

Why is Kubernetes So Hot Right Now?

Why is Kubernetes So Hot Right Now?

- Cloud computing
 - Pay-as-you-go
 - Autoscaling
 - Low-ops/No-ops
- Containerised workloads
 - Docker
- Kubernetes runs container clusters on the cloud

Cloud Computing

- AWS, Microsoft Azure, Google Cloud Platform
- 'Big Three' public cloud providers

Compute

Storage

Infra-As-A-Service

Platform-As-A-Service

IaaS, PaaS and Containers

Infra-As-A-Service

Microsoft Azure VMs

AWS Elastic Compute Cloud (EC2)

GCP Compute Engine (GCE)

Platform-As-A-Service

Microsoft App Service

AWS Elastic Beanstalk

GCP App Engine

IaaS, PaaS and Containers

Infra-As-A-Service

- Too much ops
- Migration is painful

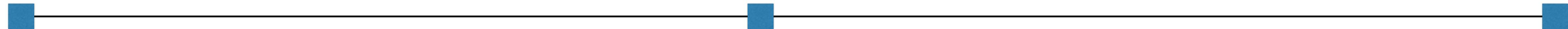
Platform-As-A-Service

- Lock-in to cloud provider
- Less control over infra

Containers as Middle Path

Infra-As-A-Service

Platform-As-A-Service



Containers

hybrid, multi-cloud



Container Clusters

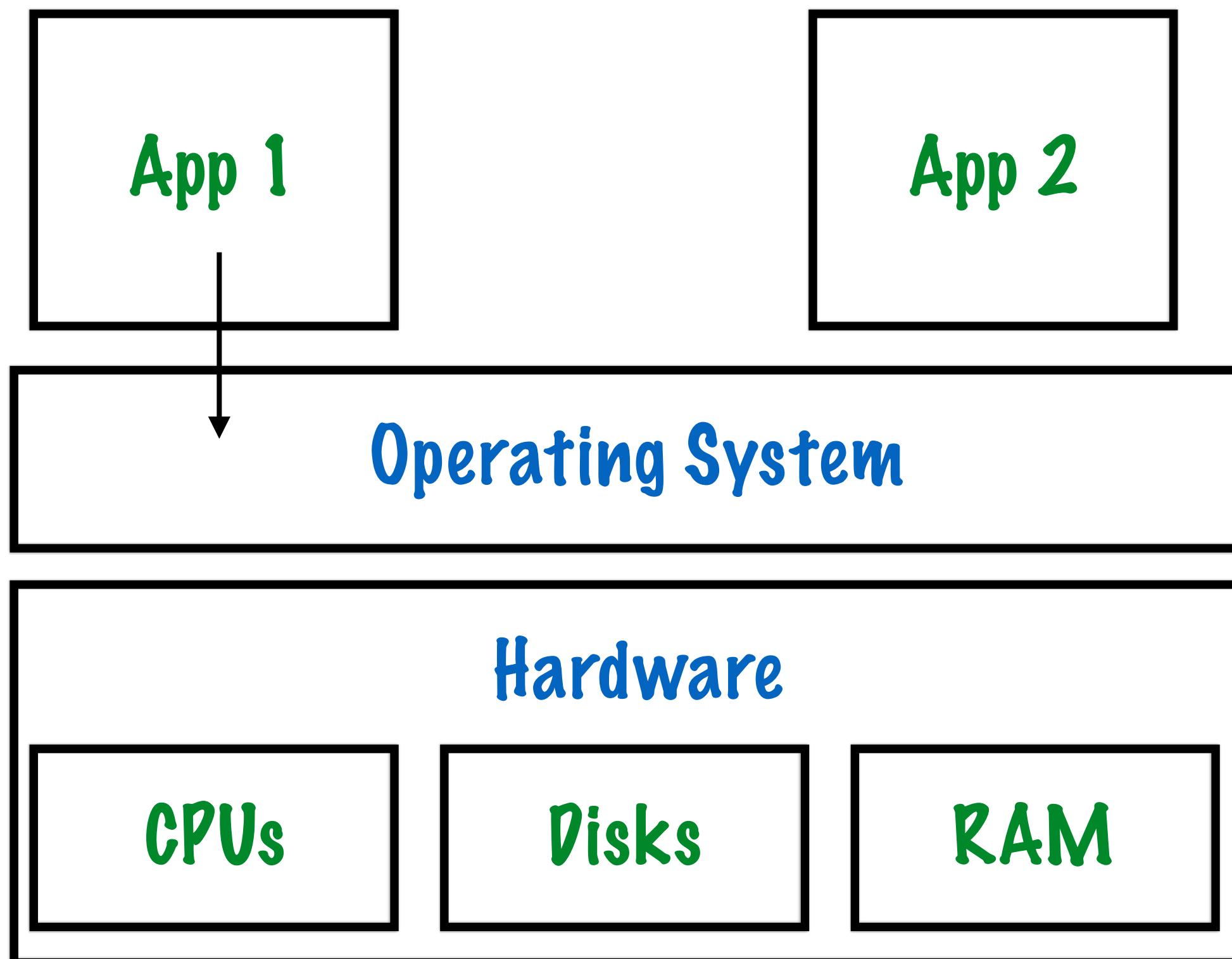


Kubernetes

What are Containers?

Bare Metal and Virtual Machines

Bare Metal



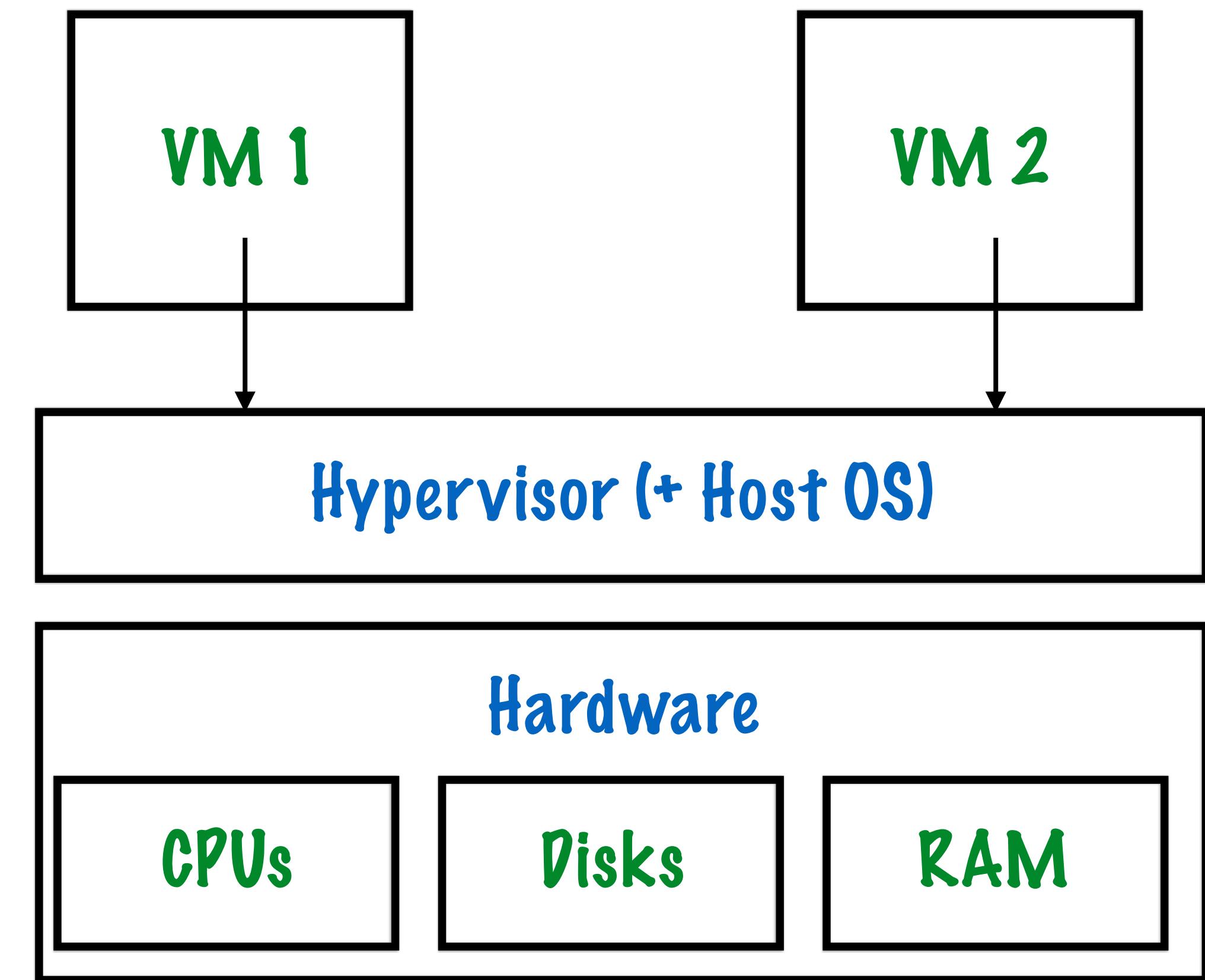
Virtual Machine

Bare Metal and Virtual Machines

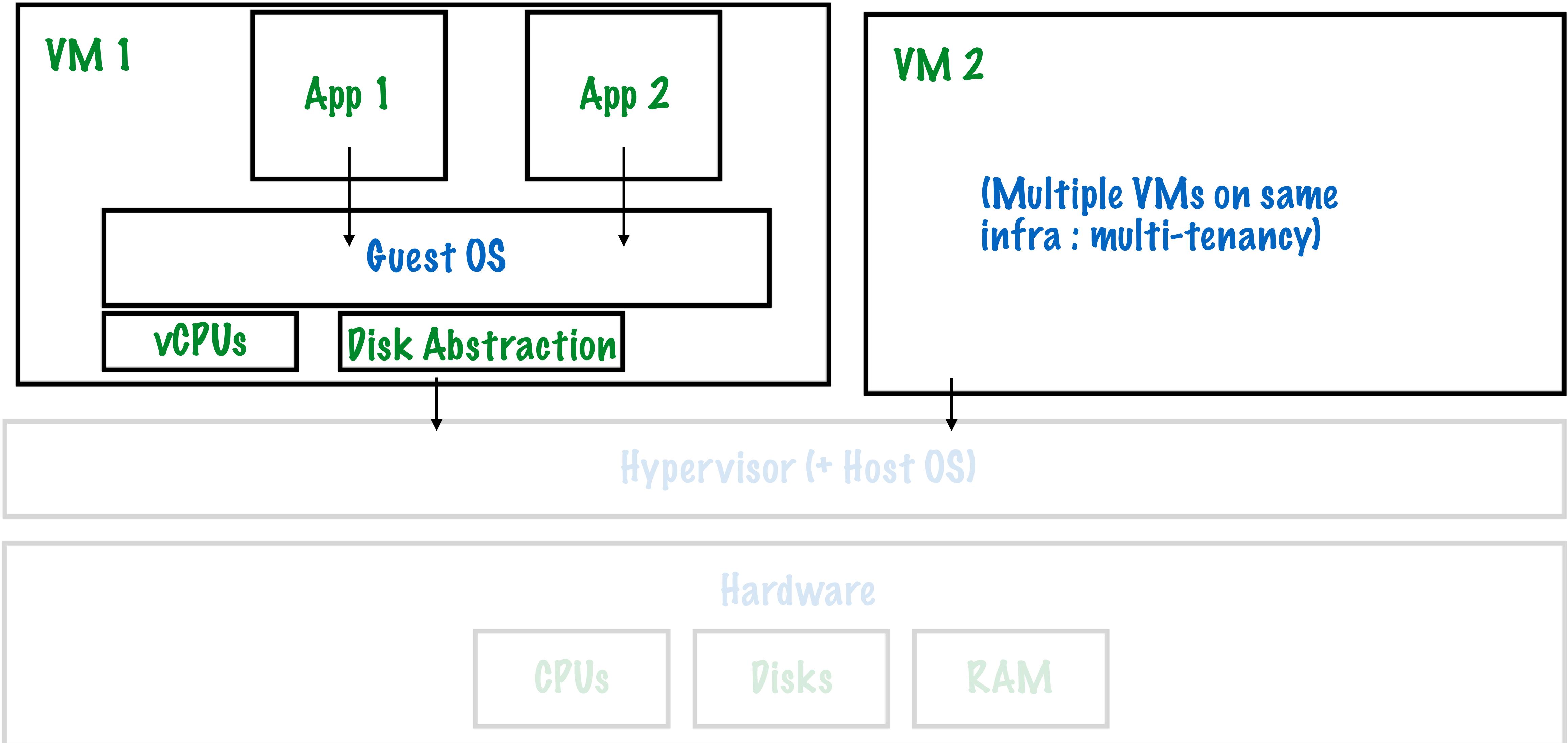
Bare Metal



Virtual Machine



Apps on Virtual Machines



VMs Are Great, But...

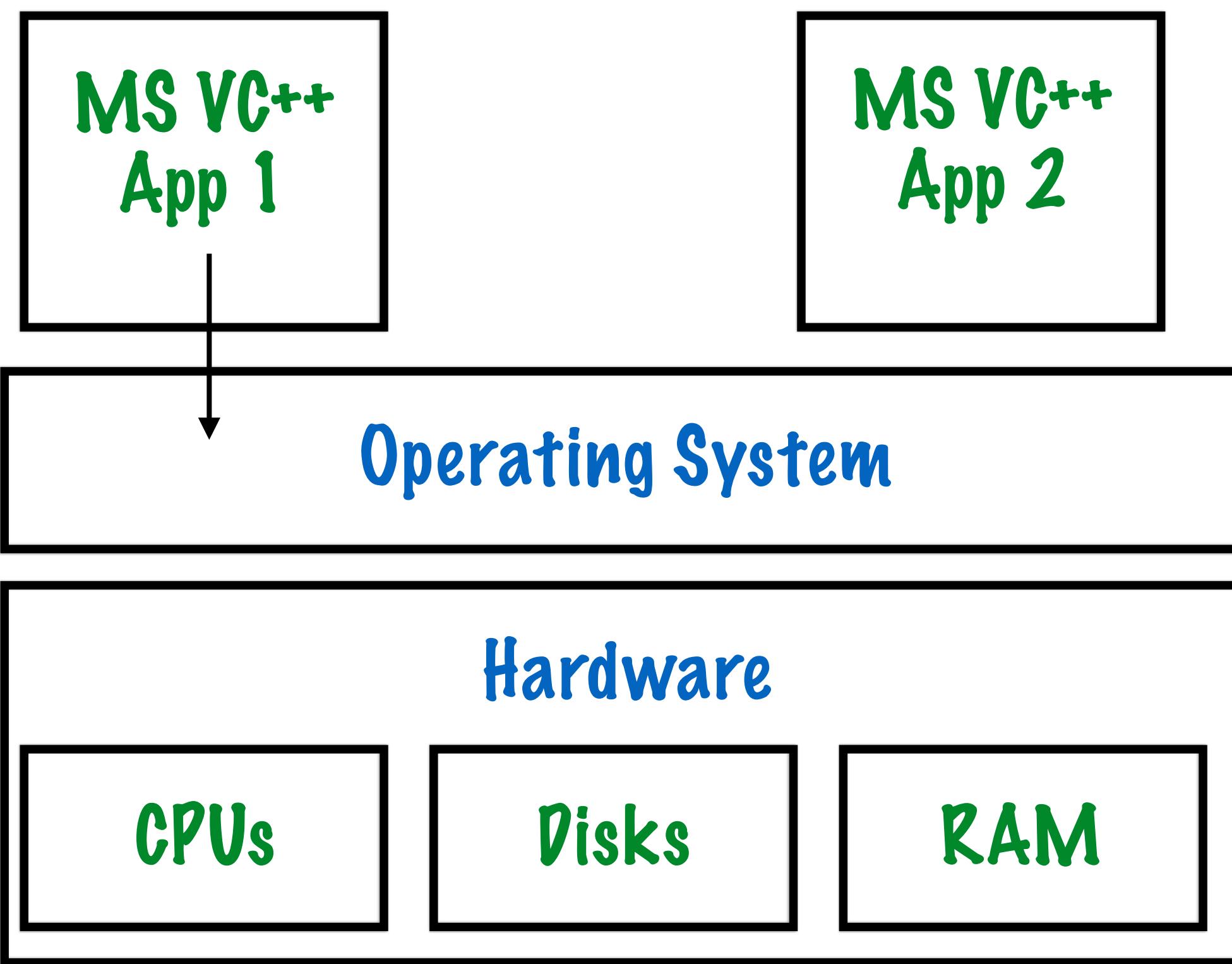
- Heavyweight images
 - GB in size
 - Contain guest OS
- Slow to boot up - autoscaling/autohealing slows down
- Not that portable - migrations are painful

Container

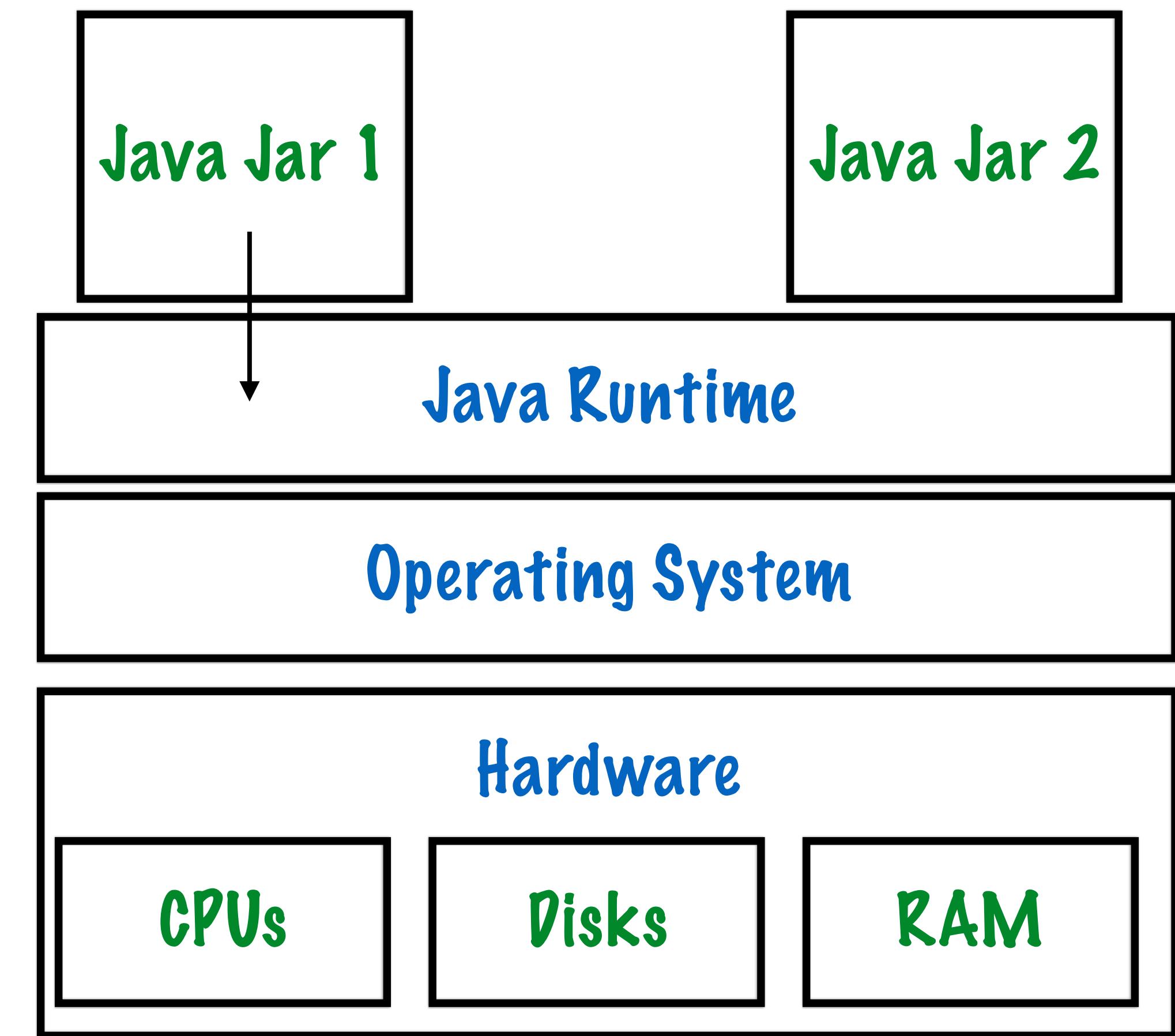
A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.

Containers Are 'Like' Jars

MS Visual C++ App

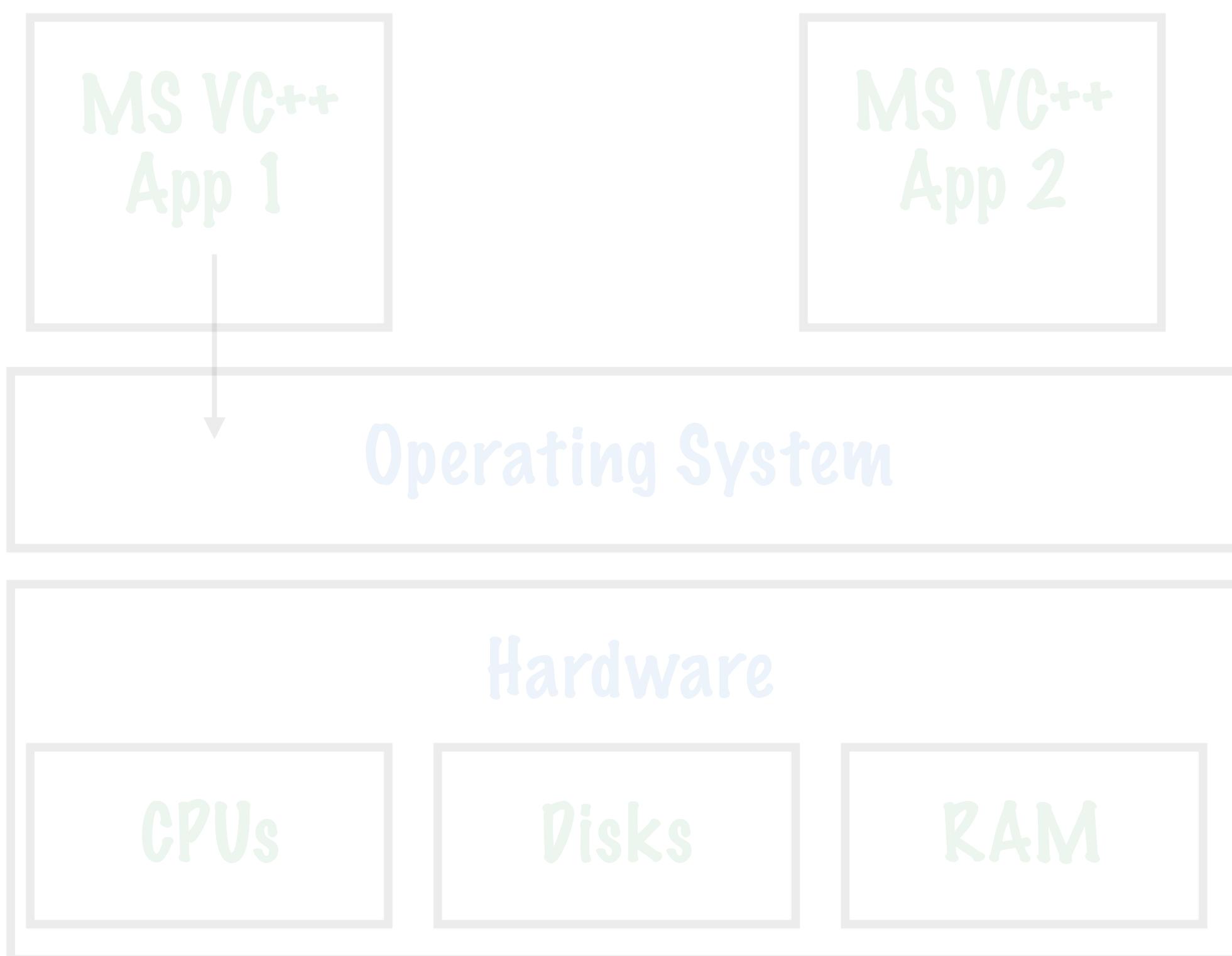


Java Jars

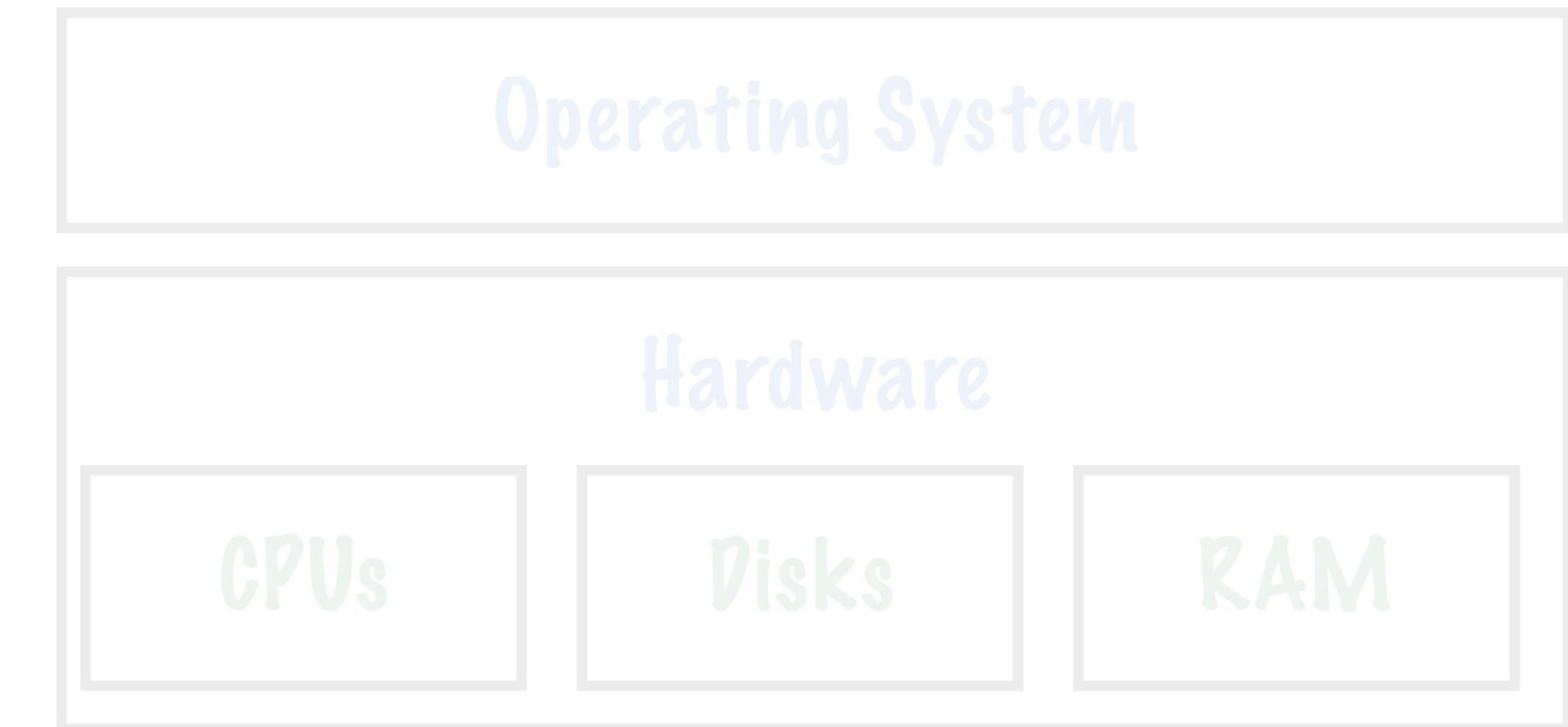


Containers Are 'Like' Jars

MS Visual C++ App



Java Jars



Containers Are 'Like' Jars

Java Jars

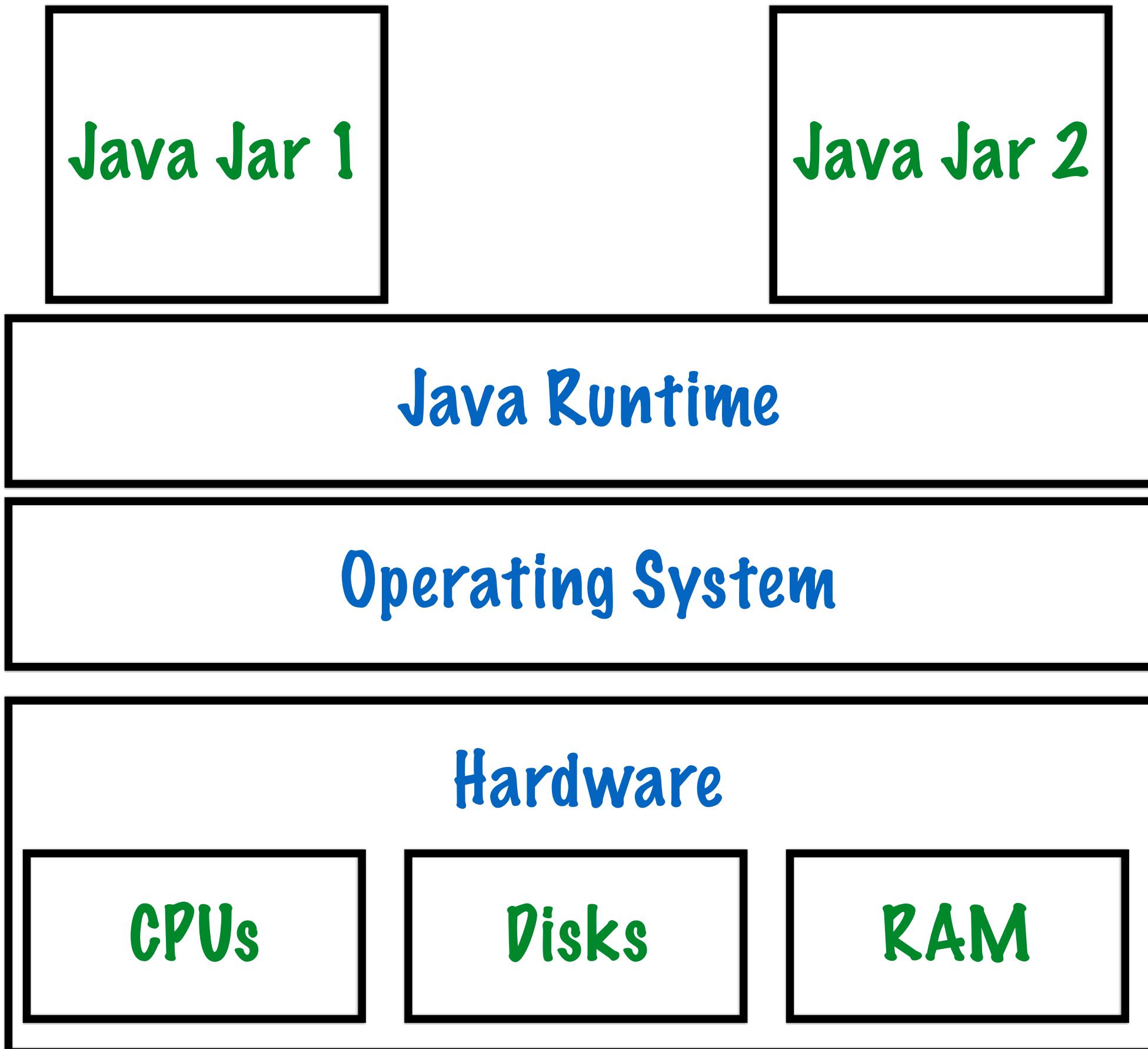
- Platform independent
- layer of abstraction (Java Runtime)
- Compile code into jars

Docker Containers

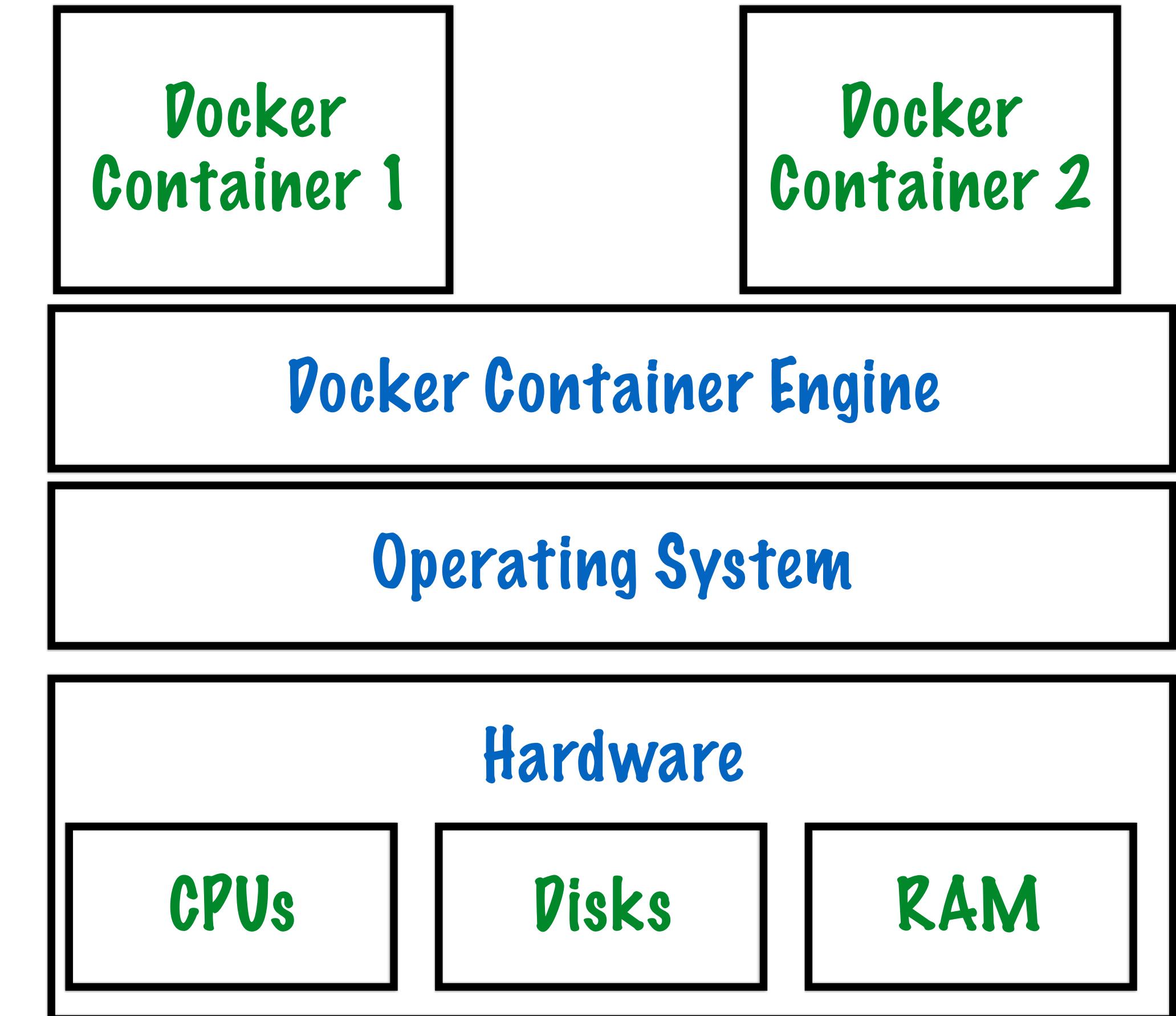
- Platform independent
- layer of abstraction (Docker Runtime)
- Containerise apps into Docker containers

Containers Are 'Like' Jars

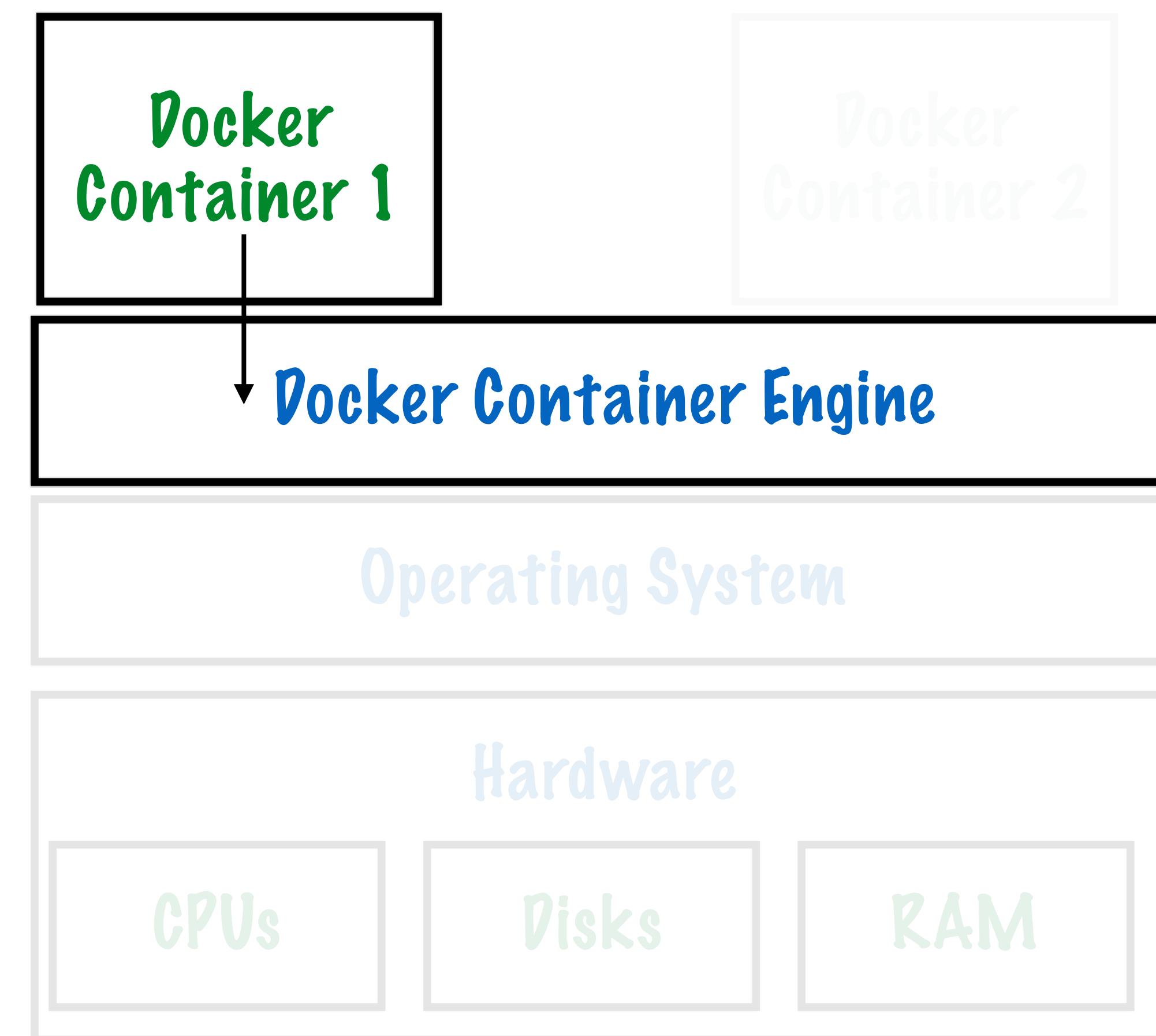
Java Jars



Docker Containers



Docker Containers



Containers Have Many Advantages

- No Guest OS
- Platform independent
- Lightweight images
 - MB in size
 - No OS
- Fast to fire up - good for autoscaling
- Hybrid, multi-cloud

What is Docker?

Docker

Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server.

Docker

- Containerisation technology
- Lightweight images
- Linux - resource isolation and other features
- Independent containers within Linux without VM overhead

Docker

- dotCloud, a PaaS company
- Open-sourced in 2013
- Most popular container engine today

Docker

Docker Compose

Create Docker containers using .yaml
manifests to define containers

Docker Swarm

Cluster orchestrator - losing out to
Kubernetes now

Docker and Kubernetes – It's Complicated

- Friends: Docker containers and Kubernetes are complements
- Rivals: Docker Swarm and Kubernetes are competitors

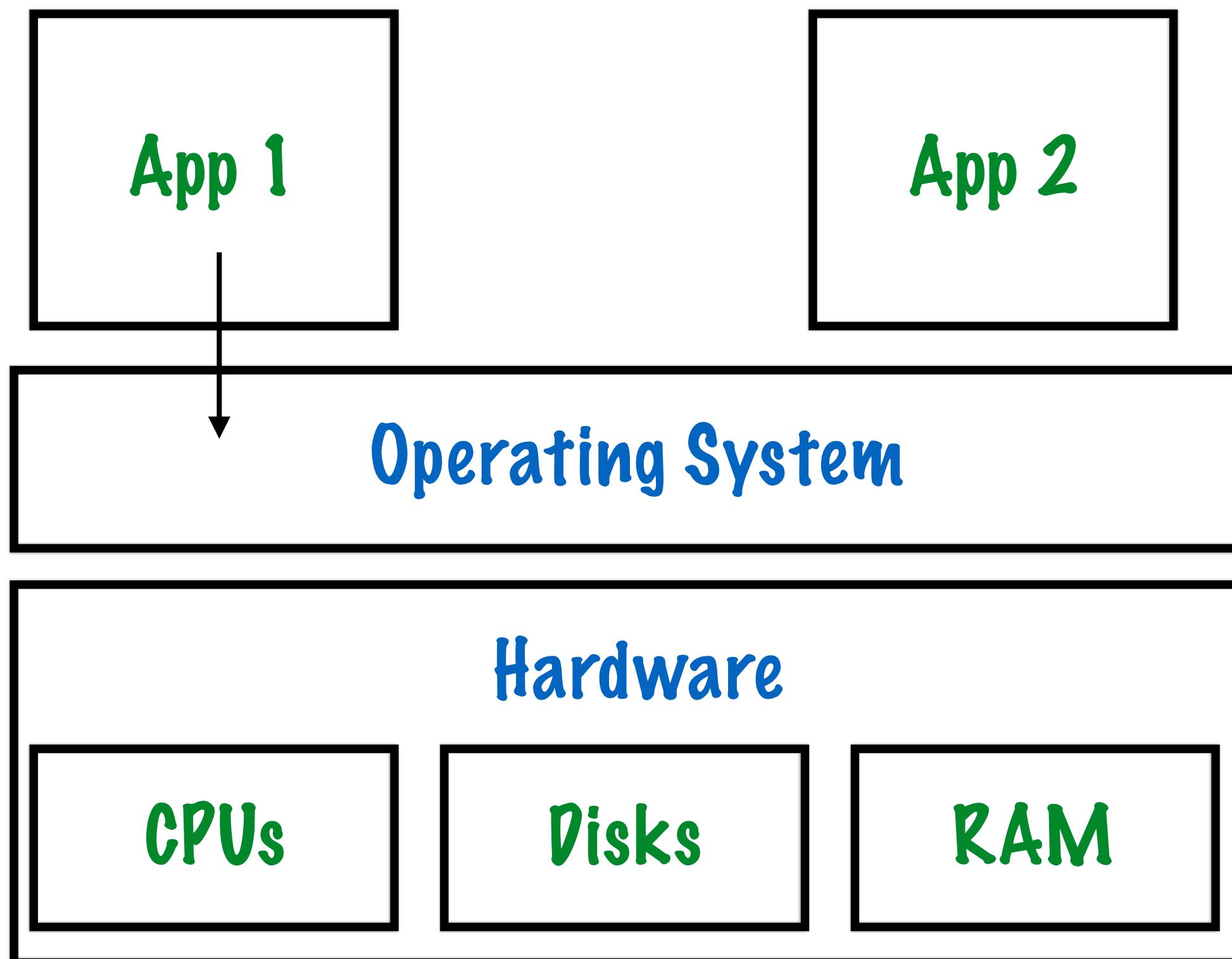
Docker and Kubernetes - It's Complicated

- Now - Native Docker support for Kubernetes
- Docker to spin off container format
containerd

What is Kubernetes?

Bare Metal and Virtual Machines

Bare Metal



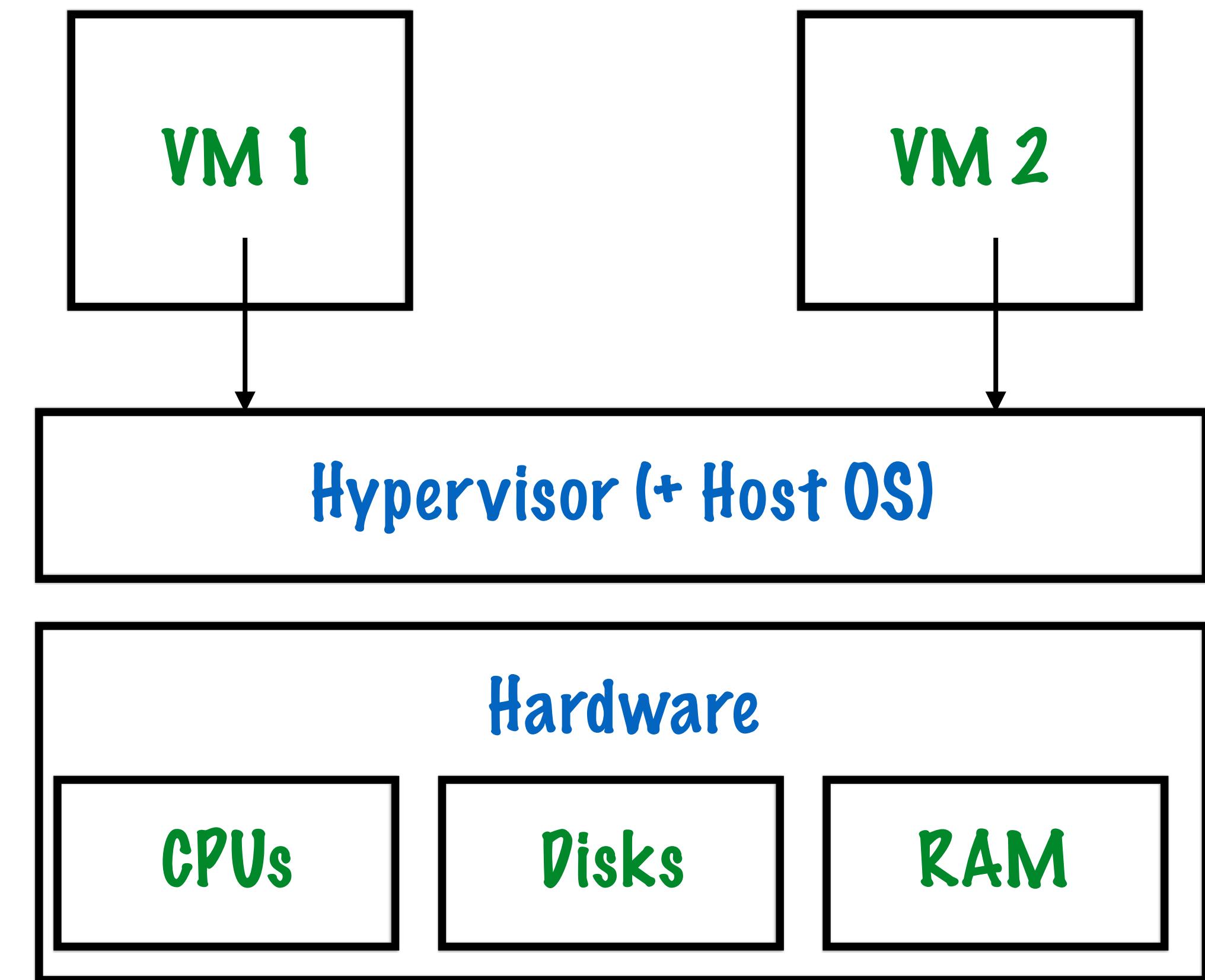
Virtual Machine

Bare Metal and Virtual Machines

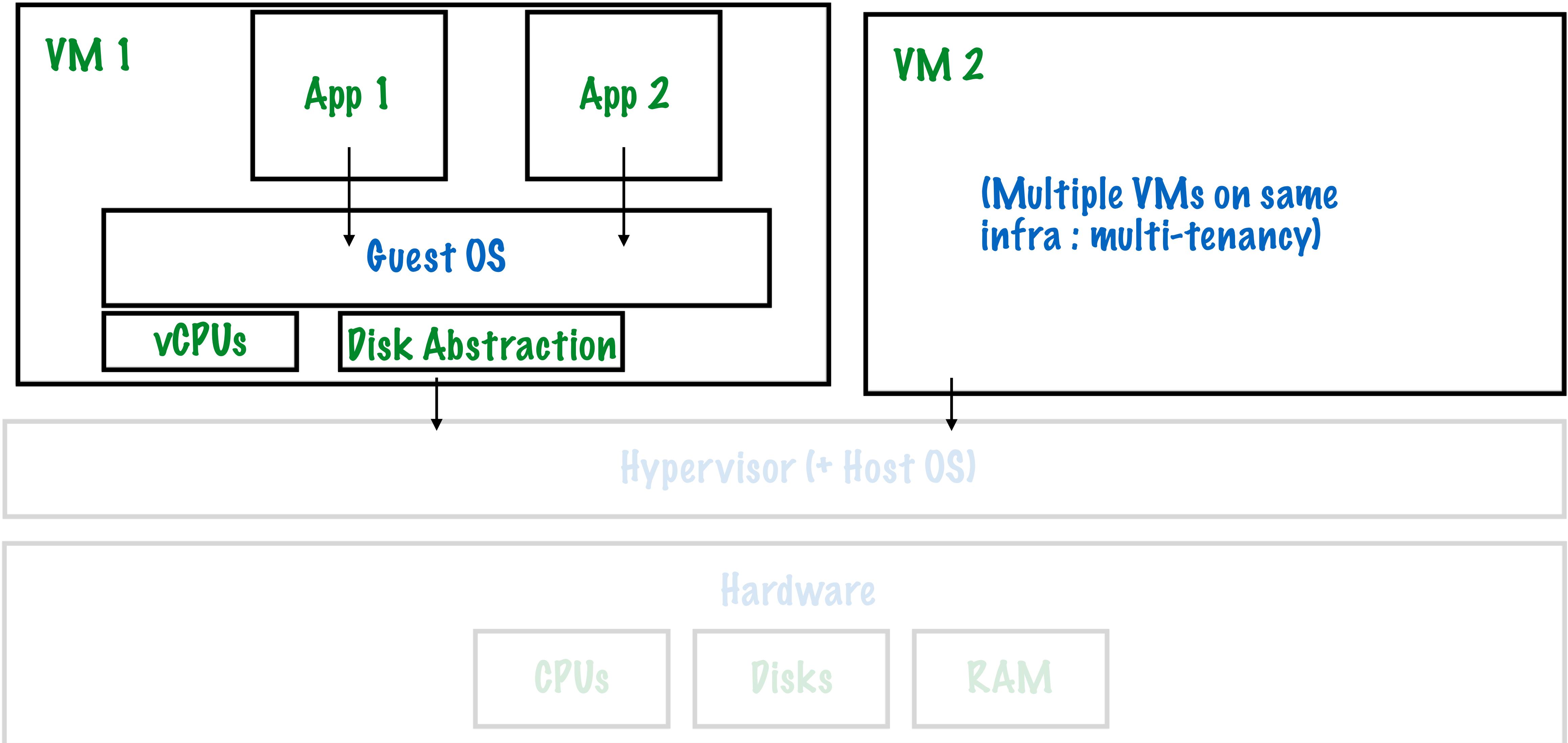
Bare Metal



Virtual Machine



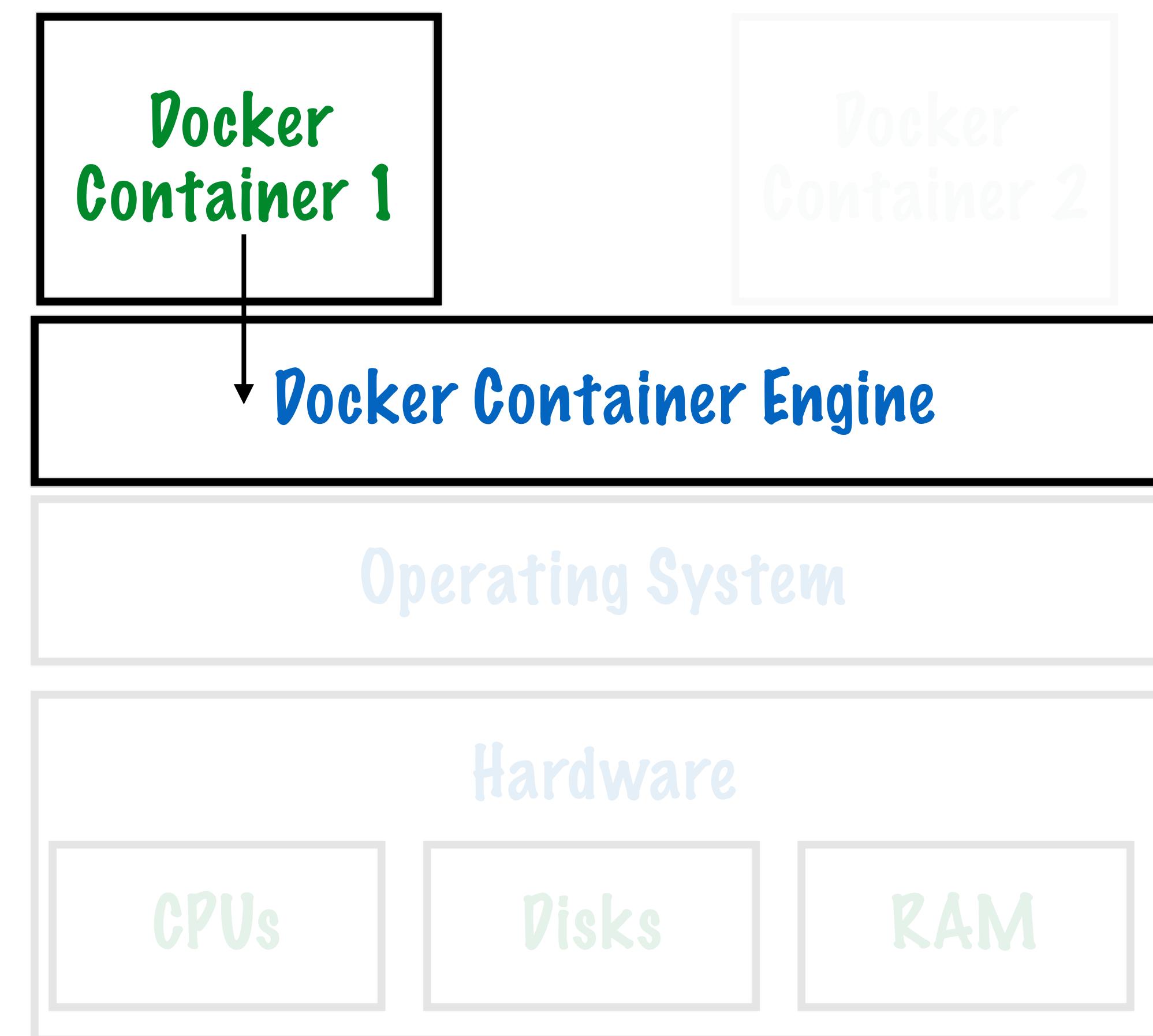
Apps on Virtual Machines



Container

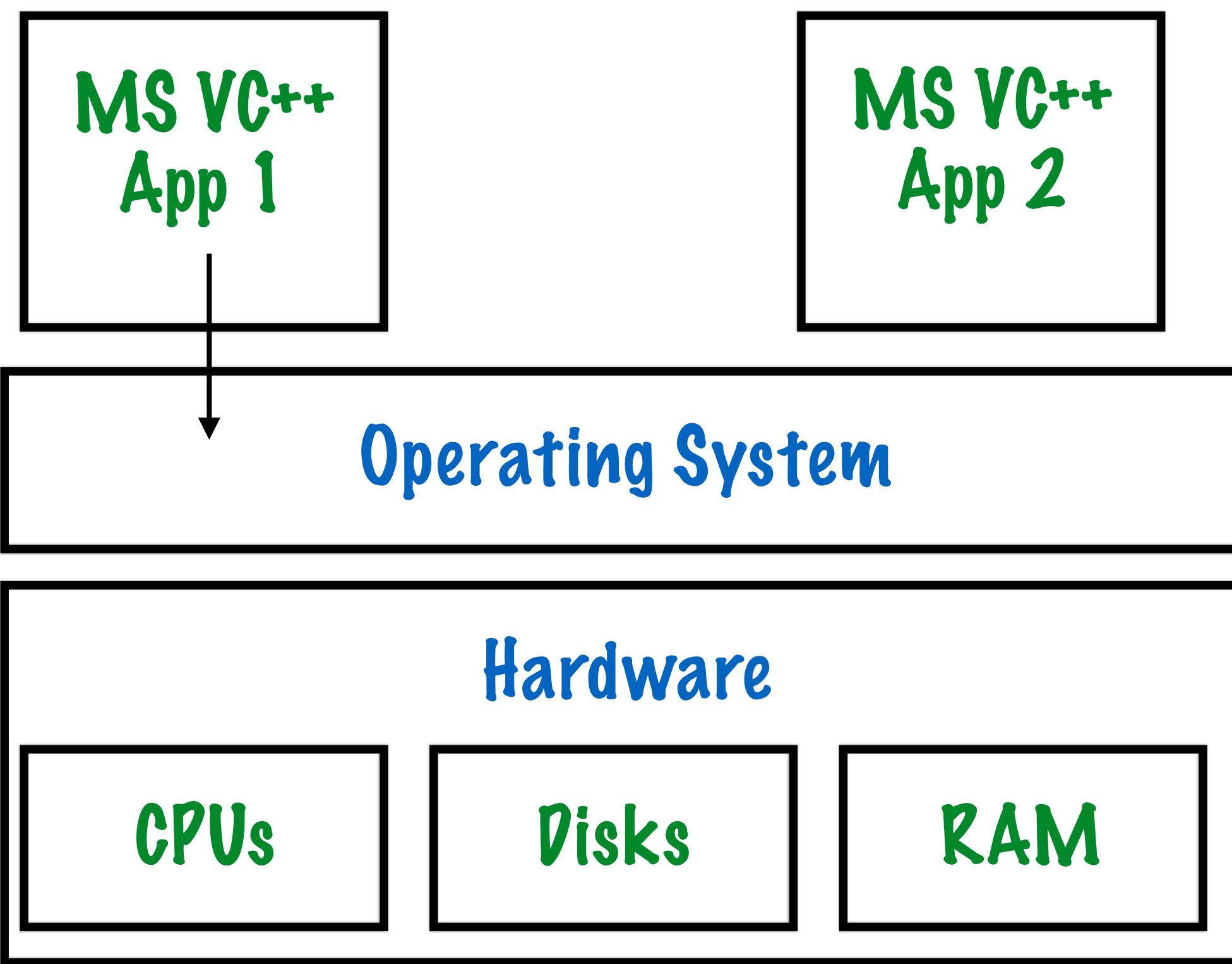
A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.

Docker Containers

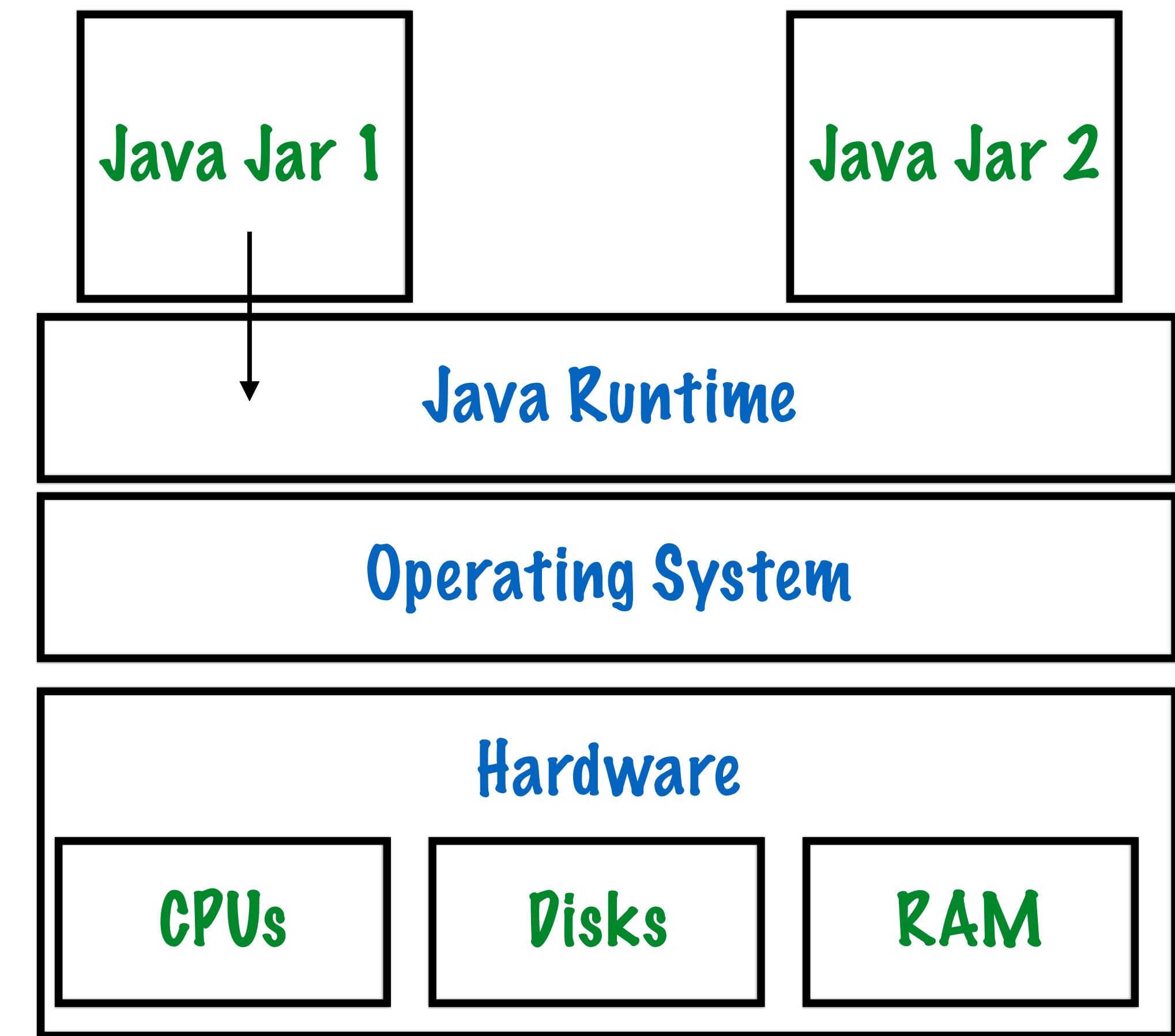


Containers Are 'Like' Jars

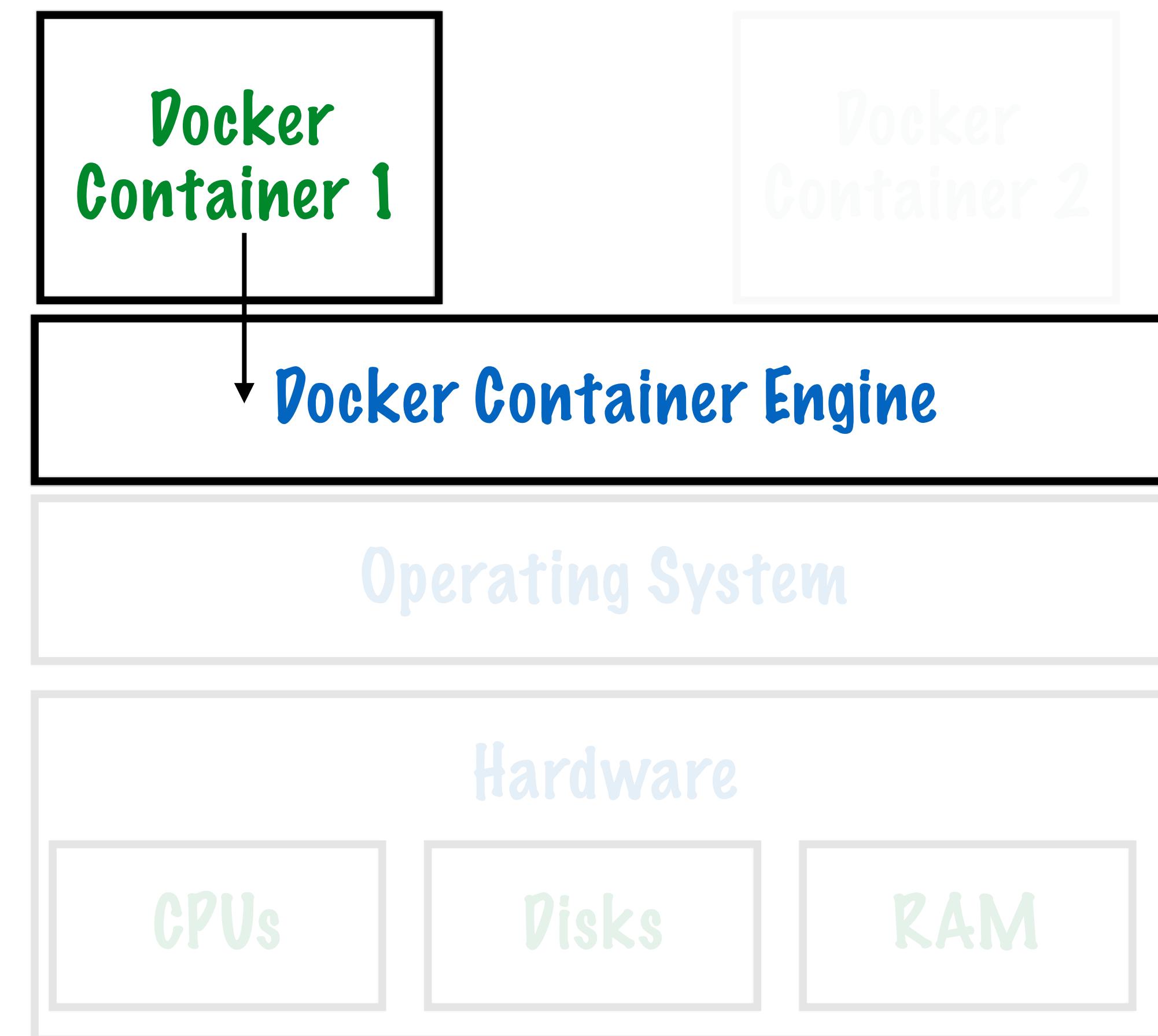
MS Visual C++ App



Java Jars



Docker Containers



Can We Have More?

- Auto-healing: Crashed containers should restart
- Auto-scaling: More clients? More demand
- Load-balancing: Distribute client requests
- Isolation: Sandboxes so that containers don't interfere

Kubernetes

Orchestration technology for containers –
convert isolated containers running on
different hardware into a **cluster**

Kubernetes for Hybrid, Multi-cloud World

- **Hybrid: On-premise datacenter + public cloud**
- **Multi-Cloud: More than one public cloud provider**
- **Why Hybrid? Cloud migration is slow, legacy DCs already exist**
- **Why Multi-cloud? Strategic reasons (Amazon/Whole Foods)**

Cloud Computing

- AWS, Microsoft Azure, Google Cloud Platform
- 'Big Three' public cloud providers

Compute

Storage

Infra-As-A-Service

Platform-As-A-Service

IaaS, PaaS and Containers

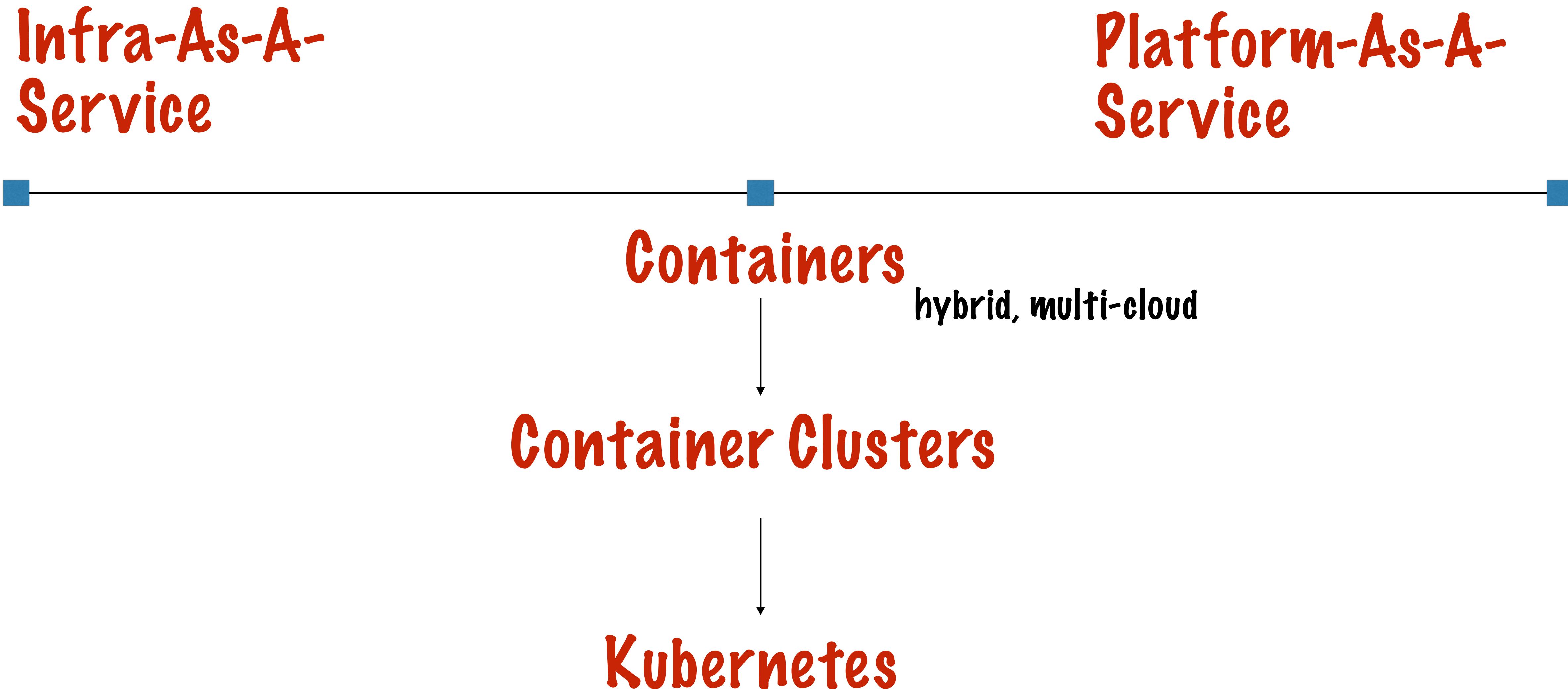
Infra-As-A-Service

- Too much ops
- Migration is painful

Platform-As-A-Service

- Lock-in to cloud provider
- Less control over infra

Kubernetes for a Hybrid, Multi-Cloud World



Kubernetes: Supported by the 'Big Three'

- AWS
- Azure
- Google Cloud Platform (GCP) - special relationship
- Kubernetes originated at Google

How Does Kubernetes Work?

Kubernetes

Orchestration technology – convert isolated containers running on different hardware into a cluster

Isolated Infra

Containers running on

Bare metal

VM Instances

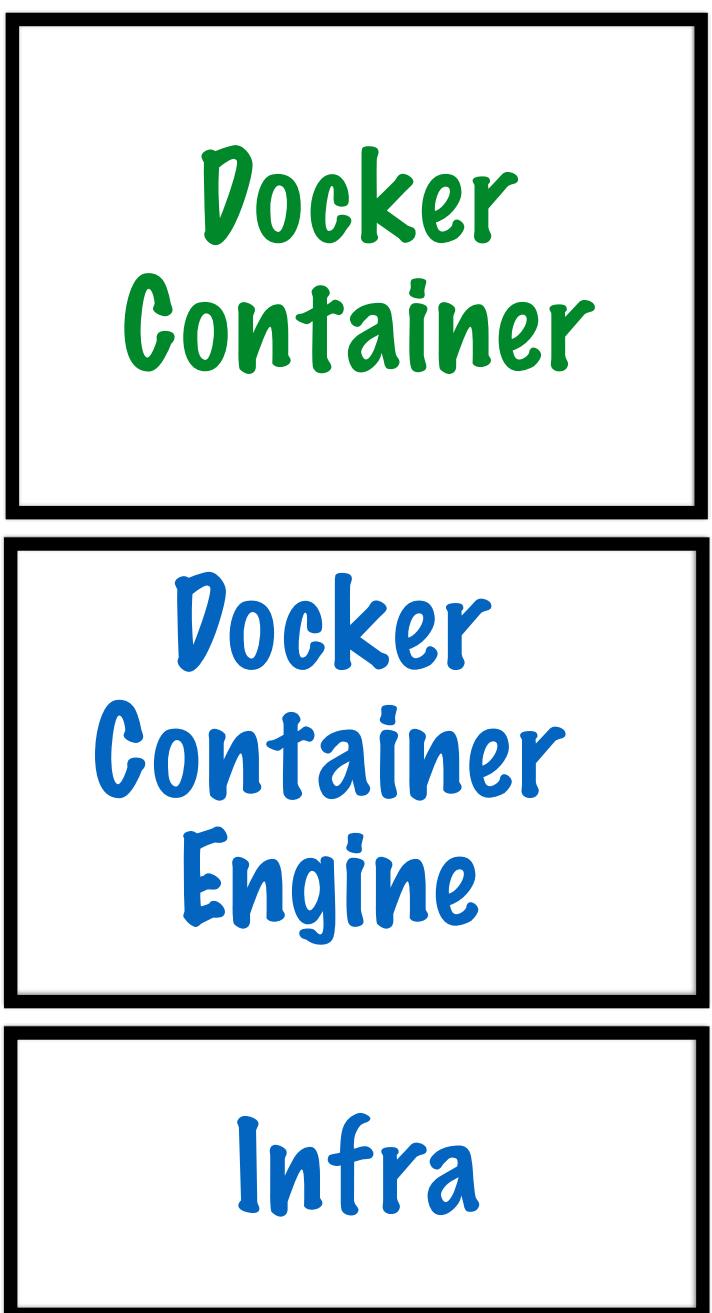
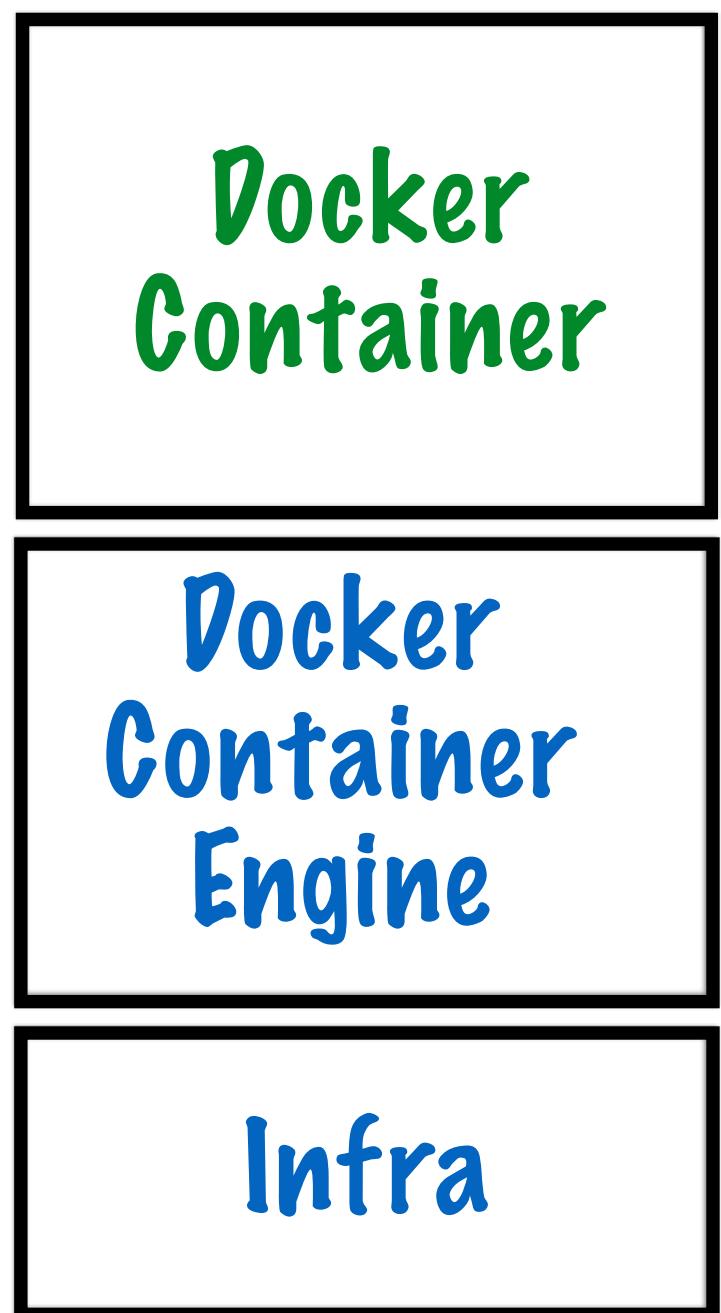
Cloud Instances

```
graph TD; A[Containers running on] --> B[Bare metal]; A --> C[VM Instances]; A --> D[Cloud Instances]
```

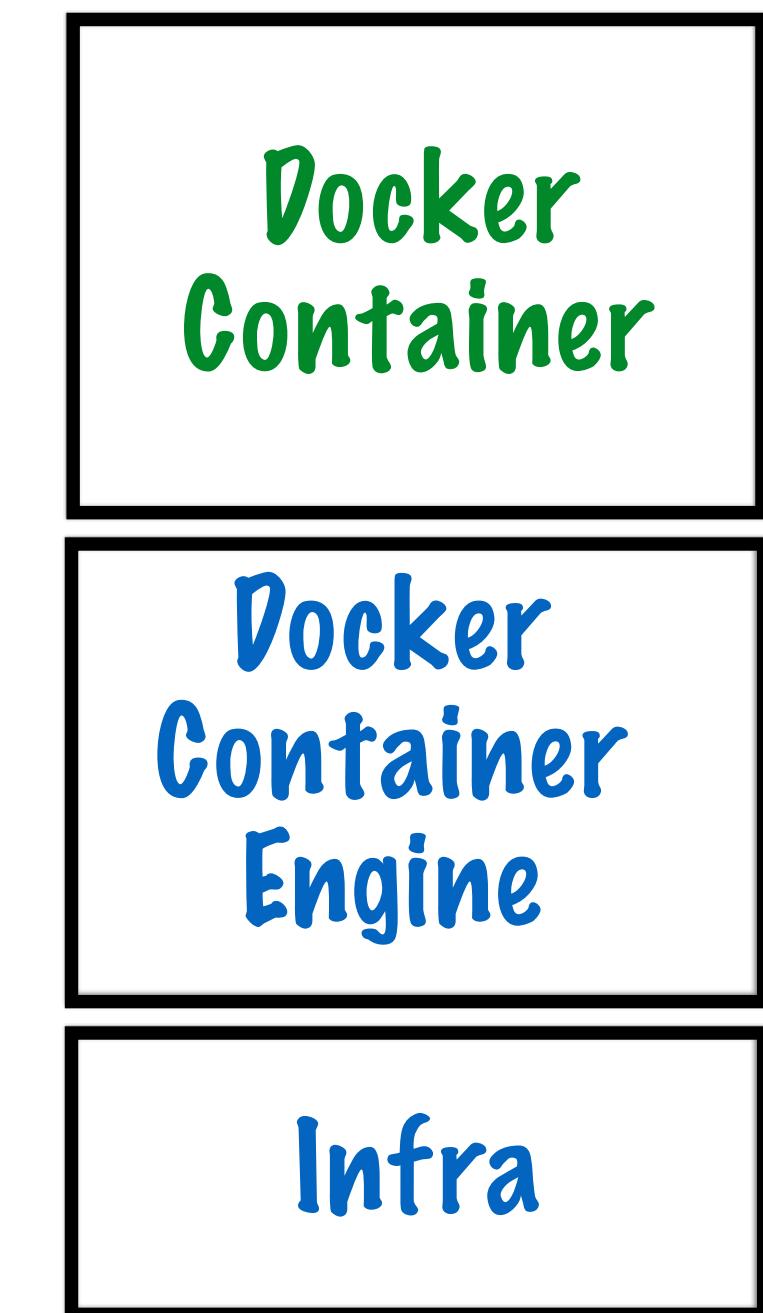
Potentially thousands of containers on hundreds of VMs

Isolated Infra

Potentially thousands of containers on hundreds of VMs

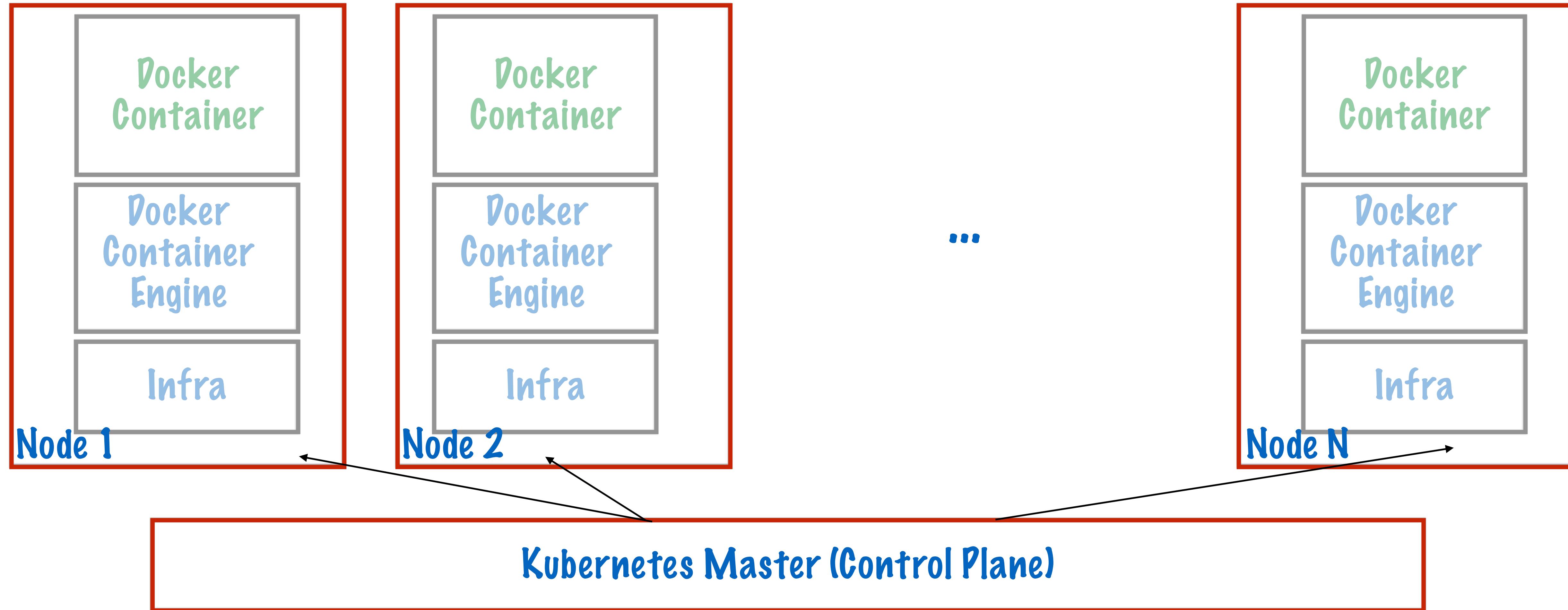


...



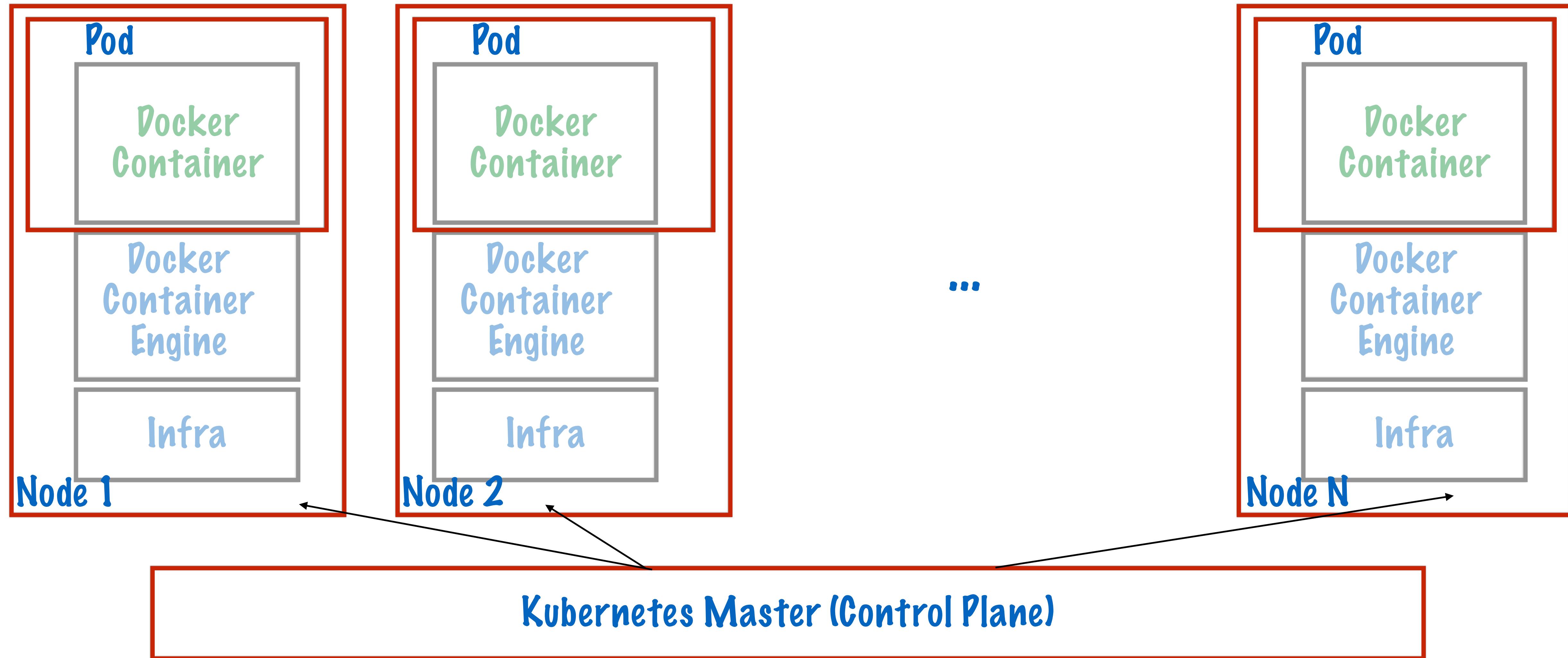
Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs

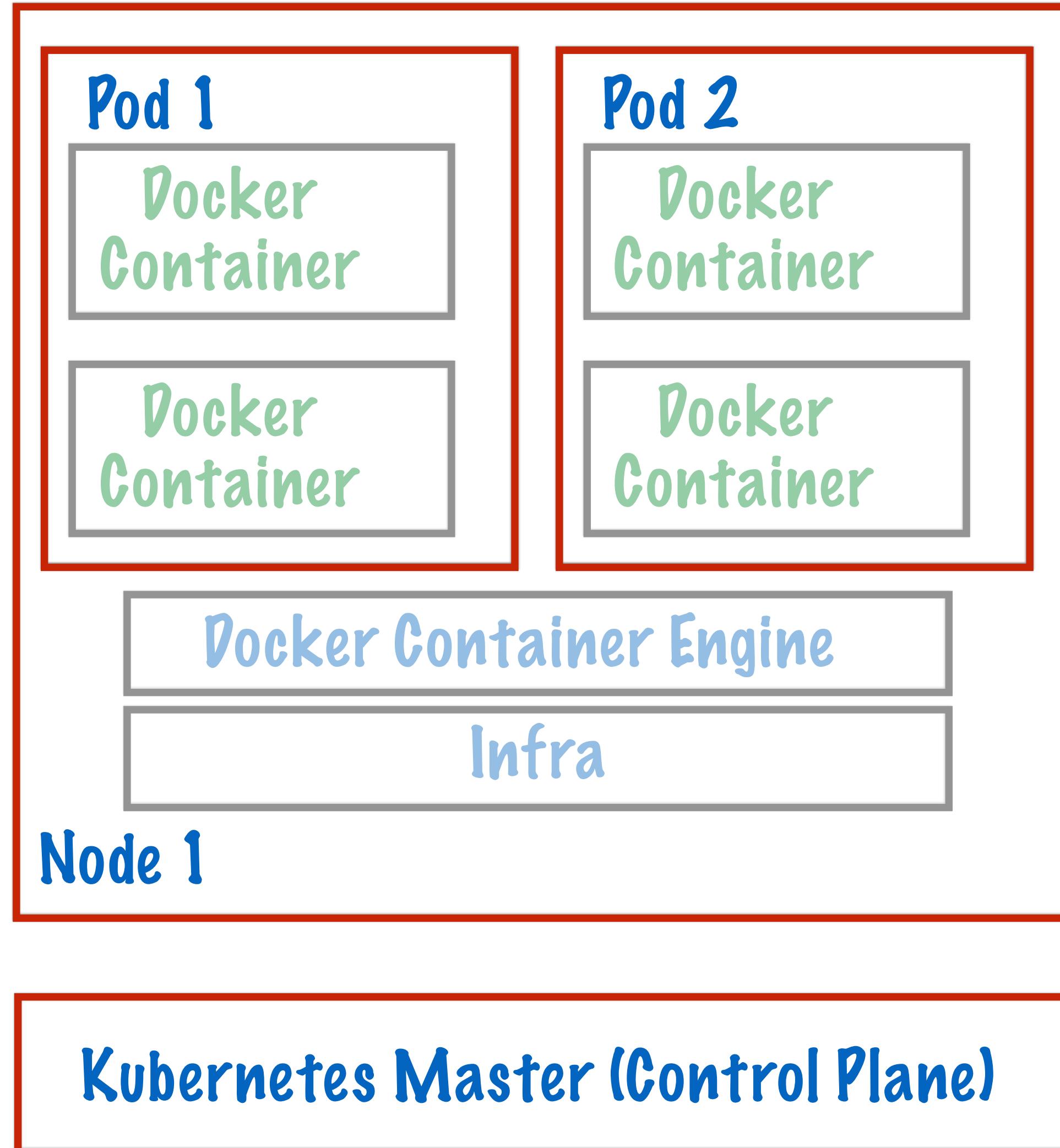


Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs



Pods on Kubernetes Nodes

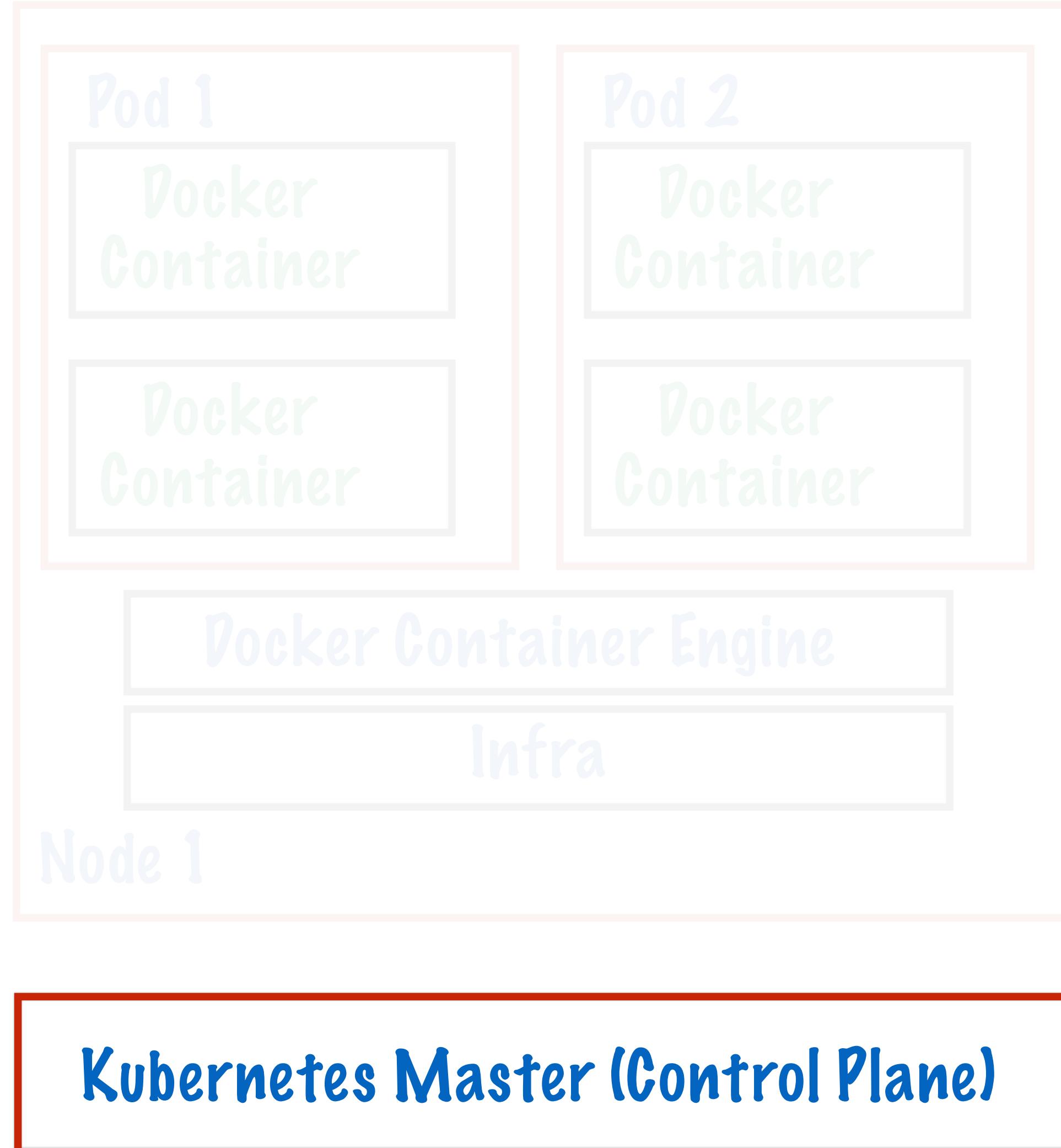


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes

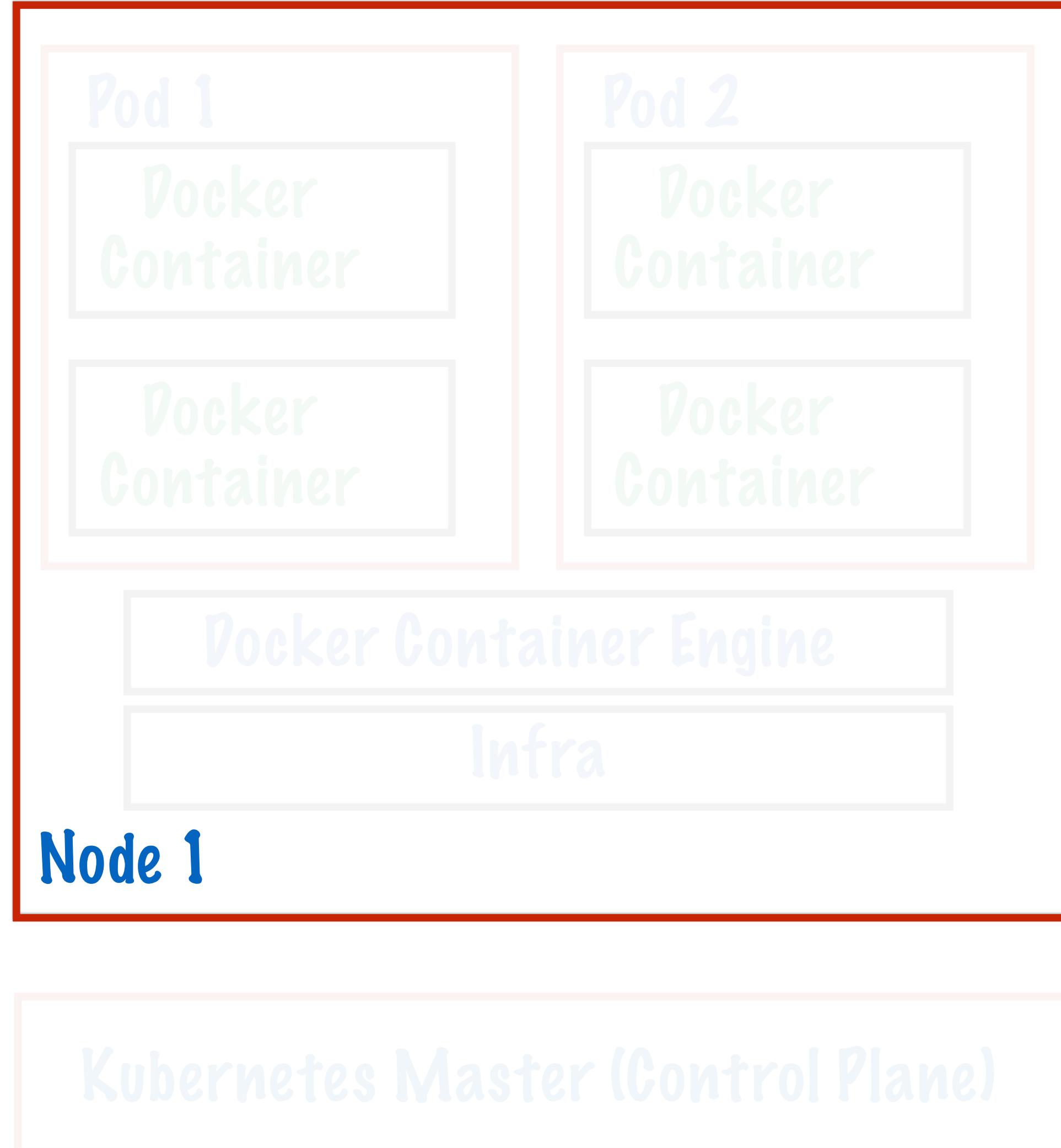


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes

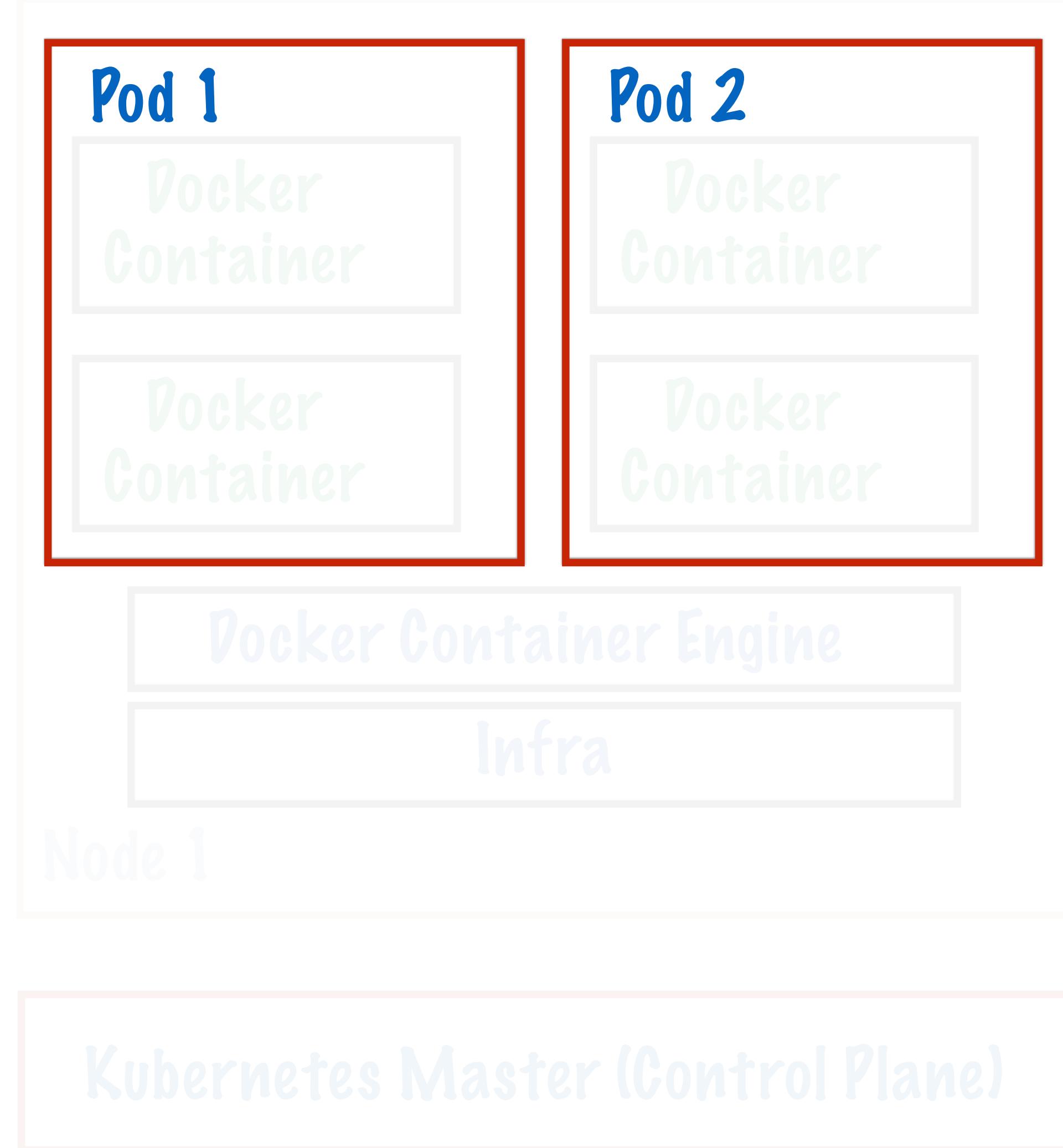


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes

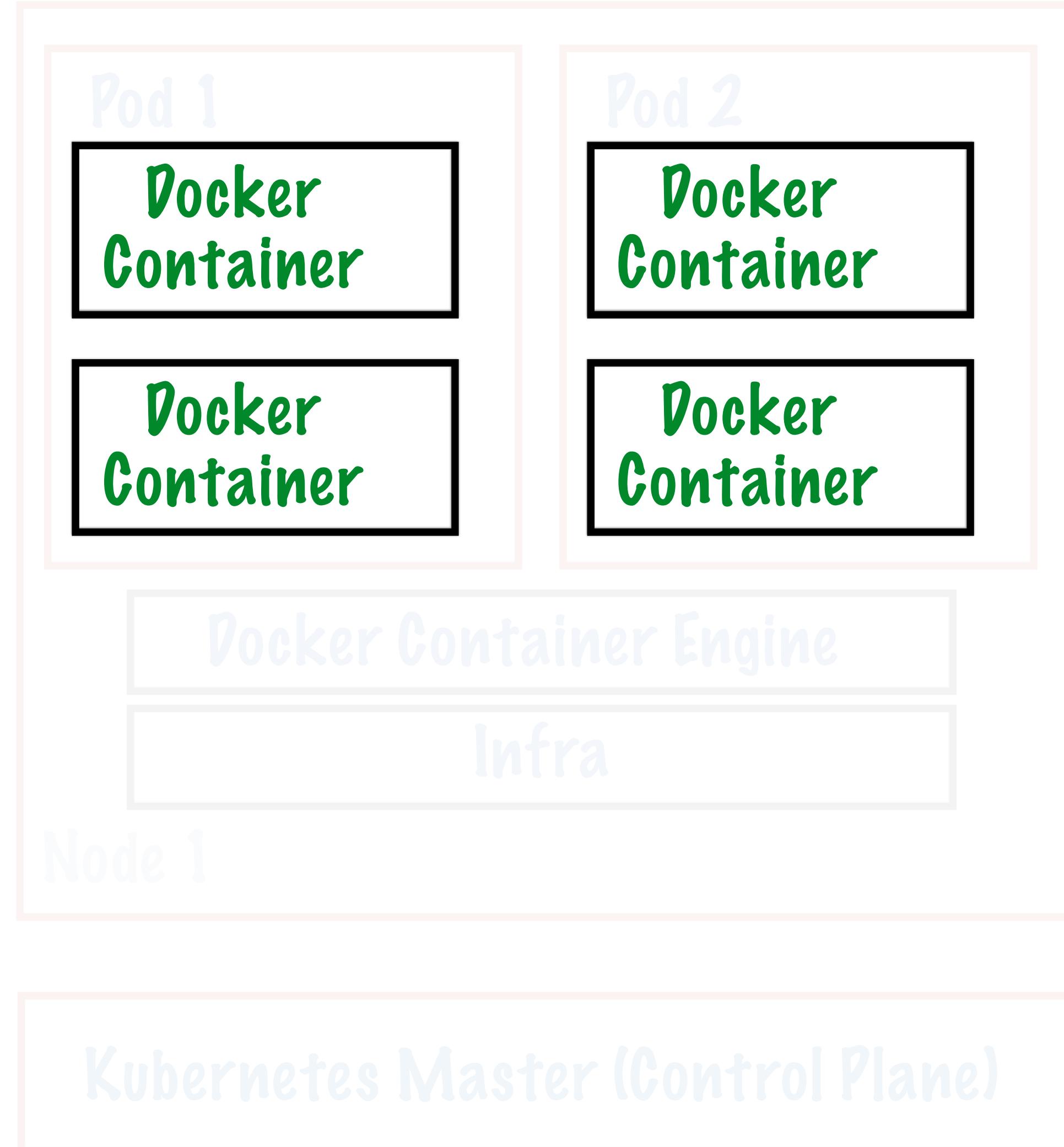


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes



Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

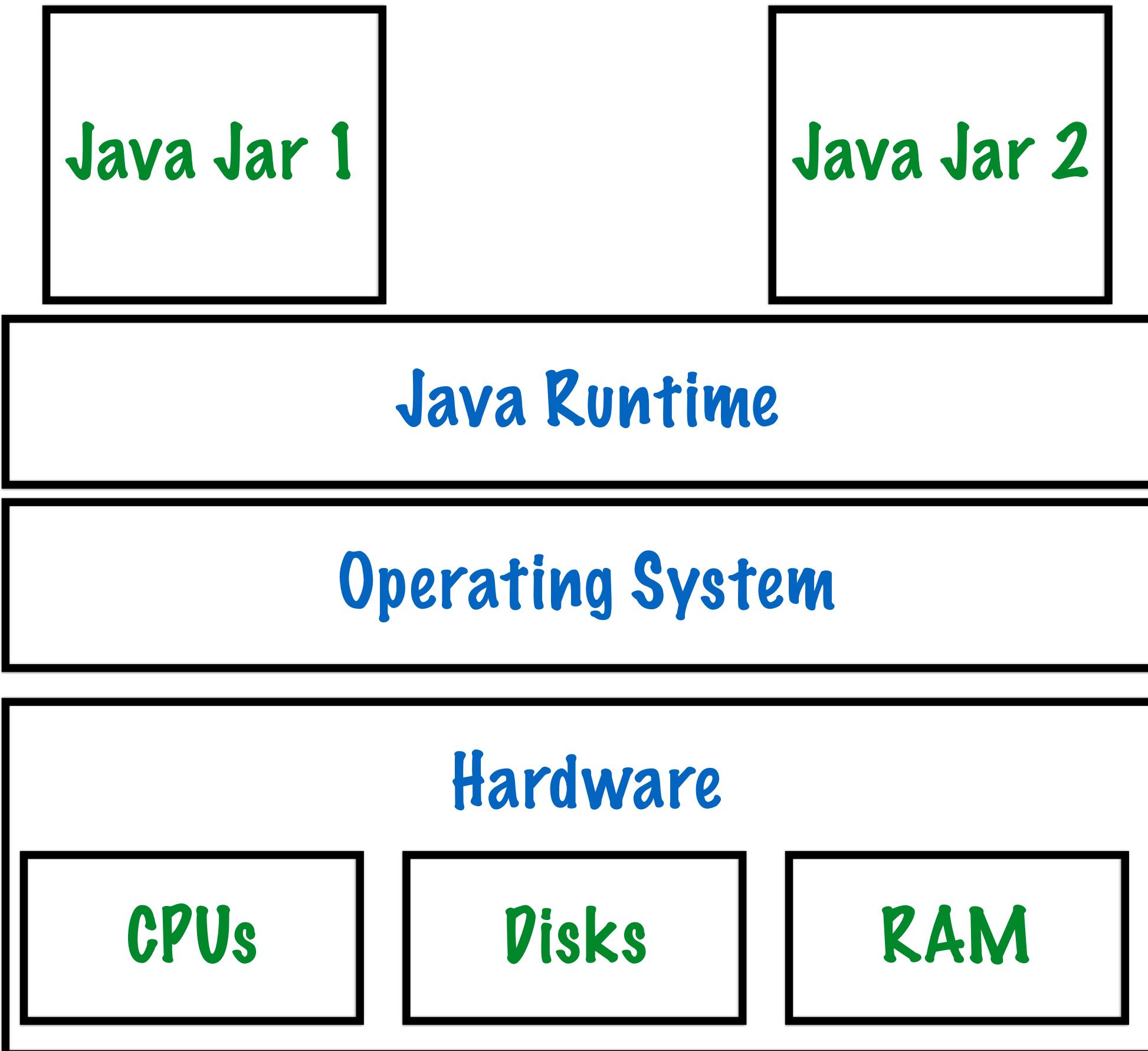
Pod runs on node, which is controlled by master

Kubernetes :: Hadoop

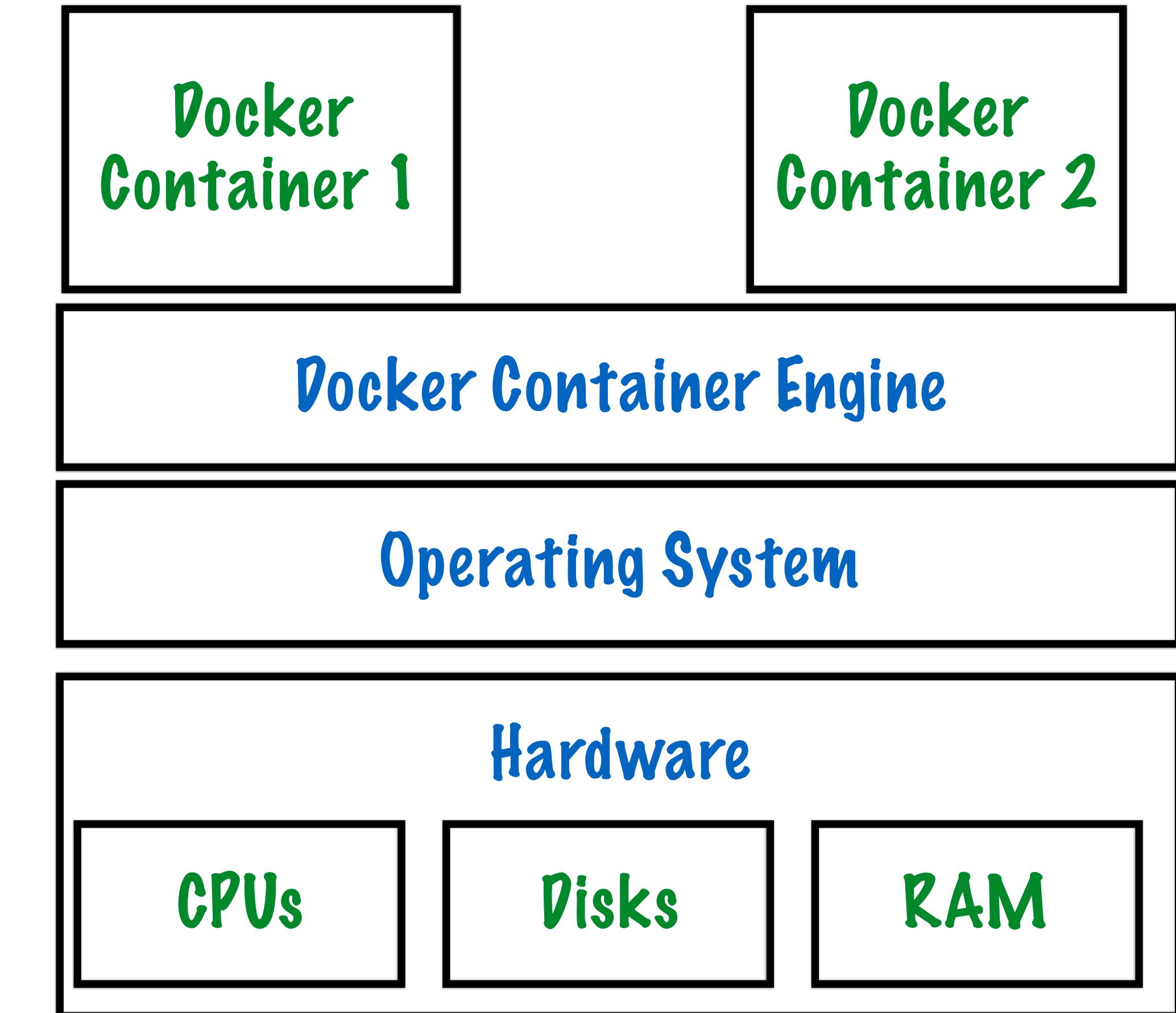
- Master
- Workers
- Job Scheduling
- Resource Allocation

Containers Are 'Like' Jars

Java Jars



Docker Containers



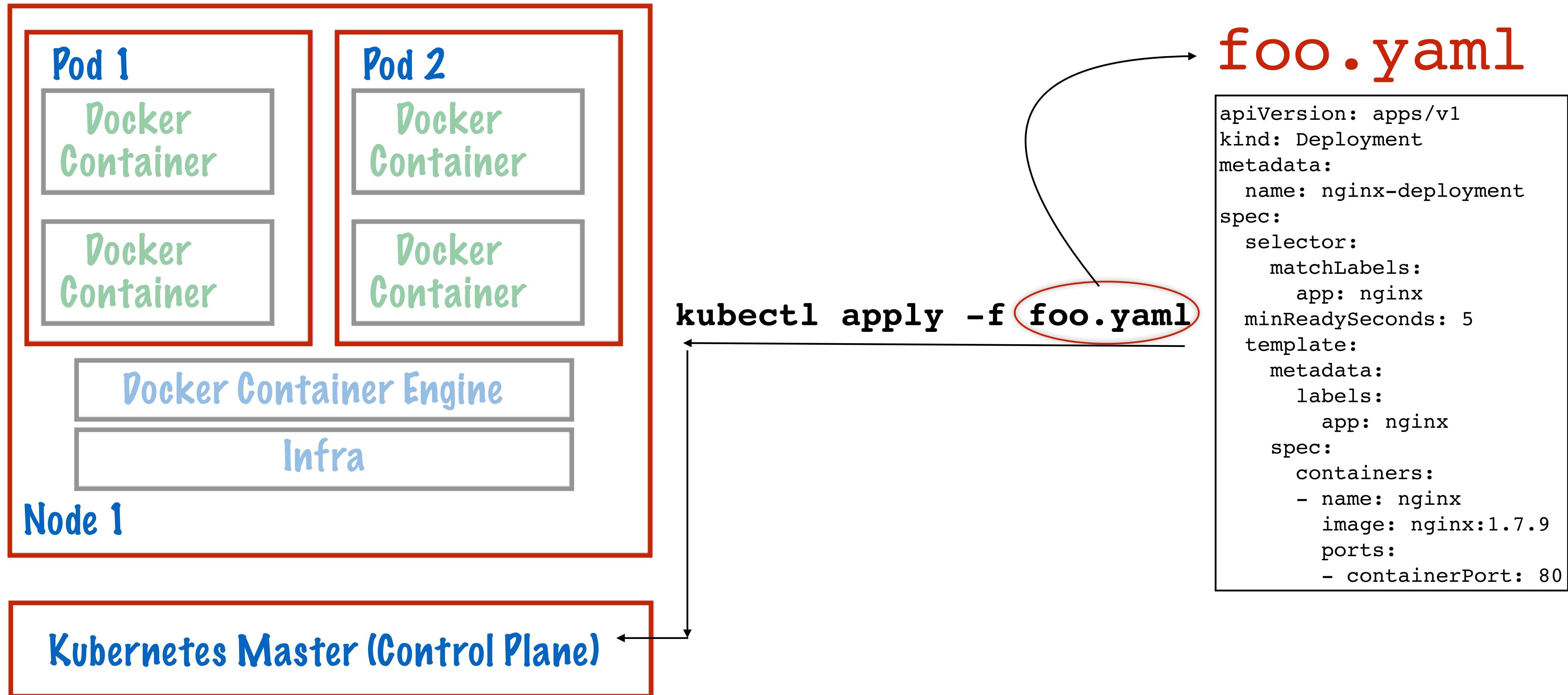
Kubernetes :: Hadoop

- Docker Container Engine -> Java Runtime
- Docker containers -> Jars
- Kubernetes -> Hadoop

Kubernetes for Orchestration

- **Fault-tolerance:** Pod/Nod failures
- **Rollback:** Advanced deployment options
- **Auto-healing:** Crashed containers restart
- **Auto-scaling:** More clients? More demand
- **Load-balancing:** Distribute client requests
- **Isolation:** Sandboxes so that containers don't interfere

Working with Kubernetes: `kubectl`



What Does the Kubernetes Master Do?

Kubernetes Cluster

Containers running on

Bare metal

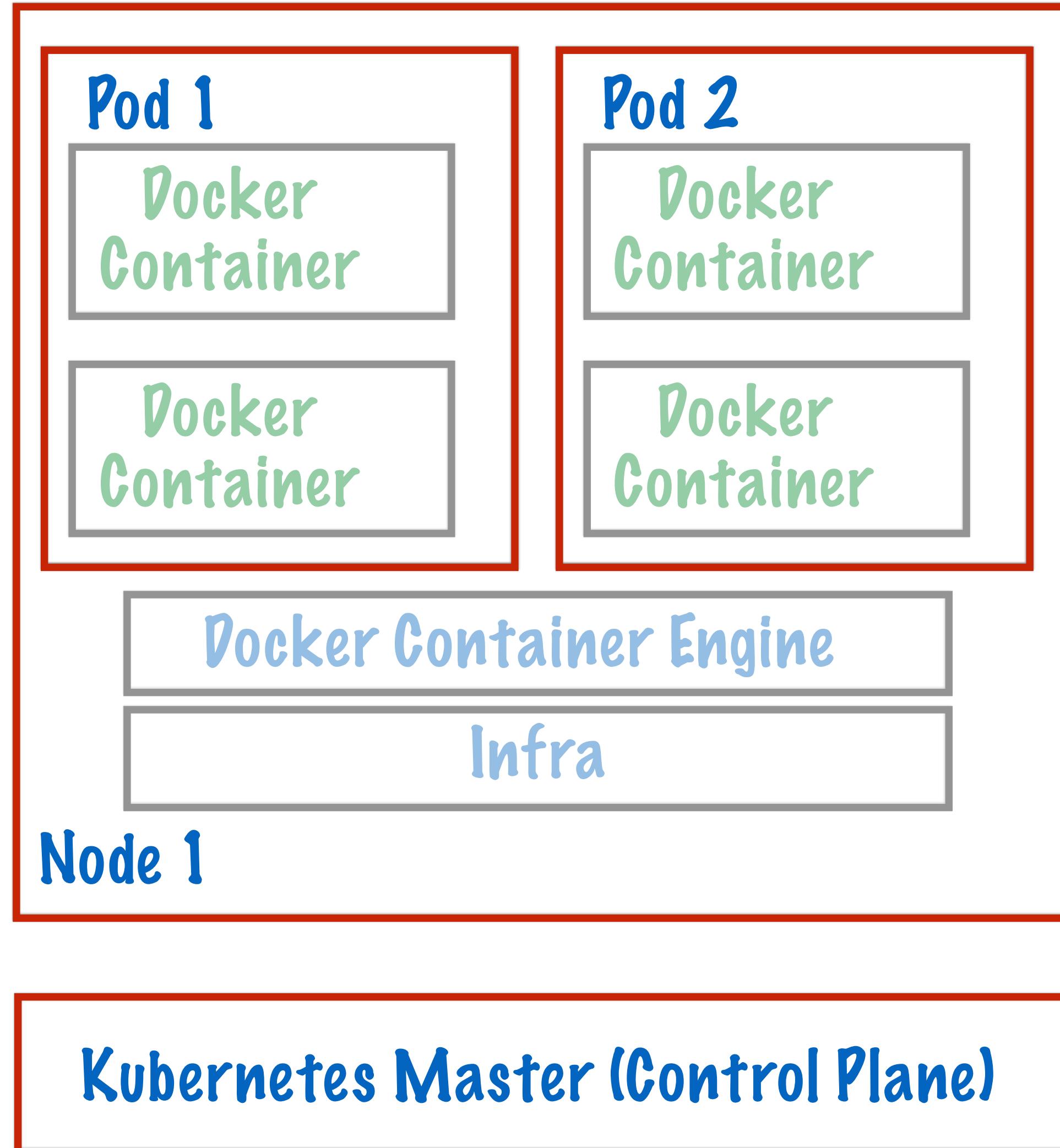
VM Instances

Cloud Instances

```
graph TD; A[Kubernetes Cluster] --> B[Containers running on]; B --> C[Bare metal]; B --> D[VM Instances]; B --> E[Cloud Instances]
```

Kubernetes designates one or more of these the master

Kubernetes Master



One or more nodes designated as master

Several kubernetes processes run on master

Multi-master for high-availability

Kubernetes Master



One or more nodes designated as master

Several kubernetes processes run on master

Multi-master for high-availability

kube-apiserver

Kubernetes Master (Control Plane)

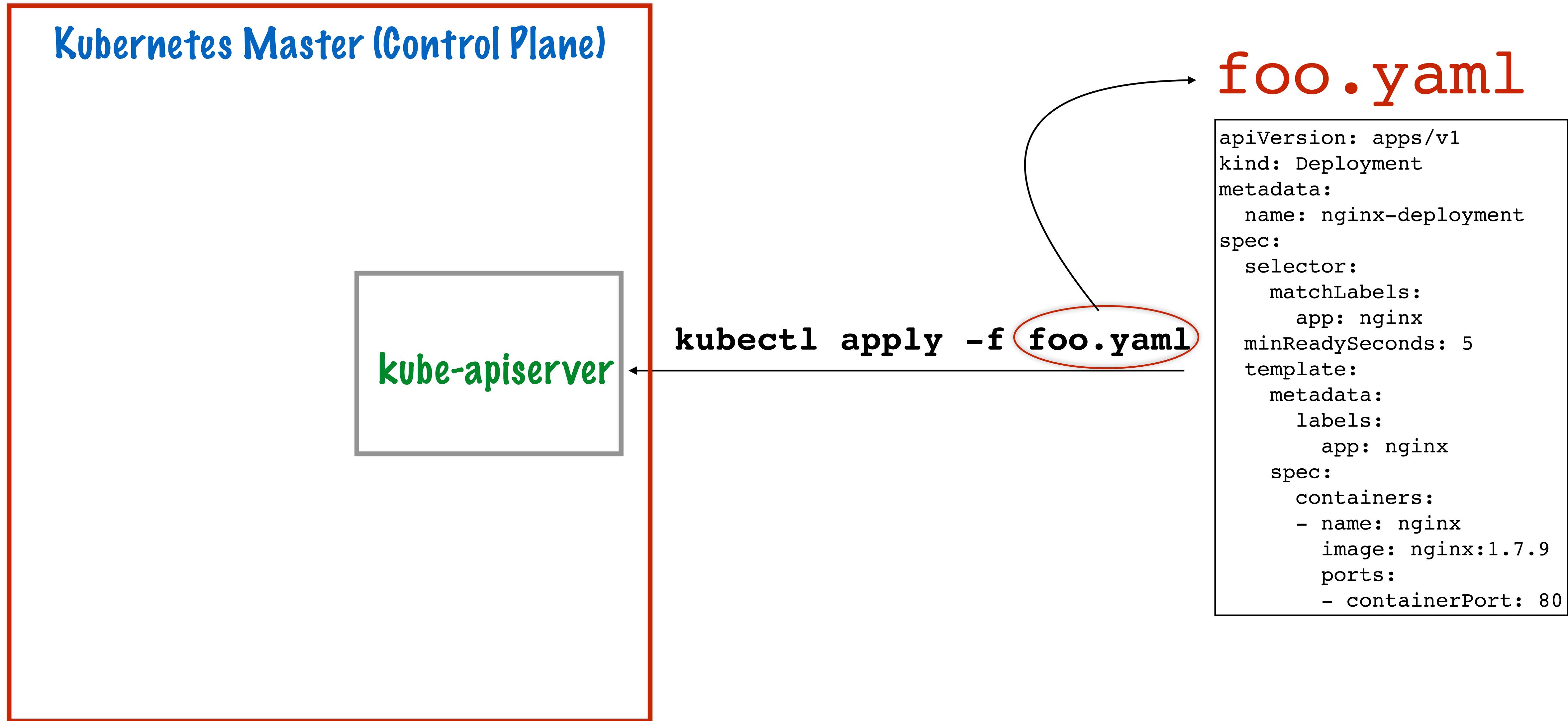
kube-apiserver

Communicates with user

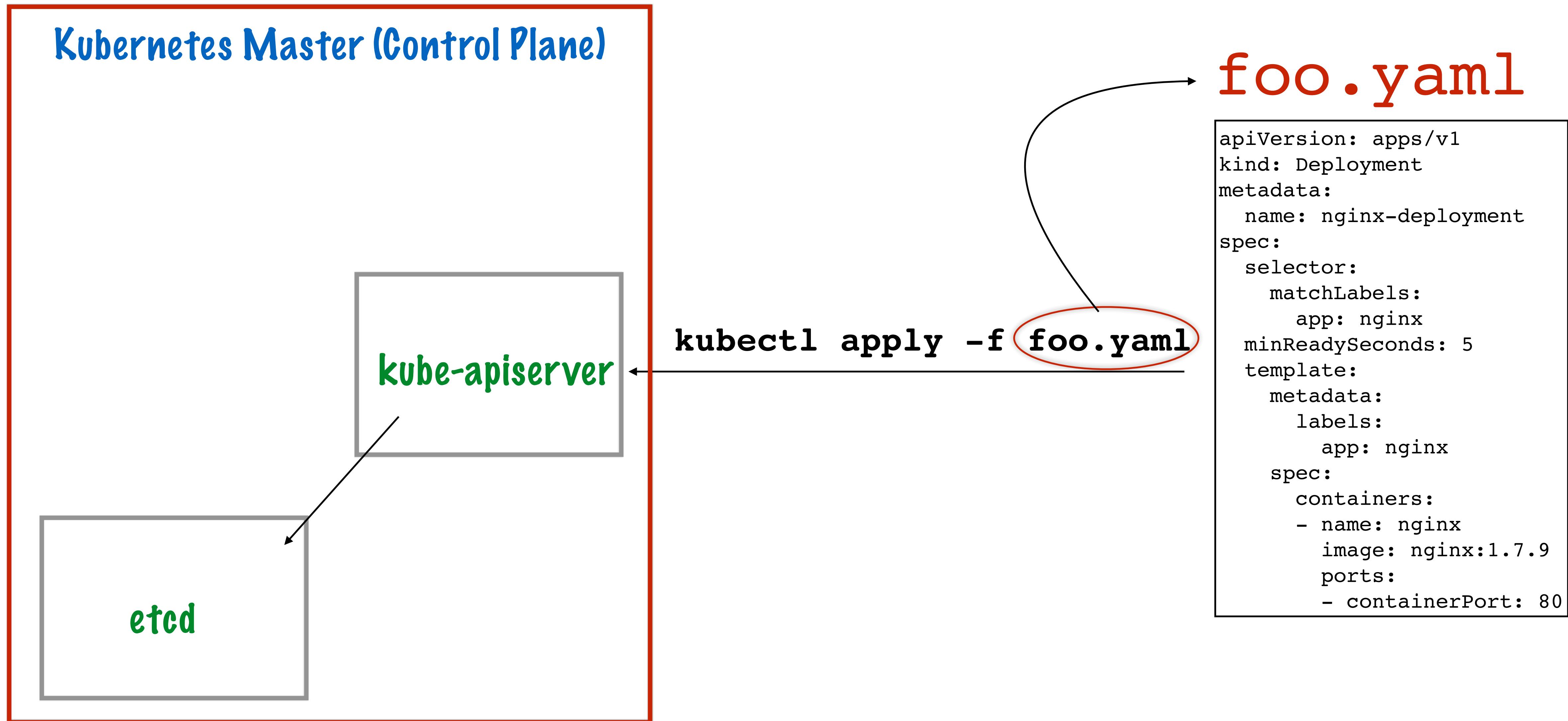
RESTful API end-points

Manifest yaml files are accepted by apiserver

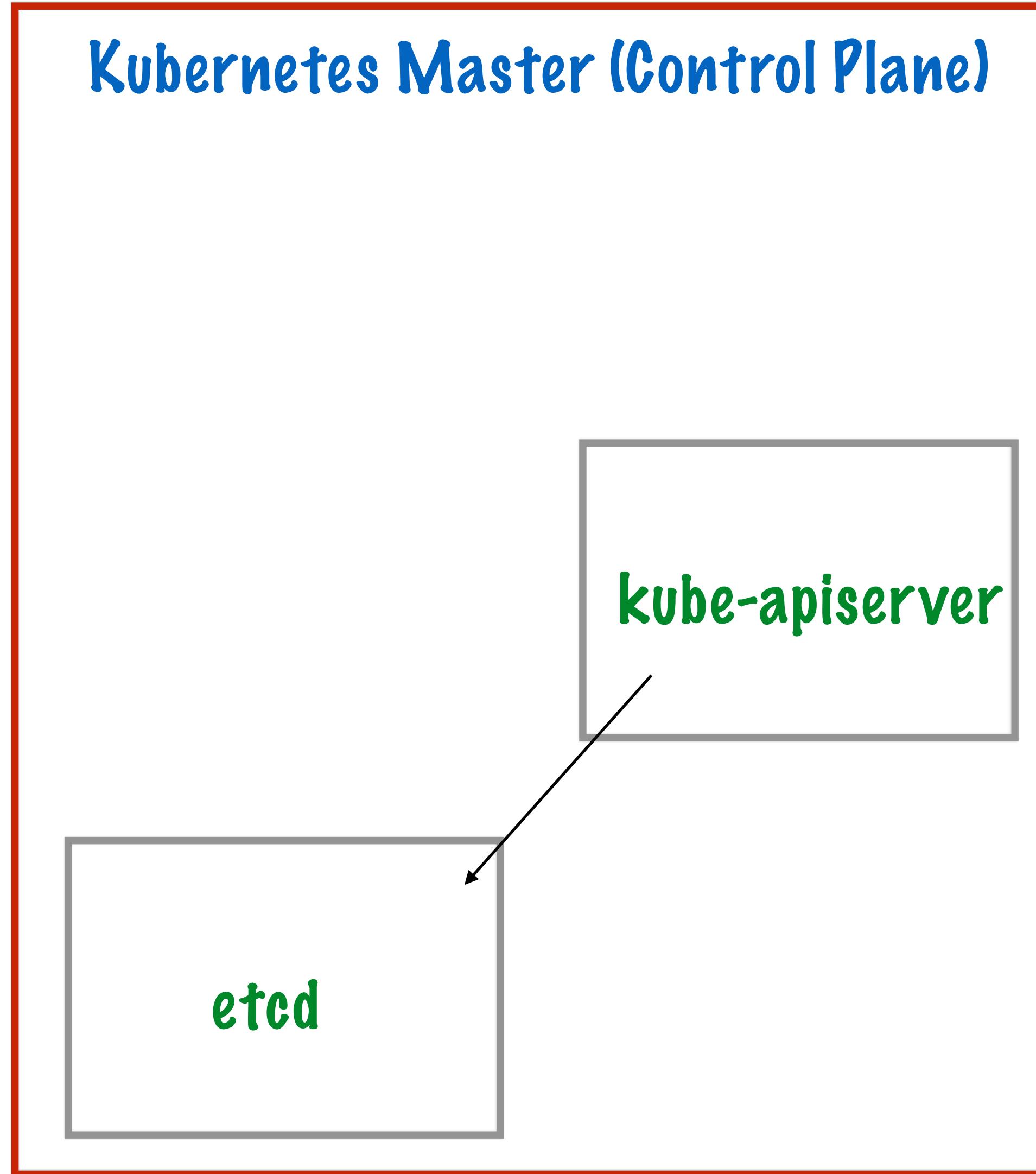
kube-apiserver



Cluster Store for Metadata



Cluster Store for Metadata



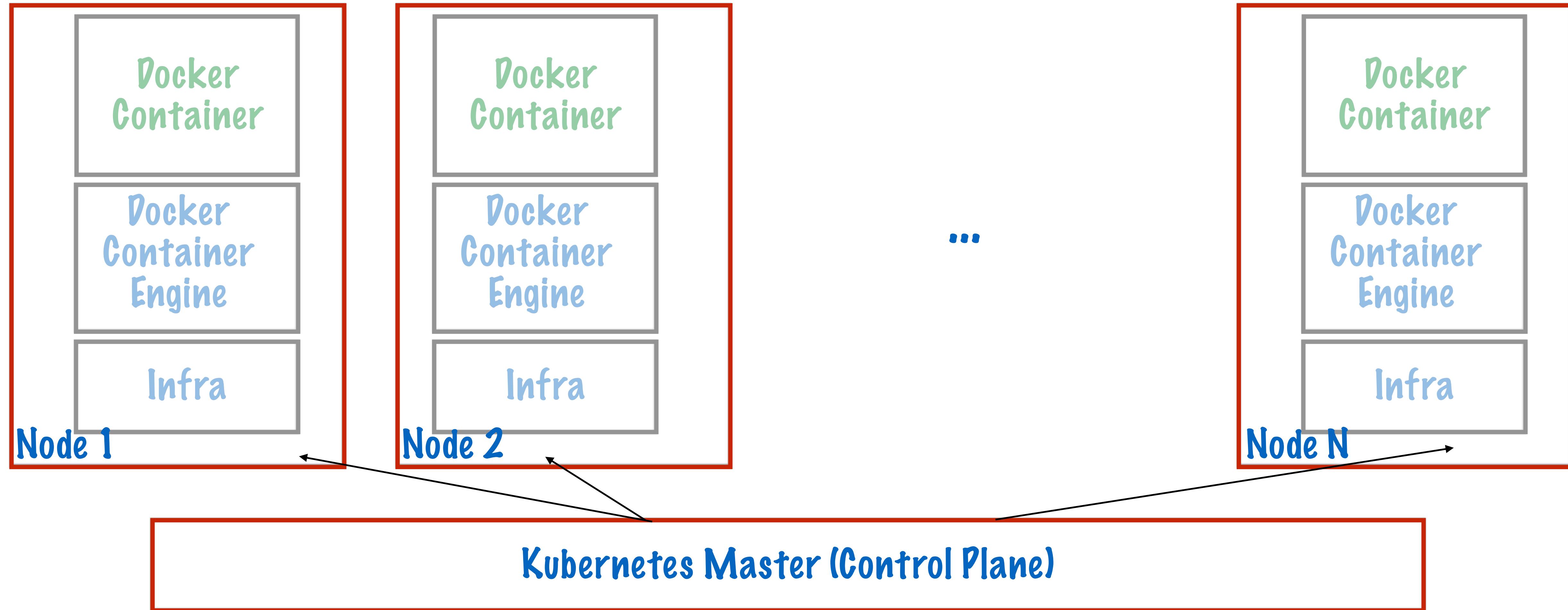
Metadata about spec and status of cluster

etcd is consistent and highly available key-value store

source-of-truth for cluster state

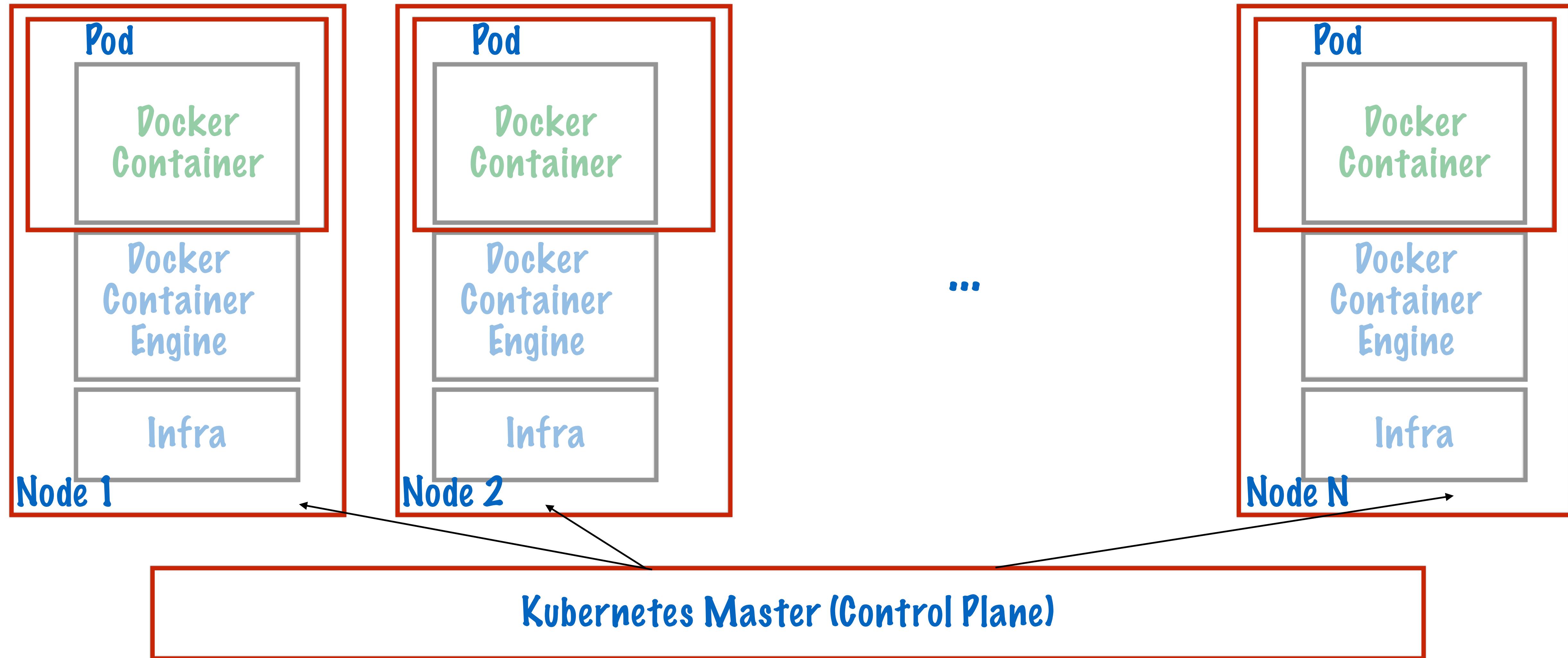
Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs

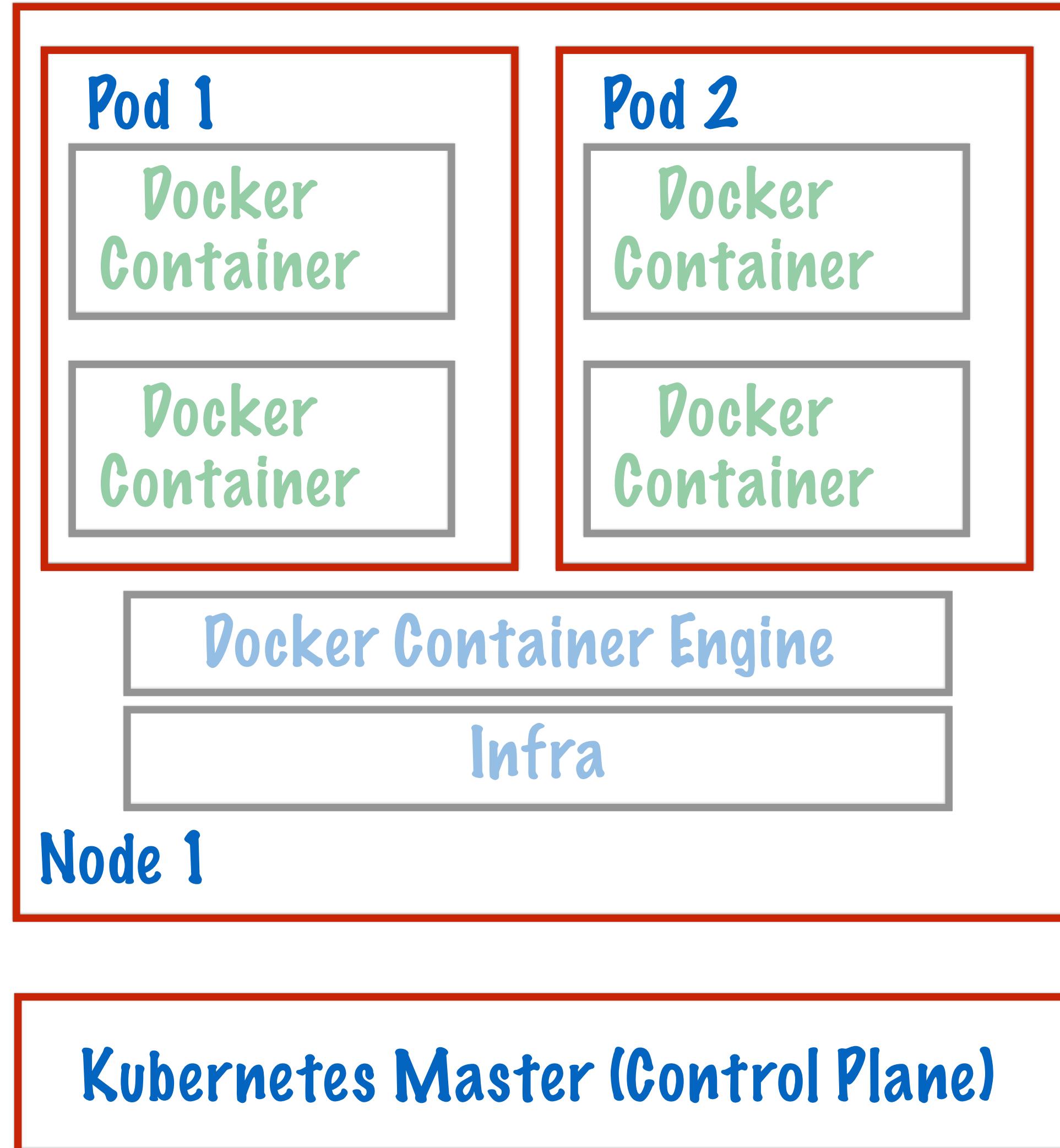


Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs



Pods on Kubernetes Nodes

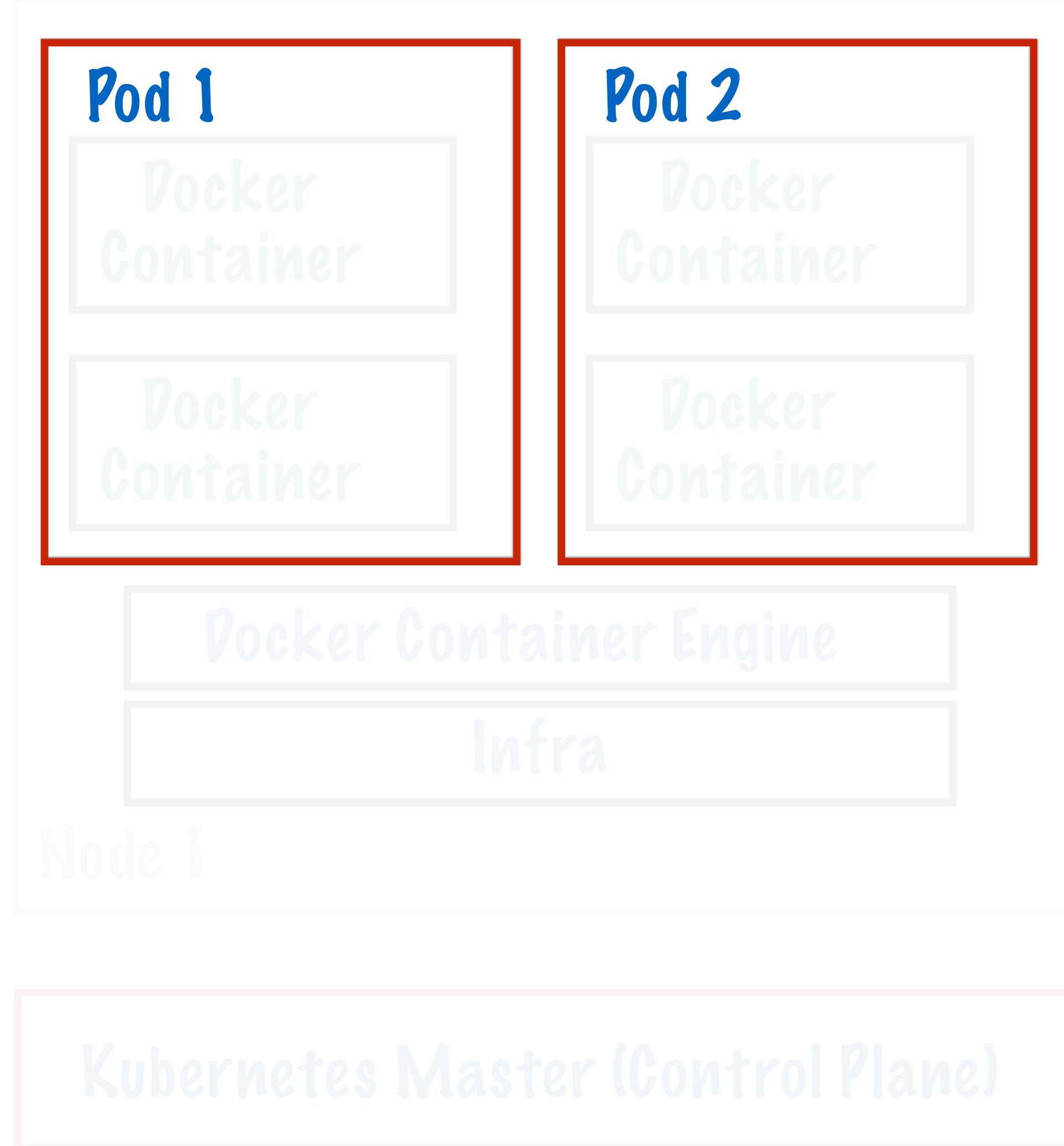


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes

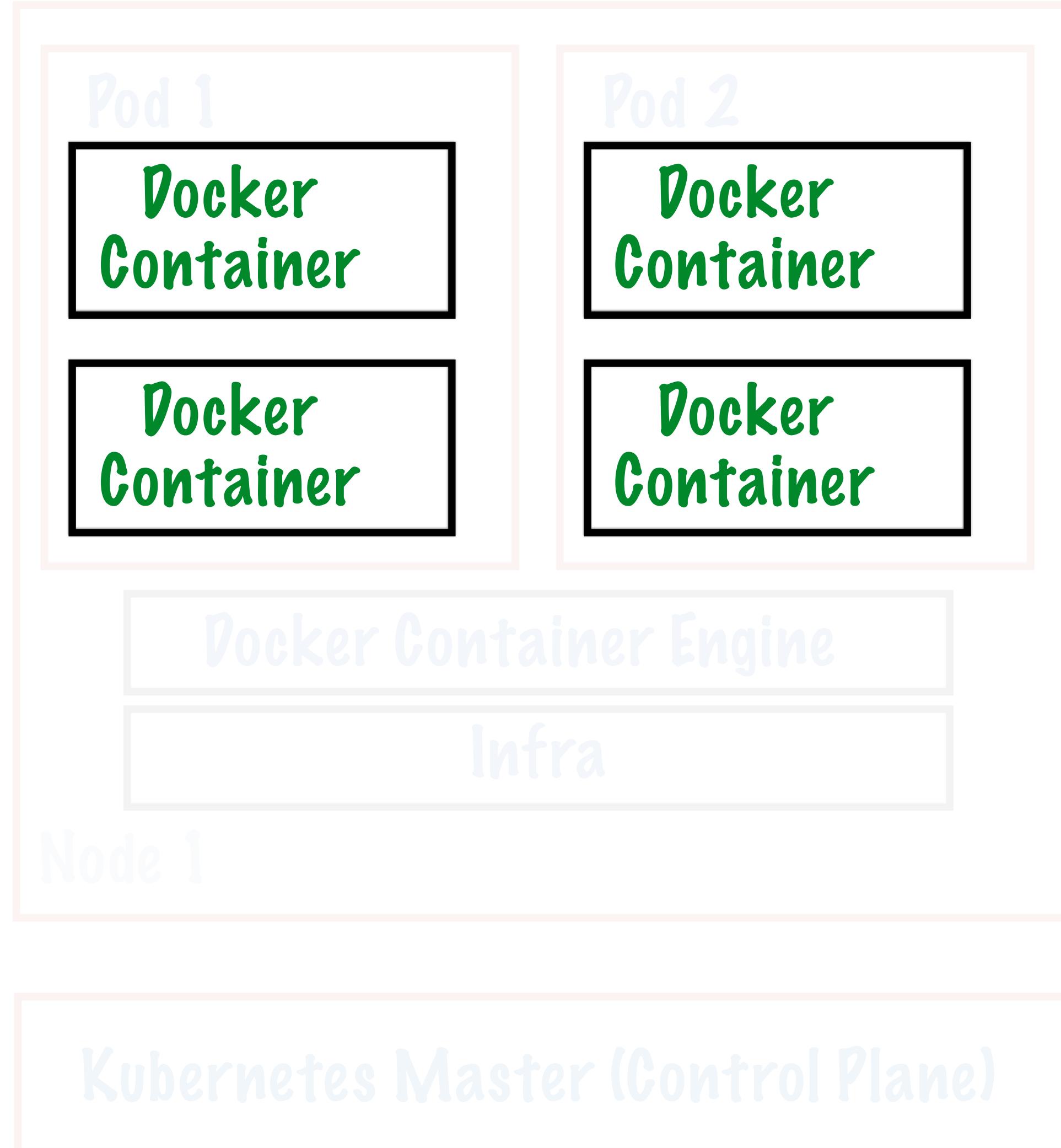


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes

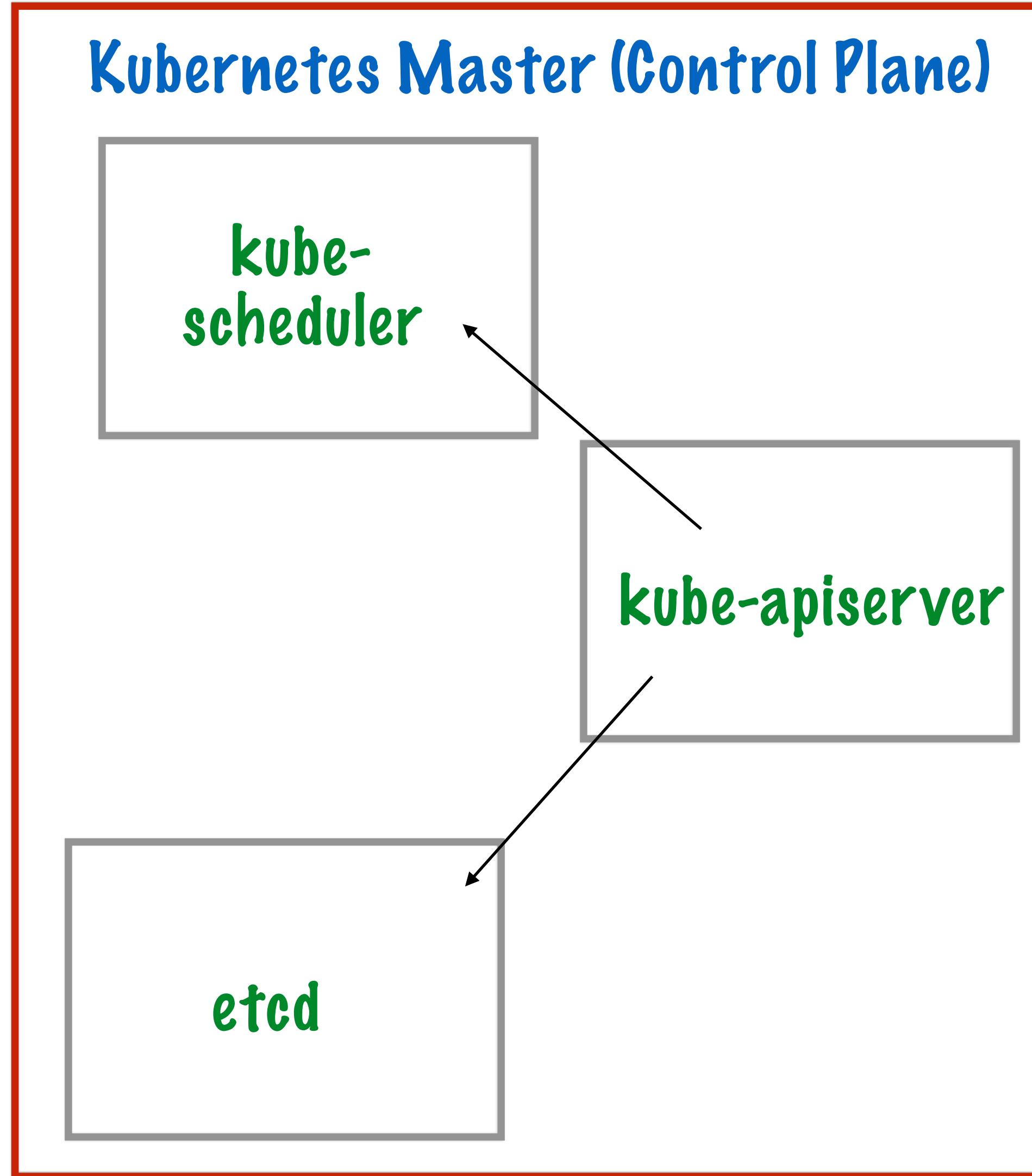


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

kube-scheduler

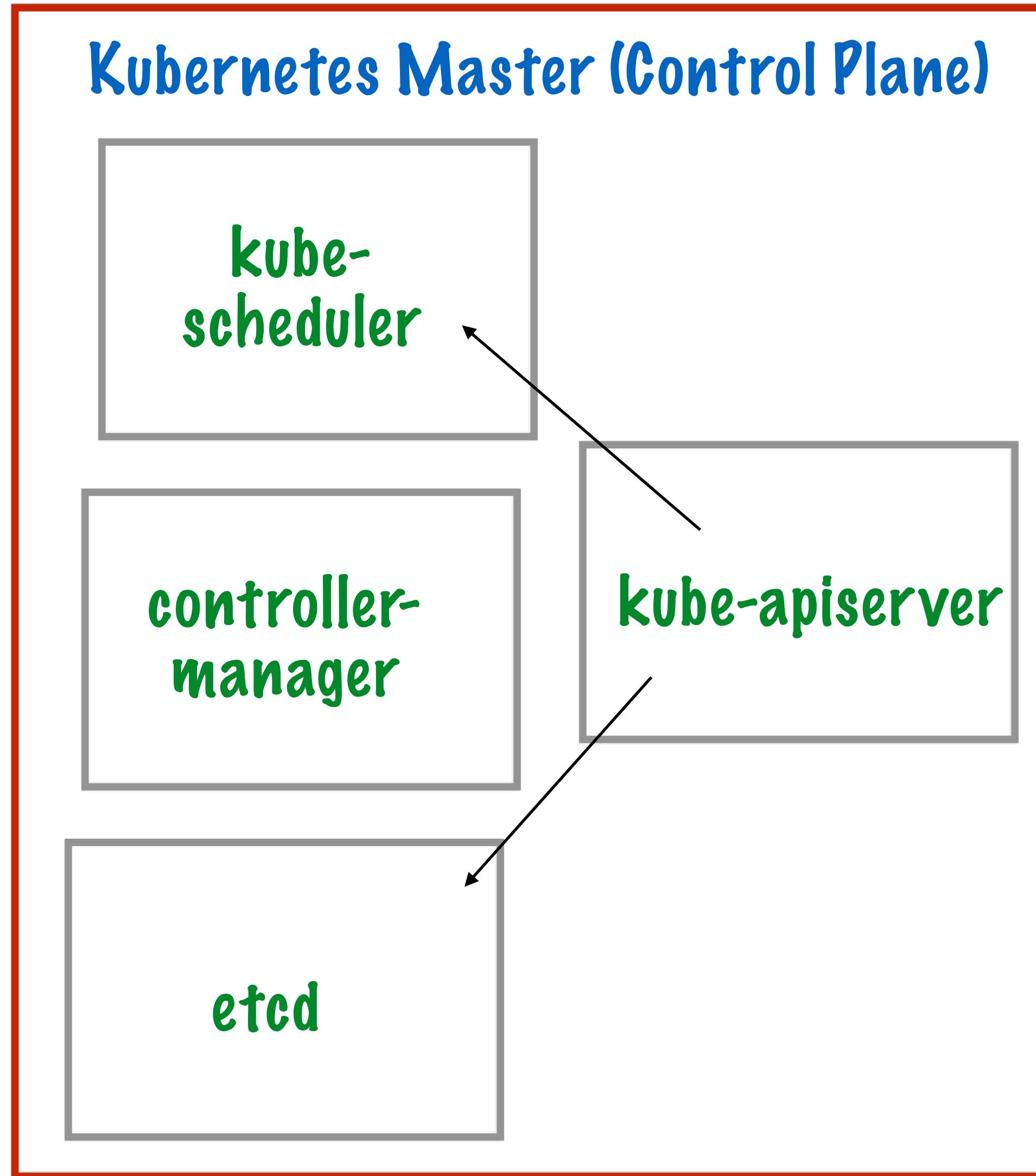


Handle pod creation and management

kube-scheduler match/assign nodes to pods

Complex - affinities, taints, tolerations, ...

controller-manager



Different master processes

Actual state <-> Desired State

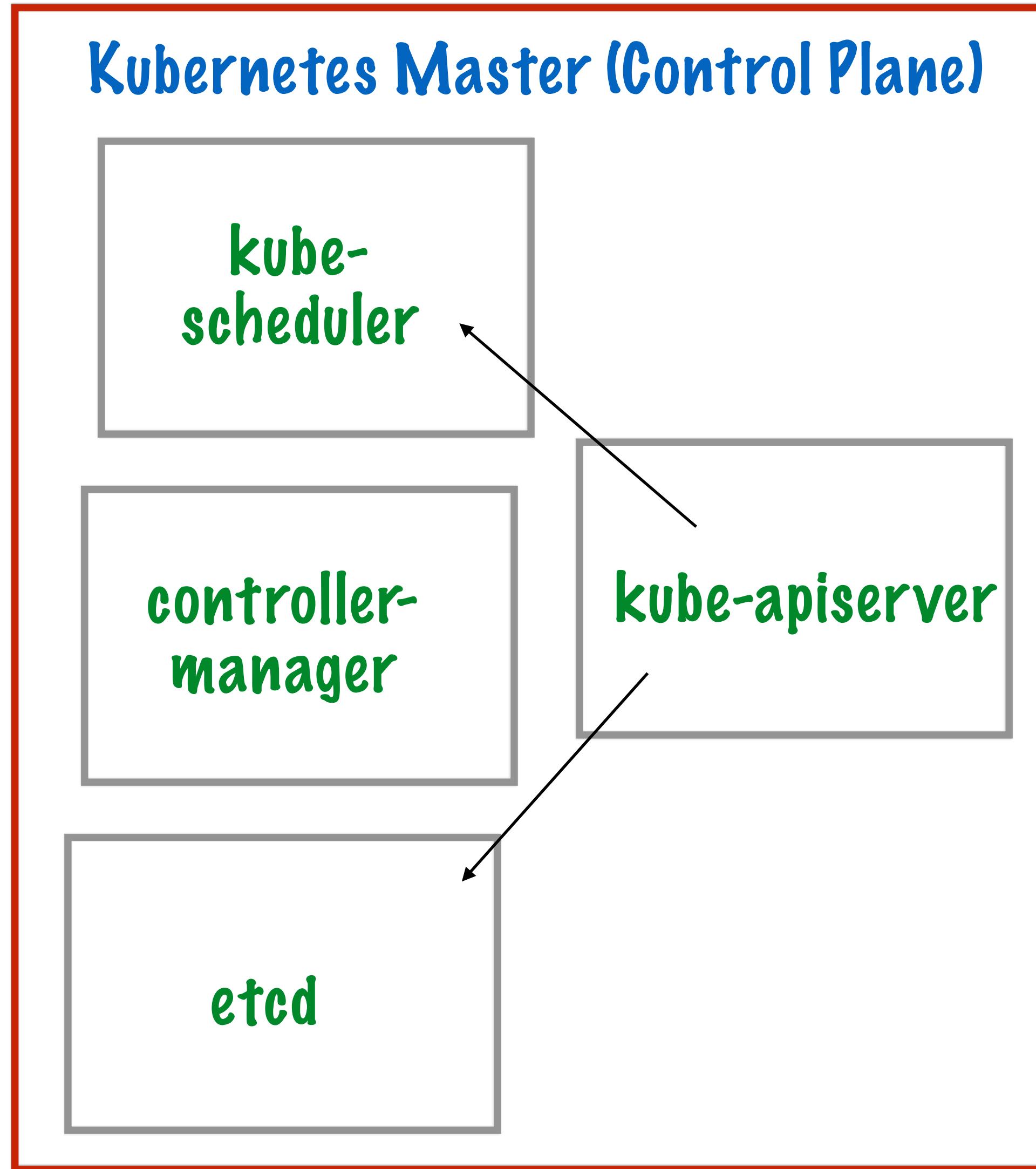
cloud-controller-manager

kube-controller-manager

controller-manager

- Node controller
- Replication controller
- Route Controller
- Volume Controller

Control Plane



Apiserver

etcd

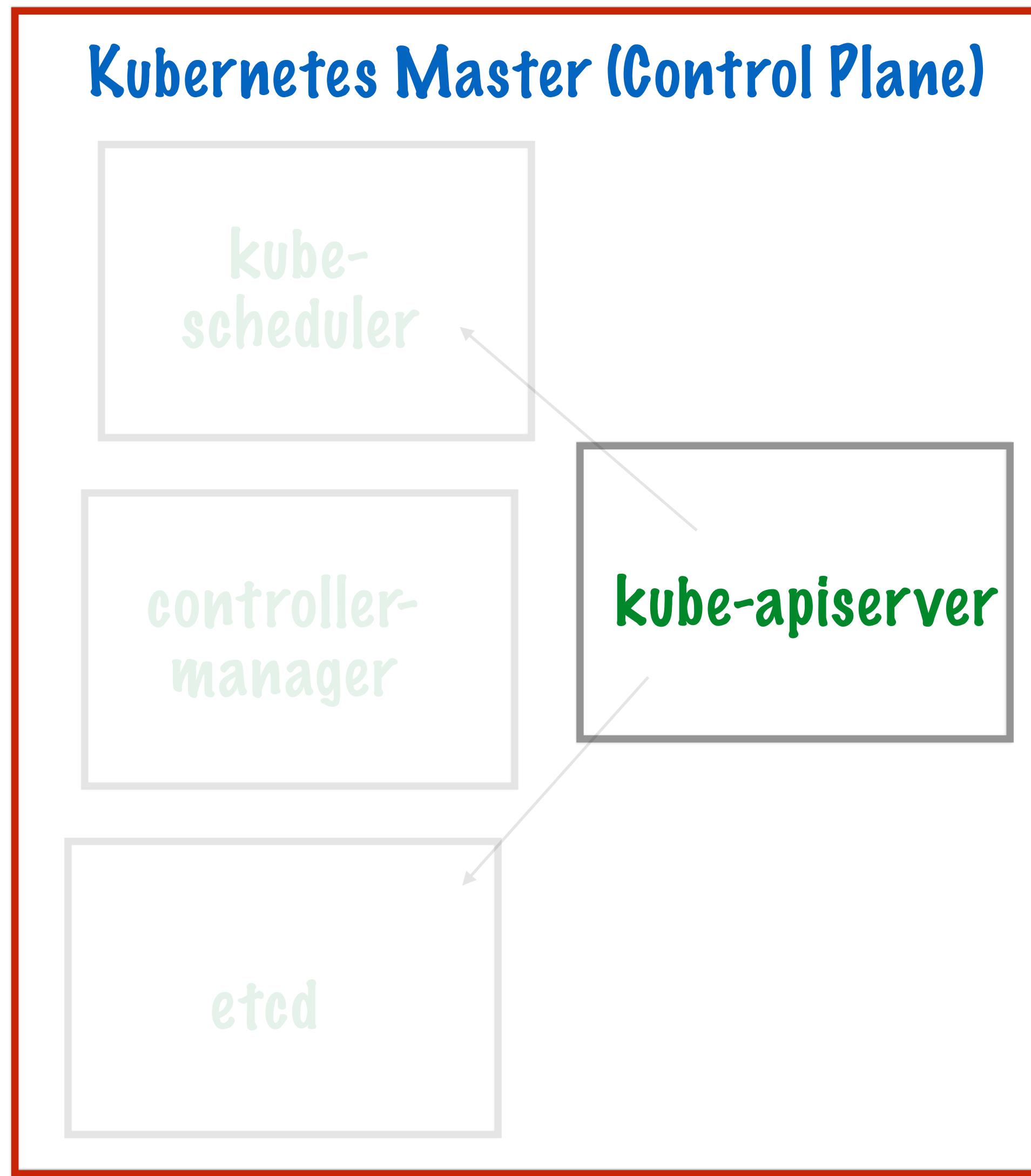
scheduler

controller-manager

cloud-controller-manager

kube-controller-manager

Control Plane



Apiserver

etcd

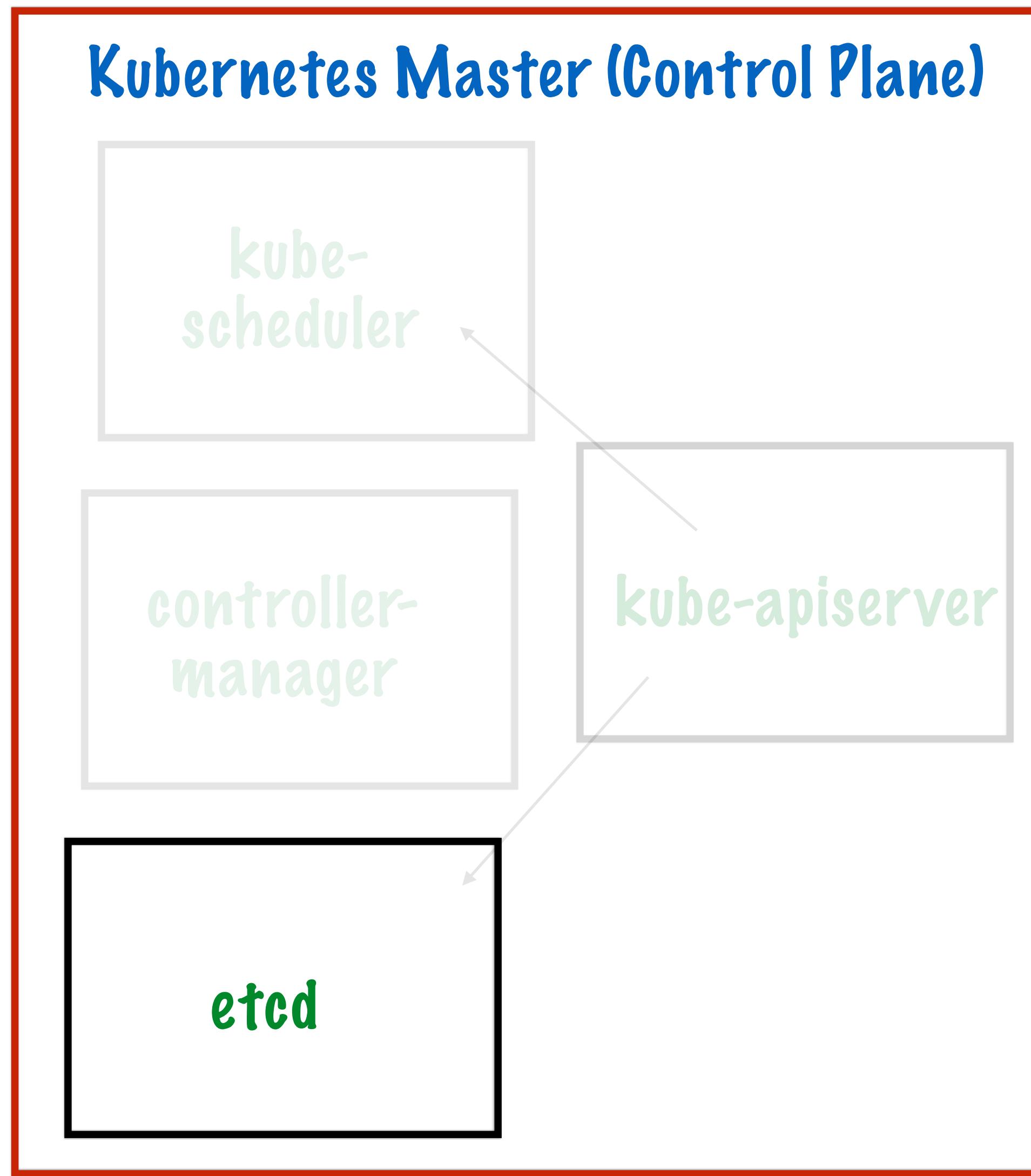
scheduler

controller-manager

cloud-controller-manager

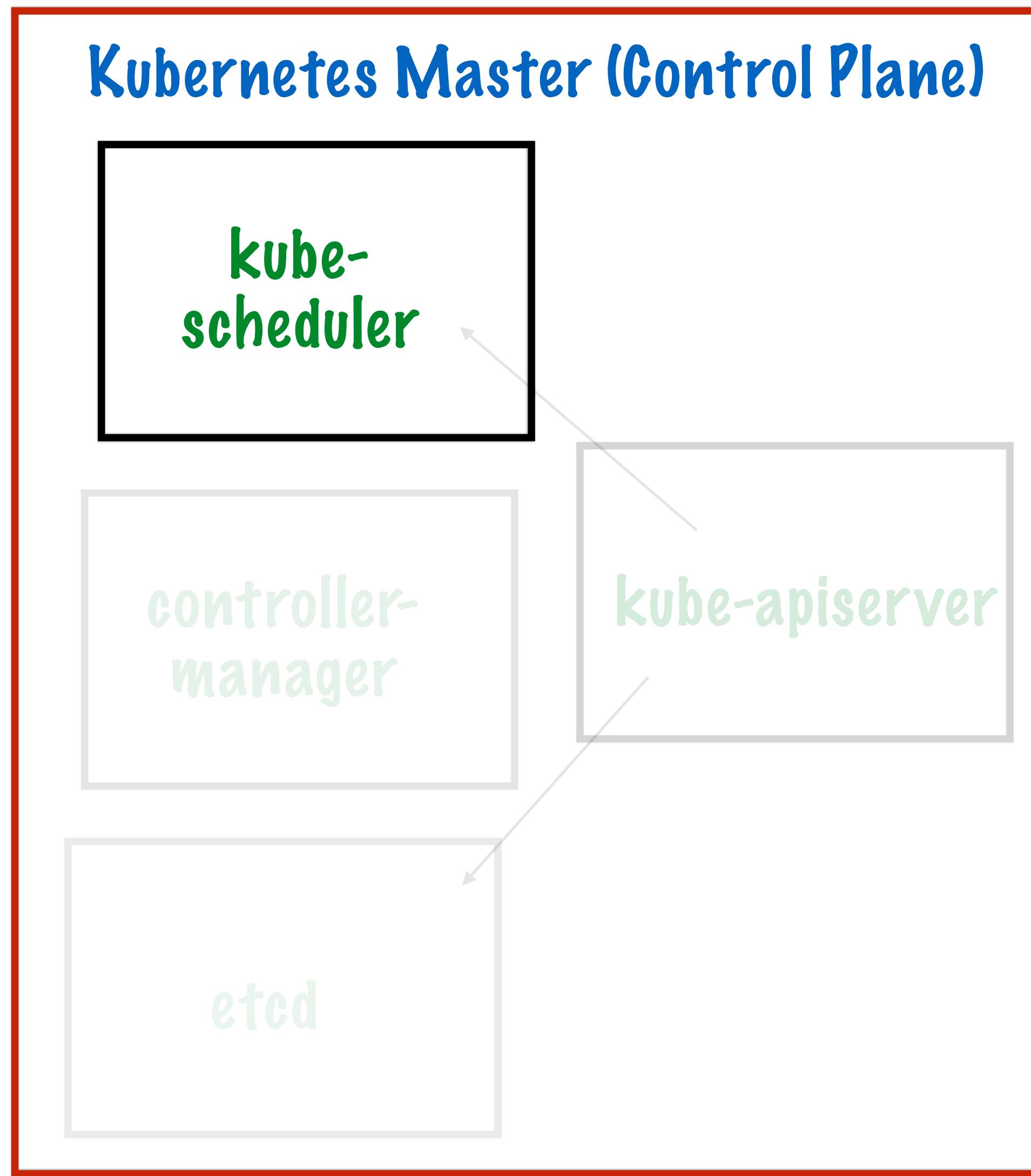
kube-controller-manager

Control Plane



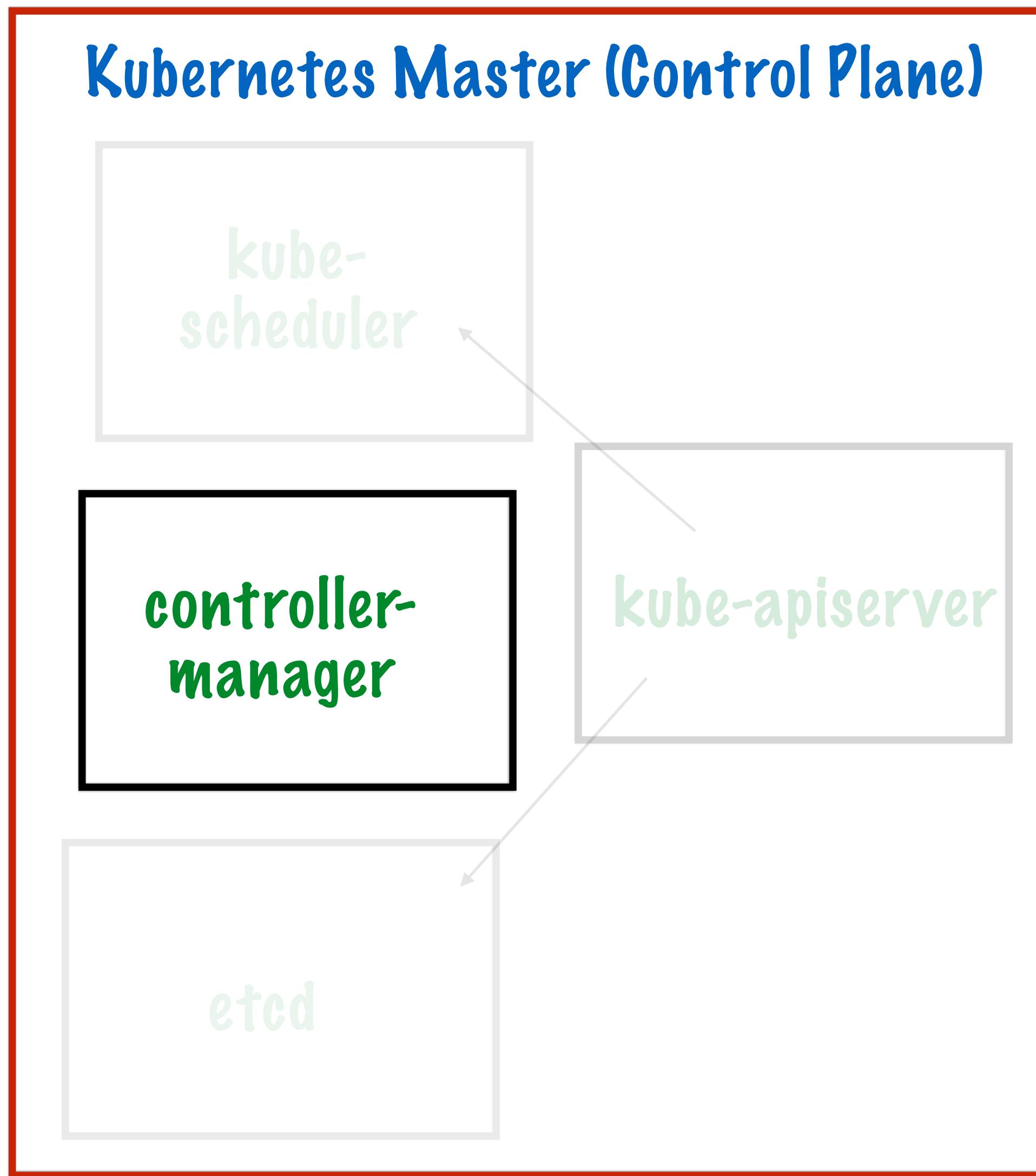
Apiserver
etcd
scheduler
controller-manager
cloud-controller-manager
kube-controller-manager

Control Plane



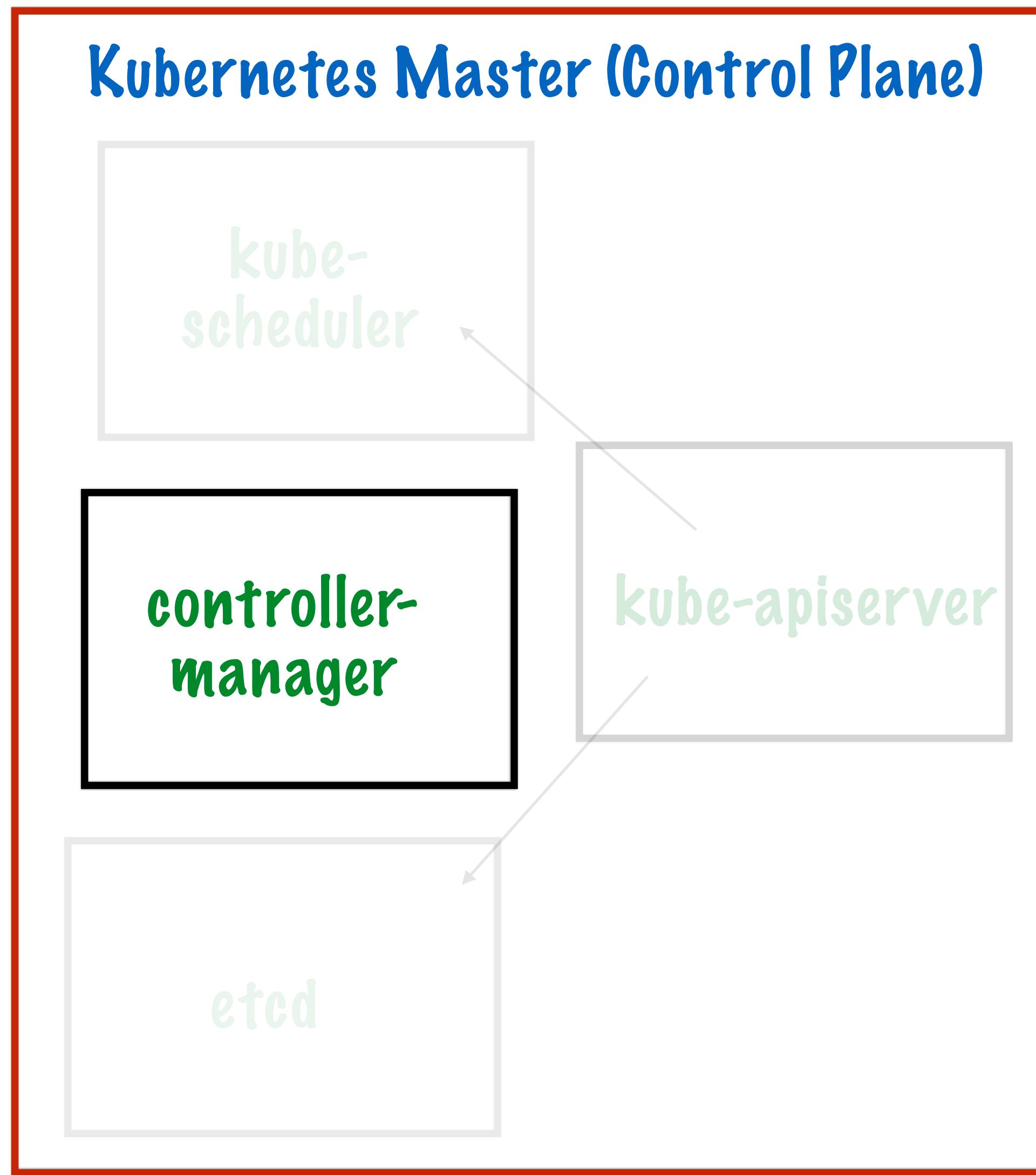
Apiserver
etcd
scheduler
controller-manager
cloud-controller-manager
kube-controller-manager

Control Plane



Apiserver
etcd
scheduler
controller-manager
cloud-controller-manager
kube-controller-manager

Control Plane



Apiserver
etcd
scheduler
controller-manager
cloud-controller-manager
kube-controller-manager

What Runs On Each Node Of A Cluster?

Kubernetes Cluster

Containers running on

Bare metal

VM Instances

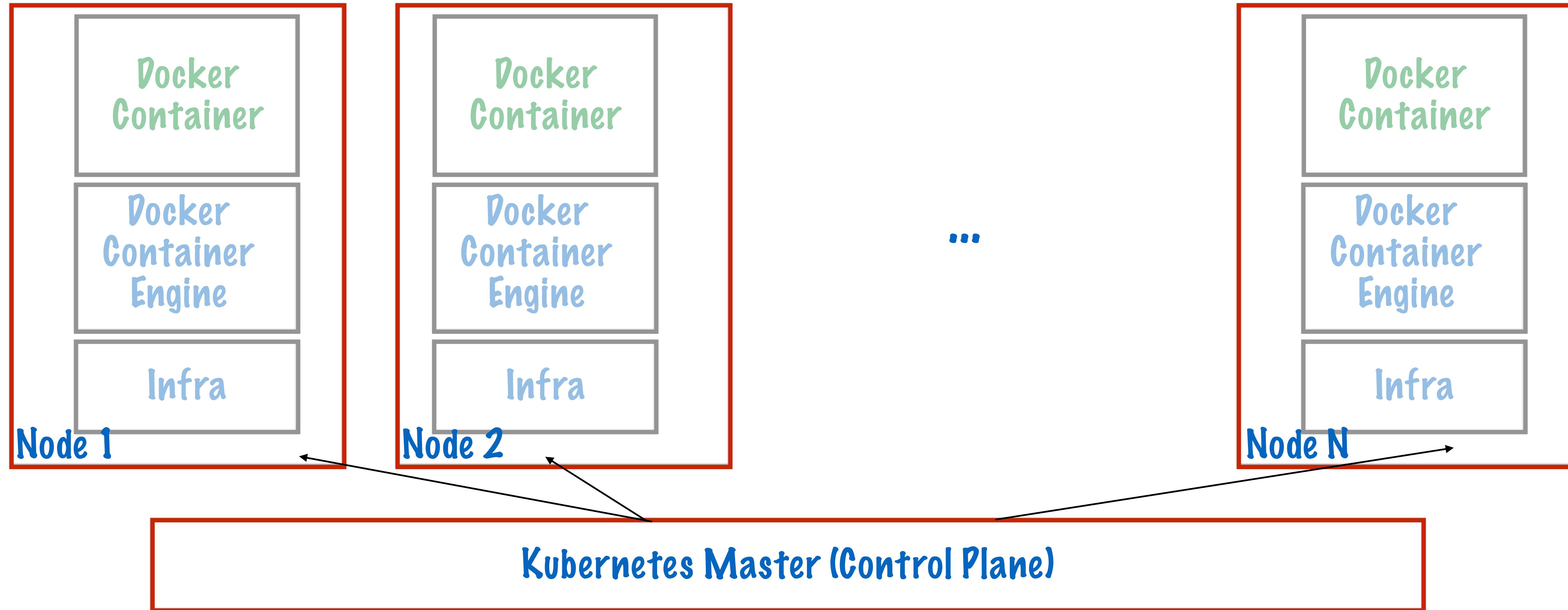
Cloud Instances

Kubernetes - designates master(s) and nodes*

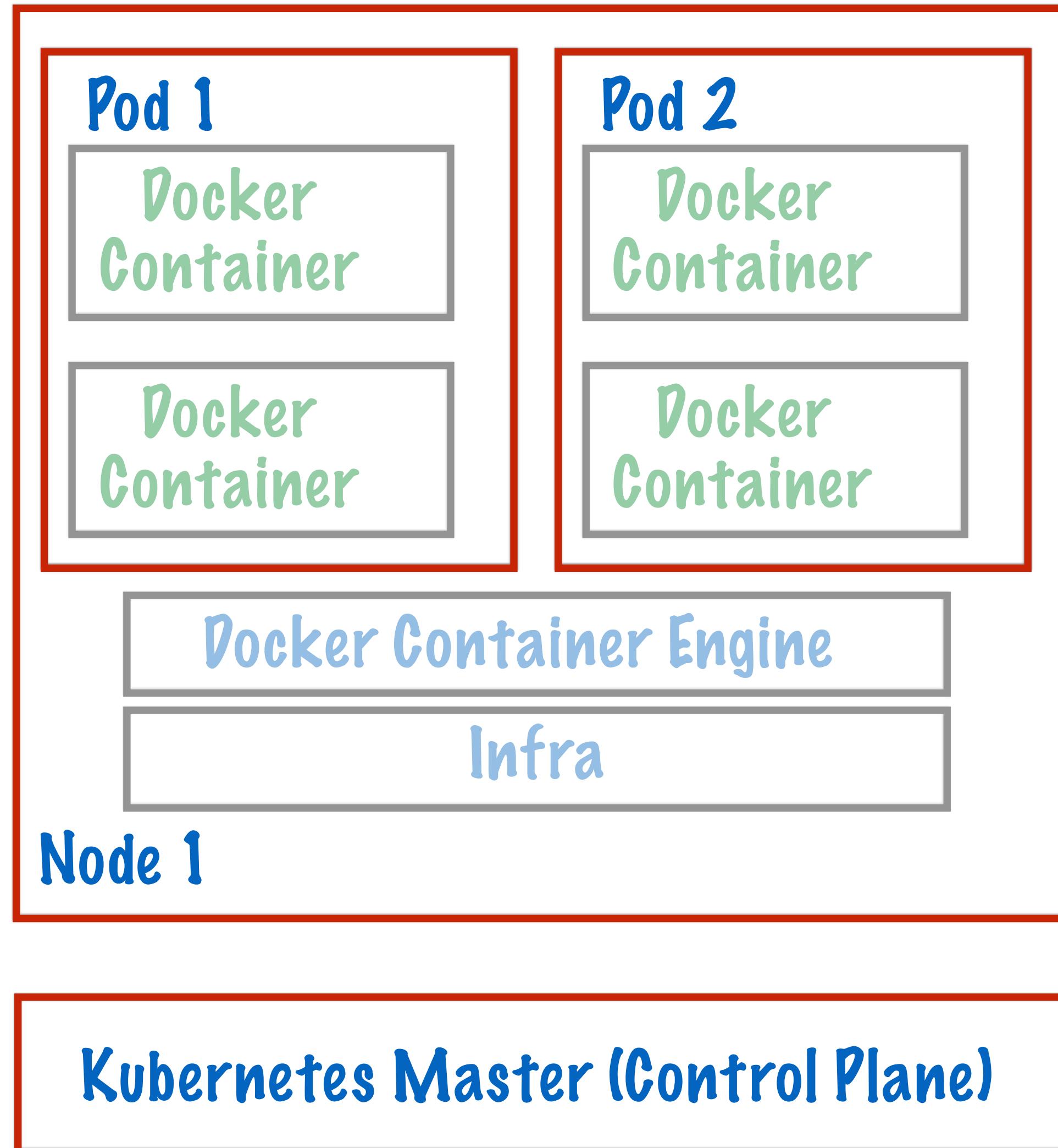
*previously called minions

Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs



Kubernetes Master



One or more nodes designated as master

Several kubernetes processes run on master

Multi-master for high-availability

Kubernetes Master

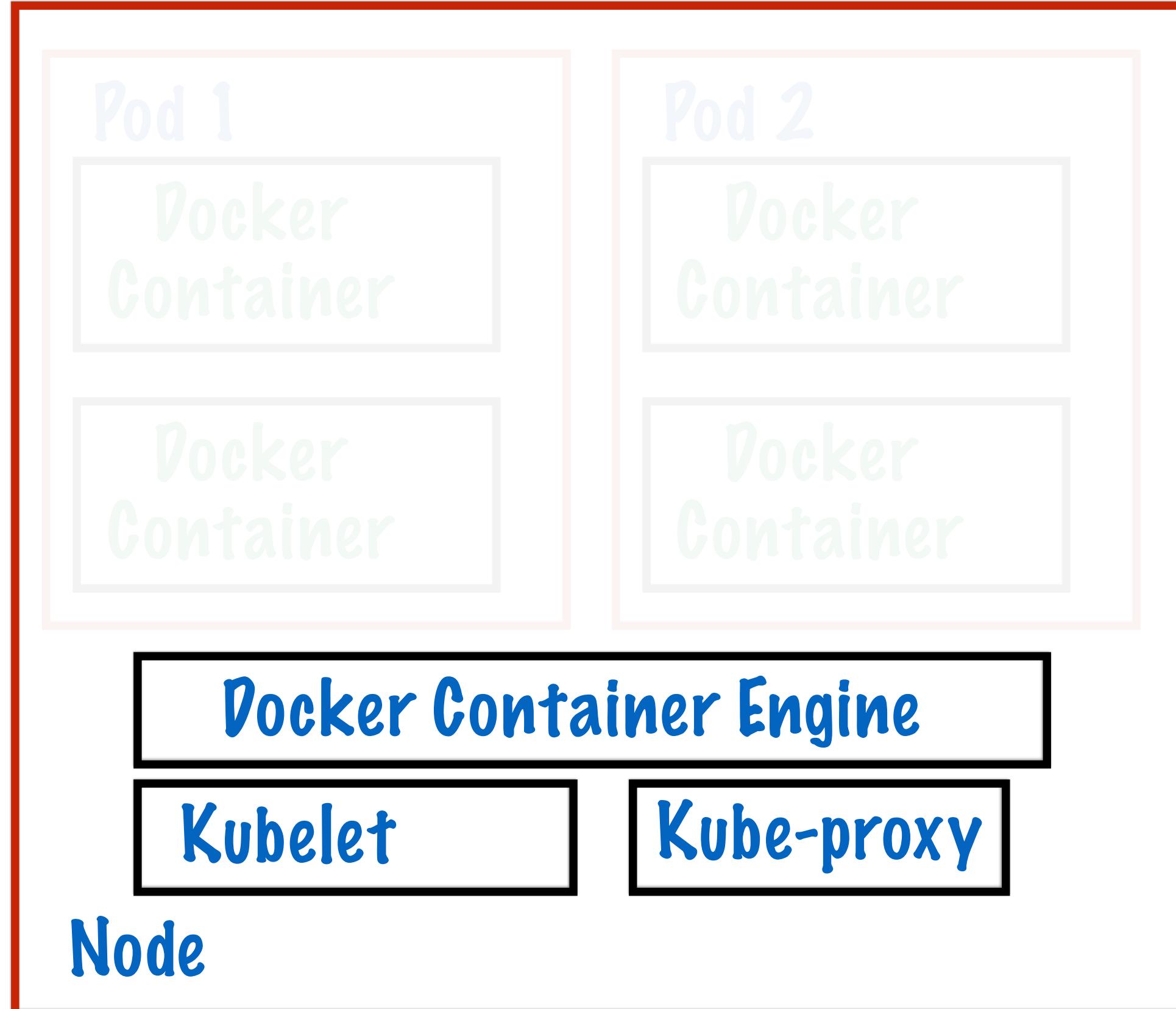


One or more nodes designated as master

Several kubernetes processes run on master

Multi-master for high-availability

Kubernetes Node (Minion)



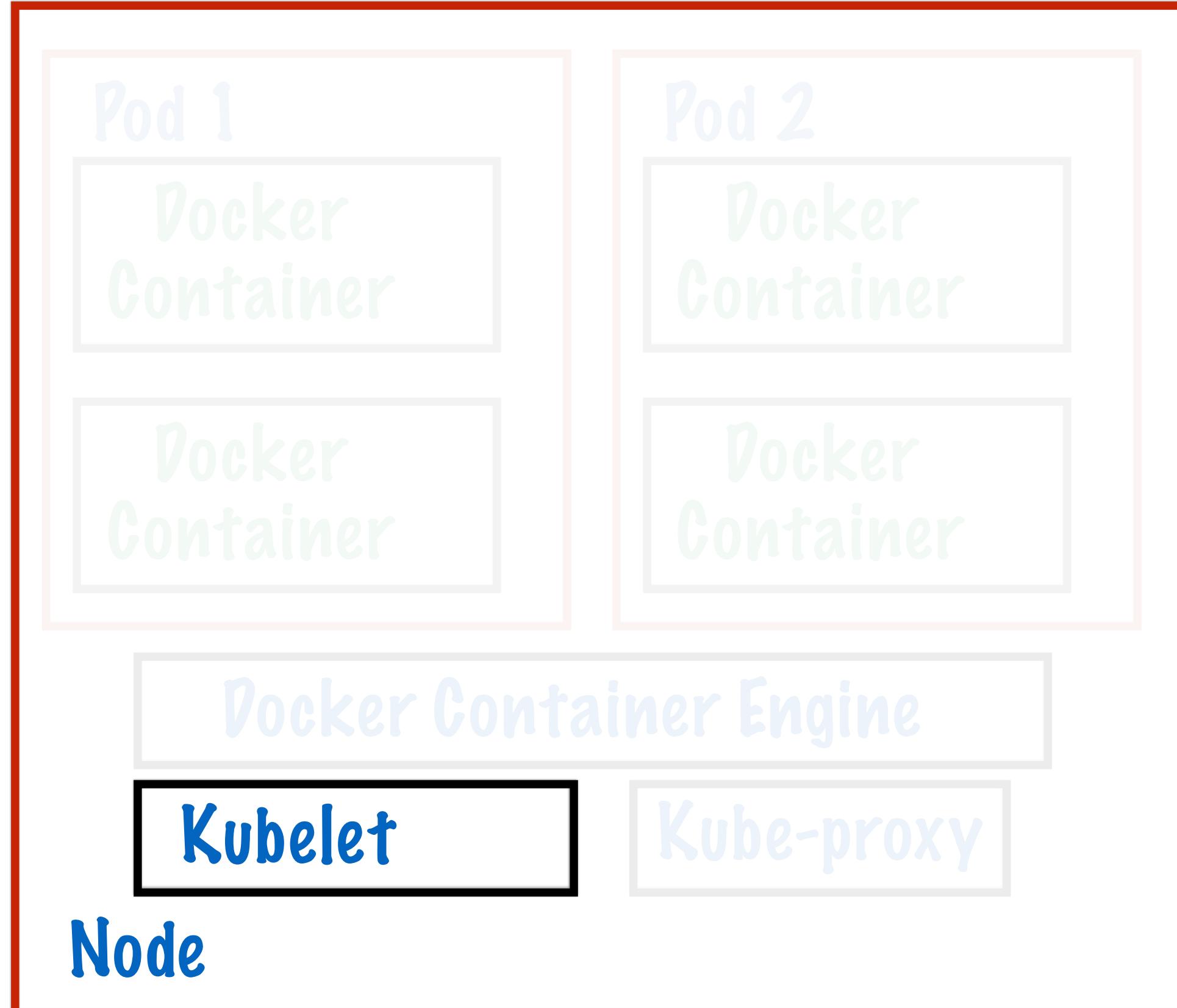
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Kubernetes Node (Minion)



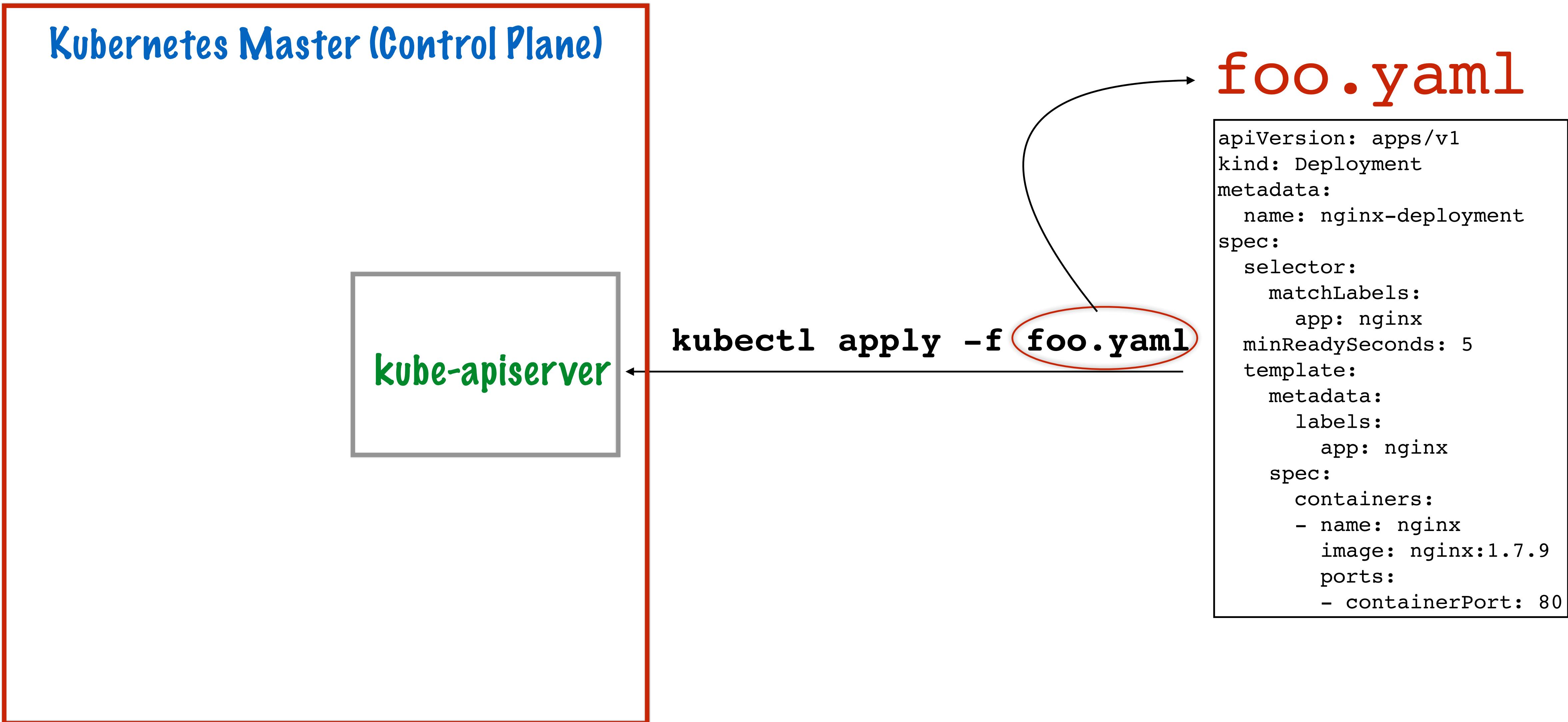
Kubelet

Agent running on this node

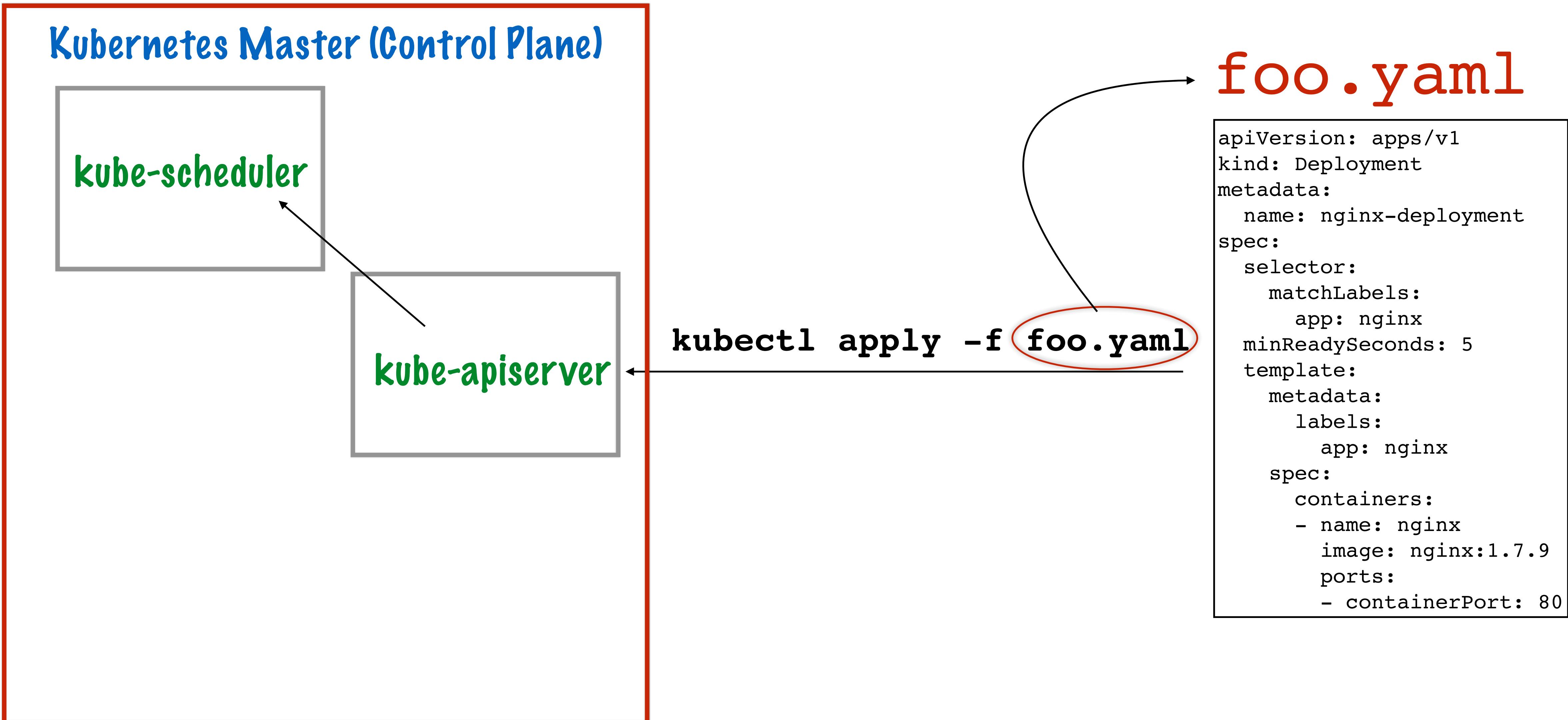
Listens to kubernetes master

Kubernetes Master (Control Plane)

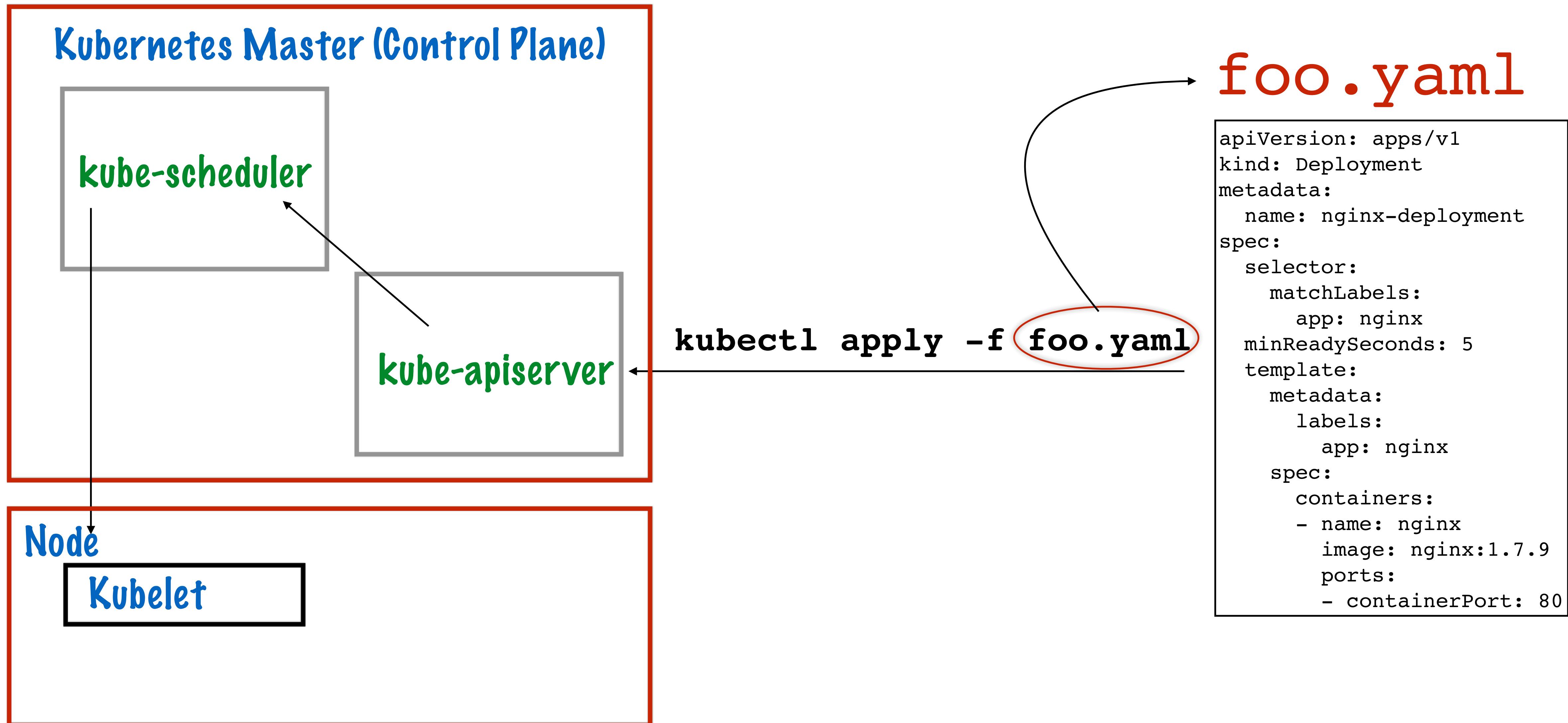
Pod Creation Request



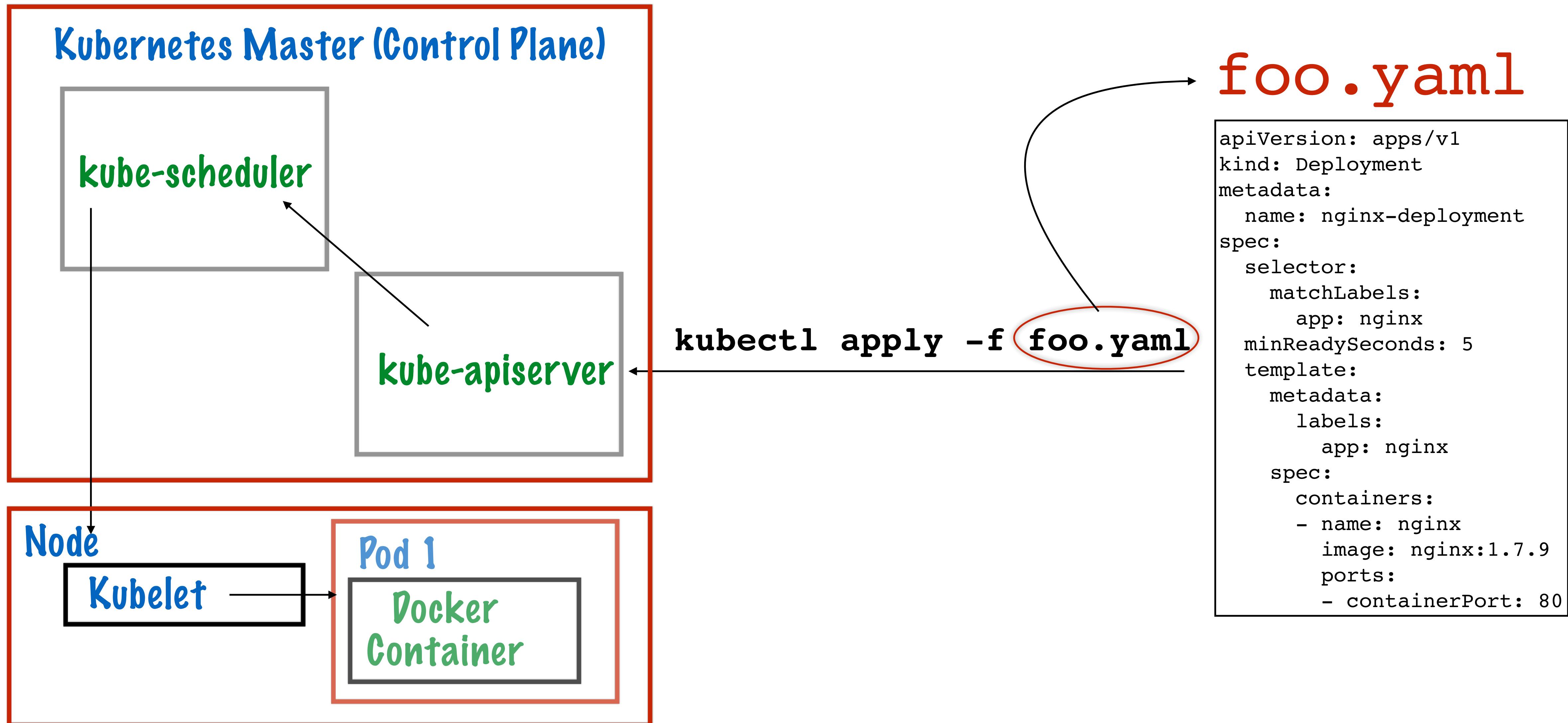
Pod Creation Request



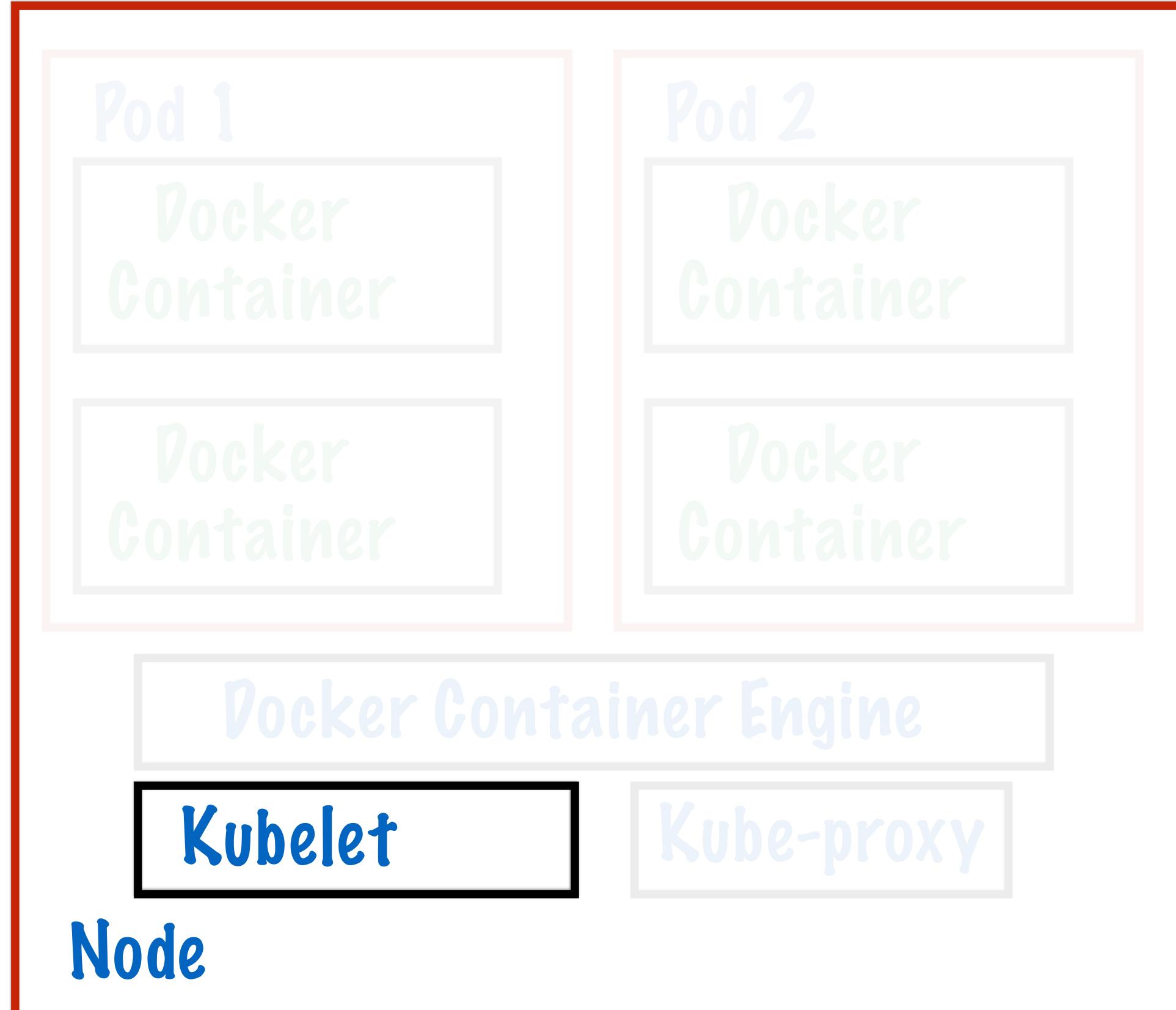
Pod Creation Request



Pod Creation Request



Kubernetes Node (Minion)



Kubernetes Master (Control Plane)

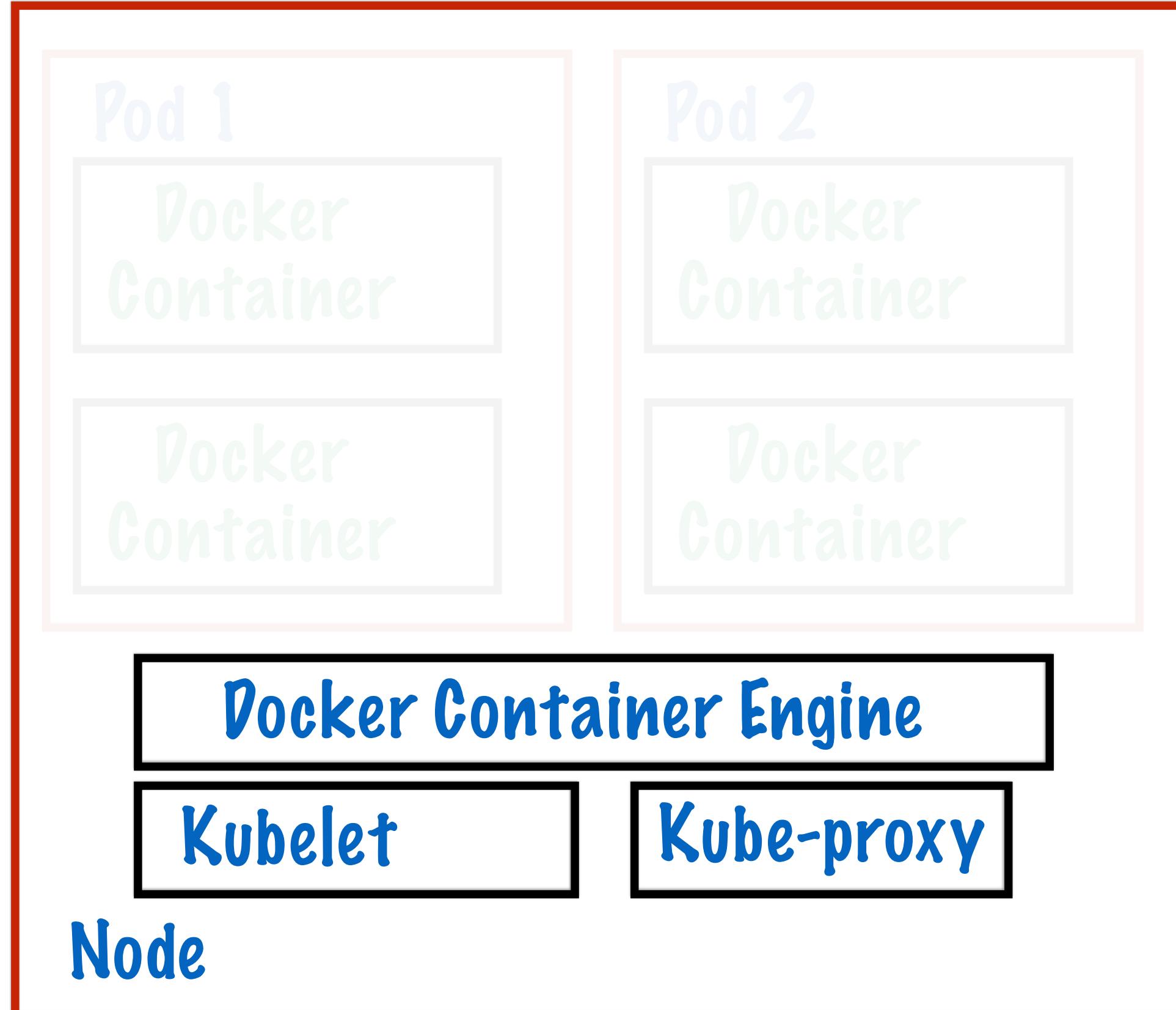
Kubelet

Agent running on this node

Listens to kubernetes master

Port 10255

Kubernetes Node (Minion)



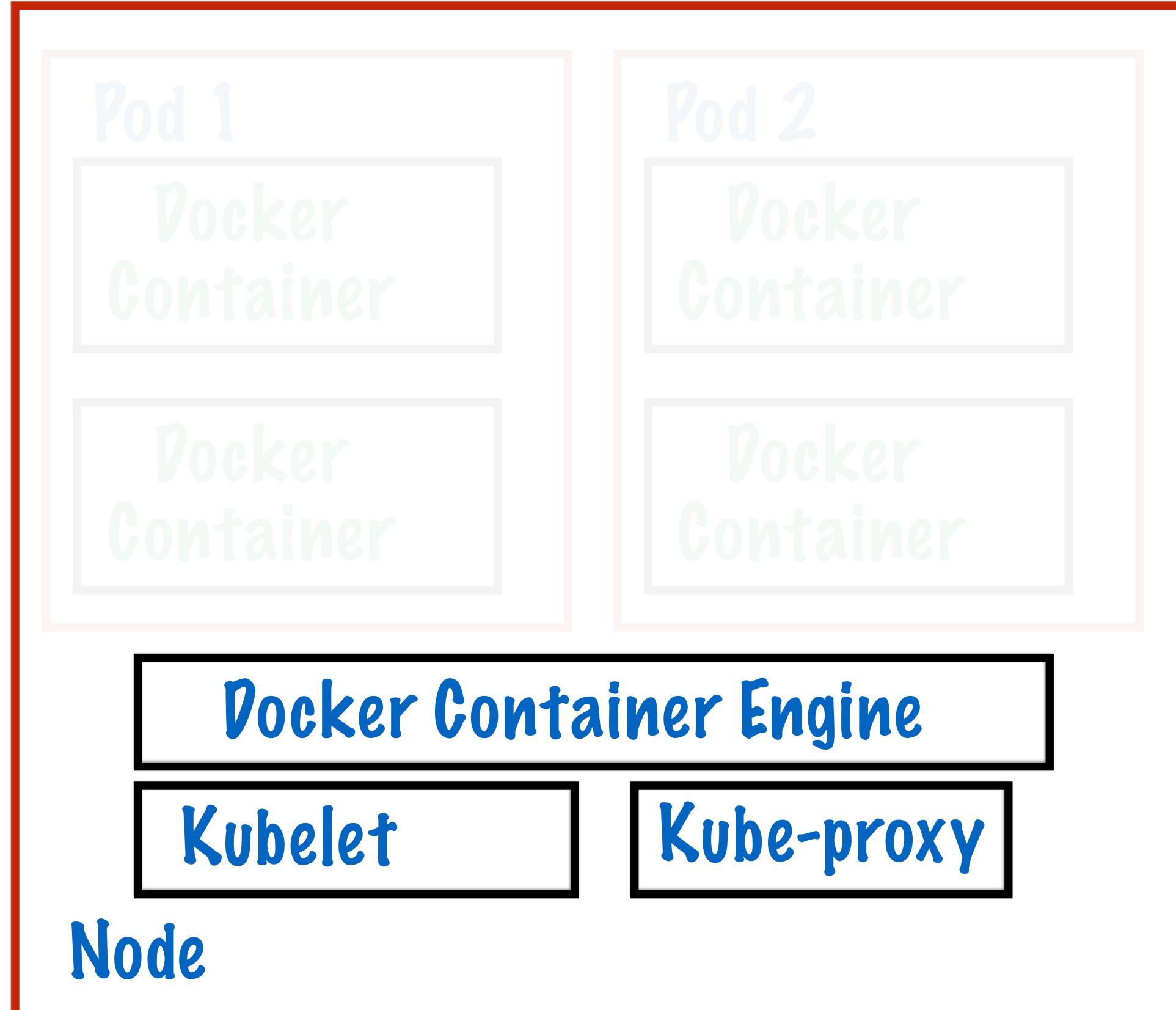
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Kubernetes Node (Minion)



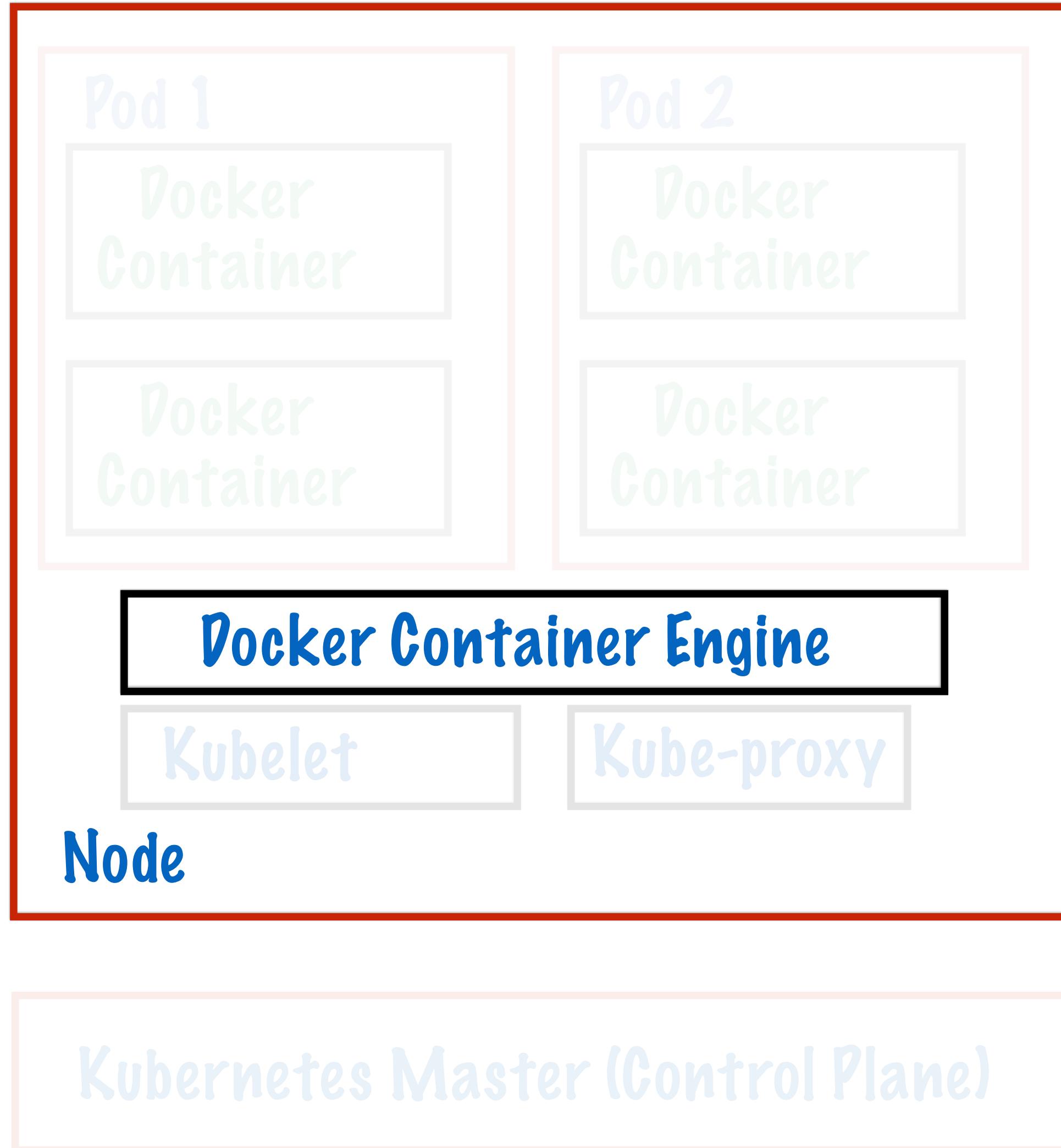
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Kubernetes Node (Minion)



Container engine

Could be Docker or rkt

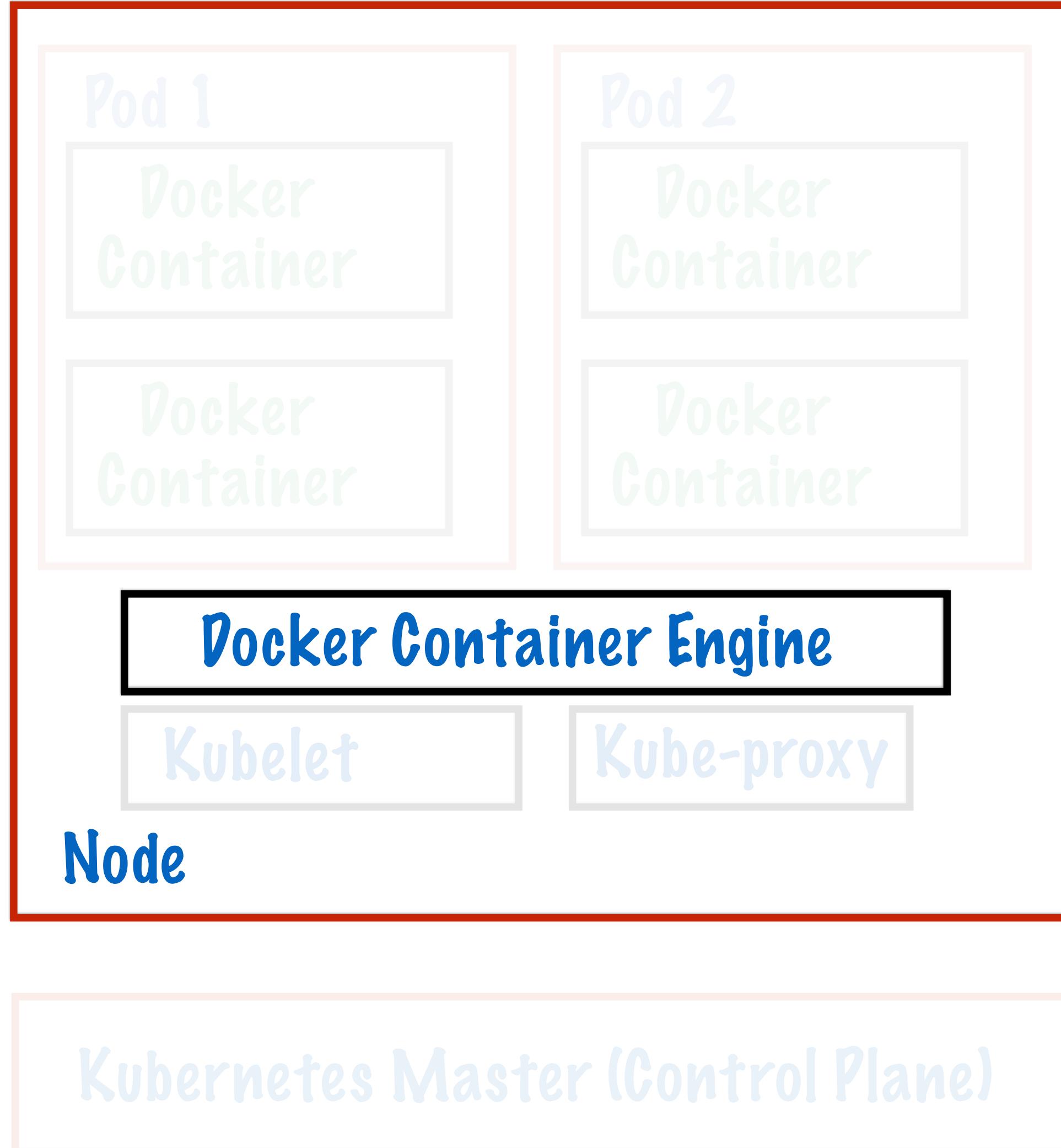
Works with kubelet

Pulling images

Start/stop

containerd

Kubernetes Node (Minion)



CRI

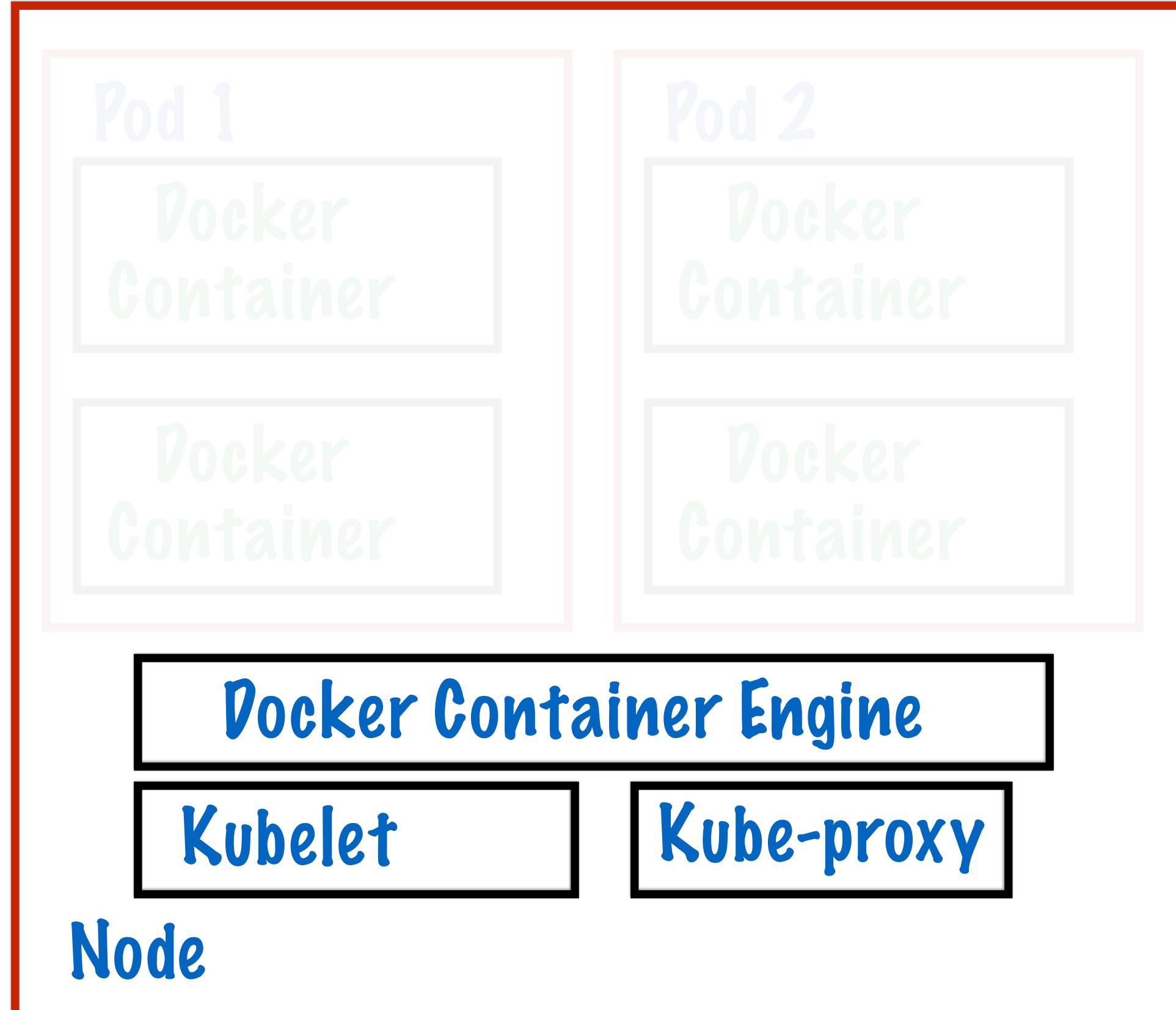
Container Runtime Interface

Tight-coupling between

Kubelet

Container Engine

Kubernetes Node (Minion)



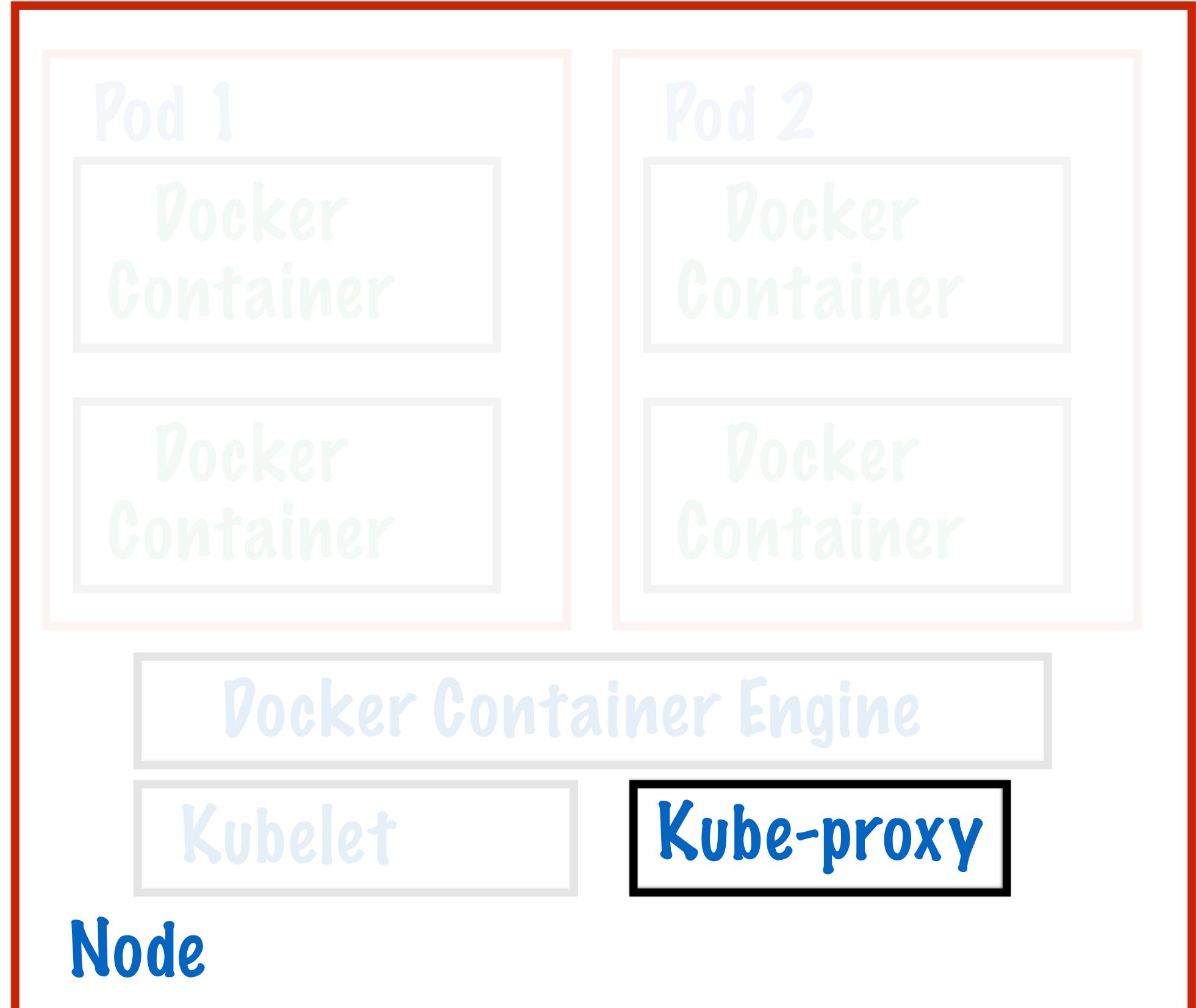
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Kubernetes Node (Minion)



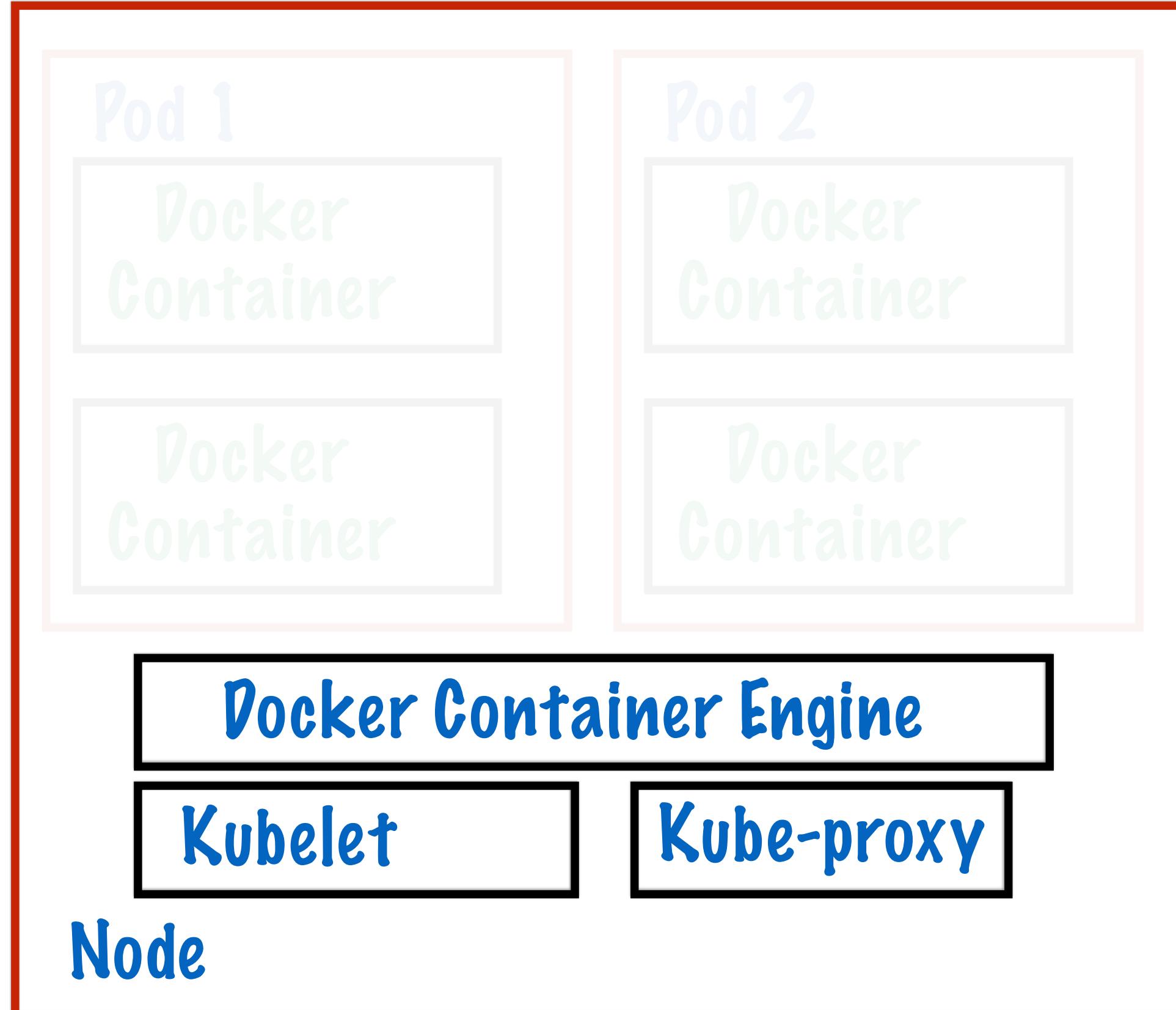
Kube-proxy

Needed because pod IP addresses are ephemeral

Networking - will make sense when we discuss Service objects

Kubernetes Master (Control Plane)

Kubernetes Node (Minion)



Kubernetes Master (Control Plane)

Kubelet

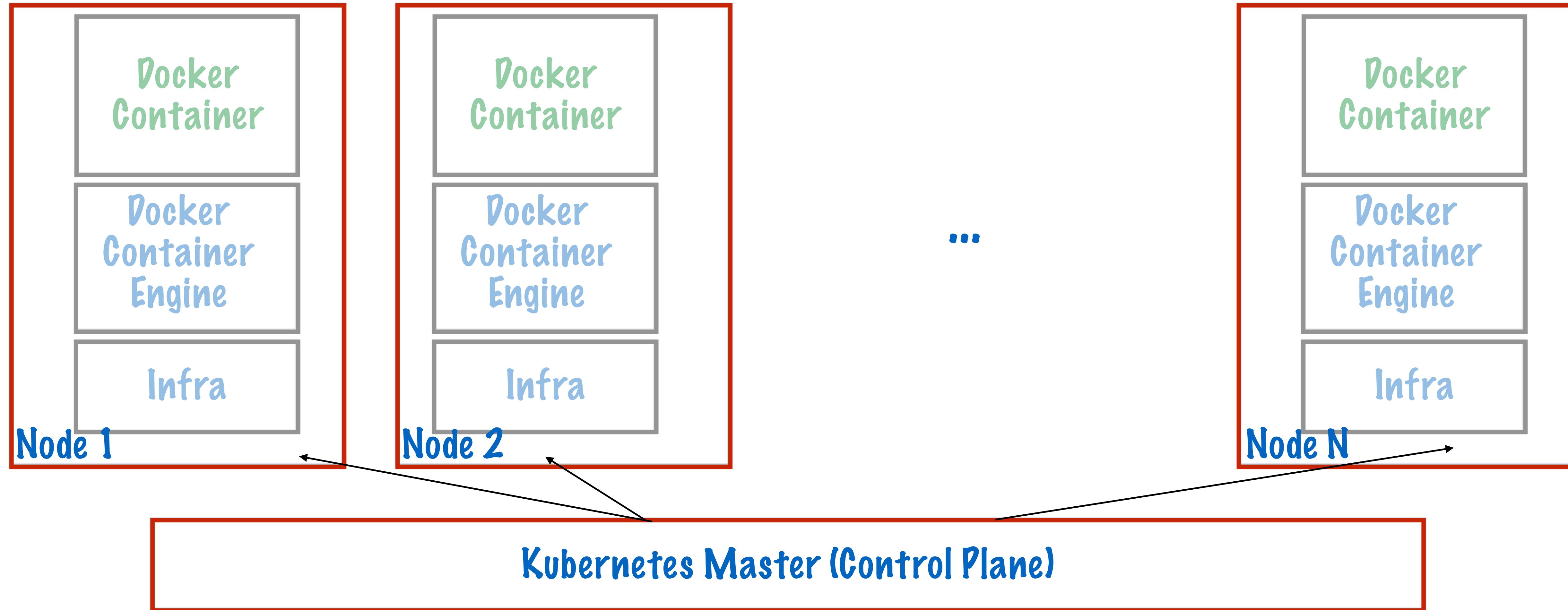
Kube-proxy

Container engine

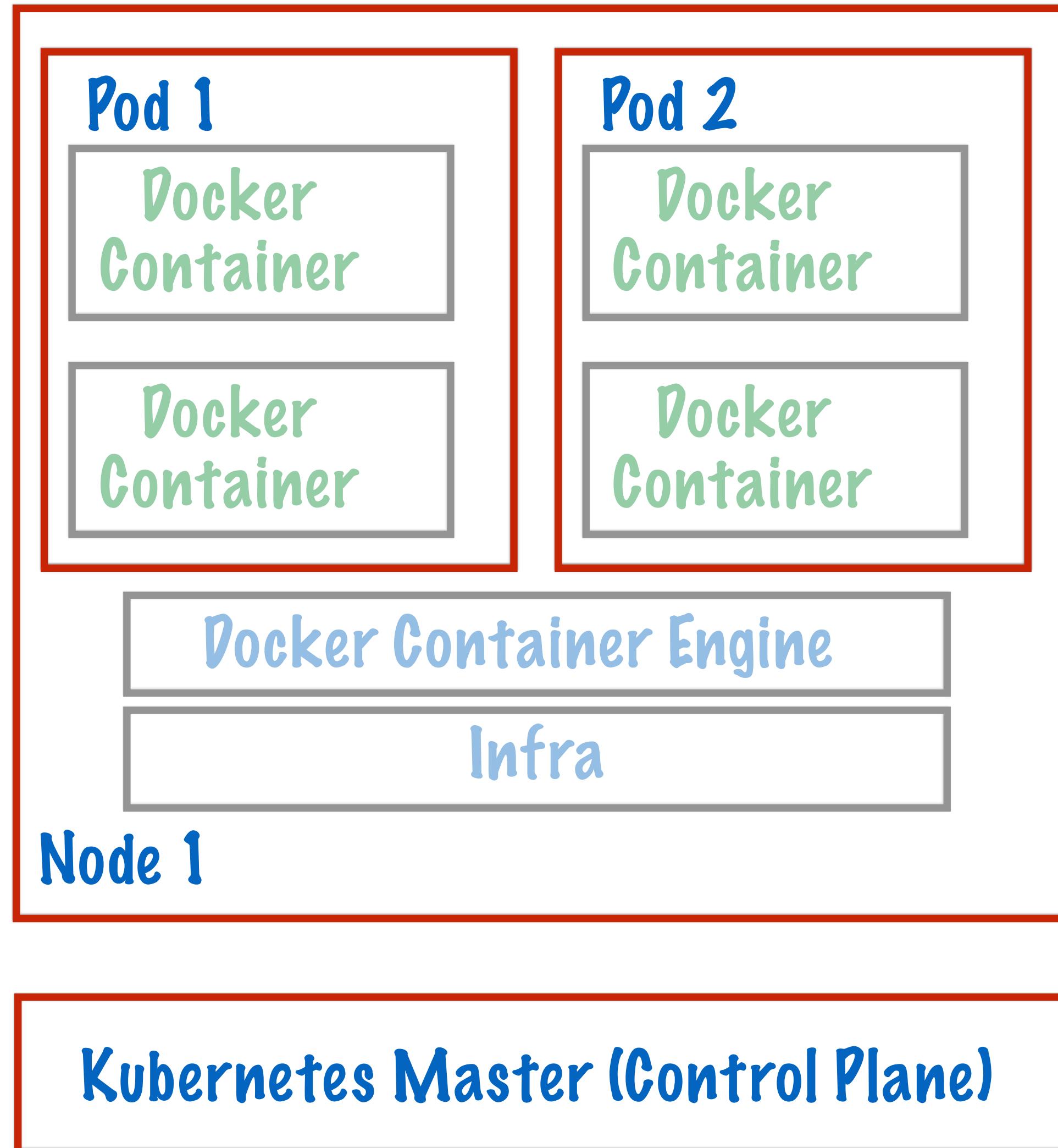
What are Pods?

Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs



Pods on Kubernetes Nodes

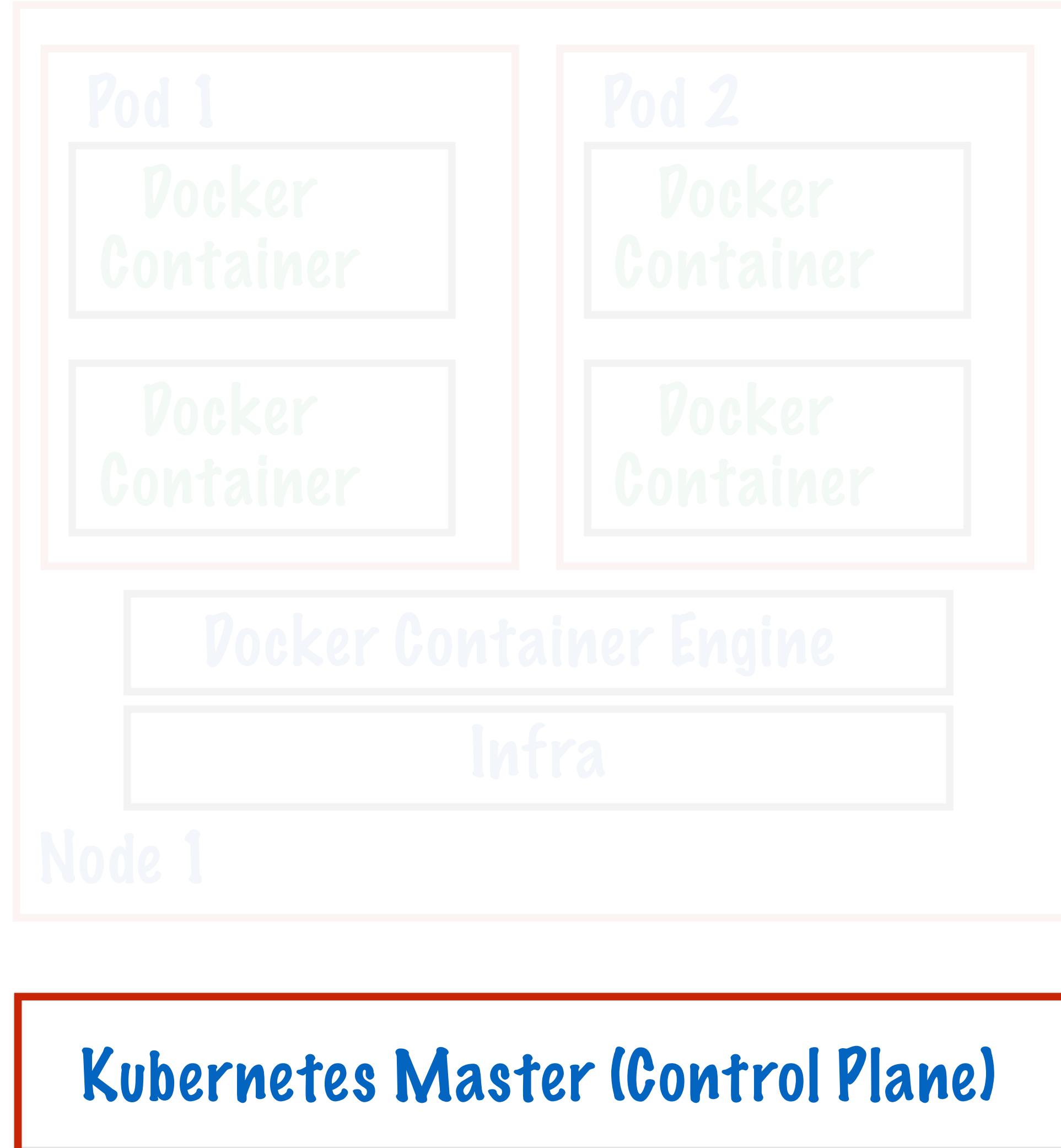


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes

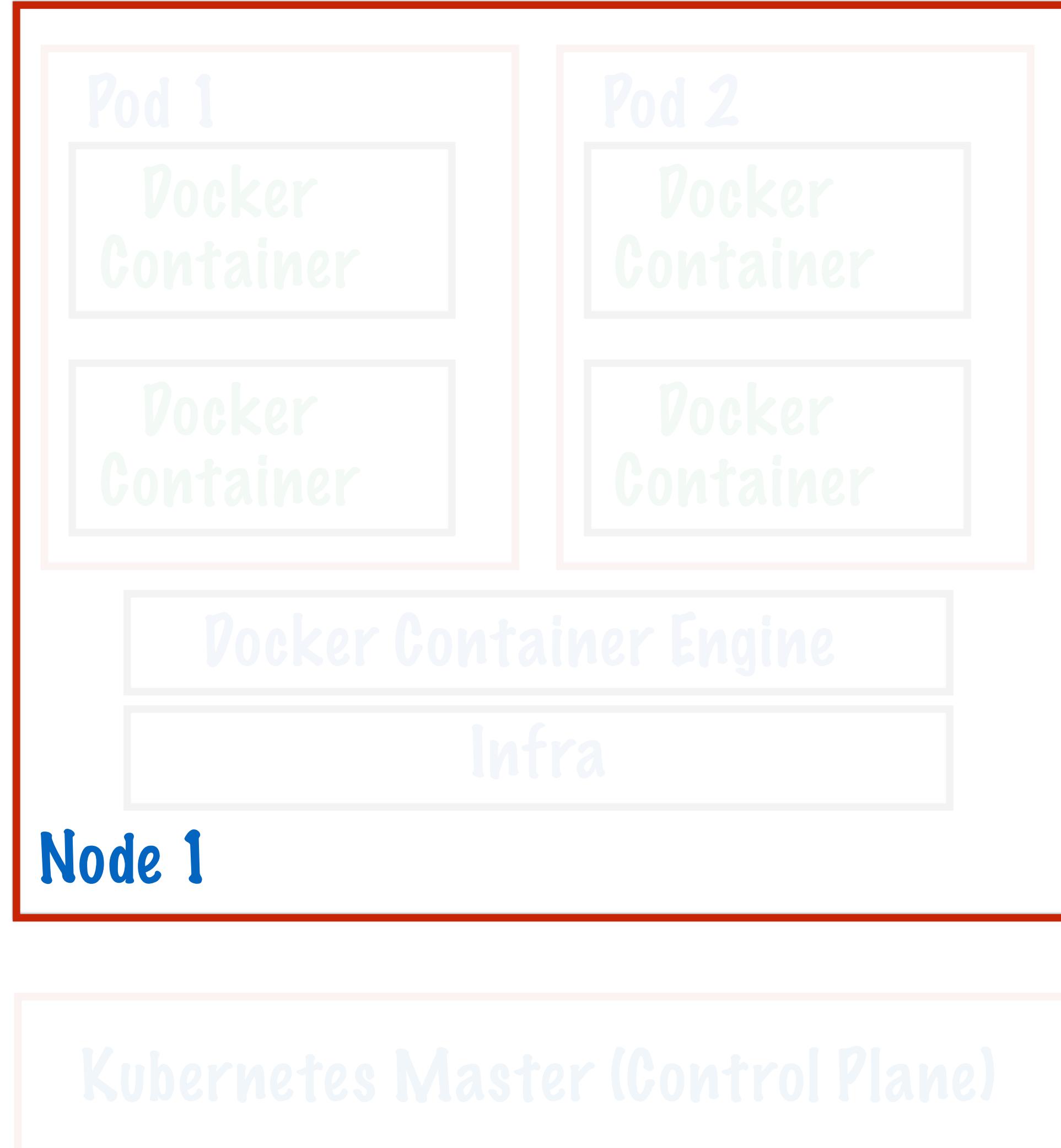


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes



Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes

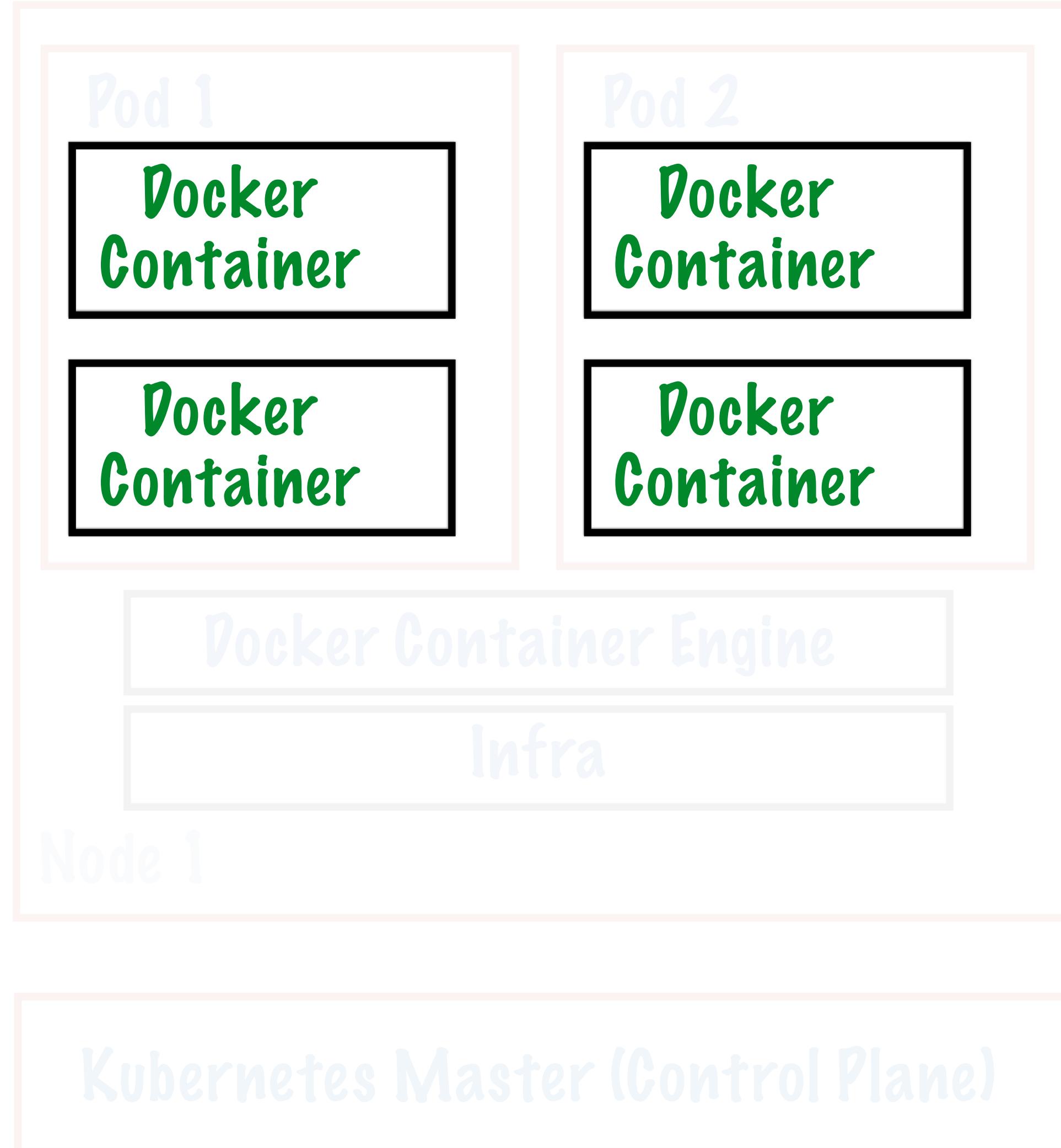


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes



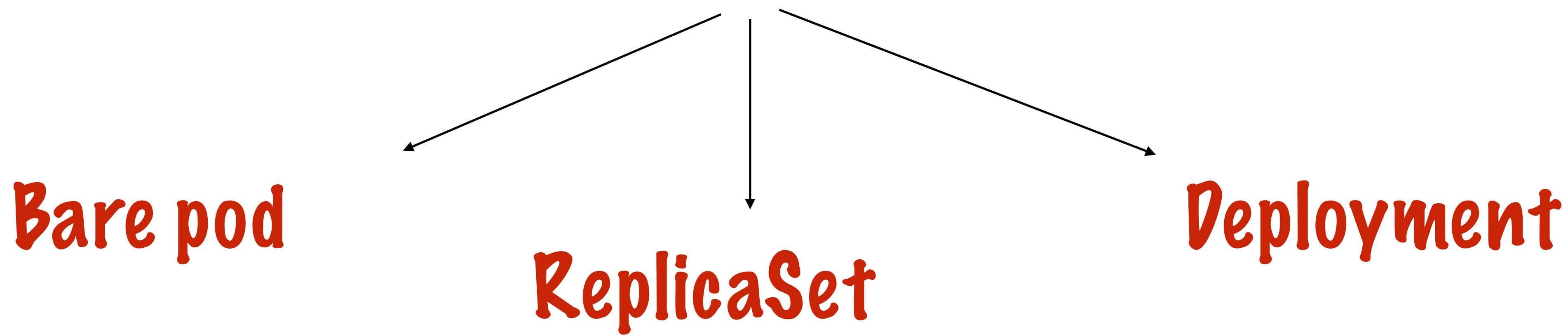
Kubernetes only knows about pods

Can not start container without a pod

Pod => Sandbox for 1 or more containers

Pods on Kubernetes

Pod Creation Requests



Pods are usually not created directly

Higher level abstractions preferred

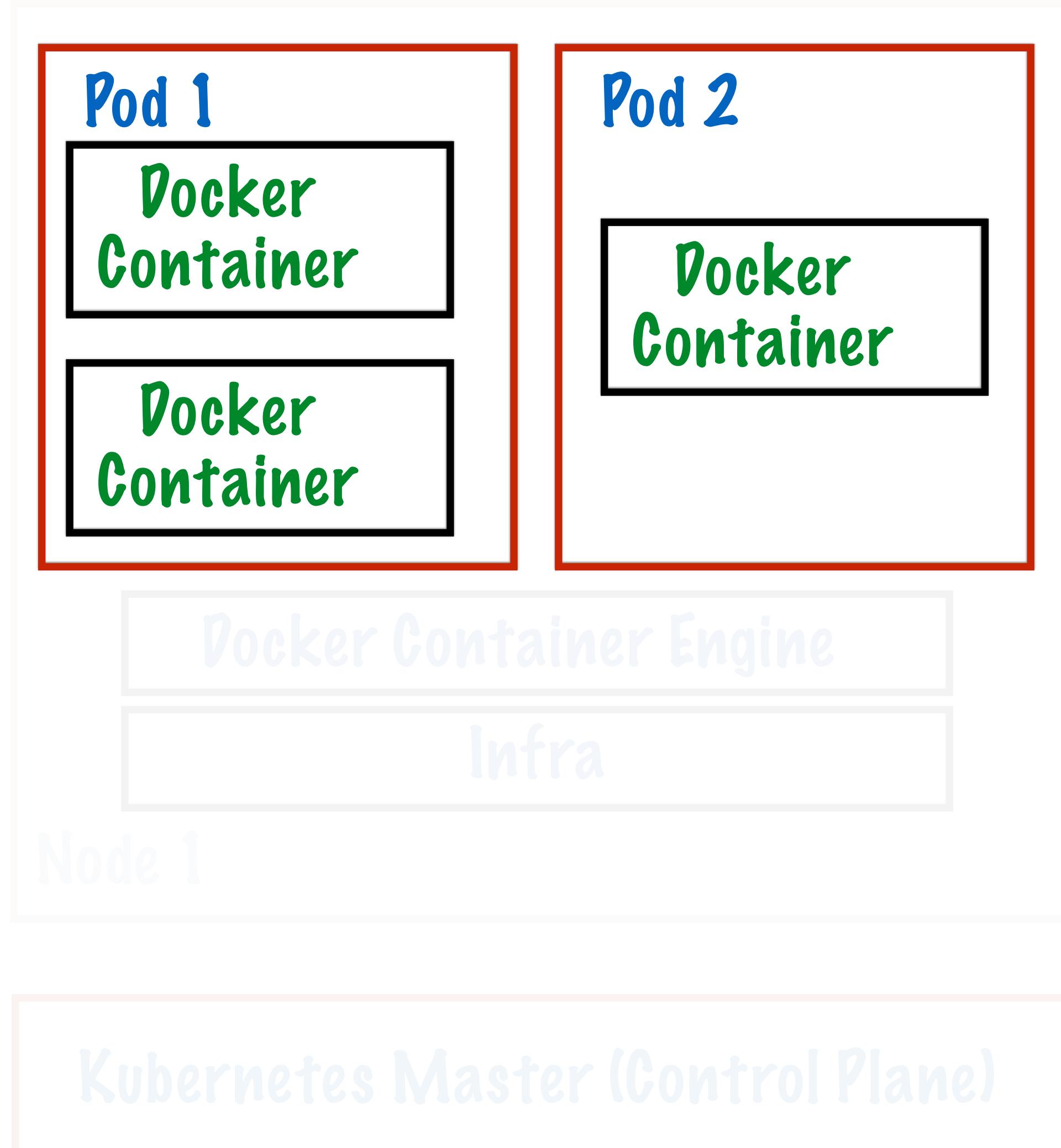
Pod Creation YAML

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
  livenessProbe:
    httpGet:
      # when "host" is not defined, "PodIP" will be used
      # host: my-host
      # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
      # scheme: HTTPS
      path: /healthz
      port: 8080
      httpHeaders:
      - name: X-Custom-Header
        value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

Which container imag(s)?

Available on which port?

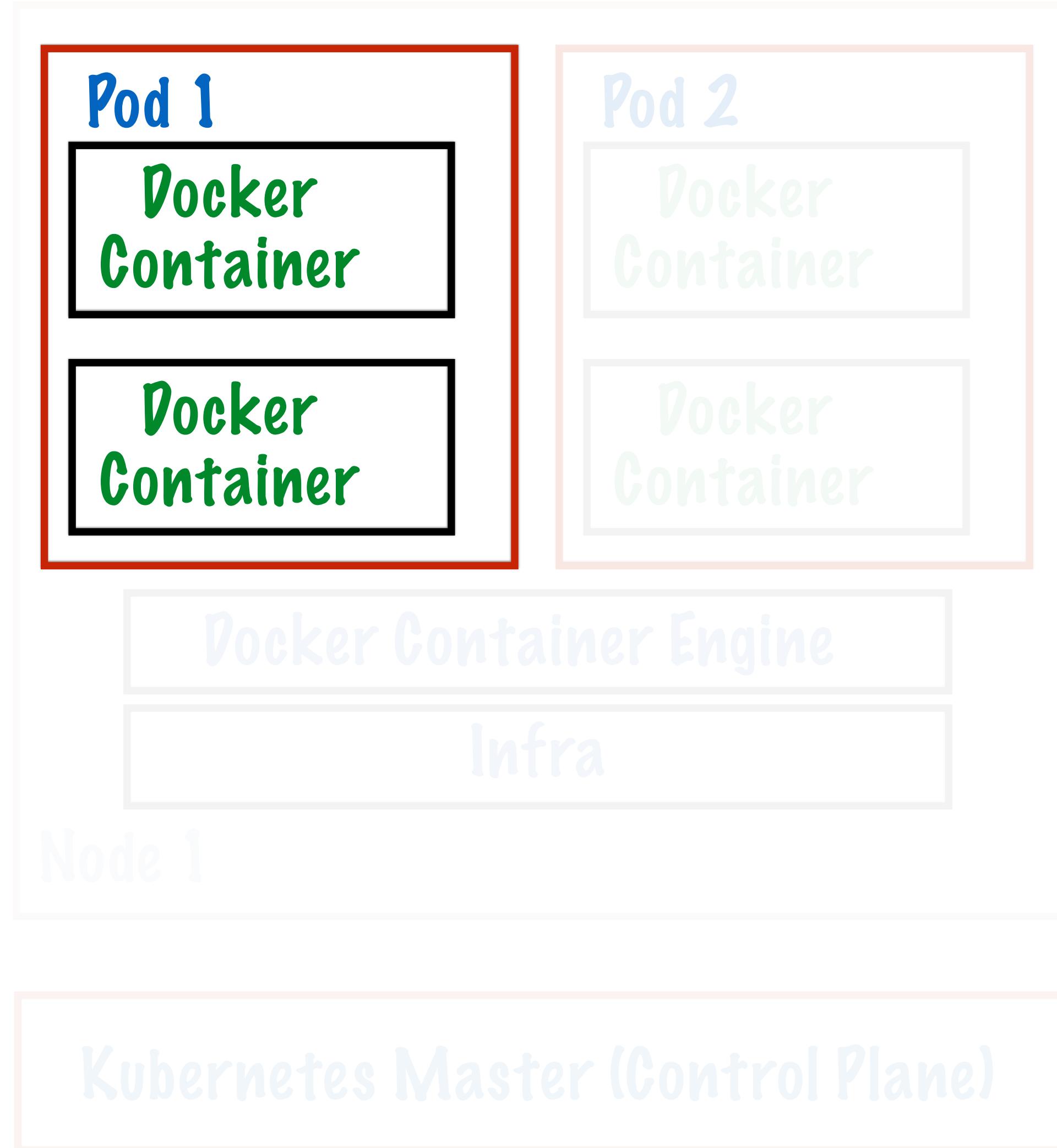
Single and Multi-Container Pods



Pod 1 is a multi-container pod

Pod 2 is a single container pod

Multi-Container Pods

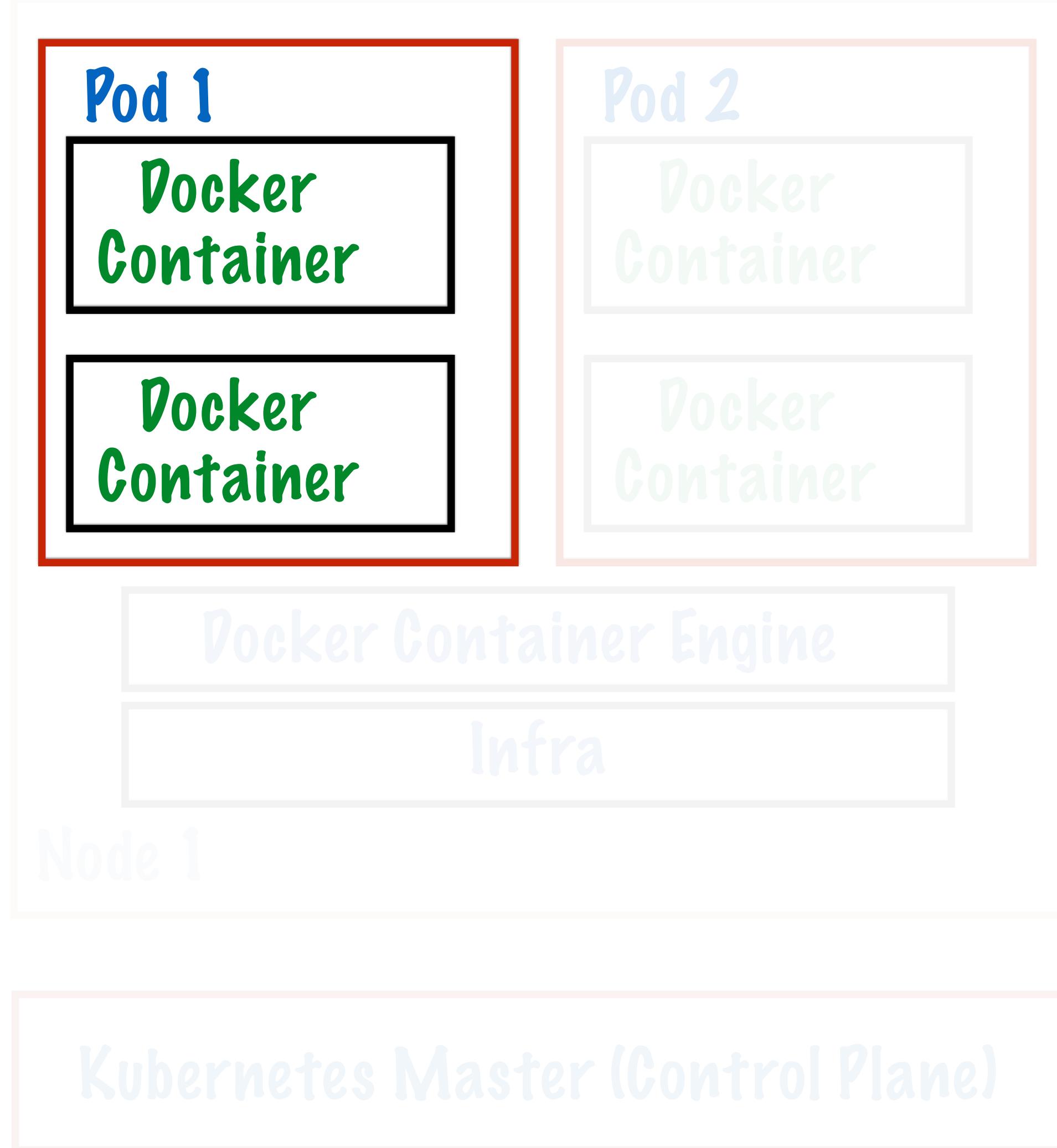


1 pod usually contains 1 container

Multi-container pods are possible too

Such containers are tightly coupled

Multi-Container Pods



Share access to memory space

Connect to each other using localhost

Share access to the same volumes (storage abstraction)

Pods on Kubernetes

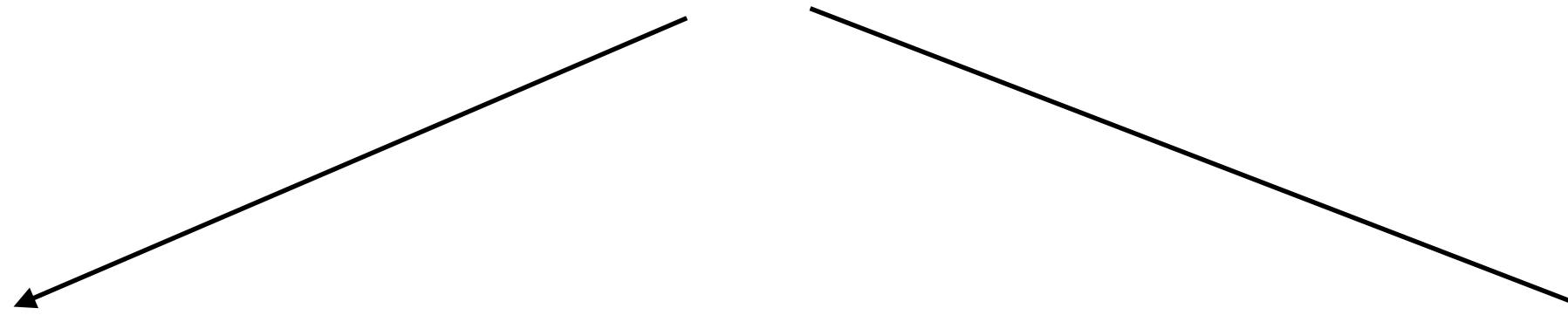
Atomic Unit on Kubernetes

Container Deployment

Containers within pod
are deployed in an all-
or-nothing manner

Node Association

Entire pod is hosted
on the same node



Pods Limitations

- No auto-healing or scaling
- Pod crashes? Must be handled at higher level
 - ReplicaSet, Deployment, Service
- Ephemeral: IP addresses are ephemeral

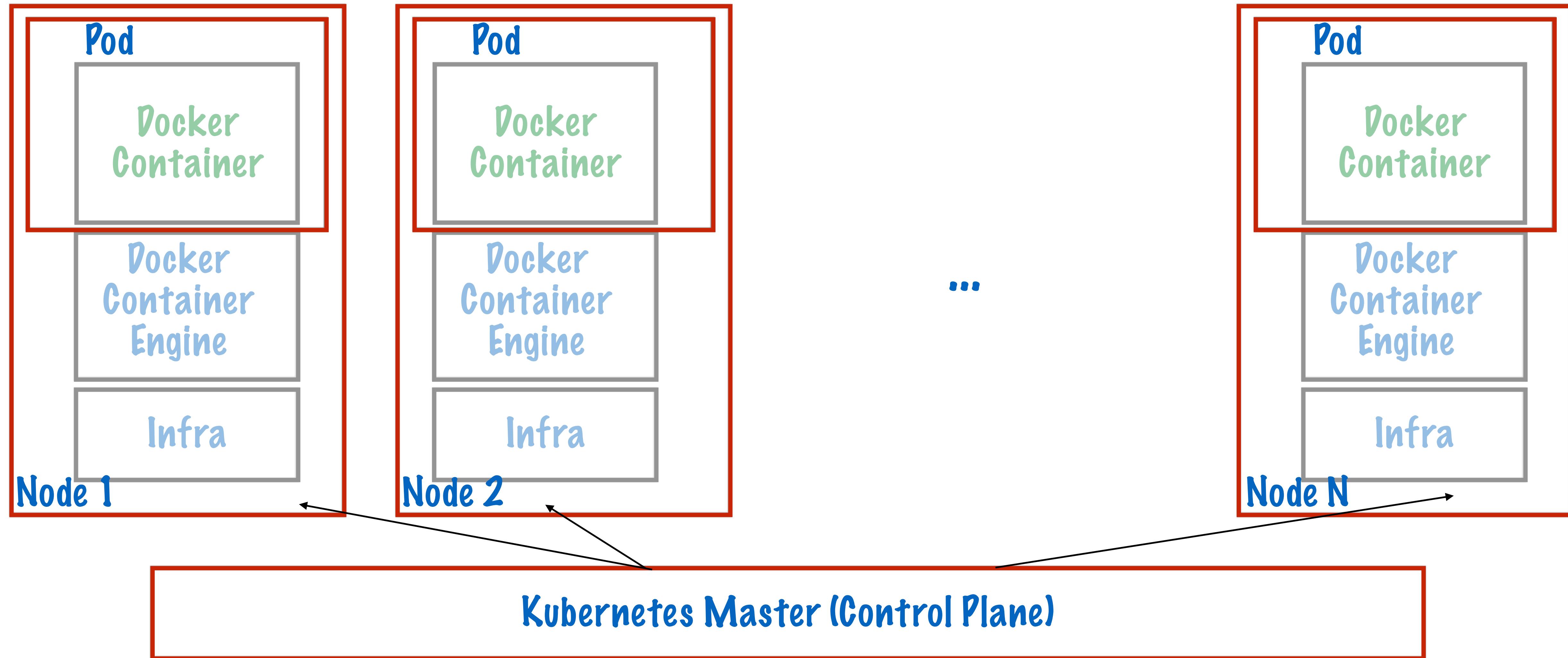
Higher Level Kubernetes Objects

- **ReplicaSet, ReplicationController:** Scaling and healing
- **Deployment:** Versioning and rollback
- **Service:** Static (non-ephemeral) IP and networking
- **Volume:** Non-ephemeral storage

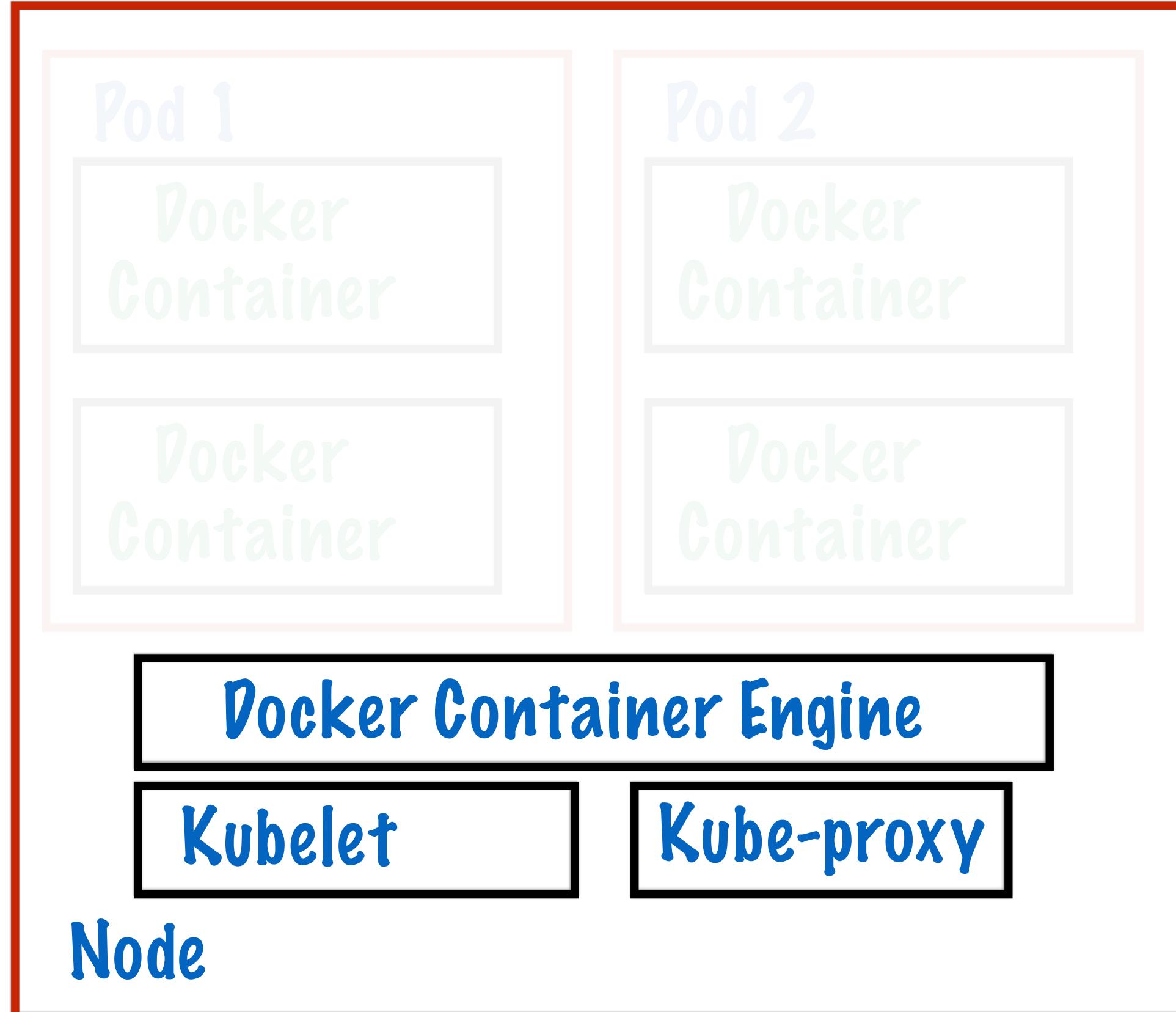
Where Do Pods Run?

Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs



Kubernetes Node (Minion)



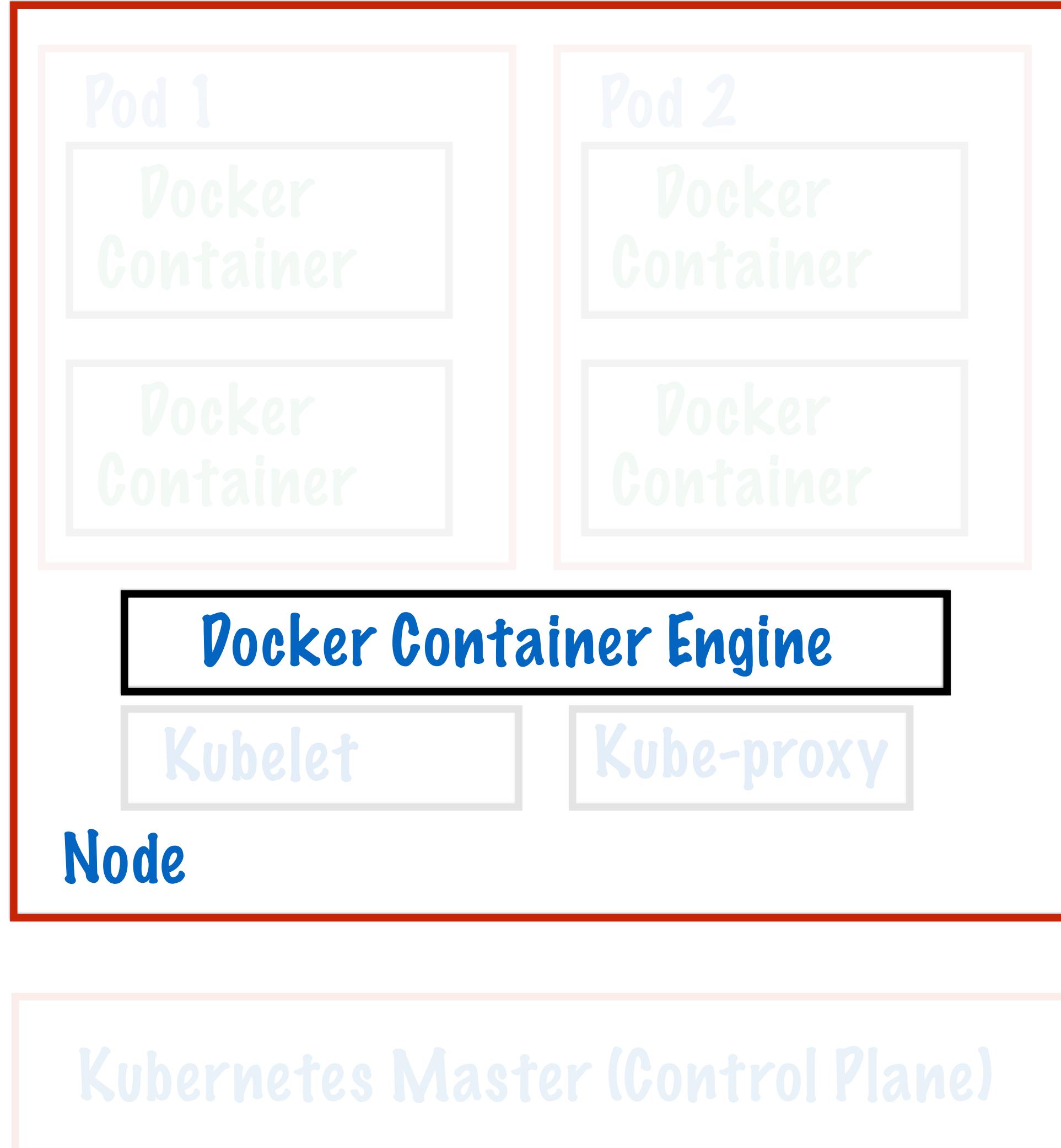
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Kubernetes Node (Minion)



CRI

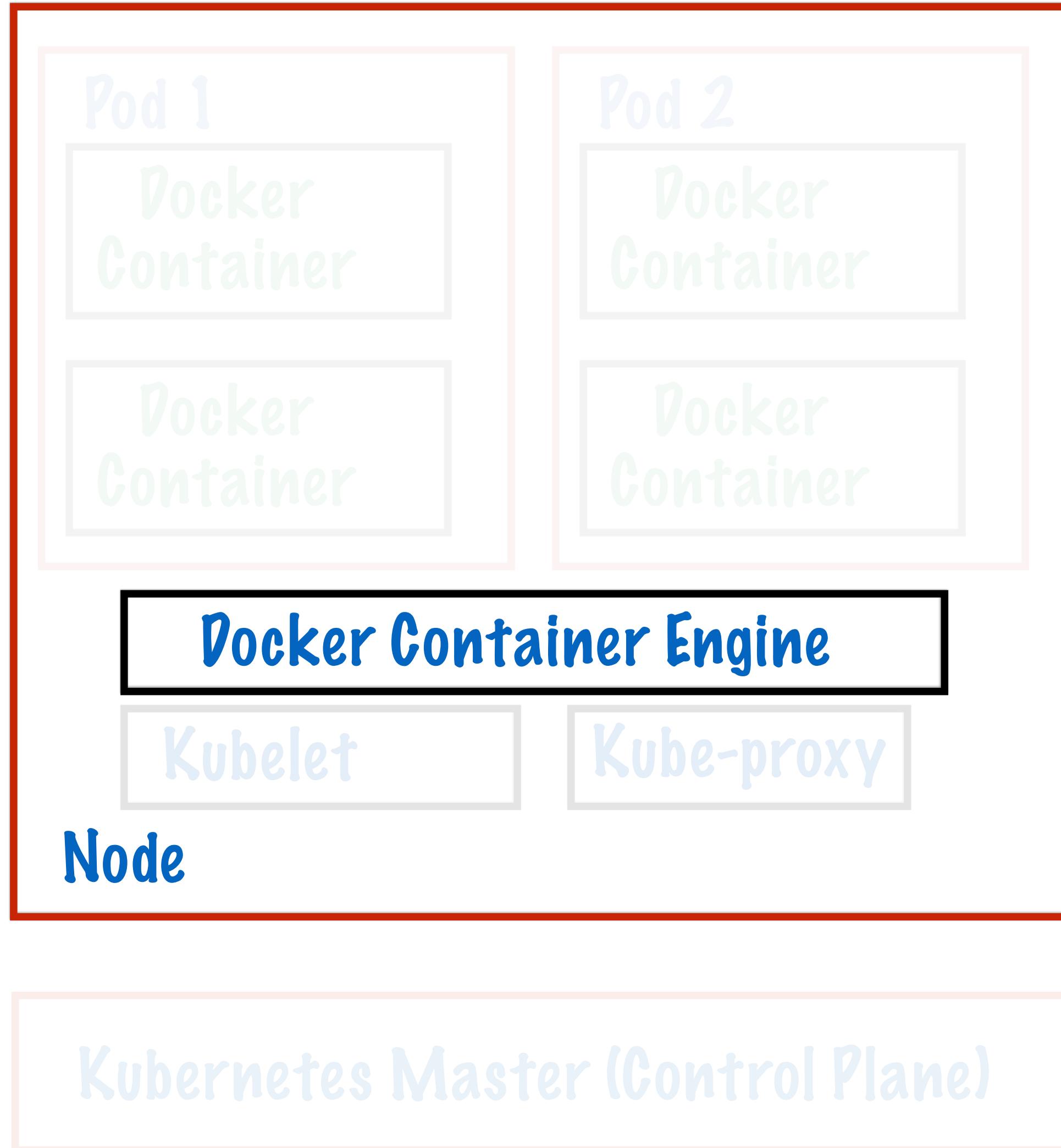
Container Runtime Interface

Tight-coupling between

Kubelet

Container Engine

Kubernetes Node (Minion)



Container engine

Could be Docker or rkt

Works with kubelet

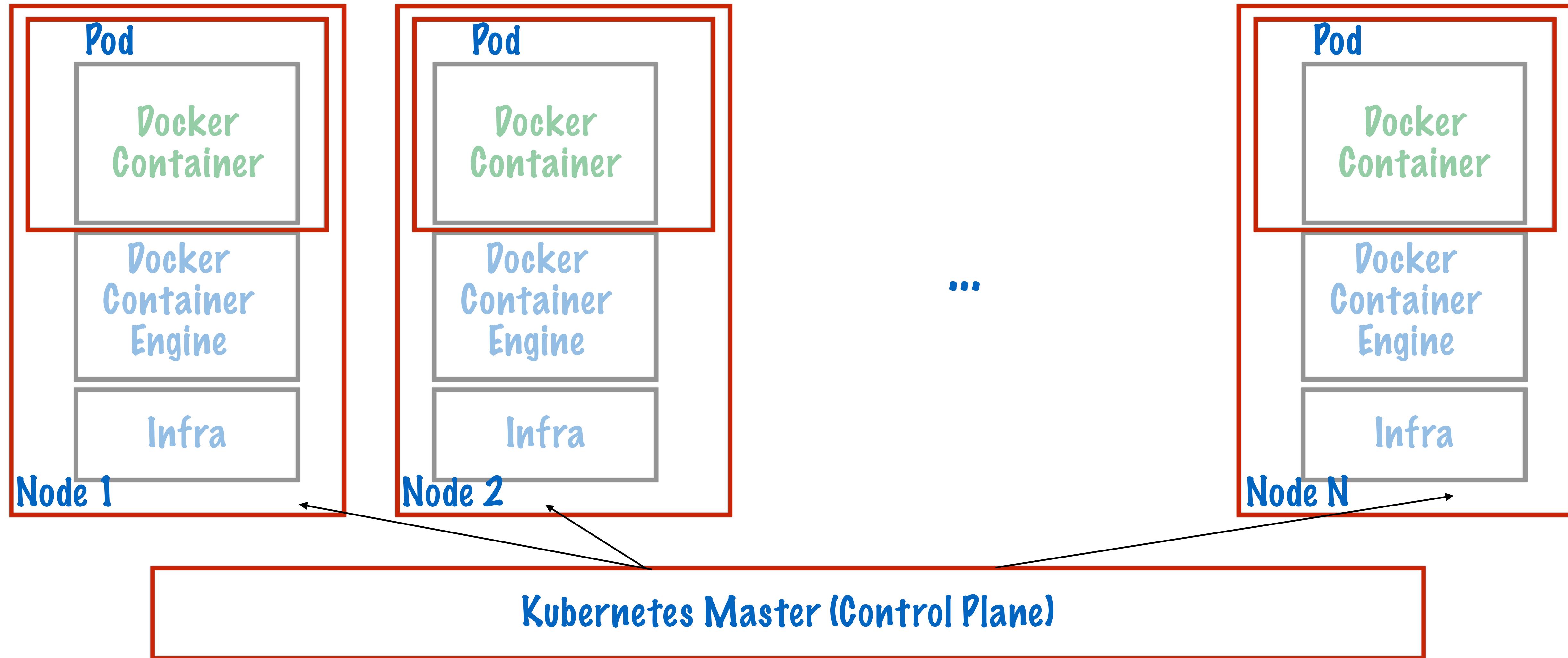
Pulling images

Start/stop

containerd

Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs



Kubernetes Cluster

Containers running on

Bare metal

VM Instances

Cloud Instances

Kubernetes - designates master(s) and nodes*

*previously called minions

Pods Limitations

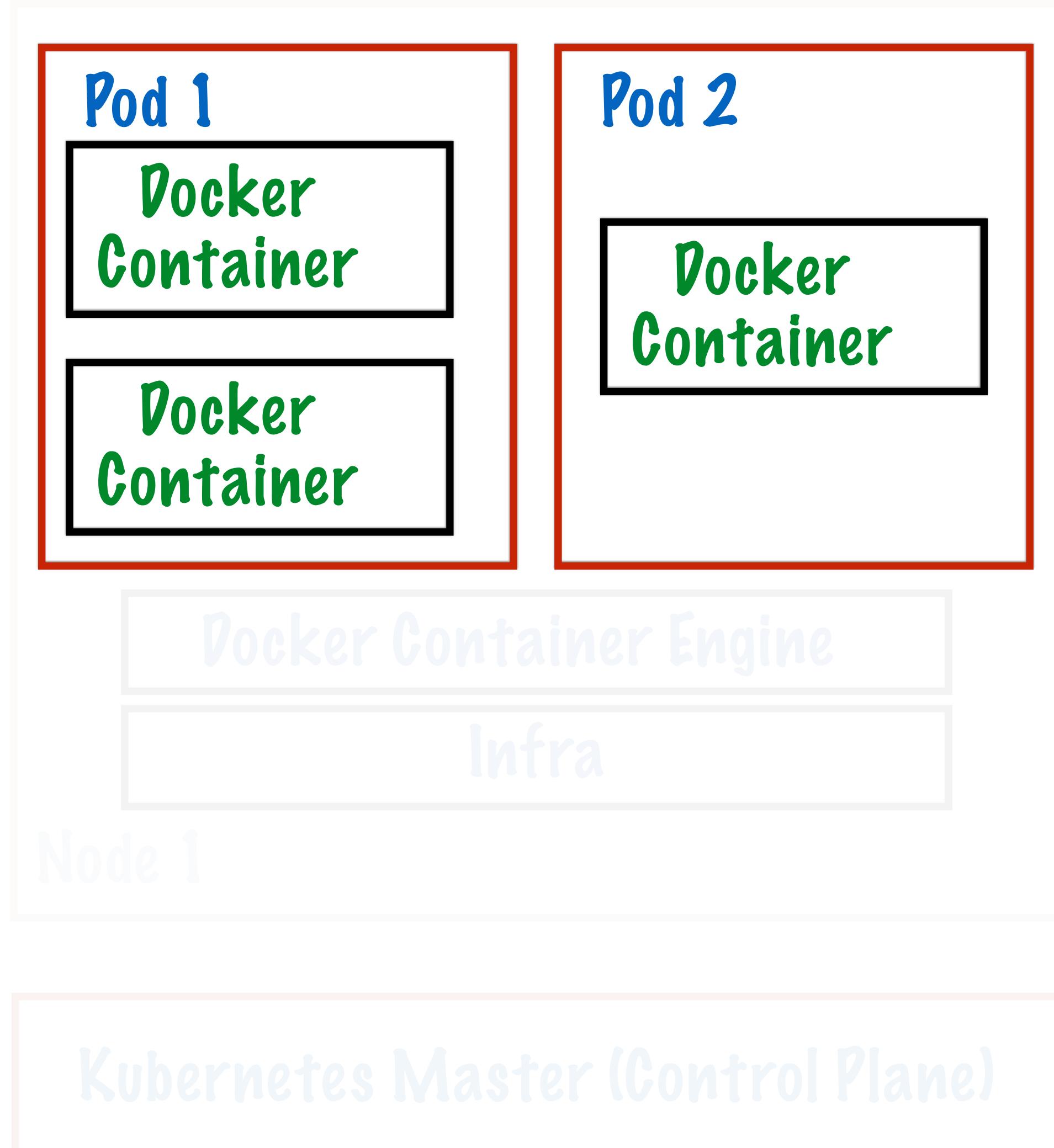
- No auto-healing or scaling or rollback
- Pod crashes? Must be handled at higher level
 - ReplicaSet, Deployment, Service
- Ephemeral: IP addresses are ephemeral

Higher Level Kubernetes Objects

- **ReplicaSet, ReplicationController:** Scaling and healing
- **Deployment:** Versioning and rollback
- **Service:** Static (non-ephemeral) IP and networking
- **Volume:** Non-ephemeral storage

Can A Pod Contain Multiple Containers?

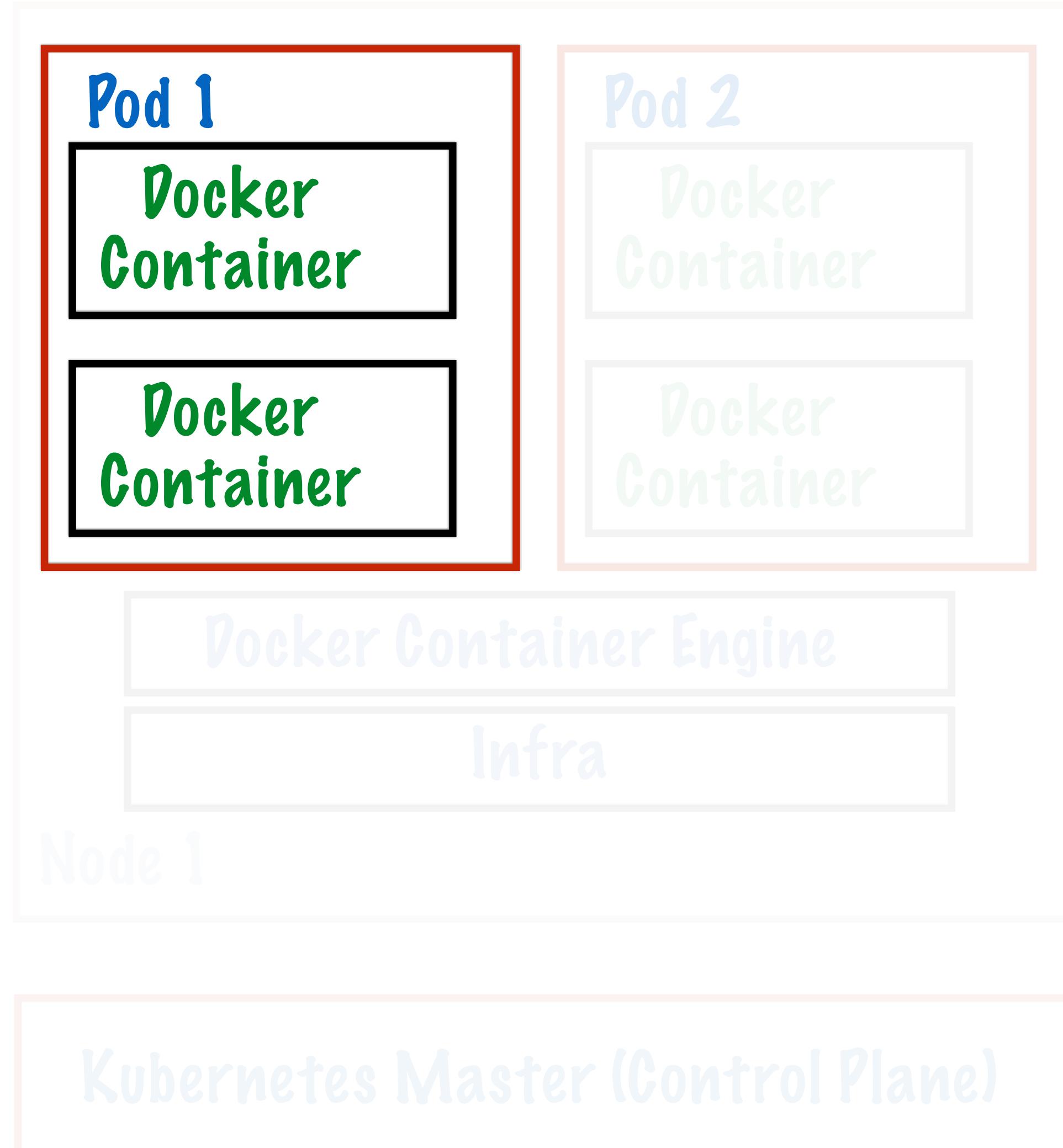
Single and Multi-Container Pods



Pod 1 is a multi-container pod

Pod 2 is a single container pod

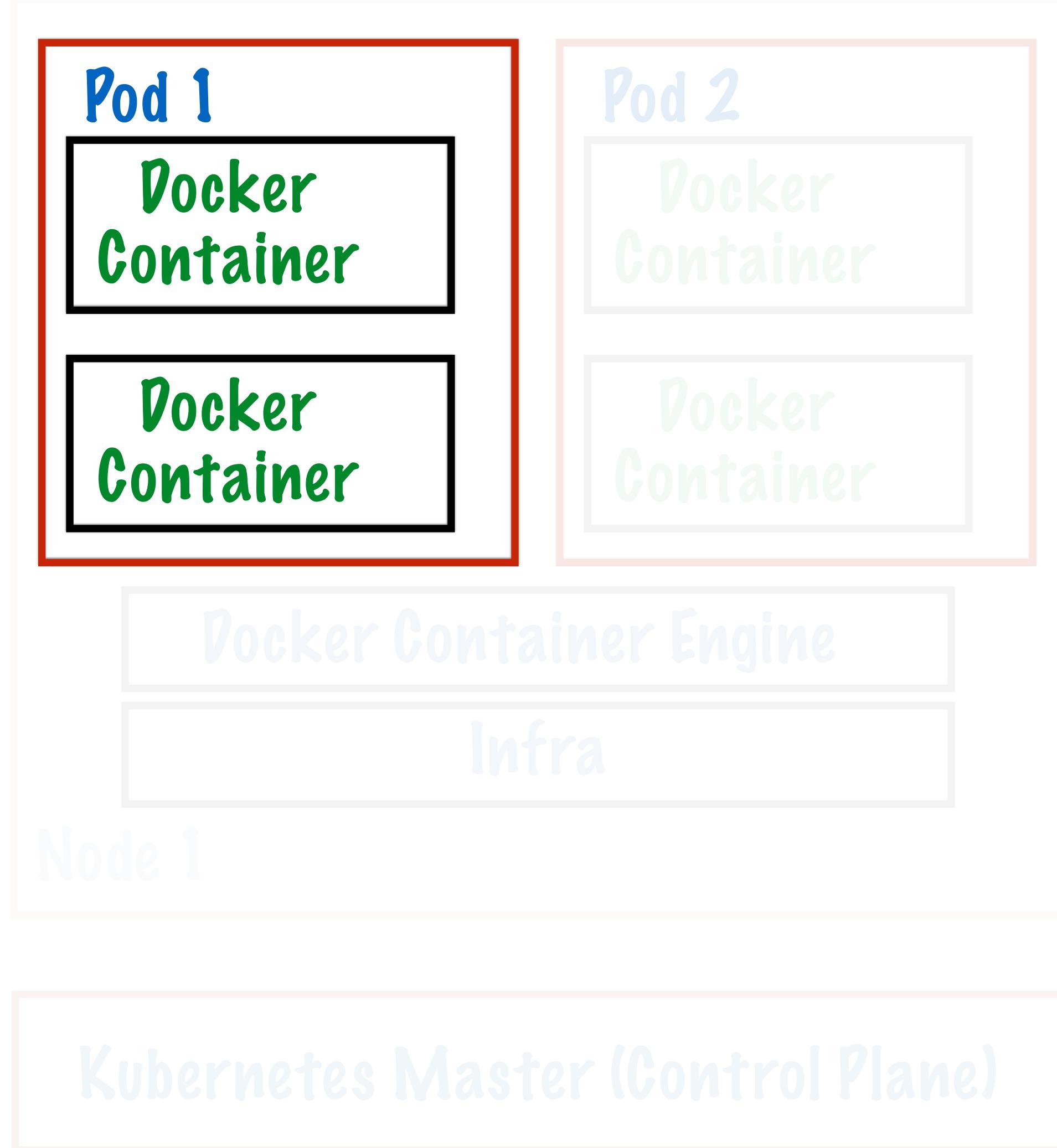
Multi-Container Pods



Containers in same pod share
the same sandbox

Such containers are tightly
coupled

Multi-Container Pods

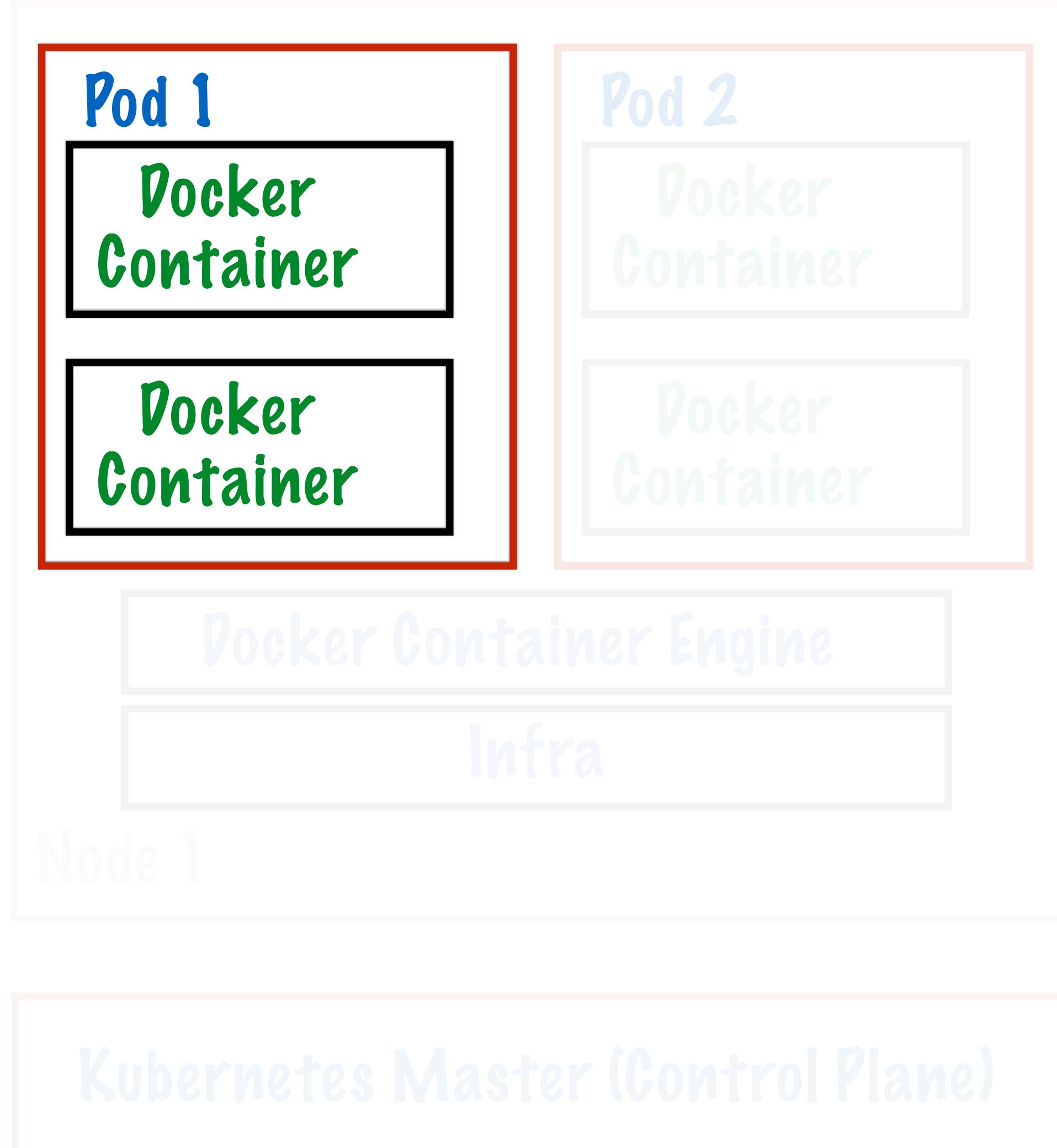


Share access to memory space

Connect to each other using localhost

Share access to the same volumes (storage abstraction)

Multi-Container Pods



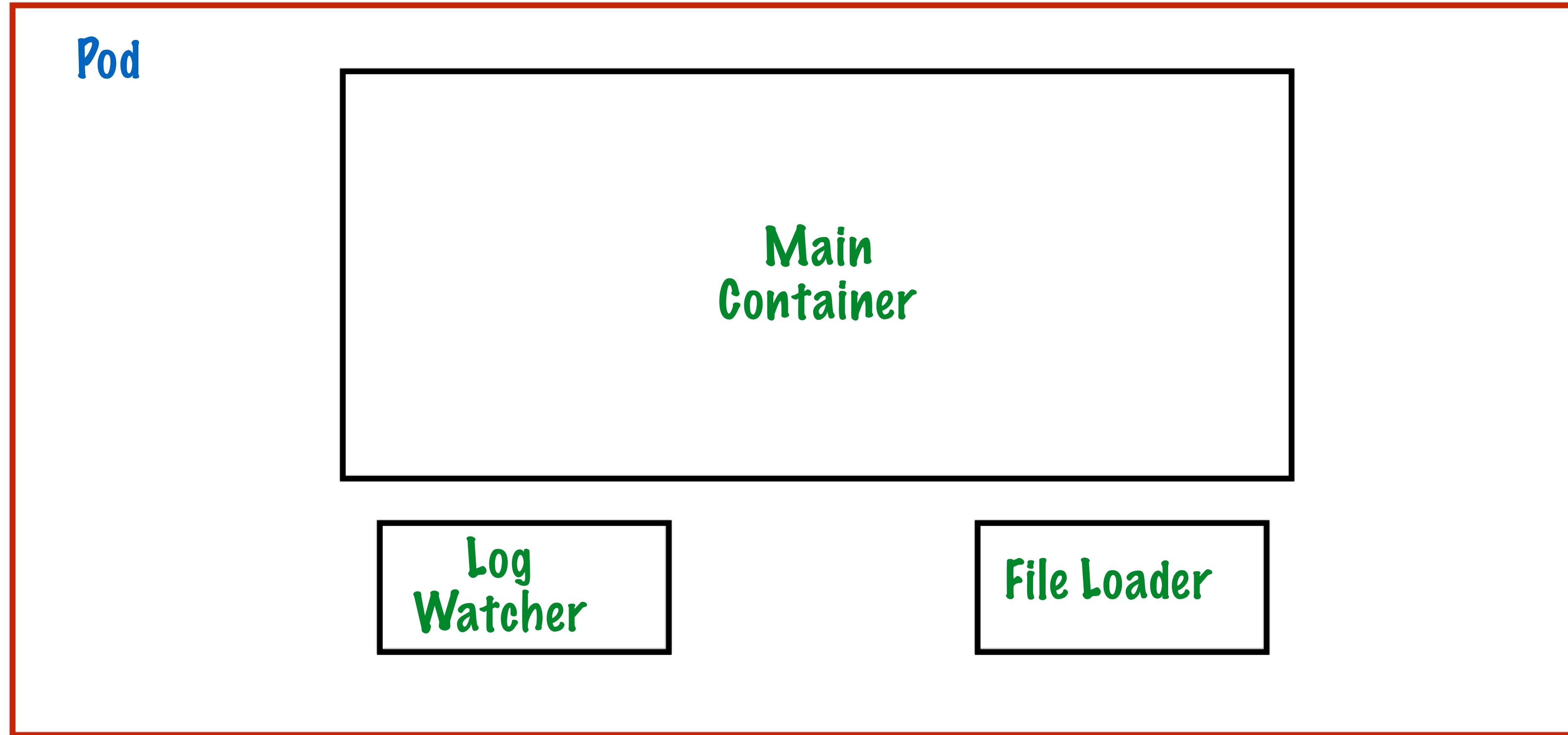
Same parameters such as configMaps

Tight coupling is dangerous

One crashes, all crash

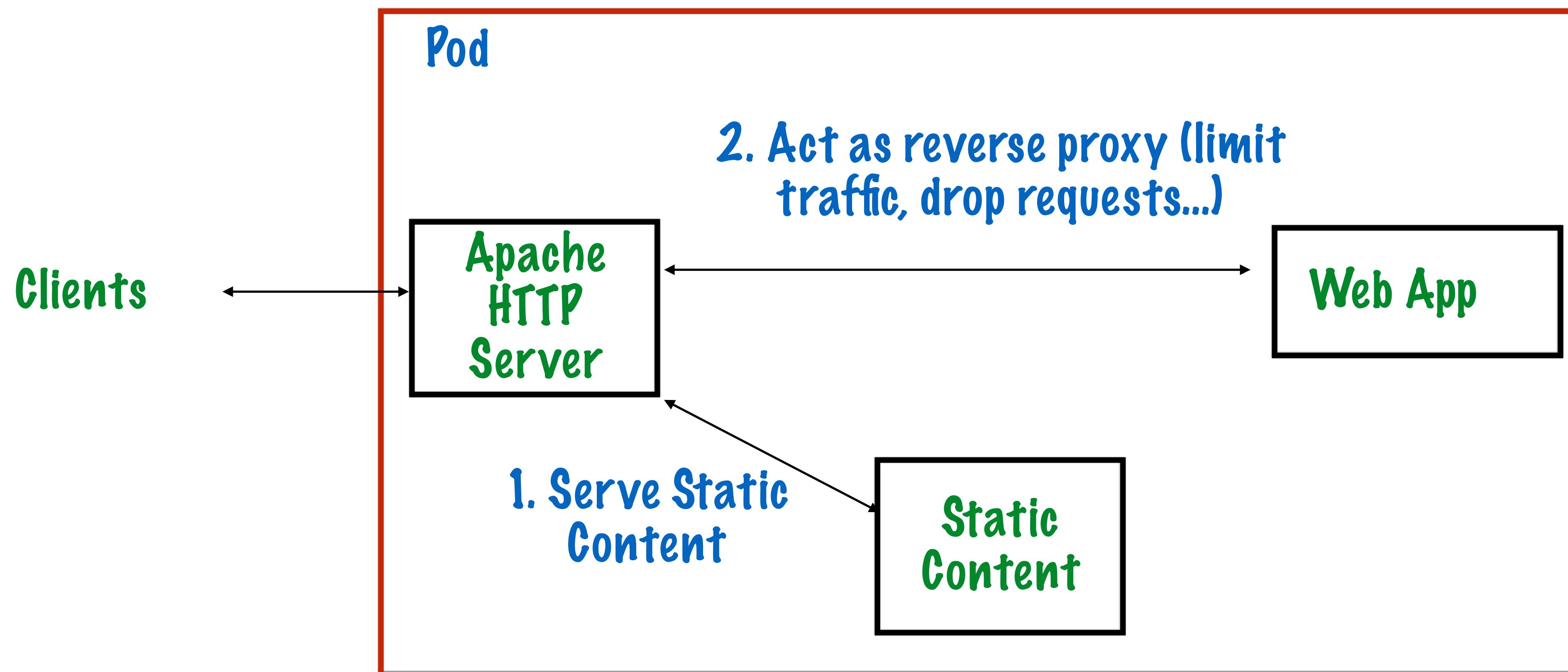
Use Cases for Multi-Container Pod

- Main Container, “Sidecar” supporting containers



Use Cases for Multi-Container Pod

- Proxies, Bridges, Adapters



Anti-Patterns For Multi-Container Pods

- Avoid packing three-tier web app into 1 pod
- Do not pack multiple similar containers in the same pod for scaling
- Use Deployments or ReplicaSets instead
- Micro-services: Simple, independent components

Micro-Services

- Bare metal: apps were very tightly coupled
- Virtual machines: less tightly coupled, but not yet micro-services
- Containers for Micro-services: Simple, independent components

How Does Master-Node Communication Work?

How Does Master-Node Communication Work?

Why do we care?

General Understanding

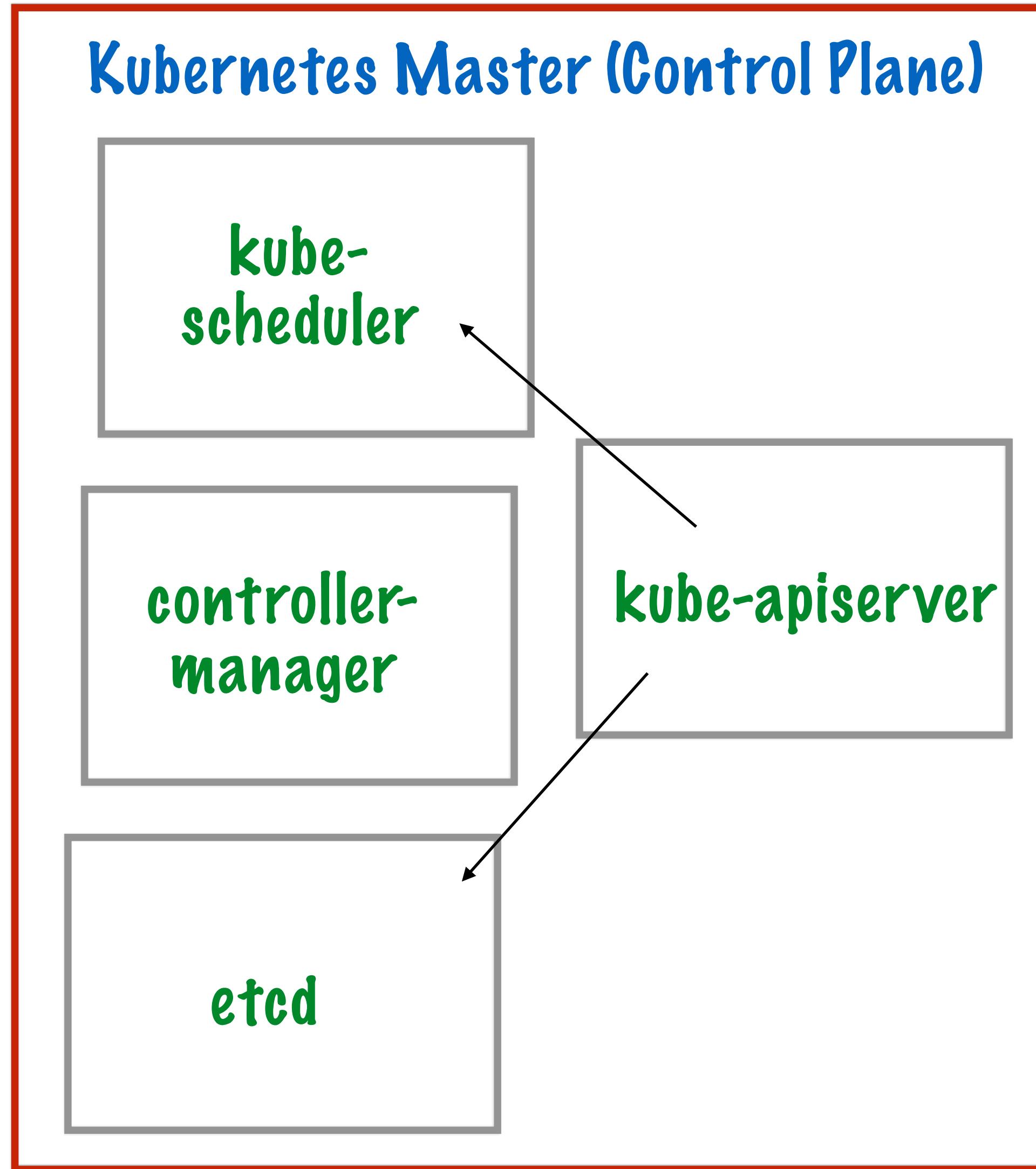
Security

How does it happen?

Cluster to Master

Master to Cluster

Control Plane



Apiserver

etcd

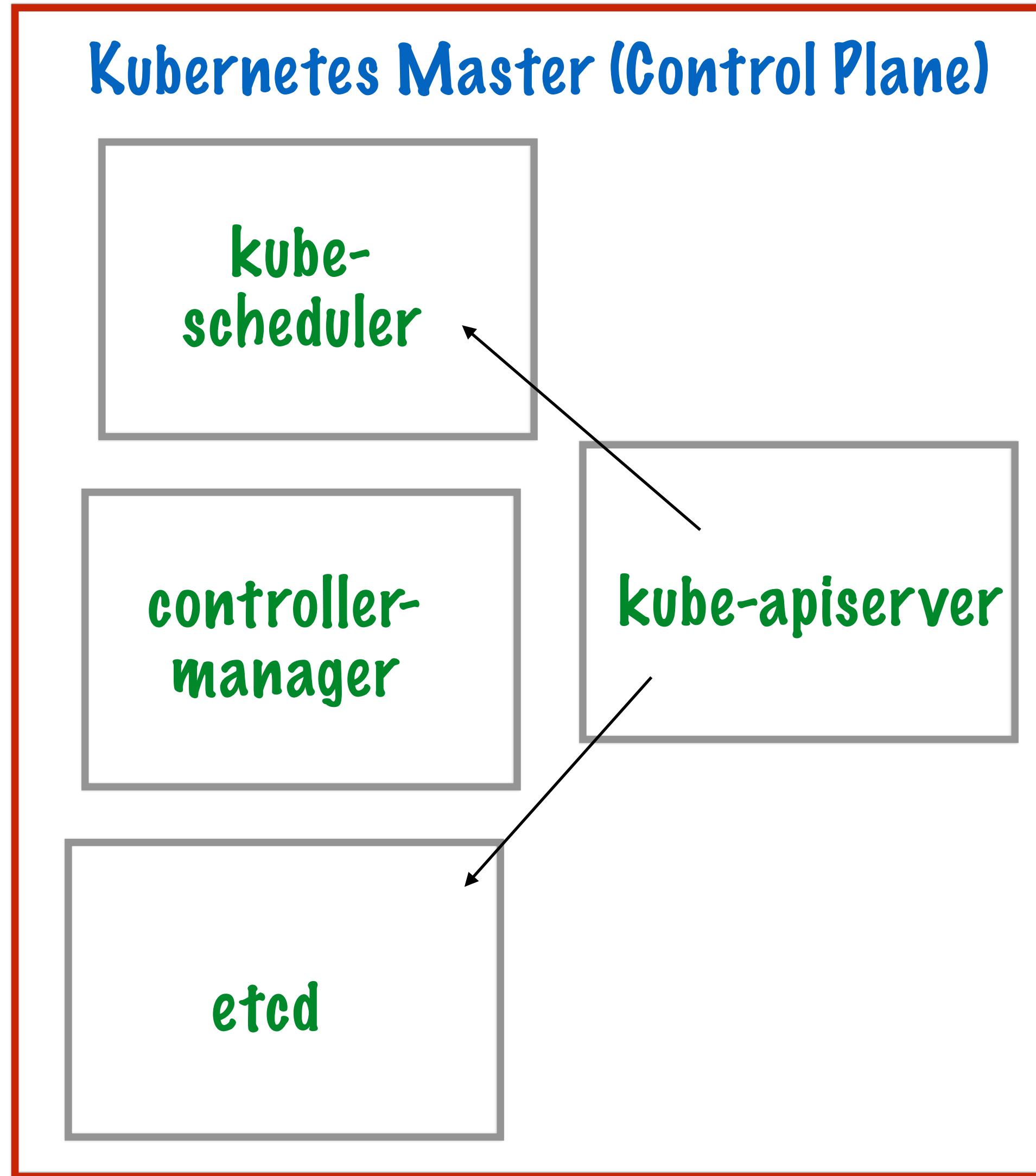
scheduler

controller-manager

cloud-controller-manager

kube-controller-manager

Control Plane



Apiserver

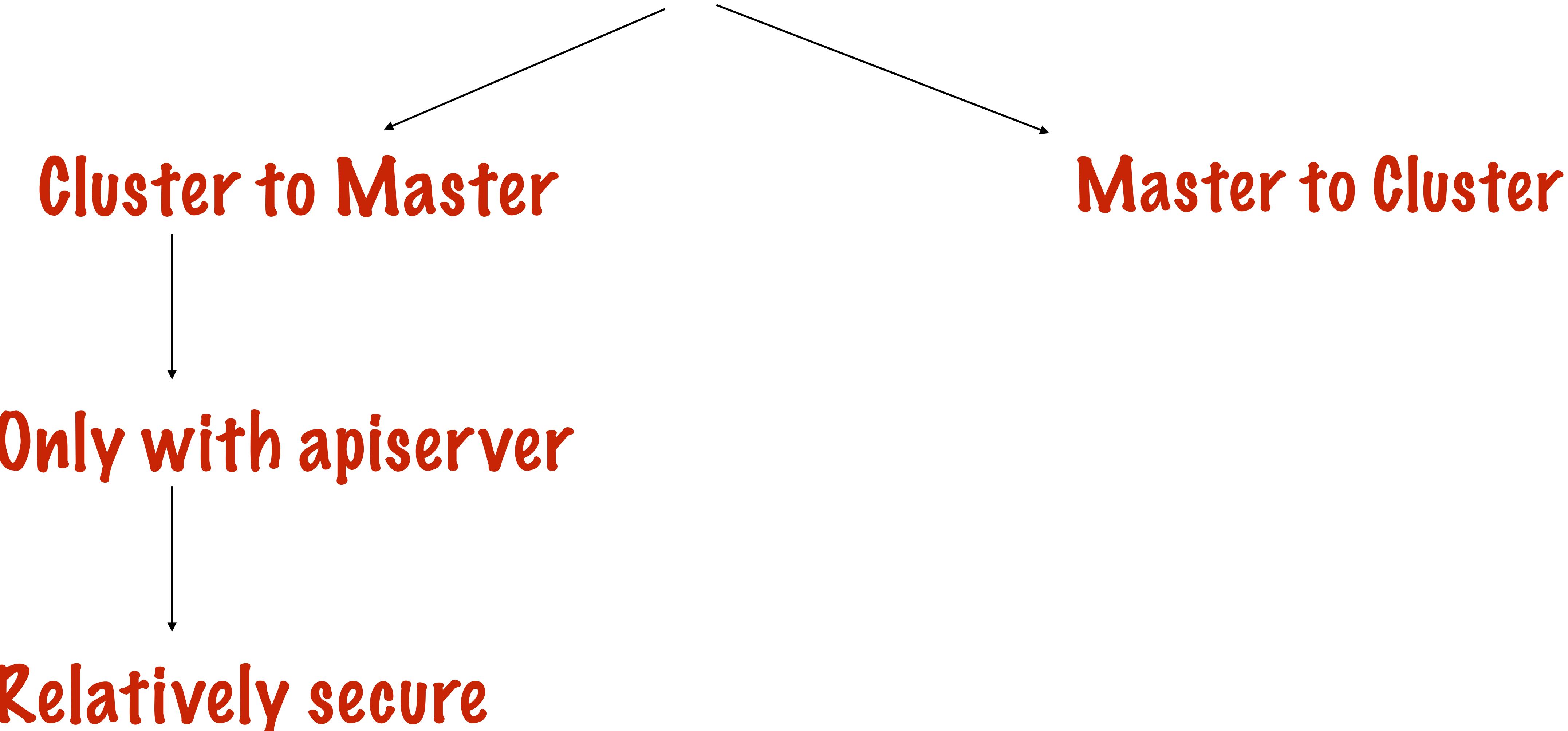
All cluster -> master communication only with apiserver

HTTPS (443)

Relatively secure

How Does Master-Node Communication Work?

How does it happen?



How Does Master-Node Communication Work?

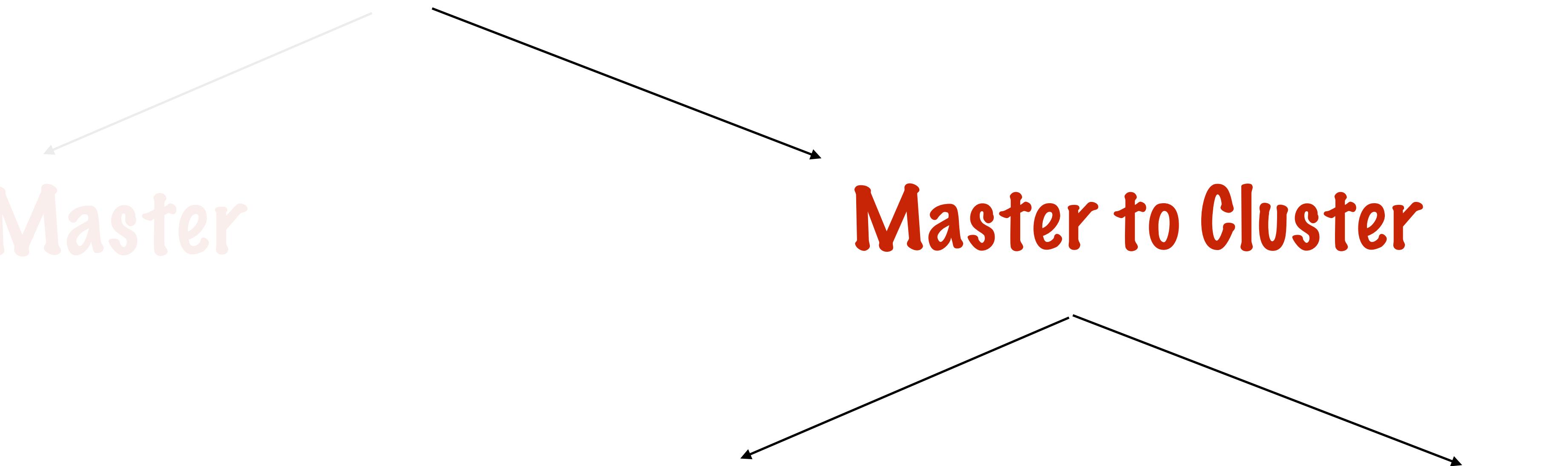
How does it happen?

Cluster to Master



Relatively secure

Master to Cluster



Master -> Cluster Communication

apiserver -> kubelet

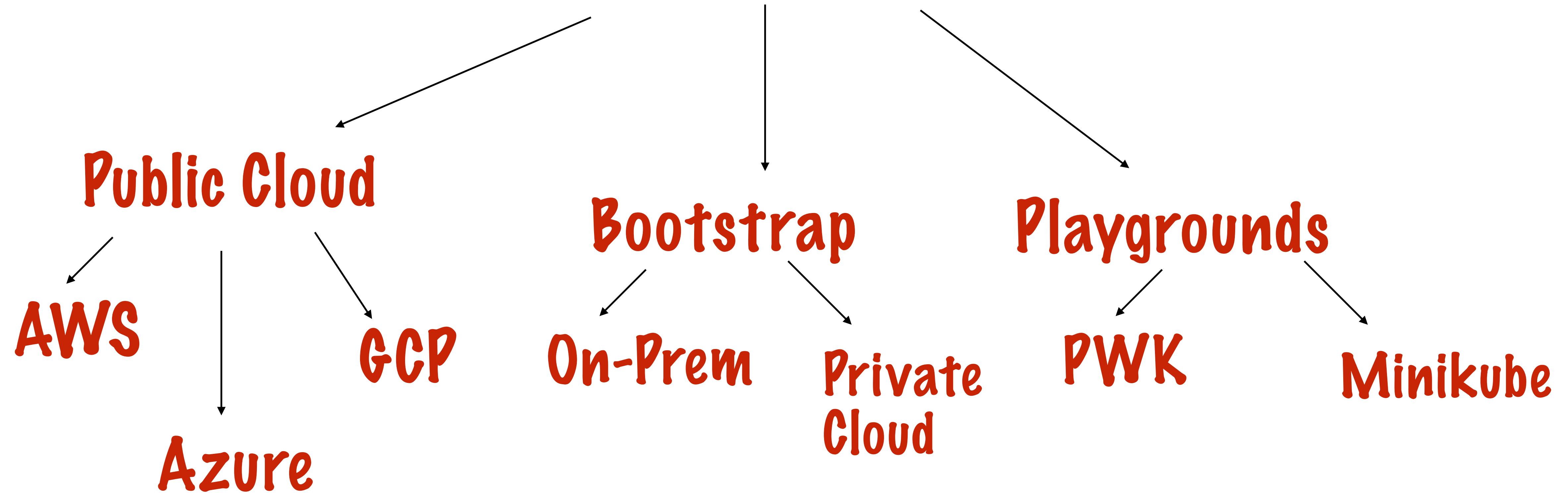
- Certificate not verified by default
- Vulnerable to man-in-the-middle attacks
- Don't run on public network
- To harden
 - set --kubelet-certificate-authority
 - use SSH tunnelling

apiserver -> nodes/pods/services

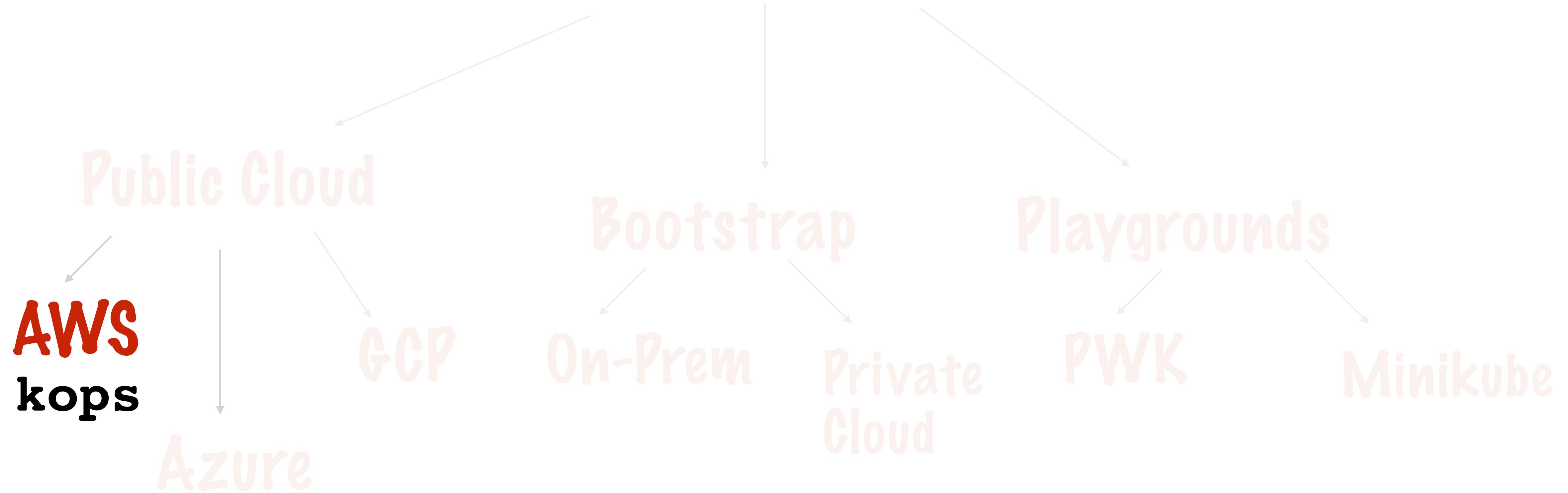
- Not safe
- Plain HTTP
- Neither authenticated or encrypted
- On public clouds, SSH tunnelling provided by cloud provider e.g. GCP

Where Can We Run Kubernetes?

Where Can We Run Kubernetes?



Where Can We Run Kubernetes?



Where Can We Run Kubernetes?

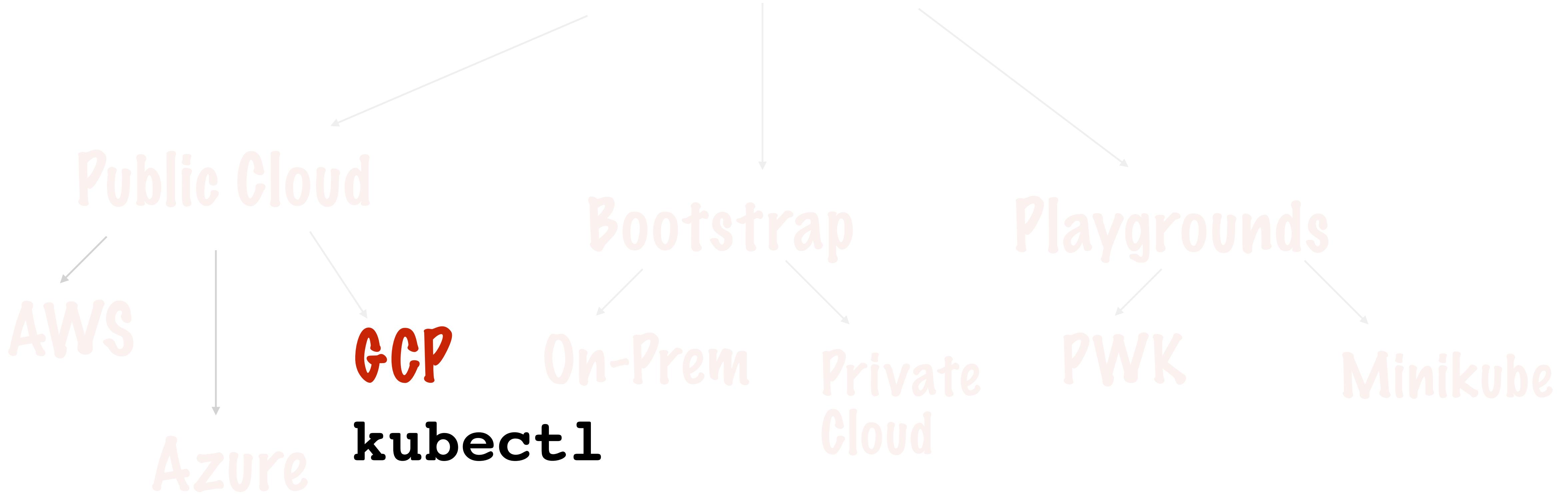


GKE: Google Kubernetes Engine

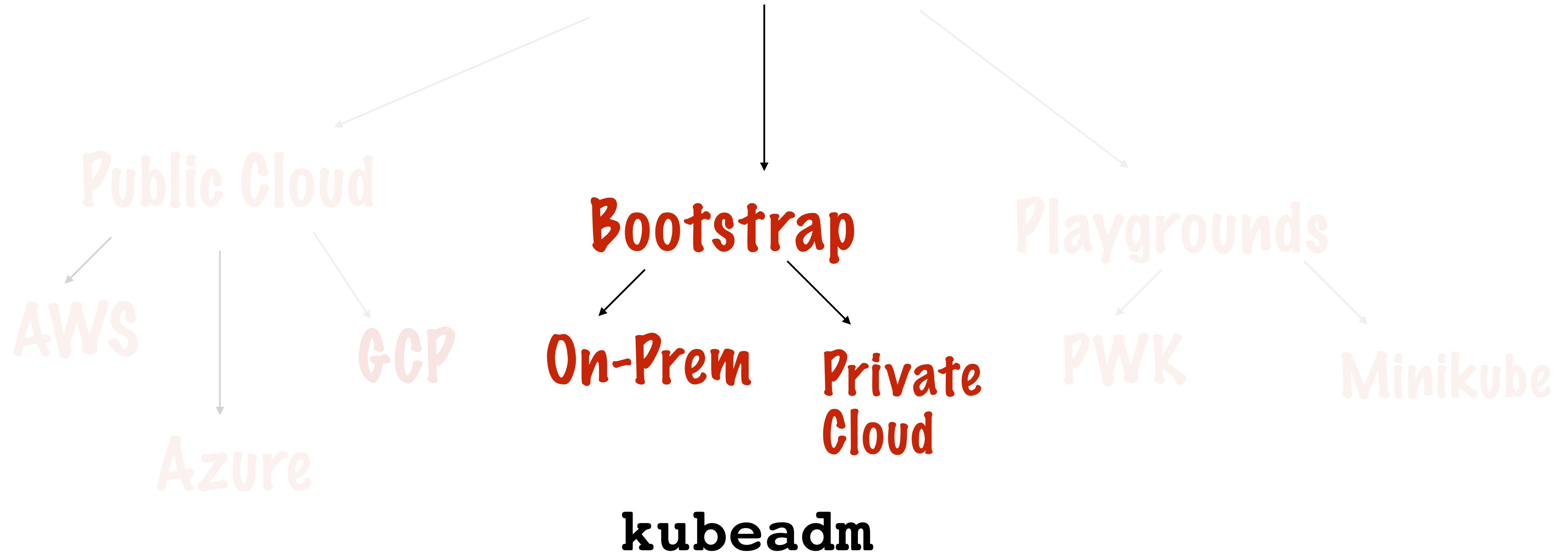
Infra on GCE Virtual machines

GCE: Google Compute Engine (IaaS)

Where Can We Run Kubernetes?



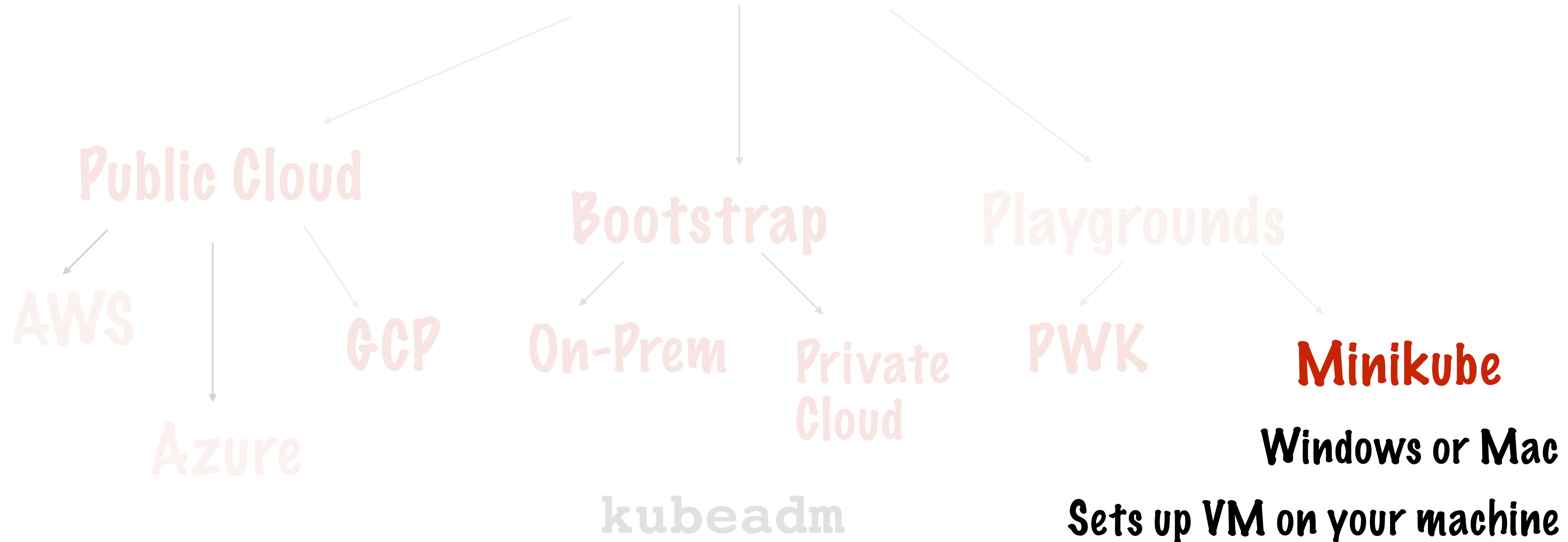
Where Can We Run Kubernetes?



Where Can We Run Kubernetes?



Where Can We Run Kubernetes?



Can Kubernetes Be Used In A Hybrid, Multi-Cloud World?

Hybrid, Multi-Cloud

- Hybrid = On-prem + Public Cloud
 - On-premise: bare metal or VMs
 - Legacy infra, large on-prem datacenters
 - Medium-term importance
- Multi-Cloud
 - More than 1 public cloud provider
 - Strategic reasons, vendor lock-in (Amazon buying Whole Foods)

IaaS, PaaS and Containers

Infra-As-A-Service

Microsoft Azure VMs

AWS Elastic Compute Cloud (EC2)

GCP Compute Engine (GCE)

Platform-As-A-Service

Microsoft App Service

AWS Elastic Beanstalk

GCP App Engine

IaaS, PaaS and Containers

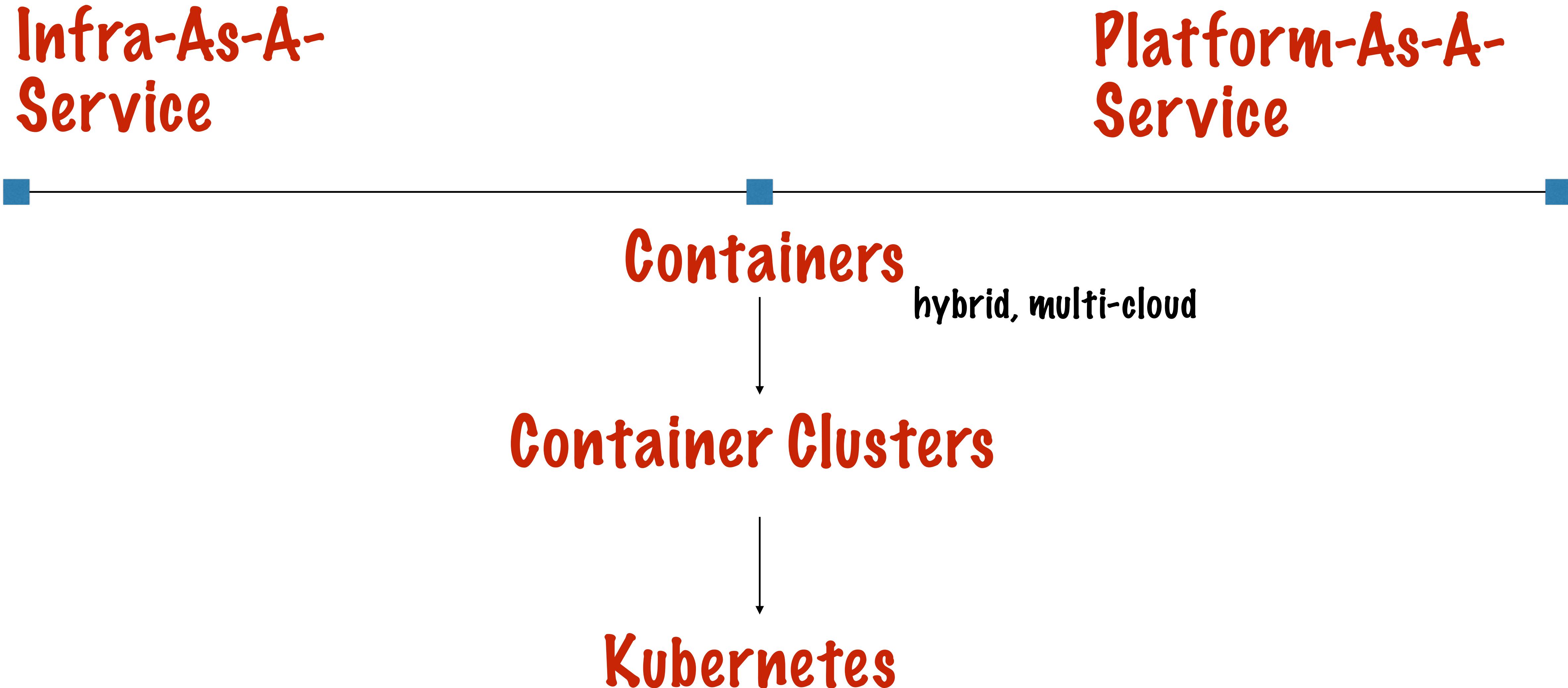
Infra-As-A-Service

- Migration is painful
- VMs not that portable

Platform-As-A-Service

- Tie-in to cloud provider
- Curtail control over infra

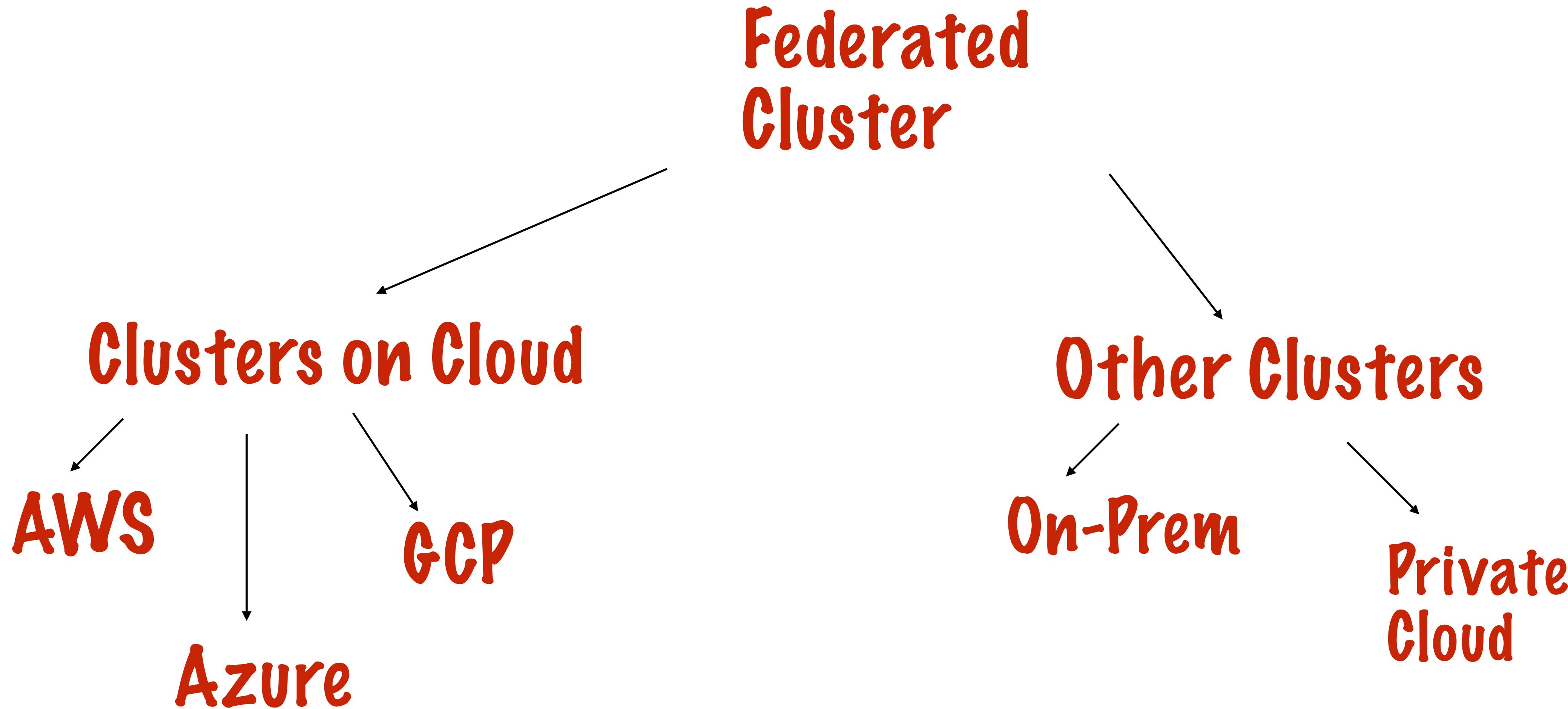
Containers as Middle Path



Kubernetes for Orchestration

- **Fault-tolerance:** Pod/Node failures
- **Rollback:** Advanced deployment options
- **Auto-healing:** Crashed containers restart
- **Auto-scaling:** More clients? More demand
- **Load-balancing:** Distribute client requests
- **Isolation:** Sandboxes so that containers don't interfere

Federated Clusters on Kubernetes



Federated Clusters

Individual Cluster

- All nodes on same infra
- Administer with `kubectl`

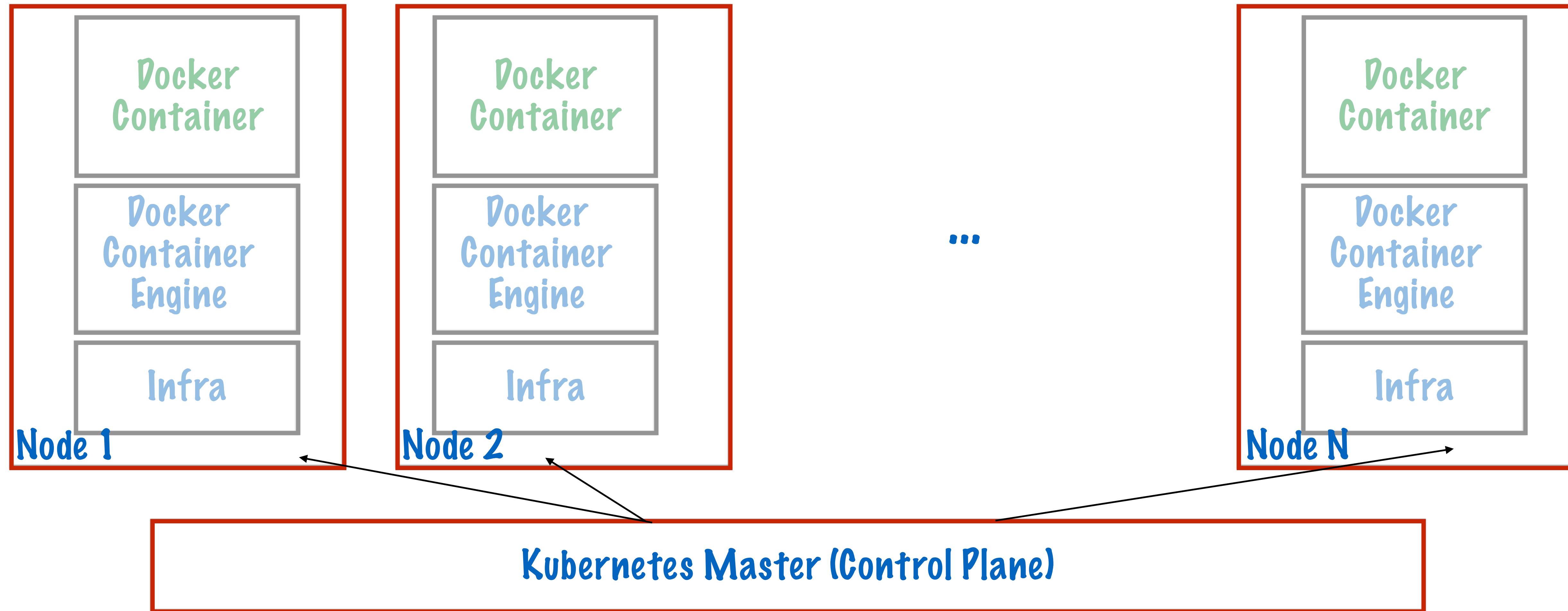
Federation (Federated Cluster)

- Nodes in multiple clusters
- Administer with `kubefed`

Do The Internals Of Kubernetes Vary Based On Where Cluster Runs?

Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs



Kubernetes Master

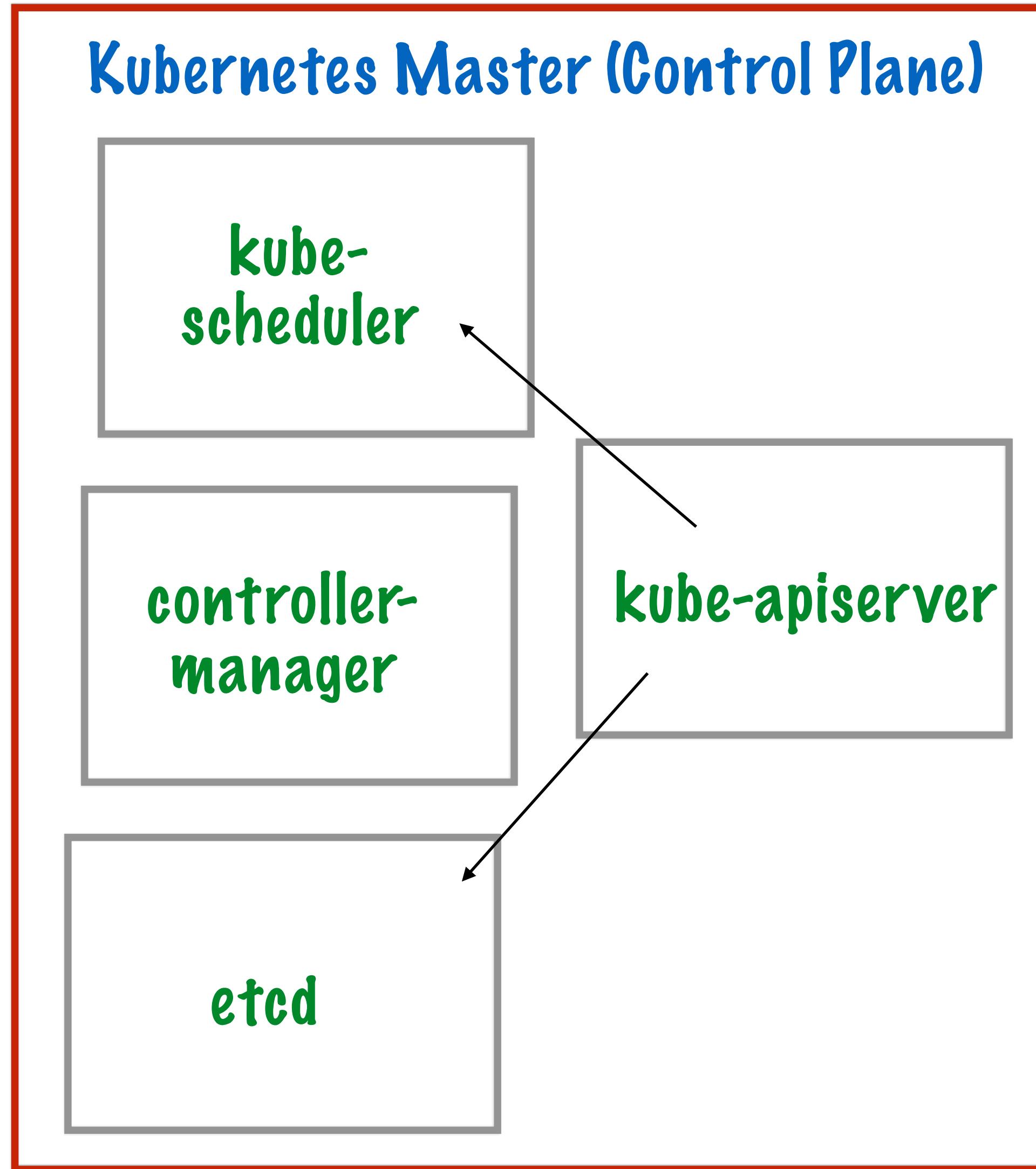


One or more nodes designated as master

Several kubernetes processes run on master

Multi-master for high-availability

Control Plane



Apiserver

etcd

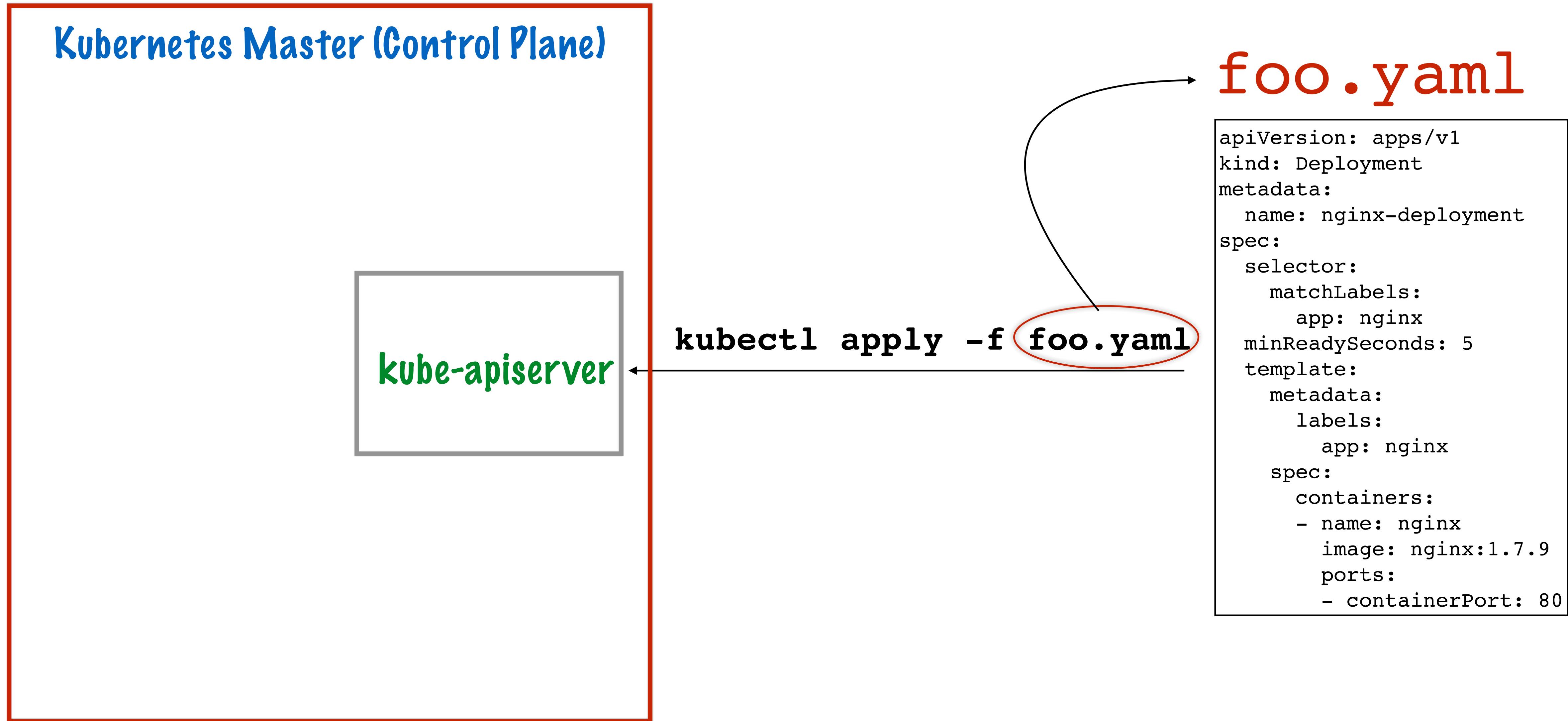
scheduler

controller-manager

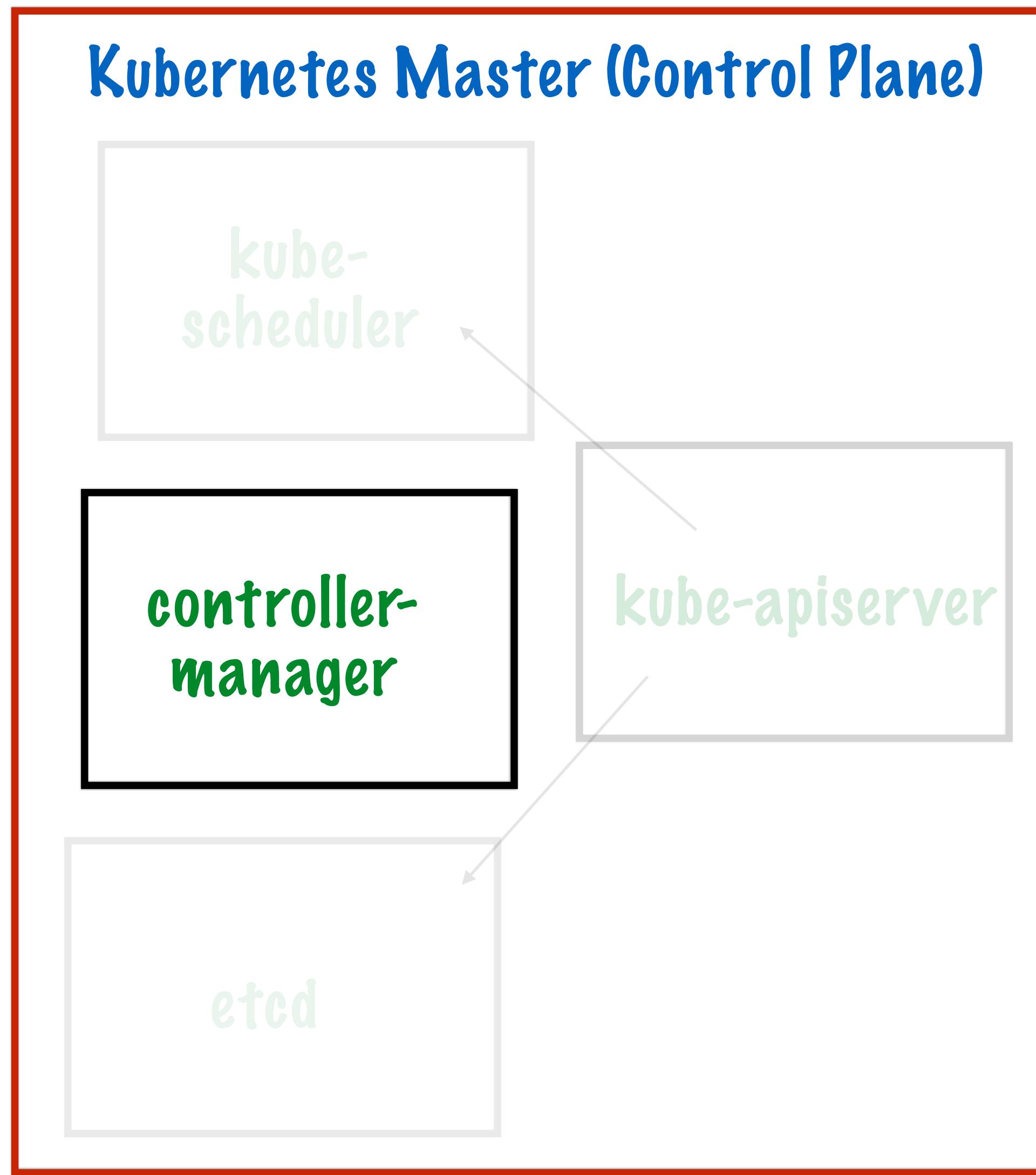
cloud-controller-manager

kube-controller-manager

kube-apiserver

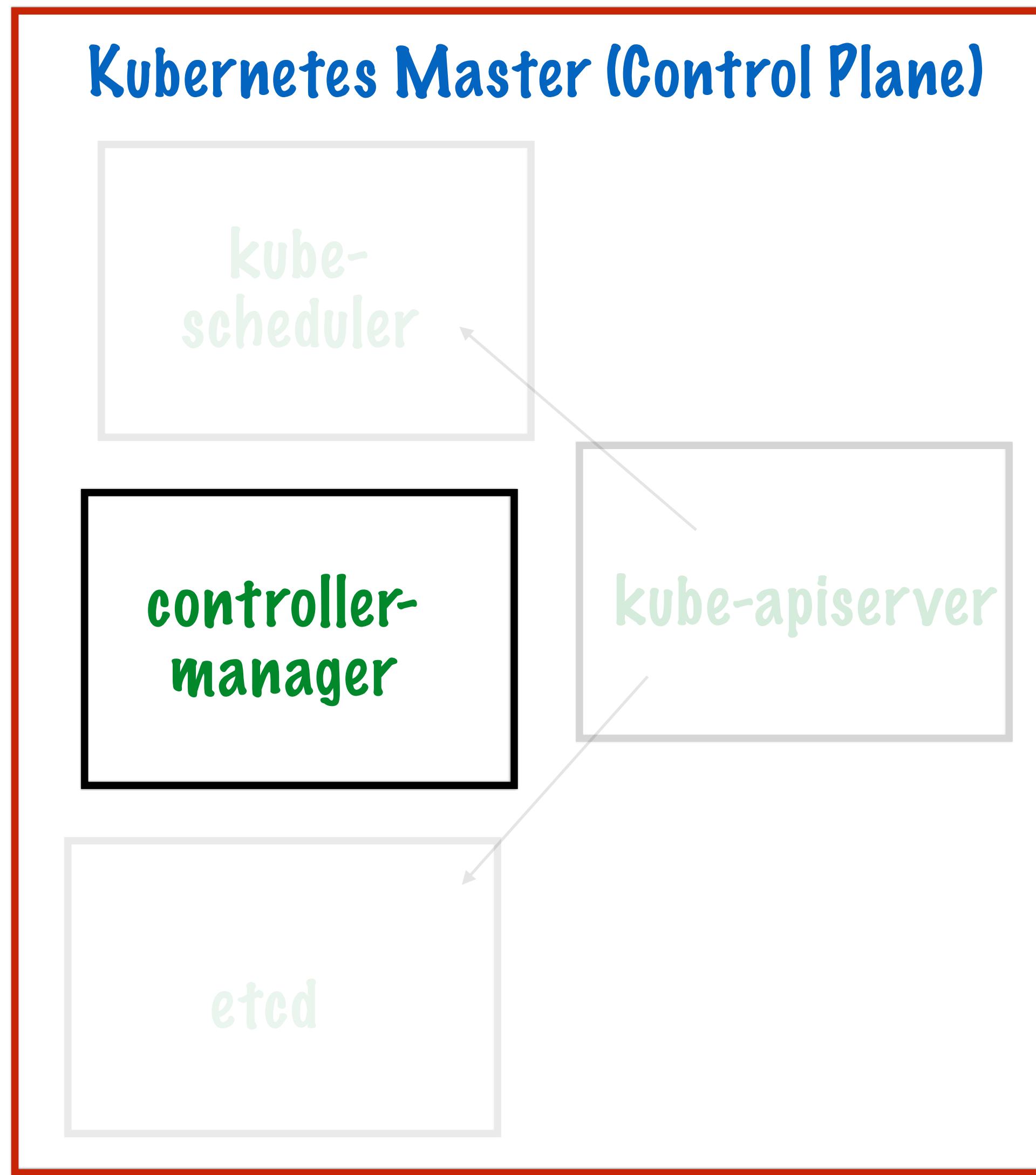


Control Plane



Apiserver
etcd
scheduler
controller-manager
cloud-controller-manager
kube-controller-manager

Control Plane



Apiserver
etcd
scheduler
controller-manager
cloud-controller-manager
kube-controller-manager

controller-manager

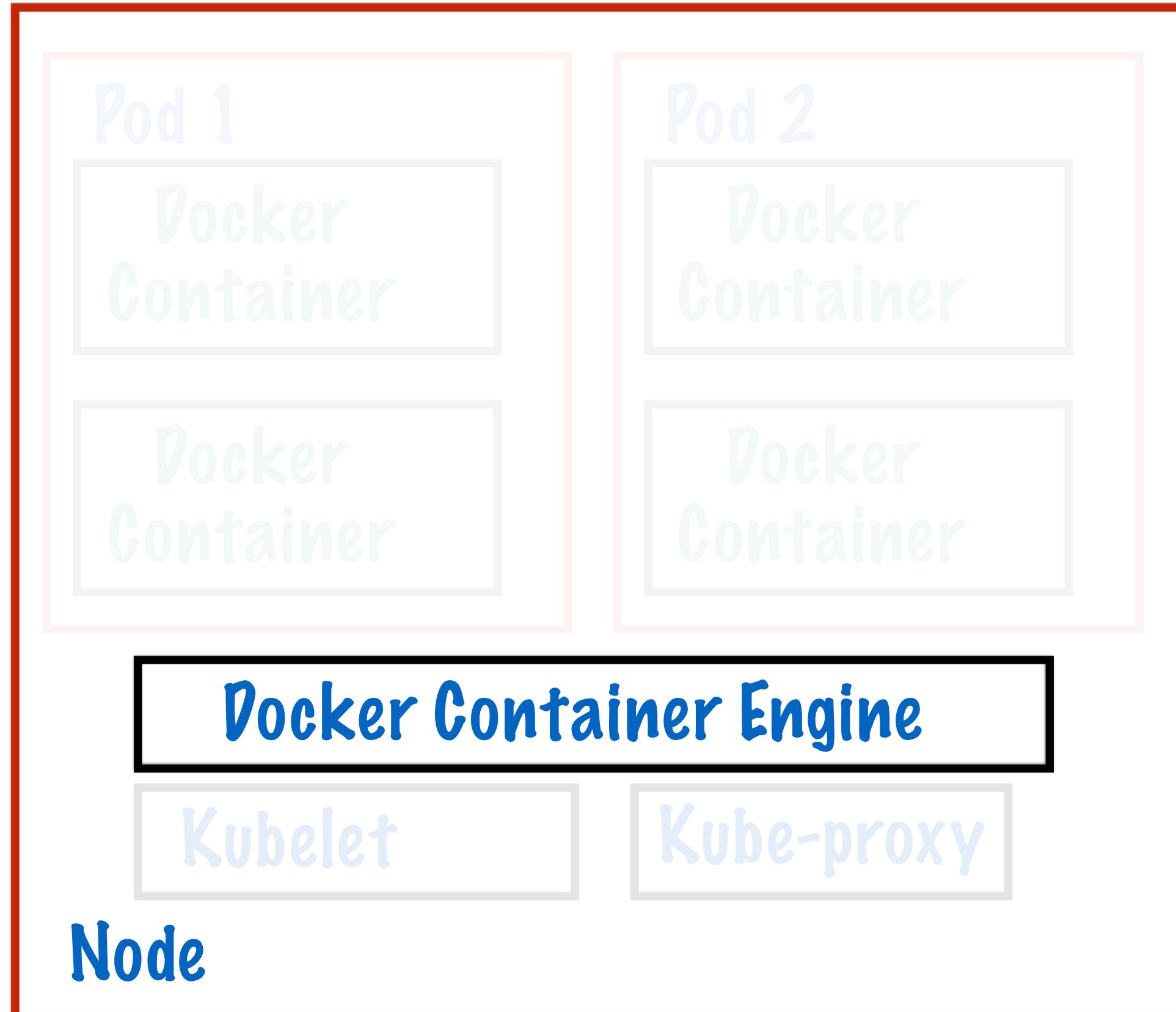
cloud-controller-manager

- Used when k8s runs on cloud
- Cloud-specific

kube-controller-manager

- Used when not running on public cloud
- Not infra-specific

Kubernetes Node (Minion)



Kubernetes Master (Control Plane)

Container engine

Could be Docker or rkt

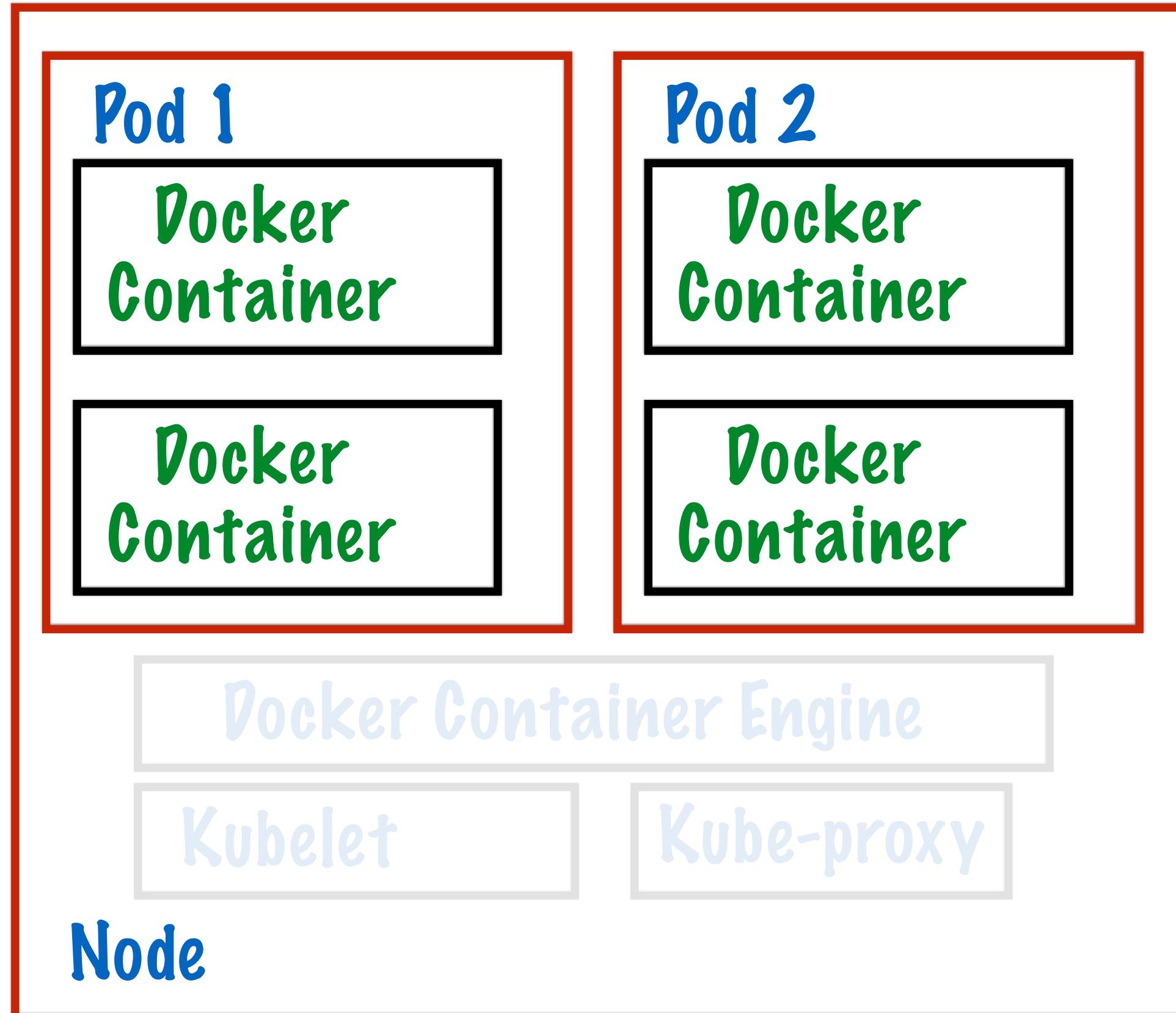
Works with kubelet

Pulling images

Start/stop

containerd

Kubernetes Node (Minion)



Kubernetes Master (Control Plane)

CRI

Container Runtime Interface

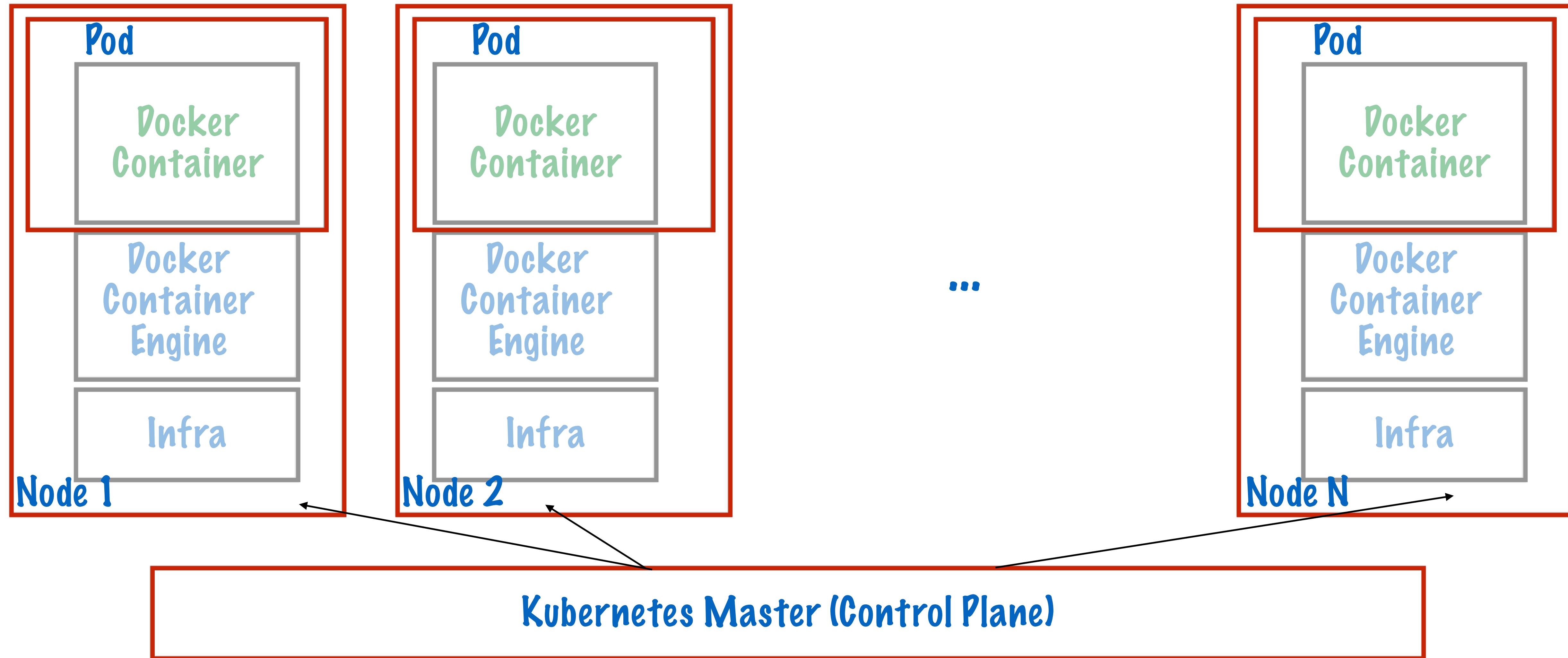
Tight-coupling between

Kubelet

Container Engine

Kubernetes: Cluster Orchestration

Potentially thousands of containers on hundreds of VMs



Pods Limitations

- No auto-healing or scaling or rollback
- Pod crashes? Must be handled at higher level
 - ReplicaSet, Deployment, Service
- Ephemeral: IP addresses are ephemeral

Higher Level Kubernetes Objects

- **ReplicaSet, ReplicationController:** Scaling and healing
- **Deployment:** Versioning and rollback
- **Service:** Static (non-ephemeral) IP and networking
- **Volume:** Non-ephemeral storage

Volumes Are Cloud-Specific

- Different volume abstractions for various cloud providers
 - `awsElasticBlockStore`
 - `azureDisk`
 - `azureFile`
 - `gcePersistentDisk`

How Do We Work With Kubernetes?

How Do We Work With Kubernetes?

kubectl

Most common command line utility

Makes POST requests to apiserver of control plane

kubeadm

Bootstrap cluster when not on cloud Kubernetes service

To create cluster out of individual infra nodes

kubefed

Administer federated clusters

Federated cluster -> group of multiple clusters (multi-cloud, hybrid)

kubelet, kube-proxy, ...

How Do We Work With Kubernetes?

kubectl

Most common command line utility

Makes POST requests to apiserver of control plane

kubeadm

Under the hood, all of these tools are interacting with

To create cluster out of individual infra nodes

The Kubernetes API

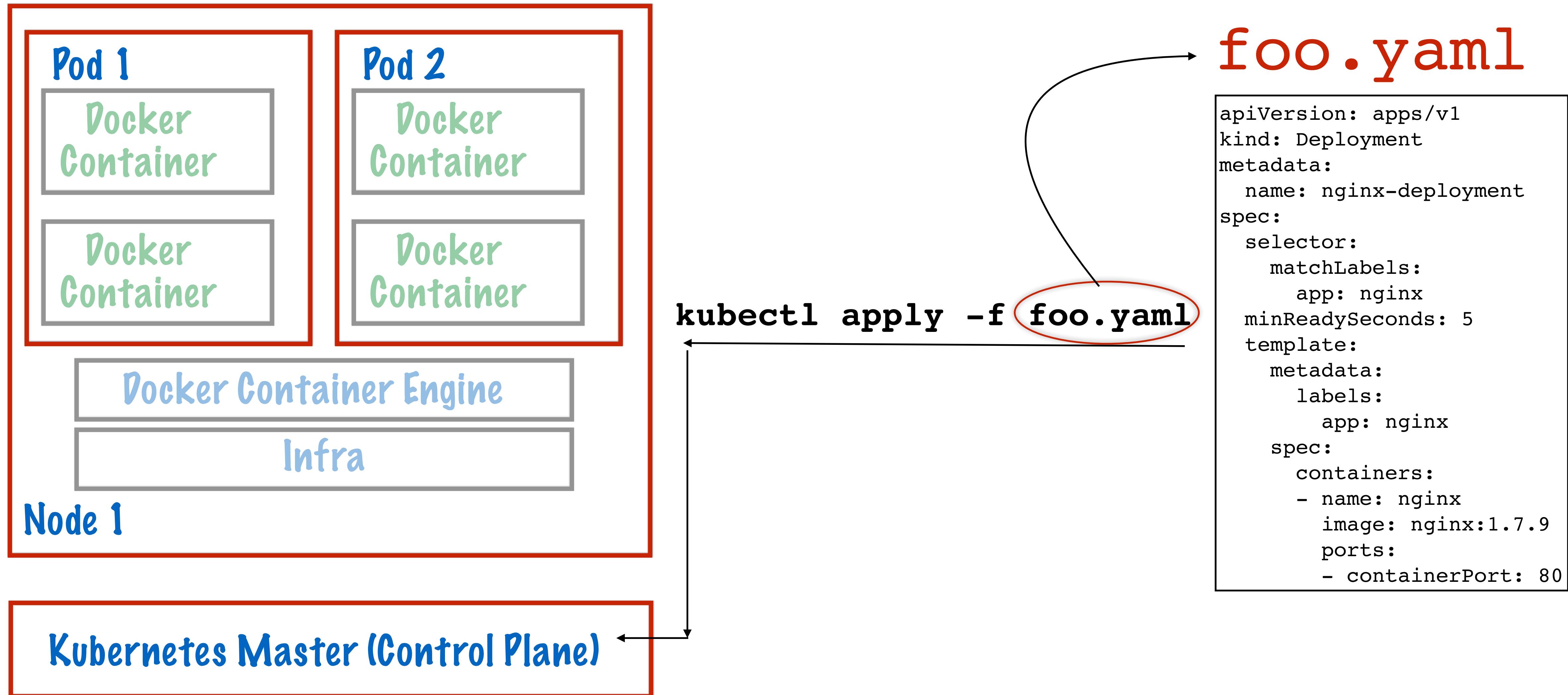
kubefed

Administer federated clusters

Federated cluster -> group of multiple clusters (multi-cloud, hybrid)

kubelet, kube-proxy, ...

Working with Kubernetes: `kubectl`



The Kubernetes API

- Apiserver within control plane exposes API endpoints
- Clients hit these endpoints with RESTful API calls
- These clients could be command line tools such as kubectl, kubeadm...
- Could also be programmatic calls using client libraries

Objects

- Kubernetes Objects are persistent entities
- Everything is an object...
- Pod, ReplicaSet, Deployment, Node ... all are objects
- Send object specification (usually in .yaml or .json)

Pod Object Specification

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
      httpHeaders:
      - name: X-Custom-Header
        value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

What type of Object? Pod

ReplicaSet Object Specification

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
      env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and uncomment the
          # line below.
          # value: env
      ports:
        - containerPort: 80
```

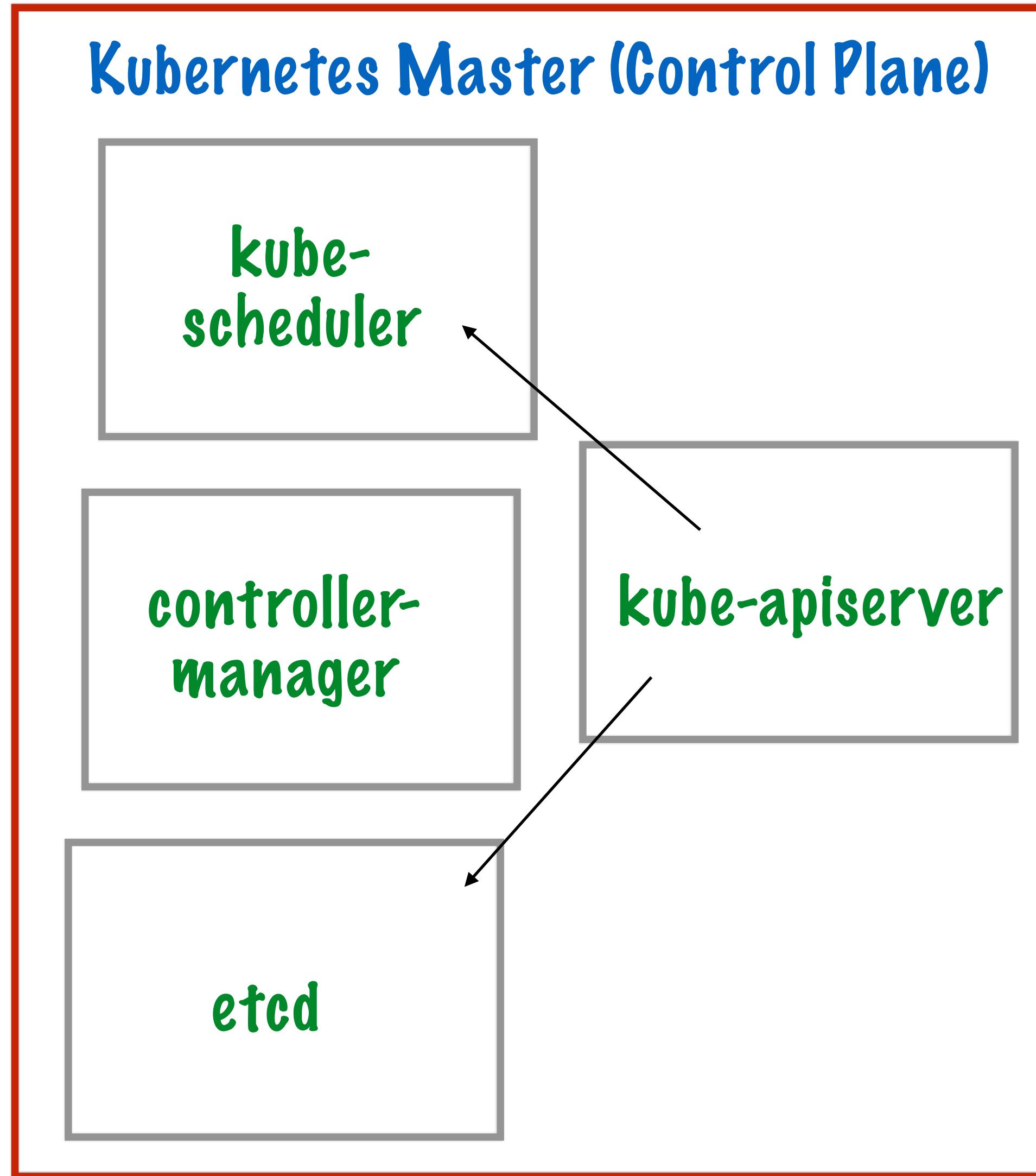
What type of Object? ReplicaSet

Deployment Object Specification

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

What type of Object? Deployment

controller-manager



Different master processes

Actual state <-> Desired State

cloud-controller-manager

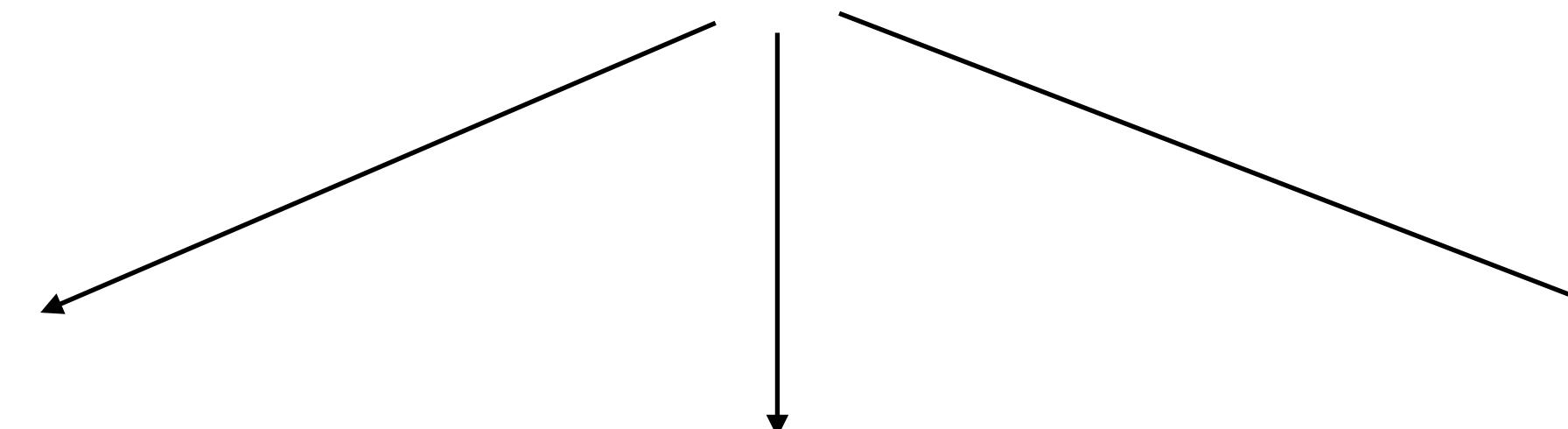
kube-controller-manager

Three Object Management Methods

Imperative
Commands

Imperative Object
Configuration

Declarative Object
Configuration

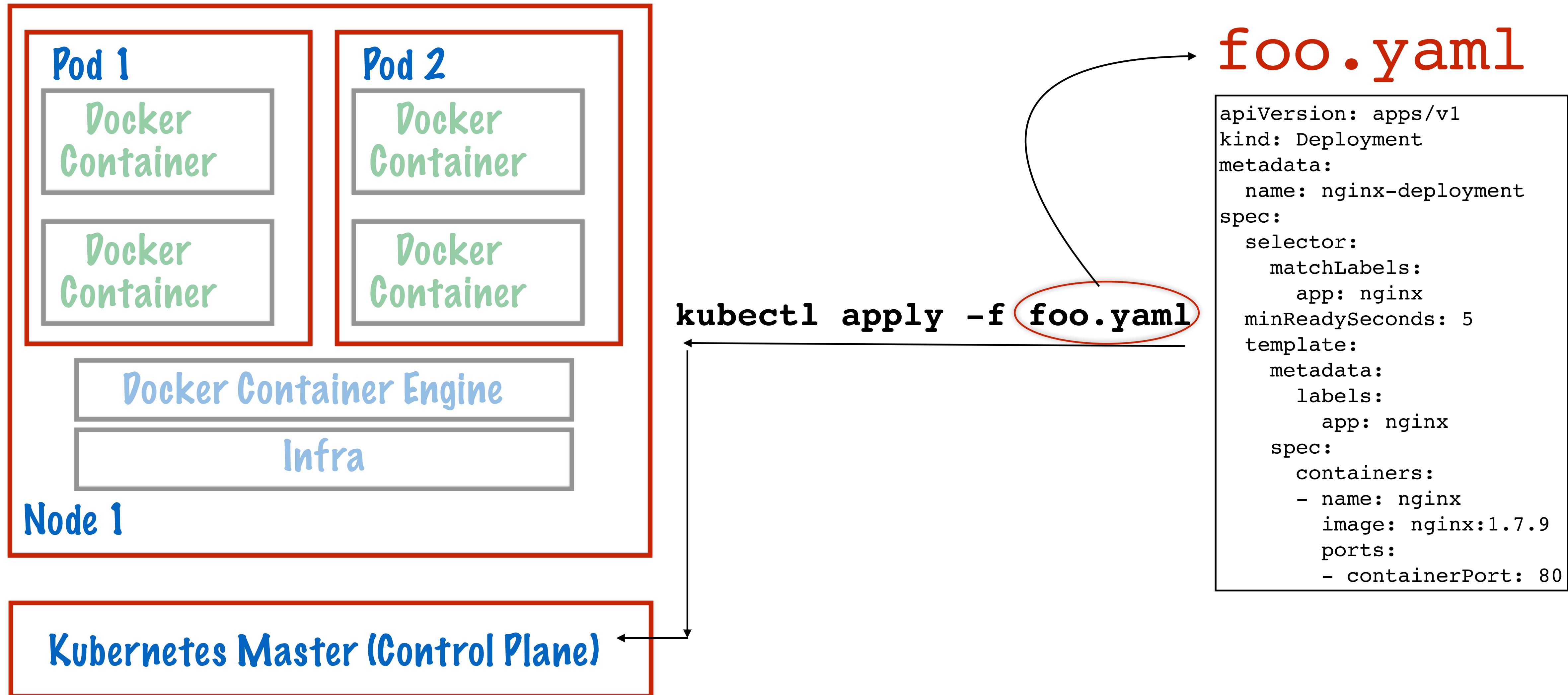


Don't mix and match!

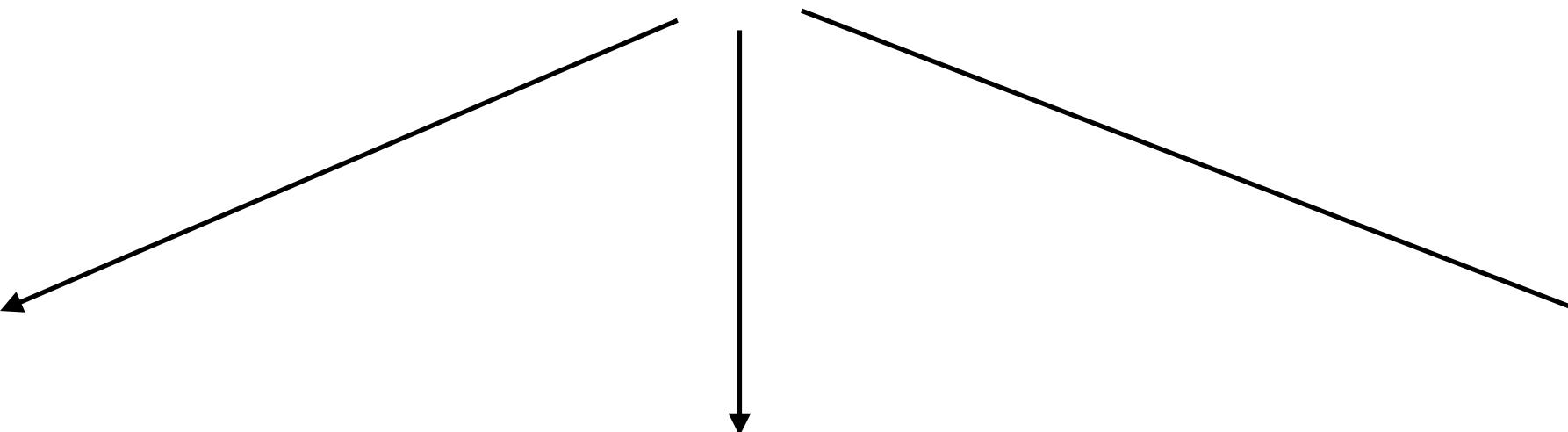
Declarative is preferred

What are the imperative and declarative methods of object management?

Working with Kubernetes: `kubectl`



Three Object Management Methods



**Imperative
Commands**

No .yaml or config files

kubectl run ...

kubectl expose ...

kubectl autoscale ...

**Imperative Object
Configuration**

**kubectl + yaml or
config files used**

kubectl create -f config.yaml

kubectl replace -f config.yaml

kubectl delete -f config.yaml

**Declarative Object
Configuration**

**Only .yaml or config
files used**

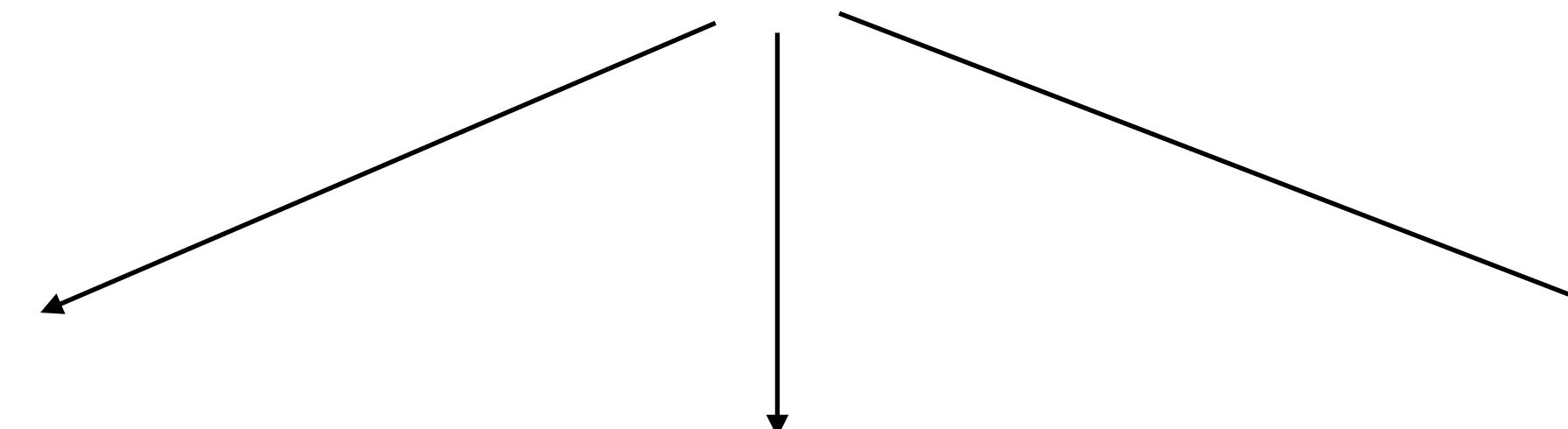
kubectl **apply** -f config.yaml

Three Object Management Methods

Imperative
Commands

Imperative Object
Configuration

Declarative Object
Configuration



Don't mix and match!

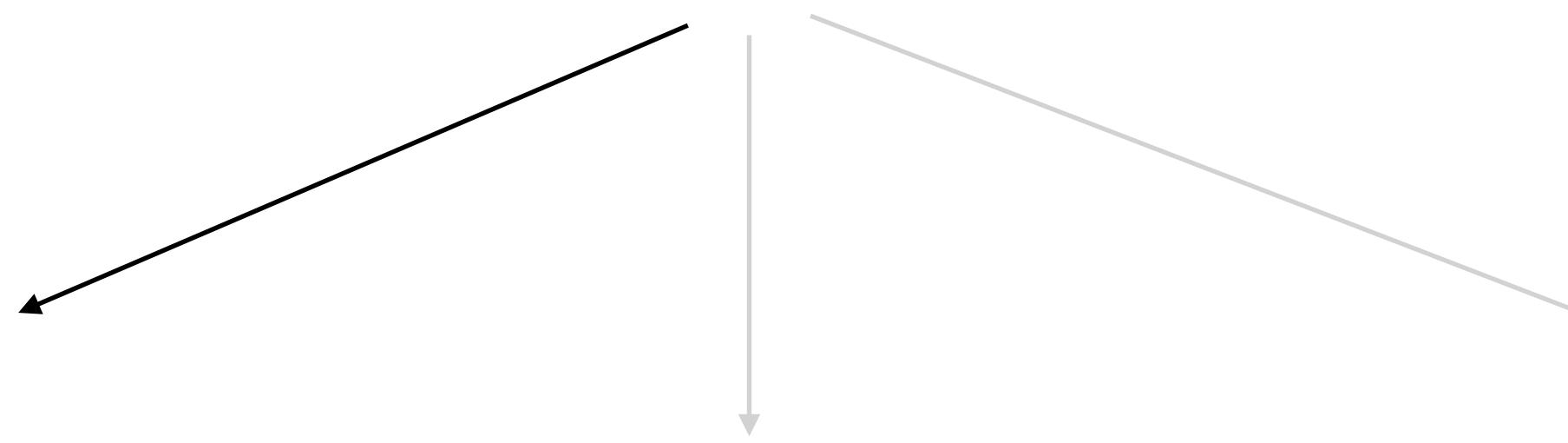
Declarative is preferred

Three Object Management Methods

**Imperative
Commands**

Imperative Object
Configuration

Declarative Object
Configuration



Imperative Commands

```
kubectl run nginx --image nginx
```

```
kubectl create deployment nginx --image nginx
```

No config file

Imperative: intent is in command

Pro:

- Simple

Cons:

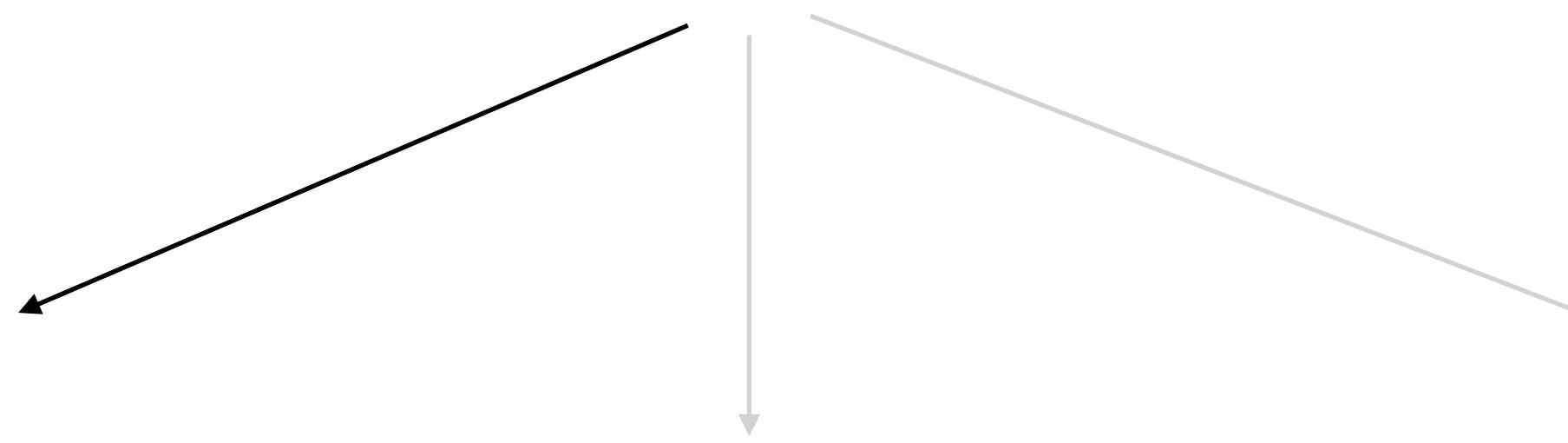
- No audit trail or review mechanism
- Can't reuse or use in template

Three Object Management Methods

**Imperative
Commands**

Imperative Object
Configuration

Declarative Object
Configuration

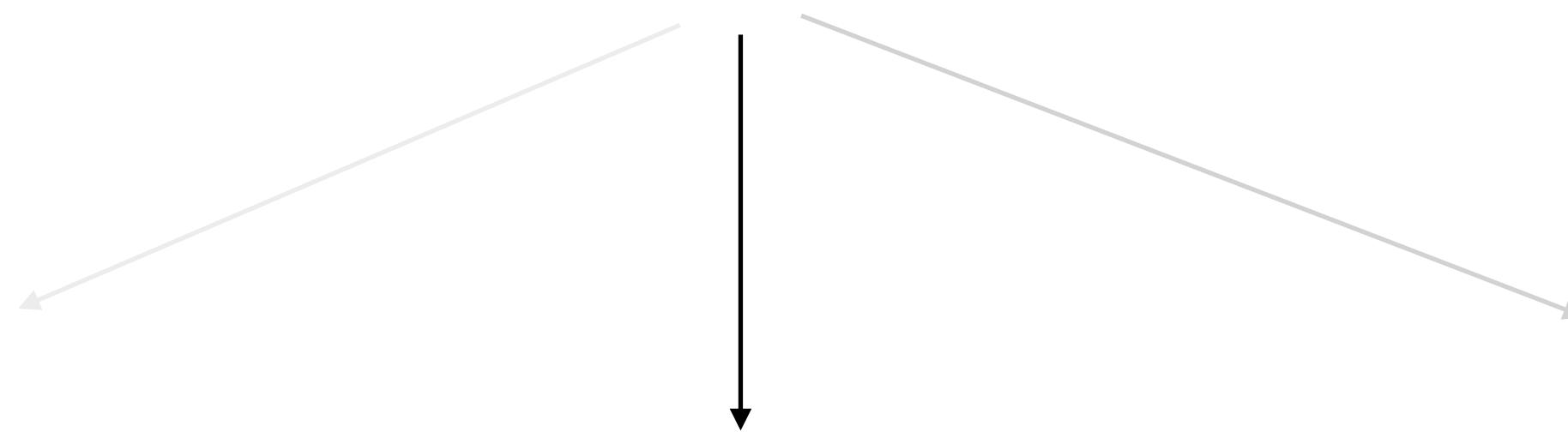


Three Object Management Methods

Imperative
Commands

Imperative Object
Configuration

Declarative Object
Configuration



Imperative Object Configuration

```
kubectl create -f nginx.yaml
```

```
kubectl delete -f nginx.yaml
```

```
kubectl replace -f nginx.yaml
```

Config file required

Still Imperative: intent is in command

Pros:

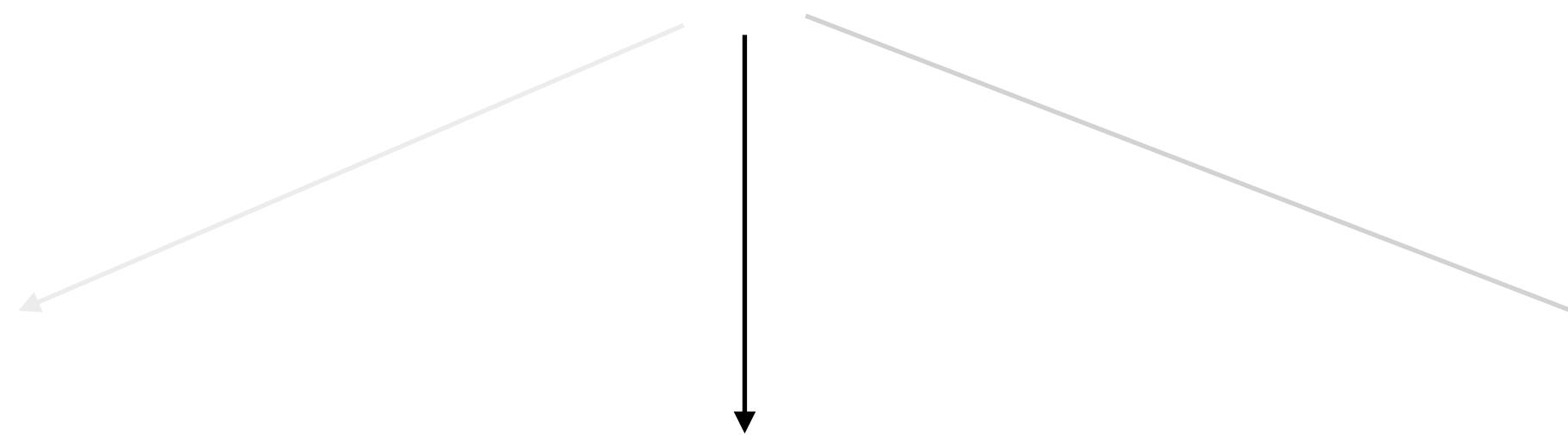
- Still simple
- Robust - files checked into repo
- One file for multiple operations!

Three Object Management Methods

Imperative
Commands

Imperative Object
Configuration

Declarative Object
Configuration

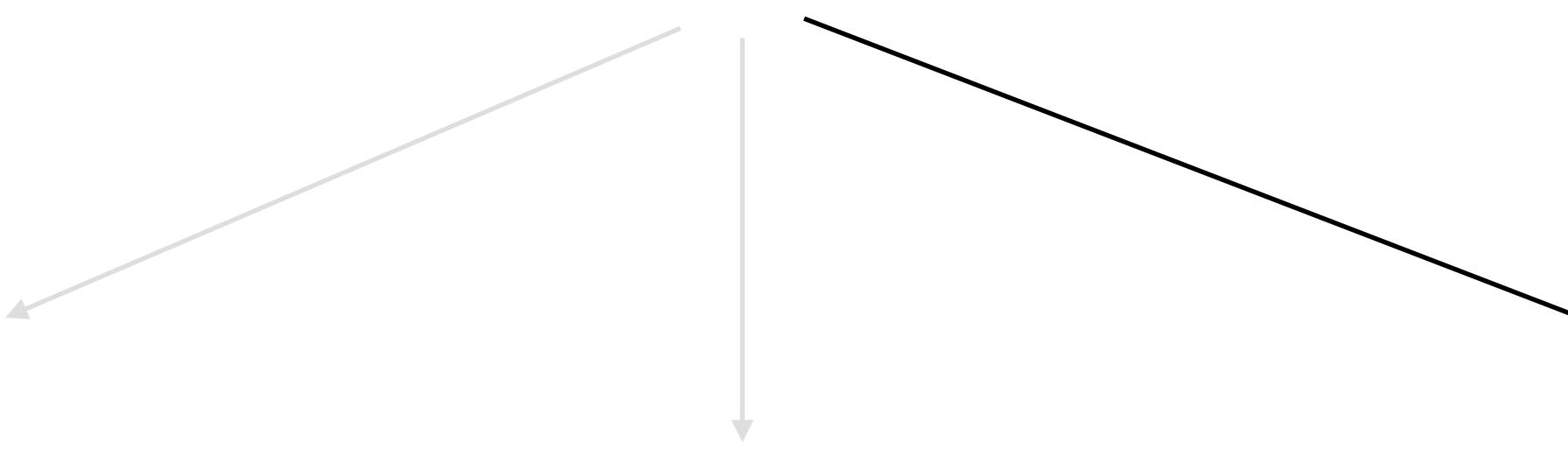


Three Object Management Methods

Imperative
Commands

Imperative Object
Configuration

Declarative Object
Configuration



Declarative Object Configuration

```
kubectl apply -f configs/
```

Config file(s) are all that is required

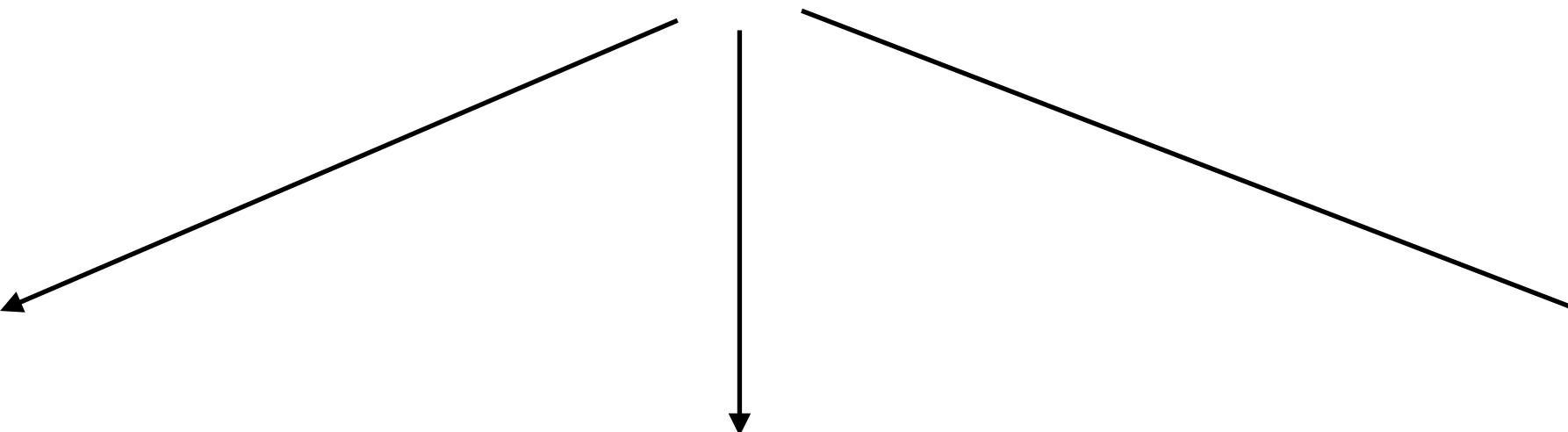
Declarative not imperative

Pros:

- Most robust - review, repos, audit trails
- K8S will automatically figure out intents
- Can specify multiple files/directories recursively

How Are Declarative Config Files Applied?

Three Object Management Methods



**Imperative
Commands**

No .yaml or config files

kubectl run ...

kubectl expose ...

kubectl autoscale ...

**Imperative Object
Configuration**

**kubectl + yaml or
config files used**

kubectl create -f config.yaml

kubectl replace -f config.yaml

kubectl delete -f config.yaml

**Declarative Object
Configuration**

**Only .yaml or config
files used**

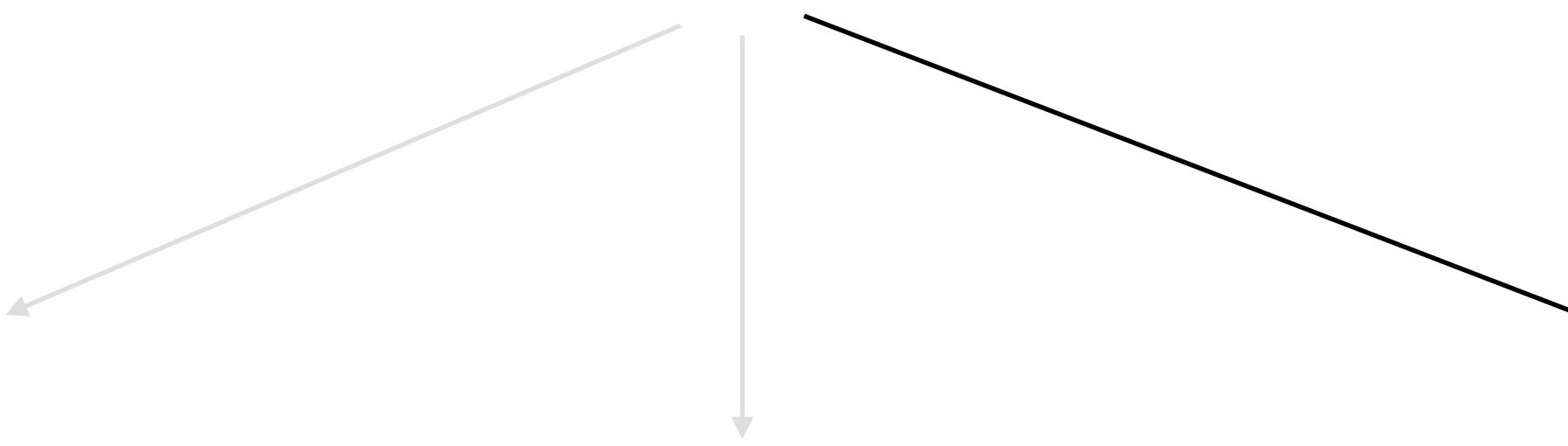
kubectl **apply** -f config.yaml

Three Object Management Methods

Imperative
Commands

Imperative Object
Configuration

Declarative Object
Configuration



Declarative Object Configuration

- **Live object configuration:** The live configuration values of an object, as observed by the Kubernetes cluster
- **Current object configuration file:** The config file we are applying in the current command
- **Last-applied object configuration file:** The last config file that was applied to the object

Merging Changes

- Primitive fields
 - String, int, boolean, images or replicas
 - Replace old state with Current object configuration file
- Map fields
 - Merge old state with Current object configuration file
- List field
 - Complex - varies by field

Three Object Management Methods

Imperative
Commands

Imperative Object
Configuration

Declarative Object
Configuration

- Live object configuration
- Current object configuration file
- Last-applied object configuration file

What are pros and cons of imperative and declarative object management?

Declarative Or Imperative?

Declarative

kubectl **apply** -f config.yaml

- Robust
- Track in repos, review
- Recursively apply to directories

Imperative

kubectl run ...

kubectl expose ...

kubectl autoscale ...

- Simple, intention is very clear
- Preferred for deletion

How Are Objects Named?

How Are Objects Named?

- Objects - persistent entities
 - pods, replicsets, services, volumes, nodes,...
 - information maintained in etcd
- Identified using
 - names: client-given
 - UIDs: system-generated

What Are Namespaces?

Namespaces

- Divide physical cluster into multiple virtual clusters
- Three pre-defined namespaces:
 - default: if none specified
 - kube-system: for internal kubernetes objects
 - kube-public: auto-readable by all users

Namespaces

- Name scopes: names need to be unique only inside a namespace
- Future version: namespace -> common access control
- Don't use namespaces for versioning
 - Just use labels instead

Objects Without Namespaces

- **Nodes**
- **PersistentVolumes**
- **Namespaces themselves:-)**

What are Labels?

Labels

- key/value pairs attached to objects
- metadata
- need not be unique (same label to multiple objects)
- No semantics for kubernetes (meaningful only to humans)

Labels

- Loose coupling via selectors
- Can be added
 - at creation time
 - after creation
- Multiple objects can have same label
- Same label can't repeat key though

How Do Label Selectors Provide Loose Coupling Of Pods?

Ephemeral Pod IP Addresses

- Containers expose ports in Pod spec
- But Pod IP addresses keep changing!
- For instance, when ReplicaSets or Deployments take pods up/down, IP addresses will change
- How is a client to know how to access the code in a container?

Services For Stable IP Addresses

- Service object solves this problem!
- Service = Logical set of backend pods + stable front-end
- Front-end: Static IP address + Port + DNS Name
- Back-end: Logical set of backend pods (label selector)

ClusterIP

- When service object created, ClusterIP is assigned
- Tied to service object through lifetime
- Independent of lifespan of any backend pod
- Any other pods can talk to ClusterIP and always access backend
- Service object also has a static port assigned to it

Pod Selector

Service Object

```
selector:  
  version:1.0  
  zone: south-asia
```

Pod 1

```
Labels  
{  
  version:1.0,  
  env: prod,  
  zone: south-asia  
}
```

Pod 2

```
Labels  
{  
  version:1.2,  
  env: dev,  
  zone: us-central  
}
```

Pod 3

```
Labels  
{  
  version:1.0,  
  env: prod,  
  zone: eur-central  
}
```

Pod Selector

Service Object

selector:

version:1.0

zone: south-asia

Pod 1

Labels

{

version:1.0,

env: prod,

zone: south-asia

}

Pod 2

Labels

{

version:1.2,

env: dev,

zone: us-central

}

Pod 3

Labels

{

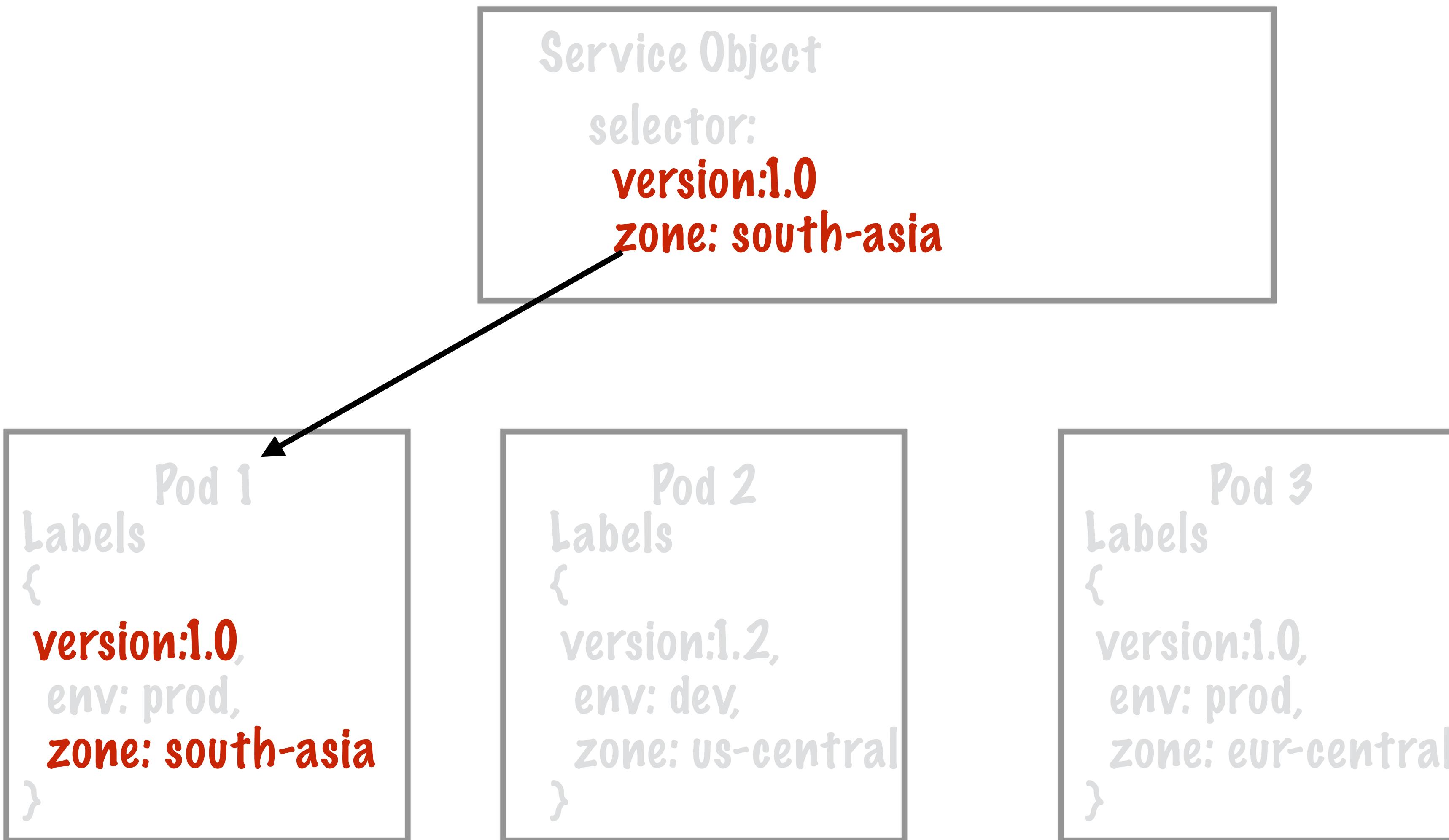
version:1.0,

env: prod,

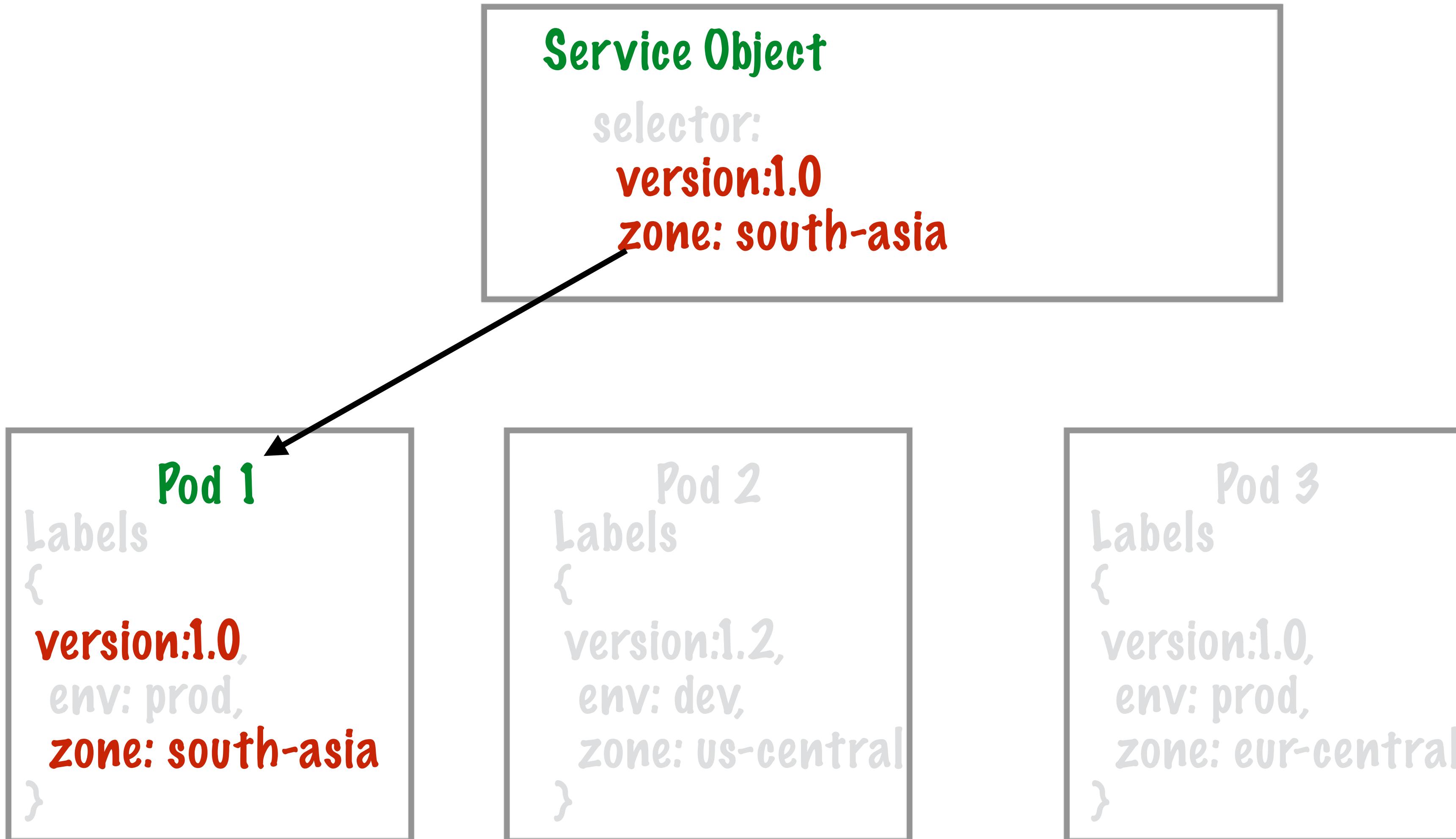
zone: eur-central

}

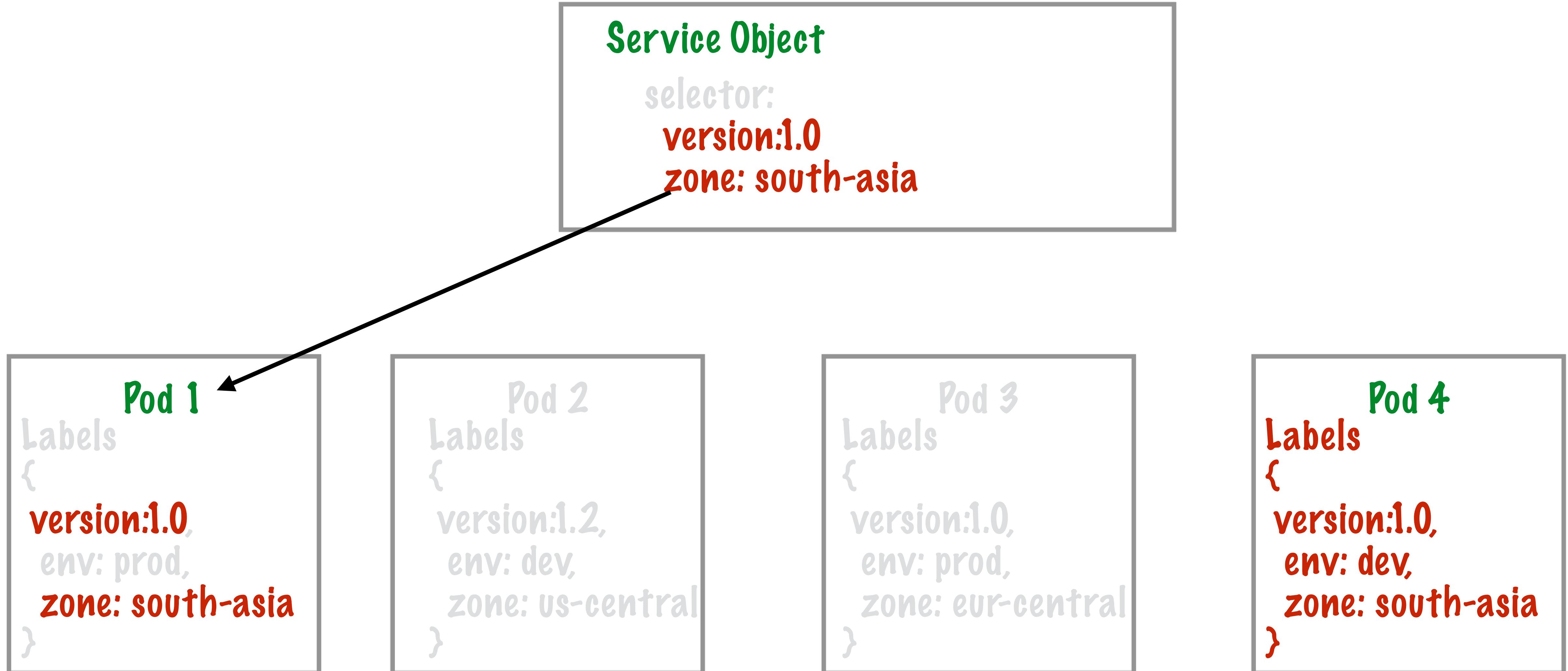
Pod Selector



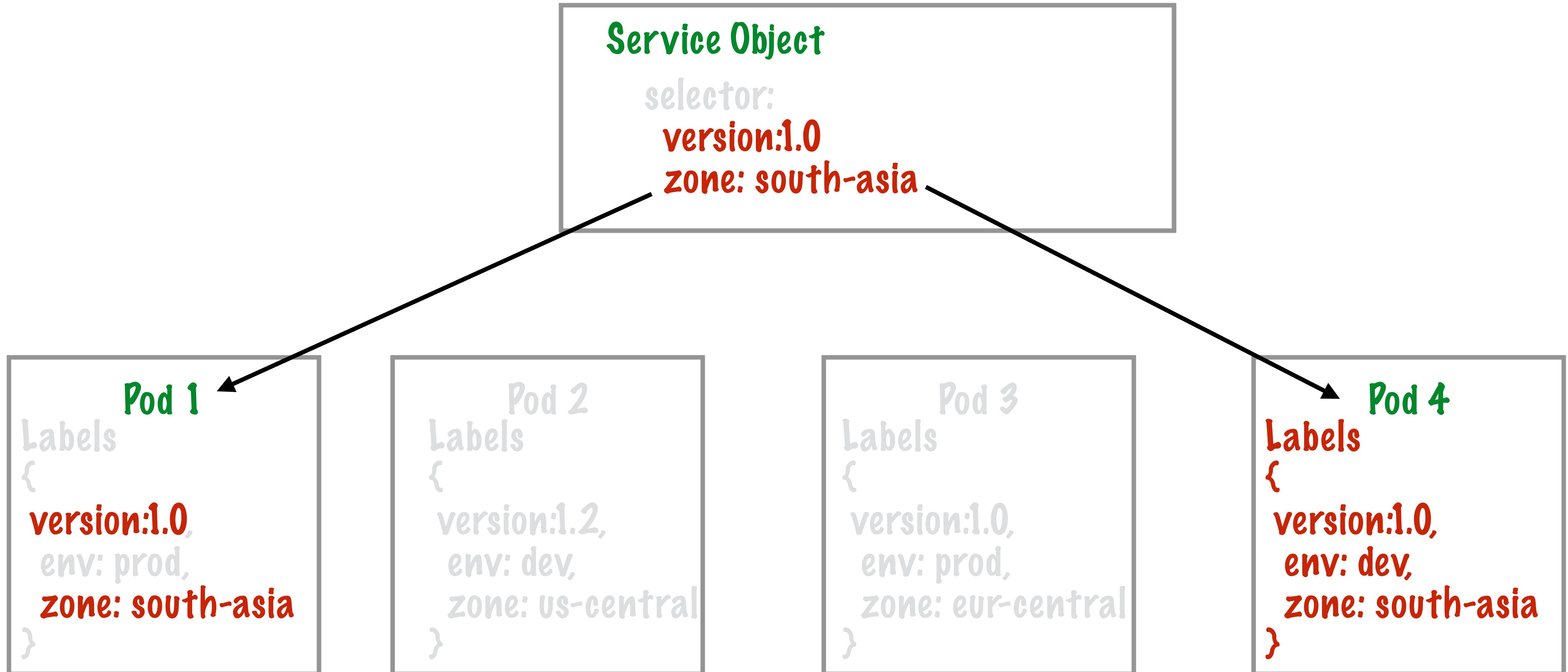
Pod Selector



Pod Selector



Pod Selector



Labels and Label Selectors

- Pods come up and go down
- Label selectors dynamically checked against pod labels
- Coupling is loose - not hardcoded

What are Annotations?

Annotations

- key/value pairs attached to objects
- metadata
- very similar to labels
- labels - usually used for identification and selection
- annotations - usually for metadata required by the object itself

Annotations

- Larger than labels
- Special characters
- Binaries, phone numbers...

What are Volumes?

Pods Limitations

- No auto-healing or scaling
- Pod crashes? Must be handled at higher level
 - ReplicaSet, Deployment, Service
- Ephemeral: Storage and file abstractions vanish on pod crash

Need For Volumes

- **Permanence:** Storage that lasts longer than lifetime of a pod
- **Shared State:** Multiple containers in a pod need to share state/files
- **Volumes:** Address both these needs

Need For Volumes

- **Permanence:** Storage that lasts longer than lifetime of
 - a container inside pod (true for all volumes)
 - a pod (only true for persistent volumes)
- **Shared State:** Multiple containers in a pod need to share state/files
- **Volumes:** Address both these needs

Volumes and Persistent Volumes

- **Volumes (in general): lifetime of abstraction = lifetime of pod**
 - Note that this is longer than lifetime of any container inside pod
- **Persistent Volumes: lifetime of abstraction independent of pod lifetime**

Using Volumes

- Define volume in pod spec
- Have each container 'mount' the volume
 - each container will mount independently
 - different paths in each container

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod
spec:
  containers:
    - name: test
      image: busybox
      volumeMounts:
        - name: config-vol
          mountPath: /etc/config
  volumes:
    - name: config-vol
      configMap:
        name: log-config
      items:
        - key: log_level
          path: log_level
```

Types of Volumes

- `awsElasticBlockStore`
- `azureDisk`
- `azureFile`
- `gcePersistentDisk`
- Many non-cloud-specific volume types as well

What are the types of Volumes?

Cloud Volumes

AWS

- awsElasticBlockStore

Azure

- azureDisk
- azureFile

GCP

- gcePersistentDisk

All these are persistent volumes
(lifetime extends beyond any pod)

Important Types Of Volumes

- configMap

- emptyDir

- gitRepo

- secret

- hostPath

What are Persistent Volumes?

Volumes and Persistent Volumes

- **Volumes (in general): lifetime of abstraction = lifetime of pod**
 - Note that this is longer than lifetime of any container inside pod
- **Persistent Volumes: lifetime of abstraction independent of pod lifetime**

Persistent Volume Objects

- Low-level objects, like nodes
- Two types of provisioning:
 - Static: Administrator pre-creates the volumes
 - Dynamic: Containers need to file a PersistentVolumeClaim

Cloud Clusters

AWS

- awsElasticBlockStore

Azure

- azureDisk
- azureFile

GCP

- gcePersistentDisk

All these are persistent volumes

What Volume Types Would Kubernetes Clusters On The Cloud Use?

Cloud Clusters

AWS

- awsElasticBlockStore

Azure

- azureDisk
- azureFile

GCP

- gcePersistentDisk

All these are persistent volumes

Cloud Clusters

AWS

- awsElasticBlockStore

Only on EC2 instances

Azure

- azureDisk
- azureFile

Only on Azure VMs

GCP

- gcePersistentDisk

Only on GCE VMs

What Are Some Important Types Of Volumes?

Important Types Of Volumes

- `emptyDir`

- `configMap`

- `gitRepo`

- `hostPath`

- `secret`

Important Types Of Volumes

- `emptyDir`

- `configMap`

- `gitRepo`

- `hostPath`

- `secret`

Important Types Of Volumes

- **emptyDir**

- configMap

- gitRepo

- hostPath

- secret

emptyDir

- Not persistent
- Created when pod is created on node
- Initially empty
- Share space/state across containers in same pod
- Containers can mount at different times

emptyDir

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
    - image: k8s.gcr.io/test-webserver
      name: test-container
    volumeMounts:
      - mountPath: /cache
        name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}
```

emptyDir volume named 'cache-volume'

emptyDir

- When pod removed/crashes, data lost
- When container crashes, data remains
- Use cases: Scratch space, checkpointing...

Important Types Of Volumes

- **emptyDir**

- configMap

- gitRepo

- hostPath

- secret

Important Types Of Volumes

- `emptyDir`

- `configMap`

- `gitRepo`

- `hostPath`

- `secret`

hostPath

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
    - image: k8s.gcr.io/test-webserver
      name: test-container
    volumeMounts:
      - mountPath: /test-pd
        name: test-volume
  volumes:
    - name: test-volume
      hostPath:
        # directory location on host
        path: /data
        # this field is optional
        type: Directory
```

hostPath volume to
directory '/data' on node

hostPath

- Mount file/directory from node filesystem into pod
- Uncommon - pods should be independent of nodes
- Makes pod-node coupling tight
- Use cases: Access docker internals, running cAdvisor
- Block devices or sockets on host

Important Types Of Volumes

- `emptyDir`

- `configMap`

- `gitRepo`

- `hostPath`

- `secret`

Important Types Of Volumes

- *emptyDir*

- *configMap*

- **gitRepo**

- *hostPath*

- *secret*

gitRepo

```
apiVersion: v1
kind: Pod
metadata:
  name: server
spec:
  containers:
  - image: nginx
    name: nginx
  volumeMounts:
  - mountPath: /mypath
    name: git-volume
  volumes:
  - name: git-volume
    gitRepo:
      repository: "git@somewhere:me/my-git-repository.git"
      revision: "22f1d8406d464b0c0874075539c1f2e96c253775"
```

Clone this git repo to our
volume

Important Types Of Volumes

- *emptyDir*

- *configMap*

- **gitRepo**

- *hostPath*

- *secret*

Important Types Of Volumes

- *emptyDir*

- **configMap**

- *gitRepo*

- *hostPath*

- *secret*

configMap

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-pod
spec:
  containers:
    - name: test
      image: busybox
      volumeMounts:
        - name: config-vol
          mountPath: /etc/config
  volumes:
    - name: config-vol
      configMap:
        name: log-config
        items:
          - key: log_level
            path: log_level
```

Mount data from a
ConfigMap object

configMap

- configMap volume mounts data from ConfigMap object
- ConfigMap objects define key-value pairs
- ConfigMap objects inject parameters into pods
- Two main usecases:
 - Providing config information for apps running inside pods
 - Specifying config information for control plane (controllers)

Important Types Of Volumes

- *emptyDir*

- **configMap**

- *gitRepo*

- *hostPath*

- *secret*

Important Types Of Volumes

- *emptyDir*
- *configMap*
- *gitRepo*
- *hostPath*
- **secret**

secret

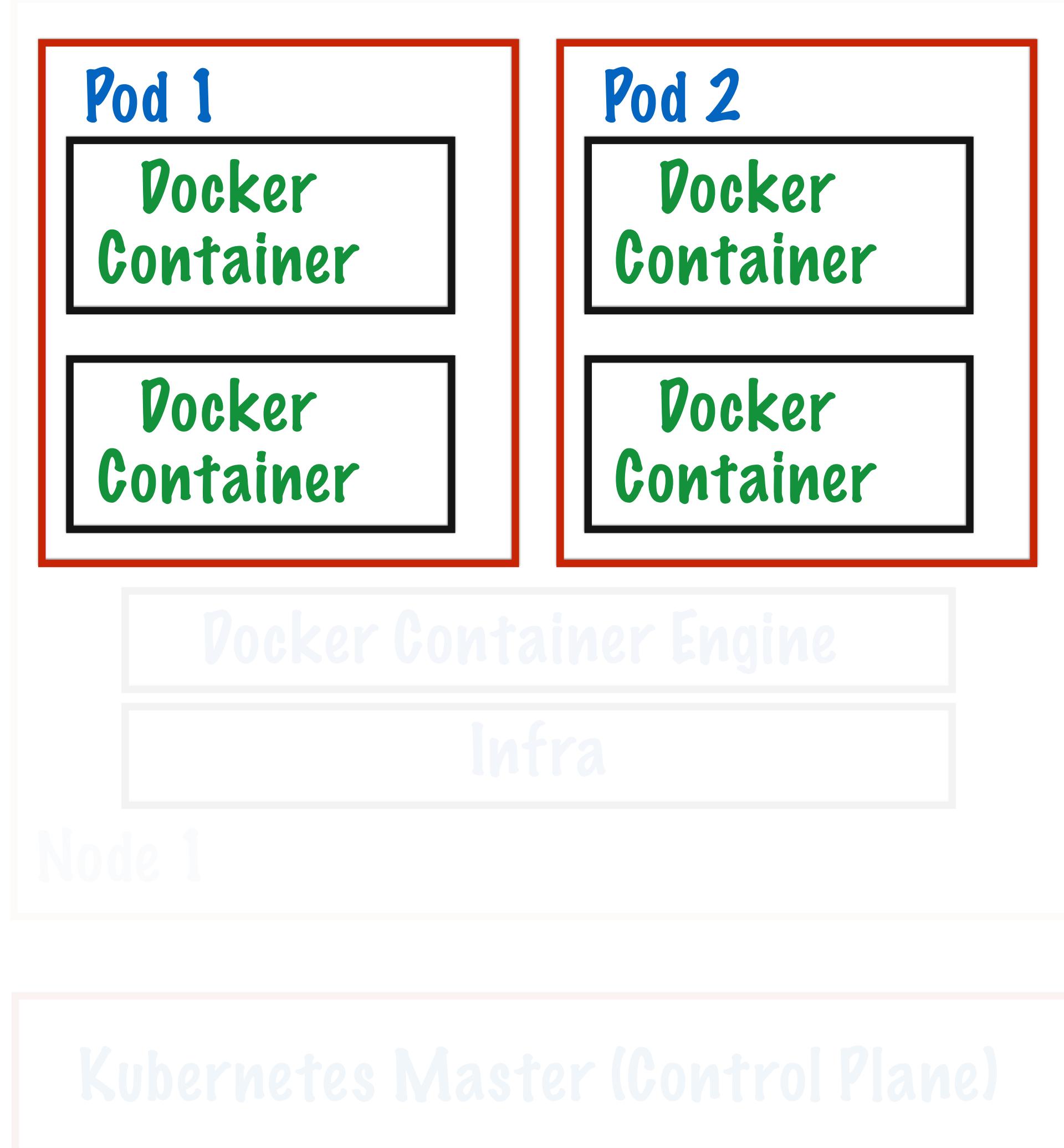
- Pass sensitive information to pods
- First create secret so it is stored in control plane
 - `kubectl create secret`
- Mount that secret as a volume so that it is available inside pod
- Secret is stored in RAM storage (not written to persistent disk)

Important Types Of Volumes

- *emptyDir*
- *configMap*
- *gitRepo*
- *hostPath*
- **secret**

Where Do Containers In A Pod Come From?

Pods on Kubernetes Nodes

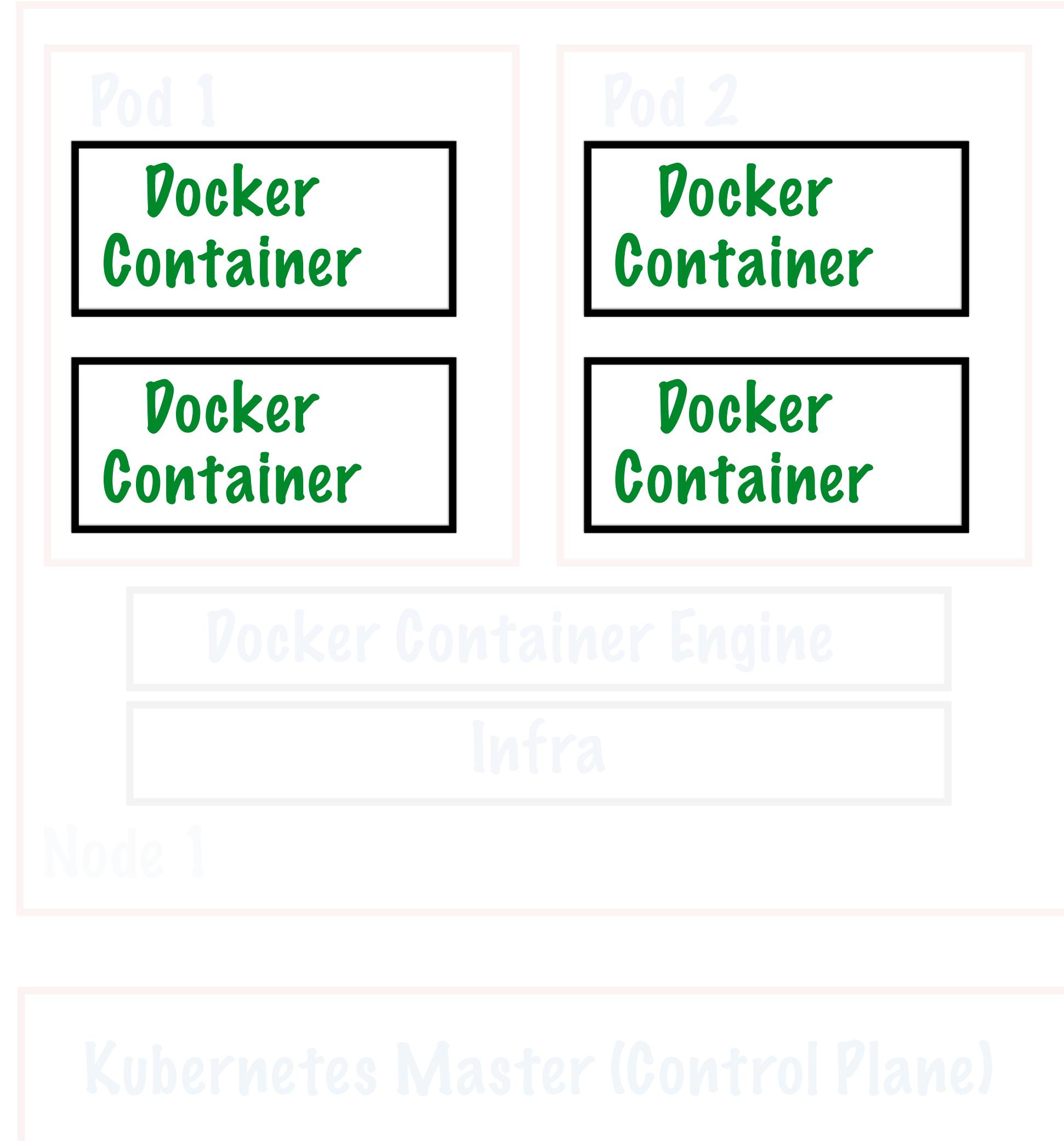


Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Containers In Pod



Container Image should already be in some registry

Private registries can be used as well

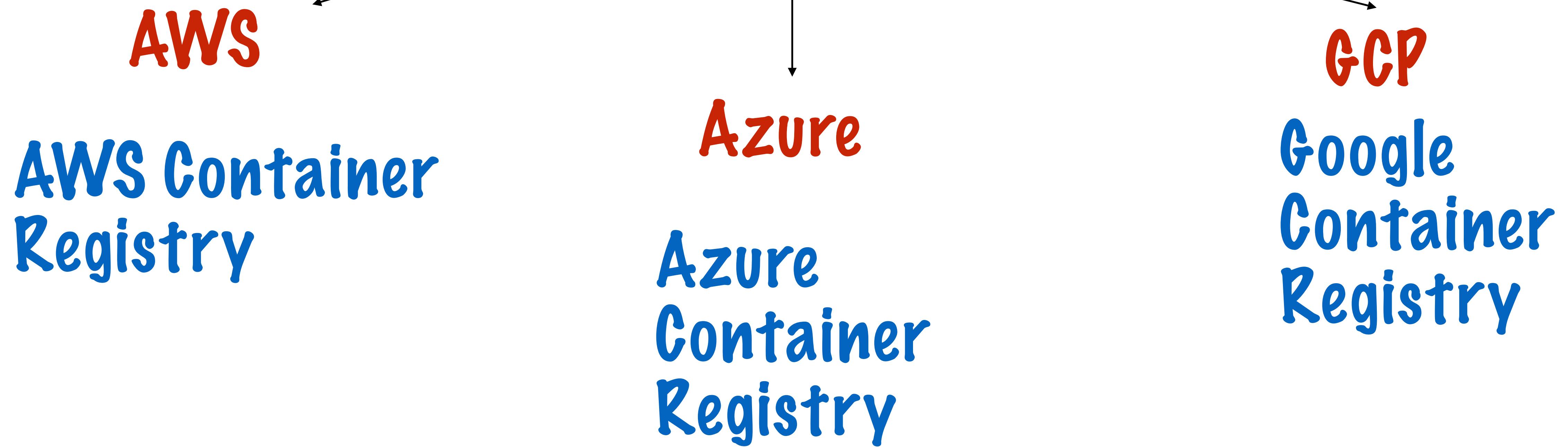
Image Tag To Specify Container

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
  livenessProbe:
    httpGet:
      # when "host" is not defined, "PodIP" will be used
      # host: my-host
      # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
      # scheme: HTTPS
      path: /healthz
      port: 8080
      httpHeaders:
      - name: X-Custom-Header
        value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

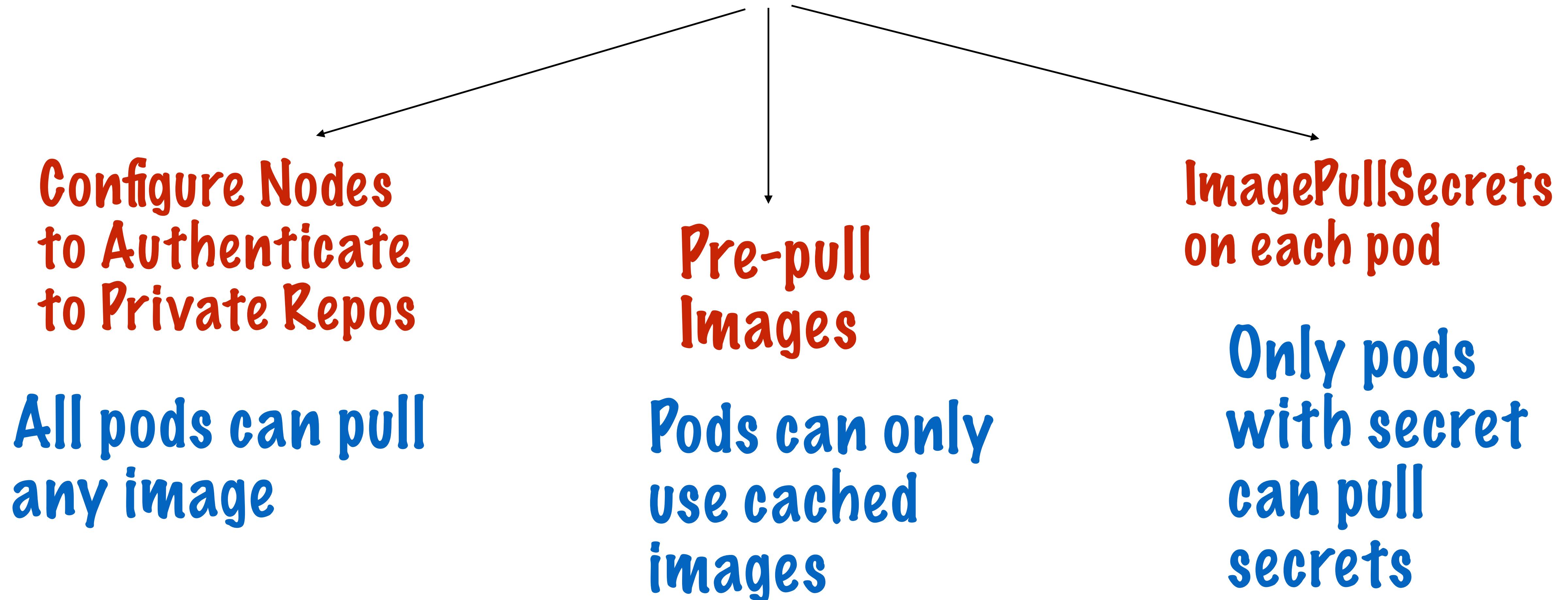
Which container image(s)?

Available on which port?

Cloud Clusters

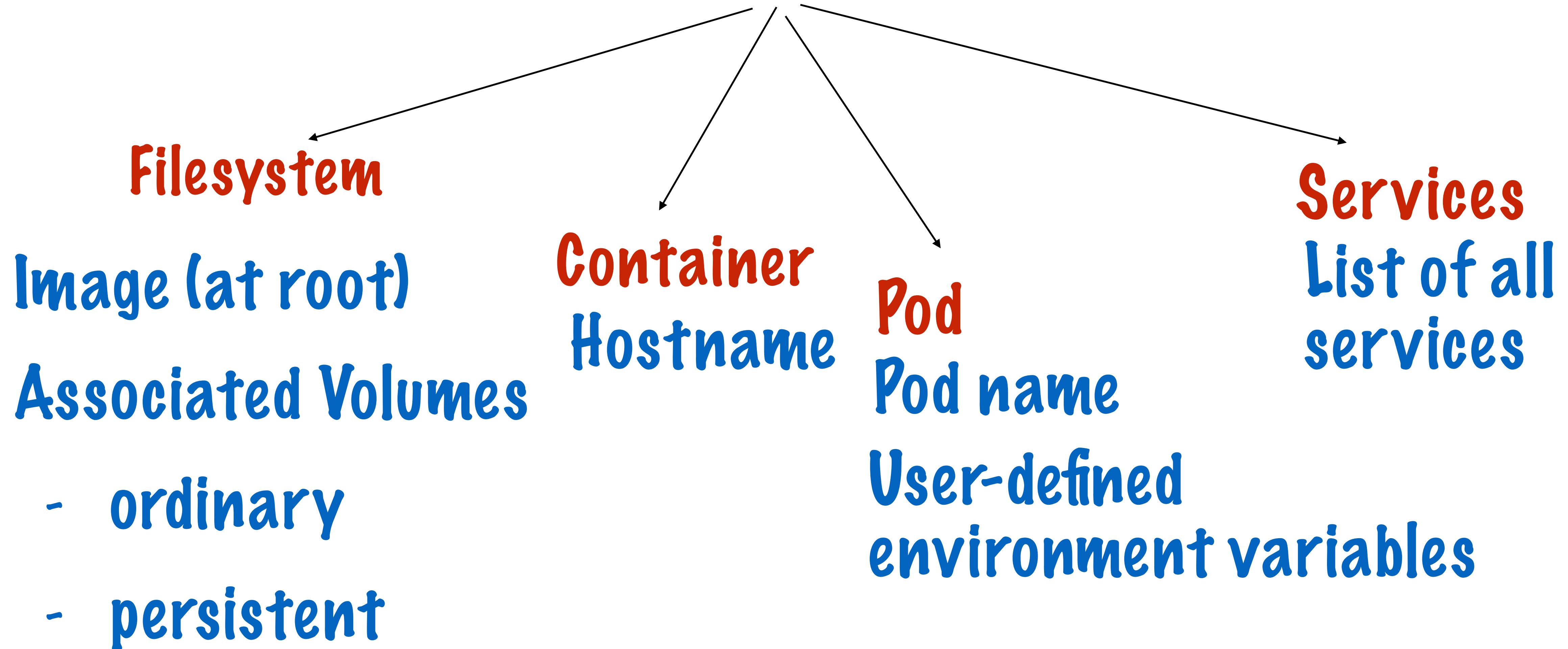


Non-Cloud Clusters

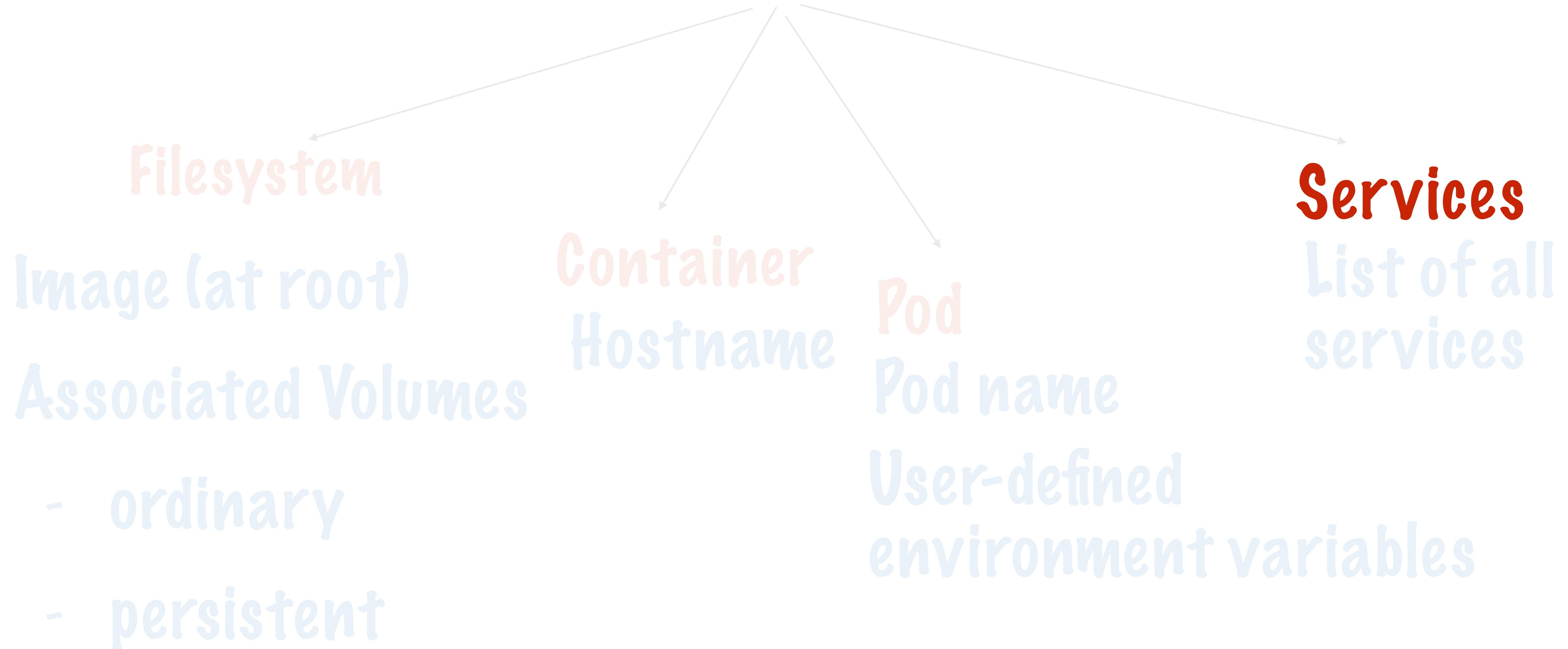


What Environment Do Containers See?

Cloud Clusters



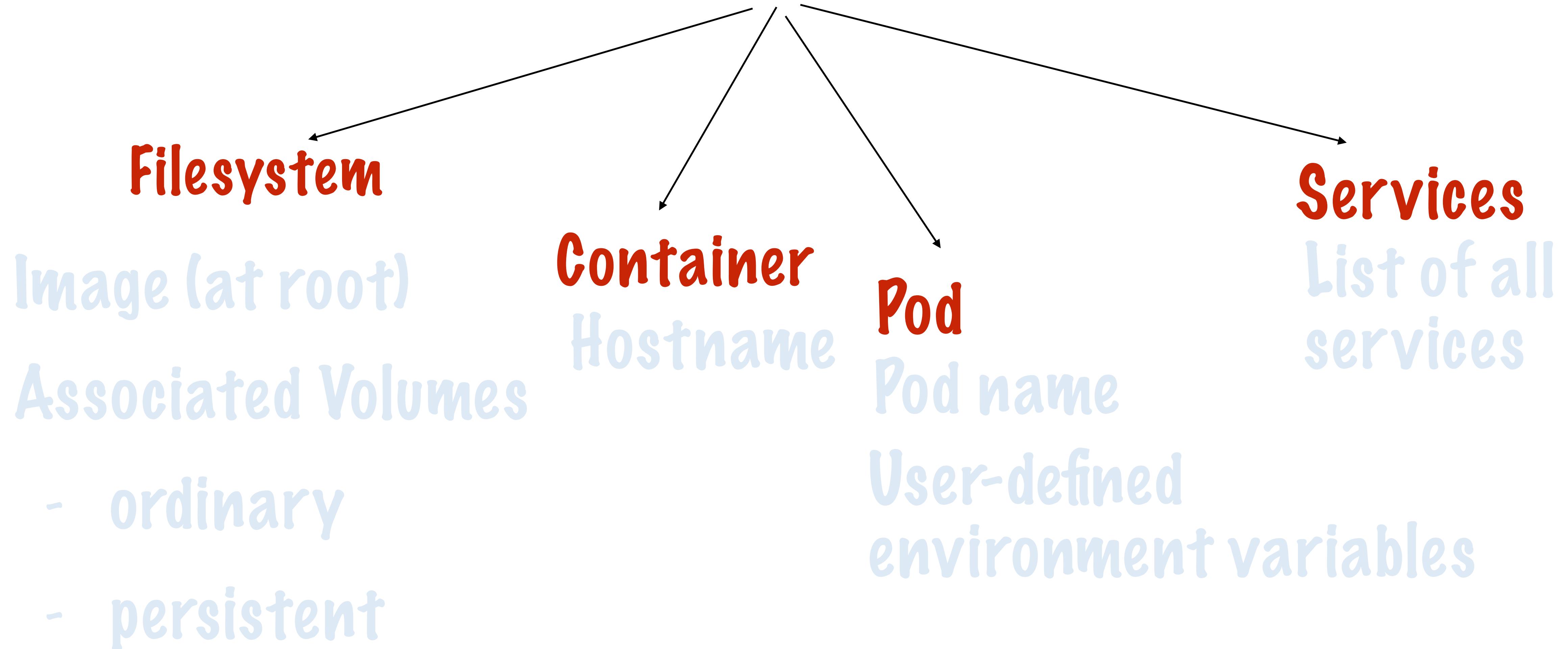
Cloud Clusters



Services For Stable IP Addresses

- Service object - load balancer
- Service = Logical set of backend pods + stable front-end
- Front-end: Static ClusterIP address + Port + DNS Name
- Back-end: Logical set of backend pods (label selector)

Cloud Clusters



Can Containers React To Events In Their Own Lifecycle?

Container Lifecycle Hooks

PostStart

Called immediately
after container
created

No parameters

PreStop

Immediately before
container terminates

Blocking - must
complete before
container can be
deleted

Container Lifecycle Hooks

- Hook handlers
 - Exec
 - HTTP
- Delivery is atleast-once
- Hook may be invoked multiple times

How Can Pods Be Assigned To Specific Nodes?

Kubernetes Master



One or more nodes designated as master

Several kubernetes processes run on master

Multi-master for high-availability

kube-apiserver

Kubernetes Master (Control Plane)

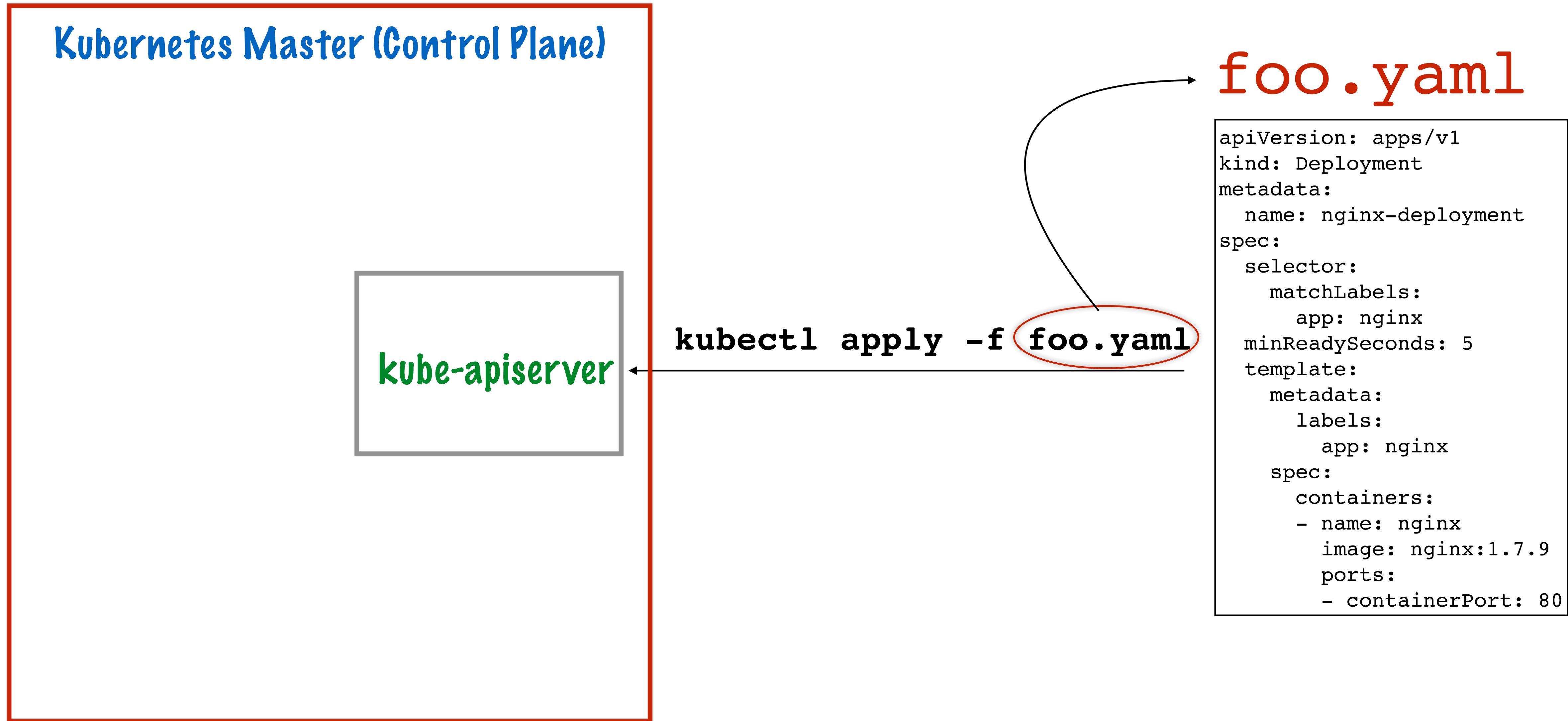
kube-apiserver

Communicates with user

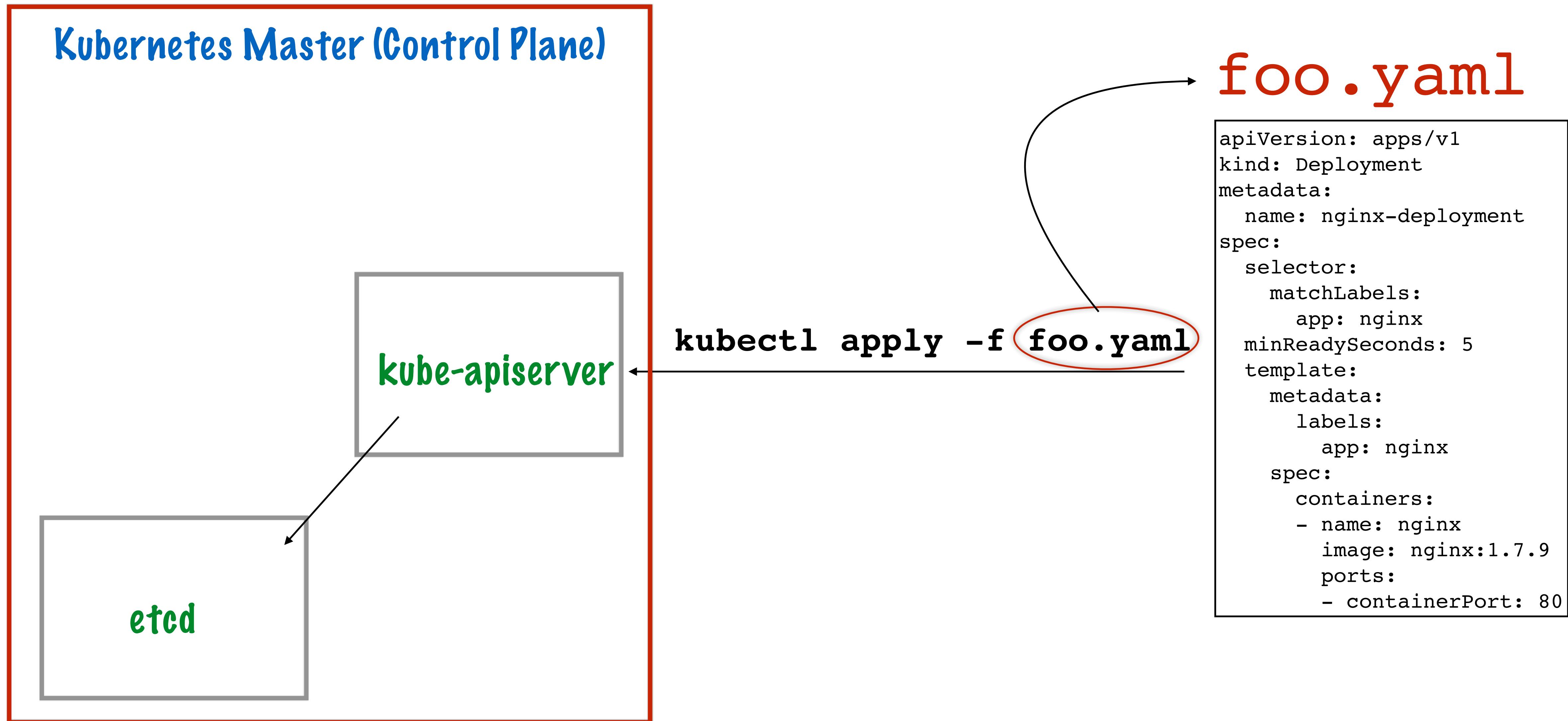
RESTful API end-points

Manifest yaml files are accepted by apiserver

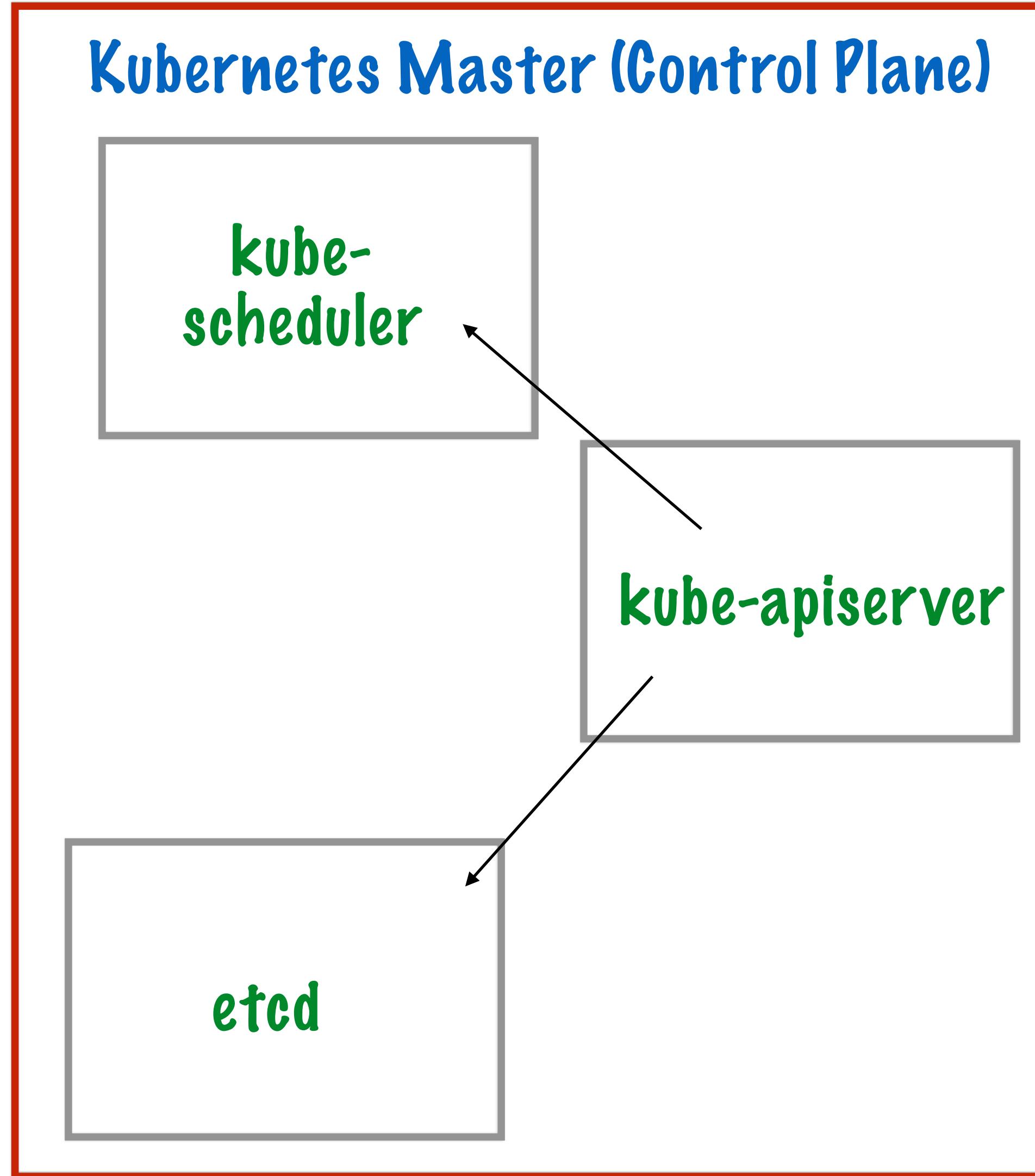
kube-apiserver



Cluster Store for Metadata



kube-scheduler

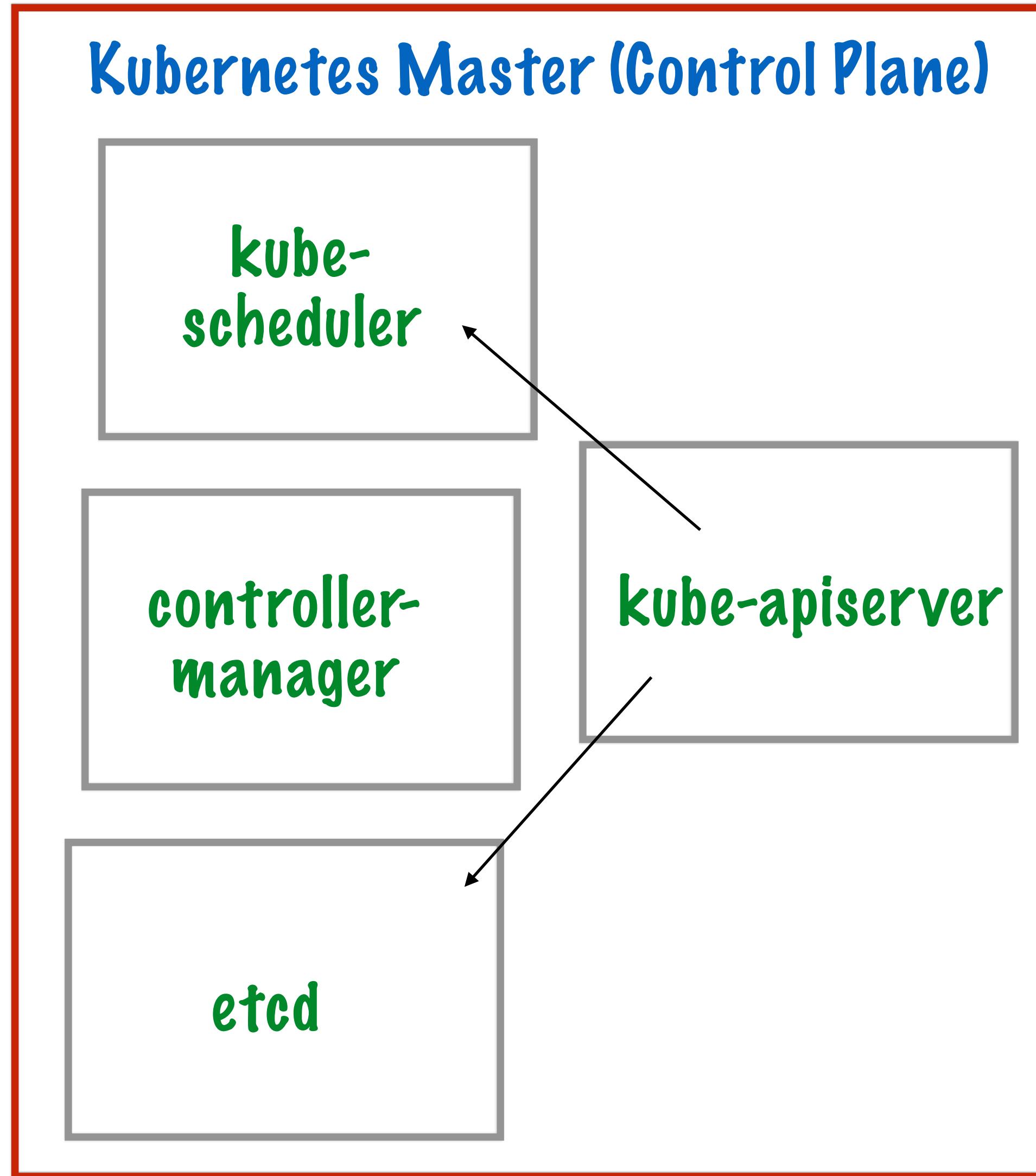


Handle pod creation and management

kube-scheduler match/assign nodes to pods

Complex - affinities, taints, tolerations, ...

controller-manager



Different master processes

Actual state <-> Desired State

cloud-controller-manager

kube-controller-manager

Pod-Node Assignments

- Handled by `kube-scheduler`
 - Quite Smart
- Granular usecases:
 - Specific hardware: SSD required by pod
 - Colocate pods on same node: they communicate a lot
 - High-availability: force pods to be on different nodes

Pod-Node Assignments

nodeSelector

- Simple
- Tag nodes with labels
- Add `nodeSelector` to pod template
- Pods will only reside on nodes that are selected by `nodeSelector`
- Simple but crude - hard constraint

- Nodes have predefined labels
 - Hostname, zone, OS, instance type...

Affinity and Anti-Affinity

Node Affinity

- Steer pod to node
- Can be 'soft'
- Only affinity (for anti-affinity use taints)

Pod Affinity

- Steer pods towards or away from pods
- Affinity: pods close to each other
- Anti-Affinity: pods away from each other

How Are Taints And Tolerations Used To Make Pods Avoid Certain Nodes?

Pod-Node Assignments

nodeSelector

- Simple
- Tag nodes with labels
- Add `nodeSelector` to pod template
- Pods will only reside on nodes that are selected by `nodeSelector`
- Simple but crude - hard constraint

- Nodes have predefined labels
 - Hostname, zone, OS, instance type...

Affinity and Anti-Affinity

Node Affinity

- Steer pod to node
- Can be 'soft'
- Only affinity (for anti-affinity use taints)

Pod Affinity

- Steer pods towards or away from pods
- Affinity: pods close to each other
- Anti-Affinity: pods away from each other

Pod-Node Assignments

nodeSelector

- Simple
- Tag nodes
- Add nodeSelector to pod template
- Pods will only reside on nodes that are selected by nodeSelector
- Simple but crude - hard constraint

- Nodes have predefined labels
 - Hostname, zone, OS, instance type...

Affinity and Anti-Affinity

Node Affinity

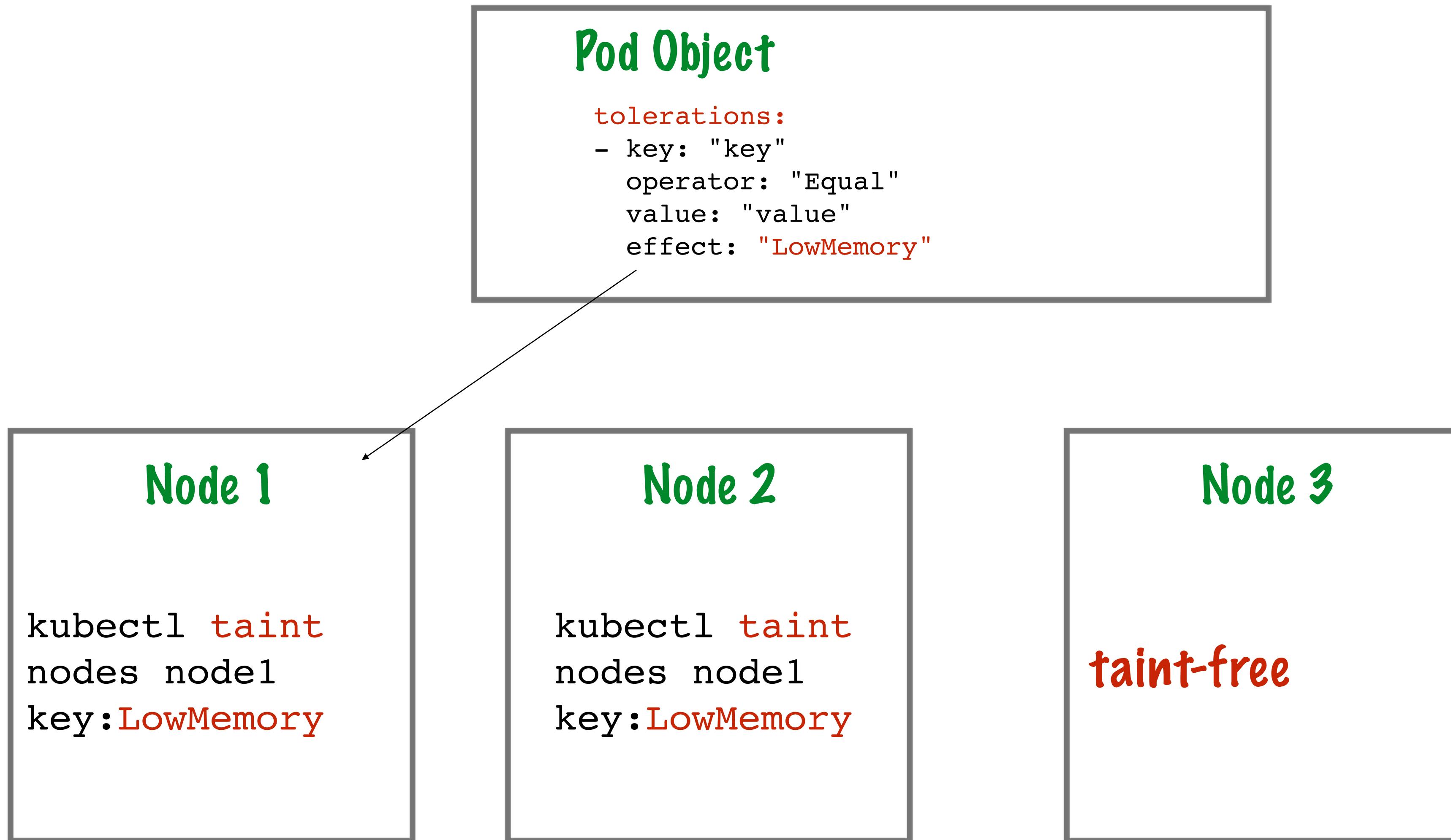
- Steer pods to node
- Can be 'soft'
- Only affinity (for anti-affinity use taints)

Pod Affinity

- Steer pods towards or away from pods
- Affinity: pods close to each other
- Anti-Affinity: pods away from each other

use Taints and Tolerations

Taints and Tolerations



Usecases for Taints and Tolerations

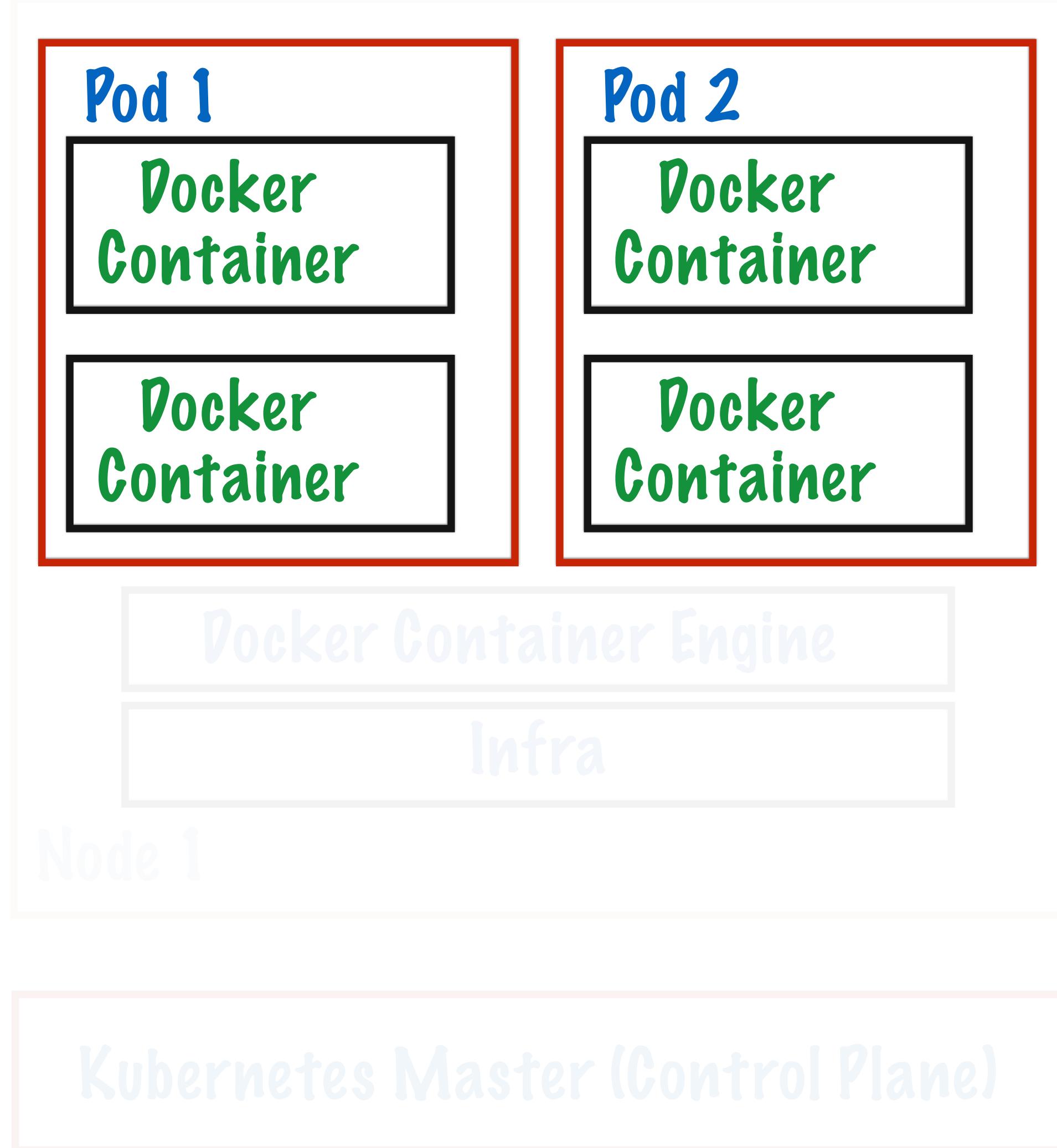
- Dedicated nodes for certain users
 - Taint subset of nodes
 - Only add tolerations to pods of those users
- Nodes with special hardware
 - Taint nodes with GPUs
 - Add toleration only pods running ML jobs

Taints Based On Node Condition

- New feature - in Alpha in v1.8
- Taints added by node controller
 - `node.kubernetes.io/memory-pressure`
 - `node.kubernetes.io/disk-pressure`
 - `node.kubernetes.io/out-of-disk`
- This will happen if flag set on nodes
 - `TaintNodesByCondition=true`

What Are Init Containers?

Pods on Kubernetes Nodes



Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

These are app-containers (actually host app)

Init Containers

App Containers

- Usual containers that host compute
- Pod may or may not restart after crash

Init Containers

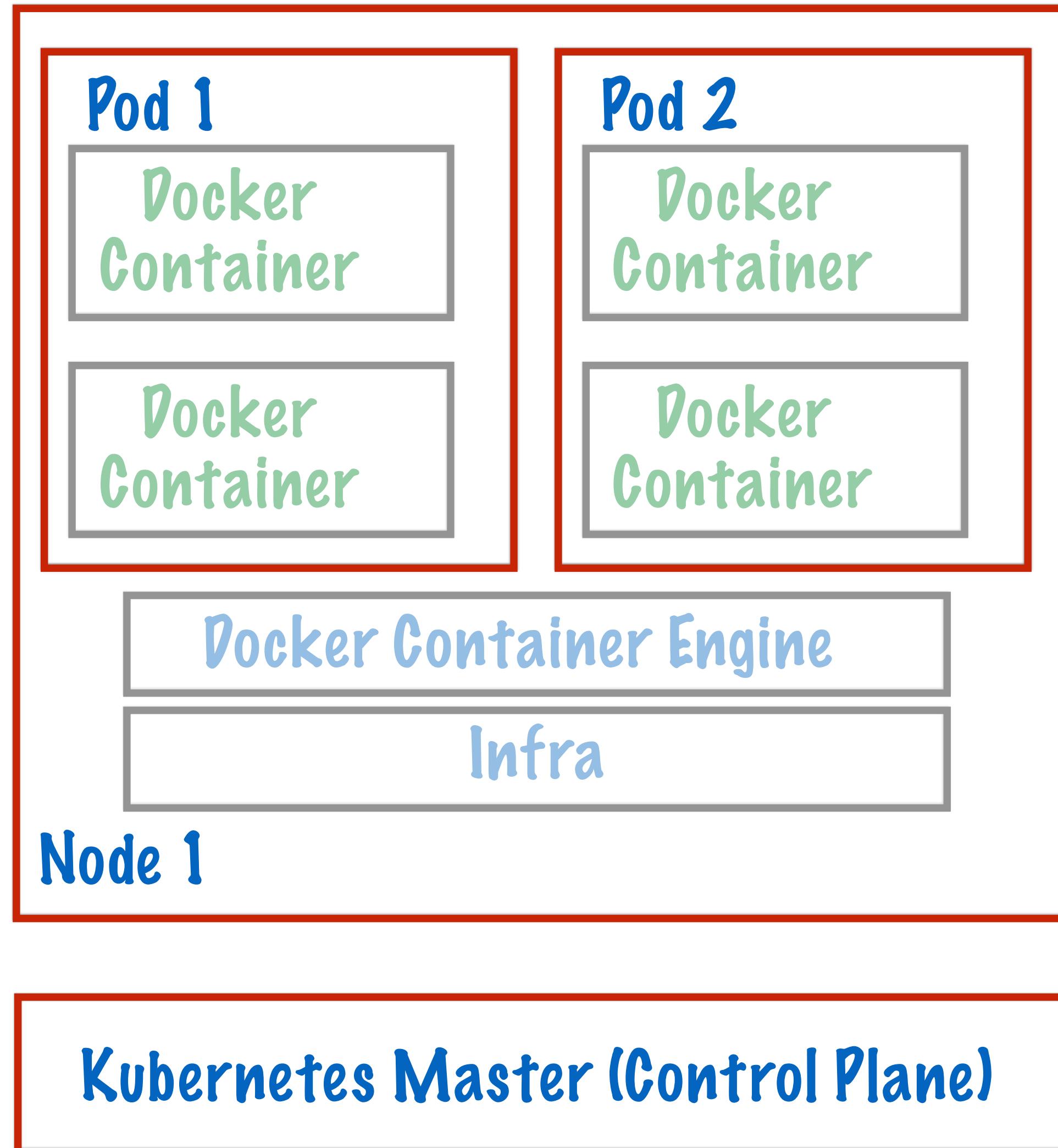
- Run before app containers
- Always run-to-completion
- Run serially (each only starts after previous one finishes)

Usecases Of Init Containers

- Run utilities that should run before app container
- Different namespace/isolation from app containers
- Security reasons
- Include utilities or setup (git clone, register app)
- Block or delay start of app container

What Is The Lifecycle Of A Pod?

Pods on Kubernetes Nodes



Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

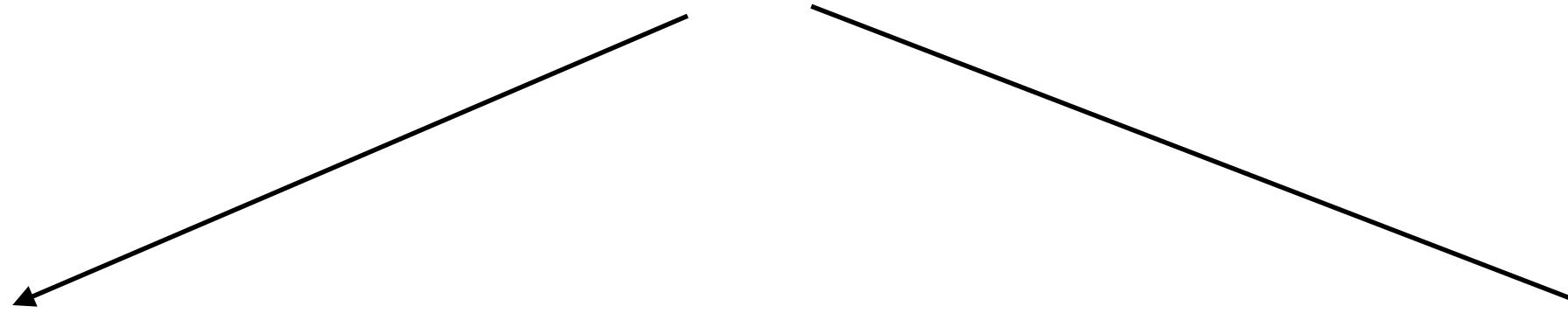
Atomic Nature Of Pods

Container Deployment

Containers within pod
are deployed in an all-
or-nothing manner

Node Association

Entire pod is hosted
on the same node



Pods Limitations

- No auto-healing or scaling
- Pod crashes? Must be handled at higher level
 - ReplicaSet, Deployment, Service
- Ephemeral: IP addresses are ephemeral

PodStatus.Phase

- Pending
- Running
- Succeeded
- Failed
- Unknown

PodStatus.Phase

- Pending: Request accepted, but not yet fully created
- Running
- Succeeded
- Failed
- Unknown

PodStatus.Phase

- Pending: Request accepted, but not yet fully created
- **Running: Pod bound to node, all containers started**
- Succeeded
- Failed
- Unknown

PodStatus.Phase

- Pending: Request accepted, but not yet fully created
- Running: Pod bound to node, all containers started
- **Succeeded: All containers terminated successfully (will not be restarted)**
- Failed
- Unknown

PodStatus.Phase

- Pending: Request accepted, but not yet fully created
- Running: Pod bound to node, all containers started
- Succeeded: All containers terminated successfully (will not be restarted)
- Failed: All containers have terminated, and at least one failed
- Unknown

PodStatus.Phase

- Pending: Request accepted, but not yet fully created
- Running: Pod bound to node, all containers started
- Succeeded: All containers terminated successfully (will not be restarted)
- Failed: All containers have terminated, and at least one failed
- Unknown: Pod status could not be queried - host error likely

PodStatus.Phase

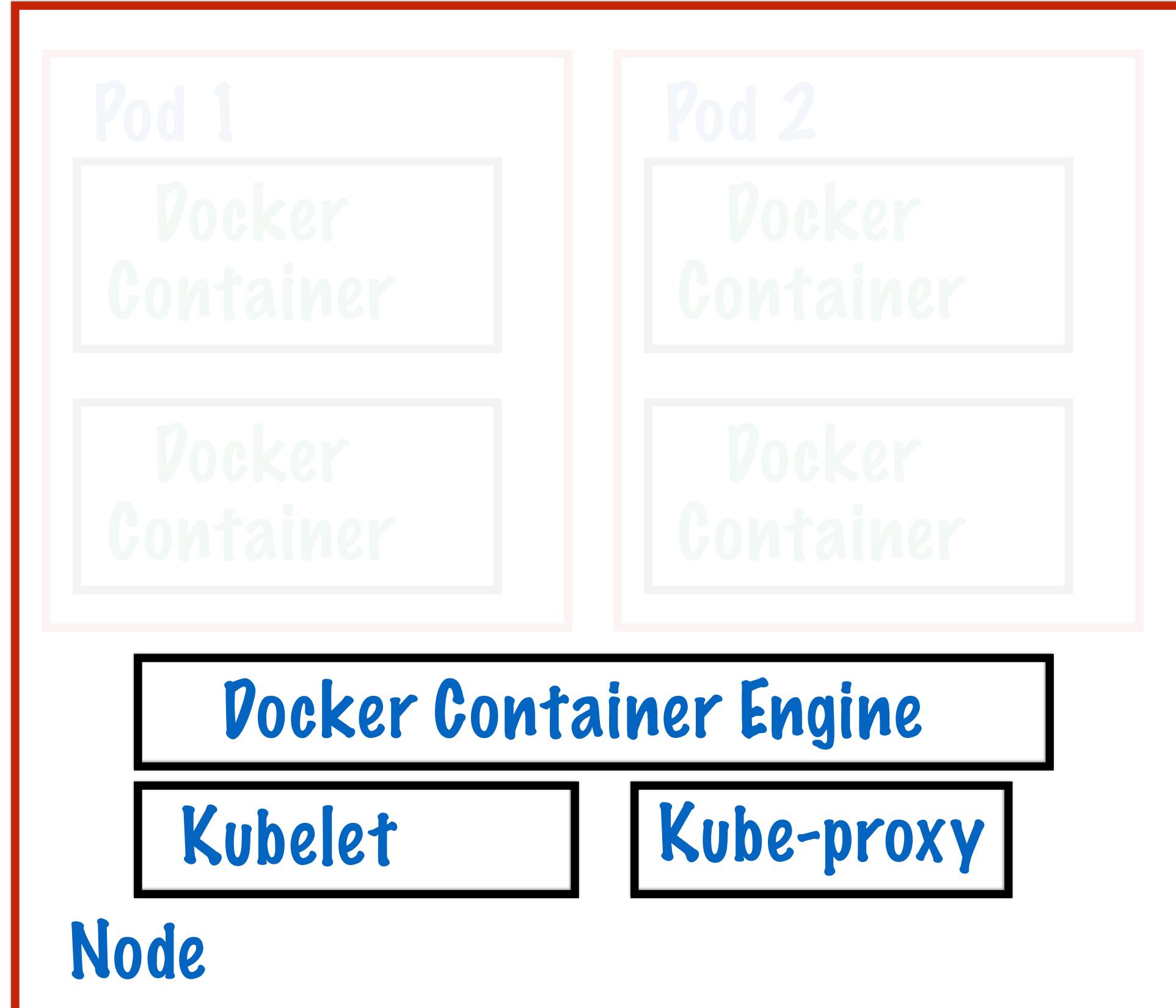
- Pending
- Running
- Succeeded
- Failed
- Unknown

Restart Policy For Containers In A Pod

- Always (default)
- On-failure
- Never

How Does Kubernetes Know Pod Status?

Kubernetes Node (Minion)



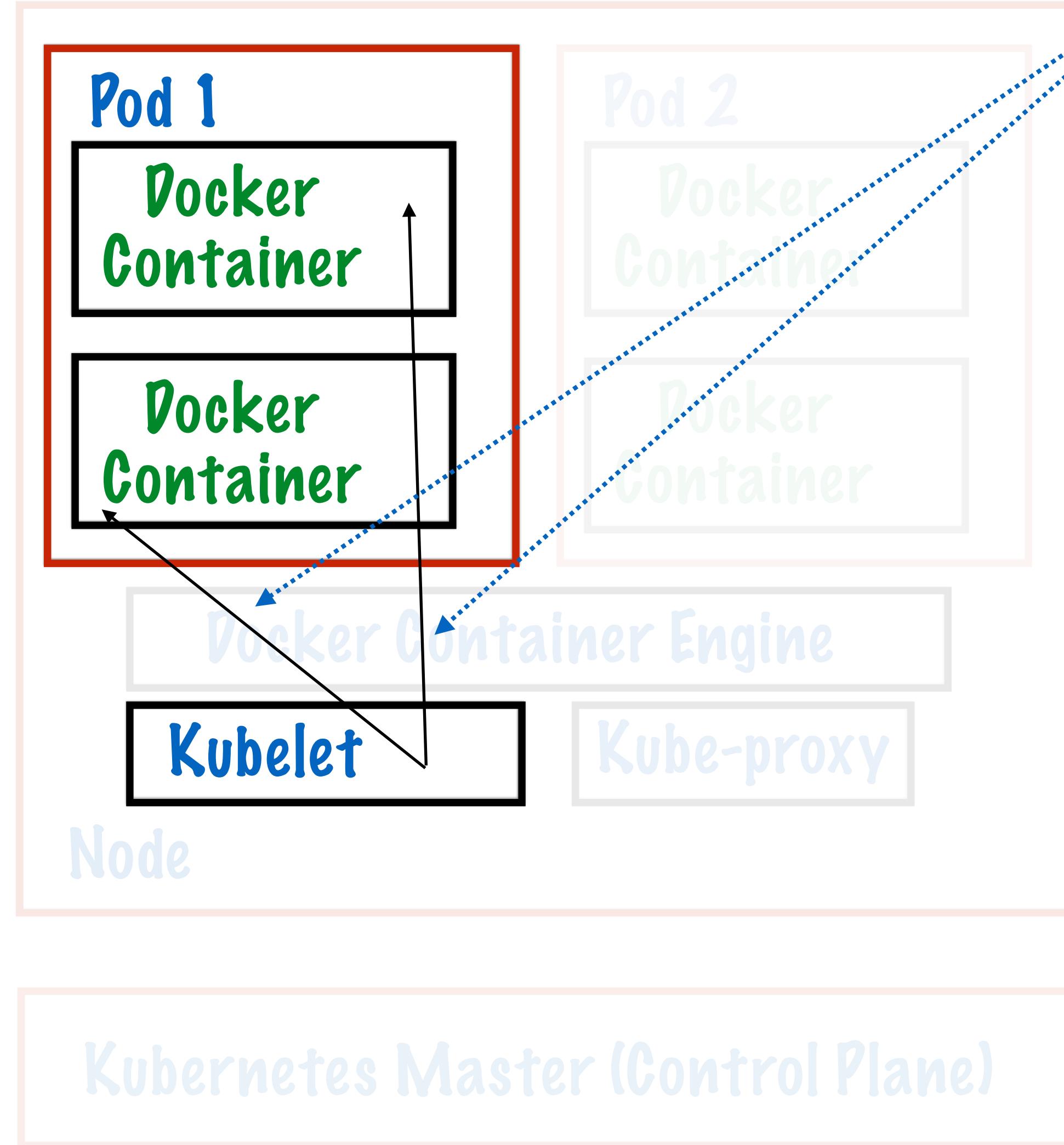
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Probes



Kubelet sends probes to containers

All succeeded? Pod status = Succeeded

Any failed? Pod status = Failed

Any running? Pod status = Running

Additional Types Of Probes

Liveness Probes

- Failed? Kubelet assumes container dead
- Restart policy will kick in

Readiness Probes

- Ready to service requests?
- Failed? Endpoint object will remove pod from services

Usecases

Liveness Probes

Usecase: Kill and restart if probe fails

1. Add liveness probe
2. Specify restart policy of Always or On-Failure

Readiness Probes

Usecase: Send traffic only after probe succeeds?

1. Pod goes live
2. But will only accept traffic after readiness probe succeeds

Usecase: container that takes itself down

What Are Pod Presets?

Pod Preset

- Injecting additional runtime into a Pod at creation time
- Label selectors-Specify the pods

How Do Pod Priorities Work?

Pod Priorities

- In alpha in v1.9, added v1.8
- Pod priorities can be set
- Scheduler might pre-empt or evict pods

Setting Pod Priorities

Step 1: Create PriorityClass Object

```
apiVersion: scheduling.k8s.io/v1alpha1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

Step 2: Reference from Pod Spec

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority
```

Pod Priorities

Scheduling Order

High-priority pod can
‘jump the queue’

Preemption

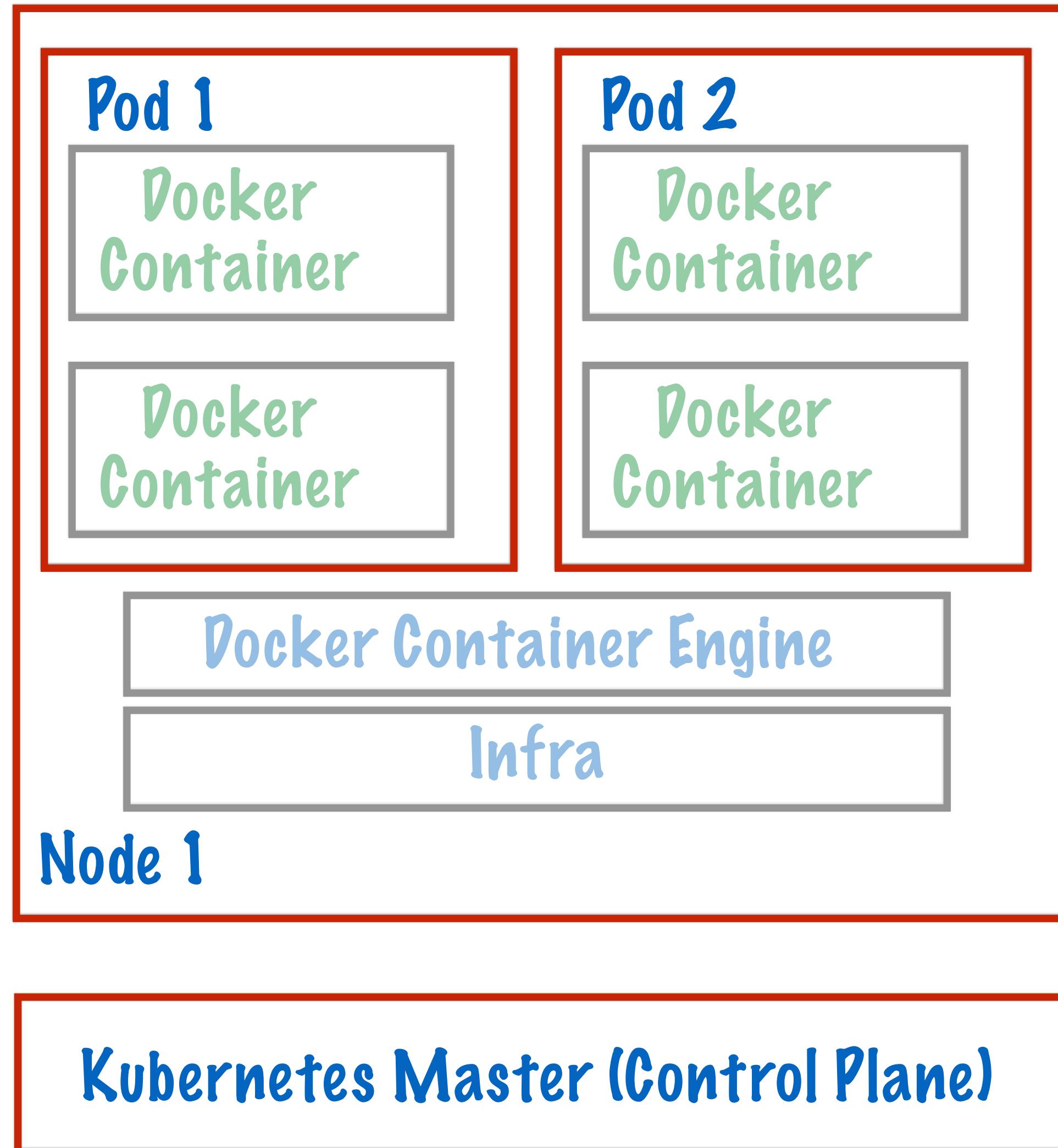
Low-priority pod maybe
pre-empted to make way

(If no node currently available
to run high-priority pod)

Preempted pod gets a
graceful termination period

What are Controllers?

Pods on Kubernetes Nodes



Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes Nodes



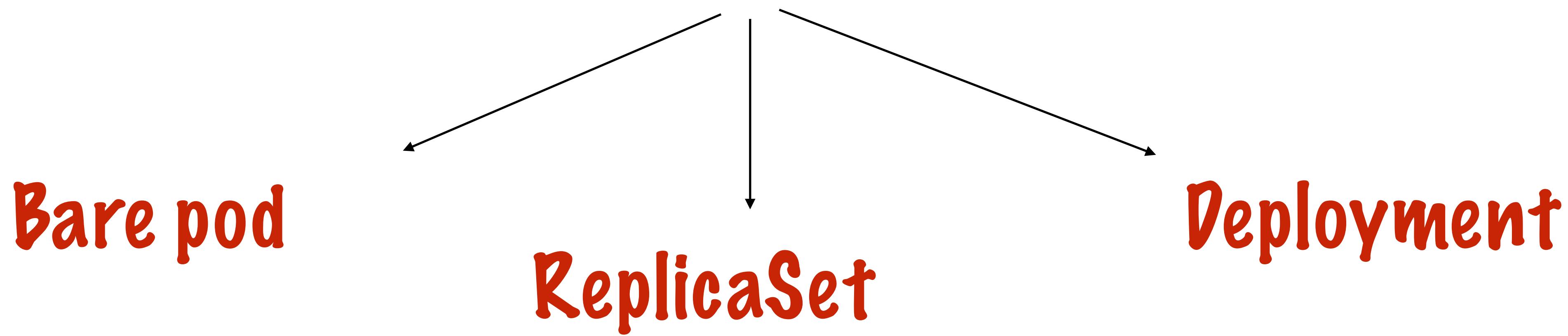
Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes

Pod Creation Requests



Pods are usually not created directly

Higher level abstractions preferred

Pods Limitations

- No auto-healing or scaling
- Pod crashes? Must be handled at higher level
 - ReplicaSet, Deployment, Service
- Ephemeral: IP addresses are ephemeral

Container -> Pod

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
  livenessProbe:
    httpGet:
      # when "host" is not defined, "PodIP" will be used
      # host: my-host
      # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
      # scheme: HTTPS
      path: /healthz
      port: 8080
      httpHeaders:
      - name: X-Custom-Header
        value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

Which container imag(s)?

Available on which port?

Pod -> ReplicaSet

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

Pod Template

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicaset...
  # modify it accordingly
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
    - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and uncomment the
          # line below.
          # value: env
      ports:
      - containerPort: 80
```

Number of replicas

ReplicaSet → Deployment

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
template:
  metadata:
    labels:
      app: guestbook
      tier: frontend
  spec:
    containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
          - name: GET_HOSTS_FROM
            value: dns
            # If your cluster config does not include a dns service, then to
            # instead access environment variables to find service host
            # info, comment out the 'value: dns' line above, and uncomment the
            # line below.
            # value: env
        ports:
          - containerPort: 80
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Replica Template

Higher Level Kubernetes Objects

- Pod
 - Containers inside pod template
- ReplicaSet:
 - pod template
 - number of replicas
 - self-healing and scaling
- Deployment:
 - Contains spec of ReplicaSet within it
 - Versioning
 - Fast rollback
 - Advanced deployments

Kubernetes Master



One or more nodes designated as master

Several kubernetes processes run on master

Multi-master for high-availability

kube-apiserver

Kubernetes Master (Control Plane)

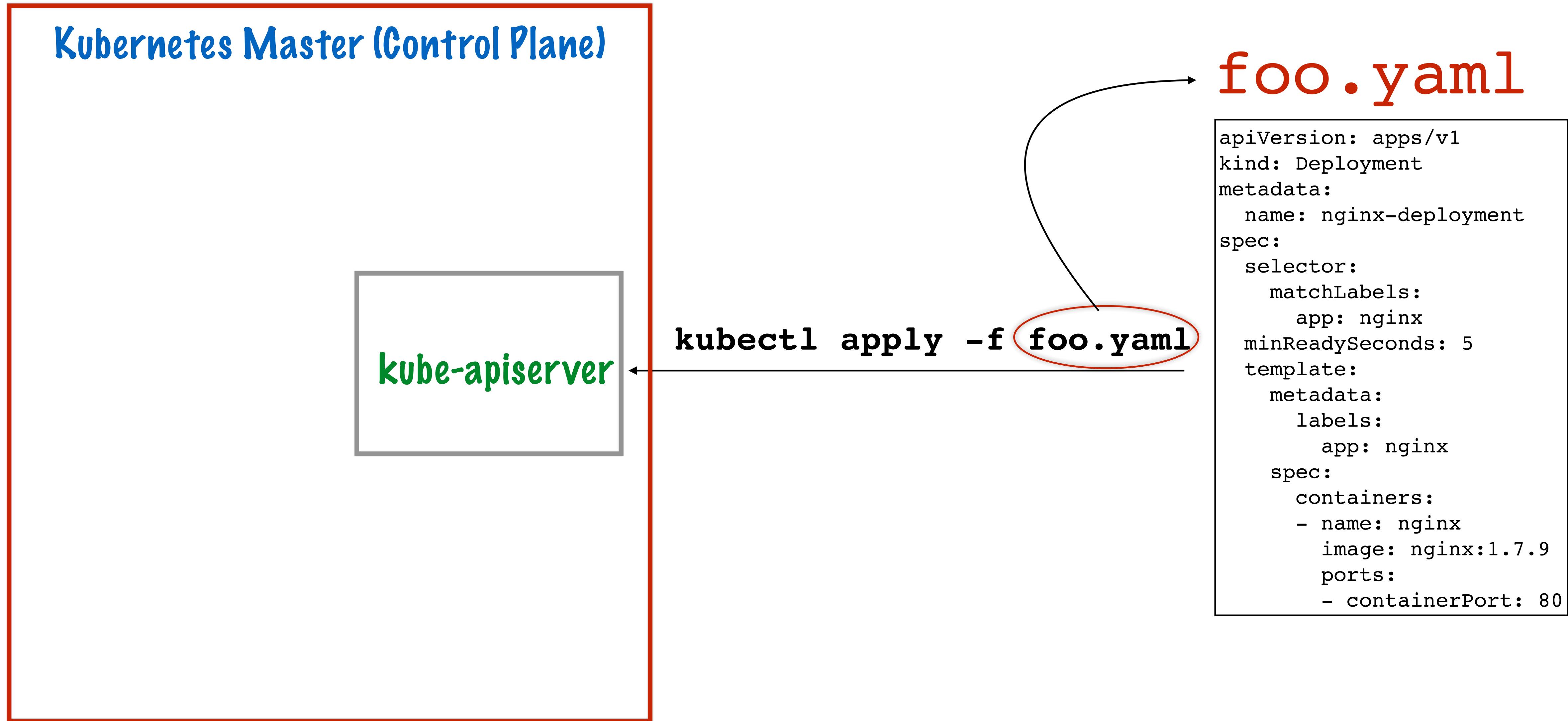
kube-apiserver

Communicates with user

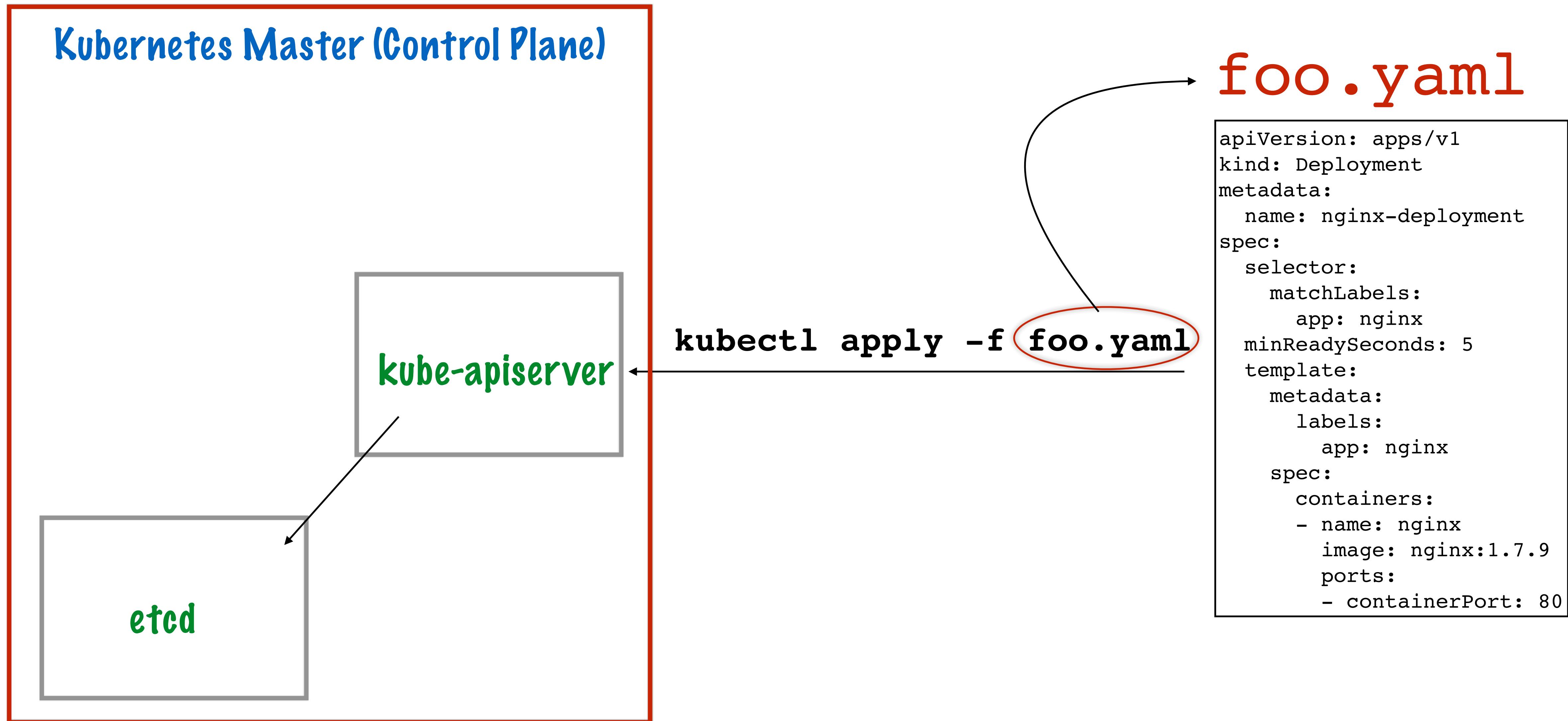
RESTful API end-points

Manifest yaml files are accepted by apiserver

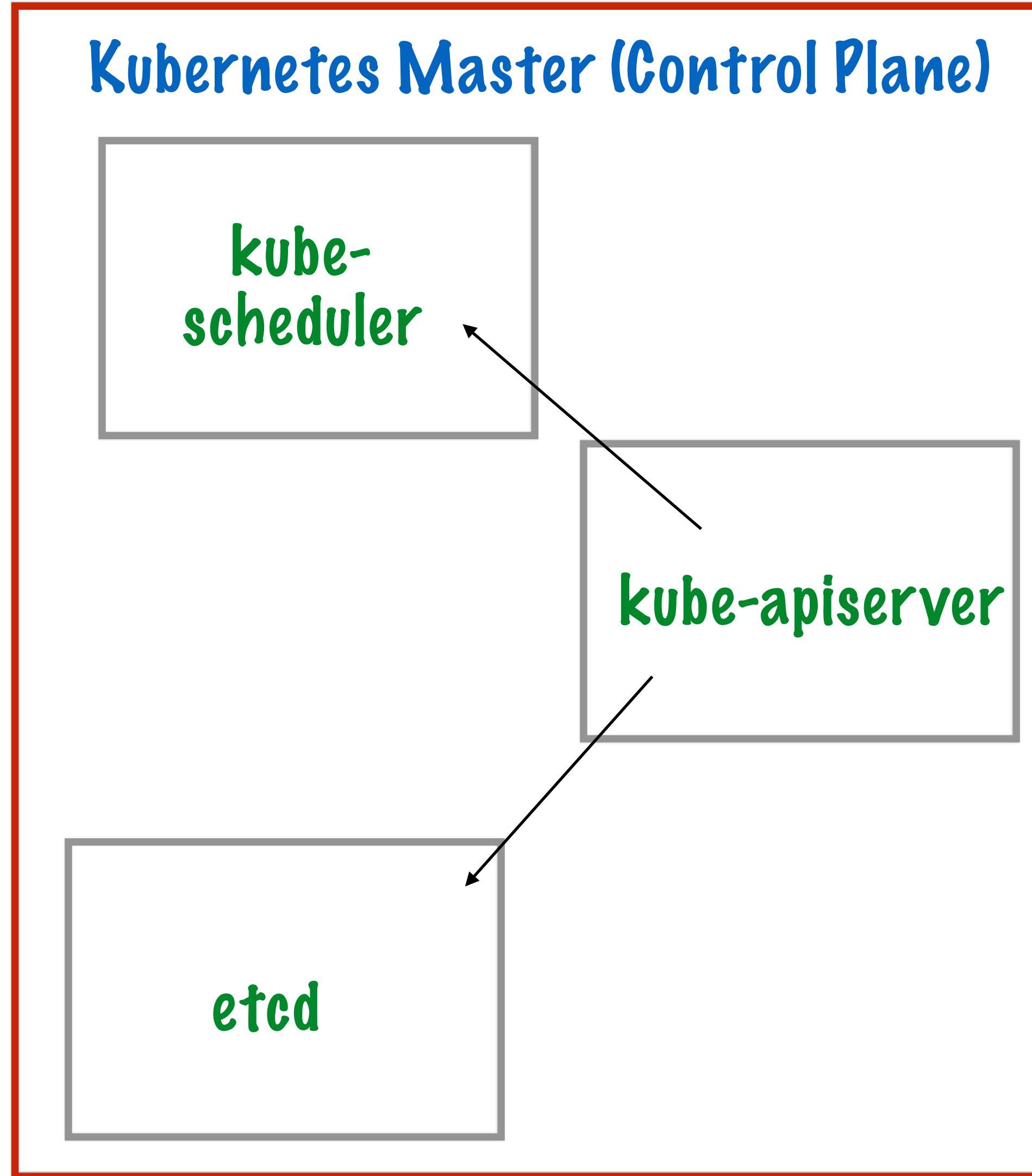
kube-apiserver



Cluster Store for Metadata



kube-scheduler

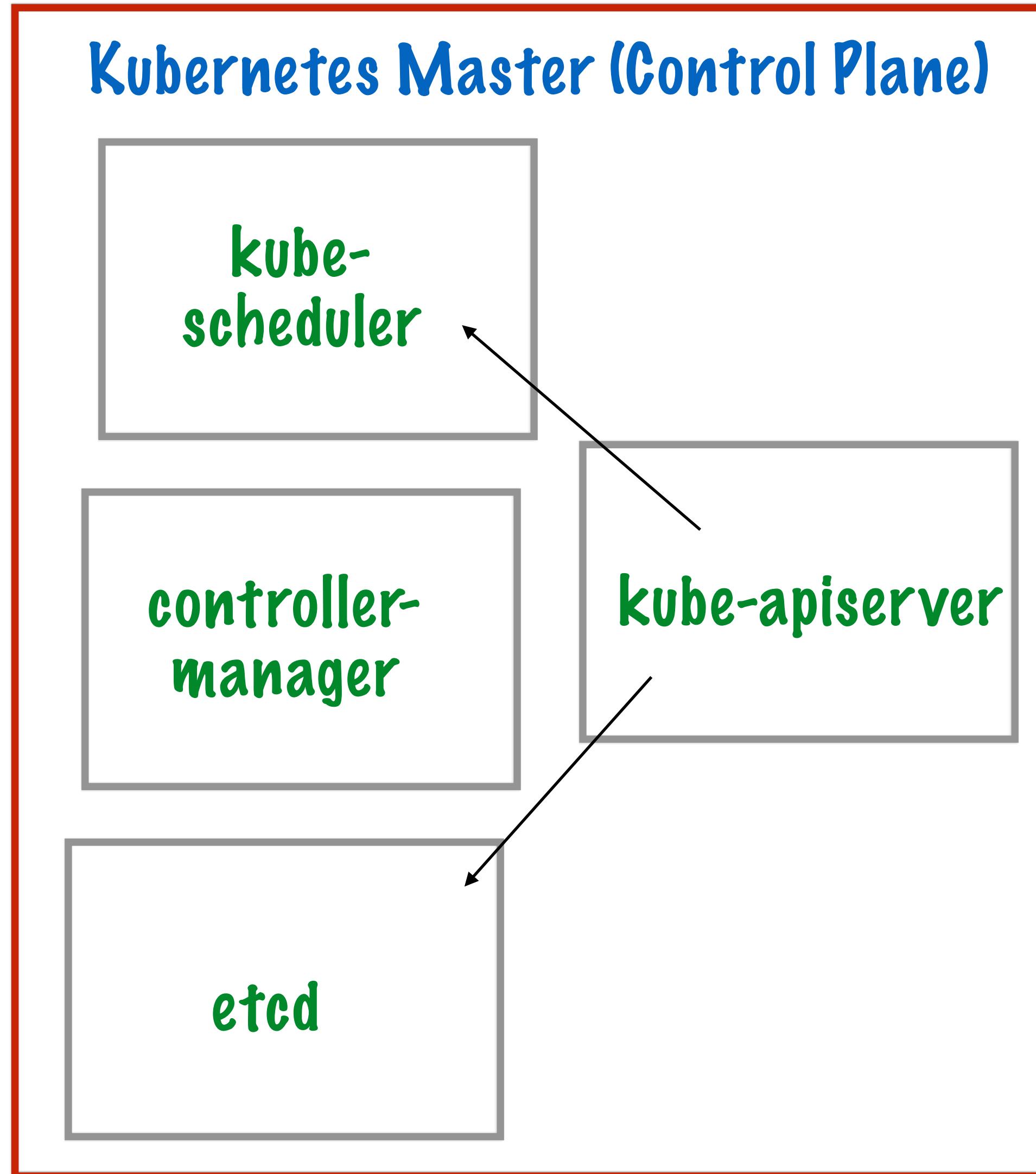


Handle pod creation and management

kube-scheduler match/assign nodes to pods

Complex - affinities, taints, tolerations, ...

controller-manager



Controllers for each object type

ReplicaSet, Deployment, DaemonSet, StatefulSet

Actual state \leftrightarrow Desired State
reconciliation loops

Higher Level Kubernetes Objects

- **ReplicaSet, ReplicationController: Scaling and healing**
- **Deployment: Versioning and rollback**
- **StatefulSet: Individual pods are unique (not fungible)**
- **Run-to-Completion Jobs: Pods that do their job, then go away**
 - **Cronjob**

What are ReplicaSets?

Pods on Kubernetes

Bare pod

ReplicaSet

Deployment

Encapsulates pod template within it

Also specifies number of replicas of the pod

Pods on Kubernetes

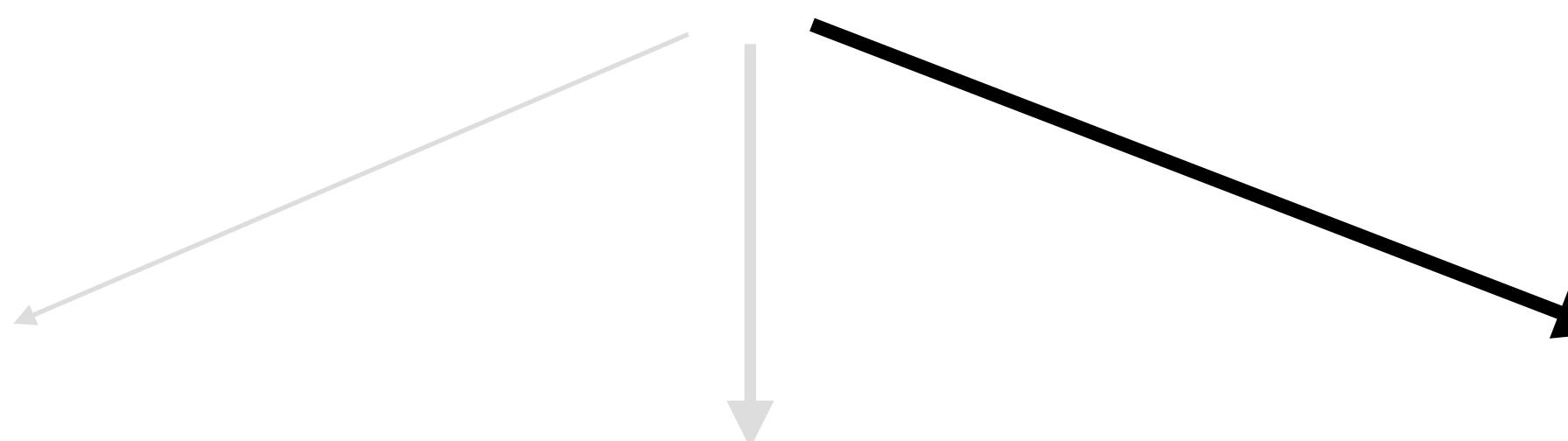
Bare pod

ReplicaSet

Deployment

Encapsulates replicaset
template within it

Versioning, rollback,...



Specification of ReplicaSet

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
    name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

Pod Template
Container Image
Port

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
    - {key: tier, operator: In, values: [frontend]}
template:
  metadata:
    labels:
      app: guestbook
      tier: frontend
  spec:
    containers:
    - name: php-redis
      image: gcr.io/google_samples/gb-frontend:v3
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      env:
      - name: GET_HOSTS_FROM
        value: dns
        # If your cluster config does not include a dns service, then to
        # instead access environment variables to find service host
        # info, comment out the 'value: dns' line above, and uncomment the
        # line below.
        # value: env
      ports:
      - containerPort: 80
```

Specification of ReplicaSet

Number of replicas

If pod crashes, ReplicaSet will start a new one

This field is key to scaling and healing

3 pods will be created, all replicas of each other

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
      ports:
        - containerPort: 80
```

Specification of ReplicaSet

Pod Selector

Q: Which pods should be governed by this ReplicaSet?

A: All pods with labels that match this selector

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
          ports:
            - containerPort: 80
```

Specification of ReplicaSet

Labels carried by ReplicaSet

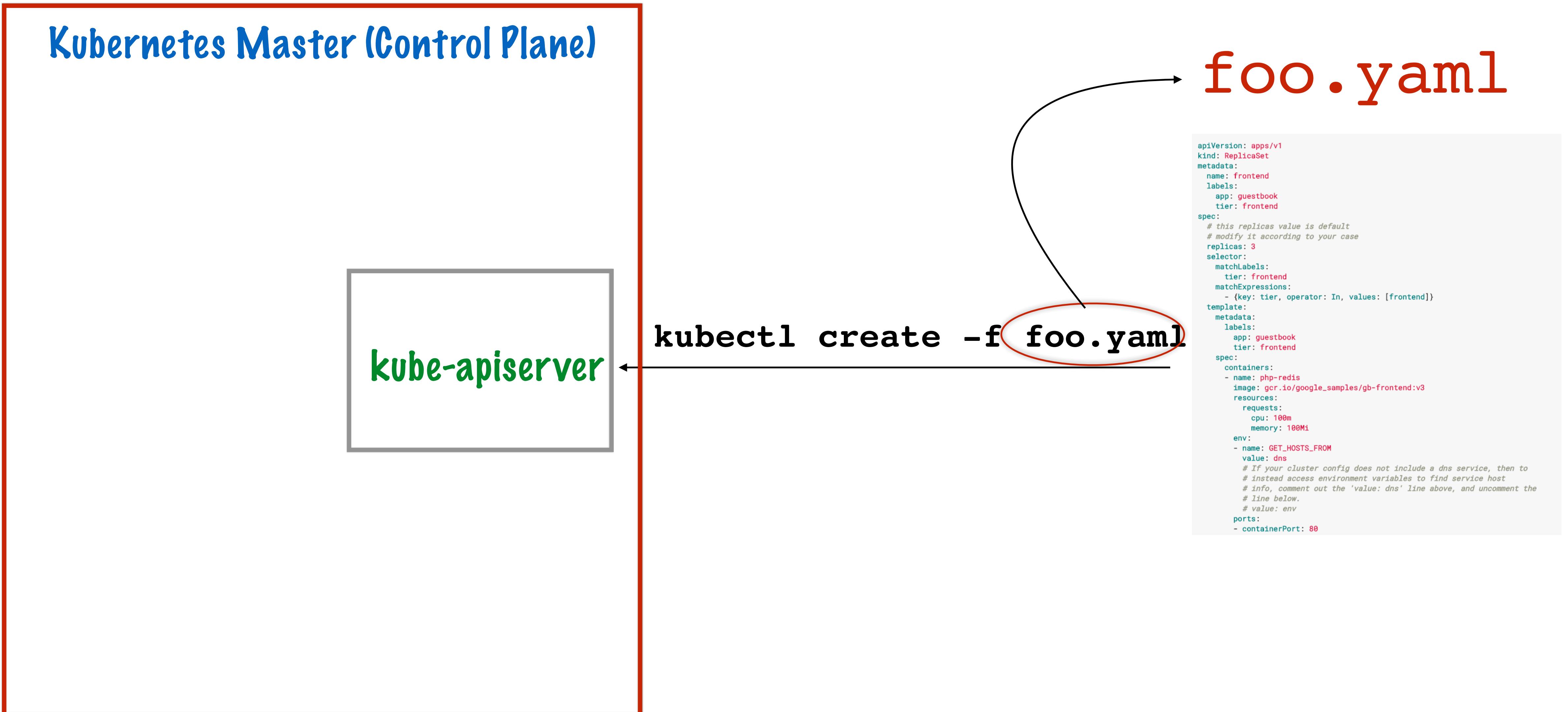
This is metadata, not part of selector

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
        ports:
          - containerPort: 80
```

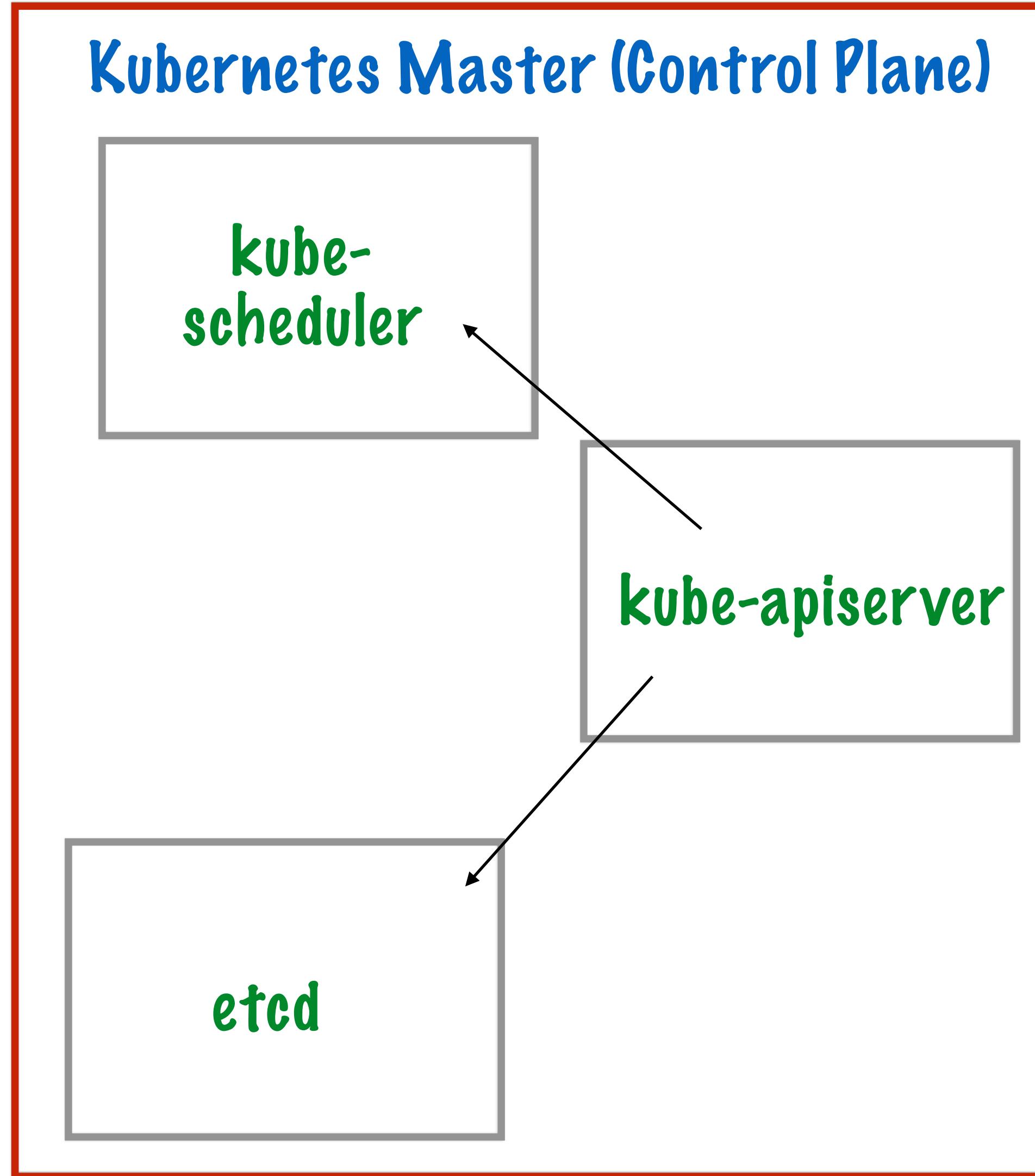
Components of ReplicaSet Specification

- Pod template
- Pod Selector
- Labels of ReplicaSet
- Number of replicas

Create ReplicaSet Using kubectl



Create ReplicaSet Using kubectl

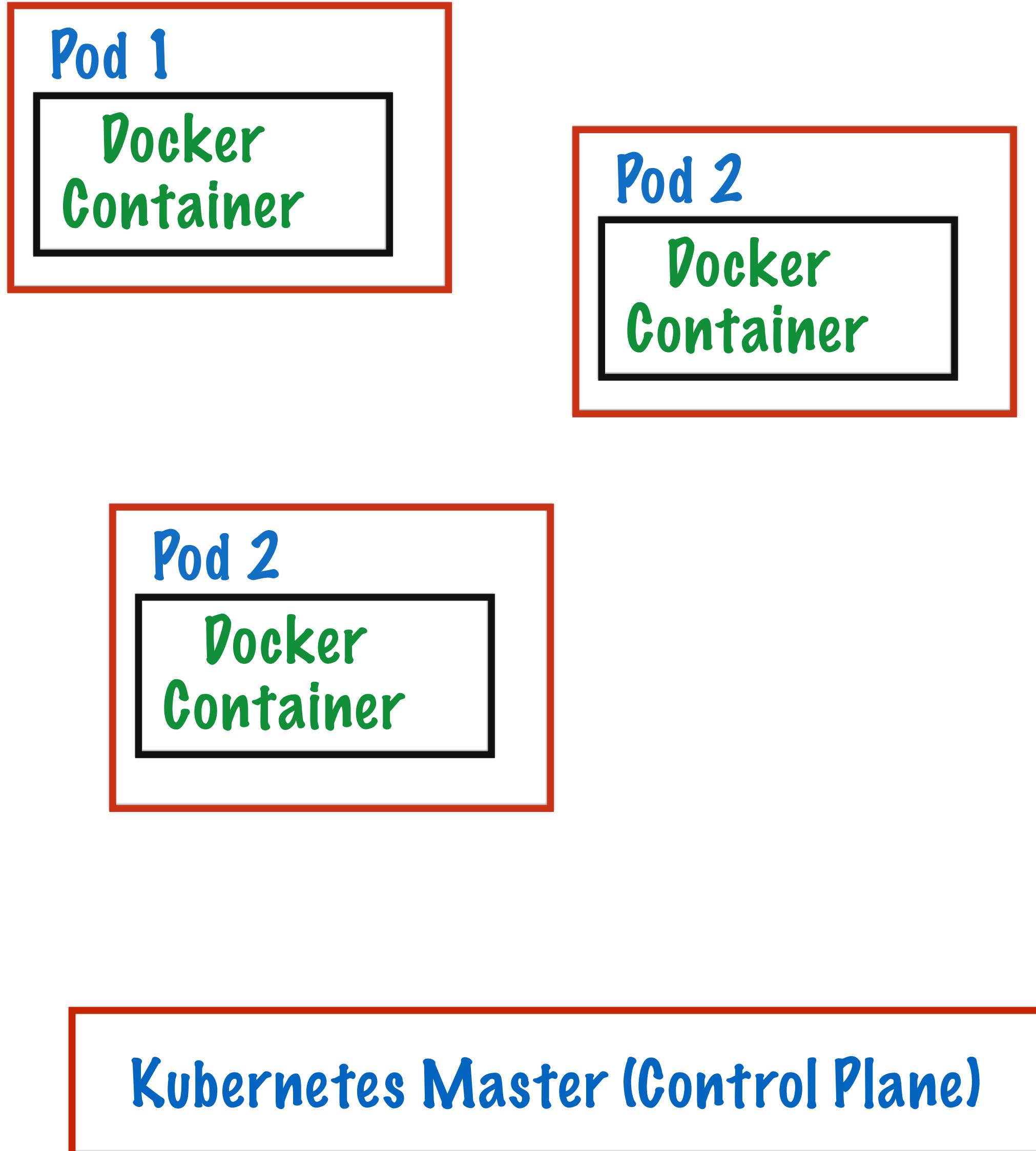


kube-scheduler will take over

kube-scheduler match/assign nodes to pods

Will create three replicas of the same pod

Create ReplicaSet Using kubectl



Three pods created (since num replicas = 3)

kube-scheduler creates these on some nodes in cluster

(Don't worry right now about where)

Alternatives to ReplicaSets

- Deployment: Versioning and rollback
- Run-to-Completion Jobs: Batch processing
- DaemonSet: Pod lifetime tied to that of some node in the cluster
- StatefulSet: Individual pods are unique (not fungible)
- Bare pods - Not recommended - vulnerable, no healing/scaling

How Can We Work With ReplicaSets?

Pods on Kubernetes

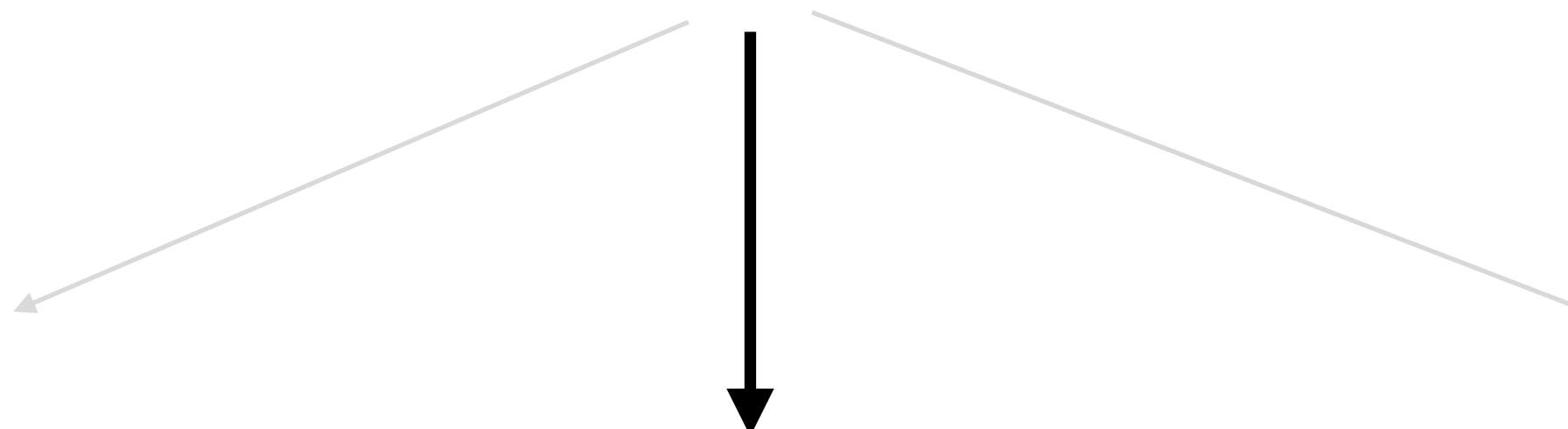
Bare pod

ReplicaSet

Deployment

Encapsulates pod template within it

Also specifies number of replicas of the pod



Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Deleting ReplicaSets

- Deleting a ReplicaSet and its Pods
 - Use `kubectl delete`
- Deleting just a ReplicaSet but not its Pods
 - Use `kubectl delete --cascade=false`
- Deleting ReplicaSet orphans its pods
 - Pods are now vulnerable to crashes
- Probably want a new ReplicaSet to adopt them
 - pod template will not apply

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Isolating Pods From ReplicaSet

Pod Selector

Q: Which pods should be governed by this ReplicaSet?

A: All pods with labels that match this selector

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
          ports:
            - containerPort: 80
```

Isolating Pods From ReplicaSet

Implication: Changing pod labels will isolate them from ReplicaSet

Pod Selector

Q: Which pods should be governed by this ReplicaSet?

A: All pods with labels that match this selector

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
          ports:
            - containerPort: 80
```

Isolating Pods From ReplicaSet

- Change labels on pods created by this ReplicaSet
- Those pods will no longer match the selector - isolated from ReplicaSet!
- ReplicaSet will notice, and create new replicas

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Scaling ReplicaSet

Number of replicas

If pod crashes, ReplicaSet will start a new one

This field is key to scaling and healing

3 pods will be created, all replicas of each other

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
      ports:
        - containerPort: 80
```

Scaling ReplicaSet

Implication: Update number of replicas to scale the ReplicaSet

Number of replicas

If pod crashes, ReplicaSet will start a new one

This field is key to scaling and healing

3 pods will be created, all replicas of each other

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
          ports:
            - containerPort: 80
```

Scaling ReplicaSet

```
kubectl apply -f configs/
```

Simply re-apply the new template

Number of replicas will be adjusted by the ReplicaSet Controller

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- **Auto-Scaling a ReplicaSet**

Auto-Scaling a ReplicaSet

- **Horizontal Pod Autoscaler Target**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

How Do ReplicaSets Ensure Loose Coupling With Pods?

Specification of ReplicaSet

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
    name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

Pod Template
Container Image
Port

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
    - {key: tier, operator: In, values: [frontend]}
template:
  metadata:
    labels:
      app: guestbook
      tier: frontend
  spec:
    containers:
    - name: php-redis
      image: gcr.io/google_samples/gb-frontend:v3
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      env:
      - name: GET_HOSTS_FROM
        value: dns
        # If your cluster config does not include a dns service, then to
        # instead access environment variables to find service host
        # info, comment out the 'value: dns' line above, and uncomment the
        # line below.
        # value: env
      ports:
      - containerPort: 80
```

Specification of ReplicaSet

Pod Selector

Q: Which pods should be governed by this ReplicaSet?

A: All pods with labels that match this selector

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
          ports:
            - containerPort: 80
```

Pod Selector

ReplicaSet Object

```
selector:  
  matchLabels:  
    tier: frontend  
  matchExpressions:  
    - {key: tier, operator: In, values: [frontend]}
```

Pod 1

Labels

```
{  
  tier: frontend,  
  env: prod,  
  geo: india  
}
```

Pod 2

Labels

```
{  
  tier: frontend,  
  env: dev,  
  geo: india  
}
```

Pod 3

Labels

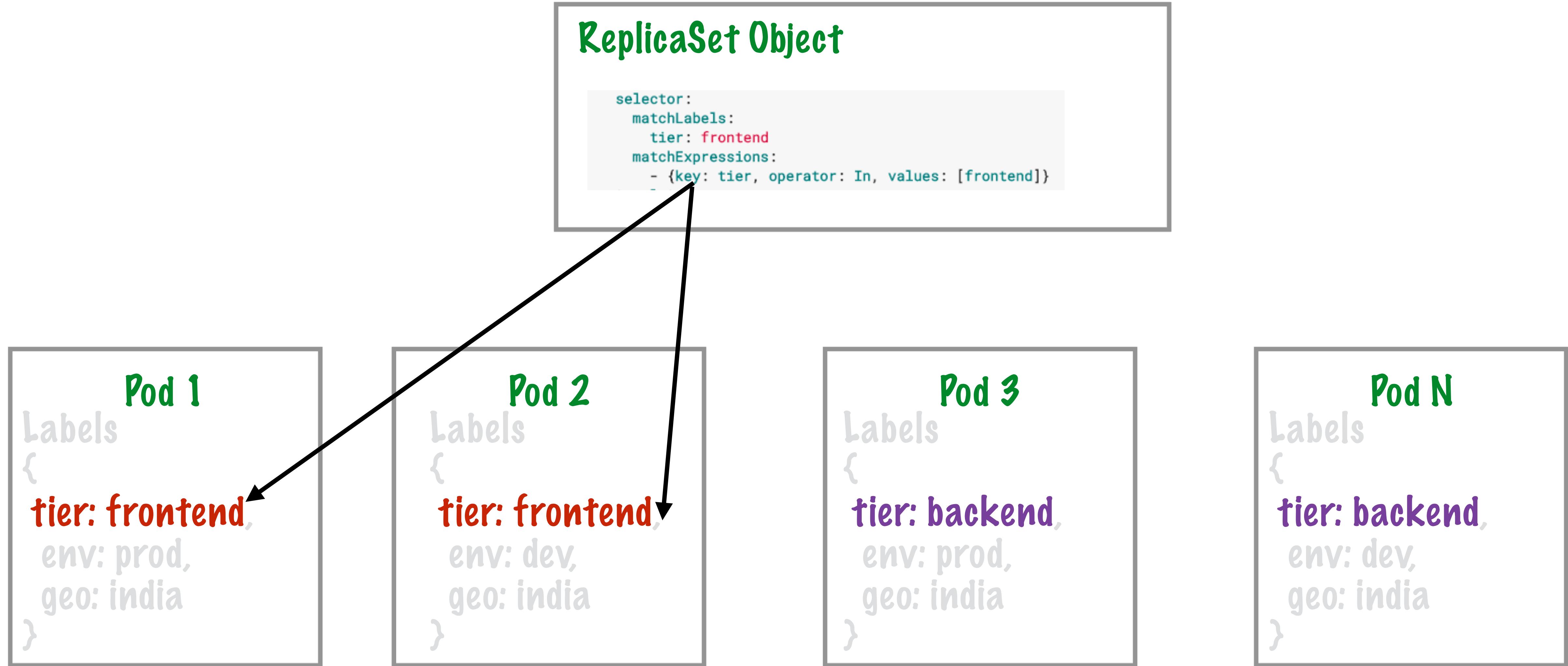
```
{  
  tier: backend,  
  env: prod,  
  geo: india  
}
```

Pod N

Labels

```
{  
  tier: backend,  
  env: dev,  
  geo: india  
}
```

Pod Selector



Specification of ReplicaSet

Labels carried by ReplicaSet

This is metadata, not part of selector

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
        ports:
          - containerPort: 80
```

Specification of ReplicaSet

Number of replicas

If pod crashes, ReplicaSet will start a new one

This field is key to scaling and healing

3 pods will be created, all replicas of each other

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
      ports:
        - containerPort: 80
```

Loose Coupling

- Can Delete a ReplicaSet without affecting pods
- Can isolate pods from a ReplicaSet
- Can change ReplicaSet governing pods on the fly

Loose Coupling

- Can Delete a ReplicaSet without affecting pods
 - Use `kubectl delete --cascade=false`
- Can isolating pods from a ReplicaSet
 - Change labels on pods so that selector no longer matches
- Can change ReplicaSet governing pods on the fly
 - Create new ReplicaSet with the same label selector

Isolating Pods From ReplicaSet

- Change labels on pods created by this ReplicaSet
- Those pods will no longer match the selector - isolated from ReplicaSet!
- ReplicaSet will notice, and create new replicas

Isolating Pods From ReplicaSet

Pod Selector

Q: Which pods should be governed by this ReplicaSet?

A: All pods with labels that match this selector

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
          ports:
            - containerPort: 80
```

Isolating Pods From ReplicaSet

Implication: Changing pod labels will isolate them from ReplicaSet

Pod Selector

Q: Which pods should be governed by this ReplicaSet?

A: All pods with labels that match this selector

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
          ports:
            - containerPort: 80
```

Loose Coupling

- Can Delete a ReplicaSet without affecting pods
 - Use `kubectl delete --cascade=false`
- Can isolating pods from a ReplicaSet
 - Change labels on pods so that selector no longer matches
- Can change ReplicaSet governing pods on the fly
 - Create new ReplicaSet with the same label selector

What Is A Horizontal Pod Autoscaler(HPA)?

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Scaling ReplicaSet

Number of replicas

If pod crashes, ReplicaSet will start a new one

This field is key to scaling and healing

3 pods will be created, all replicas of each other

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
      ports:
        - containerPort: 80
```

Scaling ReplicaSet

Implication: Update number of replicas to scale the ReplicaSet

Number of replicas

If pod crashes, ReplicaSet will start a new one

This field is key to scaling and healing

3 pods will be created, all replicas of each other

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below.
              # value: env
          ports:
            - containerPort: 80
```

Scaling ReplicaSet

```
kubectl apply -f configs/
```

Simply re-apply the new template

Number of replicas will be adjusted by the ReplicaSet Controller

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- Auto-Scaling a ReplicaSet

Working with ReplicaSets

- Deleting a ReplicaSet
 - ... and its Pods
 - ... but not its Pods
- Isolating pods from a ReplicaSet
- Scaling a ReplicaSet
- **Auto-Scaling a ReplicaSet**

Auto-Scaling a ReplicaSet

- **Horizontal Pod Autoscaler Target**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Auto-Scaling a ReplicaSet

- **Horizontal Pod Autoscaler Target**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Auto-Scaling a ReplicaSet

- **Horizontal Pod Autoscaler Target**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Auto-Scaling a ReplicaSet

- **Horizontal Pod Autoscaler Target**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Auto-Scaling a ReplicaSet

- **Horizontal Pod Autoscaler Target**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Auto-Scaling a ReplicaSet

- **Horizontal Pod Autoscaler Target**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Auto-Scaling a ReplicaSet

- **Horizontal Pod Autoscaler Target**

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

HPA

- Control-loop to track actual and desired CPU utilisation in pod
- Target: ReplicationControllers, Deployments, ReplicaSets,
- Policy CPU Utilisation or custom-metrics
- Won't work with non-scaling objects: DaemonSets

Preventing Thrashing

- Thrashing is always a risk with autoscaling
- Cooldown periods help HPA avoid this
 - --horizontal-pod-autoscaler-downscale-delay
 - --horizontal-pod-autoscaler-upscale-delay
- Interval between successive operations of the same kind

Working with HPAs

- `kubectl create hpa`
- `kubectl get hpa`
- `kubectl describe hpa`
- `kubectl autoscale rs front-end --min=3 --max=10 --cpu-percent=50`

What are ReplicationControllers?

Pods on Kubernetes

Bare pod

ReplicaSet

Deployment

Encapsulates pod template within it

Also specifies number of replicas of the pod

Pods on Kubernetes

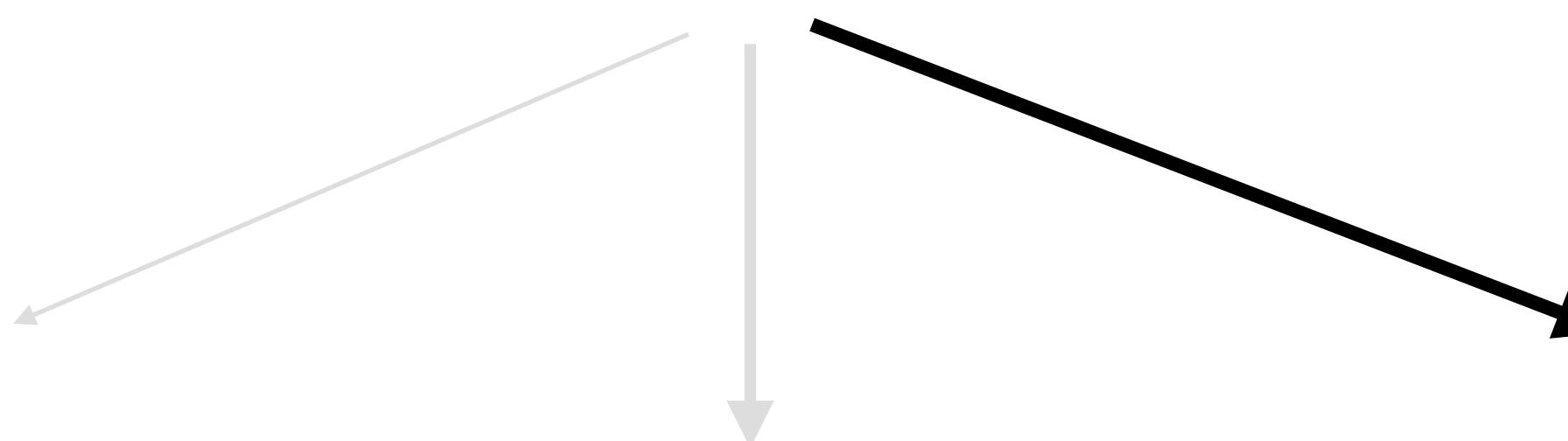
Bare pod

ReplicaSet

Deployment

Encapsulates replicaset
template within it

Versioning, rollback,...



ReplicationController

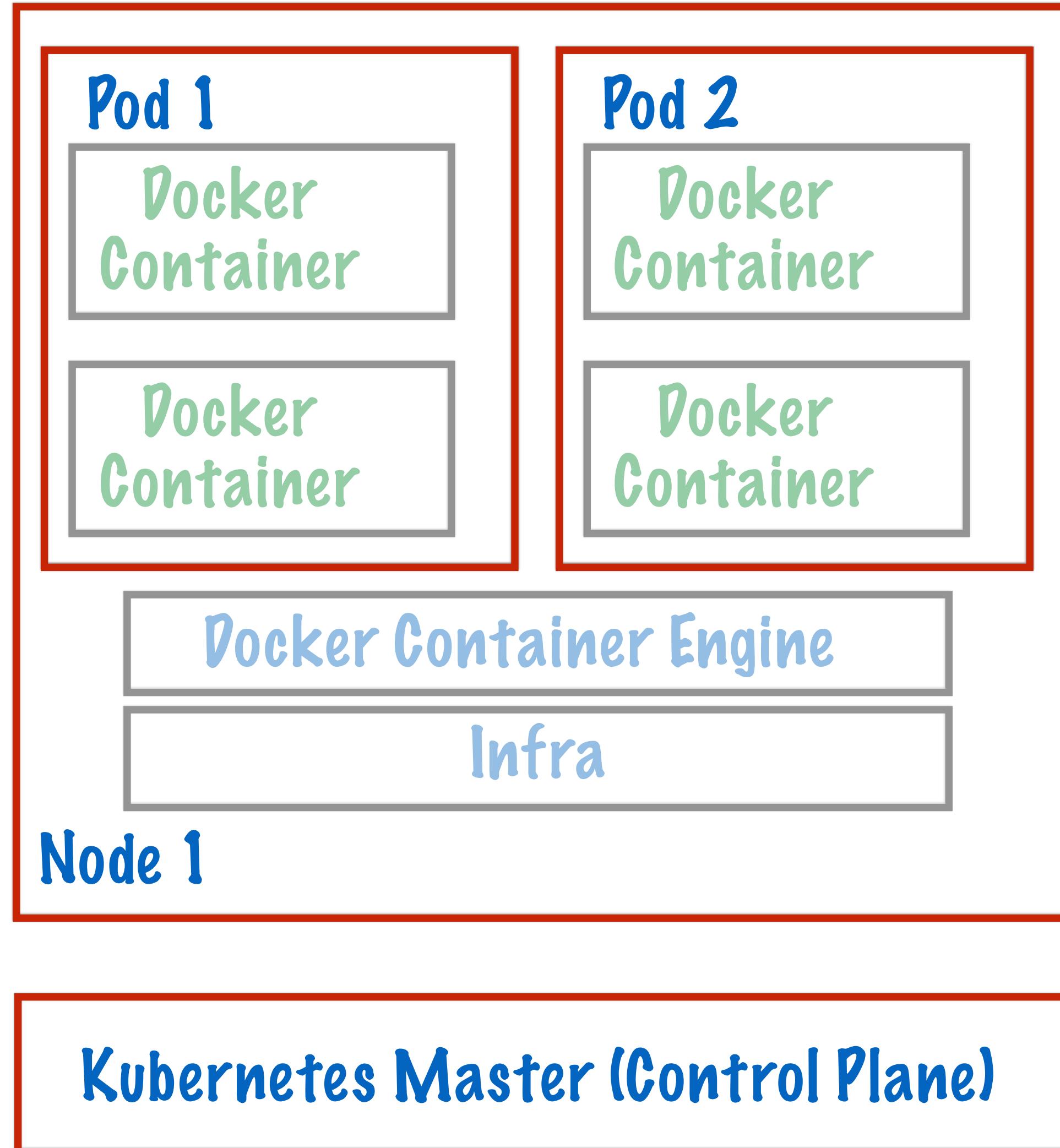
- Before deployments and ReplicaSets...
- ReplicationControllers provided the same functionality
- Like ReplicaSets - have pod template and # replicas
- Like Deployments - have rolling update functionality

ReplicationController

- ReplicationControllers selector support is only equality-based
- Selector support in ReplicaSet is more advanced
 - based on set-based selectors
- ReplicationControllers are obsolete now
 - use Deployment + ReplicaSet instead

What Are Deployments?

Pods on Kubernetes Nodes



Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods Limitations

- No auto-healing or scaling
- Pod crashes? Must be handled at higher level
 - ReplicaSet, Deployment, Service
- Ephemeral: IP addresses are ephemeral

Pods on Kubernetes

Bare pod

ReplicaSet

Deployment

Encapsulates pod template within it

Also specifies number of replicas of the pod

Pods on Kubernetes

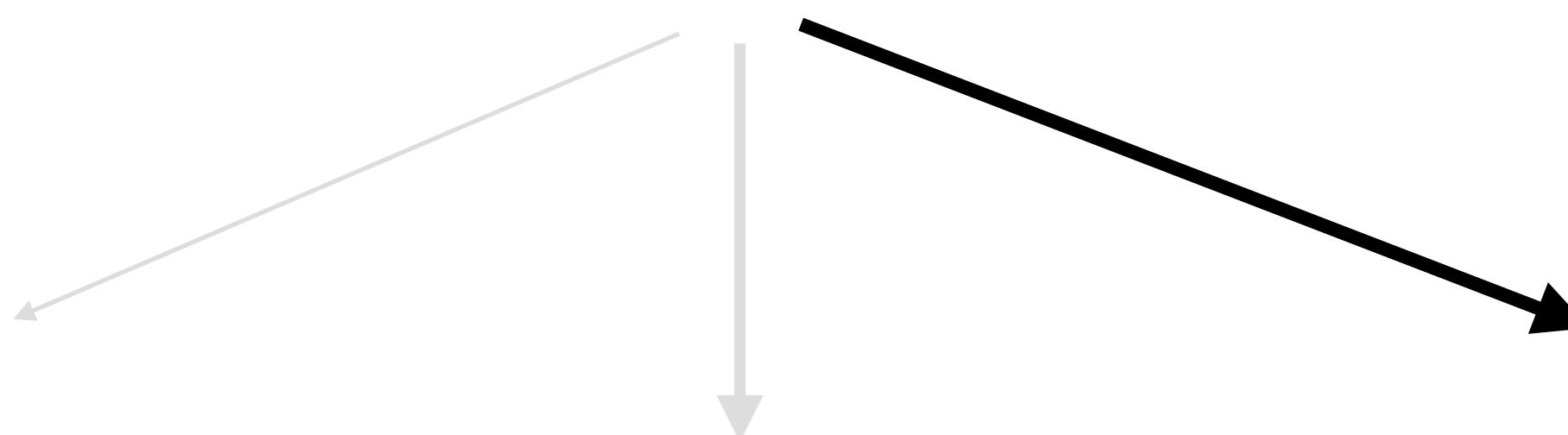
Bare pod

ReplicaSet

Deployment

Encapsulates replicaset
template within it

Versioning, rollback,...



Deployment Spec

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
        ports:
          - containerPort: 80
```

ReplicaSet → Deployment

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
template:
  metadata:
    labels:
      app: guestbook
      tier: frontend
  spec:
    containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
          - name: GET_HOSTS_FROM
            value: dns
            # If your cluster config does not include a dns service, then to
            # instead access environment variables to find service host
            # info, comment out the 'value: dns' line above, and uncomment the
            # line below.
            # value: env
        ports:
          - containerPort: 80
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Replica Template

ReplicaSet Spec

Pod Selector

Q: Which pods should be governed by this Deployment?

A: All pods with labels that match this selector

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Pod -> ReplicaSet

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

Pod Template

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicaset...
  # modify it accordingly
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
    - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and uncomment the
          # line below.
          # value: env
      ports:
      - containerPort: 80
```

Number of replicas

Magic Of Deployment Objects

- ReplicaSets associated with Deployment
- Rollback
- Magic

Magic Of Deployment Objects

- ReplicaSets associated with Deployment
- Rollback
- Magic

ReplicaSets associated with Deployment

- Update container in pod template in deployment
- New ReplicaSet and new pods created!
- Old ReplicaSet continues to exist
- Pods in old ReplicaSet gradually reduced to zero

Magic Of Deployment Objects

- ReplicaSets associated with Deployment
- Rollback
- Magic

Magic Of Deployment Objects

- ReplicaSets associated with Deployment
- Rollback
- Magic

Rollback

- Every change to deployment template is tracked
- Creates a new revision of the deployment
- Trivial to roll back to any previous revision

Magic Of Deployment Objects

- ReplicaSets associated with Deployment
- Rollback
- Magic

Magic Of Deployment Objects

- ReplicaSets associated with Deployment
- Rollback
- **Magic**

Magic

- Versioning
- Instant rollback
- Rolling deployments
- Blue-green
- Canary

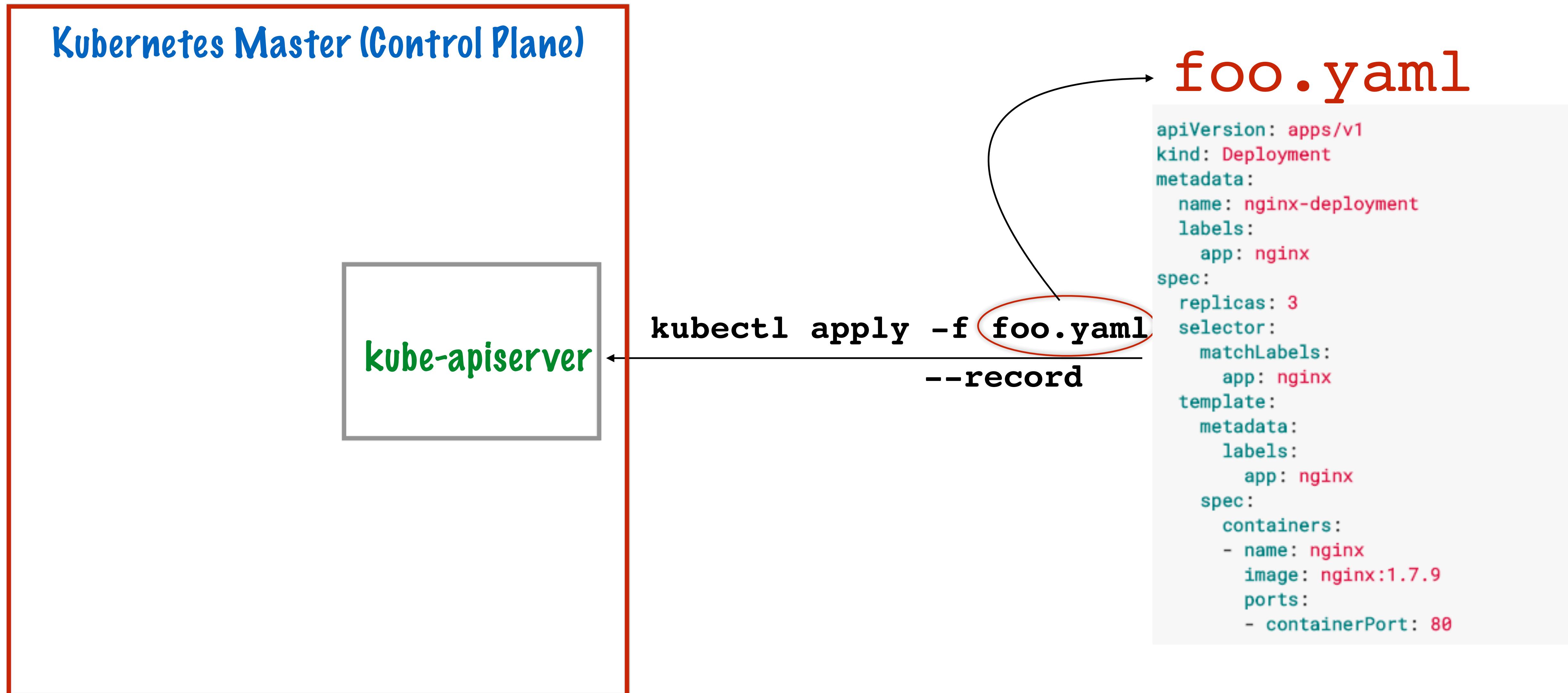
What Are Usecases For Deployment Objects?

Deployment Usecases

- Primary usecase: To rollout a ReplicaSet (create new pods)
- Update state of existing deployment: just update pod template
 - new replicates created, pods moved over in a controlled manner
- Rollback to earlier version: simply go back to previous revision of deployment
- Scale up: edit number of replicas
- Pause/Resume deployments mid-way (after fixing bugs)
- Check status of a deployment (using the status field)
- Clean up old replicaset that are not needed any more

How are Deployments Created?

Creating Deployments



ReplicaSet -> Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
        ports:
          - containerPort: 80
```

ReplicaSet → Deployment

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
template:
  metadata:
    labels:
      app: guestbook
      tier: frontend
  spec:
    containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
          - name: GET_HOSTS_FROM
            value: dns
            # If your cluster config does not include a dns service, then to
            # instead access environment variables to find service host
            # info, comment out the 'value: dns' line above, and uncomment the
            # line below.
            # value: env
        ports:
          - containerPort: 80
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Replica Template

Pod -> ReplicaSet

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
    initialDelaySeconds: 15
    timeoutSeconds: 1
  name: liveness
```

Pod Template

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicaset...
  # modify it accordingly
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
    - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and uncomment the
          # line below.
          # value: env
      ports:
      - containerPort: 80
```

Number of replicas

ReplicaSet -> Deployment

Pod Selector

Q: Which pods should be governed by this Deployment?

A: All pods with labels that match this selector

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Fields in Deployment Specification

- **Selector**
 - Matches pods that have the same labels as those specified in the selector
- **Strategy: How will old pods be replaced?**
 - `.spec.strategy.type==Recreate`
 - `.spec.strategy.type==RollingUpdate`
- **More Hooks for Rolling Update:**
 - `.spec.strategy.rollingUpdate.maxUnavailable`
 - `.spec.strategy.rollingUpdate.maxSurge`

More Fields - More Details to Come

- progressDeadlineSeconds
- minReadySeconds
- rollbackTo
- revisionHistoryLimit
- paused

How are Deployments Rolled Back?

Rollback

- Every change to deployment template is tracked
- Creates a new revision of the deployment
- Trivial to roll back to any previous revision

Fine Print: Only PodTemplate Changes Tracked!

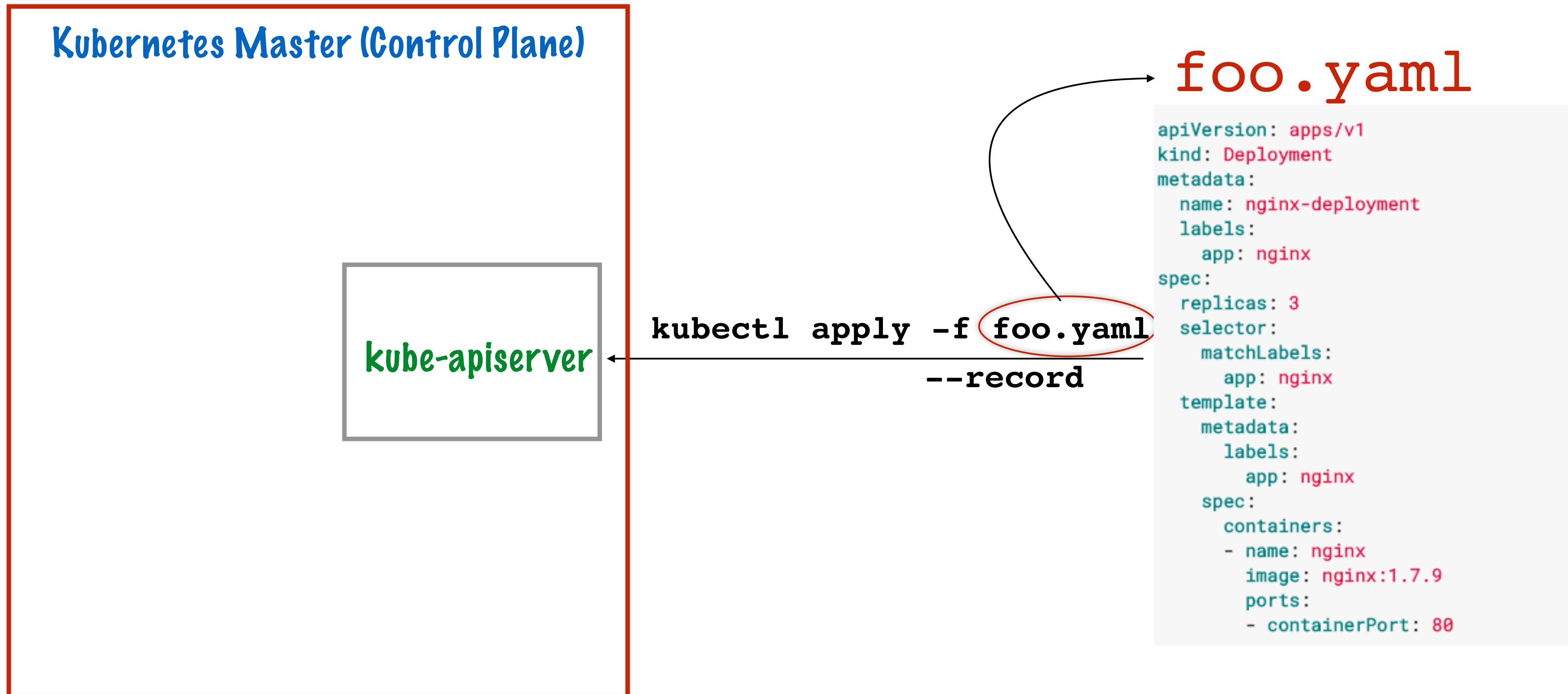
- New revisions created for any change in pod template
 - These changes are trivial to roll back
- Other changes to manifest: eg. scaling do not create new revision
 - Can not roll back scaling easily

Single Command To Rollback

```
kubectl rollout undo deployment/  
nginx-deployment
```

```
kubectl rollout undo deployment/  
nginx-deployment --to-revision=2
```

Create Using --record Switch



How Can Deployments Be Scaled?

Scaling Deployments

- Imperative: `kubectl scale` commands

```
$ kubectl scale deployment nginx-deployment --replicas=10
deployment "nginx-deployment" scaled
```

- Declarative: Change number of replicas and re-apply
 - Scaling does not change pod template
 - So does not trigger creation of a new revision
 - Can't rollback scaling that easily
- Can also scale using Horizontal Pod Autoscaler (HPA)

```
$ kubectl autoscale deployment nginx-deployment --min=10 --max=15 --cpu-percent=80
deployment "nginx-deployment" autoscaled
```

Proportionate Scaling

- During rolling deployments, two ReplicaSets exist
 - old version
 - new version
- Proportionate scaling will scale pods in both ReplicaSets

How Can Deployments Be Paused And Resumed?

Pausing Deployments

- Imperative: `kubectl pause/resume commands`

```
$ kubectl rollout resume deployment/nginx-deployment  
deployment "nginx" resumed
```

```
$ kubectl rollout pause deployment/nginx-deployment  
deployment "nginx-deployment" paused
```

- Declarative: Change spec.Paused boolean

- Does not change pod template
- Does not trigger revision creation

Pausing Deployments

- Can make changes or debug while paused
- Changes to pod template while paused will not take effect until resumed
- Can not rollback paused deployment (need to resume it first)

What Does Deployment Clean-Up Policy Control?

Clean-Up Policy

- Important: Don't change this unless you understand it
- ReplicaSets associated with Deployment
 - New ReplicaSet for each revision
 - So, one ReplicaSet for each change to pod template
 - `.spec.revisionHistoryLimit` controls how many such revisions kept
- Setting `.spec.revisionHistoryLimit = 0` cleans up all history, no rollback possible

What Are Stateful Sets?

Stateful sets

- Manage Pods
- Maintains a sticky identity
- Pods are created from the same spec
- Not interchangeable
- Identifier maintains across any rescheduling

Use cases

- Ordered, graceful deployment and scaling
- Ordered, graceful deletion and termination
- Ordered, automated rolling updates
- Stable, unique network identifiers
- Stable, persistent storage

Limitations

- Pod must either be provisioned by a PersistentVolume Provisioner
- Deleting and/or scaling a StatefulSet down will not delete the volumes associated with the StatefulSet
- StatefulSets currently require a Headless Service

Deployment and Scaling Guarantees

- N replicas, when Pods are being deployed, created sequentially, in order from {0..N-1}
- When Pods are being deleted, they are terminated in reverse order, from {N-1..0}
- Before a scaling operation is applied to a Pod, all of its predecessors must be Running and Ready.
- Before a Pod is terminated, all of its successors must be completely shutdown.

What Are DaemonSets?

DaemonSet

- Ensure all (or some subset) Nodes run a copy of a Pod
- As nodes are added - pods added
- As nodes removed - pods are garbage collected

DaemonSet

- Cluster storage daemons
- log collection daemons
- node monitoring daemons

Alternatives

- Initialisation script
 - can't monitor daemon logs
 - can't use same language as rest of cluster
 - can't specify resource limits
- Bare pods
 - No auto-healing, scaling etc
- Static pods: pods not created using kubectl
 - create pod by writing to directories watched by kubelet

What are Jobs?

Higher Level Kubernetes Objects

- **ReplicaSet, ReplicationController: Scaling and healing**
- **Deployment: Versioning and rollback**
- **StatefulSet: Individual pods are unique (not fungible)**
- **Run-to-Completion Jobs: Pods that do their job, then go away**
 - **Cronjob**

Job Objects

- Create pods
- Track their completion
- Ensure specified number terminate successfully
- Deleting job cleans up pods

Types of Jobs

- Non-parallel jobs
- Parallel jobs with fixed completion count
- Parallel jobs with work queue

Types of Jobs

- Non-parallel jobs: can use to force 1 pod to run successfully
- Parallel jobs with fixed completion count: job completes when number of completions reaches target
- Parallel jobs with work queue: requires coordination

Tracking Pods of Jobs

- Once completed: no more pods created
- Existing pods not deleted
- State set to Terminated
- Can find them using `kubectl show pods -a`

Prevent Infinite Loops

- If pods keep failing, jobs keep creating
- Can lead to infinite loop
- Use `spec.activeDeadlineSeconds` field to prevent this

What Are CronJobs?

Use cases

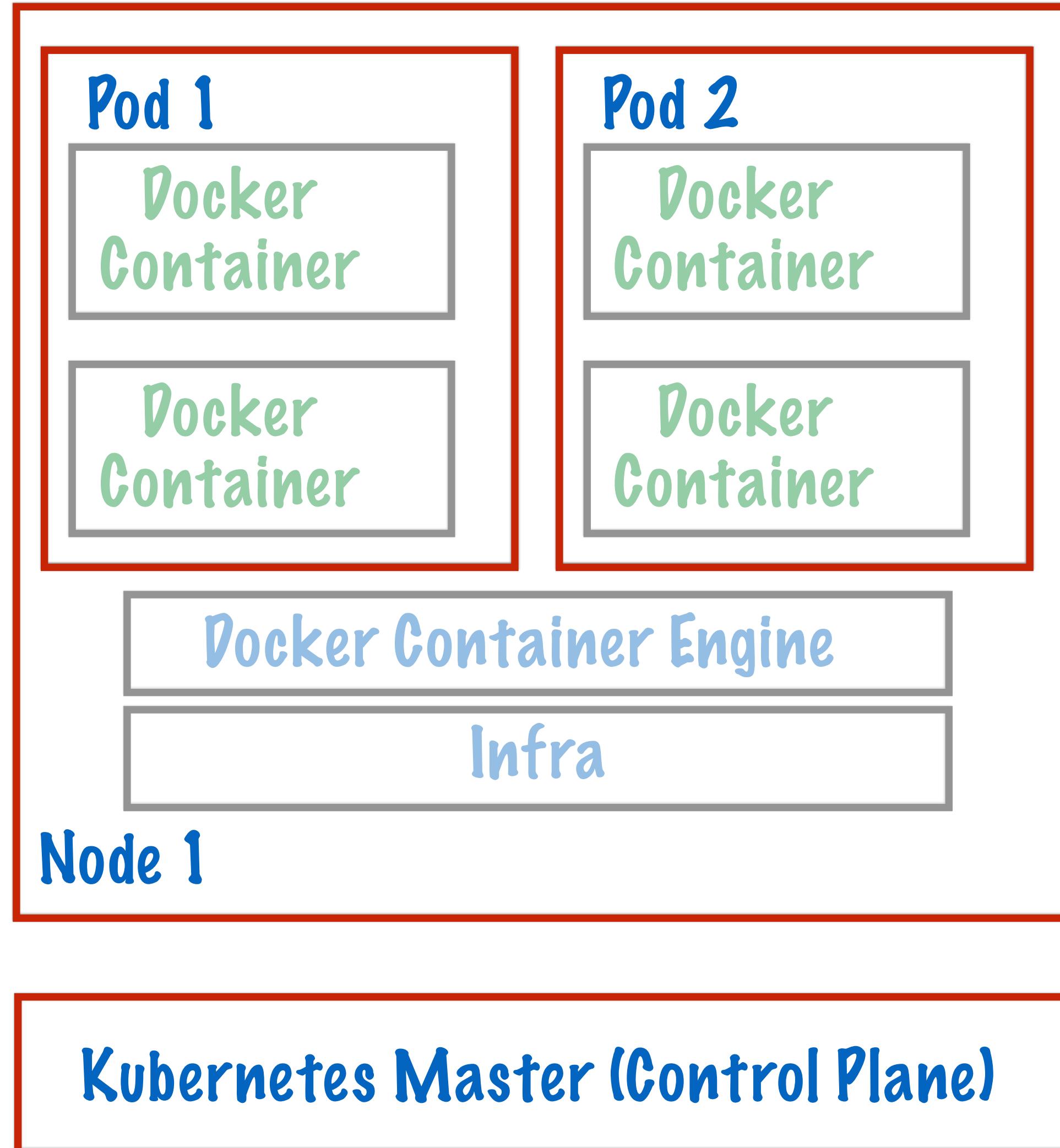
- Manages time based job
 - Once at a specified point in time
 - Repeatedly at a specified point in time
- Schedule a job execution at a given point in time
- Create a periodic job, e.g. database backup, sending emails

Limitations

- jobs should be idempotent
- Only responsible for creating Jobs that match its schedule

What is a Service?

Pods on Kubernetes Nodes



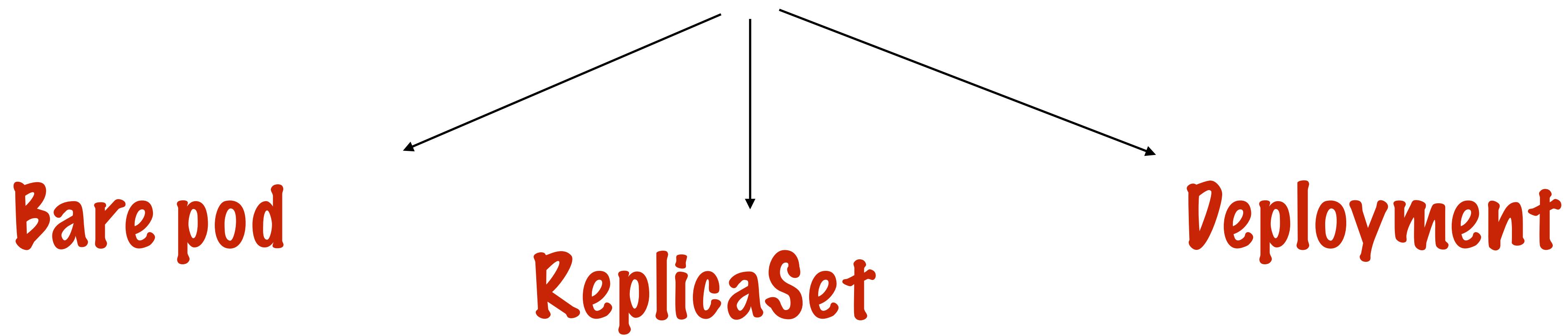
Pod: Atomic unit of deployment in Kubernetes

Consists of 1 or more tightly coupled containers

Pod runs on node, which is controlled by master

Pods on Kubernetes

Pod Creation Requests



Pods are usually not created directly

Higher level abstractions preferred

Pods Limitations

- No auto-healing or scaling
- Pod crashes? Must be handled at higher level
 - ReplicaSet, Deployment, Service
- Ephemeral: IP addresses are ephemeral

Ephemeral IP Addresses

- Containers expose ports in Pod spec
- But Pod IP addresses keep changing!
- For instance, when ReplicaSets or Deployments take pods up/down, IP addresses will change
- How is a client to know how to access the code in a container?

Services For Stable IP Addresses

- Service object solves this problem!
- Service = Logical set of backend pods + stable front-end
- Front-end: Static IP address + Port + DNS Name
- Back-end: Logical set of backend pods (label selector)

Pod Selector

Service Object

```
selector:  
  matchLabels:  
    tier: frontend  
  matchExpressions:  
    - {key: tier, operator: In, values: [frontend]}
```

Pod 1

Labels

```
{  
  tier: frontend,  
  env: prod,  
  geo: india  
}
```

Pod 2

Labels

```
{  
  tier: frontend,  
  env: dev,  
  geo: india  
}
```

Pod 3

Labels

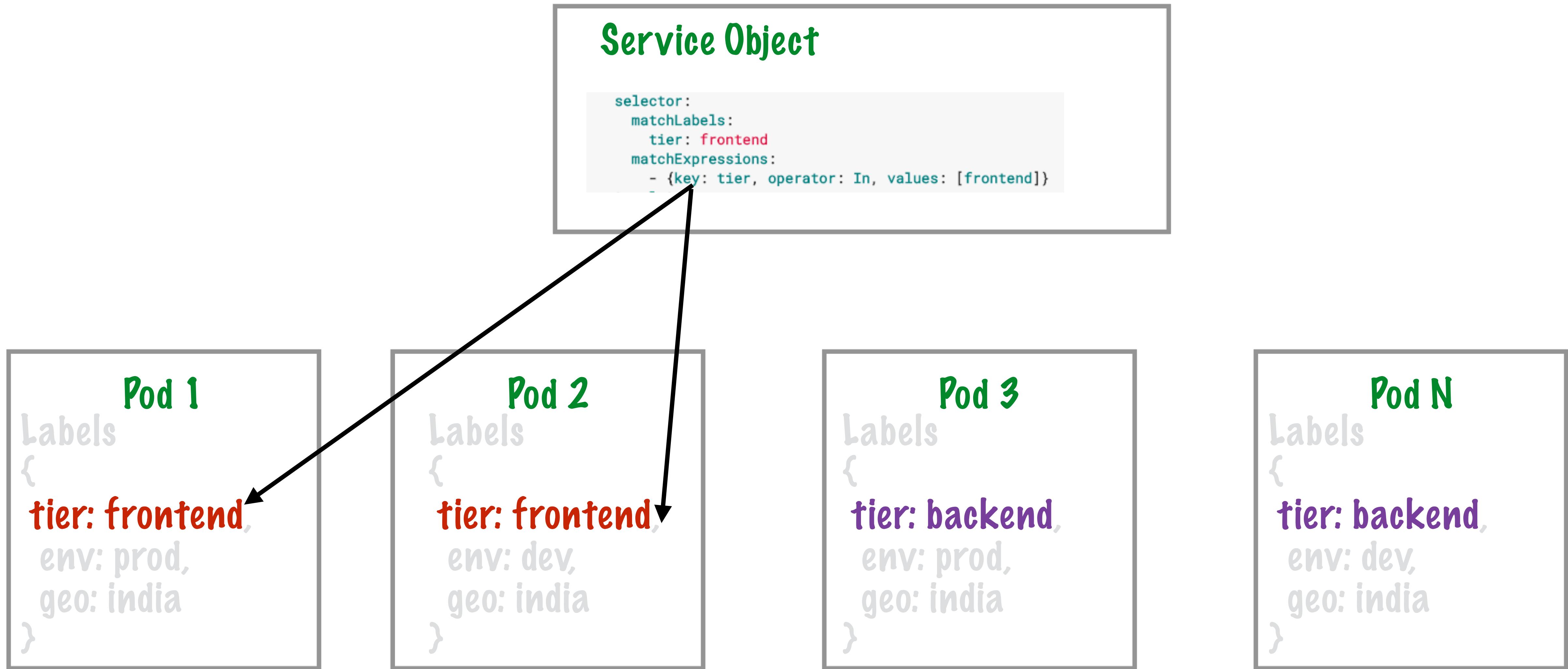
```
{  
  tier: backend,  
  env: prod,  
  geo: india  
}
```

Pod N

Labels

```
{  
  tier: backend,  
  env: dev,  
  geo: india  
}
```

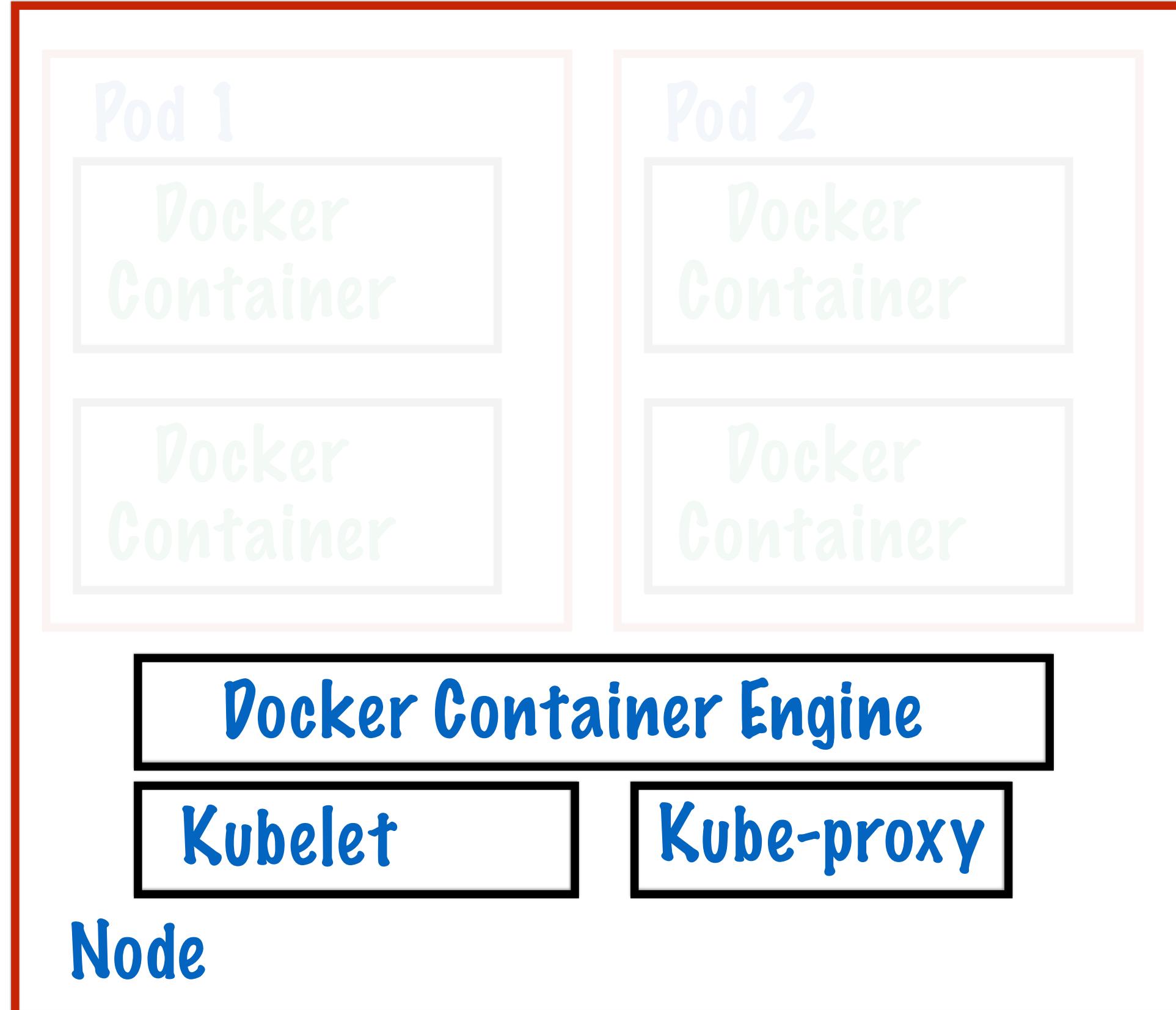
Pod Selector



Endpoint Object

- Dynamic list of pods that are selected by a Service
- Each service object has an associated endpoint object
- Kubernetes evaluates service label selector vs. all pods in cluster
- Dynamic list is updated as pods are created/deleted

Kubernetes Node (Minion)



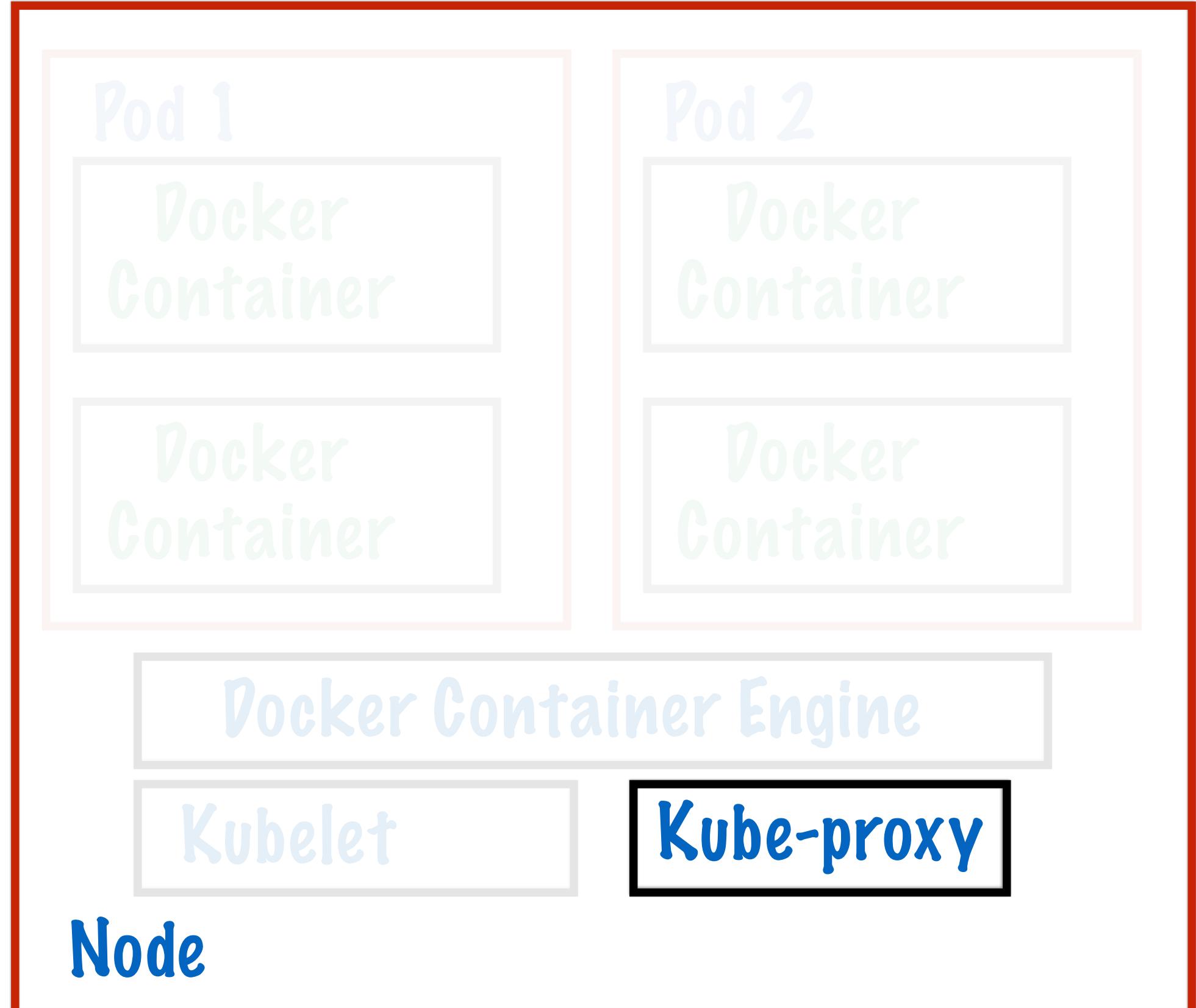
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Kubernetes Node (Minion)



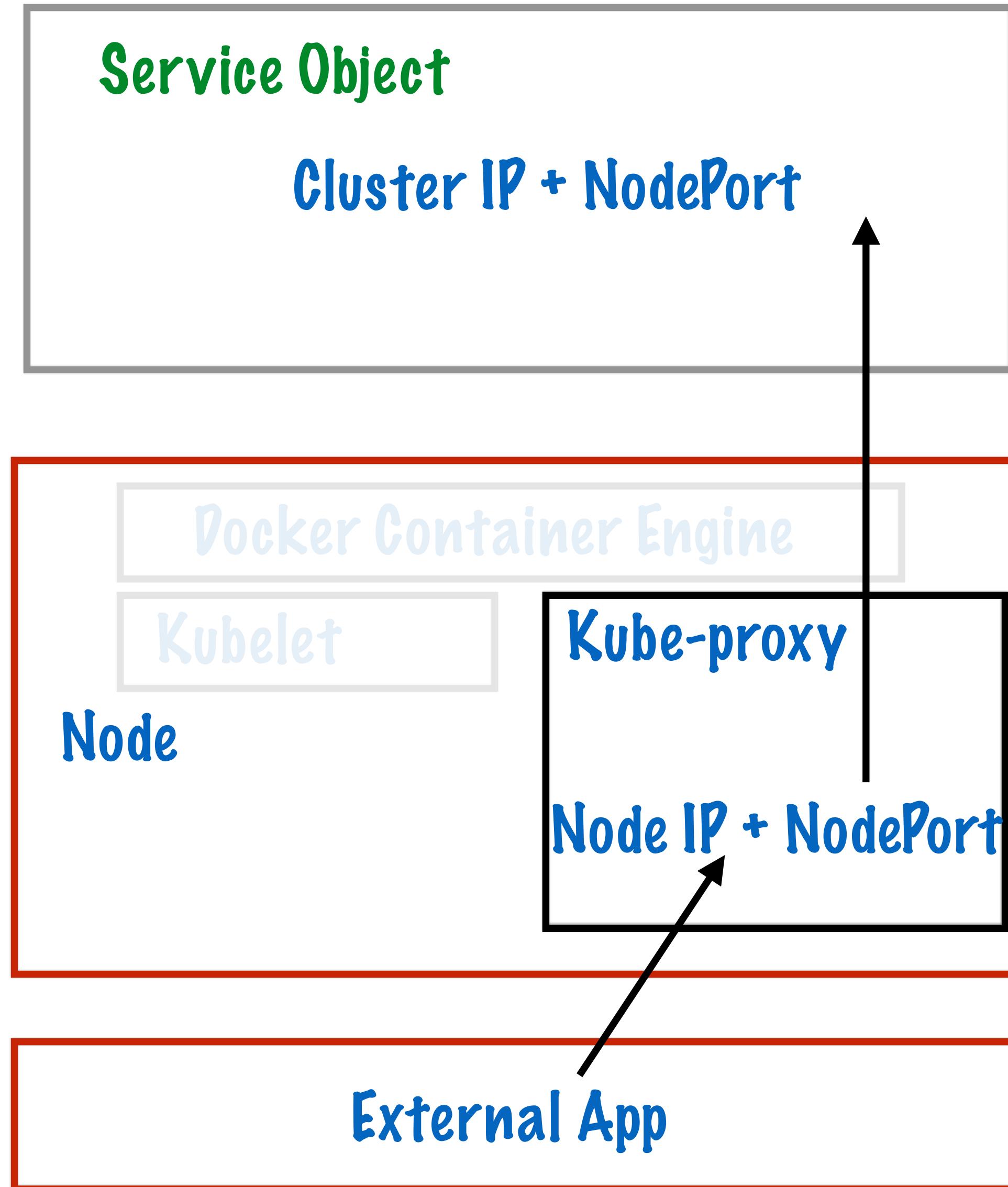
Kube-proxy

Needed because pod IP addresses are ephemeral

Networking - will make sense when we discuss Service objects

Kubernetes Master (Control Plane)

Kubernetes Node (Minion)



Kube-proxy

Needed because pod IP addresses are ephemeral

Networking - will make sense when we discuss Service objects

Virtual IP (VIP) Access To Service

- Can be used by clients from outside cluster to access a service object
- Implemented by kube-proxy which runs on each node of cluster
- Each kube-proxy will relay external traffic to correct VIP

What are the types of Service Objects?

Services For Stable IP Addresses

- Service object - load balancer
- Service = Logical set of backend pods + stable front-end
- Front-end: Static IP address + Port + DNS Name
- Back-end: Logical set of backend pods (label selector)

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

ClusterIP Services

- Type #1 - ClusterIP: Static lifetime IP of service
 - Service only accessible within cluster
 - ClusterIP address is independent of backend pods
 - Default type of service
 - Created by default even for NodePort, LoadBalancer service objects

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

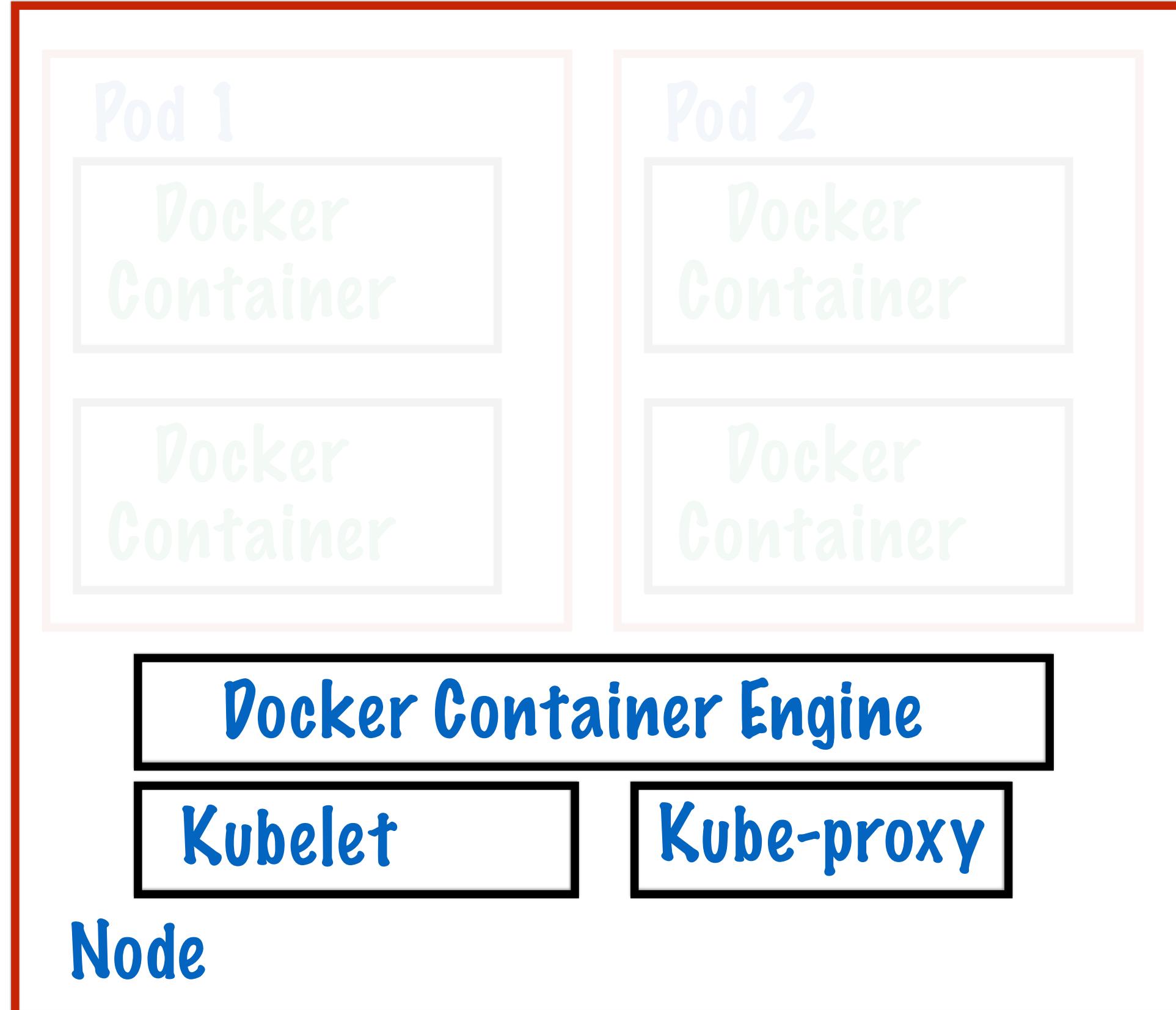
Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

NodePort Services

- Type #1 - ClusterIP: Static lifetime IP of service
 - Only within cluster
- Type #2 - NodePort: Service will also be exposed on each node on static port
 - External clients can hit Node IP + NodePort
 - Request will be relayed to ClusterIP + NodePort

Kubernetes Node (Minion)



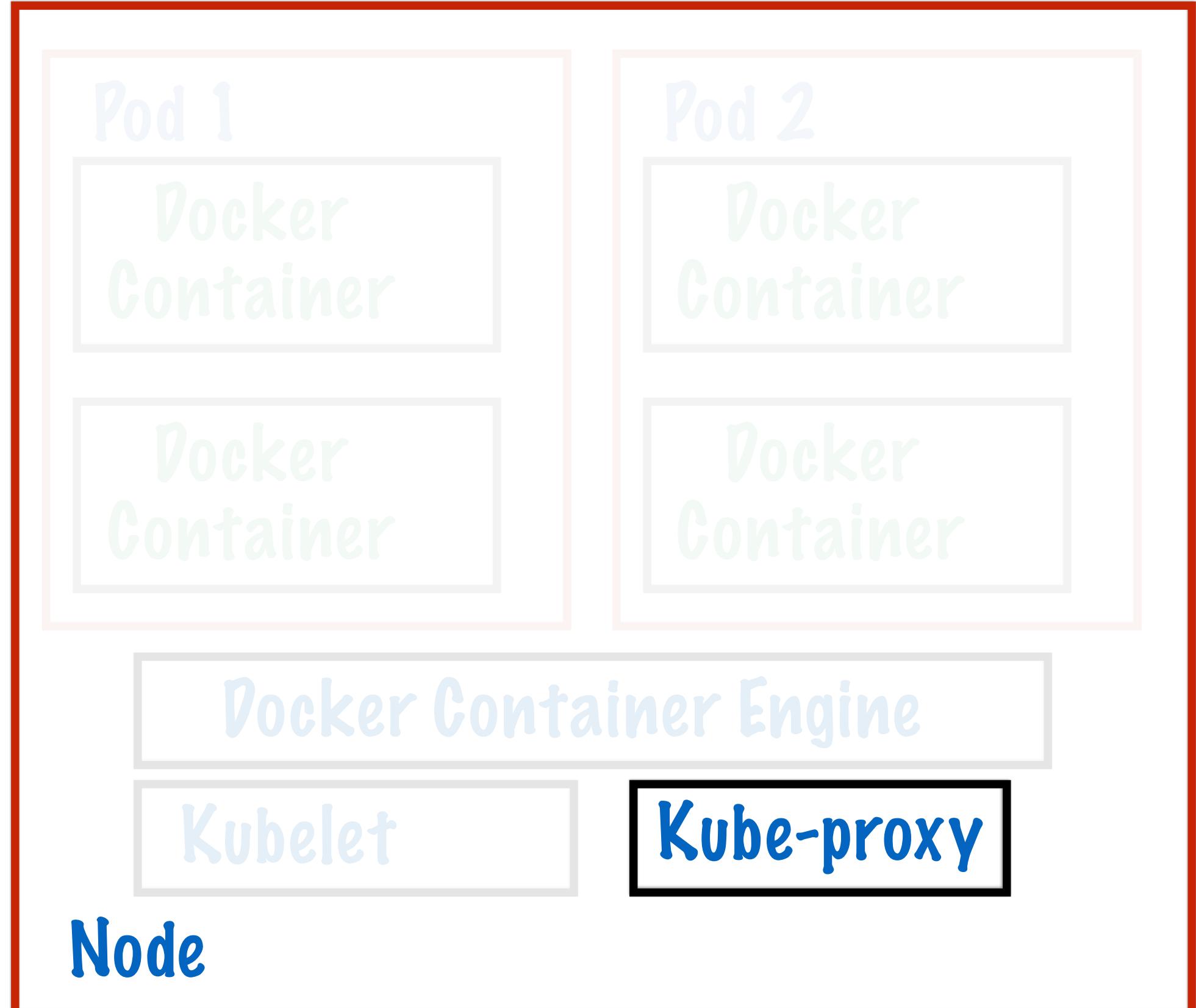
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Kubernetes Node (Minion)



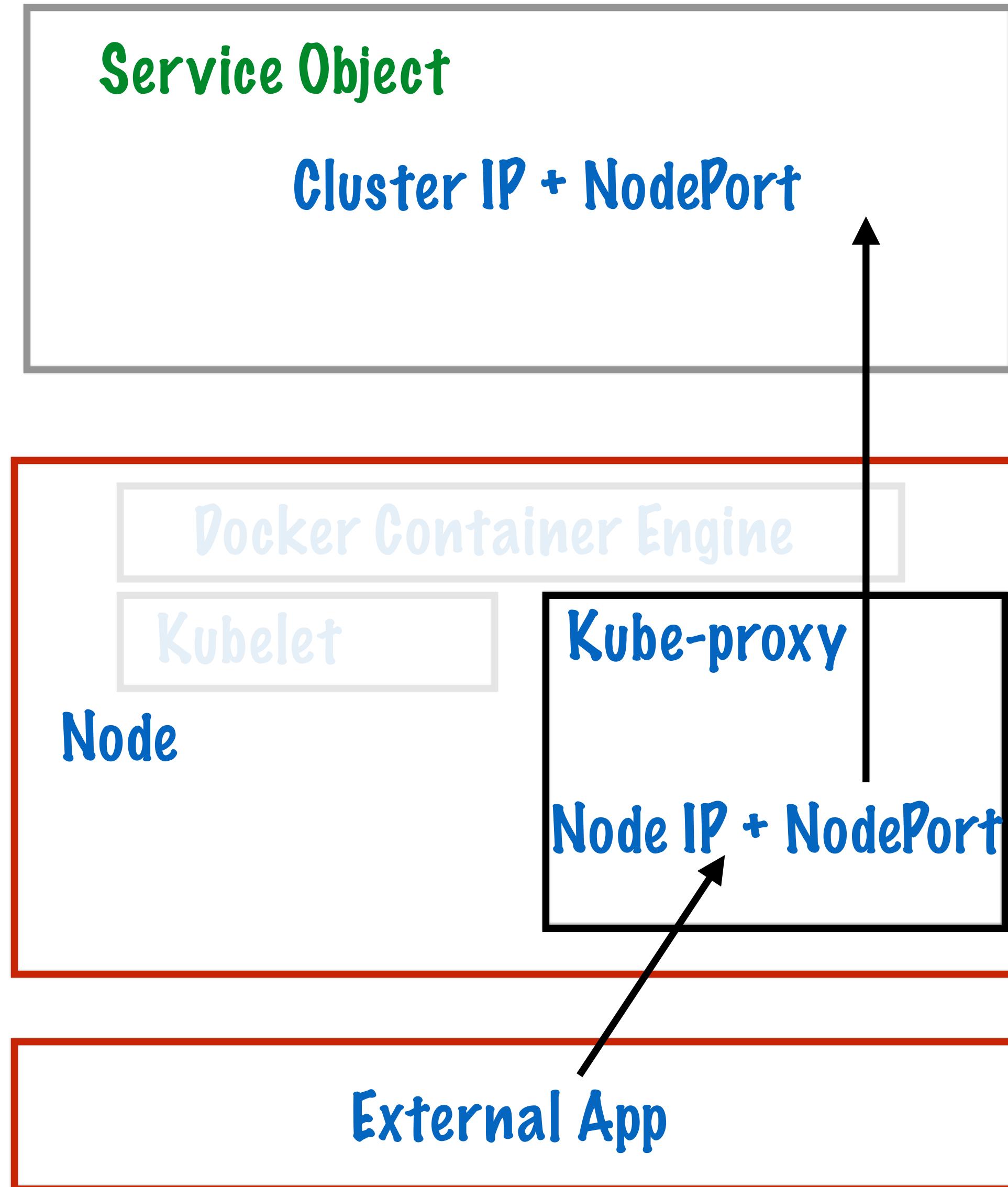
Kube-proxy

Needed because pod IP addresses are ephemeral

Networking - will make sense when we discuss Service objects

Kubernetes Master (Control Plane)

Kubernetes Node (Minion)



Kube-proxy

Needed because pod IP addresses are ephemeral

Networking - will make sense when we discuss Service objects

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

LoadBalancer Services

- Type #3 - LoadBalancer: External loadbalancer object
 - Use LBs provided by AWS, GCP, Azure...
 - Will automatically create NodePort and ClusterIP services under-the-hood
 - External LB -> NodePort -> ClusterIP -> Backend Pod

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- Type #4 - **ExternalName:** Map service to external service residing outside the cluster
 - Can only be accessed via `kube-dns`
 - No selectors in service spec

How do Endpoint Objects Dynamically Update Service Backends?

Networking Pods and Containers

Docker

- Host-private private networking
- Ports must be allocated on node IPs
- Containers need to allocate ports
- Burden of networking lies on containers

Kubernetes

- Pods can always communicate with each other
- Inter-pod communication independent of nodes
- Pods have private IP addresses (within cluster)
- Containers within pod: use localhost
- Containers across pods: pod IP address

Ephemeral Pod IP Addresses

- Containers expose ports in Pod spec
- But Pod IP addresses keep changing!
- For instance, when ReplicaSets or Deployments take pods up/down, IP addresses will change
- How is a client to know how to access the code in a container?

Services For Stable IP Addresses

- Service object solves this problem!
- Service = Logical set of backend pods + stable front-end
- Front-end: Static IP address + Port + DNS Name
- Back-end: Logical set of backend pods (label selector)

ClusterIP

- When service object created, ClusterIP is assigned
- Tied to service object through lifetime
- Independent of lifespan of any backend pod
- Any other pods can talk to ClusterIP and always access backend
- Service object also has a static port assigned to it

Pod Selector

Service Object

```
selector:  
  matchLabels:  
    tier: frontend  
  matchExpressions:  
    - {key: tier, operator: In, values: [frontend]}
```

Pod 1

Labels

```
{  
  tier: frontend,  
  env: prod,  
  geo: india  
}
```

Pod 2

Labels

```
{  
  tier: frontend,  
  env: dev,  
  geo: india  
}
```

Pod 3

Labels

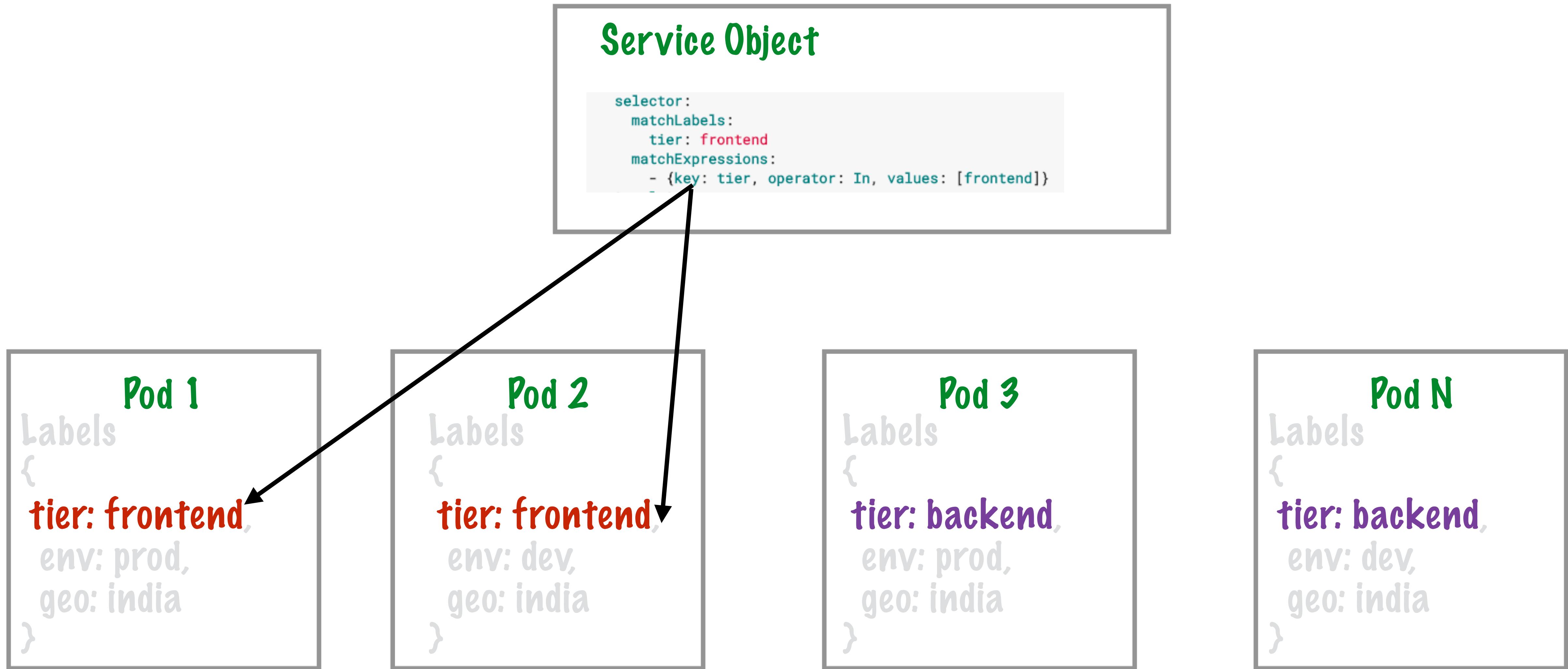
```
{  
  tier: backend,  
  env: prod,  
  geo: india  
}
```

Pod N

Labels

```
{  
  tier: backend,  
  env: dev,  
  geo: india  
}
```

Pod Selector



Endpoint Object

- Dynamic list of pods that are selected by a Service
- Each service object has an associated endpoint object
- Kubernetes evaluates service label selector vs. all pods in cluster
- Dynamic list is updated as pods are created/deleted

When Might Services Omit A Selector?

Pod Selector

Service Object

```
selector:  
  matchLabels:  
    tier: frontend  
  matchExpressions:  
    - {key: tier, operator: In, values: [frontend]}
```

Pod 1

Labels

```
{  
  tier: frontend,  
  env: prod,  
  geo: india  
}
```

Pod 2

Labels

```
{  
  tier: frontend,  
  env: dev,  
  geo: india  
}
```

Pod 3

Labels

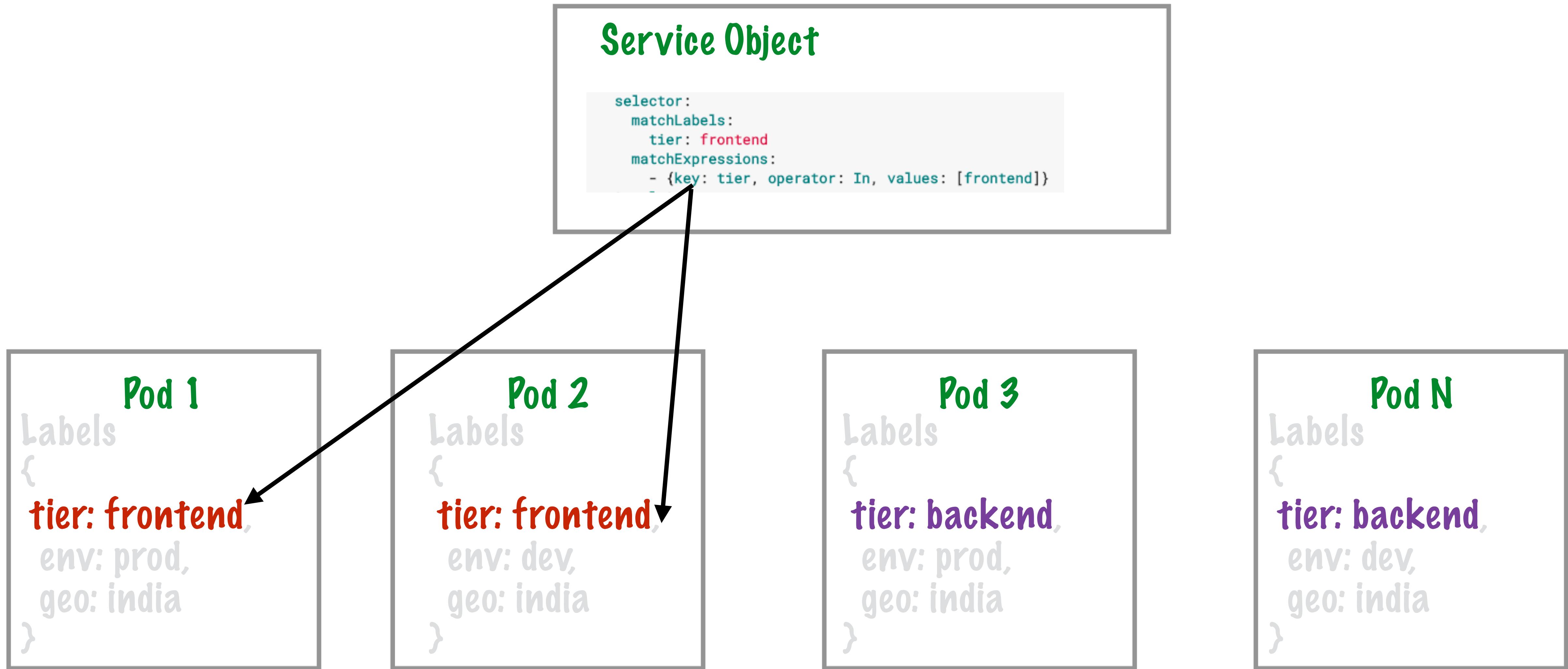
```
{  
  tier: backend,  
  env: prod,  
  geo: india  
}
```

Pod N

Labels

```
{  
  tier: backend,  
  env: dev,  
  geo: india  
}
```

Pod Selector



Endpoint Object

- Dynamic list of pods that are selected by a Service
- Each service object has an associated endpoint object
- Kubernetes evaluates service label selector vs. all pods in cluster
- Dynamic list is updated as pods are created/deleted

Omitting A Selector

- Use when your backend is not pods in this cluster
 - e.g. external database
 - service in another cluster
 - migrating from on-premise, some backends not on K8S

No Selector -> No Endpoint Object

- No endpoint object created
 - Need to manually map the service to specific IP or address
- ExternalName service: this is a service with no selector, no port
 - alias to external service in another cluster

How Does The kube-proxy Implement Virtual IPs for Services?

Services For Stable IP Addresses

- Service object - load balancer
- Service = Logical set of backend pods + stable front-end
- Front-end: Static ClusterIP address + Port + DNS Name
- Back-end: Logical set of backend pods (label selector)

Services For Stable IP Addresses

From Within Cluster

Endpoint object

Dynamic list of pods

Based on label selection

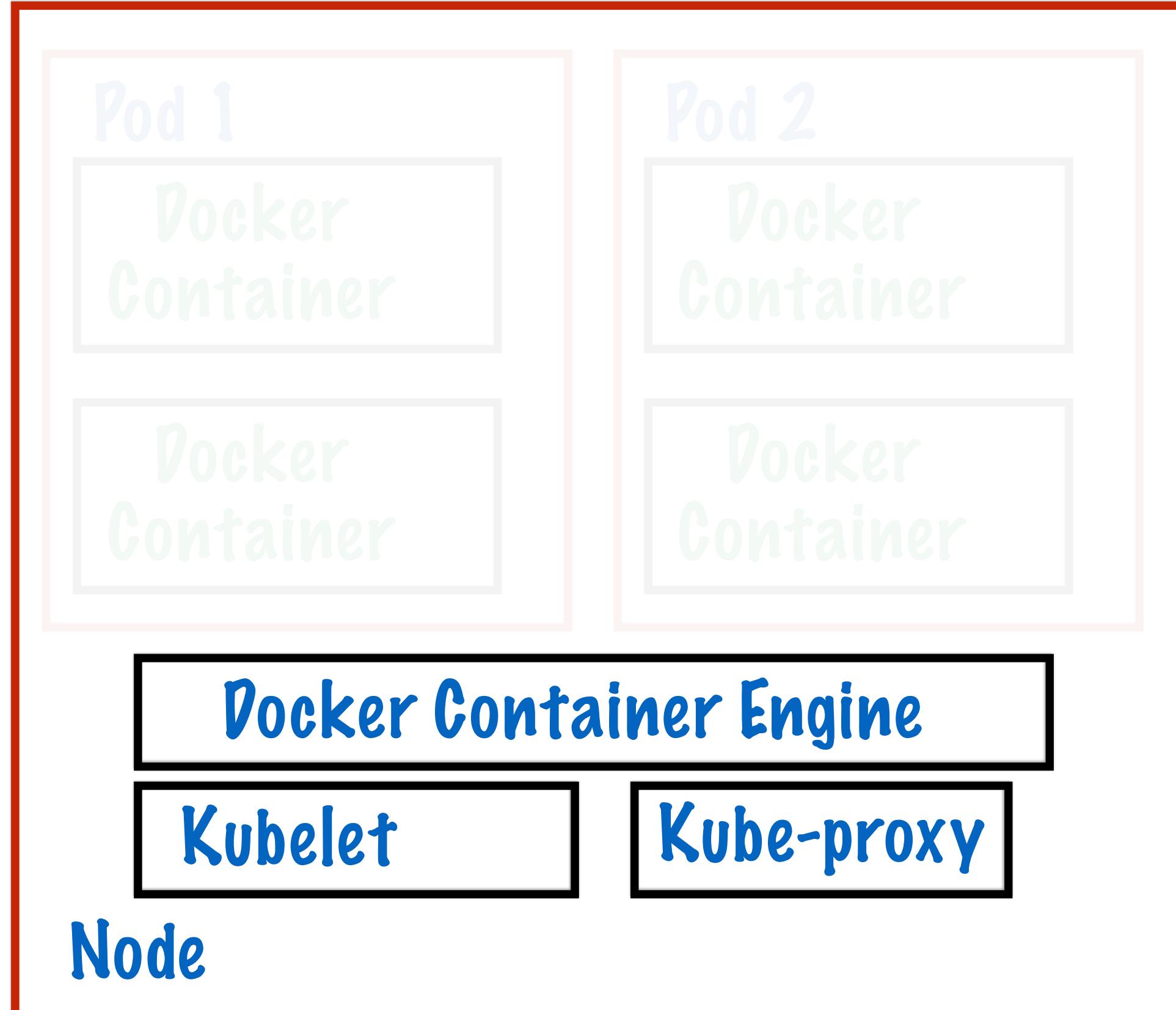
From Outside Cluster

Virtual IP

Can be accessed via any Node IP

Node will relay to ClusterIP

Kubernetes Node (Minion)



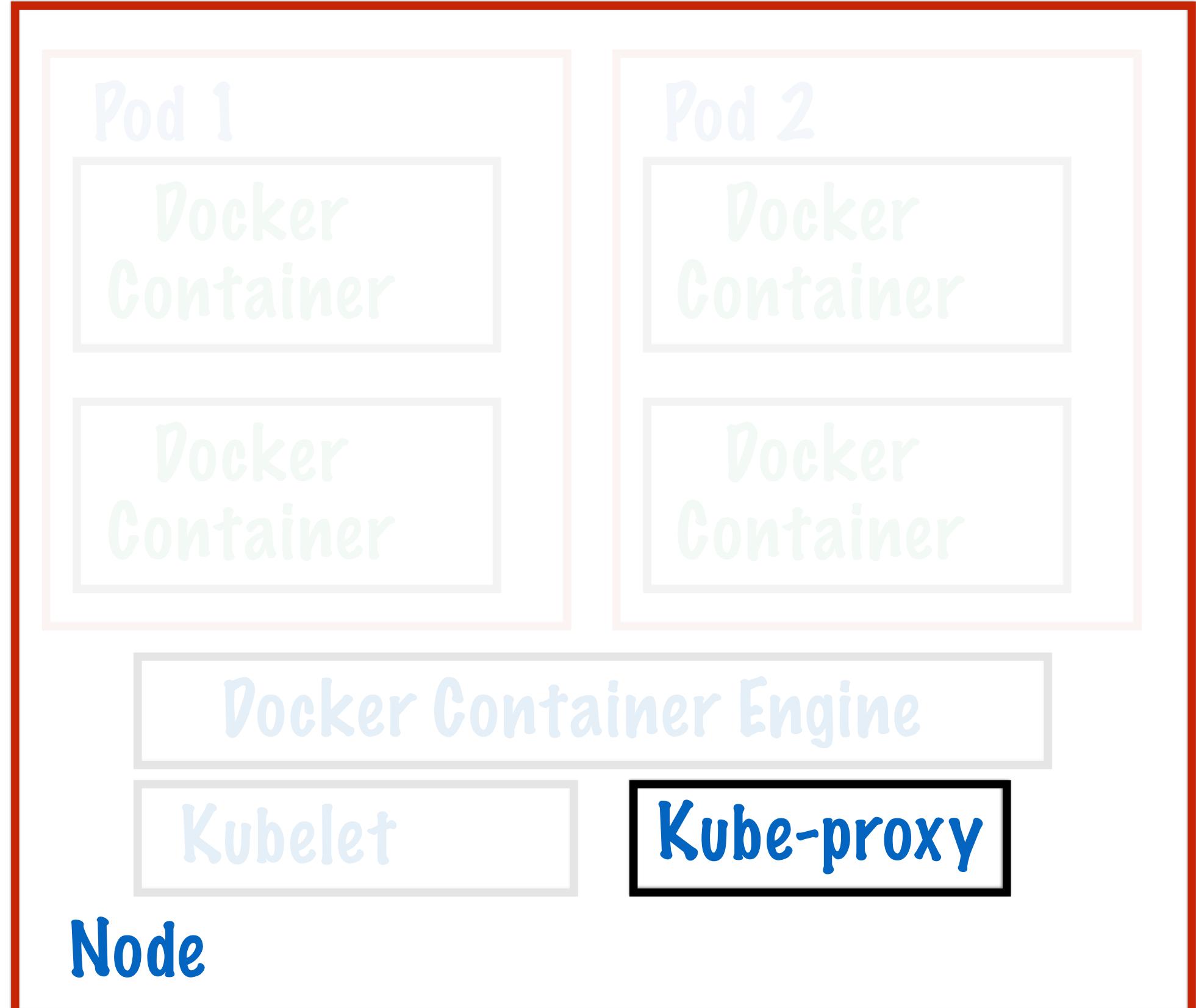
Kubelet

Kube-proxy

Container engine

Kubernetes Master (Control Plane)

Kubernetes Node (Minion)



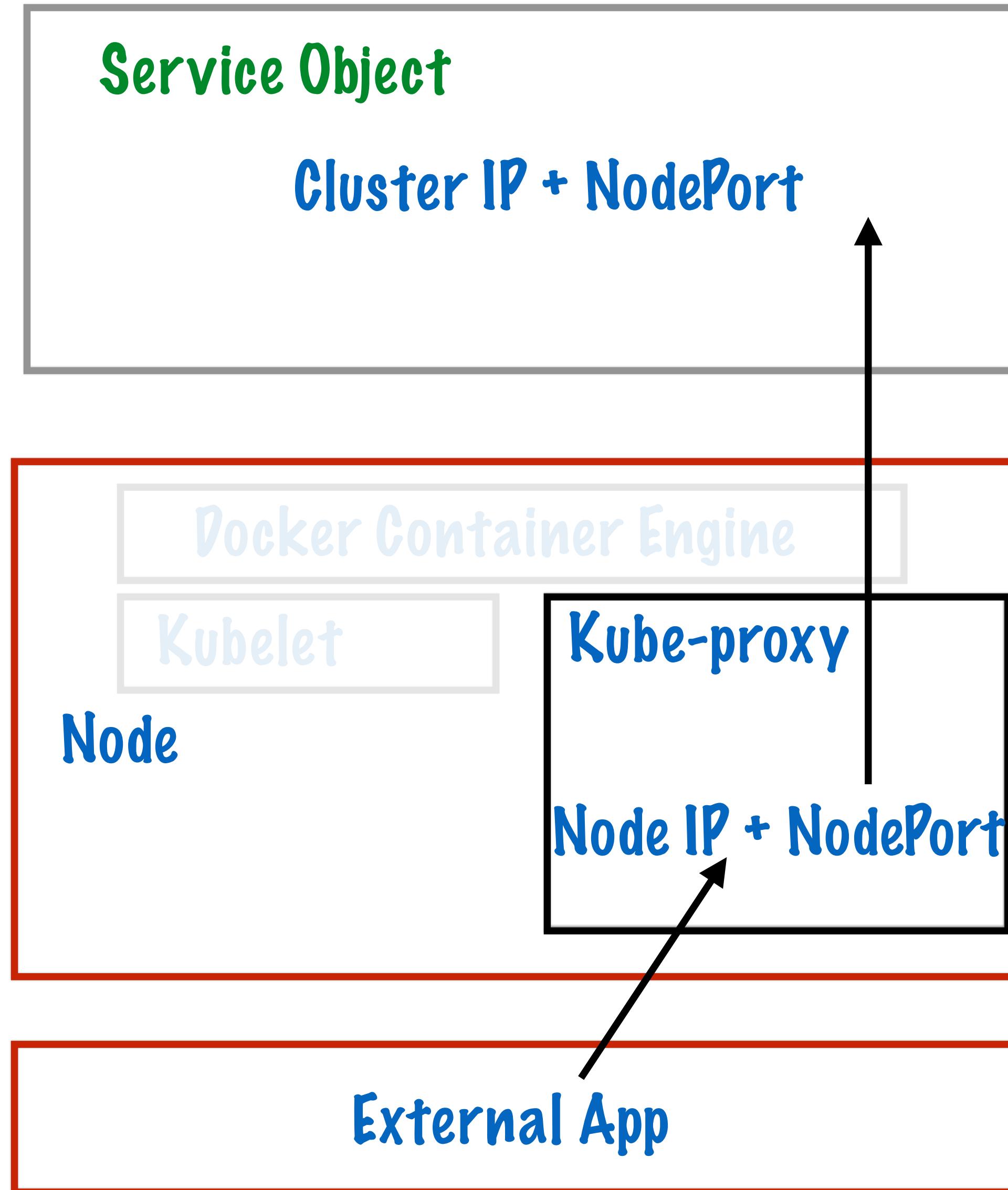
Kube-proxy

Needed because pod IP addresses are ephemeral

Networking - will make sense when we discuss Service objects

Kubernetes Master (Control Plane)

Kubernetes Node (Minion)



Kube-proxy

Needed because pod IP addresses are ephemeral

Networking - will make sense when we discuss Service objects

Evolution of kube-proxy

- Back in the day (v1.0)
 - Services were layer 4 construct
 - Proxy-mode: userspace
- In v1.1
 - Ingress API makes services layer 7 (HTTP)
 - Proxy-mode: iptables
- In v1.2: Proxy-mode: iptables became default
- In v1.8.0, ipvs proxy was added

Are Multi-Port Services Allowed?

Multi-Port Services

- Simply add multiple ports in the service spec
- Each port must be named
 - will have DNS SRV record

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
    - name: https
      protocol: TCP
      port: 443
      targetPort: 9377
```

How Does Service Discovery Happen?

Service Discovery

- Say a pod knows it needs to access some service
- How do containers in that pod actually go about doing so?
- This is called Service Discovery
- Two methods
 - DNS lookup: Preferred
 - Environment Variables

DNS Service Discovery

- (Requires dns add-on)
- DNS server listens on creation of new services
- When new service object created, DNS records created
- All nodes can resolve service using name alone

DNS Service Discovery of ClusterIP

- Service name: my-service, Namespace: my-namespace
- Pods in my-namespace: simply DNS name lookup my-service
- Pods in other namespaces: DNS name lookup my-service.my-namespace
- DNS Name Lookup will return ClusterIP of service

DNS Service Discovery Of Service Port

- DNS SRV records for named ports
- Given port name, client performs DNS SRV query
- Port number is returned
- This way - can lookup ClusterIP + ServicePort

Service Discovery

DNS Lookup

- Dynamic
- Preferred
- Requires DNS add-on

Environment Variables

- Static
- Kubelet configures env variables for containers
- Each service has environment variables for
 - host
 - port
- Static - not updated after pod creation

What Is A Headless Service?

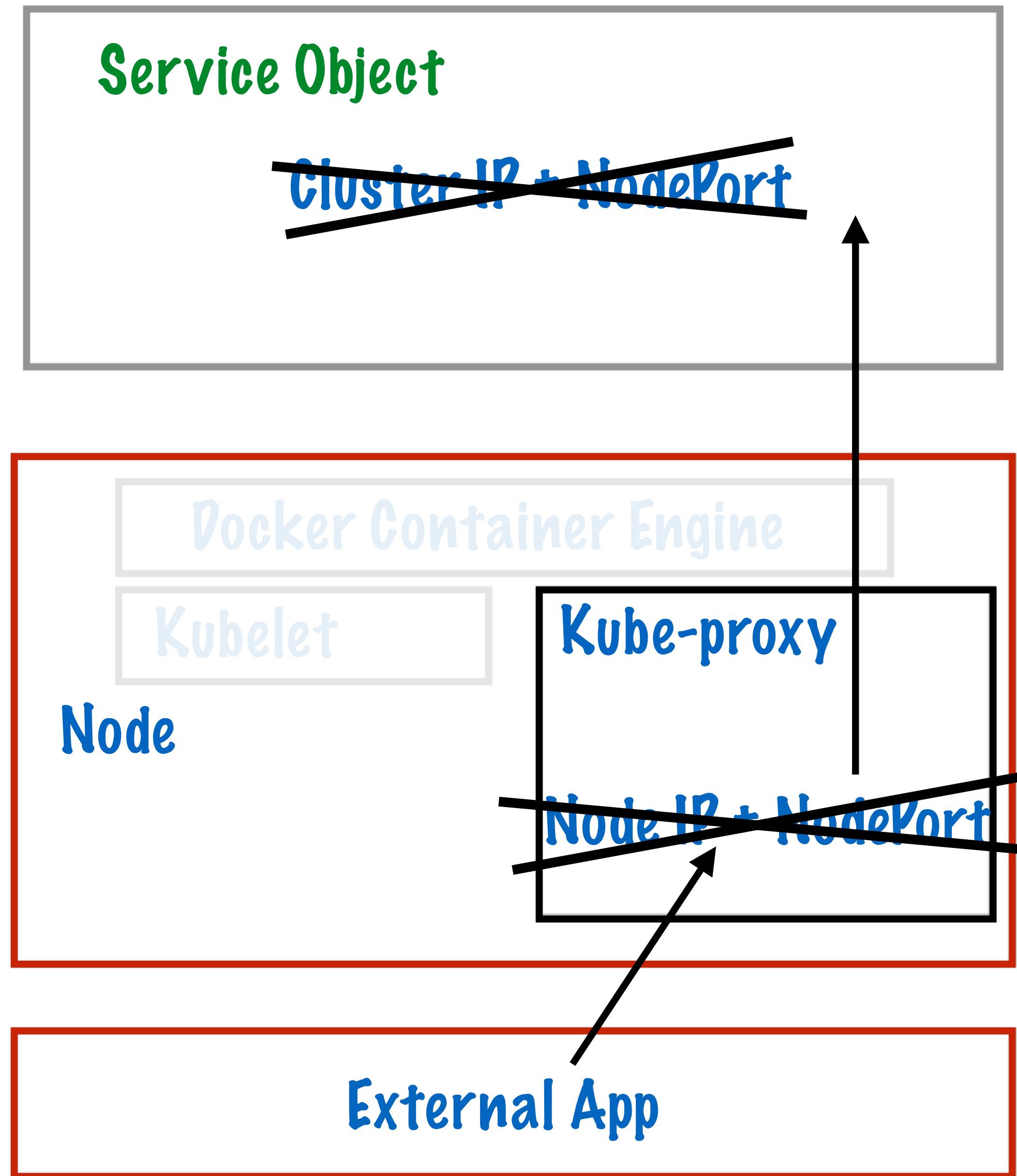
ClusterIP Services

- Type #1 - ClusterIP: Static lifetime IP of service
 - Service only accessible within cluster
 - ClusterIP address is independent of backend pods
 - Default type of service
 - Created by default even for NodePort, LoadBalancer service objects

Headless Service

- Service without Cluster IP = Headless service
- Use if you don't need
 - load balancing
 - cluster IP

Kube-Proxy Unaware Of Headless Service



Kube-proxy

Can not forward nodeport
to headless service

(There is no ClusterIP)

Selector?

- Headless with selector? Associate with pods in this cluster
- Headless without selector? Forward to ExternalName services
 - resolution for services in another cluster

What Are NodePort Services?

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

ClusterIP Services

- Type #1 - ClusterIP: Static lifetime IP of service
 - Service only accessible within cluster
 - ClusterIP address is independent of backend pods
 - Default type of service
 - Created by default even for NodePort, LoadBalancer service objects

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

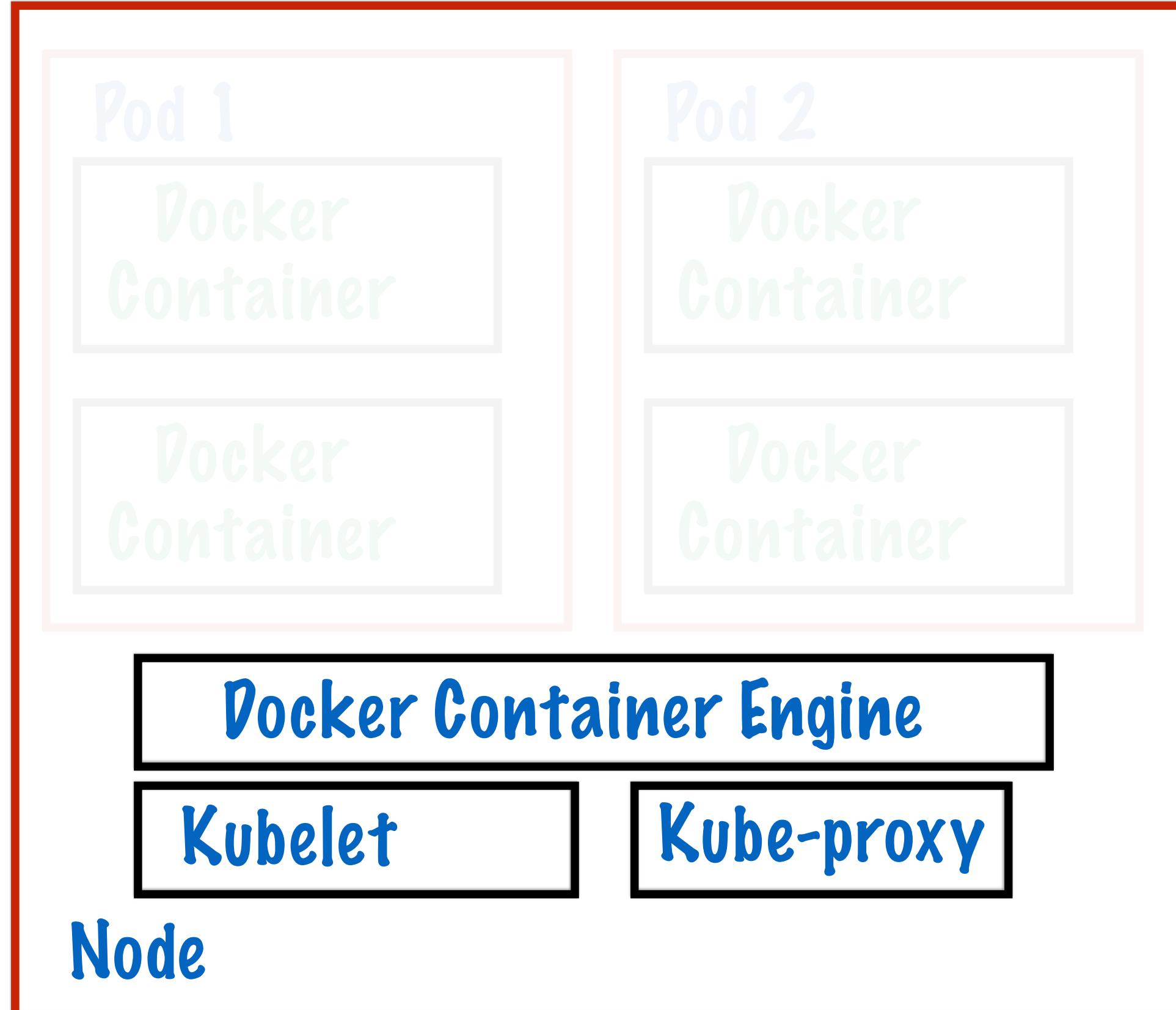
Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

NodePort Services

- Type #1 - ClusterIP: Static lifetime IP of service
 - Only within cluster
- Type #2 - NodePort: Service will also be exposed on each node on static port
 - External clients can hit Node IP + NodePort
 - Request will be relayed to ClusterIP + NodePort

Kubernetes Node (Minion)



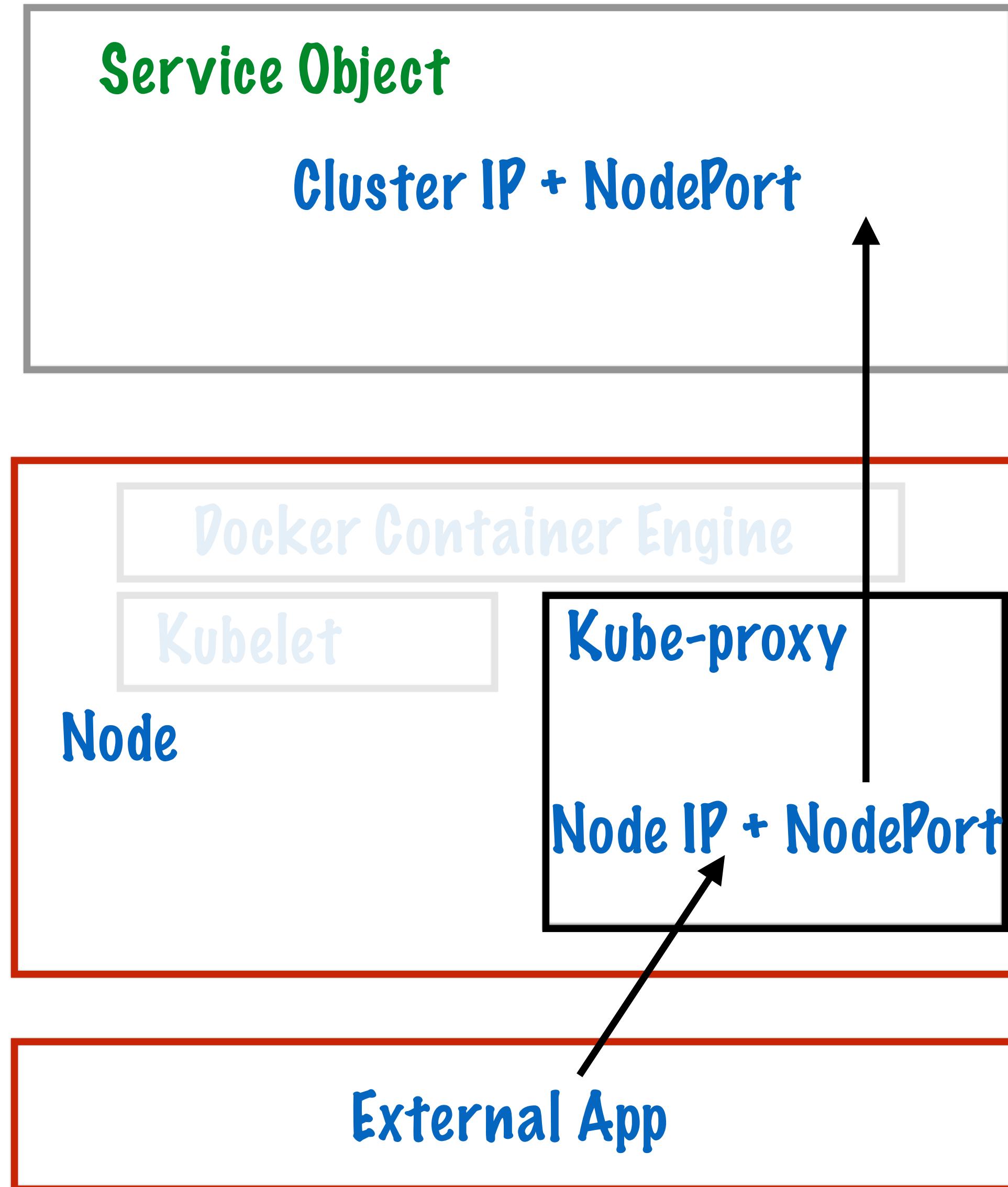
Kubernetes Master (Control Plane)

Kubelet

Kube-proxy

Container engine

Kubernetes Node (Minion)



Kube-proxy

Needed because pod IP addresses are ephemeral

Networking - will make sense when we discuss Service objects

What is a LoadBalancer Service?

Services For Cluster on the Cloud

- Two options:
- `Service.type = LoadBalancer`
- `Ingress Object`

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

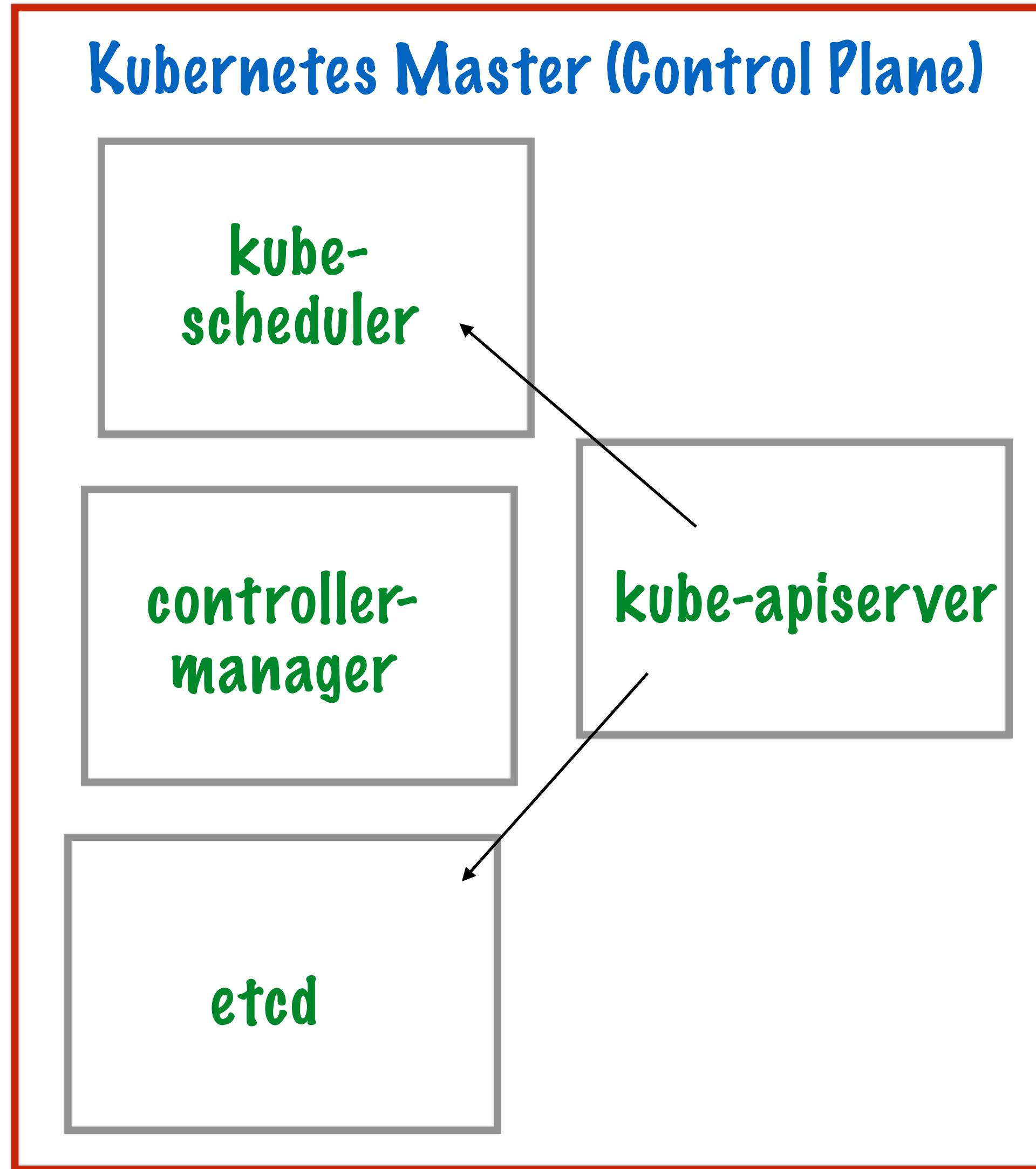
LoadBalancer Services

- Type #3 - LoadBalancer: External loadbalancer object
 - Use LBs provided by AWS, GCP, Azure...
 - Will automatically create NodePort and ClusterIP services under-the-hood
 - External LB -> NodePort -> ClusterIP -> Backend Pod

LoadBalancer Services

- Implementation tightly coupled to external cloud provider
- Cloud-controller-manager inside control plane

Control Plane



Apiserver

etcd

scheduler

controller-manager

cloud-controller-manager

kube-controller-manager

cloud-controller-manager

- Node Controller
- Route Controller
- Service Controller: Big role to play in loadbalancer service
- Volume Controller

External Load Balancers

- External load balancer will be provisioned
 - Asynchronously
 - IP address will be available in status
- Load Balancer IP: May be provided by cloud provider
 - May or may not be able to specify in spec
 - e.g. in Azure, public static IP must be created first

More Fine Print

- Internal load balancers
- SSL Support
- Proxy protocol support
- Log access
- Connection Draining
- Network load balancing

How Are External IPs Specified For Services?

External IPs for Services

- Can specify external IP to service
- Specify external IP in service spec
- Can do with any kind of service
- Incoming traffic to this IP will be routed

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 9376
  externalIPs:
  - 80.11.12.10
```

How Does DNS Service Discovery Happen?

Service Discovery

DNS Lookup

- Dynamic
- Preferred
- Requires DNS add-on

Environment Variables

- Static
- Kubelet configures env variables for containers
- Each service has environment variables for
 - host
 - port
- Static - not updated after pod creation

DNS Service Discovery of ClusterIP

- When service created, DNS server creates name
- Service name: my-service, Namespace: my-namespace
- Pods in my-namespace: simply DNS name lookup my-service
- Pods in other namespaces: DNS name lookup my-service.my-namespace
- DNS Name Lookup will return ClusterIP of service

Headless or ClusterIP?

- Headless service: No clusterIP
 - DNS record exists
 - Resolves to set of pods that constitute the service
- Non-Headless (ClusterIP exists)
 - my-svc.my-namespace.svc.cluster.local
resolves to the cluster IP of the Service.

DNS Service Discovery Of Service Port

- DNS SRV records for named ports
 - port-name.port-protocol.my-svc.my-namespace.svc.cluster.local
- Given port name, client performs DNS SRV query
- Port number is returned
- This way - can lookup ClusterIP + ServicePort

How Does DNS Work For Pods?

DNS Records for Pods

- Pods are assigned DNS records of form
 - `pod-ip-address.my-namespace.pod.cluster.local`
- Hostname is pod's `metadata.name`
- Subdomains can be used too
 - Pod with `hostname = "foo"`
 - Subdomain = "bar"
 - Namespace = "my-namespace"
 - will have the fully qualified domain name FQDN = "foo.bar.my-namespace.svc.cluster.local"

DNS Policies of Pods

- `spec.dnsPolicy = Default`
- `spec.dnsPolicy = ClusterFirst`
- `spec.dnsPolicy = ClusterFirstWithHostNet`
- `spec.dnsPolicy = None`

DNS Policies of Pods

- `spec.dnsPolicy = Default`
 - **inherits node's name resolution**
- `spec.dnsPolicy = ClusterFirst`
 - **forwarded to upstream name server from node**
- `spec.dnsPolicy = ClusterFirstWithHostNet`
 - **if host network = true, use this. Apps can then see NIC of host machine**
- `spec.dnsPolicy = None`
 - **Pod can ignore entirely all DNS settings from Kubernetes, use DNS config in pod spec instead**

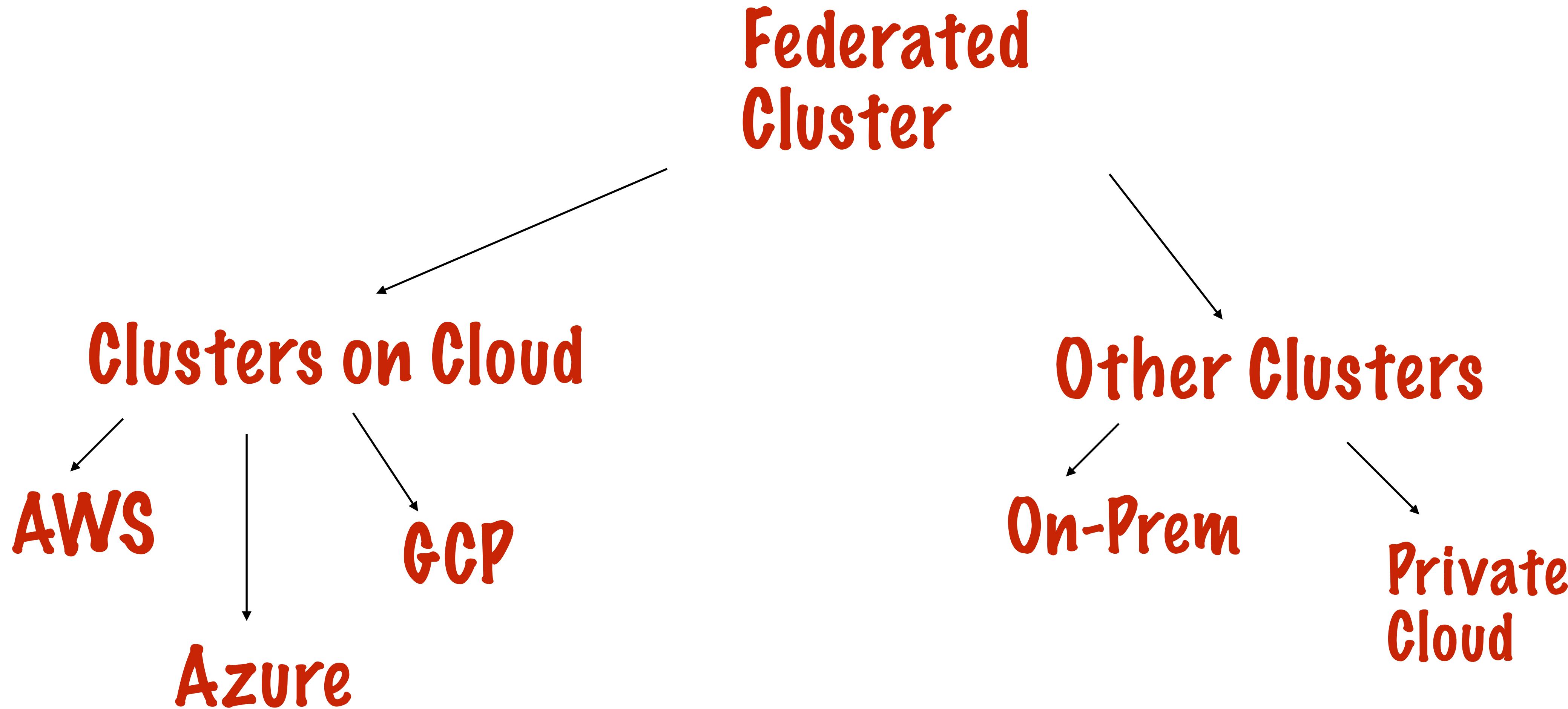
How Are Services Secured?

How Are Services Secured?

- If incoming traffic can reach your service - secure the channel
- Use HTTPs
 - Need a certificate (bought or self-signed)
 - Server configured to use the certificates
 - Also need a secret to make services accessible to pod

What Are Federated Services?

Federated Clusters on Kubernetes



Federated Services

- Combine multiple clusters into a federation (federated cluster)
- Federated services span clusters within federation
- Relevant for hybrid, multi-cloud environments
- `kubectl` for (regular) clusters, `kubefed` for federated clusters

What is an Ingress Object?

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Types of Services

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

LoadBalancer Services

- Type #3 - LoadBalancer: External loadbalancer object
 - Use LBs provided by AWS, GCP, Azure...
 - Will automatically create NodePort and ClusterIP services under-the-hood
 - External LB -> NodePort -> ClusterIP -> Backend Pod

Ingress Objects

- Recent alternative to LoadBalancer objects
- Better performance
- More centralized

Ingress on GKE

- Google Cloud Platform supports both ingress objects as well as load balancers
- On GKE, ingress object is preferred: HTTP load balancing
- On GKE, ingress controller is automatically available
 - Elsewhere, you need to set it up as a pod