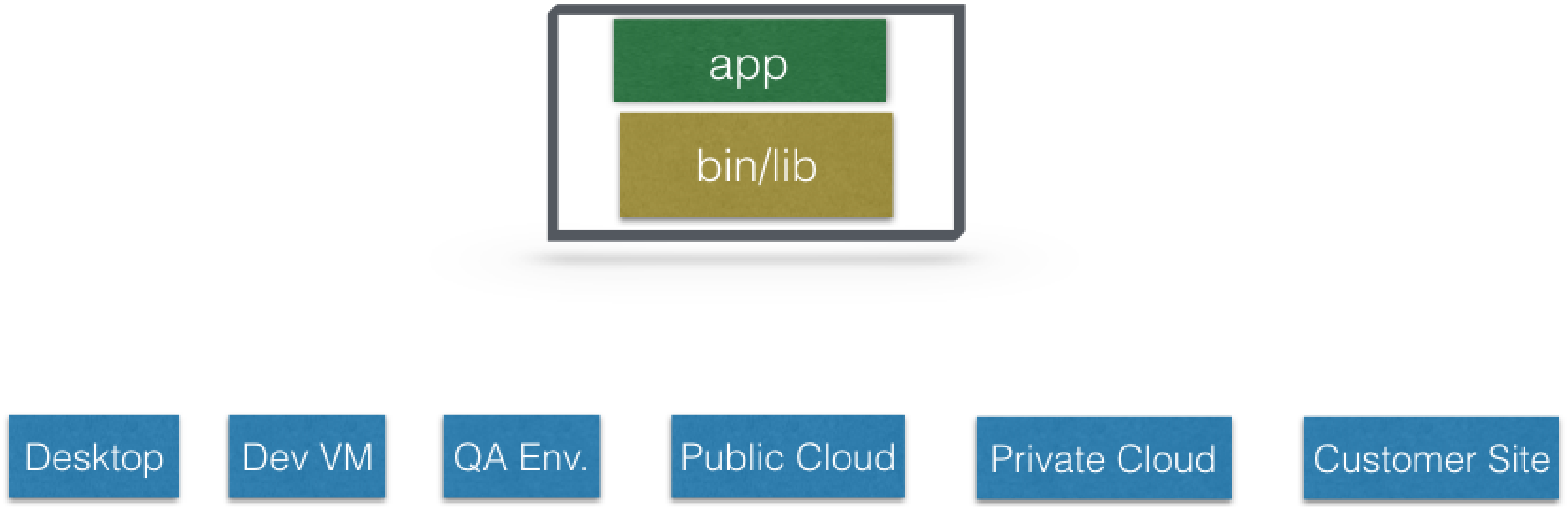
Container Orchestration

What Are Containers?



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, 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?

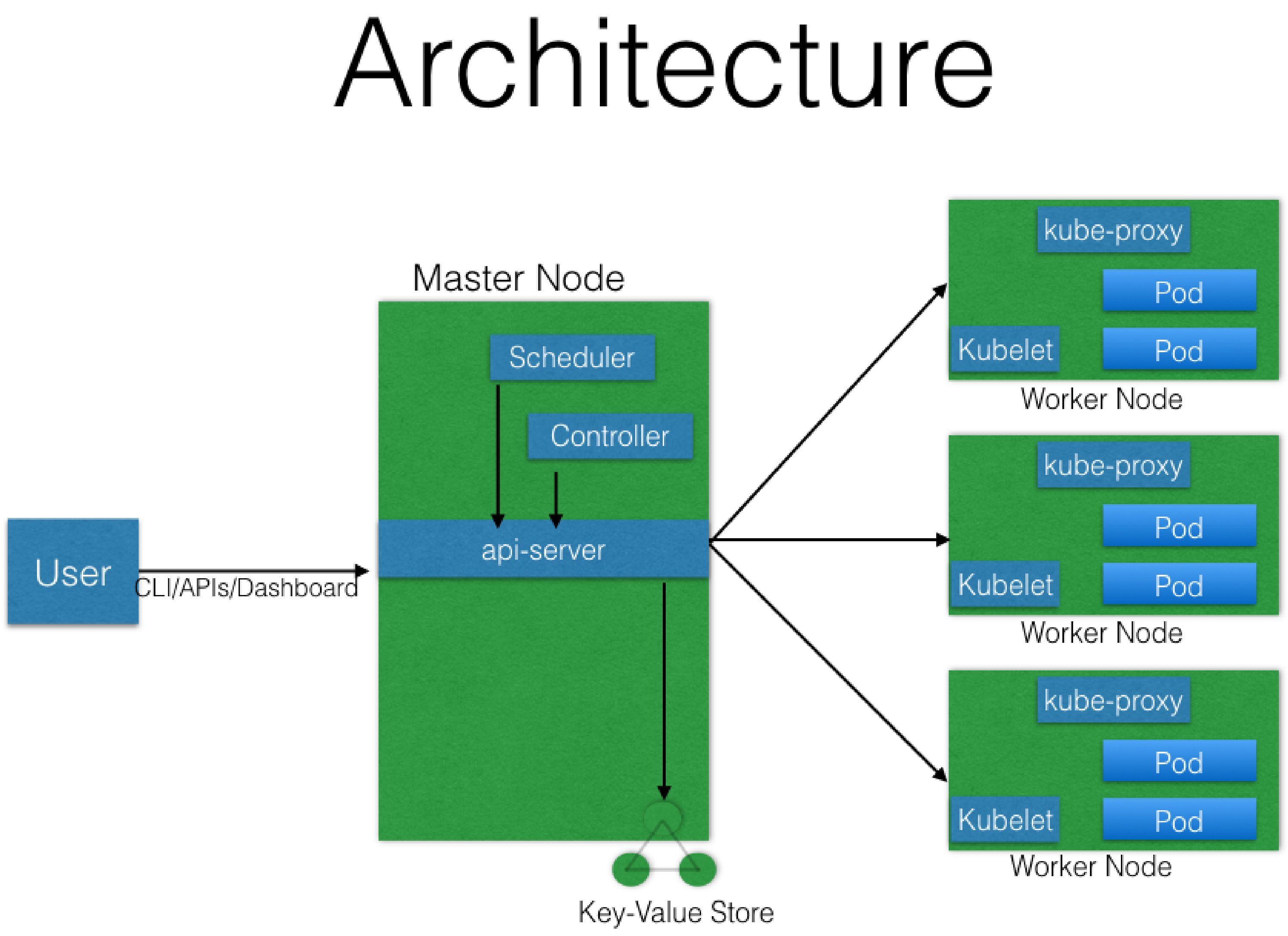
- "Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications."
- Kubernetes comes from the Greek word κυβερνήτης:, which means helmsman or ship pilot.
- Kubernetes is also referred to as k8s, as there are 8 characters between k and s.
- Kubernetes is highly inspired by the Google Borg system. It is an open source project written in the Go language, and licensed under the Apache License Version 2.0.
- 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

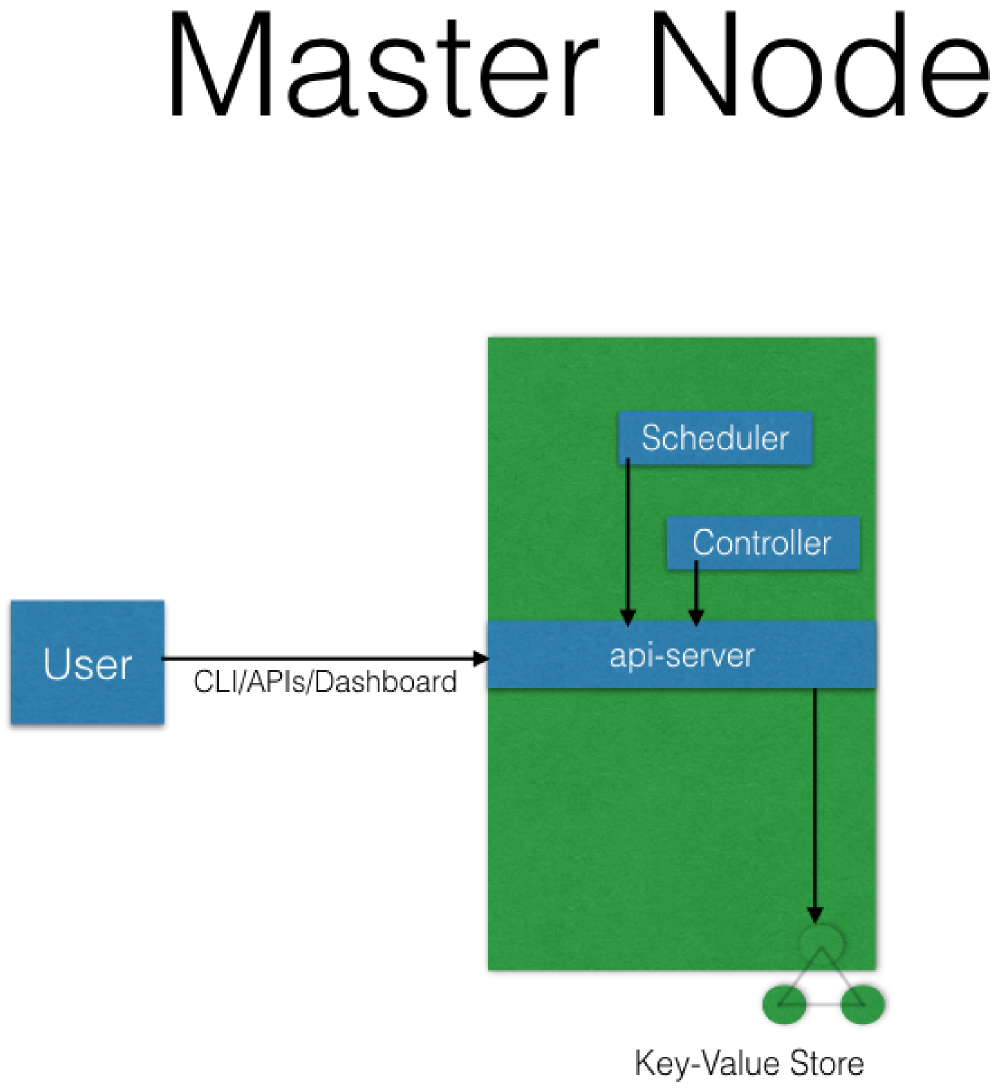
- Automatic binpacking
- Self-healing
- Horizontal scaling
- Service discovery and Load balancing
- Automated rollouts and rollbacks
- Secrets and configuration management
- Storage orchestration
- Batch execution

Kubernetes Architecture

Kubernetes Architecture



Master Node

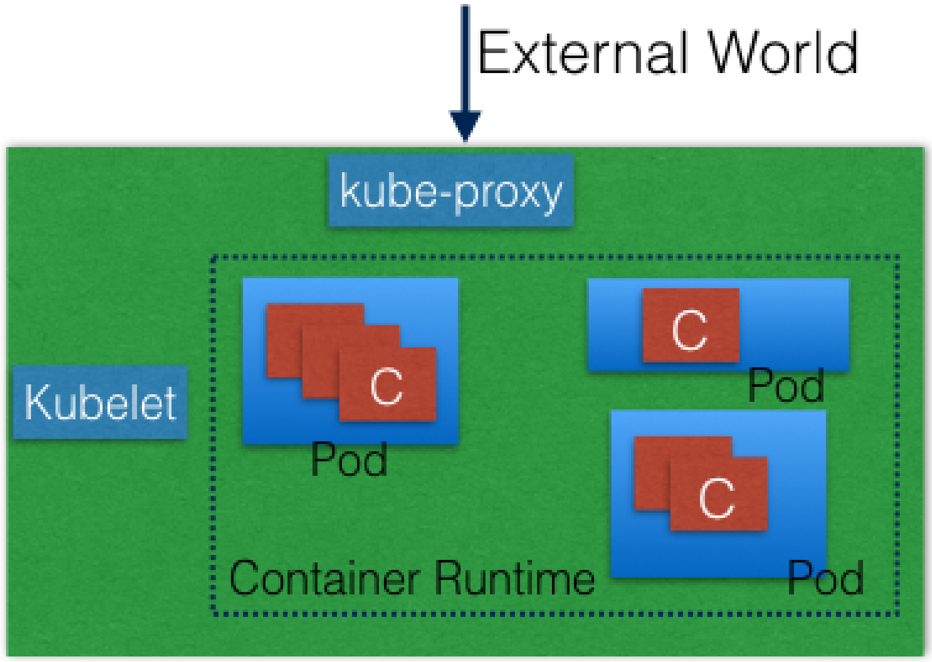

Master Node
User
CLI/APIs/Dashboard
Scheduler
Controller
api-server
Key-Value Store

Master Node Components

- API Server
- Scheduler
- Controller Manager
- etcd

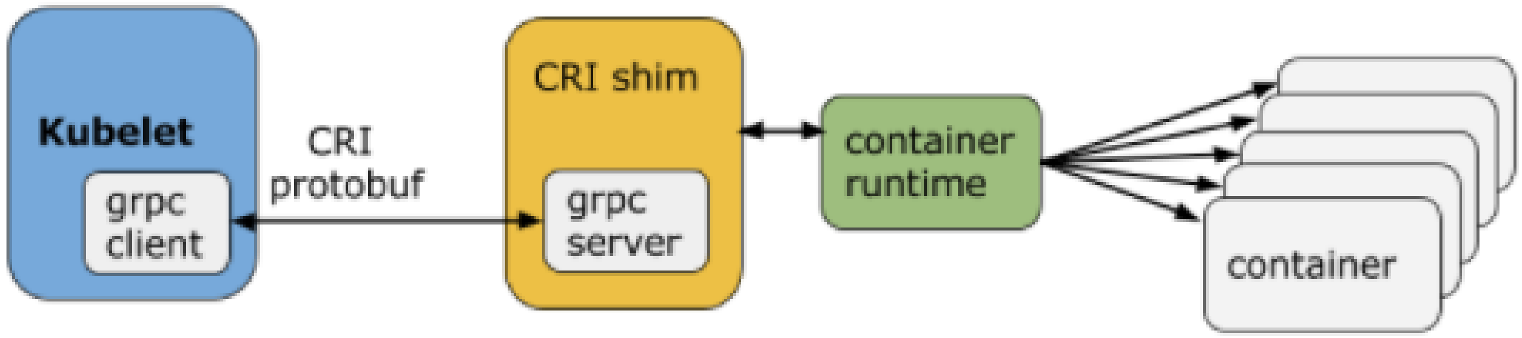
Worker Node

Worker Node



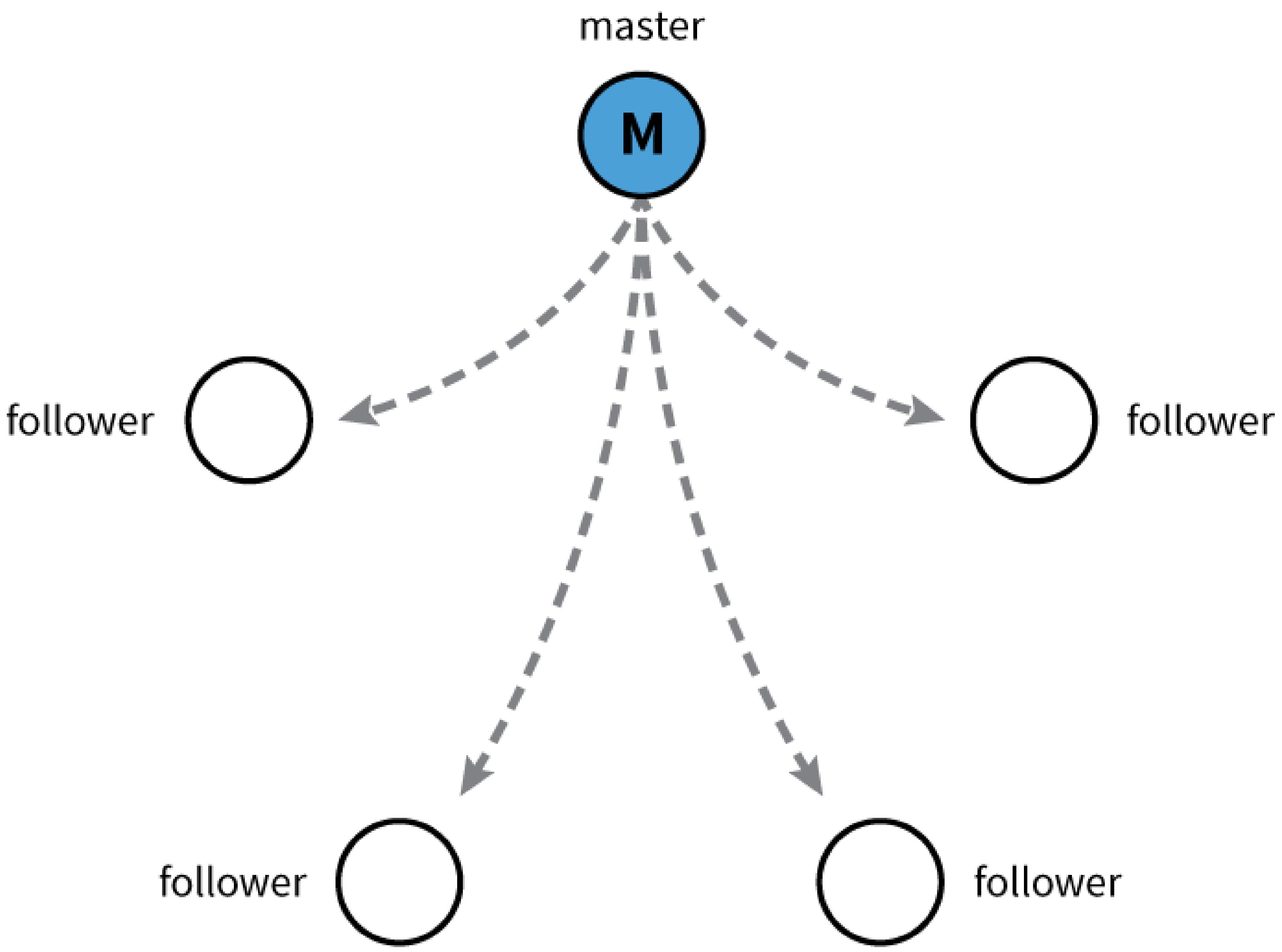
Worker Node Components

- Container Runtime
- Kubelet



- kube-proxy

State Management with etcd

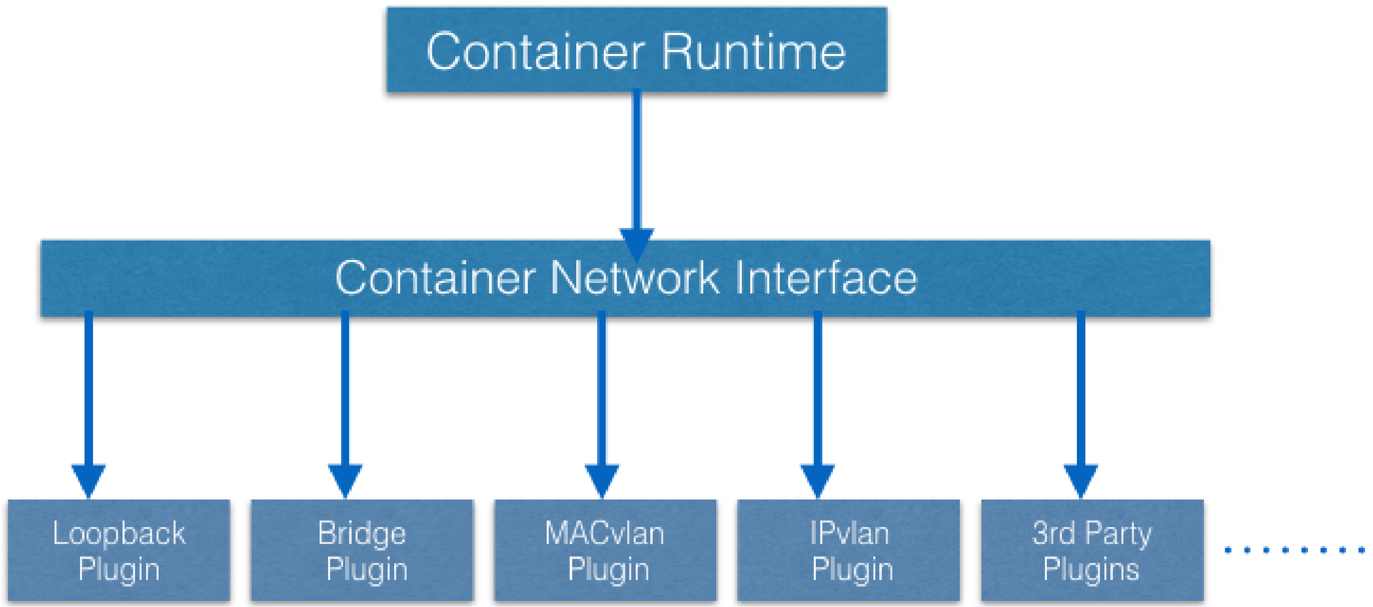


Network Setup Challenges

- A unique IP is assigned to each Pod
- Containers in a Pod can communicate to each other
- The Pod is able to communicate with other Pods in the cluster
- If configured, the application deployed inside a Pod is accessible from the external world.

Assigning a Unique IP Address to Each Pod

Container Network Interface (CNI)



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 Single-Node Installation
- Single-Node etcd, Single-Master, and Multi-Worker Installation
- Single-Node etcd, Multi-Master, and Multi-Worker Installation
- Multi-Node etcd, Multi-Master, and Multi-Worker Installation

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

Turn-key Cloud Solutions

- Google Compute Engine
- AWS EC2
- Azure
- Alibaba Cloud
- CenturyLink Cloud
- IBM Bluemix
- Stackpoint.io

On-Premise Solutions

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

Accessing Kubernetes

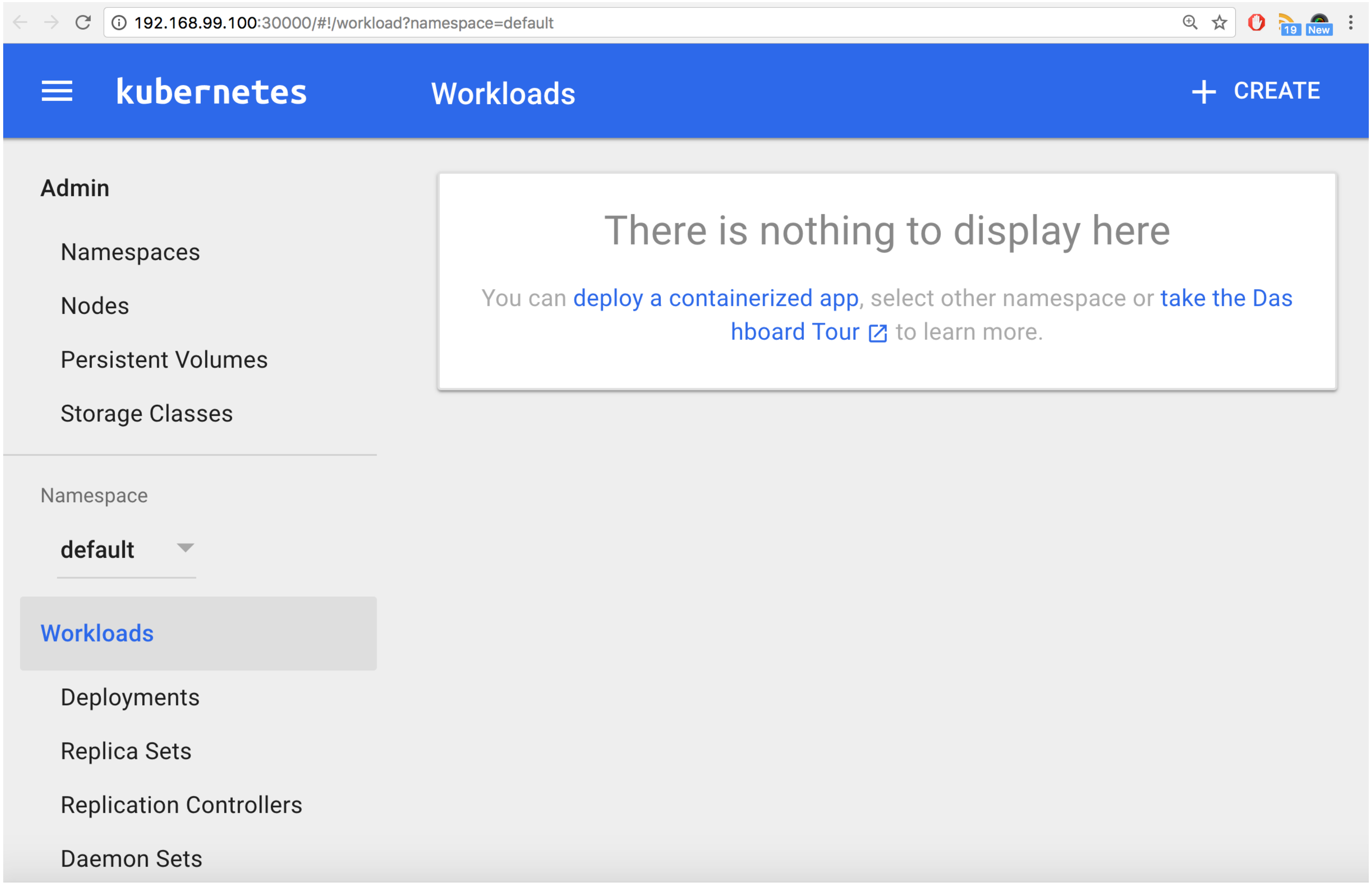
Access Kubernetes Cluster

- 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 Dashboard Tour to learn more.

kubectl proxy

- HTTP Proxy to Access the Kubernetes API
- 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

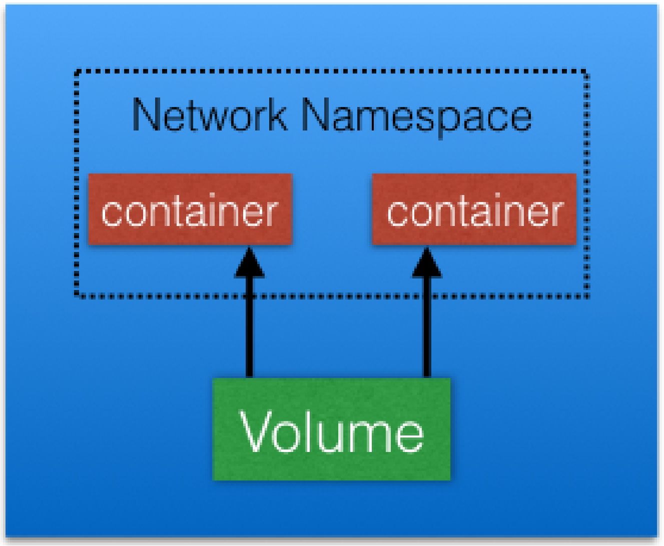
Kubernetes Building Blocks

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

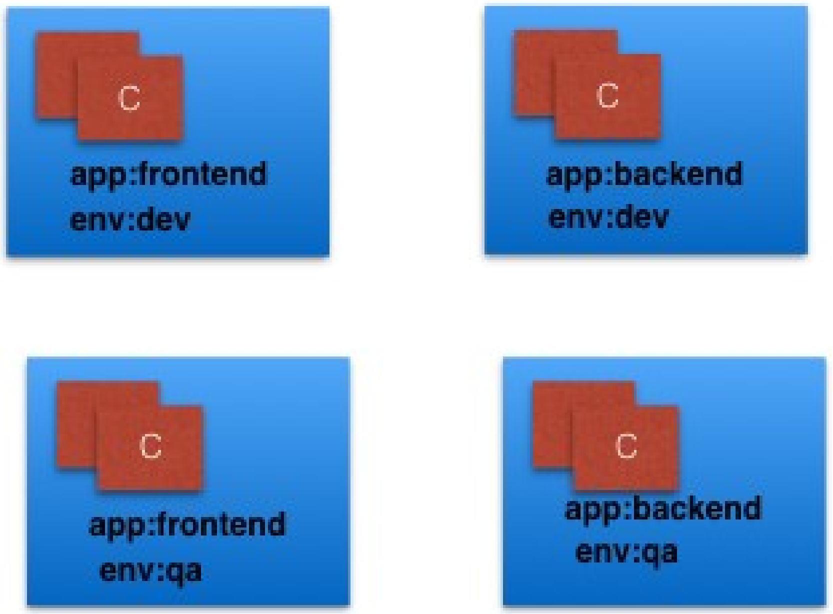
Pods



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

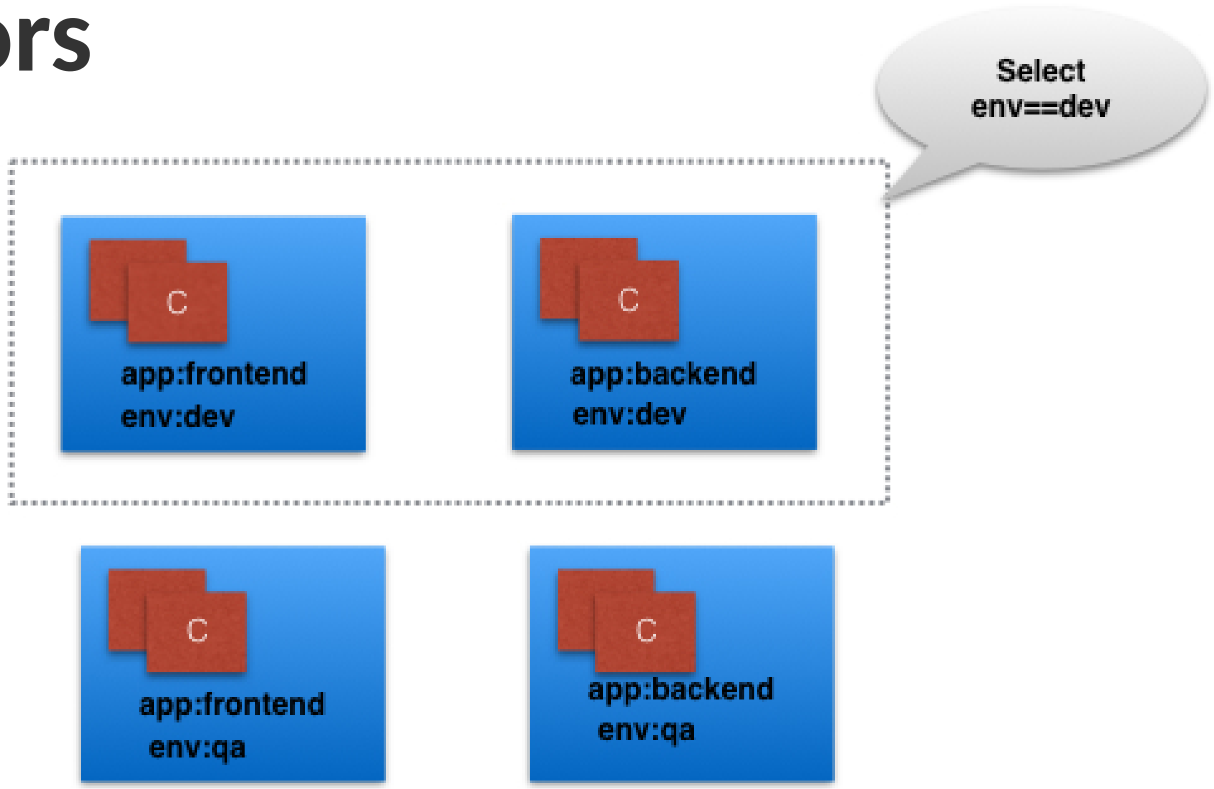
- Are scheduled together on the same host
- Share the same network namespace
- Mount the same external storage (Volumes)

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.

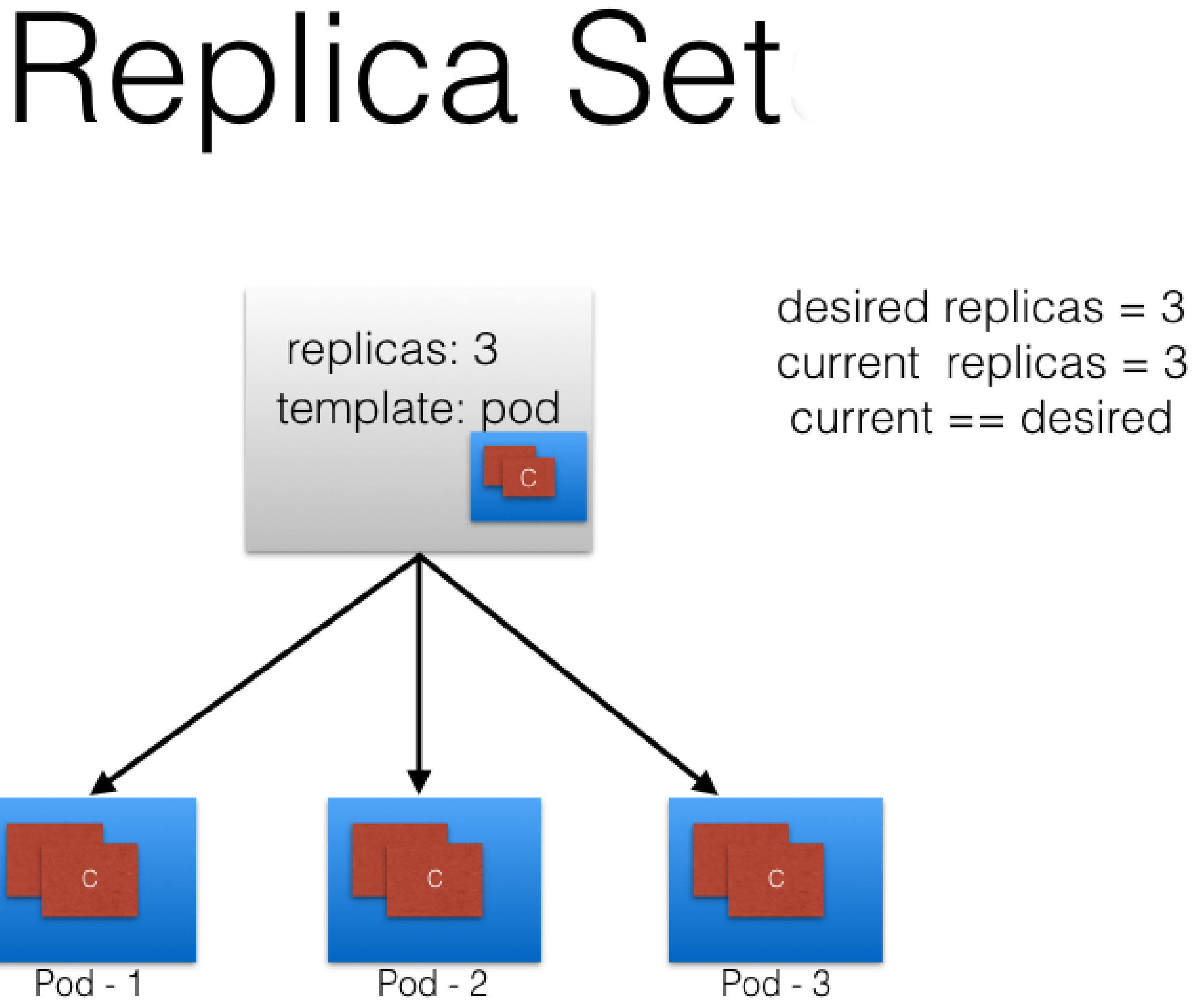
Replication Controllers (rc)

A ReplicationController (rc) is a controller that is part of the Master Node's Controller Manager. It makes sure the specified number of replicas for a Pod is running at any given point in time.

Replica Sets

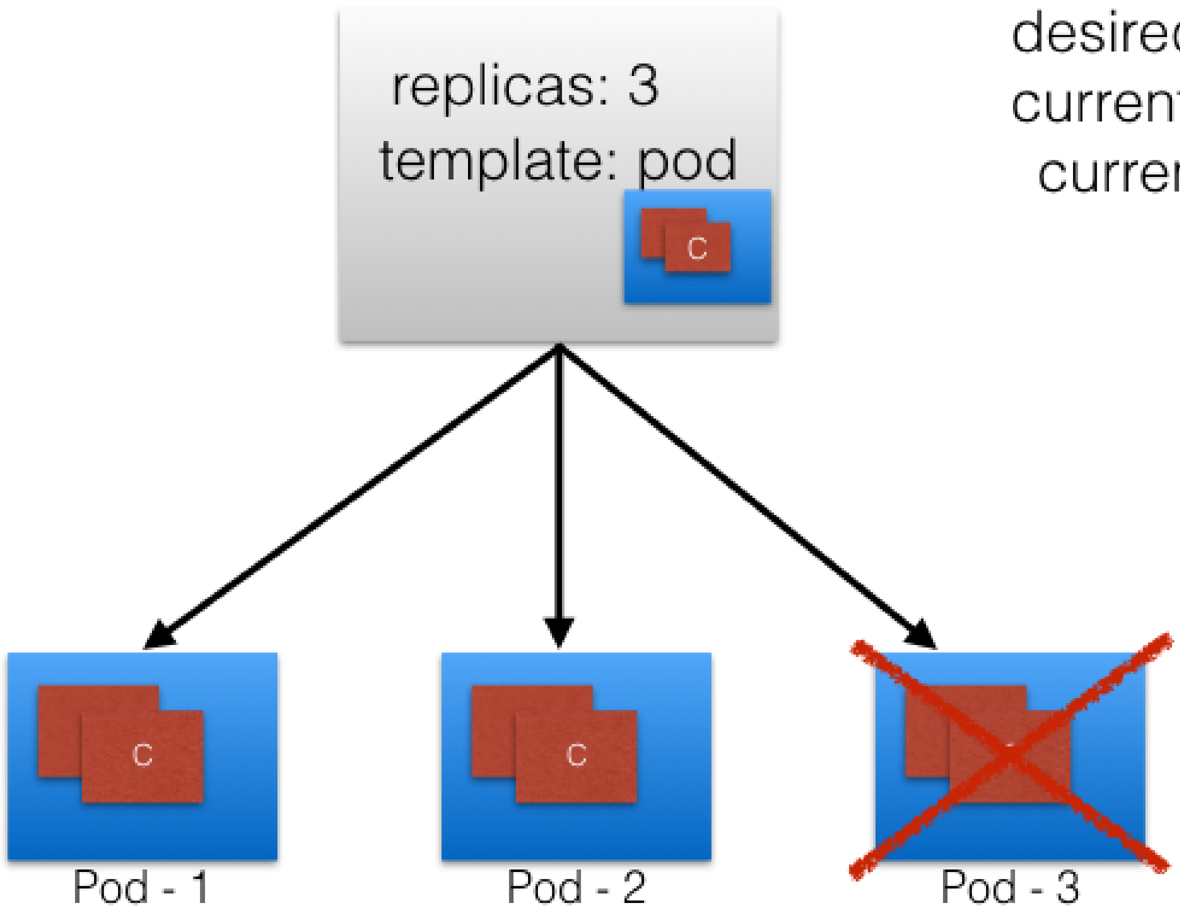
A Replica Set (rs) is the next-generation Replication Controller. Replica Sets support both equality- and set-based Selectors, whereas ReplicationControllers only support equality-based Selectors.

ReplicaSet Operations (1)



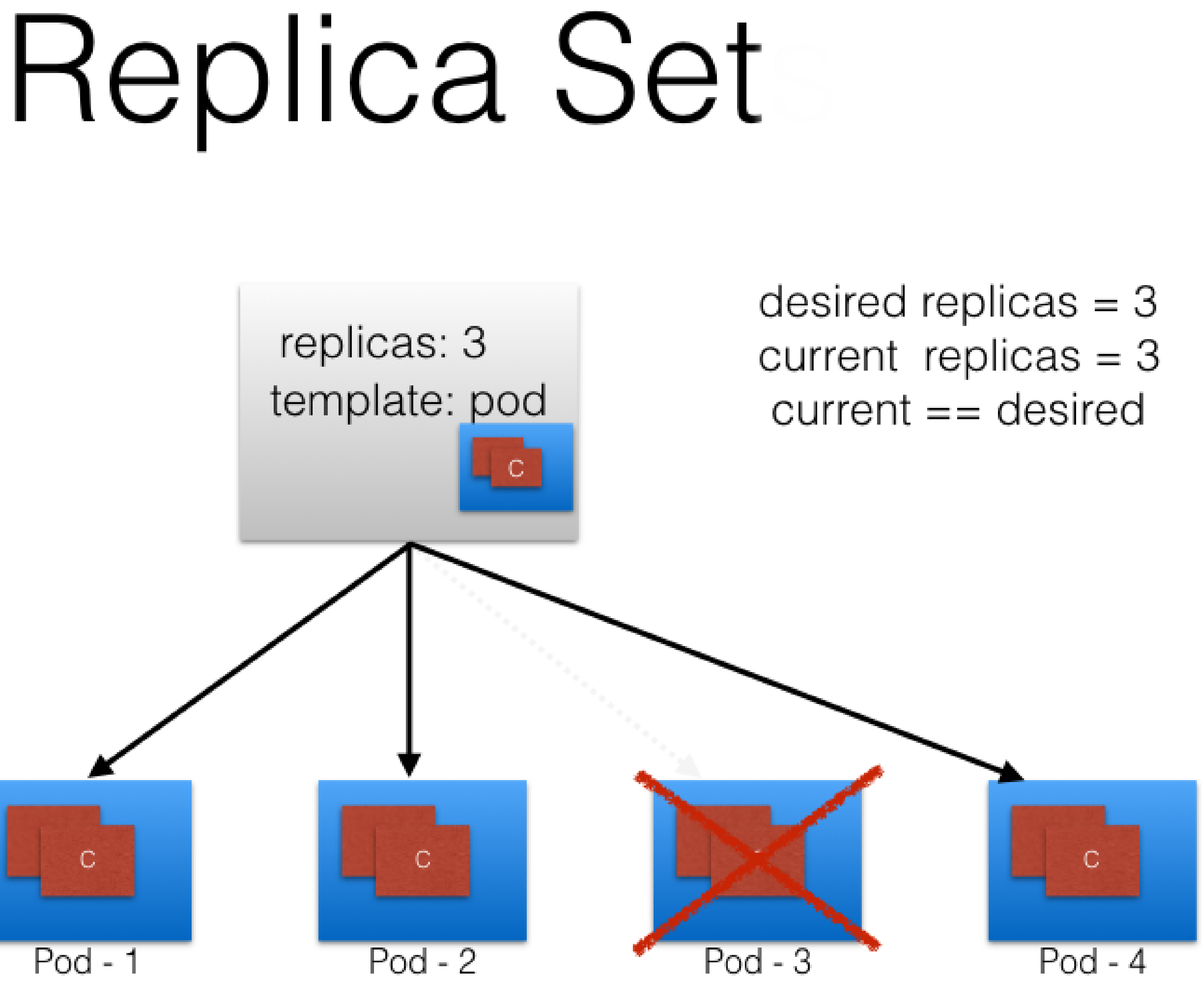
ReplicaSet Operations (2)

Replica Set



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

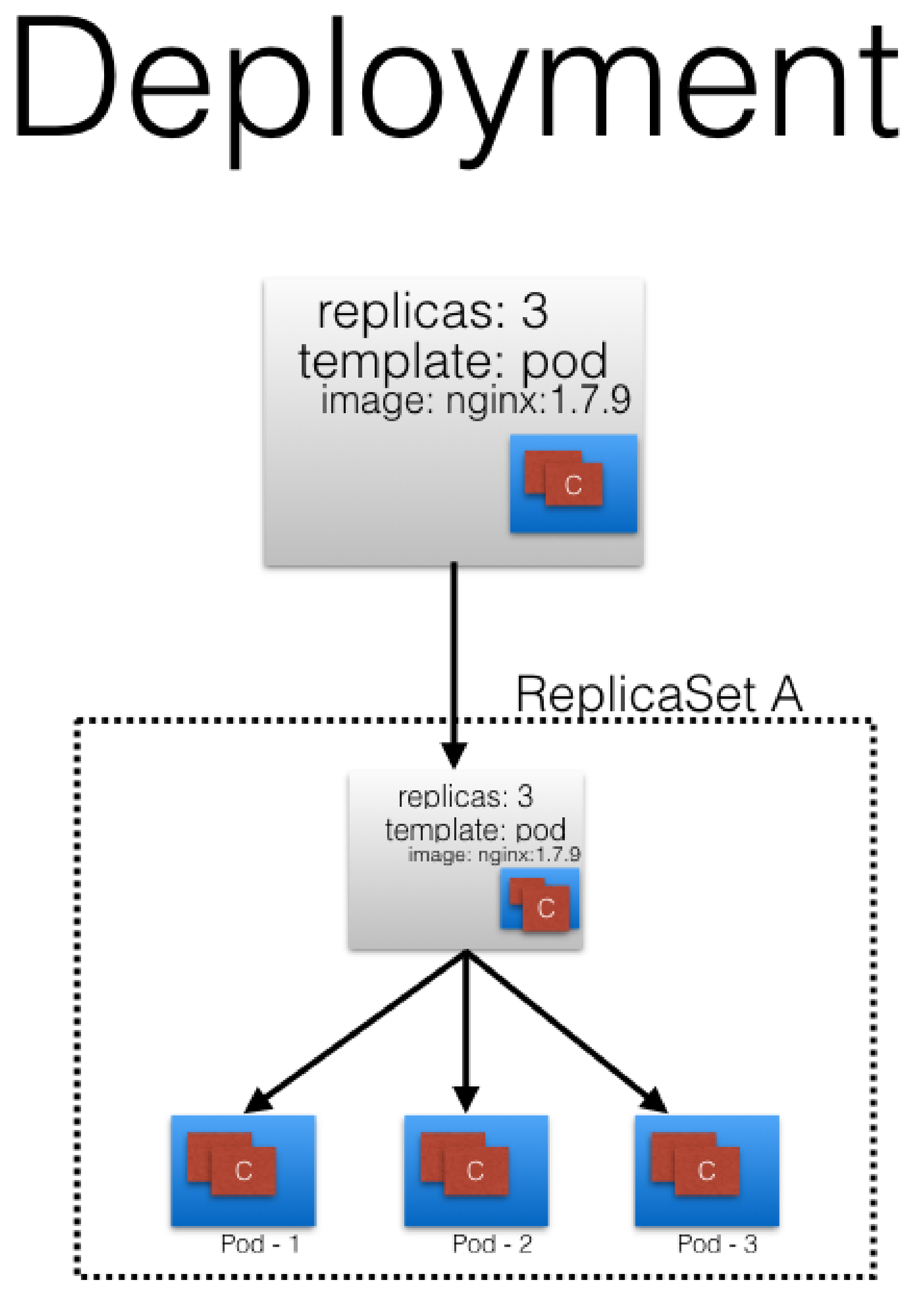
ReplicaSet Operations (3)



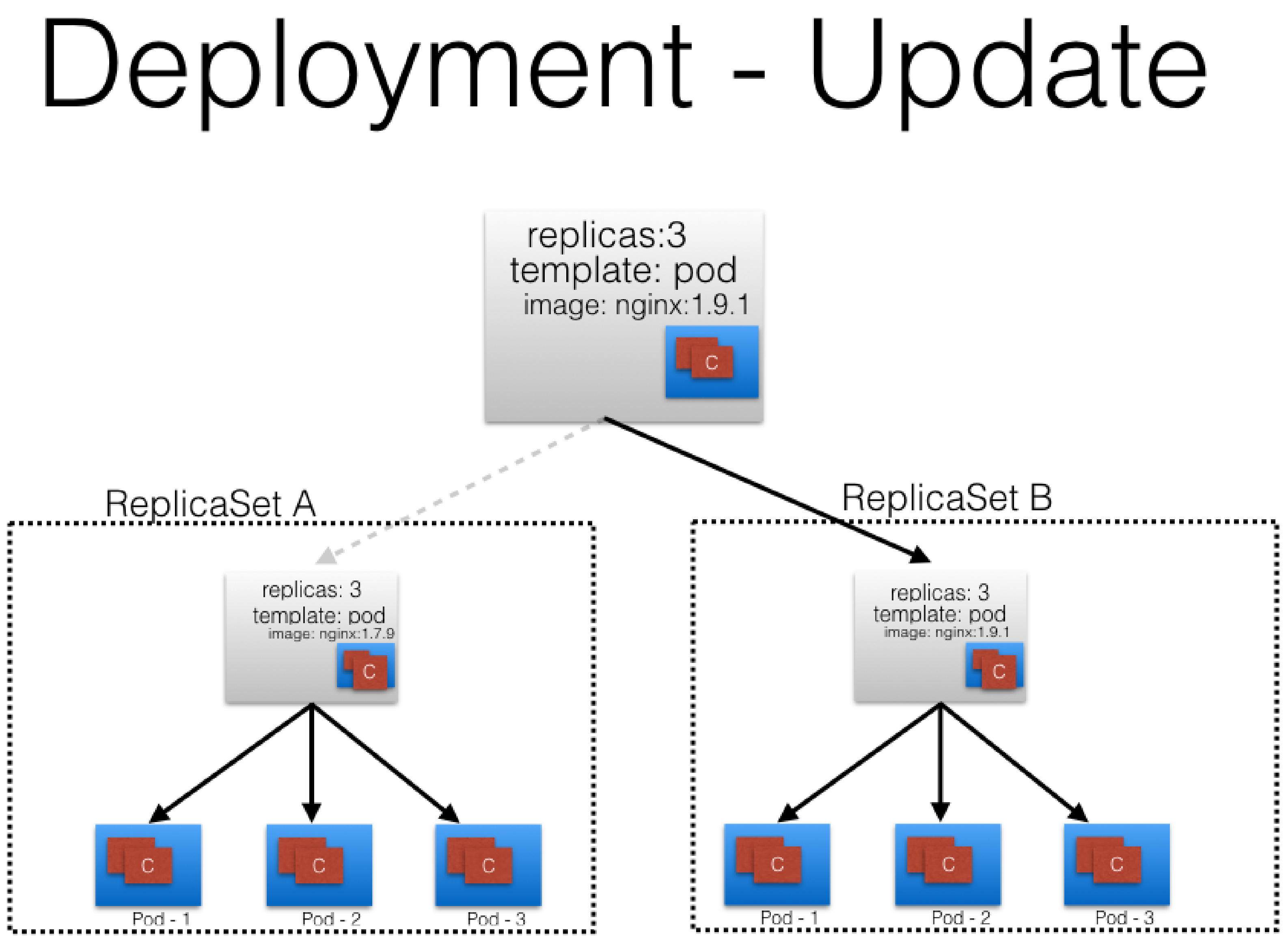
Deployments

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.

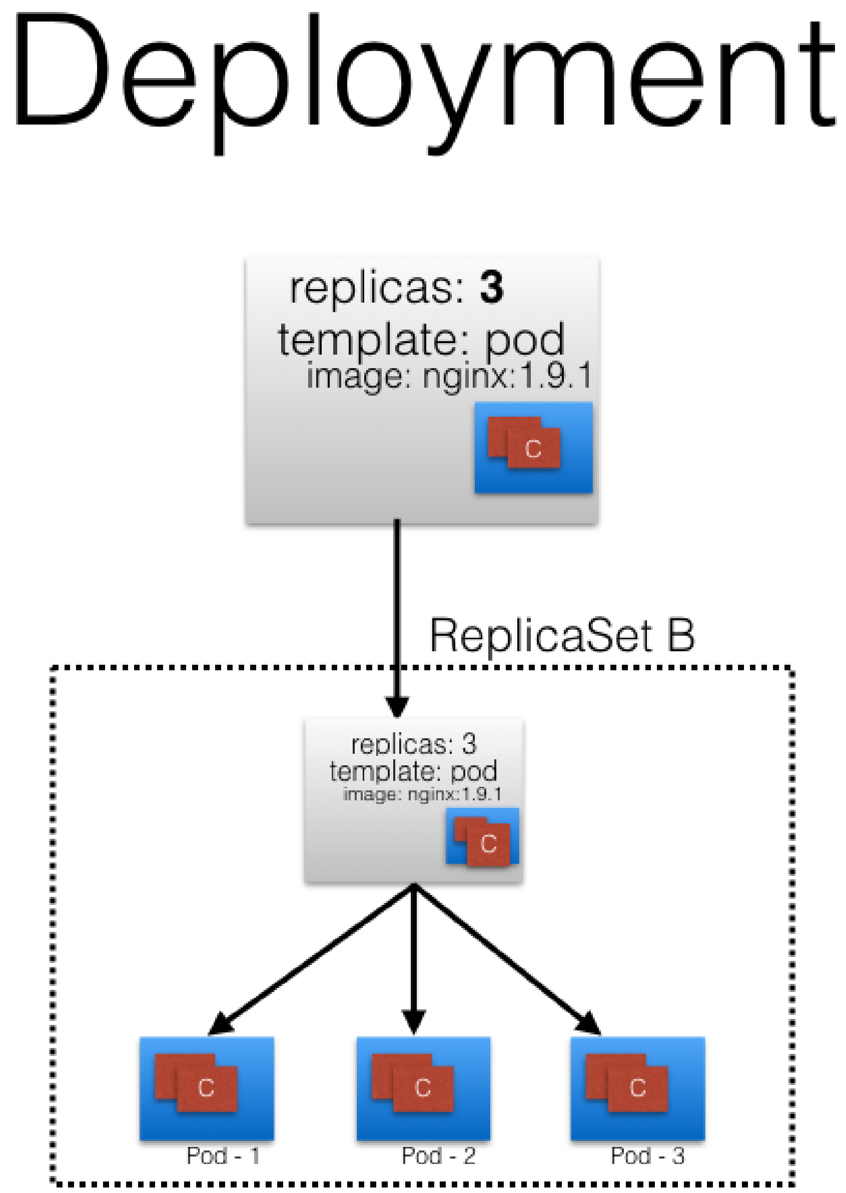
Deployment Operations (1)



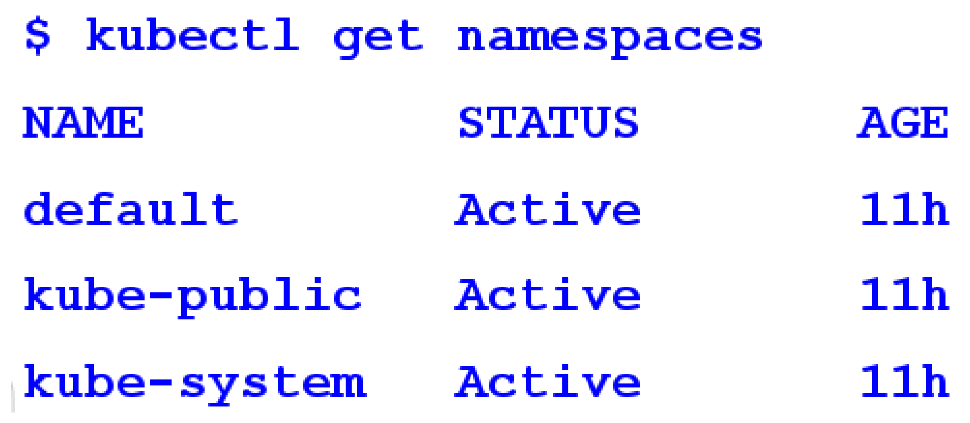
Deployment Operations (2)



Deployment Operations (3)



Namespaces

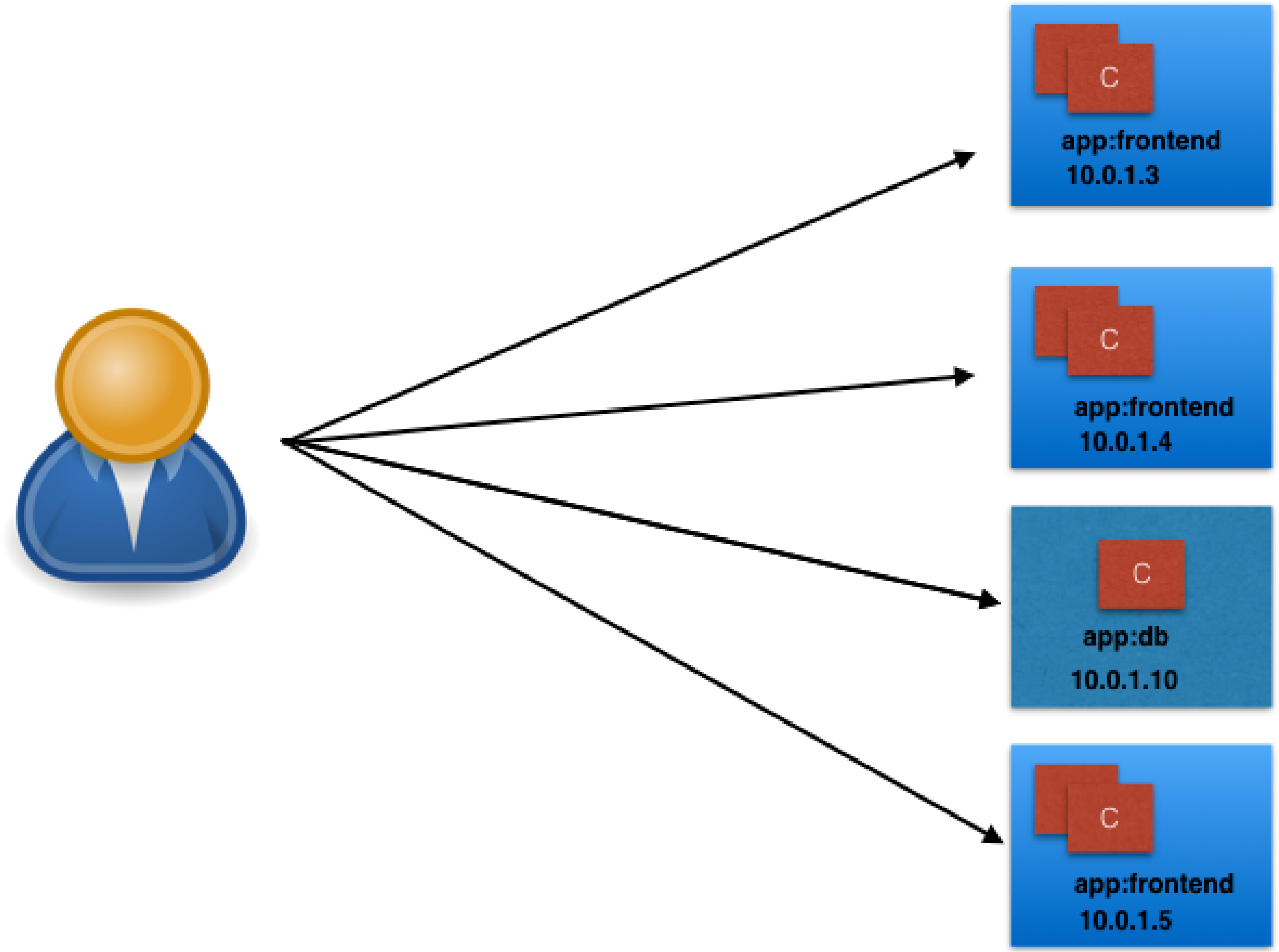


```
$ kubectl get namespaces
NAME           STATUS     AGE
default        Active     11h
kube-public    Active     11h
kube-system    Active     11h
```

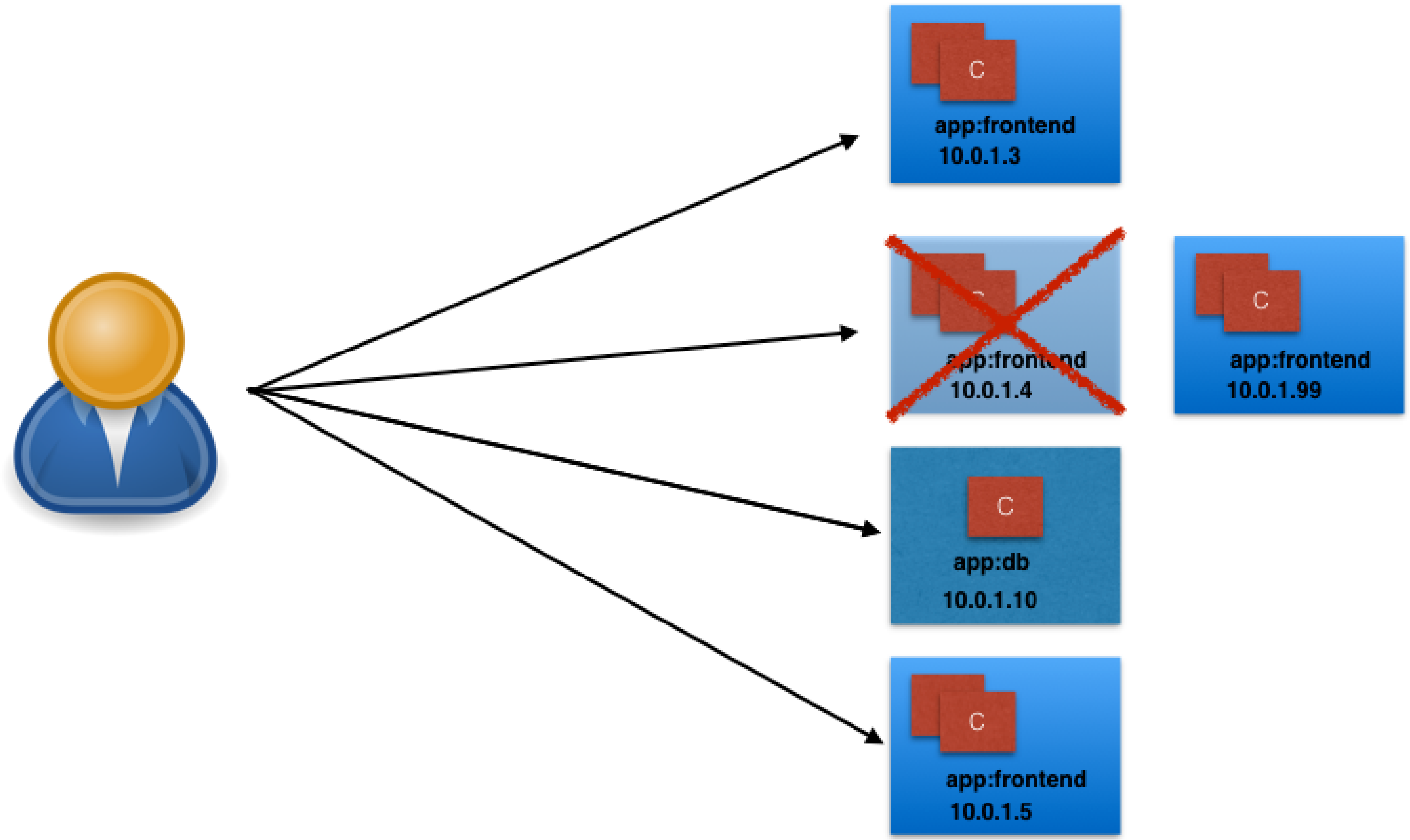
- Partition of the Kubernetes cluster into sub-clusters. The names of the resources/objects created inside a Namespace are unique, but not across Namespaces.
- The **kube-system** namespace contains the objects created by the Kubernetes system. The **default** namespace contains the objects which belong 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

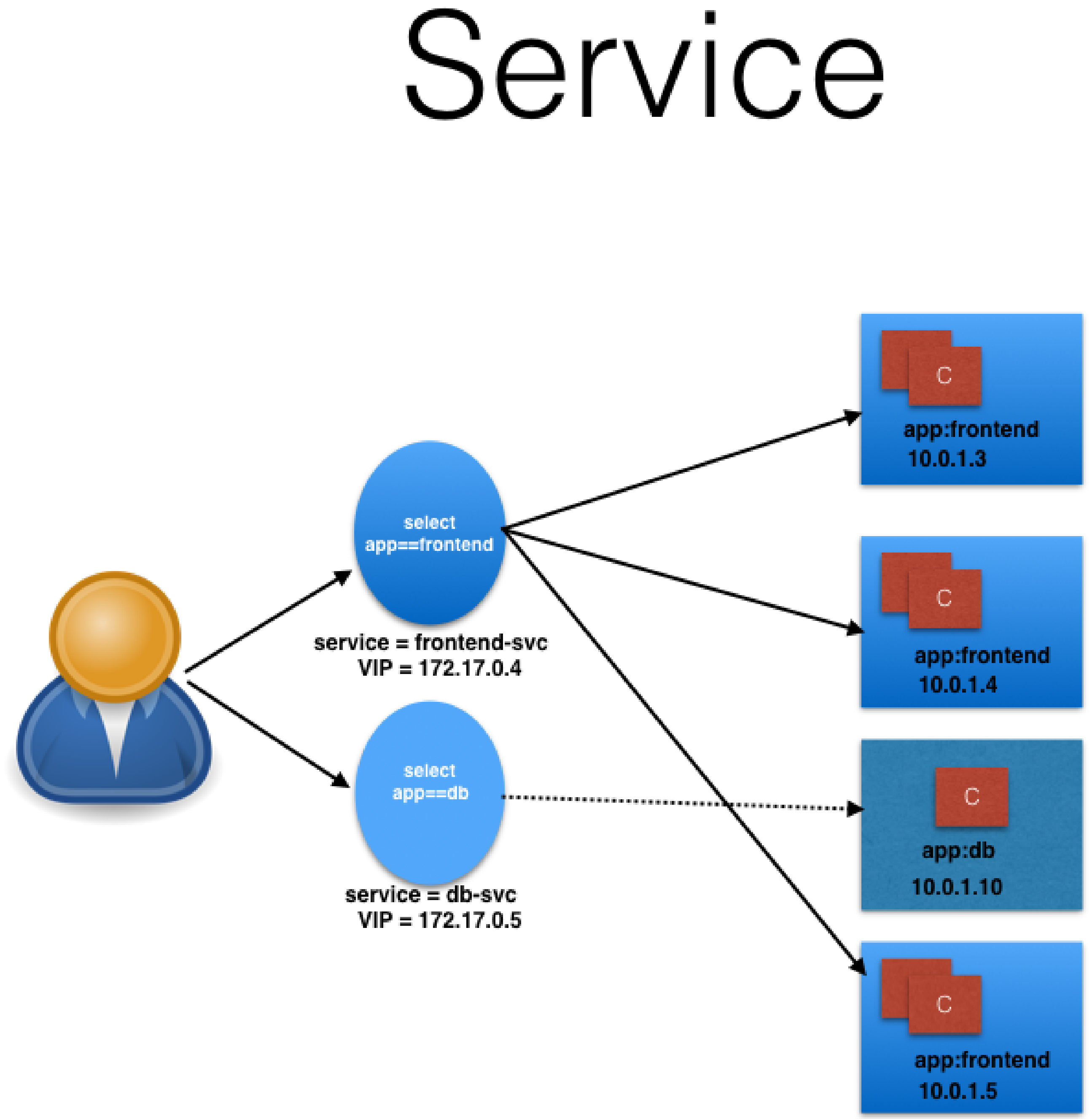
Connecting Users to Pods (1)



Connecting Users to Pods (2)



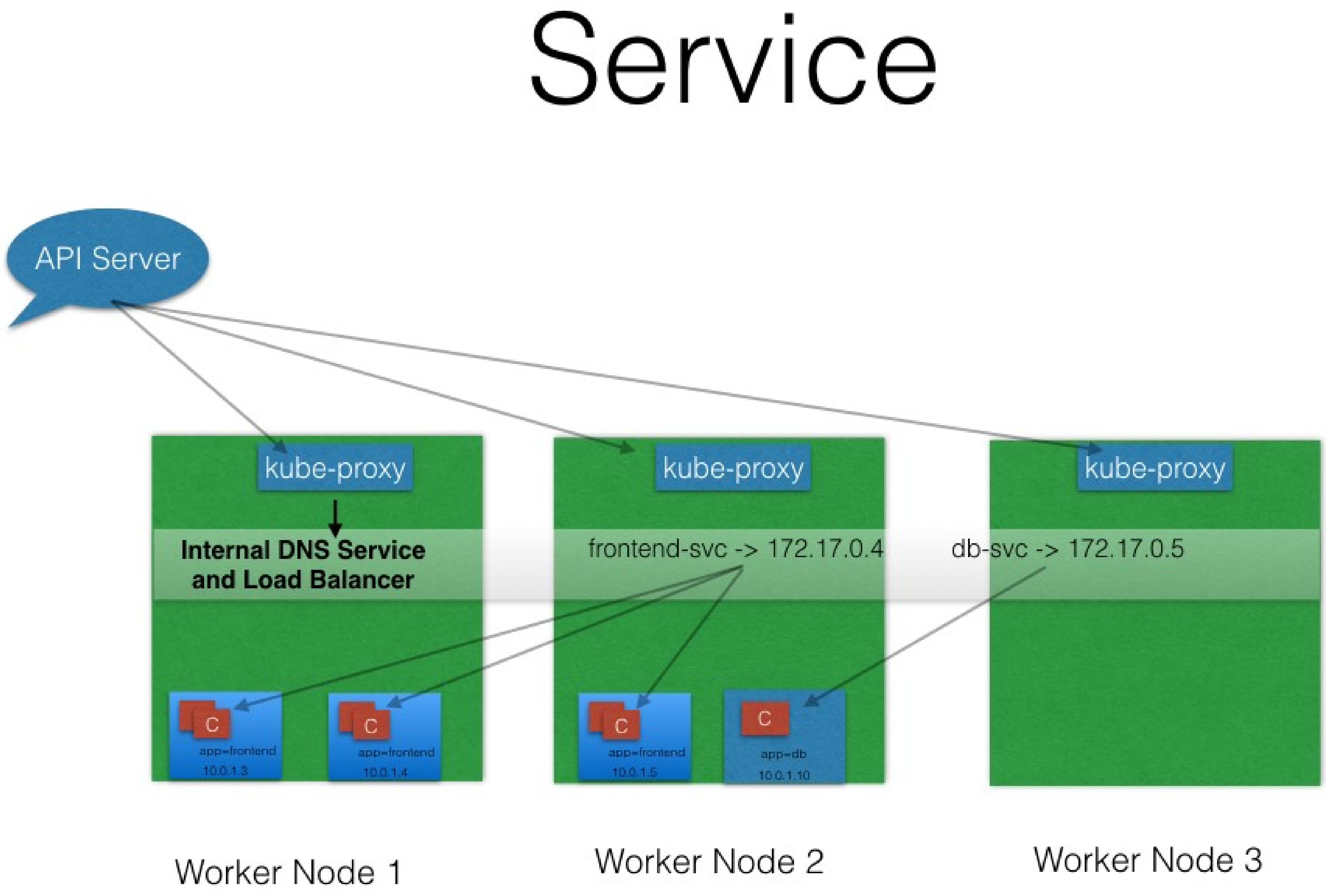
Service Name



Service Object Example

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

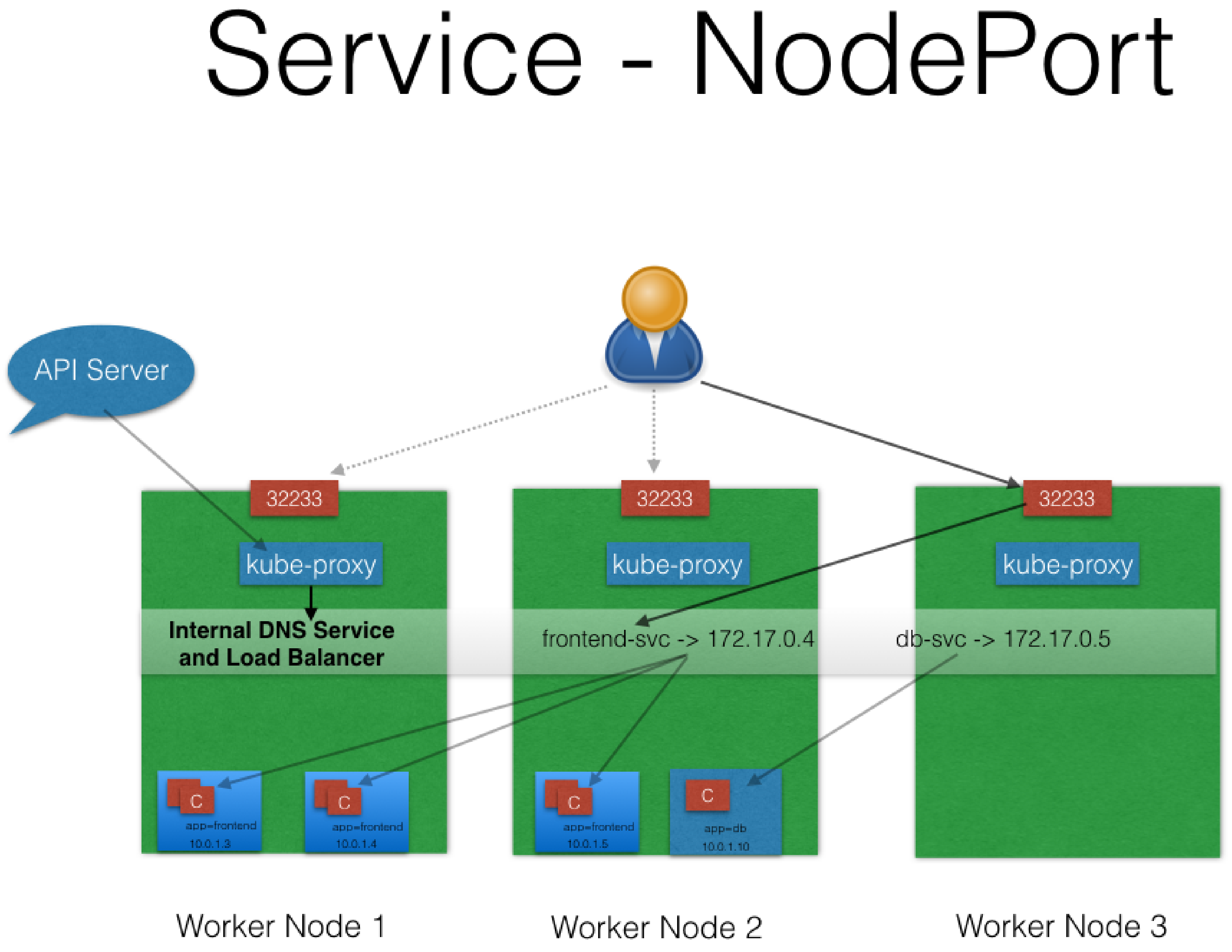
kube-proxy


Service
API Server
kube-proxy
kube-proxy
kube-proxy
Internal DNS Service and Load Balancer
frontend-svc -> 172.17.0.4
db-svc -> 172.17.0.5
C
app=frontend
10.0.1.3
C
app=frontend
10.0.1.4
C
app=frontend
10.0.1.5
C
app=db
10.0.1.10
Worker Node 1
Worker Node 2
Worker Node 3

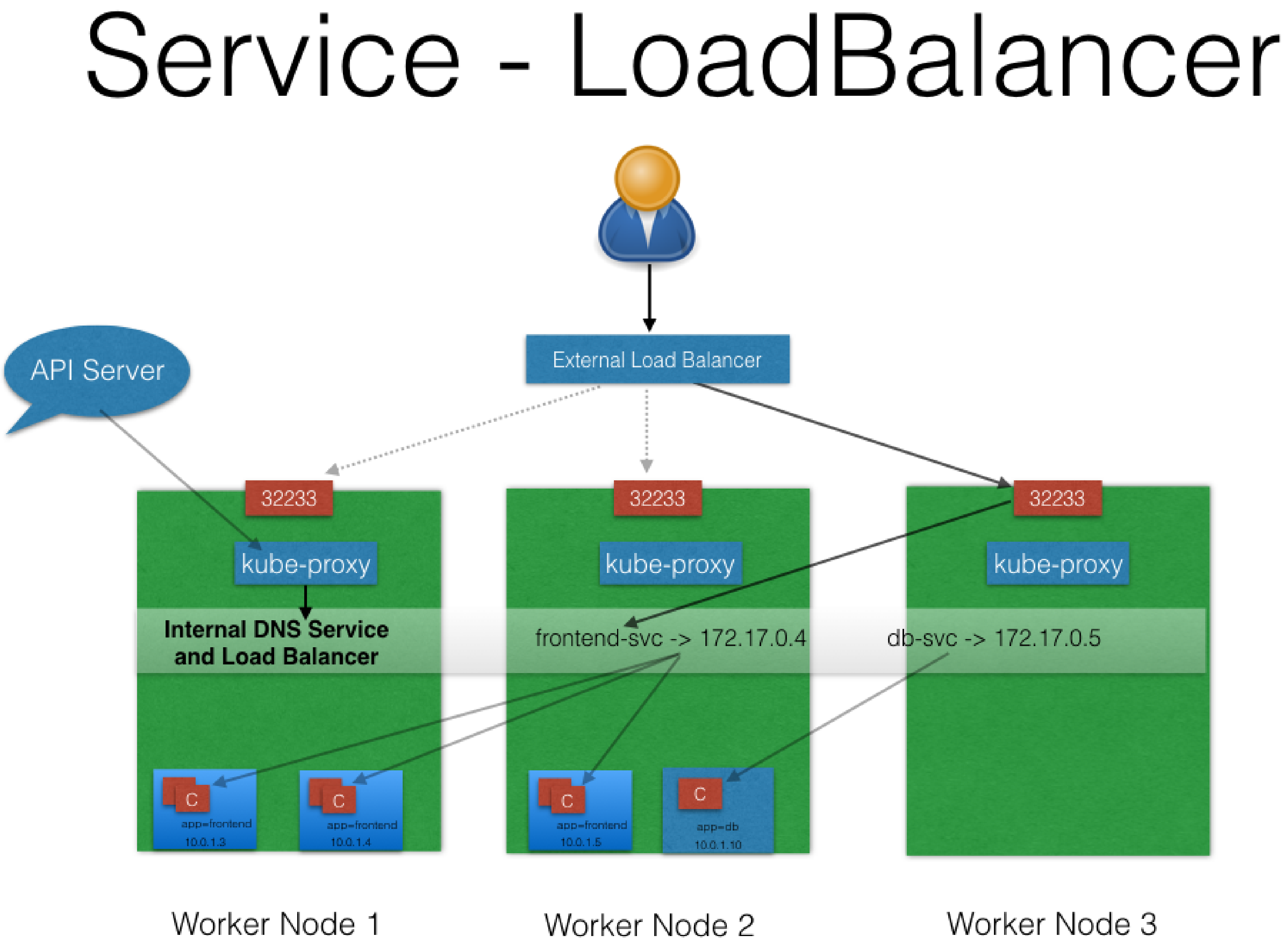
Service Discovery

- Environment Variables
- Internal DNS

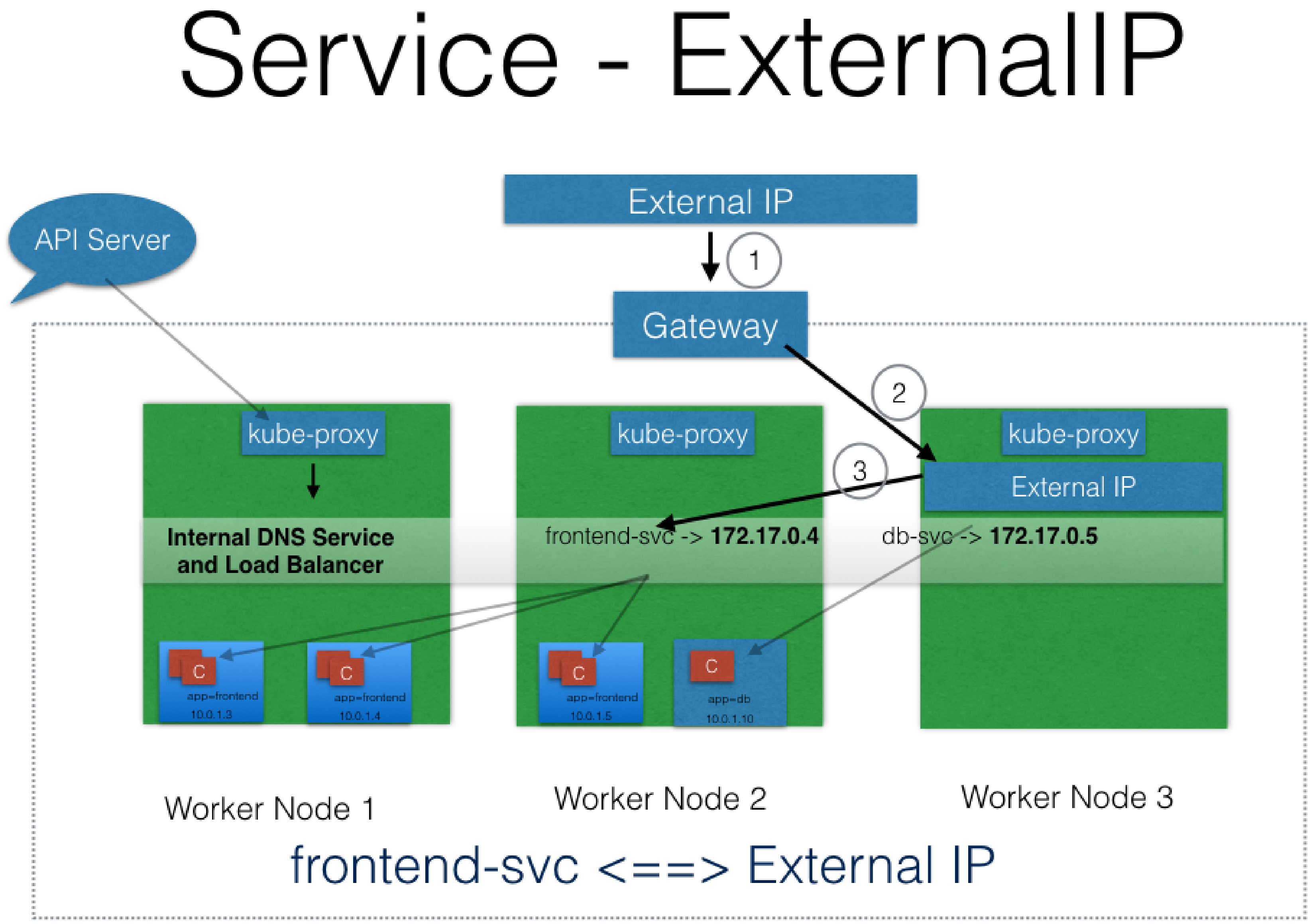
Service Type ClusterIP & NodePort



Service Type LoadBalancer



Service Type ExternalIP

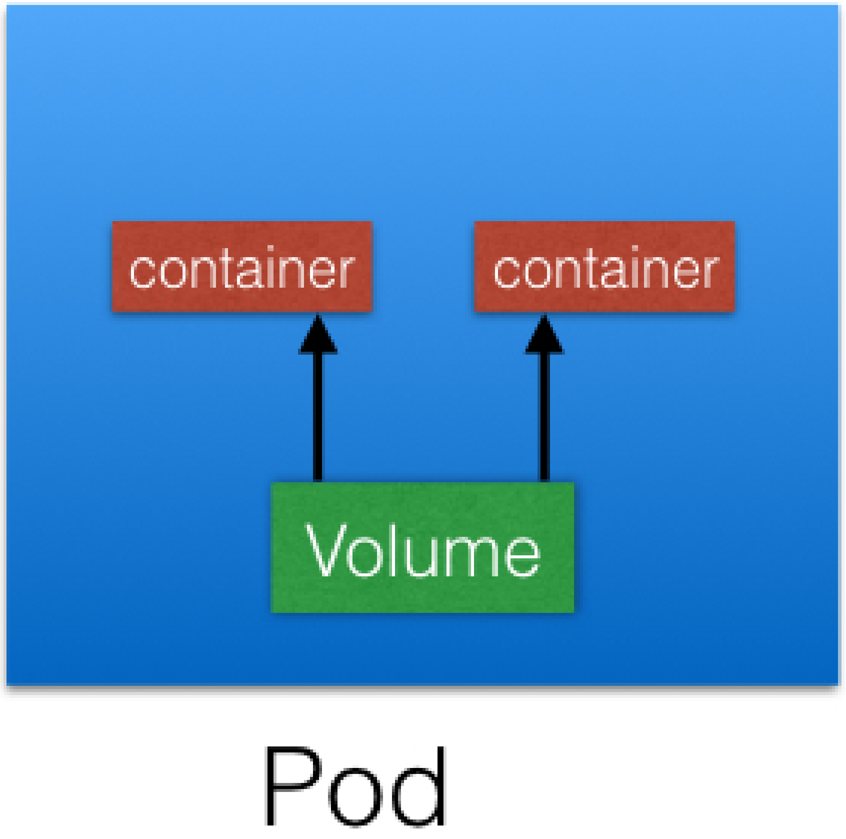


Service Type ExternalName

- ExternalName is a special ServiceType, that has no Selectors and does not define any endpoints. When accessed within the cluster, it returns a CNAME record of an externally configured service.
- The primary use case of this ServiceType is to make externally configured services like my-database.example.com available inside the cluster, using just the name, like my-database, to other services inside the same Namespace.

Kubernetes Volume Management

Volumes

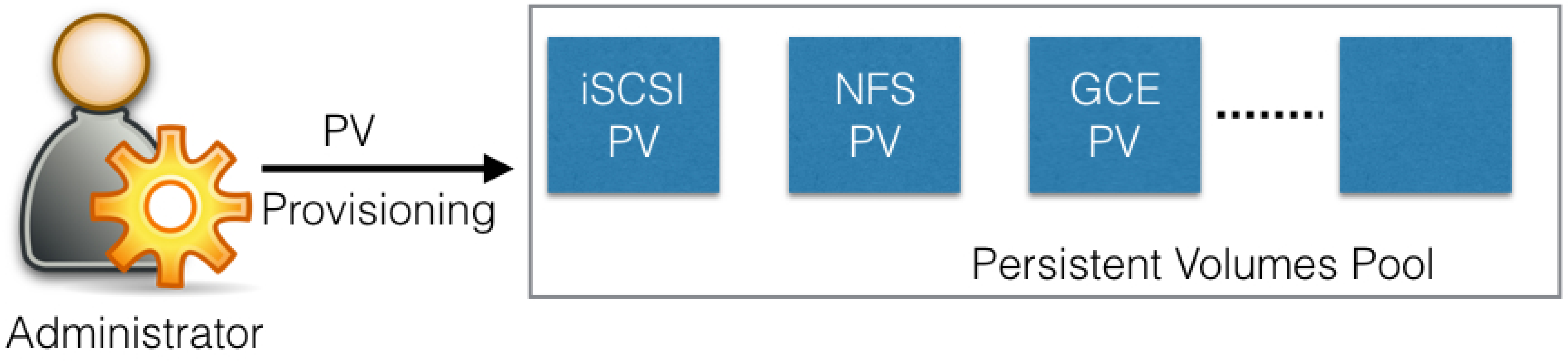


Volume Types

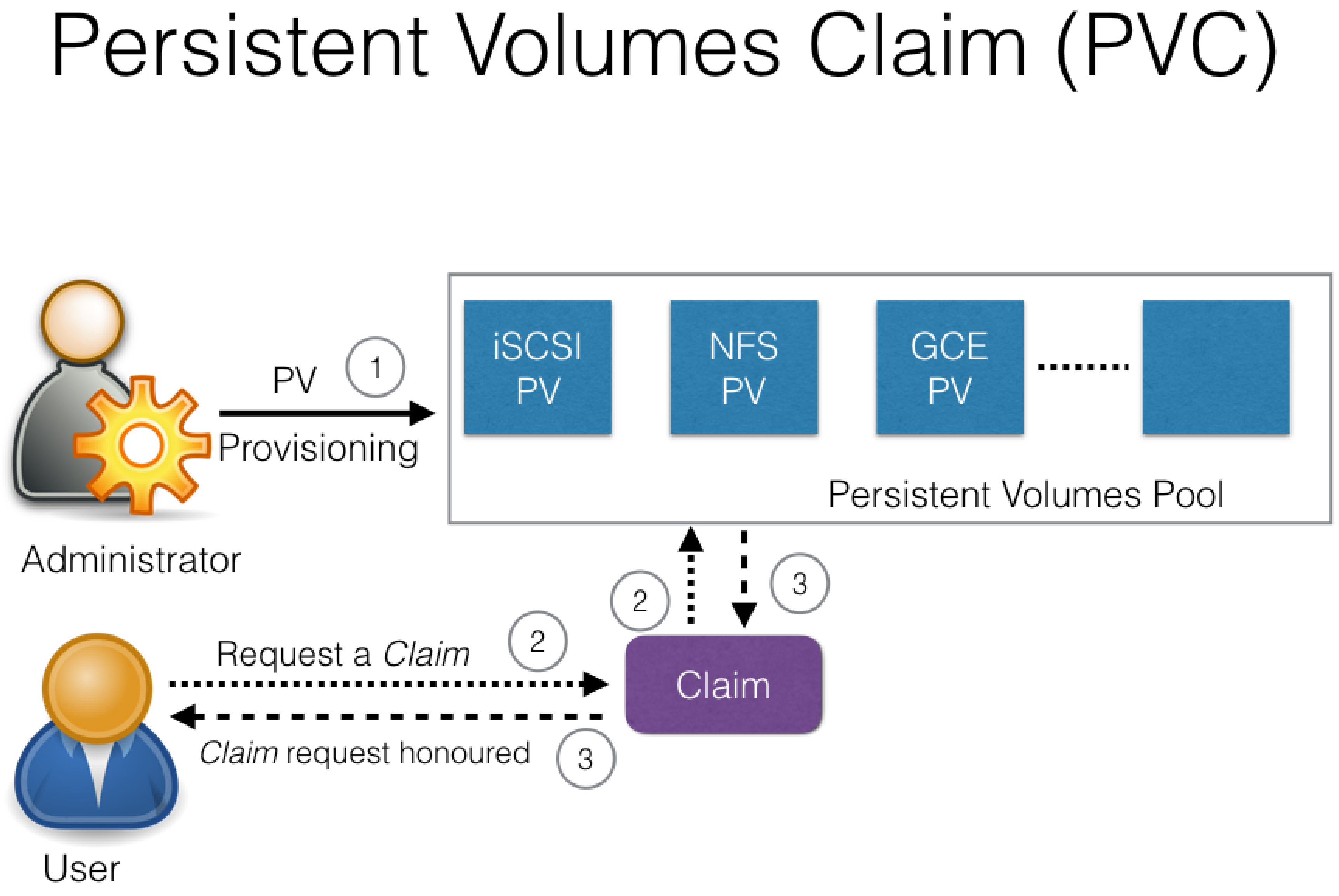
- EmptyDir
- hostPath
- gcePersistentDisk
- awsElasticBlockStore
- nfs
- iscsi
- fc
- flocker
- glusterfs
- rbd
- cephfs
- gitRepo
- secret
- persistentVolumeClaim
- downwardAPI
- projected
- FlexVolume
- AzureFileVolume
- AzureDiskVolume
- vsphereVolume
- Quobyte
- PortworxVolume
- ScaleIO
- StorageOS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

Persistent Volumes

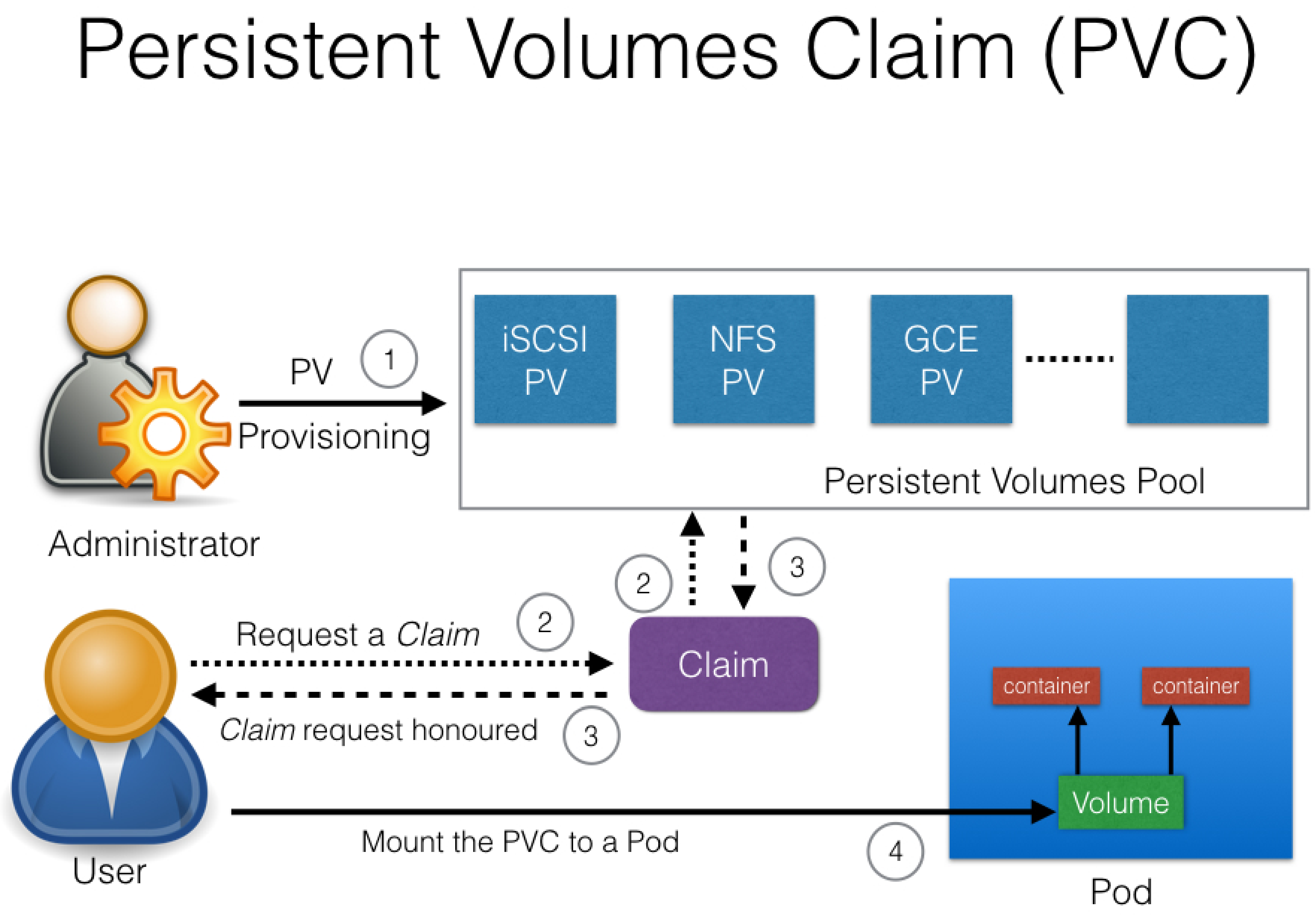
Persistent Volumes (PV)



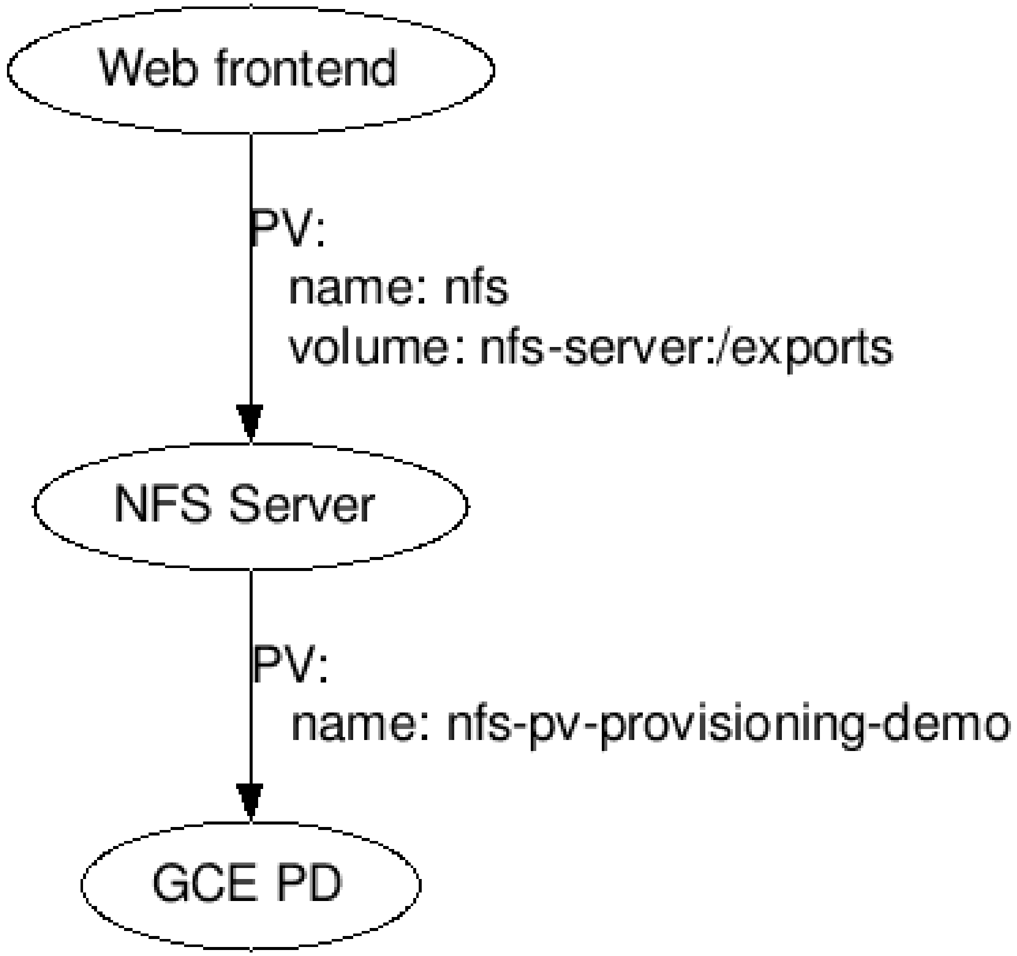
Persistent Volumes Claim (1)



Persistent Volumes Claim (2)



Lab PVE & PVC



https://github.com/kubernetes/examples/tree/maste
r/staging/volumes/nfs

Ingress

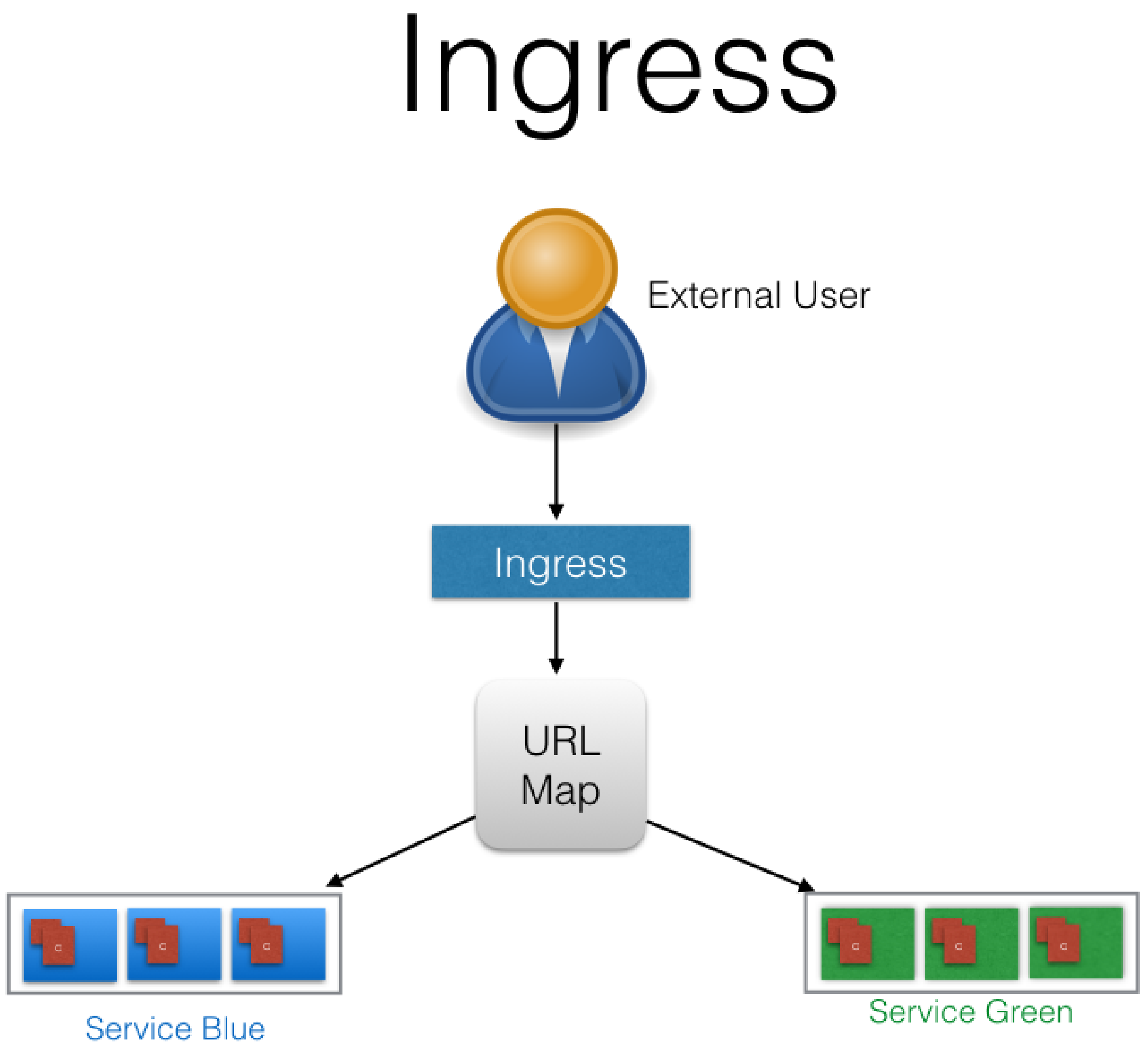
Ingress (1)

"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.

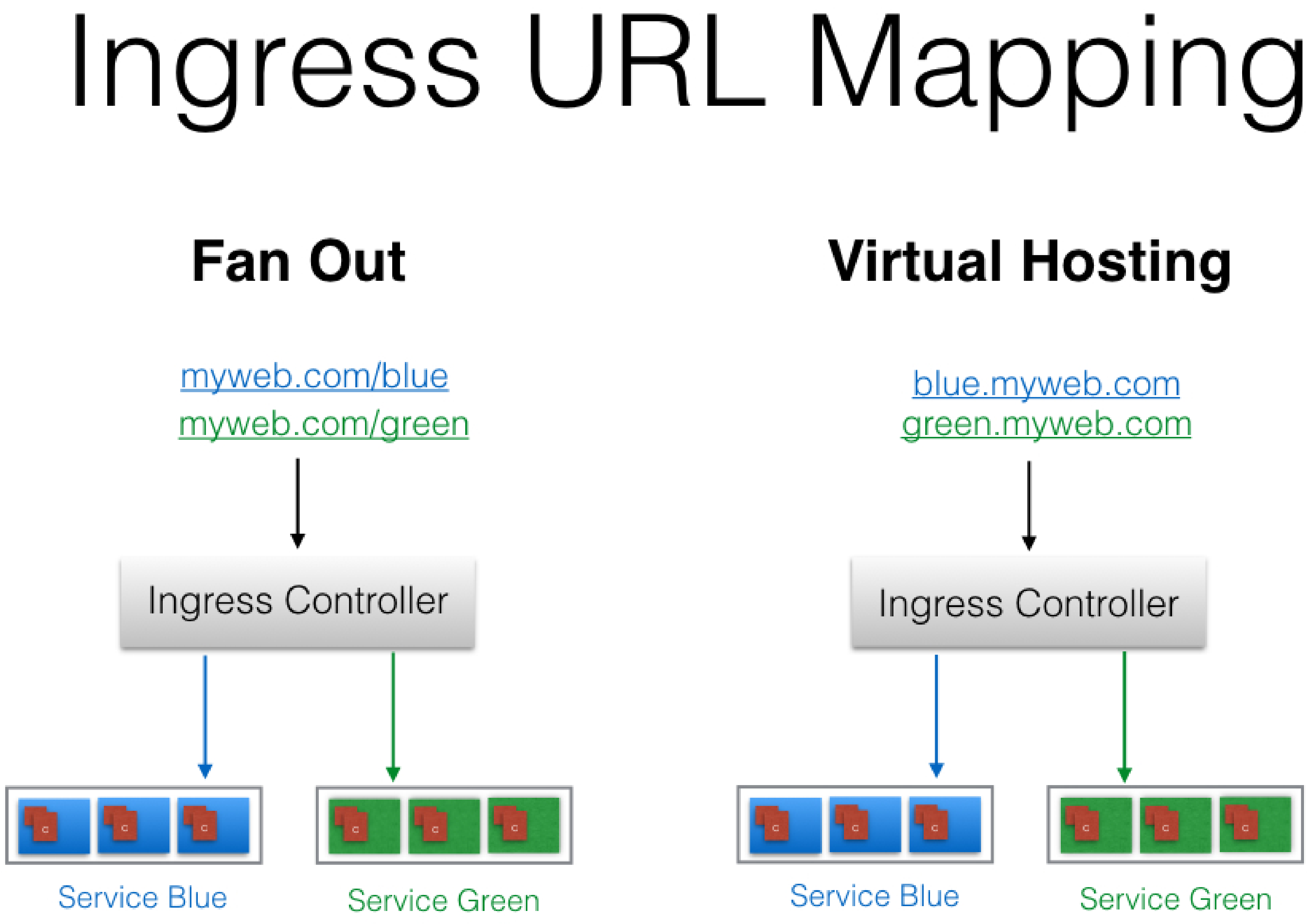
Ingress (2)


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.

www.btech.id