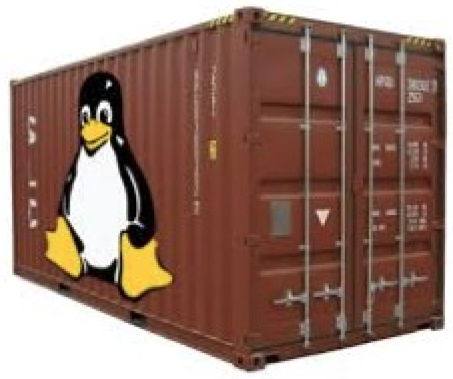
What is container ?

Containers, in short, contain applications in a way that keep them isolated from the host system that they run on. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package



What is container ?

Big internet companies like Google have been utilizing containers for nearly a decade.

The benefits we enjoy include:

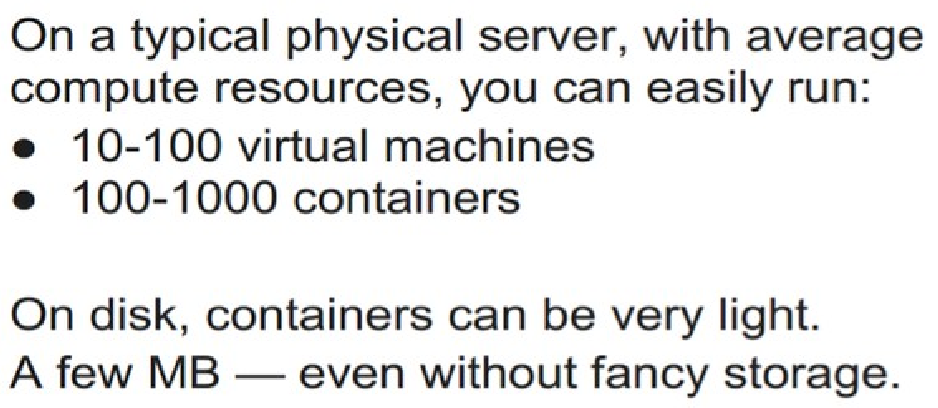
- Very fast provisioning
- Simple, high availability
- Smooth scaling
- Machine-precision consistency
- Better performance

What is container ?

Big internet companies like Google have been utilizing containers for nearly a decade.

The benefits we enjoy include:

- Very fast provisioning
- Simple, high availability
- Smooth scaling
- Machine-precision consistency
- Better performance



On a typical physical server, with average compute resources, you can easily run:

- 10-100 virtual machines
- 100-1000 containers

On disk, containers can be very light.
A few MB — even without fancy storage.

image credit: Jerome Petazzoni from dotCloud)

What is container ?

Everything at Google runs in containers

- Gmail, Web Search, Maps

- Even GCE itself: Vms in containers

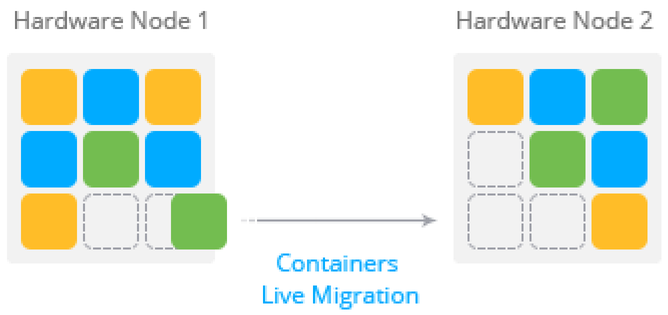
They launch over 2 billion containers per week

What is container ?

Everything at Google runs in containers

- Gmail, Web Search, Maps

- Even GCE itself: Vms in containers



They launch over 2 billion containers per week

The need for container orchestration

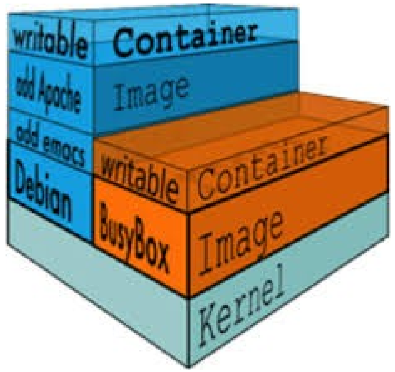
Containers are becoming the standard unit of deployment

Each container image has

- Code
- Binaries
- Configuration
- Libraries
- Frameworks
- Runtime

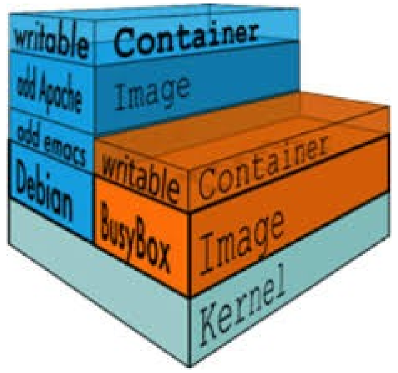
The need for container orchestration

Docker has solved the problem of packaging, deploying and running containerized applications



The need for container orchestration

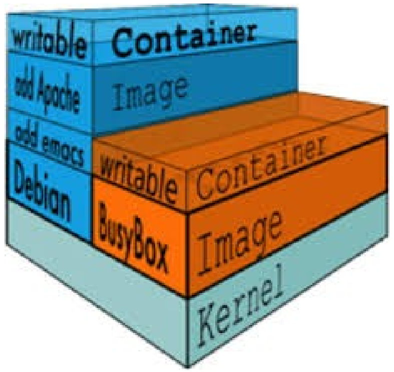
Docker has solved the problem of packaging, deploying and running containerized applications



Docker is great for managing a few containers running on a fewer machines

The need for container orchestration

Docker has solved the problem of packaging, deploying and running containerized applications



Docker is great for managing a few containers running on a fewer machines

Production applications deal with dozens of containers running on hundreds of machines

The need for container orchestration

Tools are evolving to manage the new datacenter infrastructure

- Docker Swarm
- Kubernetes
- Mesosphere DC/OS

Manage the lifecycle of containerized applications running in production

Automate the distribution of applications

Ensure higher levels of utilization and efficiency

What is Kubernetes?

The Kubernetes project was started by Google in 2014.

Kubernetes (from κυβερνήτης: Greek for "helmsman" or "pilot")

Its development and design are heavily influenced by Google's Borg system

- Kubernetes is a platform for hosting Docker containers in clustered environment
- Provides container grouping, load balancing, auto-healing, scaling
- Supports multiple cloud and bare-metal environments easily roll out new versions of application containers

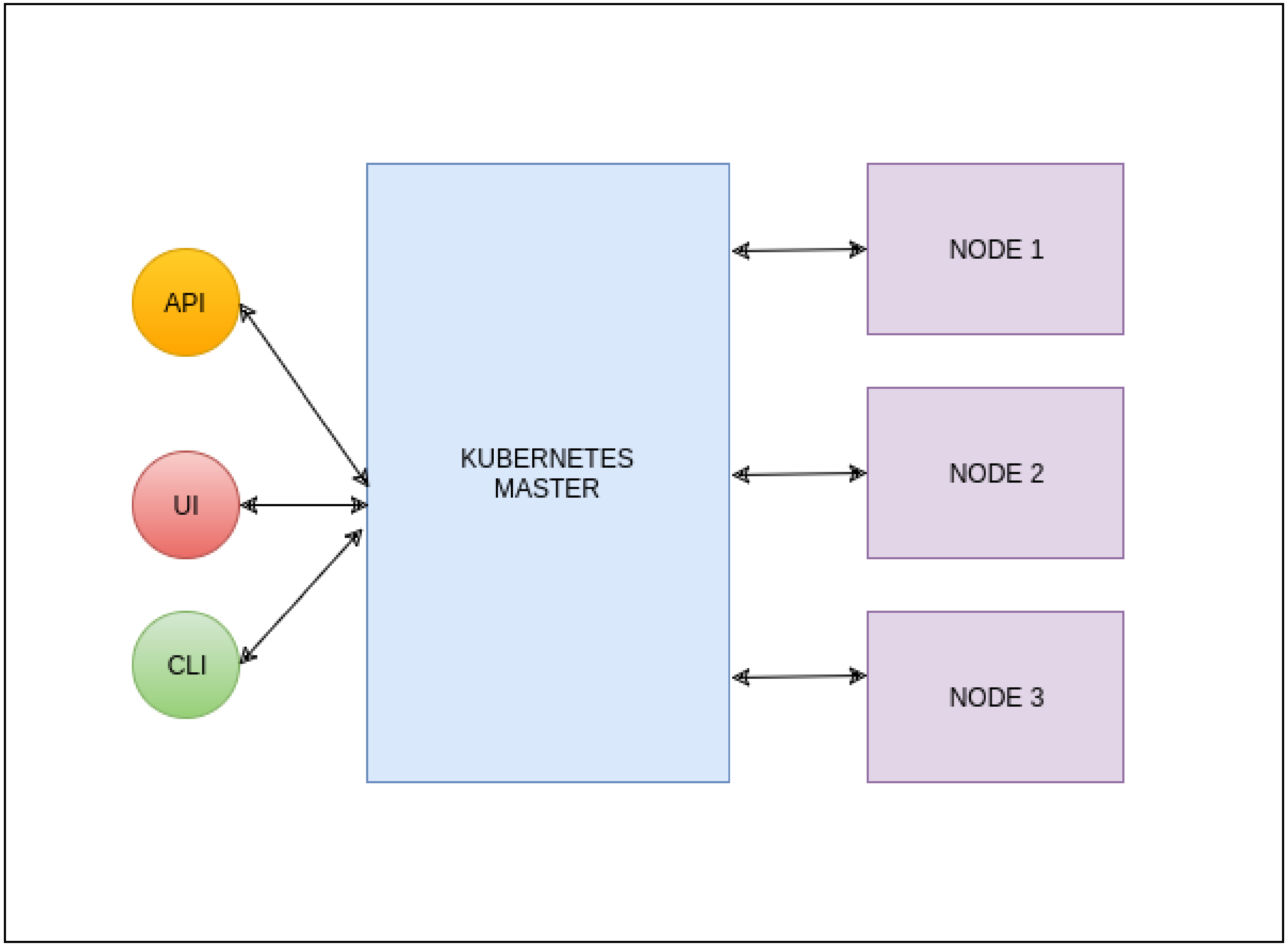
Cluster

A cluster is a group of nodes, they can be physical servers or virtual machines that has the Kubernetes platform installed.

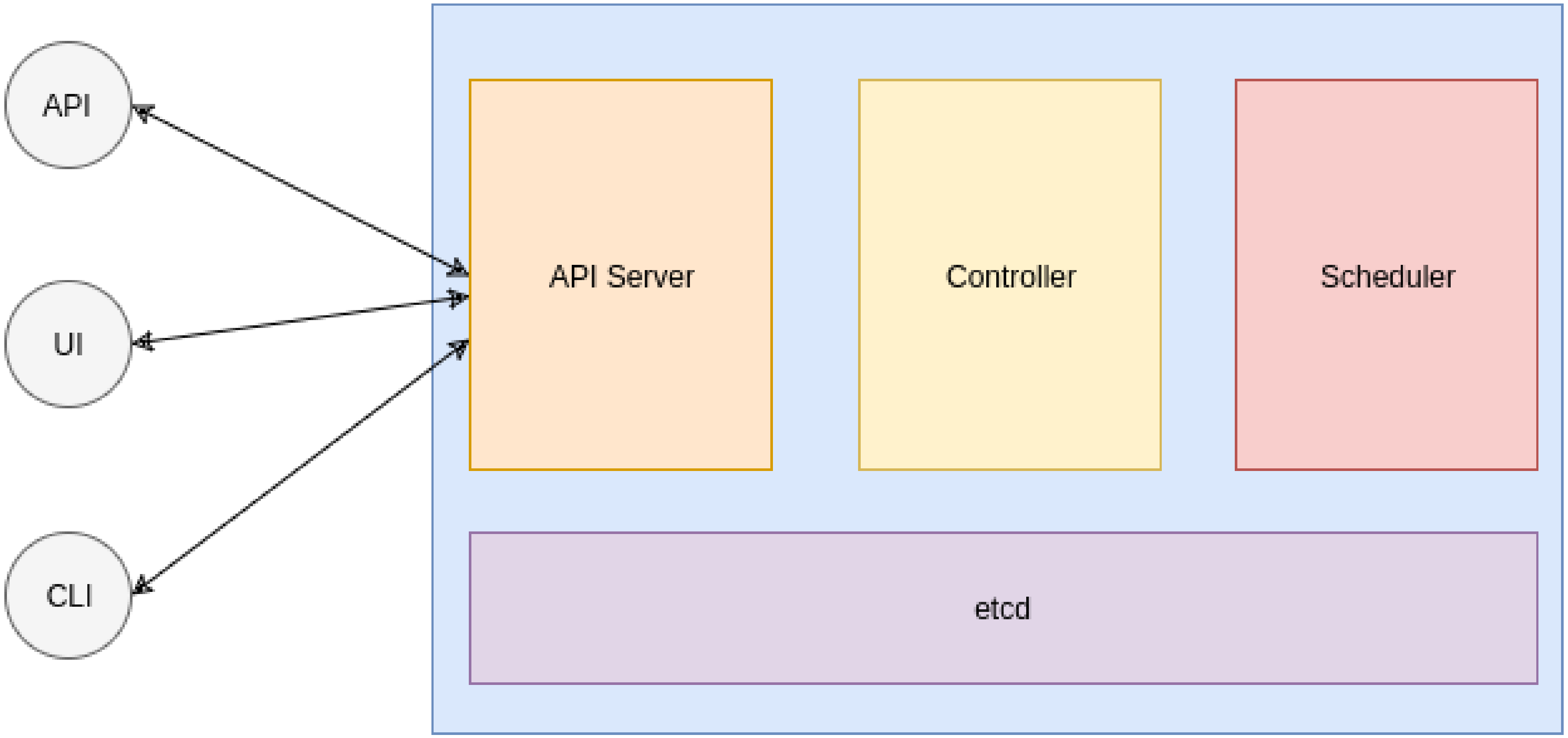
The controlling unit in a Kubernetes cluster is called the master server. It serves as the main management contact point for administrators, and it also provides many cluster-wide systems for the relatively dumb worker nodes.

The master server runs a number of unique services that are used to manage the cluster's workload and direct communications across the system.

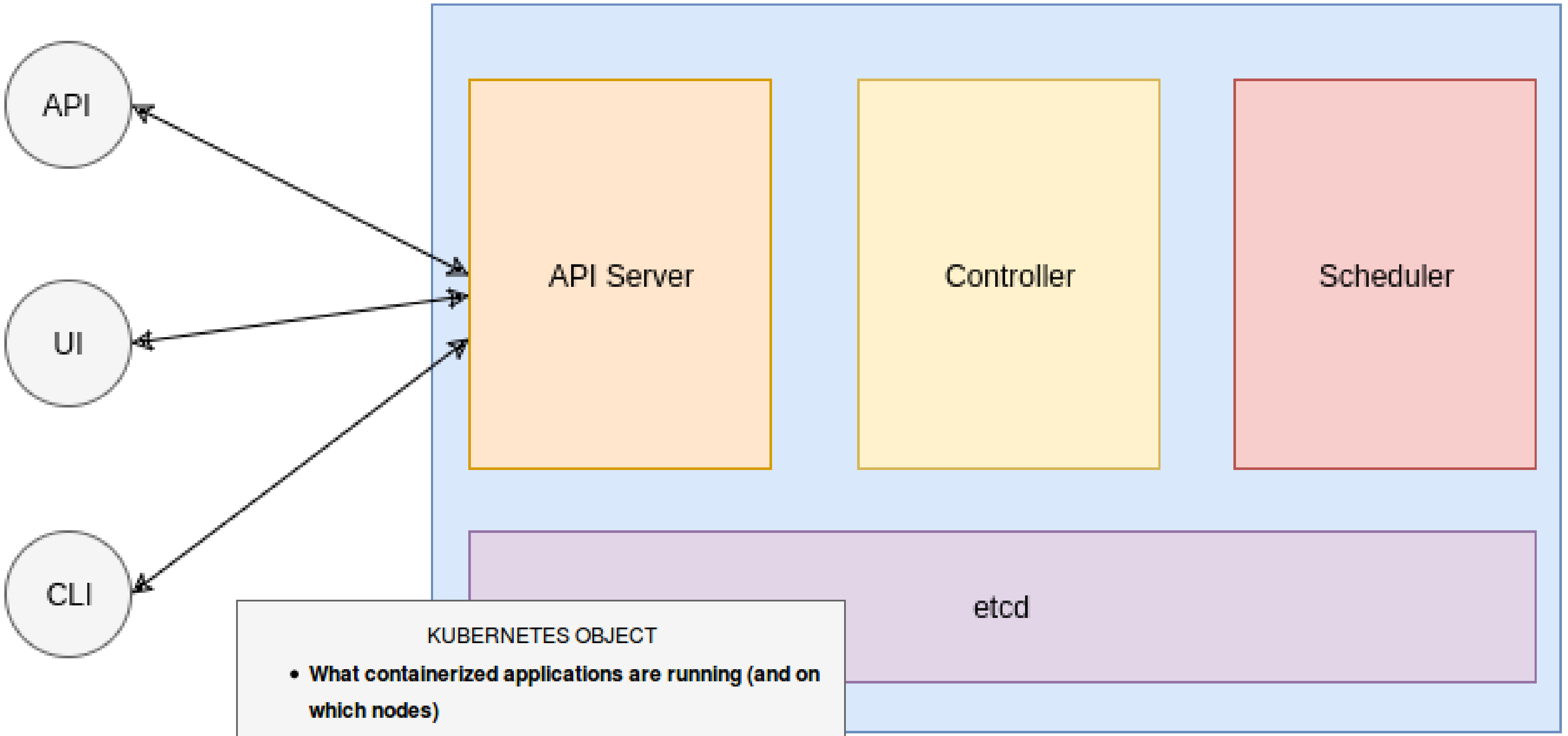
Cluster



Master



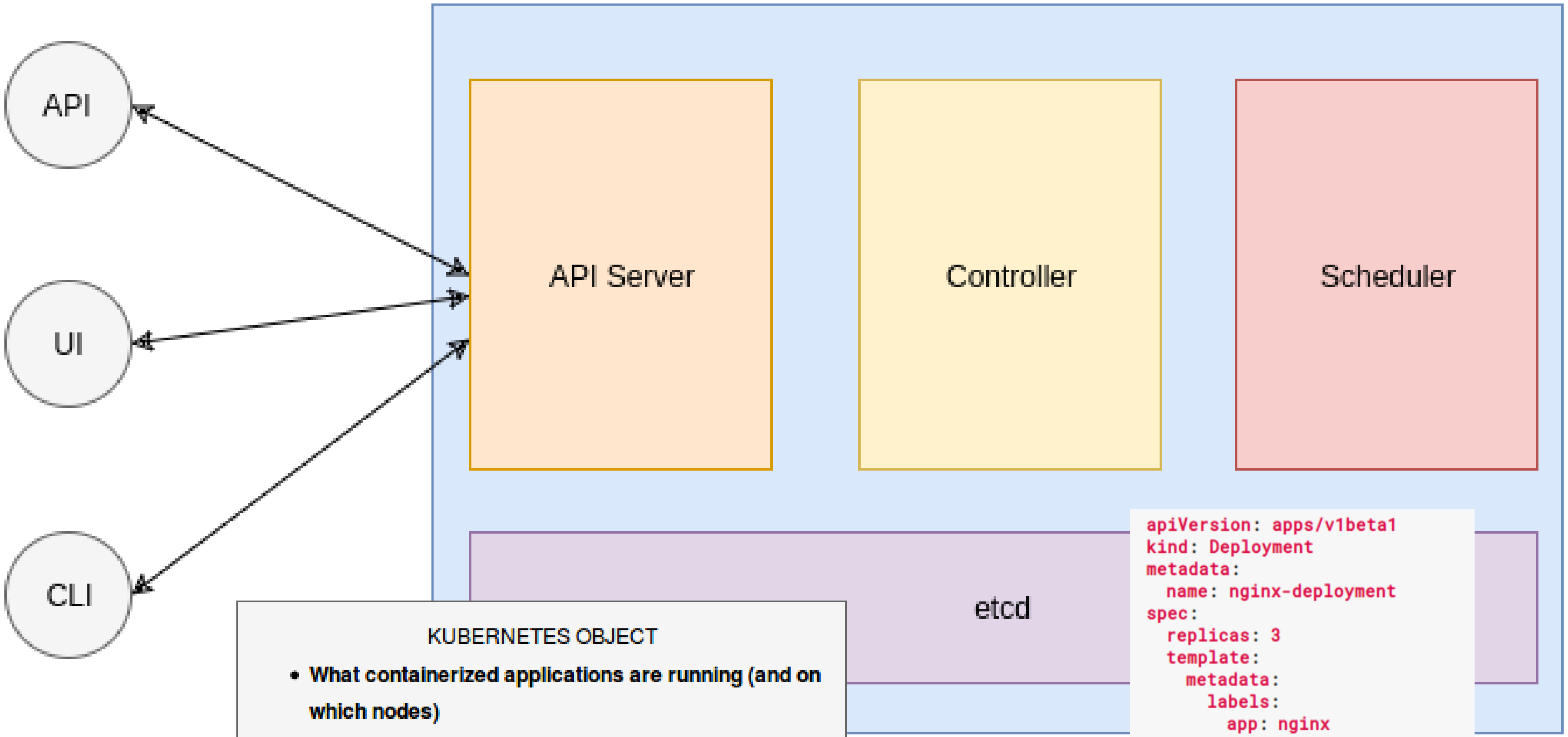
Master



KUBERNETES OBJECT

- What containerized applications are running (and on which nodes)
- The resources available to those applications
- The policies around how those applications behave, such as restart policies, upgrades

Master



KUBERNETES OBJECT

- What containerized applications are running (and on which nodes)
- The resources available to those applications
- The policies around how those applications behave, such as restart policies, upgrades

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

Namespaces

Enable you to manage different environments within the same cluster.

You can create multiple kubernetes objects without worrying about them affecting each other.

A mechanism to attach authorization and policy to a subsection of the cluster.

Kubernetes starts with two initial namespaces:

- **default** The default namespace for objects with no other namespace
- **kube-system** The namespace for objects created by the Kubernetes system

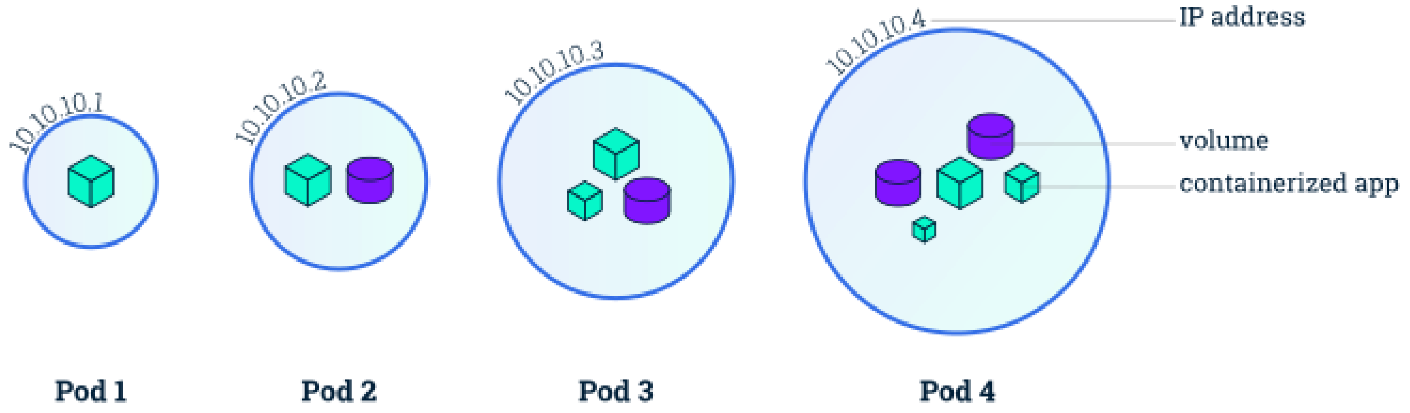
Labels & Selectors

- Key/value pairs associated with Kubernetes objects
- Used to organize and select subsets of objects
- Attached to objects at creation time but modified at any time.
- Labels are the essential glue to associate one API object with other

POD

A group of one or more containers that are always co-located and co-scheduled that share the context

Containers in a pod share the same IP address, ports, hostname and storage



POD

Modeled like a virtual machine -

- Each container represents one process
- Tightly coupled with other containers in the same pod

Pods are scheduled in Nodes

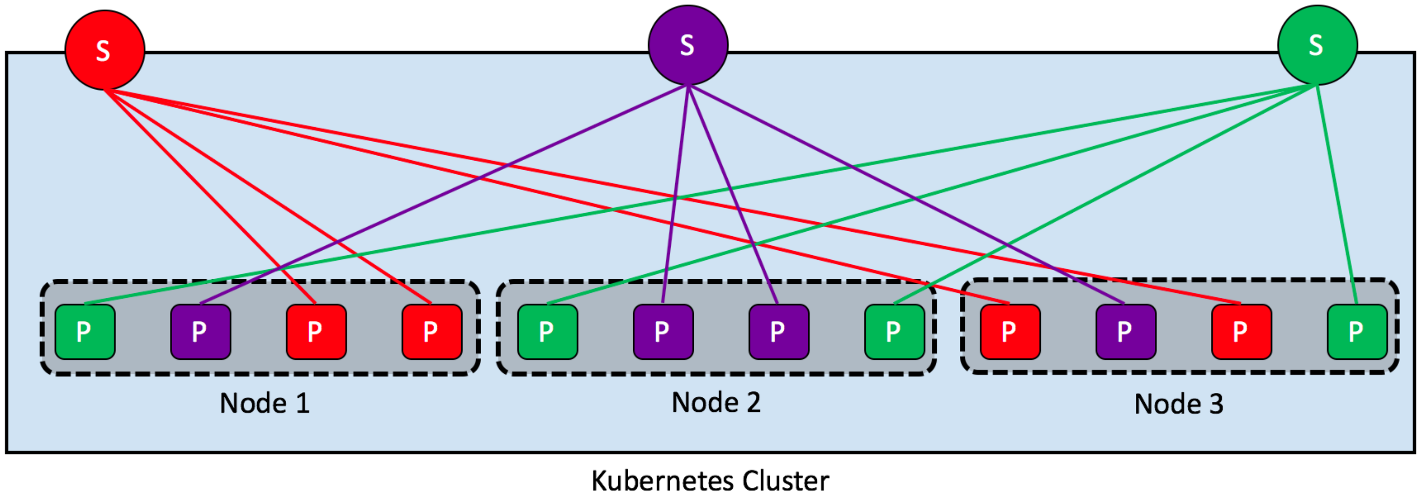
Fundamental unit of deployment in Kubernetes

Replication controller / Replica set

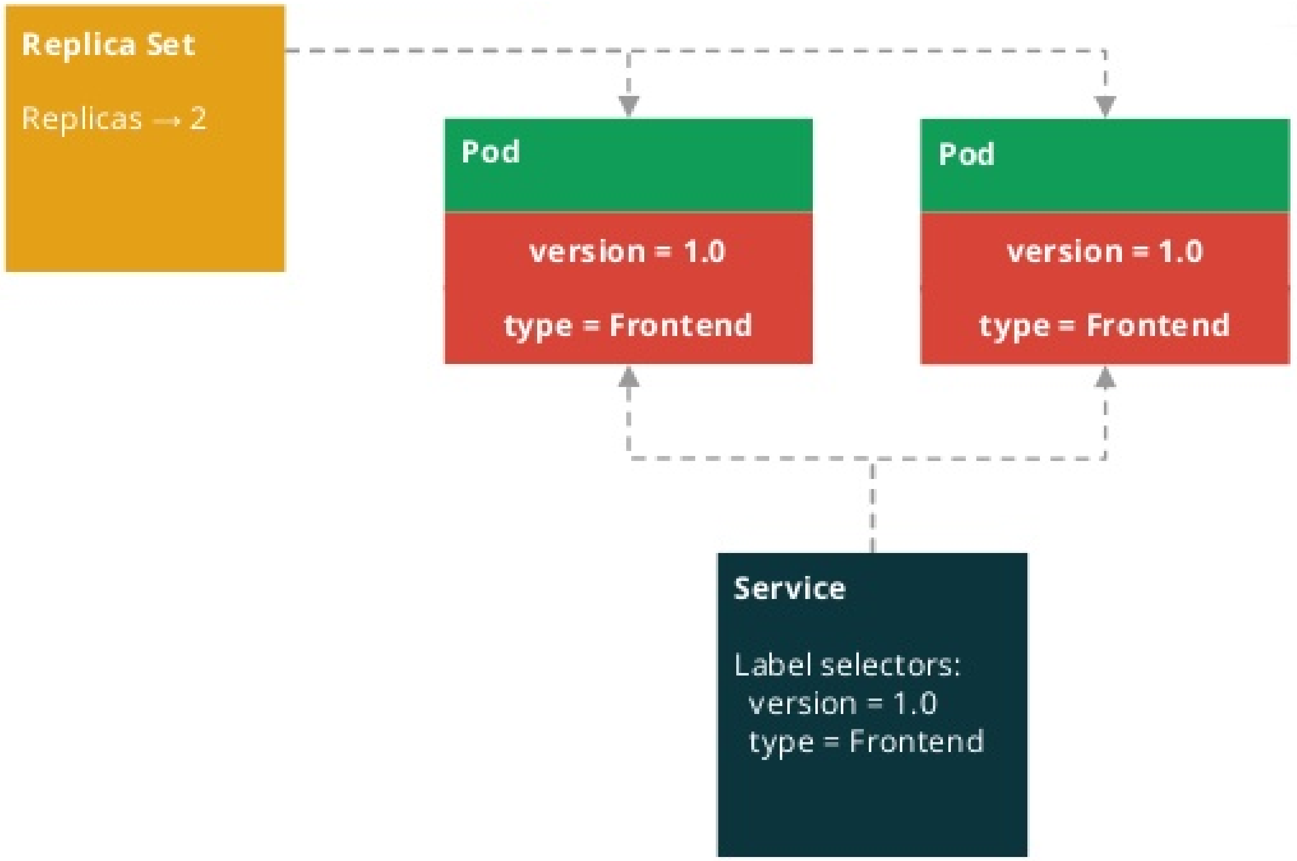
- Ensures that a Pod or set of Pods are always up and available
- Always maintains desired number of Pods
- If there are excess Pods, they get killed
- New pods are launched when they fail, get deleted, or terminated
- Creating a replication controller with a count of 1 ensures that a Pod is always available
- Replication Controller and Pods are associated through Labels

Service

A Kubernetes Service is an abstraction which defines a logical set of Pods running somewhere in your cluster, that all provide the same functionality. When created, each Service is assigned a unique IP address (also called clusterIP).



Summary


Replica Set
Replicas → 2
Pod
version = 1.0
type = Frontend
Pod
version = 1.0
type = Frontend
Service
Label selectors:
version = 1.0
type = Frontend

Summary

- Kubernetes Master runs the API, Scheduler and Controller services
- Each Node is responsible for running one or more Pods
- Pods are the unit of deployment in Kubernetes
- Labels associate one Kubernetes object with the other
- Replication Controller ensures high availability of Pods
- Services expose Pods to internal and external consumers

Networking

Kubernetes assumes that pods can communicate with other pods, regardless of which host they land on.

Every pod gets its own IP address so you do not need to explicitly create links between pods and you almost never need to deal with mapping container ports to host ports.

There are a number of ways that this network model can be implemented.

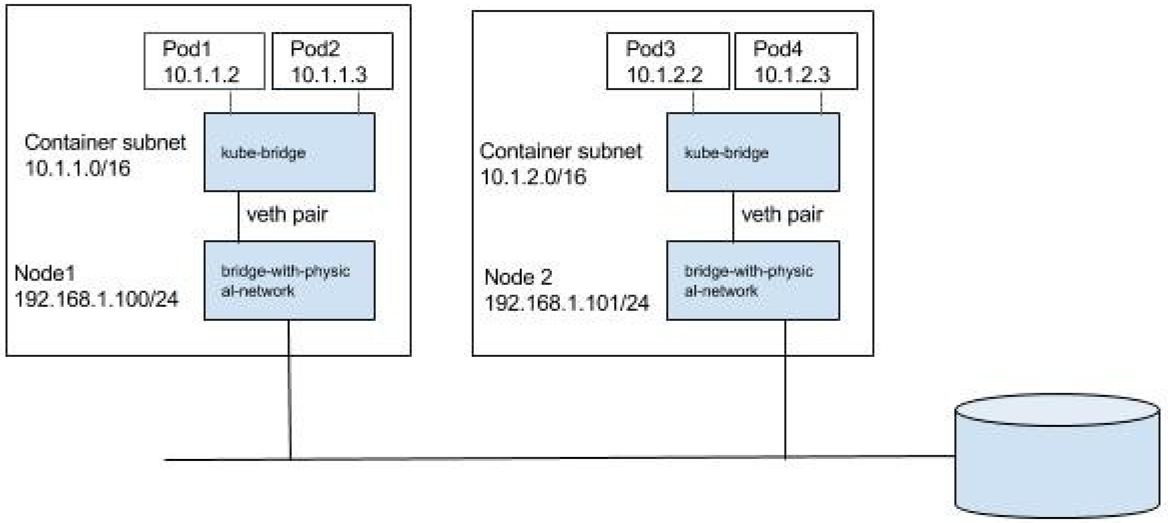
Networking

Kubernetes does not orchestrate setting up the network and offloads the job to the CNI plug-ins.
Possible network implementation options through CNI plugins:

- layer 2 (switching) solution
- layer 3 (routing) solution
- overlay solutions

Networking – layer 2

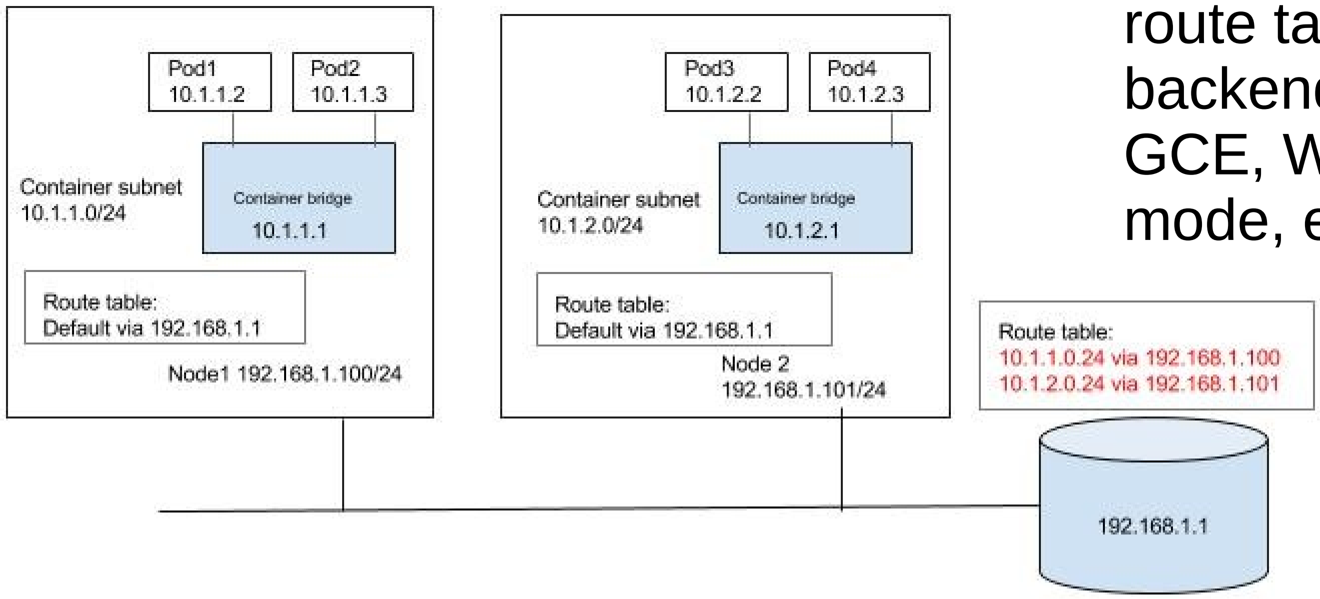
This is the simplest approach should work well for small deployments. Pods and nodes should see subnet used for pod ip's as a single l2 domain. Pod-to-pod communication (on same or across hosts) happens through ARP and L2 switching.



- Using NAT
- With Linux Bridge devices
- With Open vSwitch Bridge devices

Networking – layer 3

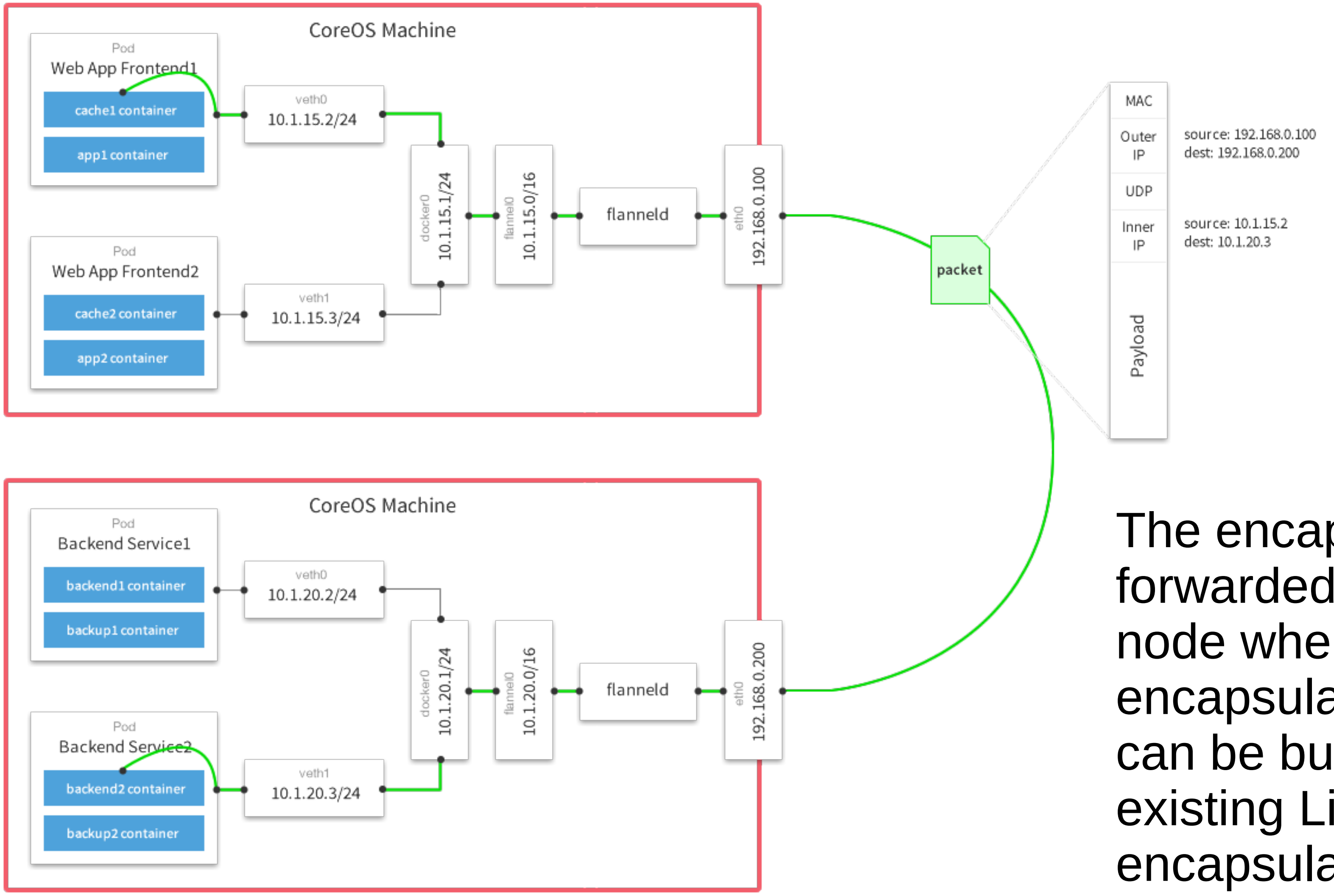
A more scalable approach is to use node routing than switching the traffic to the pods.



Pod1 10.1.1.2
Pod2 10.1.1.3
Container subnet 10.1.1.0/24
Container bridge 10.1.1.1
Route table: Default via 192.168.1.1
Node1 192.168.1.100/24
Pod3 10.1.2.2
Pod4 10.1.2.3
Container subnet 10.1.2.0/24
Container bridge 10.1.2.1
Route table: Default via 192.168.1.1
Node 2 192.168.1.101/24
Route table:
10.1.1.0.24 via 192.168.1.100
10.1.2.0.24 via 192.168.1.101
192.168.1.1

IP routing across nodes - like the cloud provider route tables, e.g. Flannel's backend for AWS and GCE, Weave's AWS-VPC mode, etc.

Networking – overlay network



The encapsulated packet is forwarded to destination node where it is de-encapsulated. A solution can be built around any existing Linux encapsulation mechanisms.

Networking

Cluster IP address - can mean 2 things: a type of service which is only accessible within a Kubernetes cluster, or the internal ("virtual") IP of components within a Kubernetes cluster.

Node IP address – physical IP address of Node

External IP address – external address for load balancer or ingress.

Only supported in cloud provider: GCE, Aazure, AWS etc.

Rolling Updates

- Rolling updates incrementally replace your resource's Pods with new ones.
- Rolling updates are designed to update your workloads without downtime.

Rolling Updates

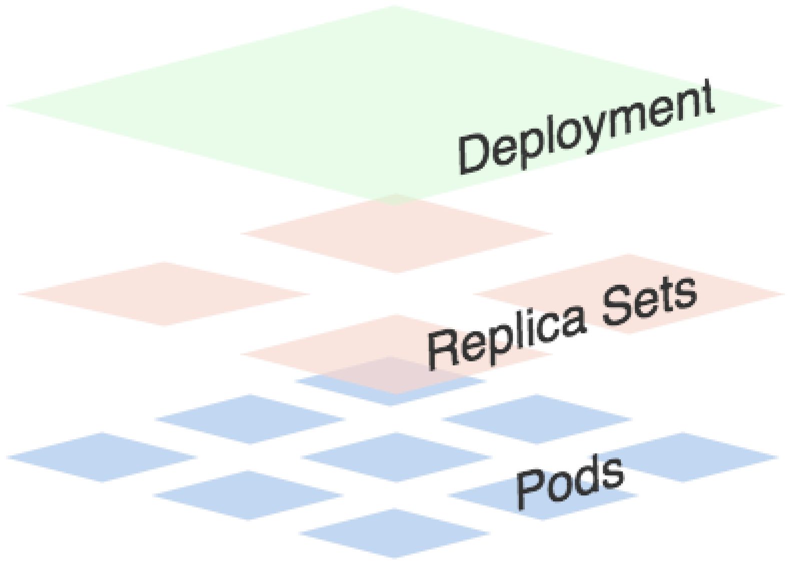
You can trigger a rolling update on these workloads by updating their Pod Template:

- DaemonSets
- Deployments
- StatefulSets

Deployments

The newer version of Kubernetes, official suggests using Deployment instead of Replication Controller(rc) to perform a rolling update.

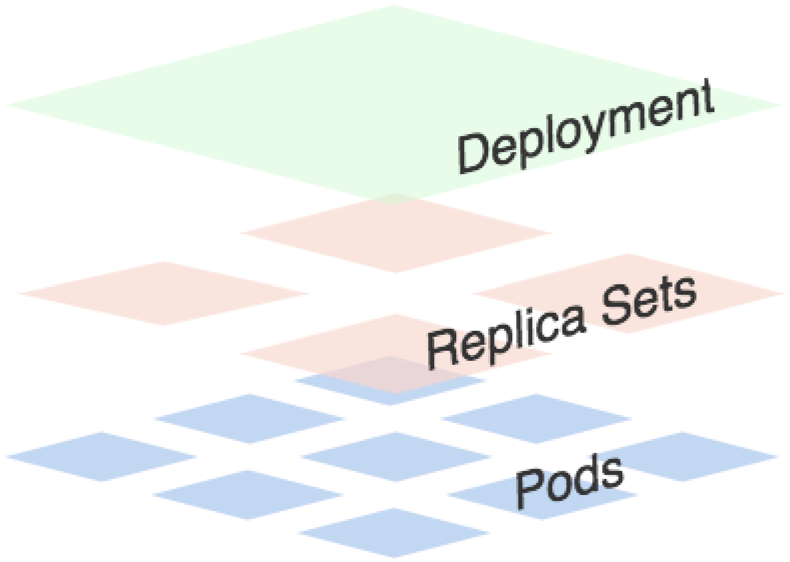
A Deployment owns and manages one or more Replica Sets. And Replica Set manages the basic units in Kubernetes - Pods.



Deployments

The newer version of Kubernetes, official suggests using Deployment instead of Replication Controller(rc) to perform a rolling update.

A Deployment owns and manages one or more Replica Sets. And Replica Set manages the basic units in Kubernetes - Pods.



Kubernetes creates a new Replica Set each time after the new Deployment config is deployed and also keeps the old Replica Set.

Deployments

Strategy

.spec.strategy specifies the strategy used to replace old Pods by new ones.

.spec.strategy.type can be “Recreate” or “RollingUpdate”. “RollingUpdate” is the default value.

Deployments

Strategy

```
minReadySeconds: 5
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 1
```

Deployments

Strategy

```
minReadySeconds: 5
```

the bootup time of your application, Kubernetes waits specific time til the next pod creation.

Kubernetes assume that your application is available once the pod created by default.

If you leave this field empty, the service may be unavailable after the update process cause all the application pods are not ready yet

Deployments

Strategy

```
minReadySeconds: 5
strategy:
  type: RollingUpdate
```

Recreate Deployment:

All existing Pods are killed before new ones are created when **.spec.strategy.type==Recreate**.

Rolling Update Deployment

The Deployment updates Pods in a rolling update fashion when **.spec.strategy.type==RollingUpdate**.

You can specify **maxUnavailable** and **maxSurge** to control the rolling update process.

Deployments

Strategy

```
minReadySeconds: 5
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 1
```

MaxSurge: amount of pods more than the desired number of Pods. This fields can be an absolute number or the percentage, ex. maxSurge: 1 means that there will be at most 4 pods during the update process if replicas is 3

MaxUnavailable: amount of pods that can be unavailable during the update process. This fields can be a absolute number or the percentage. This fields cannot be 0 if maxSurge is set to 0, ex. maxUnavailable: 1 means that there will be at most 1 pod unavailable during the update process

Deployments

Use Cases:

- Creating a Deployment
- Updating a Deployment
- Rolling Back a Deployment
- Scaling a Deployment
- Pausing and Resuming a Deployment
- Deployment status
- Clean up Policy

StatefulSet

Known as PetSet on pre 1.6 version, StatefulSet is the best way to deploy stateful applications in production, because:

- Deleting and/or scaling down a StatefulSet will not delete the volumes associated with the StatefulSet. This is done to ensure data safety, which is generally more valuable than an automatic purge of all related StatefulSet resources.
- For a StatefulSet with N replicas, when Pods are being deployed, they are 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 shut down.

StatefulSet

CAP stands for Consistency, Availability and Pratition Tolerance.

Consistency (C): All nodes see the same data at the same time. What you write you get to read.

Availability (A): A guarantee that every request receives a response about whether it was successful or failed. Whether you want to read or write you will get some response back.

Partition tolerance (P): The system continues to operate despite arbitrary message loss or failure of part of the system. Irrespective of communication cut down among the nodes, system still works. An example of a network partition is when two nodes can't talk to each other, but there are clients able to talk to either one or both of those nodes.

P is driven by wires, electricity, software and hardware and none of us have any control and often P may not be met.

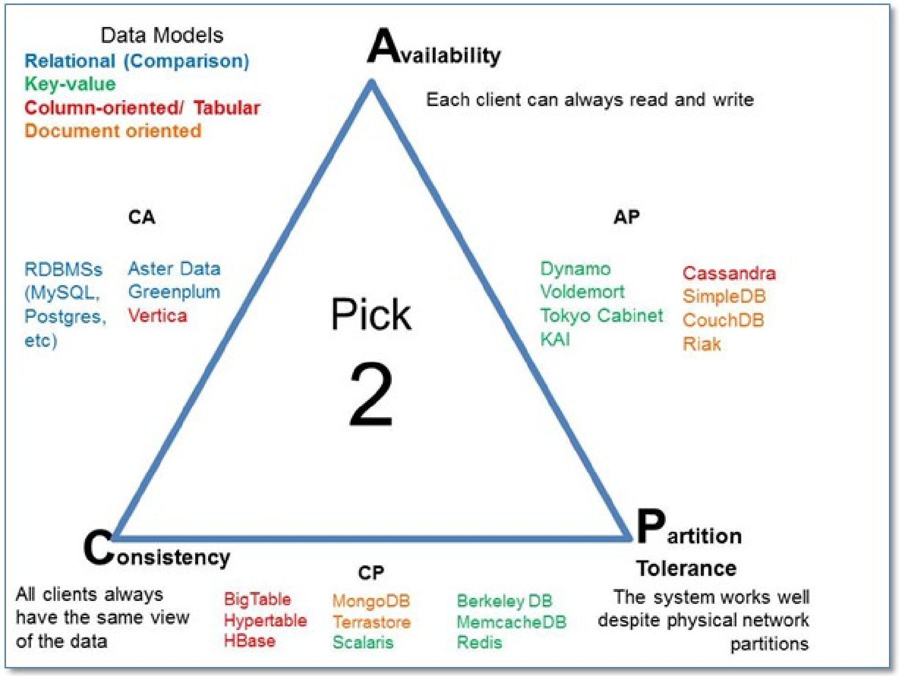
In the absence of network failure – that is, when the distributed system is running normally – both availability and consistency can be satisfied.

StatefulSet

The problem comes when P is not met. Now we have two choices to make.

AP: When there is no partition tolerance, system is available but with inconsistent data.

CP: When there is no partition tolerance, system is not fully available. But the data is consistent. You can create a rule that a result will be returned only when majority of nodes agree.


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
Pick 2
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions