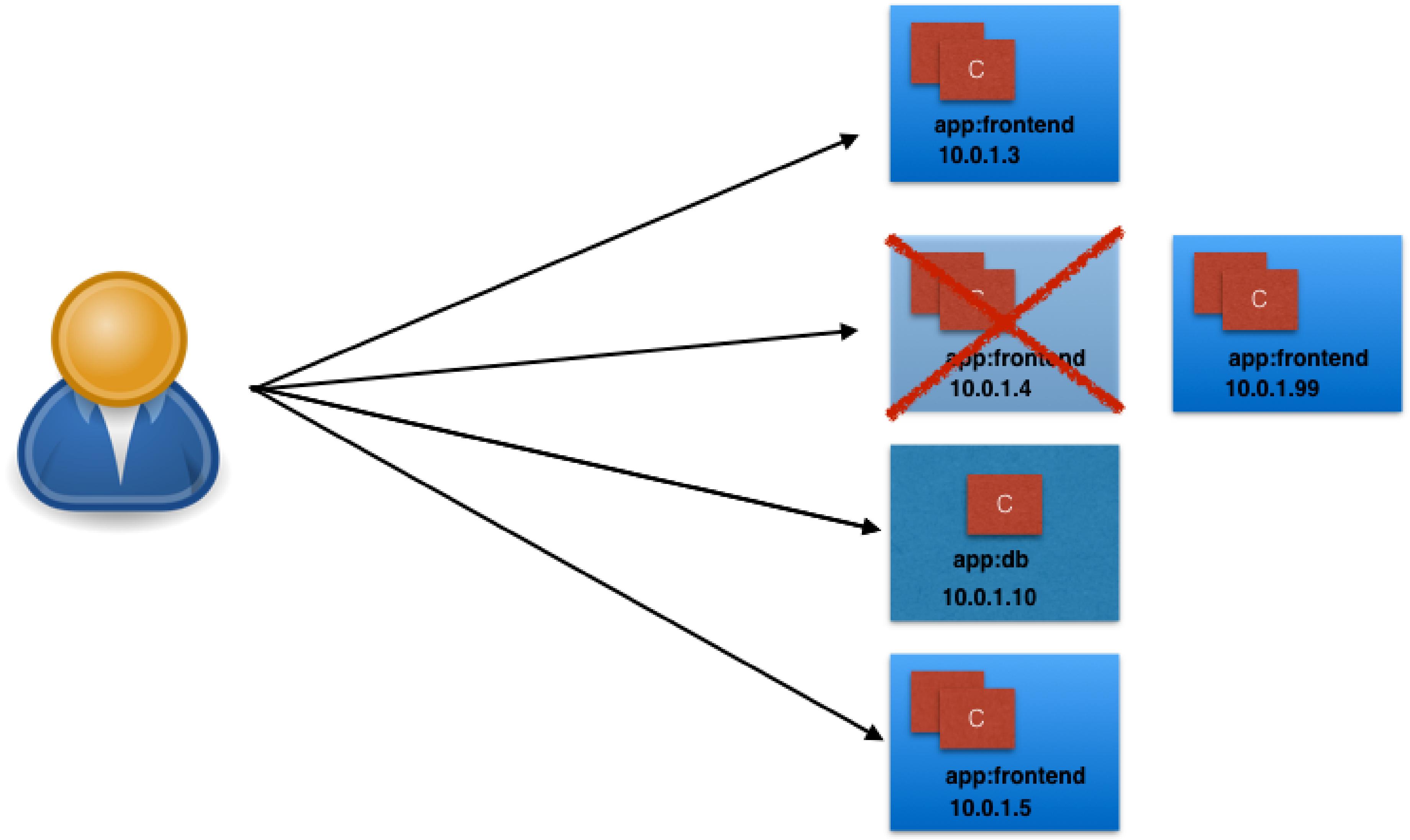
- Partition of the Kubernetes cluster into sub-clusters. The names of the resources/objects created inside a Namespace are unique, but not across Namespaces.
- The **kube-system** namespace contains the objects created by the Kubernetes system. The **default** namespace contains the objects which belong to any other namespace. **kube-public** is a special namespace, which is readable by all users and used for special purposes, like bootstrapping a cluster.

Services

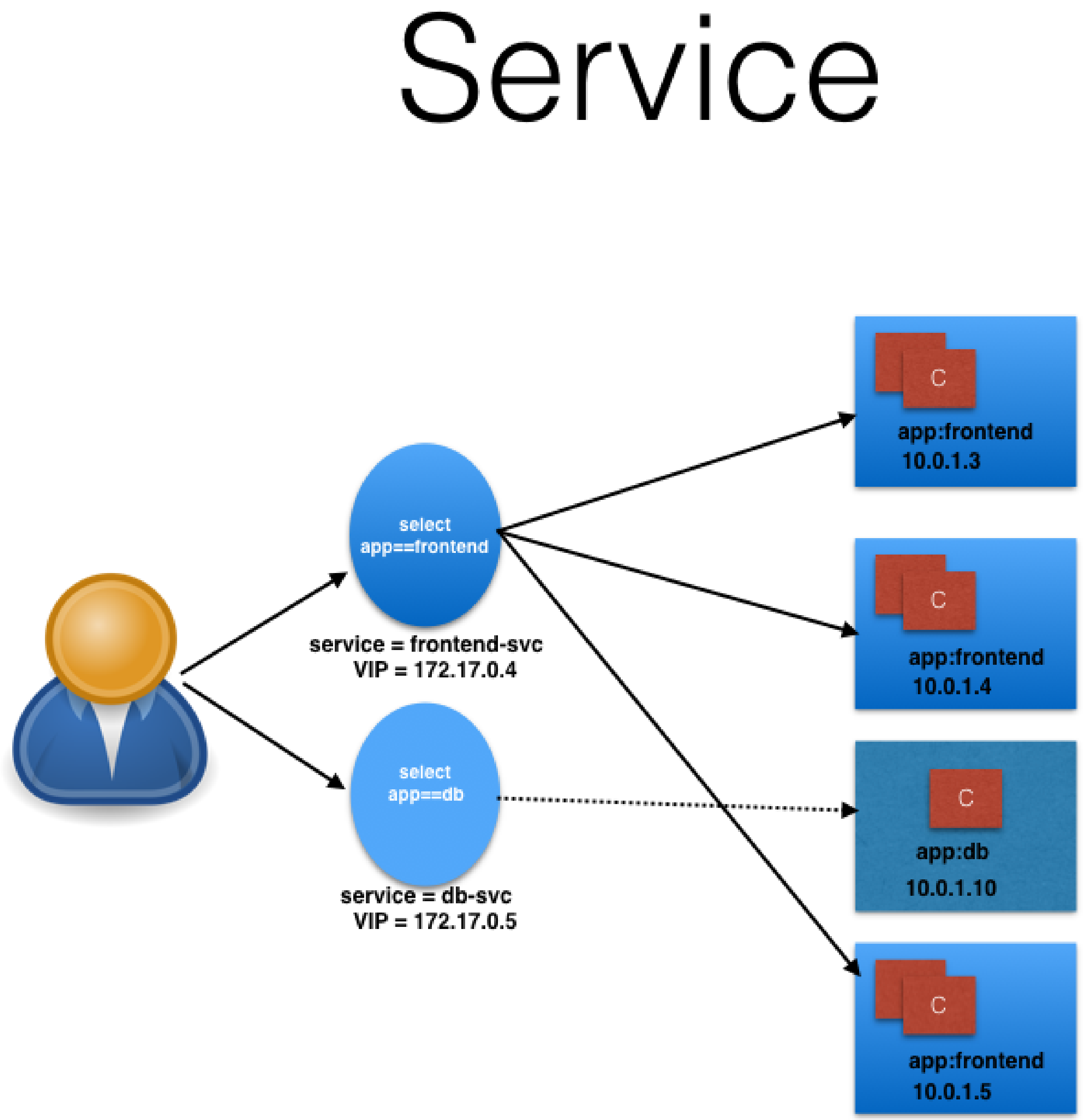
Connecting Users to Pods (1)



Connecting Users to Pods (2)



Service Name



Service Object Example

```
kind: Service
apiVersion: v1
metadata:
  name: frontend-svc
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
```

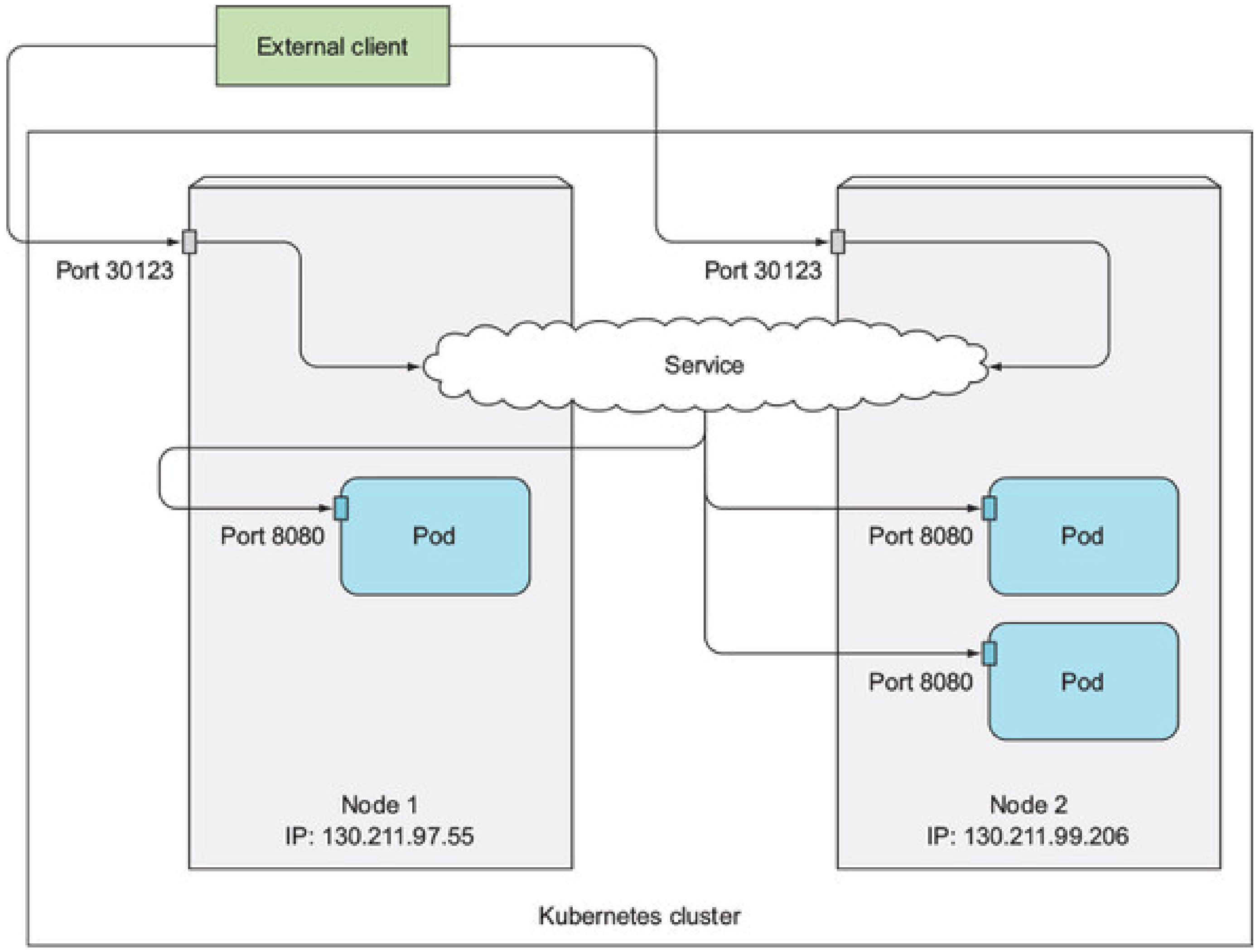
ClusterIP Service

- A ClusterIP service is the default Kubernetes service. It gives you a service inside your cluster that other apps inside your cluster can access. There is no external access.

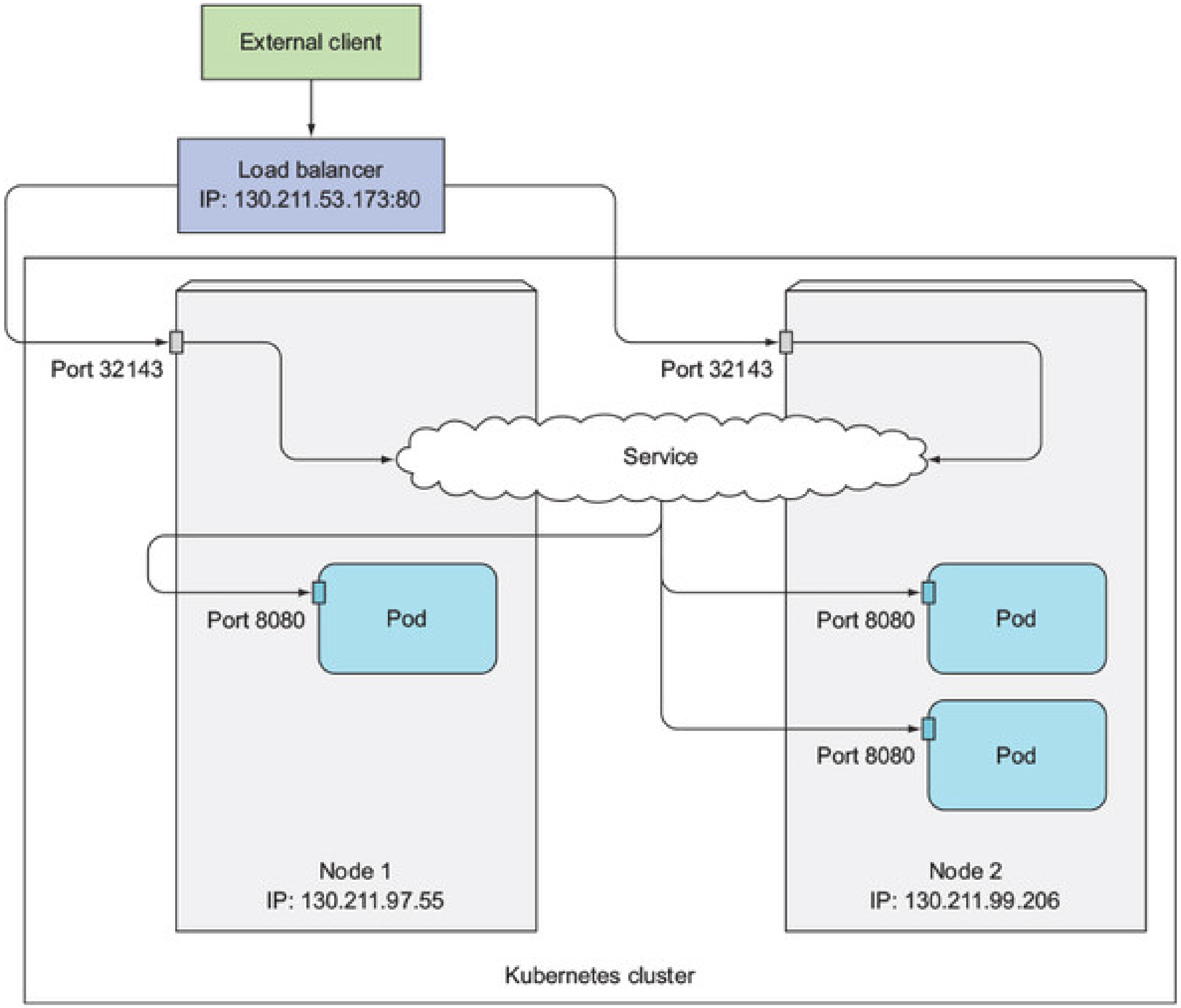
NodePort Service

- A NodePort service is the most primitive way to get external traffic directly to your service. NodePort, as the name implies, opens a specific port on all the Nodes (the VMs), and any traffic that is sent to this port is forwarded to the service.

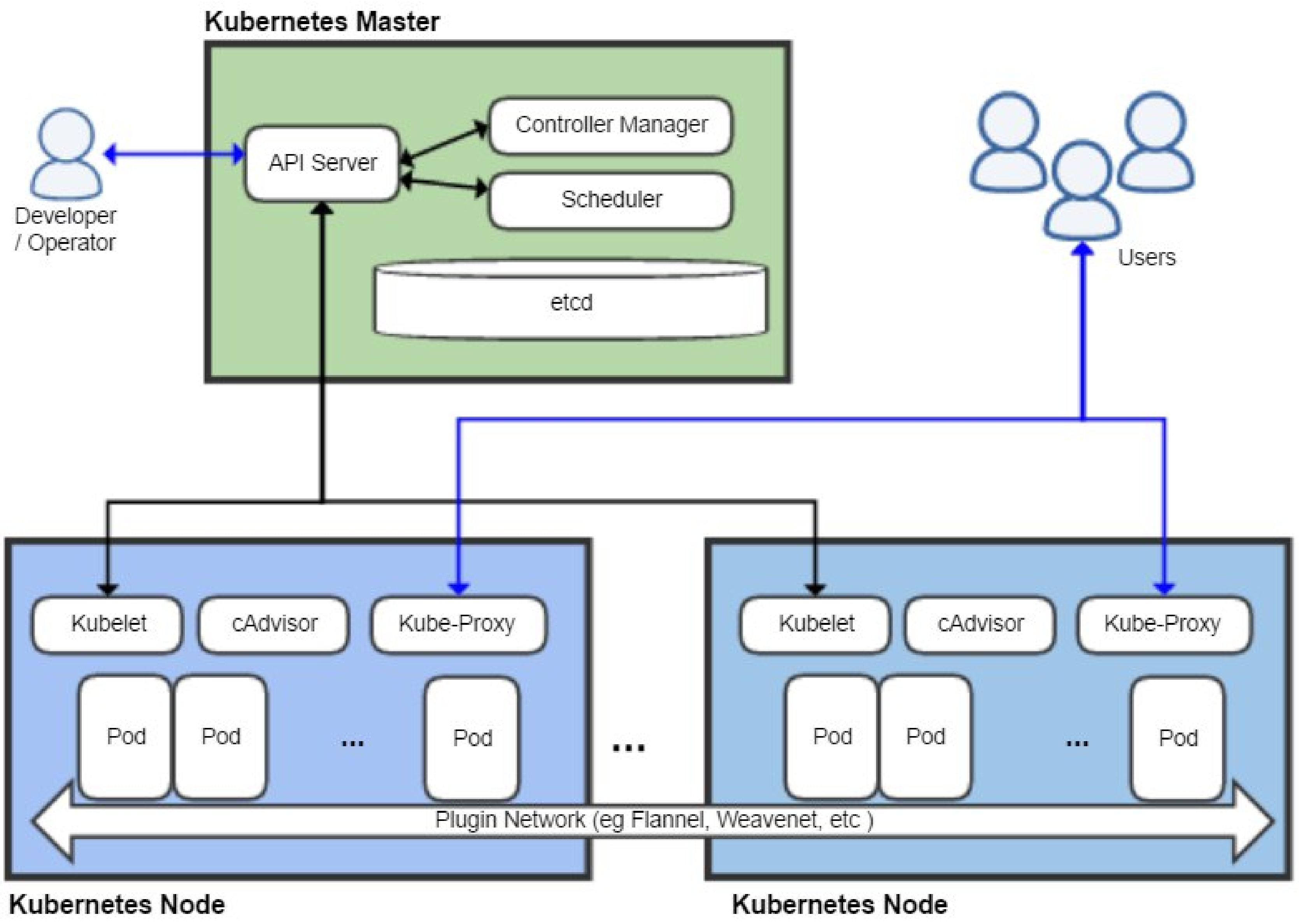
NodePort Service



LoadBalancer Service

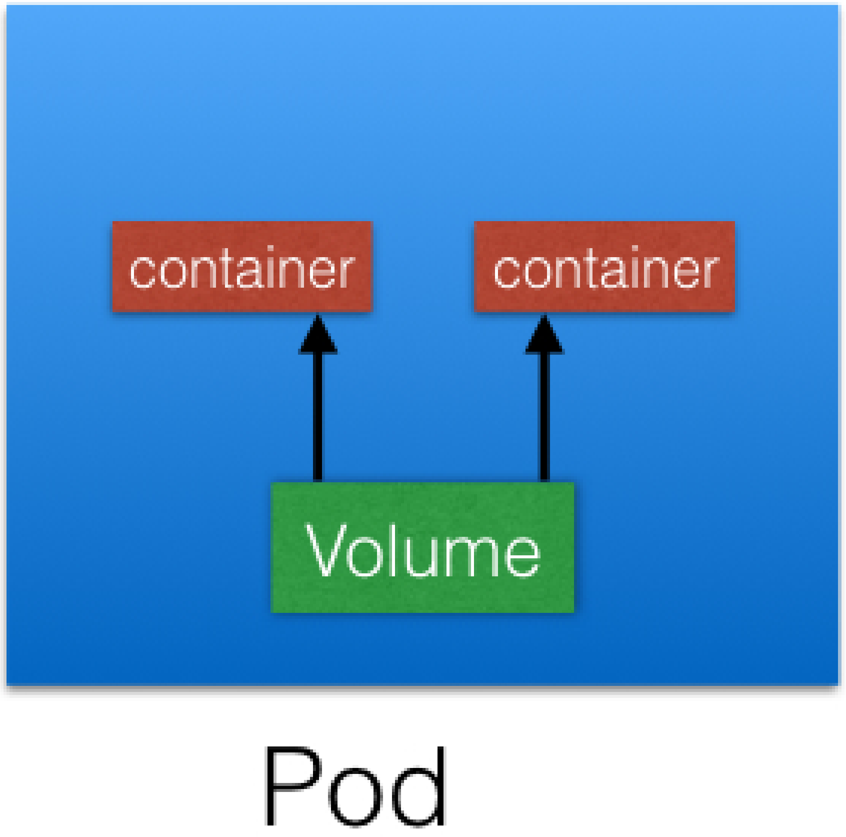


Lab Topology



Kubernetes Volume Management

Volumes

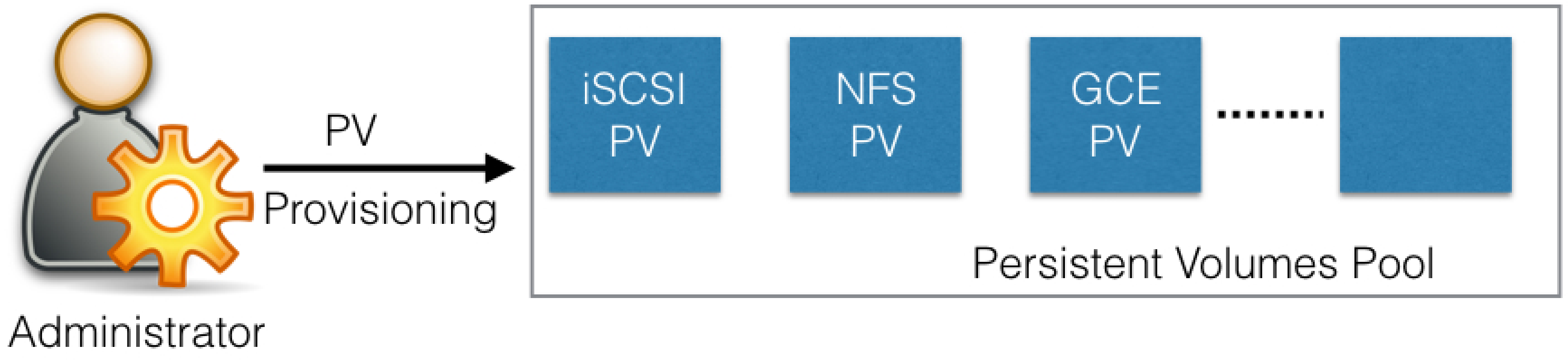


Volume Types

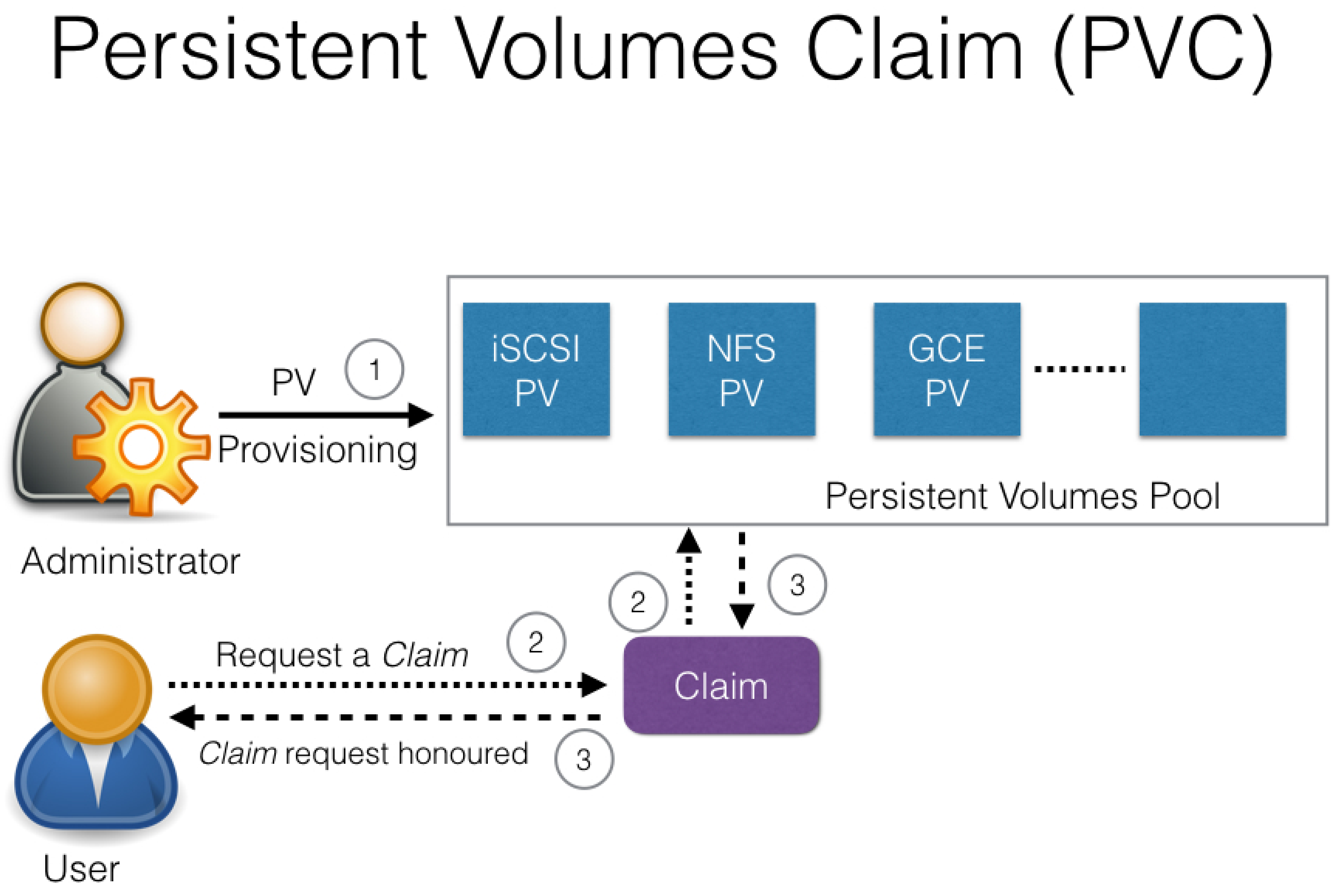
- EmptyDir
- hostPath
- gcePersistentDisk
- awsElasticBlockStore
- nfs
- iscsi
- fc
- flocker
- glusterfs
- rbd
- cephfs
- gitRepo
- secret
- persistentVolumeClaim
- downwardAPI
- projected
- FlexVolume
- AzureFileVolume
- AzureDiskVolume
- vsphereVolume
- Quobyte
- PortworxVolume
- ScaleIO
- StorageOS
- local

Persistent Volumes

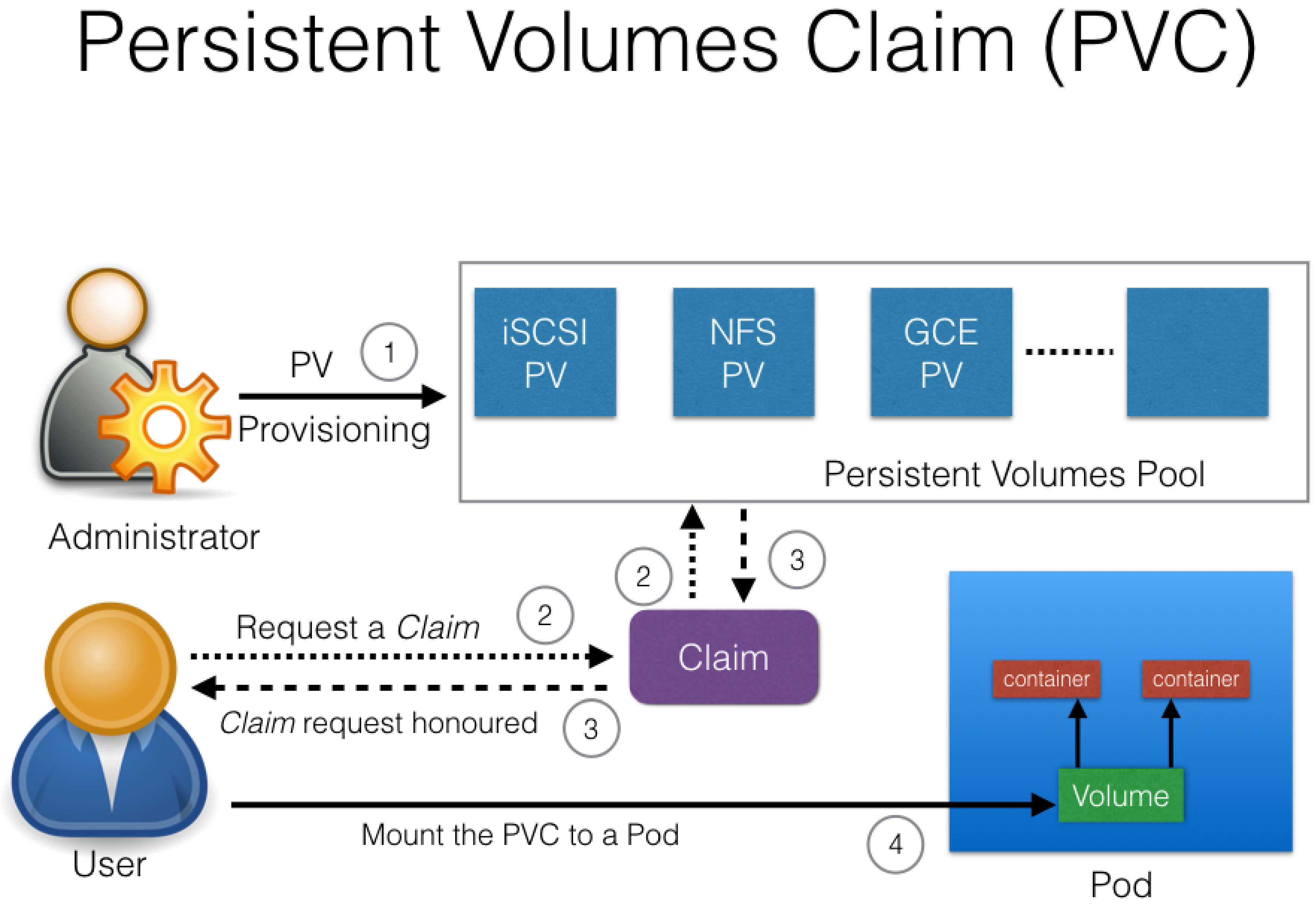
Persistent Volumes (PV)



Persistent Volumes Claim (1)

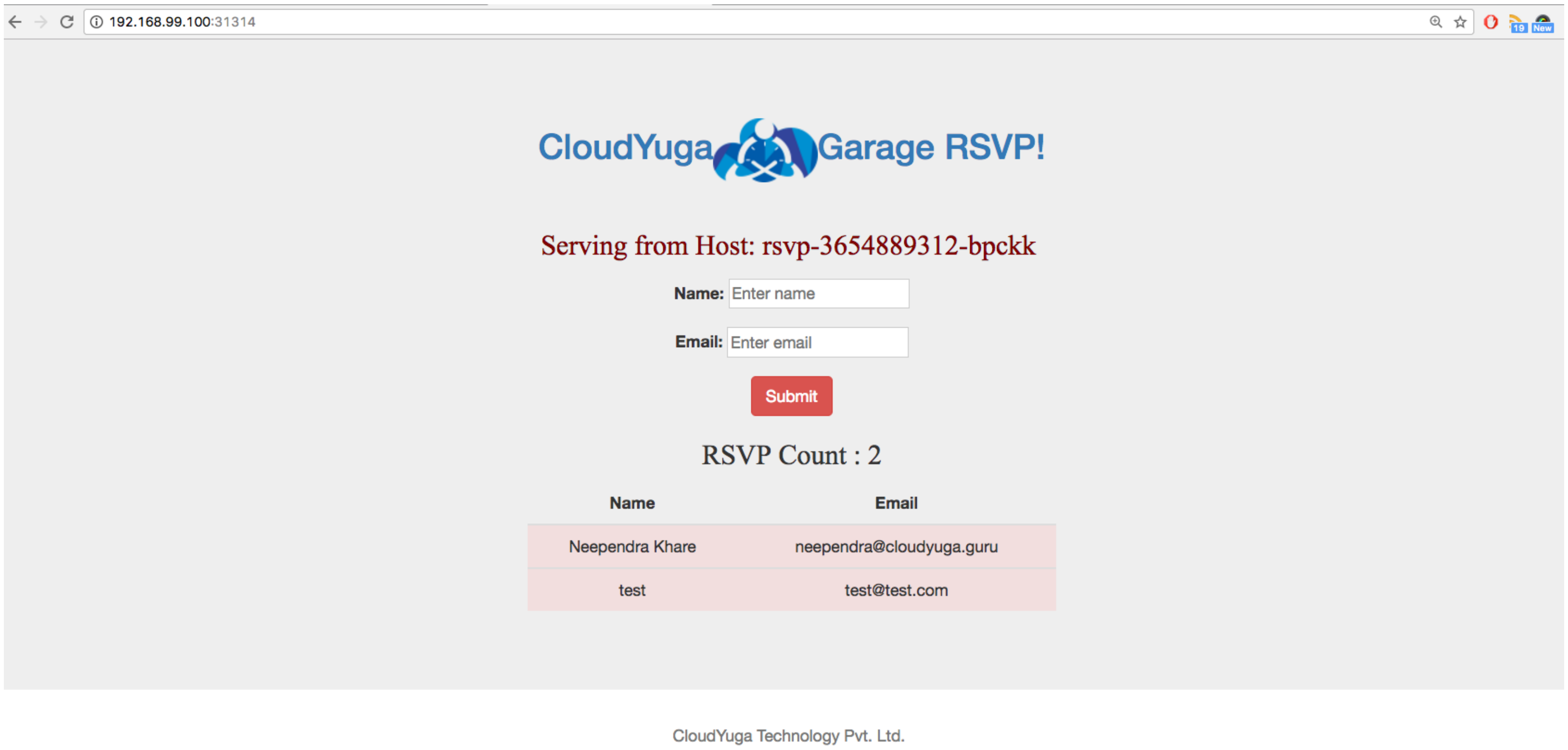


Persistent Volumes Claim (2)



Deploying a Multi-Tier Application

RSVP Application (1)


192.168.99.100:31314

CloudYuga Garage RSVP!

Serving from Host: rsvp-3654889312-bpckk

Name: Enter name

Email: Enter email

Submit

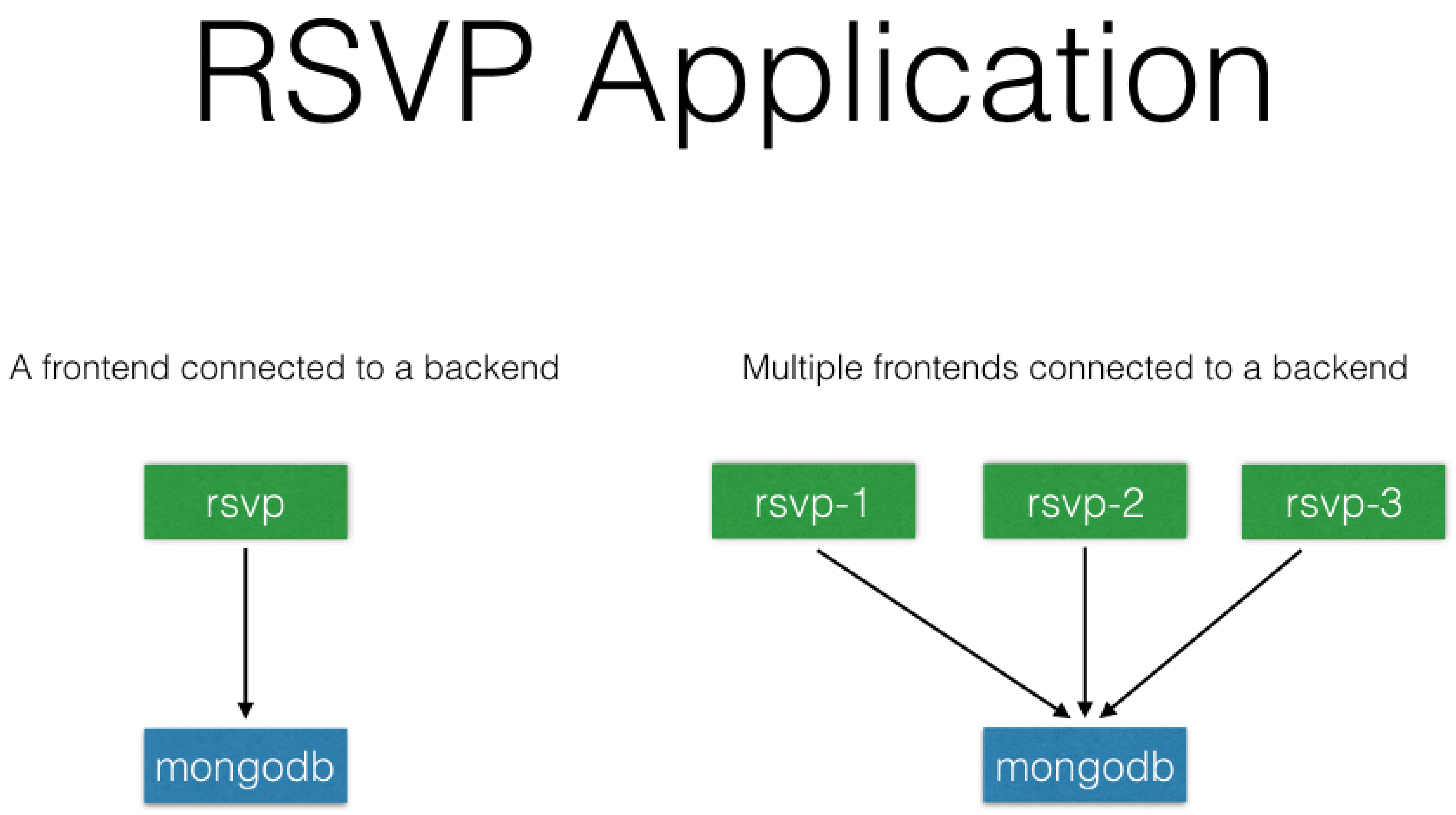
RSVP Count : 2

Name	Email
Neependra Khare	neependra@cloudyuga.guru
test	test@test.com

CloudYuga Technology Pvt. Ltd.

https://github.com/cloudyuga/rsvpapp

RSVP Application (2)


RSVP Application
A frontend connected to a backend
rsvp
mongodb
Multiple frontends connected to a backend
rsvp-1
rsvp-2
rsvp-3
mongodb

ConfigMaps and Secrets

ConfigMaps

ConfigMaps allow us to decouple the configuration details from the container image. Using ConfigMaps, we can pass configuration details as key-value pairs, which can be later consumed by Pods, or any other system components, such as controllers. We can create ConfigMaps in two ways:

- From literal values
- From files

Create ConfigMap from Literal Values

Create the ConfigMap

```
$ kubectl create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2
configmap "my-config" created
```

Get the ConfigMap Details for `my-config`

```
$ kubectl get configmaps my-config -o yaml
apiVersion: v1
data:
  key1: value1
  key2: value2
kind: ConfigMap
metadata:
  creationTimestamp: 2017-05-31T07:21:55Z
  name: my-config
  namespace: default
  resourceVersion: "241345"
  selfLink: /api/v1/namespaces/default/configmaps/my-config
  uid: d35f0a3d-45d1-11e7-9e62-080027a46057
```

Create ConfigMap from Configuration File

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: customer1
data:
  TEXT1: Customer1_Company
  TEXT2: Welcomes You
  COMPANY: Customer1 Company Technology Pct. Ltd.
```

```
$ kubectl create -f customer1-configmap.yaml
configmap "customer1" created
```

Use ConfigMap Inside Pod

```
containers:
     - name: rsvp-app
       image: teamcloudyuga/rsvpapp
       env:
       - name: MONGODB_HOST
         value: mongodb
       - name: TEXT1
         valueFrom:
           configMapKeyRef:
             name: customer1
             key: TEXT1
       - name: TEXT2
         valueFrom:
           configMapKeyRef:
             name: customer1
             key: TEXT2
       - name: COMPANY
         valueFrom:
           configMapKeyRef:
             name: customer1
             key: COMPANY
```

Secrets

With Secrets, we can share sensitive information like passwords, tokens, or keys in the form of key-value pairs. In Deployments or other system components, the Secret object is referenced, without exposing its content. The Secret data is stored as plain text inside etcd. Administrators must limit the access to the API Server and etcd.

Create Secret with Command

- Create a file with password

```
$ echo 'mysqlpassword' > password.txt
```

- Make sure there is no trailing newline in the file, after our password. To remove any newline, we can use the `tr` command:

```
$ tr -Ccsu '\n' < password.txt > .strippedpassword.txt && mv
.strippedpassword.txt password.txt
```

- Create the Secret

```
$ kubectl create secret generic my-password --from-file=password.txt
secret "my-password" created
```

'get' and 'describe' the Secret

```
$ kubectl get secret my-password
NAME            TYPE      DATA   AGE
my-password     Opaque    1      8m

$ kubectl describe secret my-password
Name:           my-password
Namespace:      default
Labels:         <none>
Annotations:    <none>

Type  Opaque

Data
====
password.txt:  13 bytes
```

Create a Secret Manually

```
$ cat password.txt | base64
bXlzcWxwYXN3b3JkCg==
```

```
apiVersion: v1
kind: Secret
metadata:
  name: my-password
type: Opaque
data:
  password: bXlzcWxwYXN3b3JkCg==
```

```
$ echo "bXlzcWxwYXN3b3JkCg==" | base64 --decode
```

Use Secrets Inside Pods

```
.....
    spec:
      containers:
      - image: wordpress:4.7.3-apache
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: my-password
              key: password.txt
.....
```

Ingress

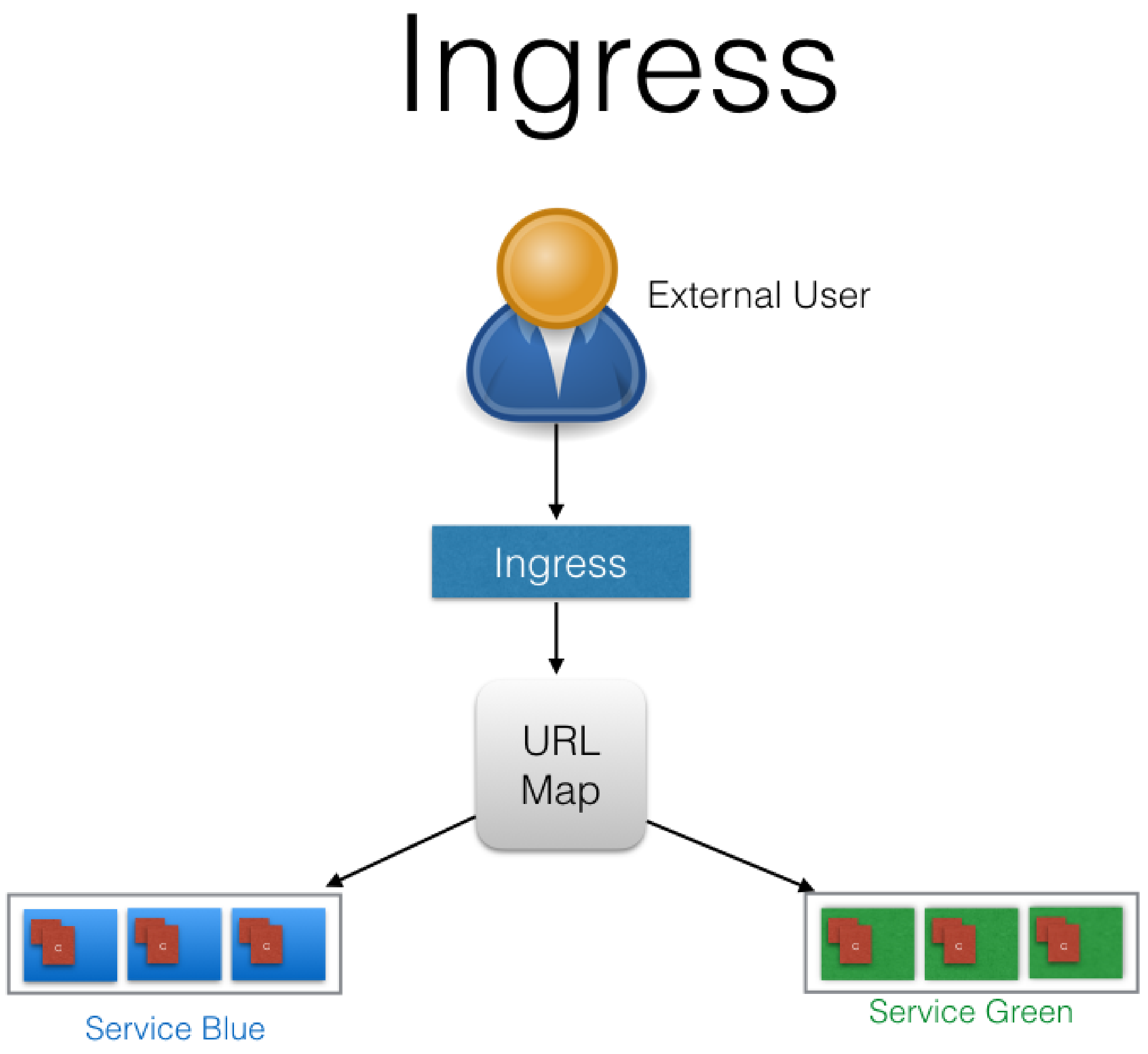
Ingress (1)

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

To allow the inbound connection to reach the cluster Services, Ingress configures a Layer 7 HTTP load balancer for Services and provides the following:

- TLS (Transport Layer Security)
- Name-based virtual hosting
- Path-based routing
- Custom rules.

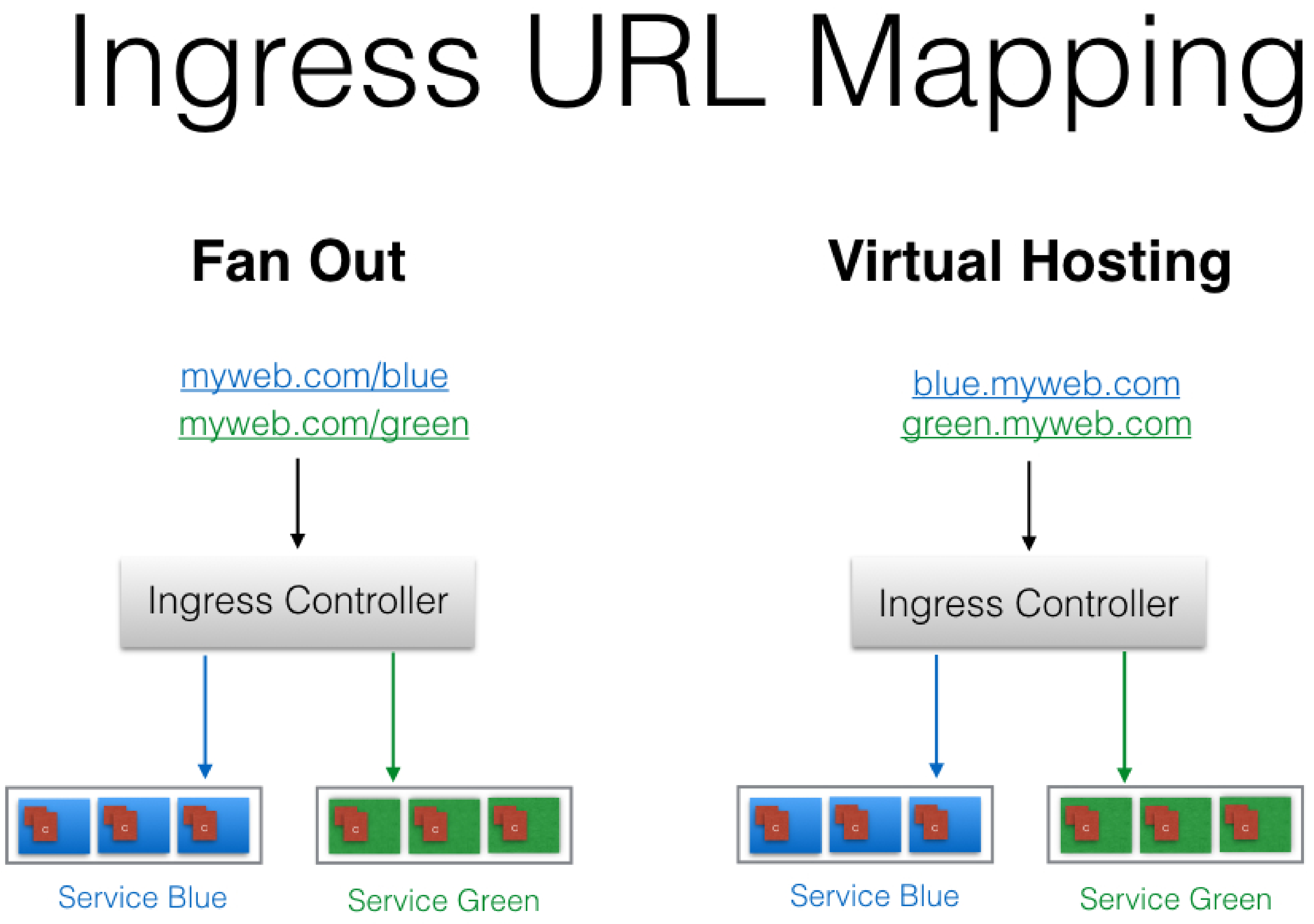
Ingress (2)


Ingress
External User
Ingress
URL Map
Service Blue
Service Green

Ingress (3)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: web-ingress
spec:
  rules:
  - host: blue.myweb.com
    http:
      paths:
      - backend:
          serviceName: blue-service
          servicePort: 80
  - host: green.myweb.com
    http:
      paths:
      - backend:
          serviceName: green-service
          servicePort: 80
```

Ingress (4)


Ingress URL Mapping
Fan Out
myweb.com/blue
myweb.com/green
Ingress Controller
Service Blue
Service Green
Virtual Hosting
blue.myweb.com
green.myweb.com
Ingress Controller
Service Blue
Service Green

Ingress Controller

An Ingress Controller is an application which watches the Master Node's API Server for changes in the Ingress resources and updates the Layer 7 load balancer accordingly. Kubernetes has different Ingress Controllers, and, if needed, we can also build our own. **GCE L7 Load Balancer** and **Nginx Ingress Controller** are examples of Ingress Controllers.

NolSatu.id