# Kubernetes

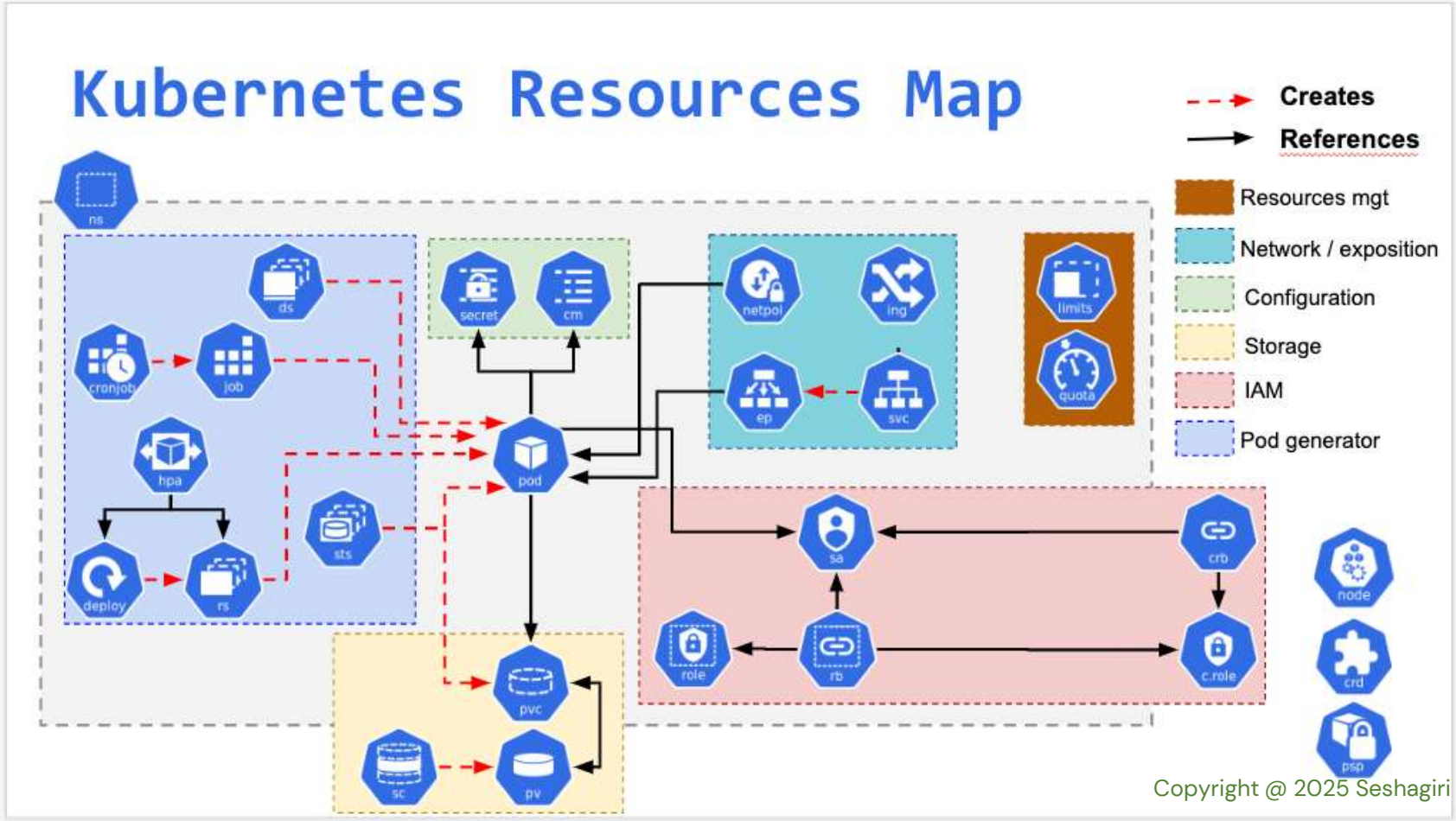
Deploying Apps with Kubernetes

# Kubernetes Introduction

- **What is Kubernetes (“k8s” or “kube”)**
- **Kubernetes Architecture**
- **Core Concepts of Kubernetes**
- Kubernetes resources explained
  - Networking
  - Storage and Persistence
  - Logging
- Application Optimization on Kubernetes
- Kubernetes effect on the software development life cycle
- Local and Distributed Abstractions and Primitives
- Container Design Patterns and best practices
- Deployment and release strategy with Kubernetes

# Kubernetes Introduction

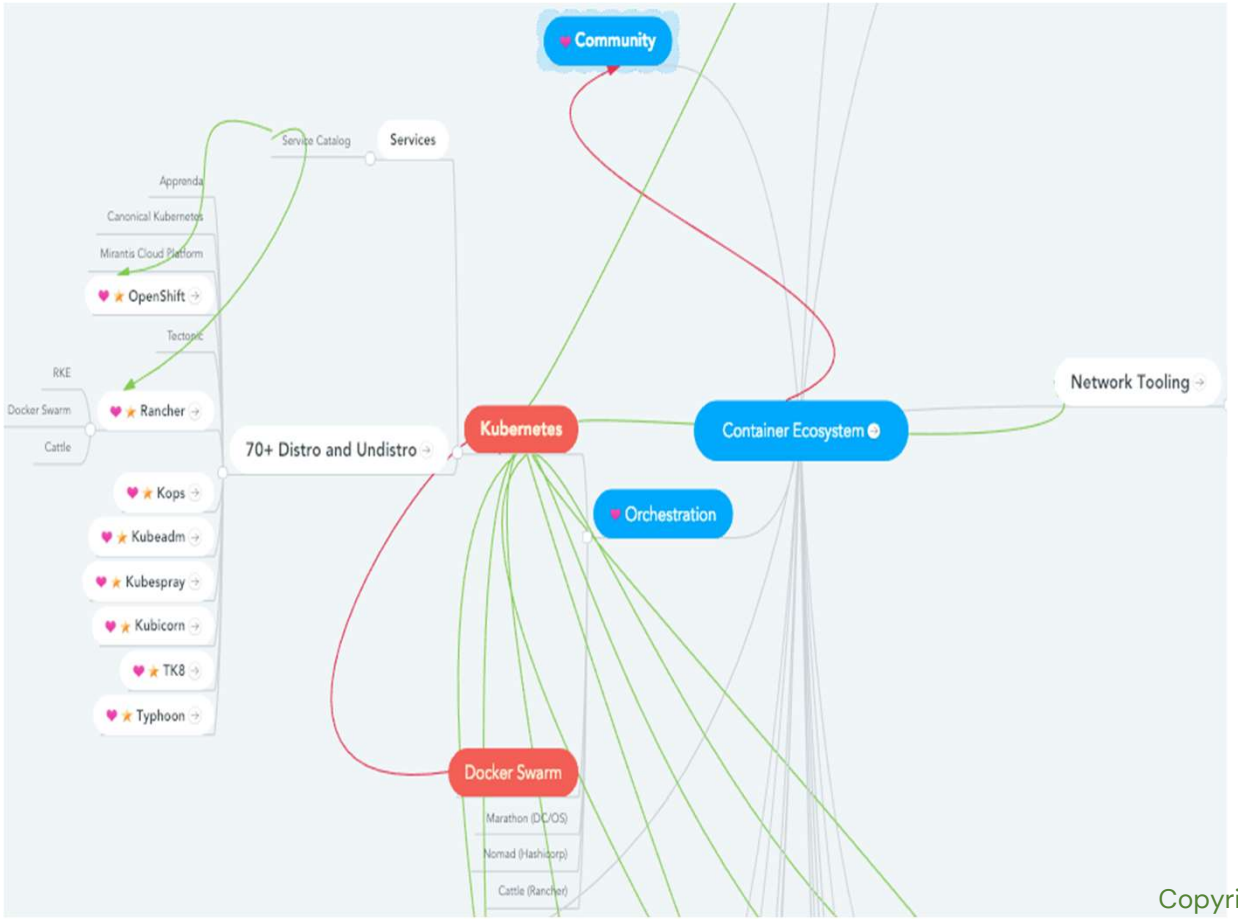
# Kubernetes – Resource Map



# Kubernetes Introduction

- What is Kubernetes
- Overview of Kubernetes Architecture
- Key Terms and Concepts
- Installation and Setup
- Exercises

# Kubernetes Introduction



# Kubernetes – What  is it

- Kubernetes is Greek for "helmsman", your guide through unknown waters)
- Kubernetes is the linux kernel of distributed systems
- Kubernetes is the linux of the cloud!
- Kubernetes is a platform and container orchestration tool for automating deployment, scaling, and operations of application containers.
- Kubernetes supports, Containerd, CRI-O, Kata containers (formerly clear and hyper) and Virtlet

# Kubernetes – Container Runtimes

- Kubernetes unlike docker does support multiple Container Runtimes
  - Docker
  - CRI-O
  - ContainerD
- The choice of the container runtime depends on a numbr of factors
- Additional Resources
  - Linux Container Internals by Scott McCarty
  - Container Runtimes and Kubernetes by Fahed Dorgaa
  - Kubernetes Runtime Comparison

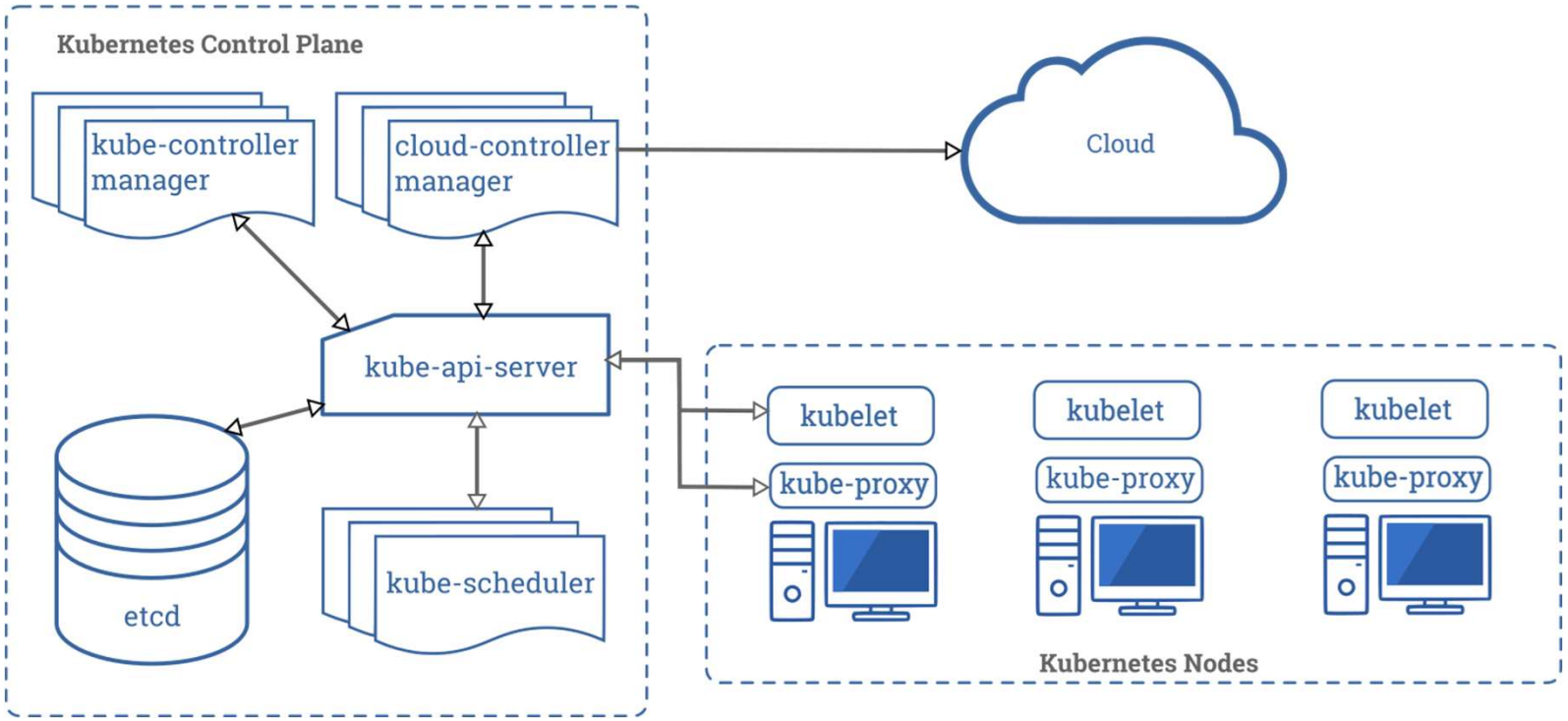
# Kubernetes – Container Runtimes

- Kubernetes unlike docker does support multiple Container Runtimes
  - Docker
  - CRI-O
  - ContainerD
- The choice of the container runtime depends on a numbr of factors
- Additional Resources
  - Linux Container Internals by Scott McCarty
  - Container Runtimes and Kubernetes by Fahed Dorgaa
  - Kubernetes Runtime Comparison

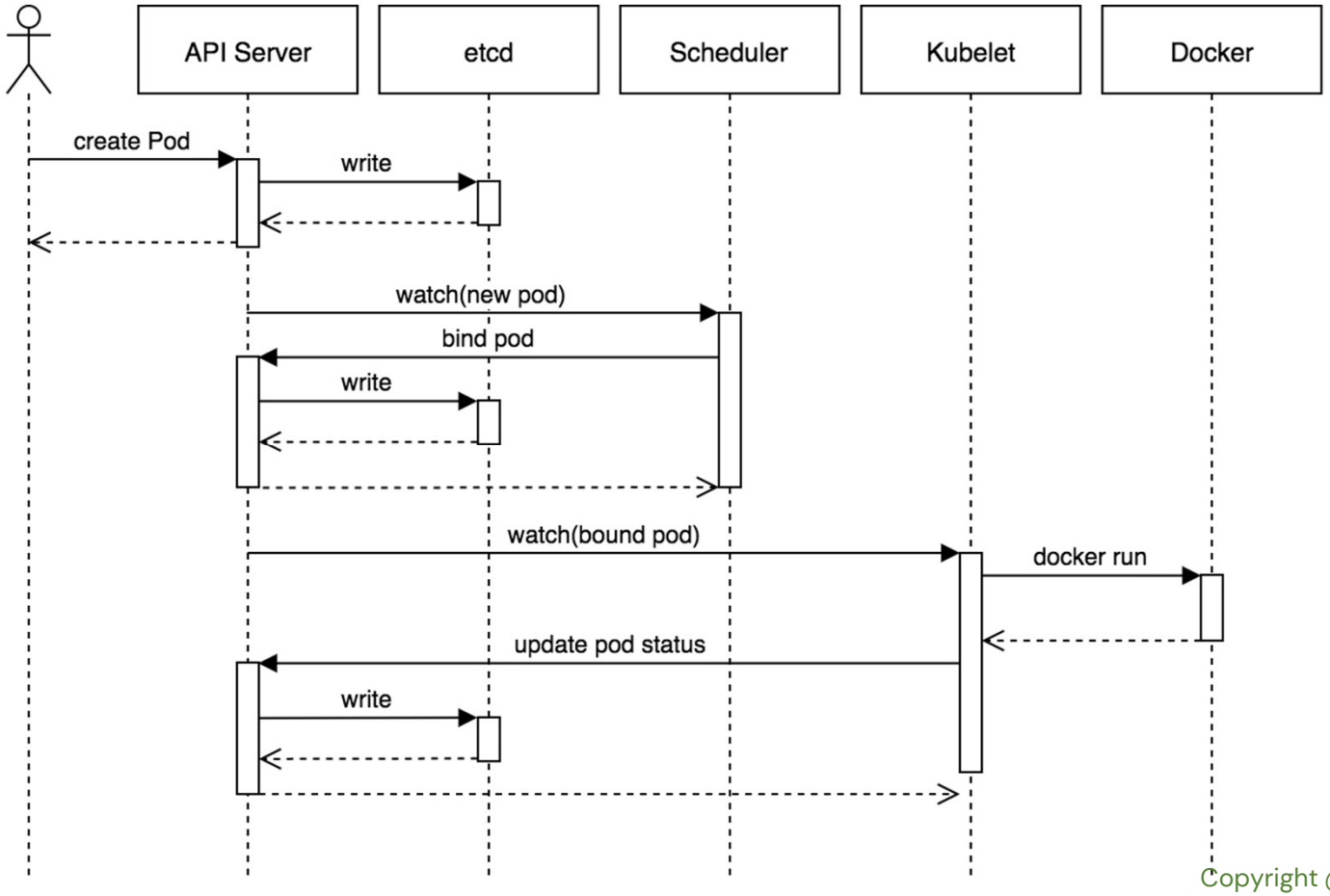
# Kubernetes – How does it work

- A main Master Node
- Multiple Worker Nodes
  - NB: In MiniKube,  only a single node
- Each Worker node can handle multiple pods
- Pods are just containers clustered as a single working unit
- We build applications and package them as pods using pod definitions and deployments  to the main master node
- The master node takes these definitions and deploys to the worker nodes.

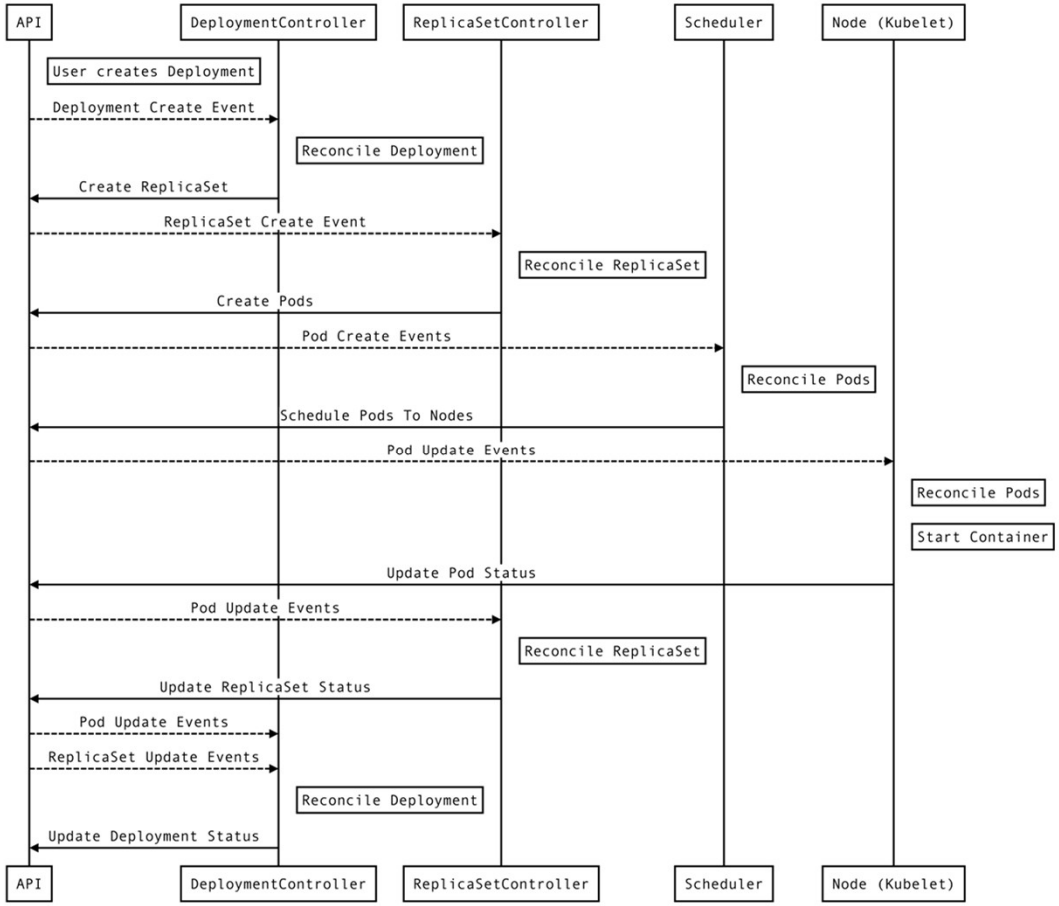
# Kubernetes – Components



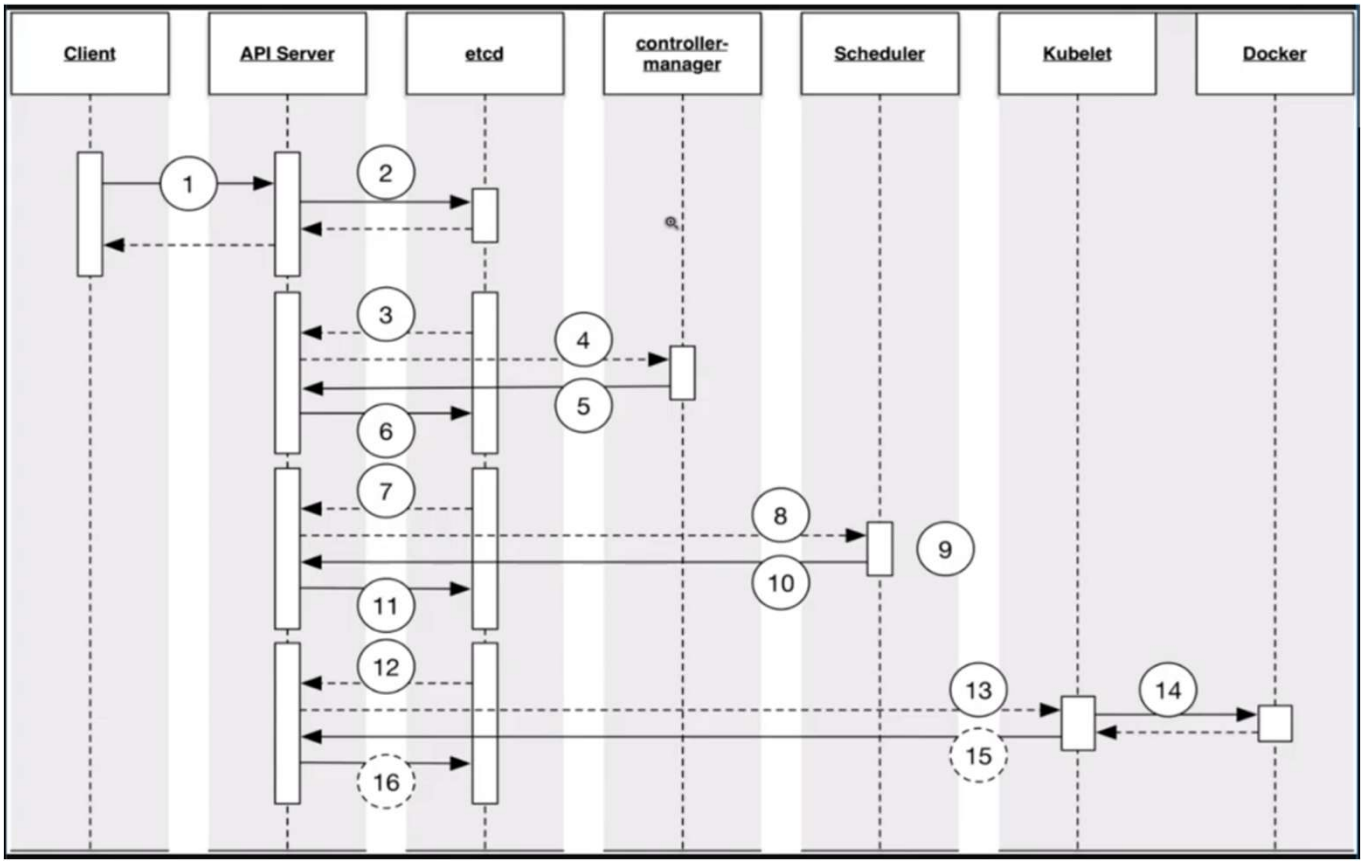
# Kubernetes – The flow of work



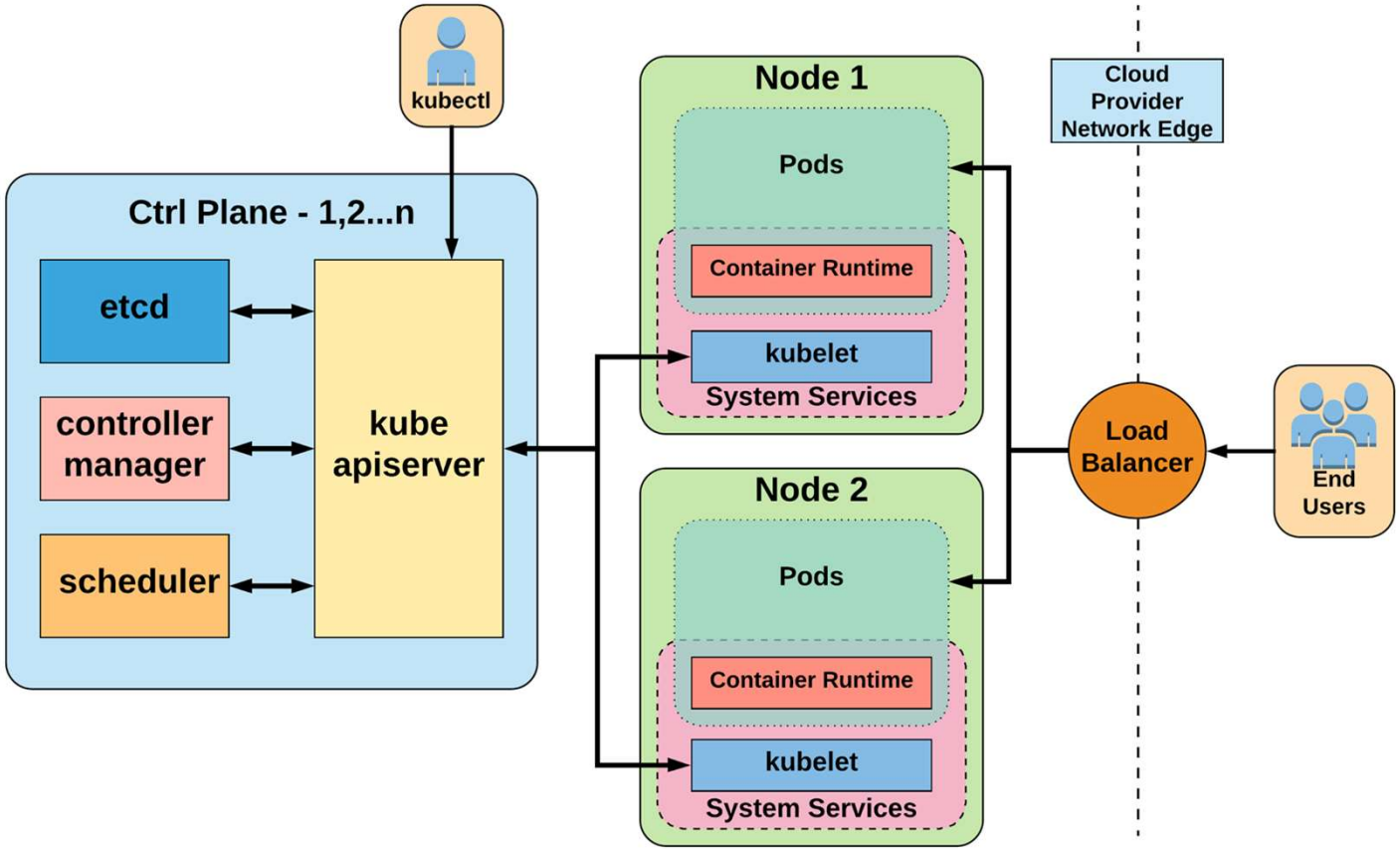
# Kubernetes – The flow of work



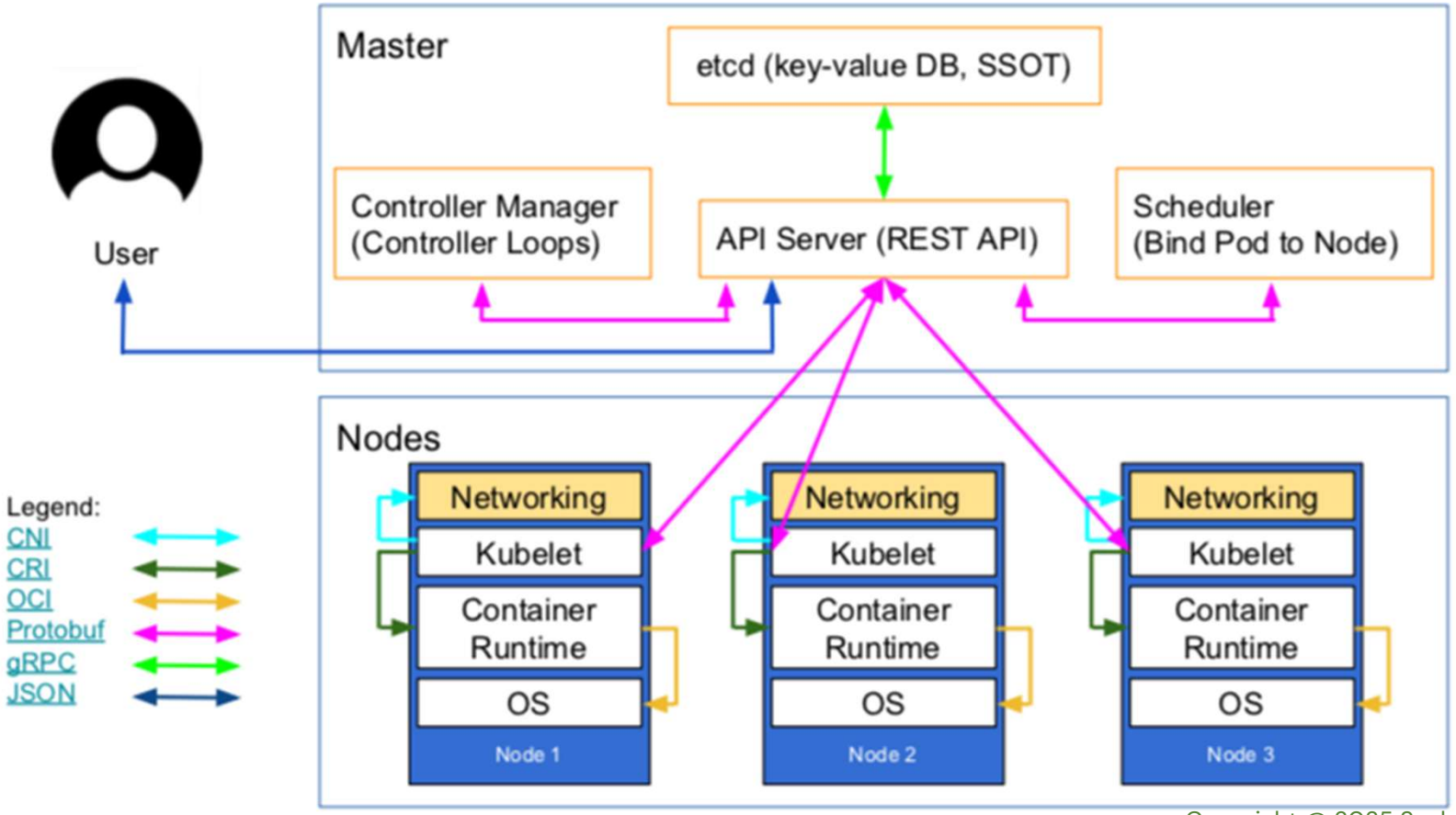
# Kubernetes – The flow of work



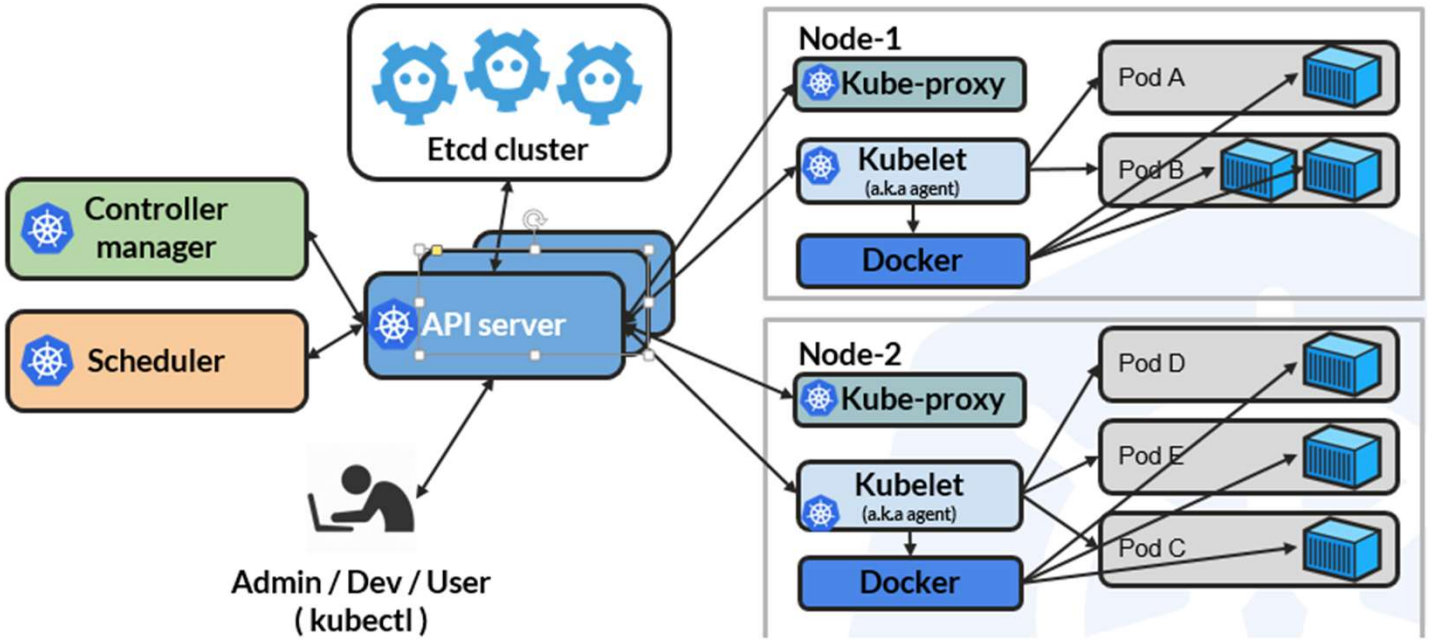
# Kubernetes Architecture Overview



# Kubernetes Architecture Overview



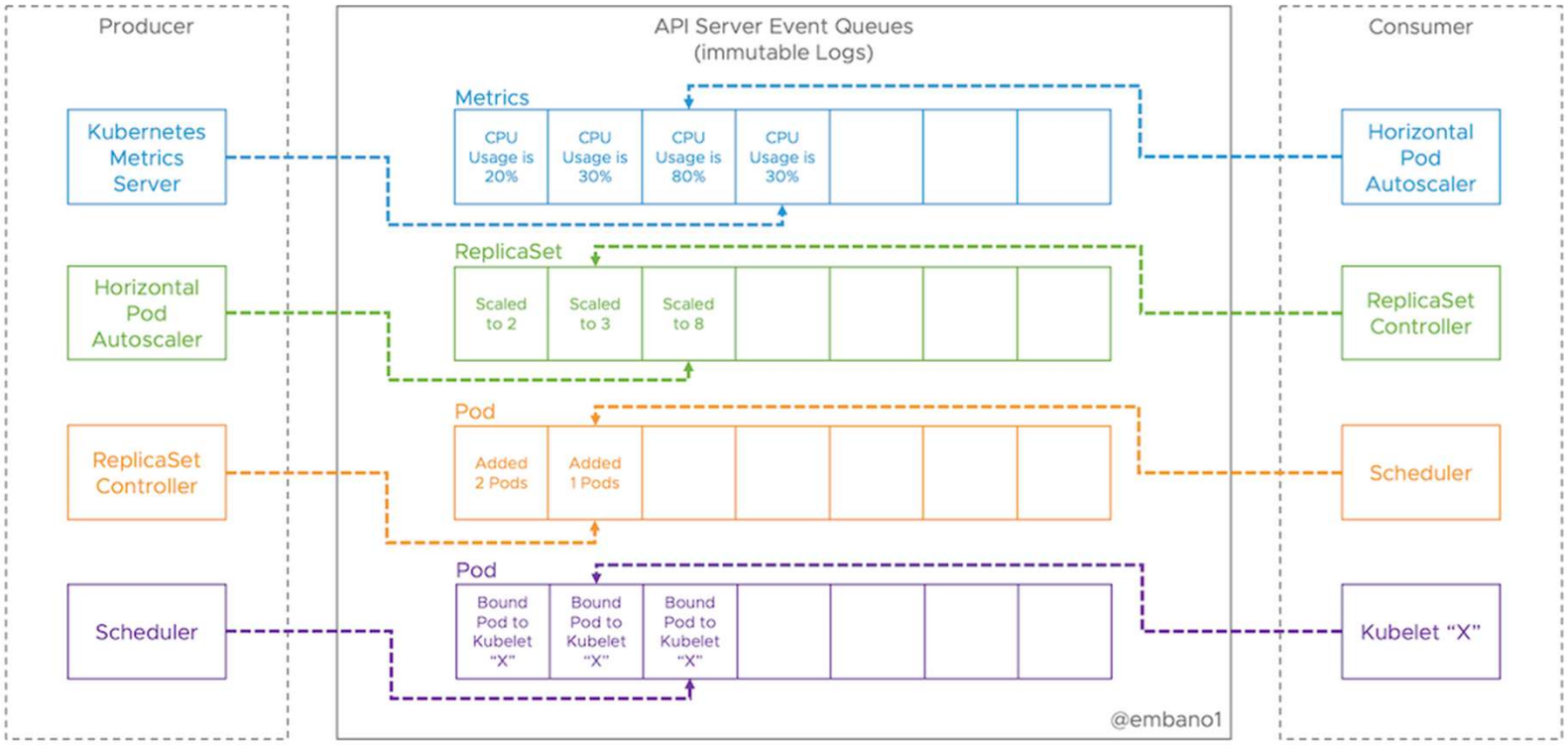
# Kubernetes Architecture Overview



# Kubernetes Architecture Overview

- Kubernetes is like Kafka
- Event Driven  Architecture
- Producers write to a queue – this  is considering as generating an event
- Consumers subscribe to these events and/or notified and respond accordingly

# Kubernetes Architecture Overview



# Kubernetes Core Concepts

- Pod
- Label and selectors
- Controllers
  - Deployments
  - ReplicaSet
  - ReplicationController
  - DaemonSet
- Service
- StatefulSets
- ConfigMaps
- Secrets
- Persistent Volumes (attaching storage to containers)
- Life Cycle of Applications in Kubernetes
  - Updating Pods
  - Rolling updates
  - Rollback

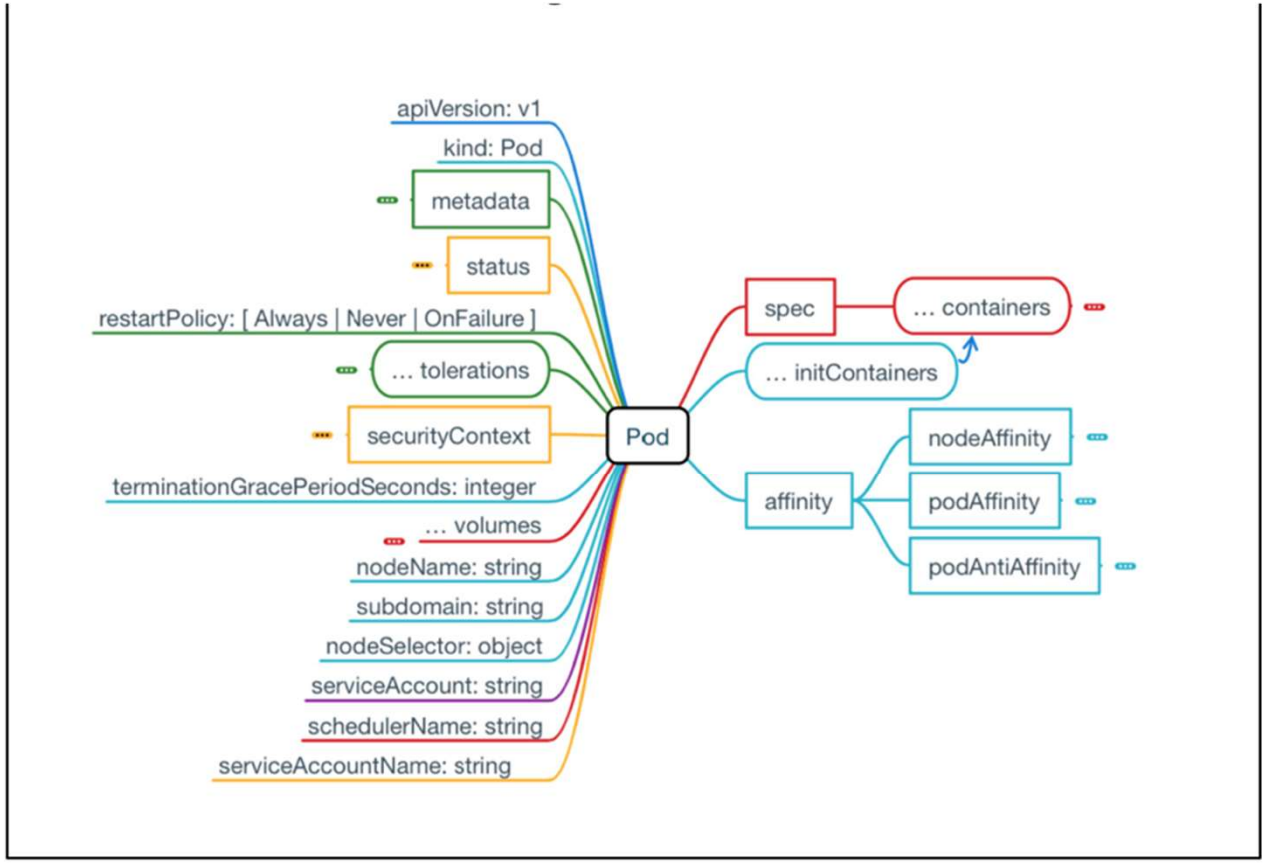
# Kubernetes Core Concepts

| | Resource (abbr.) [API version] | Description |
|---|---|---|
| | Namespace* (ns) [v1] | Enables organizing resources into non-overlapping groups (for example, per tenant) |
| Deploying Workloads | Pod (po) [v1] | The basic deployable unit containing one or more processes in co-located containers |
| | ReplicaSet | Keeps one or more pod replicas running |
| | ReplicationController | The older, less-powerful equivalent of a ReplicaSet |

# Kubernetes Core Concepts

| | Resource (abbr.) [API version] | Description |
|---|---|---|
| Deploying Workloads | Job | Runs pods that perform a completable task |
| | CronJob | Runs a scheduled job once or periodically |
| | DaemonSet | Runs one pod replica per node (on all nodes or only on those matching a node selector) |
| | StatefulSet | Runs stateful pods with a stable identity |
| | Deployment | Declarative deployment and updates of pods |

# Kubernetes Core Concepts



# Kubernetes Core Concepts

| | Resource (abbr.) [API version] | Description |
|---|---|---|
| Services | **Service** (svc) [v1]<br><br>**Endpoints** (ep) [v1]<br><br>Ingress (ing) [extensions/v1beta1] | Exposes one or more pods at a single and stable IP address and port pair<br><br>Defines which pods (or other servers) are exposed through a service<br><br>Exposes one or more services to external clients through a single externally reachable IP address |

# Kubernetes Core Concepts

| | Resource (abbr.) [API version] | Description |
|---|---|---|
| Config | ConfigMap (cm) [v1] | A key-value map for storing non-sensitive config options for apps and exposing it to them |
| | Secret [v1] | Like a ConfigMap, but for sensitive data |
| Storage | PersistentVolume* (pv) [v1] | Points to persistent storage that can be mounted into a pod through a PersistentVolumeClaim |
| | PersistentVolumeClaim (pvc) [v1] | A request for and claim to a PersistentVolume |
| | StorageClass* (sc) [storage.k8s.io/v1] | Defines the type of storage in a PersistentVolumeClaim |

# Kubernetes Core Concepts

| | Resource (abbr.) [API version] | Description |
|---|---|---|
| Scaling | HorizontalPodAutoscaler (hpa) [autoscaling/v2beta1**] | Automatically scales number of pod replicas based on CPU usage or another metric |
| | PodDisruptionBudget (pdb) [policy/v1beta1] | Defines the minimum number of pods that must remain running when evacuating nodes |
| Resources | LimitRange (limits) [v1] | Defines the min, max, default limits, and default requests for pods in a namespace |
| | ResourceQuota (quota) [v1] | Defines the amount of computational resources available to pods in the namespace |

# Kubernetes Core Concepts

| | Resource (abbr.) [API version] | Description |
|---|---|---|
| Cluster state | Node* (no) [v1] | Represents a Kubernetes worker node |
| | Cluster* [federation/v1beta1] | A Kubernetes cluster (used in cluster federation) |
| | ComponentStatus* (cs) [v1] | Status of a Control Plane component |
| | Event (ev) [v1] | A report of something that occurred in the cluster |

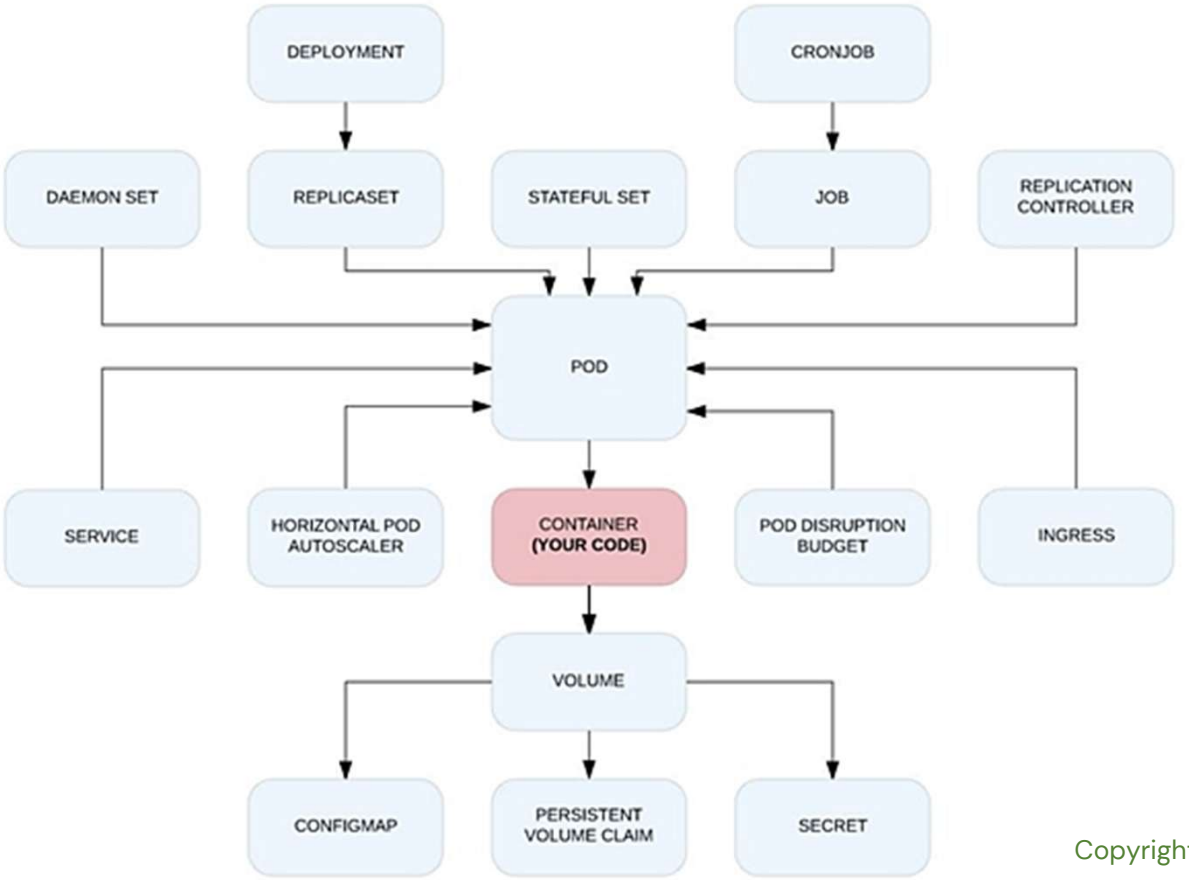
# Kubernetes Core Concepts

| | Resource (abbr.) [API version] | Description |
|---|---|---|
| Security | ServiceAccount (sa) [v1] | An account used by apps running in pods |
| | Role [rbac.authorization.k8s.io/v1] | Defines which actions a subject may perform on which resources (per namespace) |
| | ClusterRole* [rbac.authorization.k8s.io/v1] | Like Role, but for cluster-level resources or to grant access to resources across all namespaces |
| | RoleBinding [rbac.authorization.k8s.io/v1] | Defines who can perform the actions defined in a Role or ClusterRole (within a namespace) |
| | ClusterRoleBinding* [rbac.authorization.k8s.io/v1] | Like RoleBinding, but across all namespaces |

# Kubernetes Core Concepts

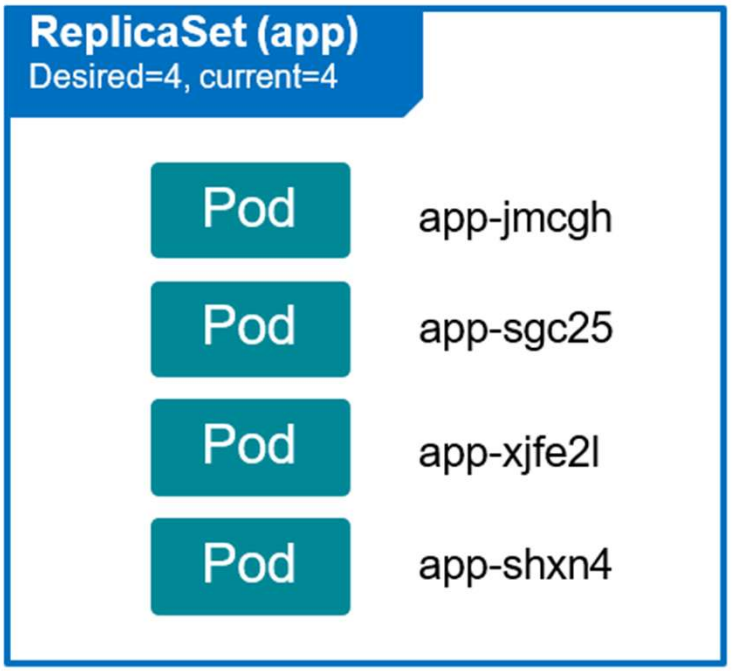
| | Resource (abbr.) [API version] | Description |
|---|---|---|
| Security | PodSecurityPolicy* (psp) [extensions/v1beta1] | A cluster-level resource that defines which security-sensitive features pods can use |
| | NetworkPolicy (netpol) [networking.k8s.io/v1] | Isolates the network between pods by specifying which pods can connect to each other |
| | CertificateSigningRequest* (csr) [certificates.k8s.io/v1beta1] | A request for signing a public key certificate |
| Ext. | CustomResourceDefinition* (crd) [apiextensions.k8s.io/v1beta1] | Defines a custom resource, allowing users to create instances of the custom resource |

# Kubernetes Dependencies



# Deployment and Release Strategy

- POD Replication – Having many pods with same definition except Name
- Replication Controller – Maintain a stable set of replica pods at any given time

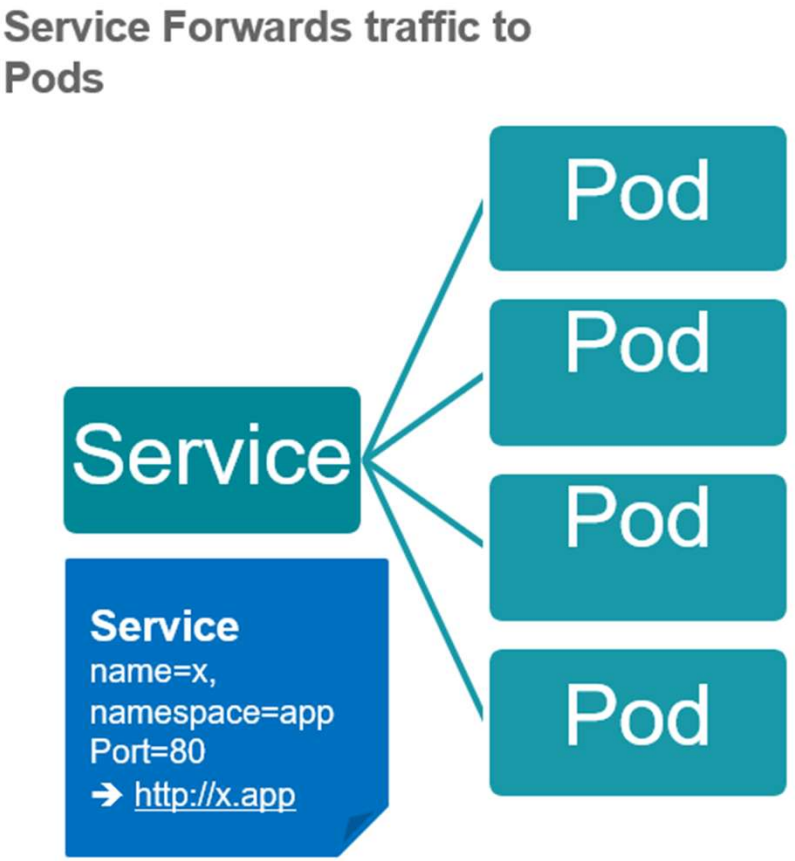


# Deployment and Release

- Scaling can be done via
  - Edit YAML file and apply
  - *kubectl edit replicaset*
  - *kubectl scale replicaset … --replicas X*

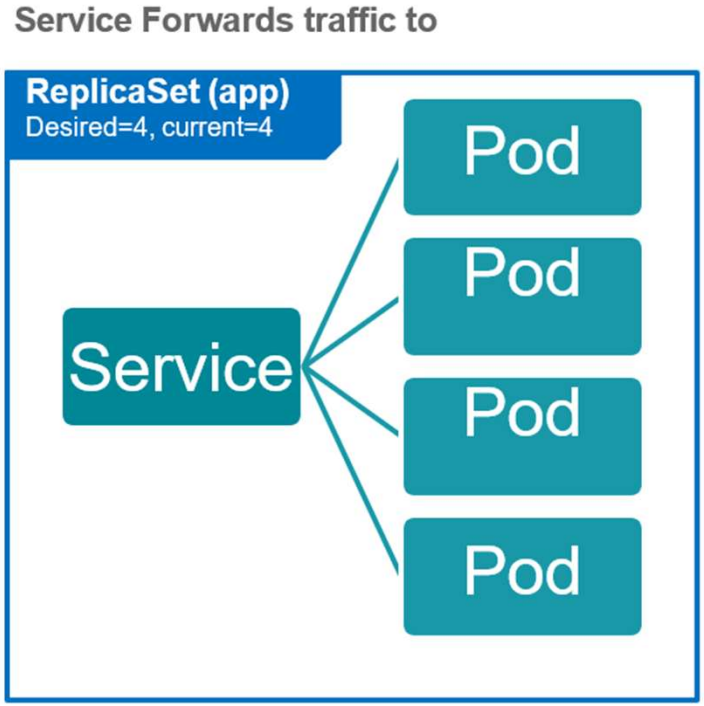
# Some Background

- All Communication to pods via service
- PODS are ephemeral in nature
- Traffic is routed using labels and selectors



# Some Background

- ReplicaSet replicates Pods with same Labels
- Because all replicas has same labels, A service with Selectors=Labels can forward traffic in round-robin way.
- Service behaves like loadbalancer for replicas ( pods With the same labels )



# Deployment and Release

- **Scenario:**
- You **deployed** an app few days ago.
- Now you want to **upgrade** your app from v1 to v2.

- **Challenges:**
  - Can you upgrade with Zero downtime?
  - Can you upgrade **replicas** sequentially one after another?
  - Can you pause and resume upgrade process?
  - Rollback upgrade to previous stable release

# Deployment and Release

- **In General:**
- **Deployment** is all of the activities that make a software system available for use
- Deployment First Release **vs** Deployment Subsequent Releases
  - First Installation **vs** Subsequent Upgrades

- **In Kubernetes:**
- Deployment is automated using Controllers
  - Deployment controller
  - ReplicaSetController

# Deployment and Release

## Controllers for Pods

Deployments

Jobs

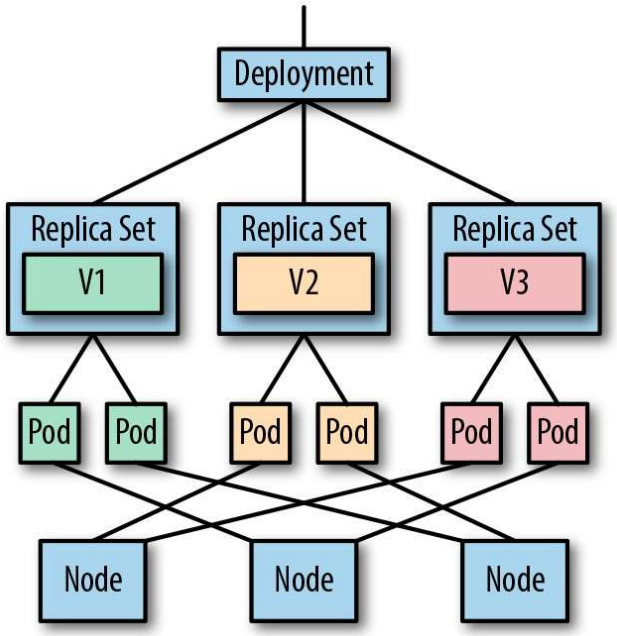
ReplicaSet

DaemonSet

CronJob

Controllers

## Controller Graphs



# Deployment and Release Strategy

- ReplicaSet : Pod Template + Replication
- Deployment: ReplicaSet Template + Update Strategy
- Deployment : Pod Template + Replication + Deployment Strategy
- Name Of Pods Created by Deployment follow this format :
  - ***<deployment-name>-<replicaset-name>-<random-string>***

# Deployment

- **Replication**

**-Deployment Strategy**

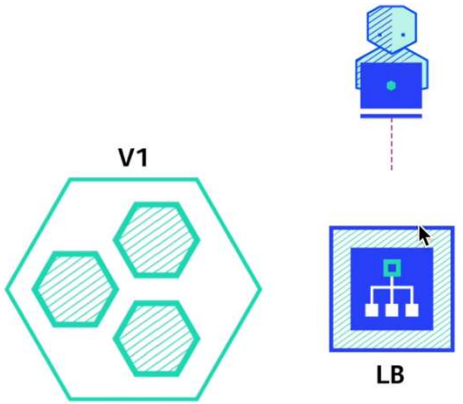
**-Pod Template**

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample
  labels:
    section: intro
spec:
  # REPLICATION INFO ============================
  replicas: 3
  # Deployment Strategy ============================
  # strategy:
  #   type: RollingUpdate
  #   rollingUpdate:
  #      maxUnavailable: 25%
  #      maxSurge: 25%
  # Pod Template ============================
  selector:
    matchLabels: #<- Must match labels of Pod
      app: sample
  template: ### PUT HERE POD DEFINITION - except kind and
    metadata:
      name: sample
      labels:
        app: sample
    spec:
      containers:
      - name: app
        image: abdennour/sample-app:v1 # Update v1 to v2
        ports:
        - containerPort: 80
```

# Deployment and Release Strategy

- **Recreate:**
- Rolling Update (Ramped or incremental)
- Rolling update with Additional
  - batch
- Blue/Green
- Canary
- A/B testing

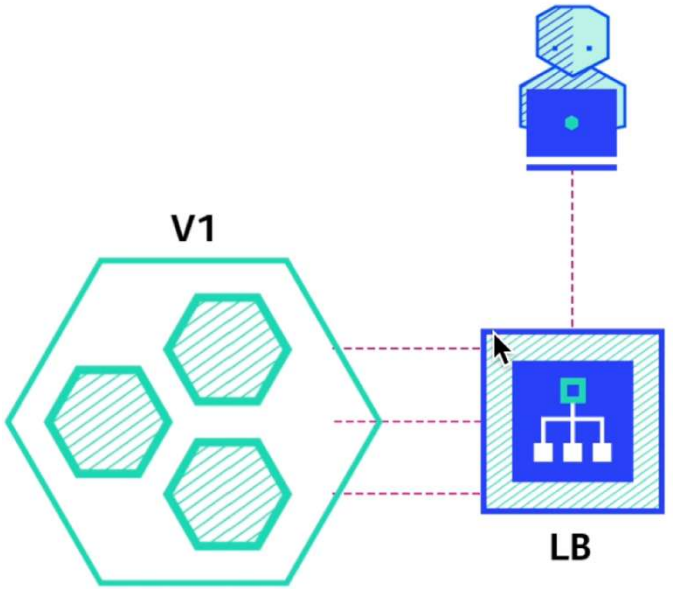
*Easy to Setup <> Expect Downtime*



# Deployment and Release Strategy

- Recreate:
- Rolling Update (Ramped or incremental)
- Rolling update with Additional batch
- Blue/Green
- Canary
- A/B testing

***Easy to Setup <> Rollout/Rollback take time***



# Deployment and Release Strategy

- **Recreate:**
- **Rolling Update (Ramped or incremental)**
- **Rolling update with Additional batch**
- **Blue/Green**
- **Canary**
- **A/B testing**

Same as ROLLING UPDATE But :

- Increase number of replicas before start the upgrade

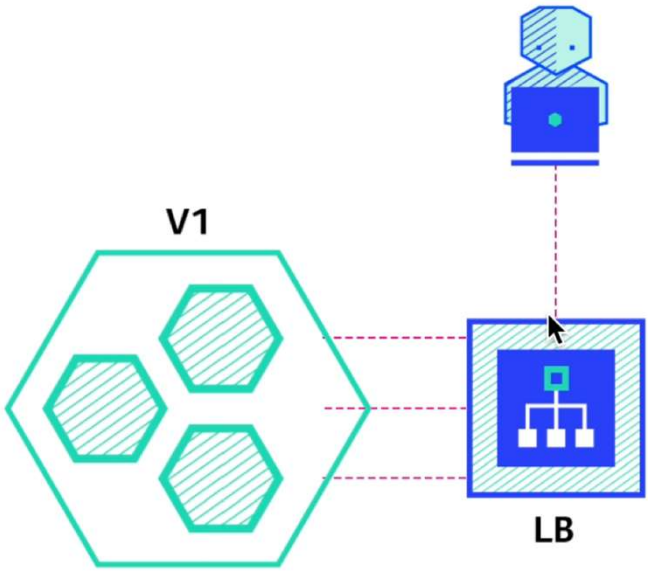
**e.g:**

- v1 with 3 replicas
- Scale out from 3 to 5 replicas
- Start rolling update v2

# Deployment and Release Strategy

- Recreate:
- **Rolling Update** (Ramped or incremental)
- Rolling update with Additional batch
- **Blue/Green**
- Canary
- A/B testing

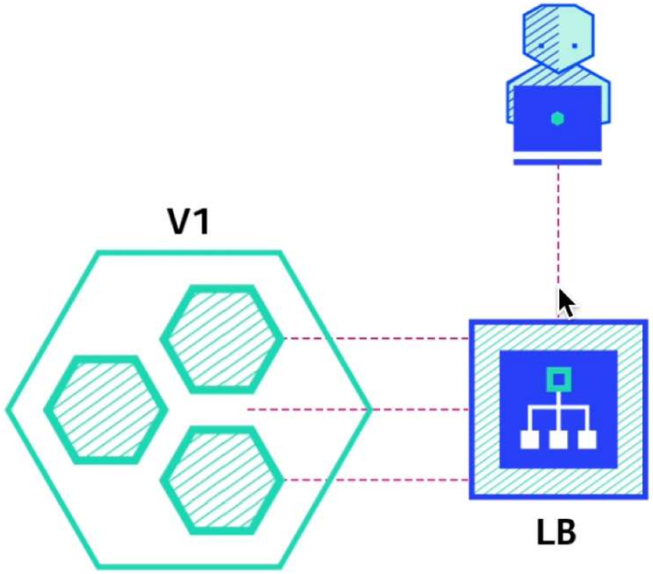
*Instant Rollback <> Expensive*



# Deployment and Release Strategy

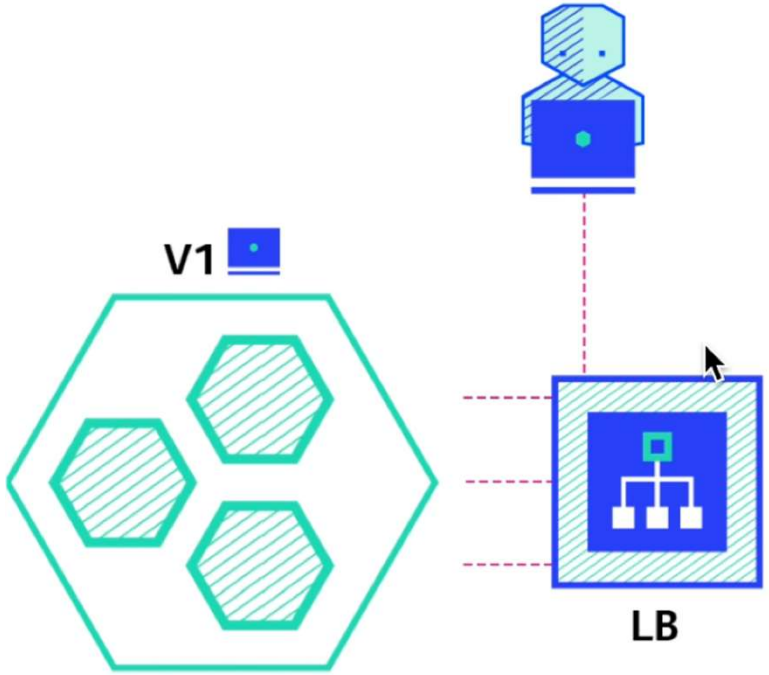
- Recreate:
- **Rolling Update** (Ramped or incremental)
- Rolling update with Additional batch
- **Blue/Green**
- **Canary**
- A/B testing

*Fast Rollback <> Slow Rollout*



# Deployment and Release Strategy

- Recreate:
- **Rolling Update** (Ramped or incremental)
- Rolling update with Additional batch
- **Blue/Green**
- **Canary**
- **A/B testing**

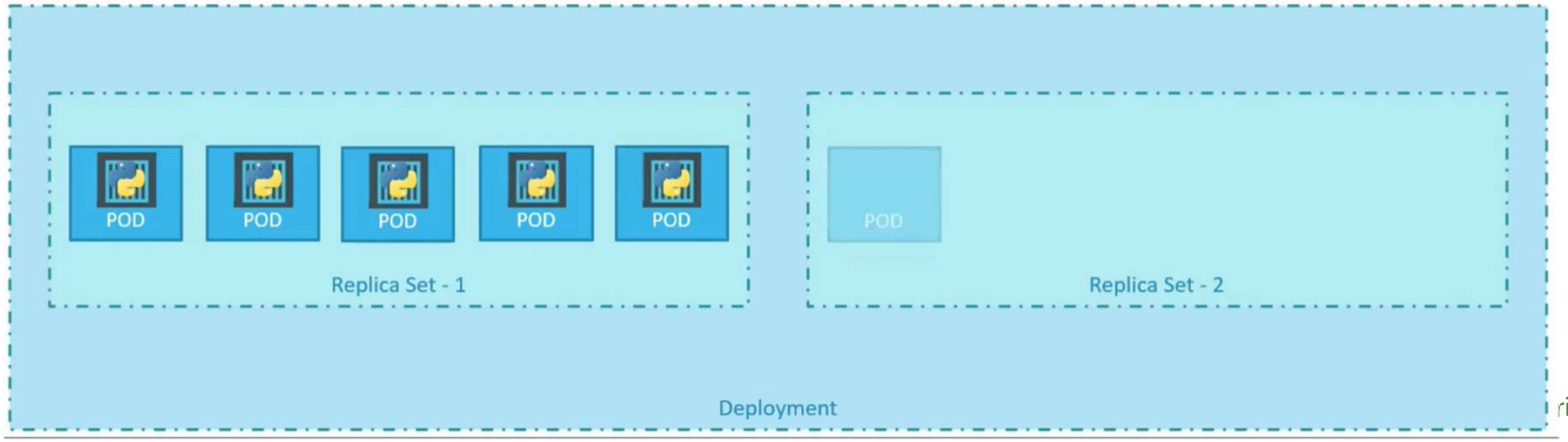


# Deploy and Release Strategy

- Recreate
  - Create a new Replica Set V2
  - Scale out Replica Set V2 to max
  - Scale Down Replica set V1 to 0
- Rolling Update or Rolling Update with Additional Batch
  - Gradually Replace existing pods in batches
  - Create new Replica Set V2
  - Scale out Replica set V2 gradually
  - Scale in Replica set v1 gradually
  - Scale in Replica Set V1 to 0

# Deploy and Release Strategy

- Rolling Update or Rolling Update with Additional Batch
  - Gradually Replace existing pods in batches
  - Create new Replica Set V2
  - Scale out Replica set V2 gradually
  - Scale in Replica set v1 gradually
  - Scale in Replica Set V1 to 0



# A Note – StatefulSets

- A StatefulSet is another Kubernetes controller that manages pods just like Deployments.
- More suited for stateful apps.
- A stateful application requires pods with a unique identity (for example, hostname). One pod should be able to reach other pods with well-defined names.
- For a StatefulSet to work, it needs a Headless Service. A Headless Service does not have an IP address.
- Internally, it creates the necessary endpoints to expose pods with DNS names. The StatefulSet definition includes a reference to the Headless Service, but you have to create it separately.

# A Note – StatefulSets

- By nature, a StatefulSet needs persistent storage so that the hosted application saves its state and data across restarts. Kubernetes provides Storage Classes, Persistent Volumes, and Persistent Volume Claims to provide an abstraction layer above the cloud provider's storage-offering mechanism.

- Once the StatefulSet and the Headless Service are created, a pod can access another one by name prefixed with the service name.

# Design Patterns – Single Container

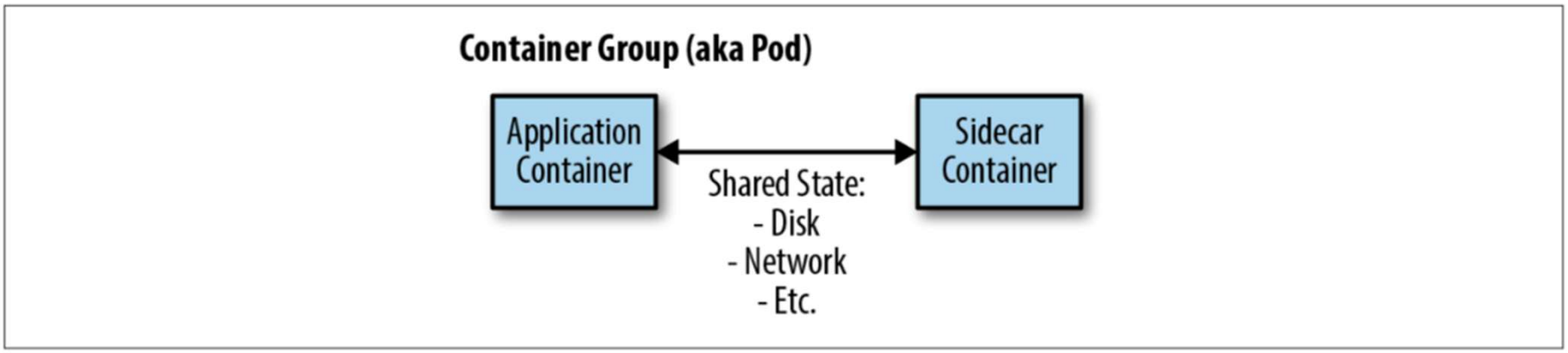
Single node base for other patterns

- Establish (reserve) resource boundaries
- Isolate resources from others (protect)
- Separation of concerns (single responsibility)

# Design Patterns – Side Car

Application Container

Extend the application (SIdeCar)

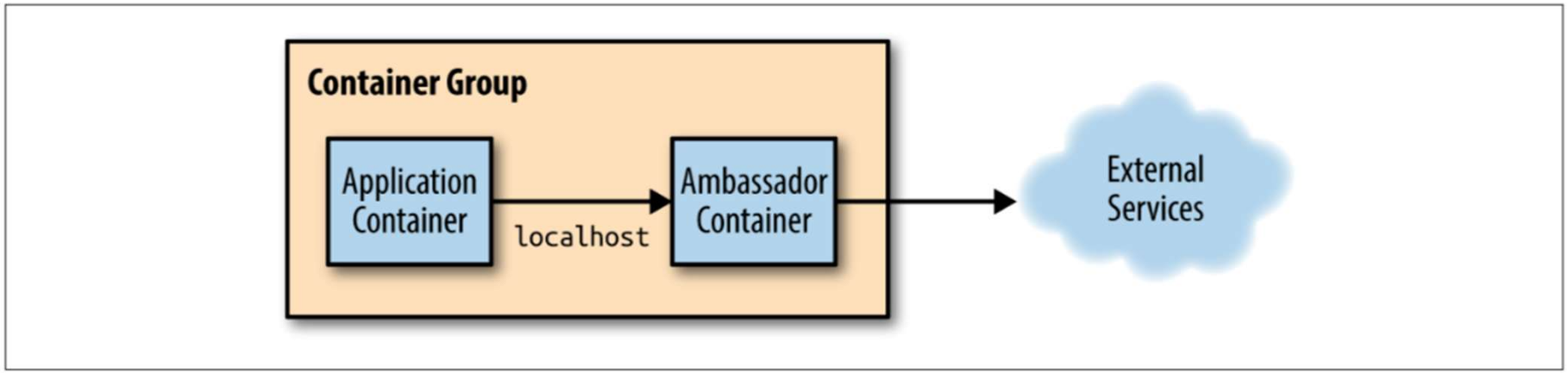


# Design Patterns – Side Car

- Co-scheduled in the same node
- Better performance when communicating
- Share resources (volumes)
- Examples:
  - HTTPS Layer
  - HTTP Proxy for Self-hosted apps

# Design Patterns – Ambassador

- Application container
- **Broker** between application and consumers
- Grouped as with sidecar pattern

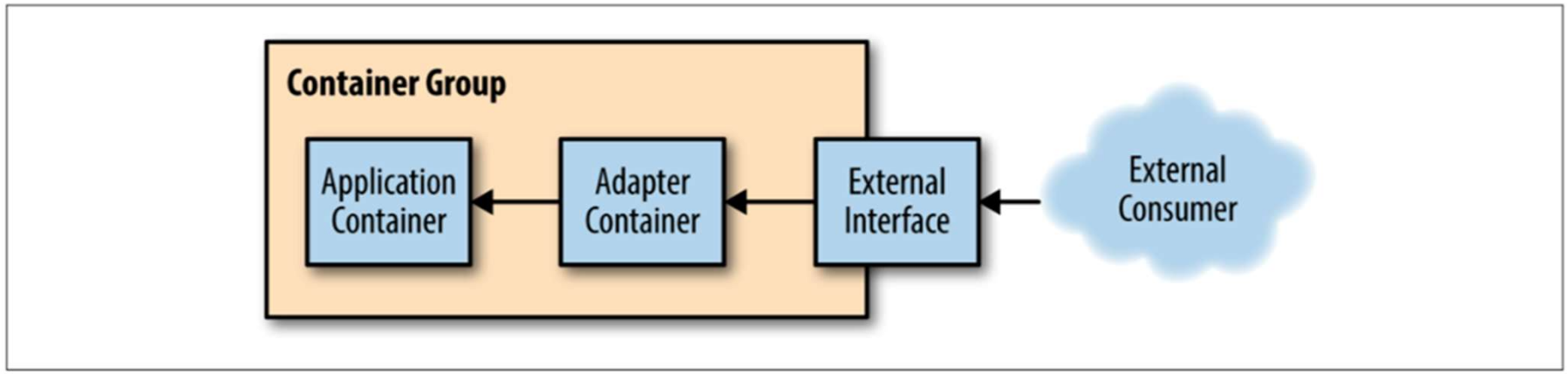


# Design Patterns – Ambassador

- Modular and reusable containers
- Reuse in other applications and environments
- Like a load balancer (tradeoffs)
- Examples:
  - Shard a service (service discovery)
  - Service brokering (IaaS or PaaS DBs)
  - A/B Testing (split traffic)

# Design Patterns – Adapter

- Application container
- Modify the interface of the application



# Design Patterns – Adapter

- Helps to set a standard in communication
- Adapts to provide a consistent interface
- Reusable modules (open source, internal, etc.)
- Examples:
  - Collect metrics
  - Parse and store logs (like Logstash, Fluentd)
  - Custom health checks of the application

# Design Patterns – Overview

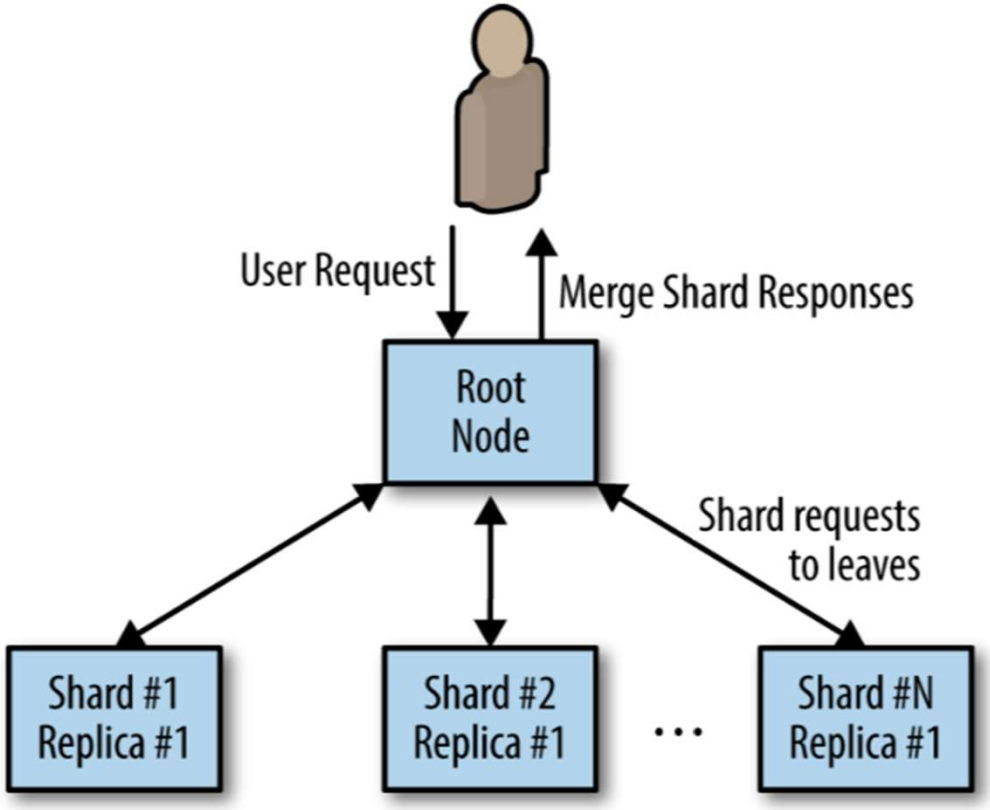
- Single Container: Isolate, Portable, etc.
- Sidecar: Extend behavior.
- Ambassador: Broker or proxy to the exterior.
- Adapter: Consistent communication.

# Design Patterns – Multi-container

- Scale at the Node Level
- Loosely Coupled
- Communication with Network Calls
- Parallel Calls Natively
- Base for Microservices (stateless)

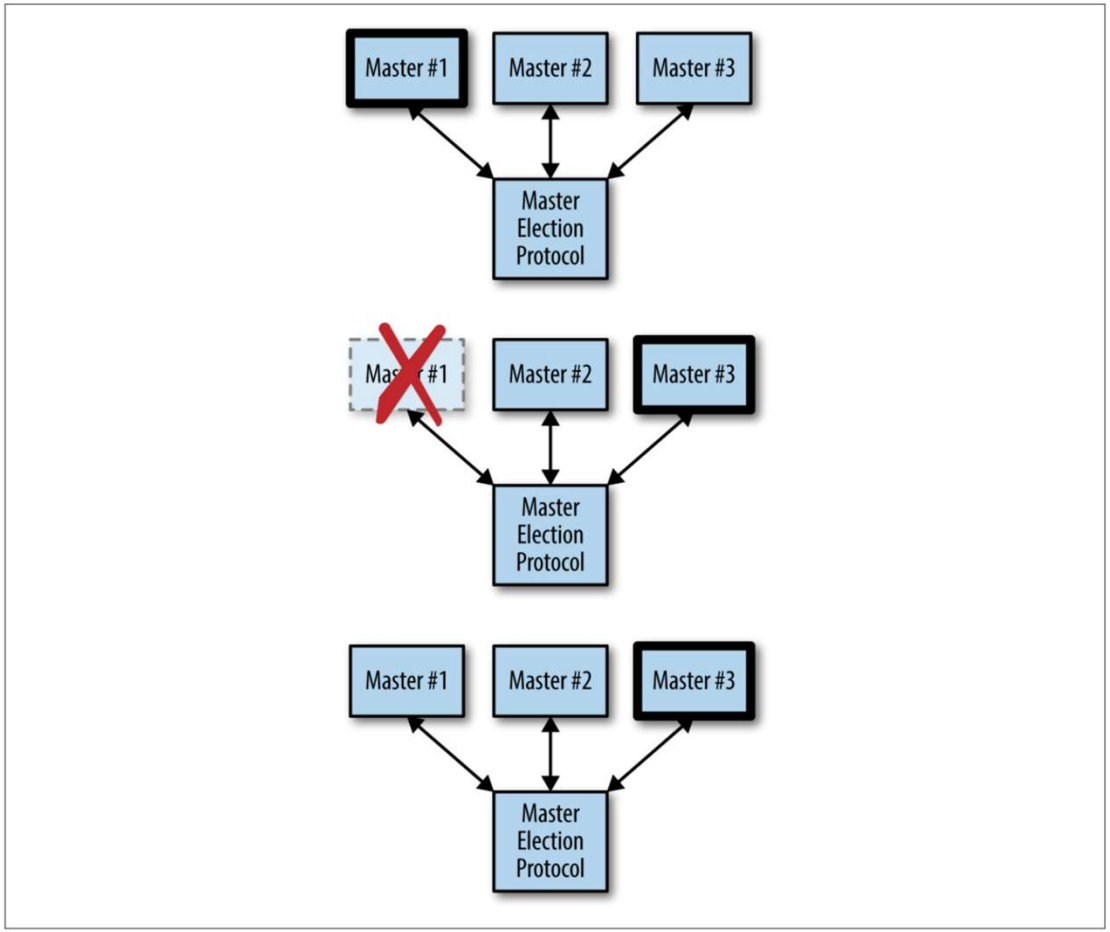
# Design Patterns – Scatter/Gather

- Split a big task
- Simultaneously
- Combined results



# Design Patterns – Leader/Election

- Master #1 is selected
- Master #1 fails
- Master #3 is selected

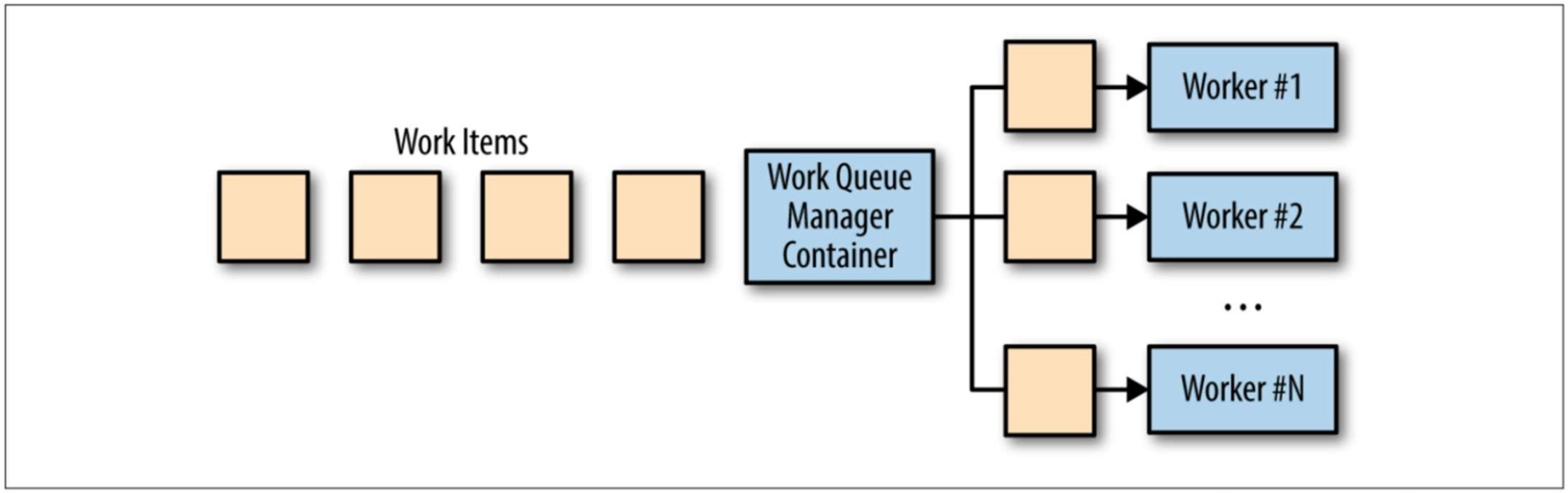


# Design Patterns – Leader/Election

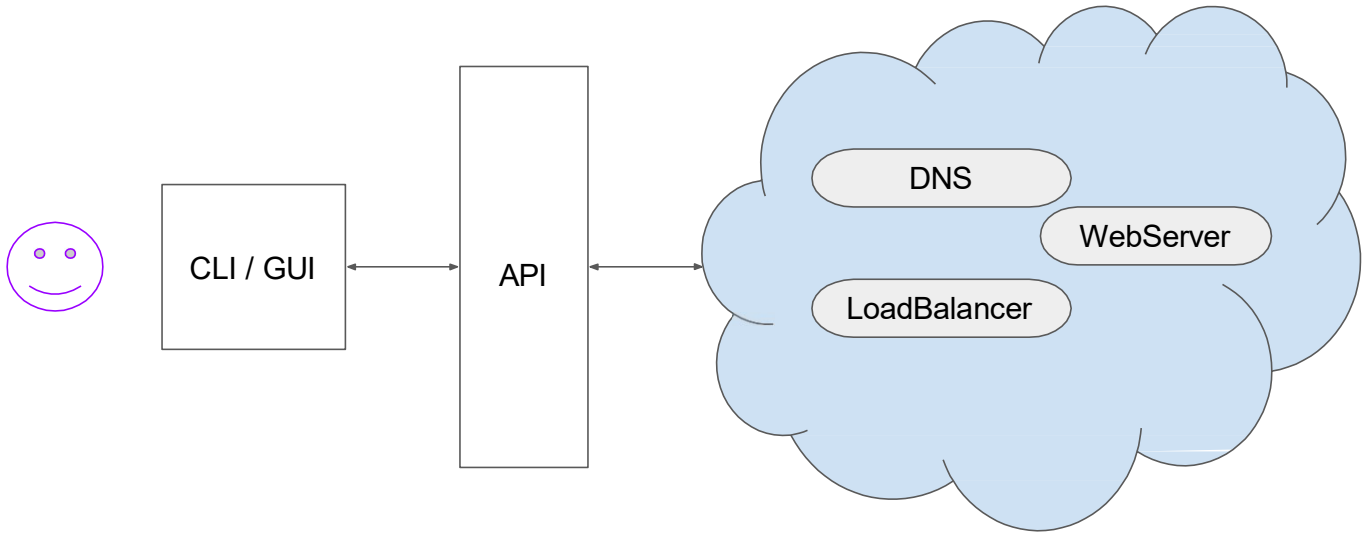
- With Kubernetes:
  - If container crash, it's recreated
  - If container hangs, it's restarted (health check)
  - If node fails, container it's re-scheduled
- But applications might need a high SLA
- Reuse, don't reinvent the wheel
- Algorithms like Paxos or RAFT
- Container implementations
  - etcd
  - Zookeper
  - consul

# Design Patterns – Work Queue

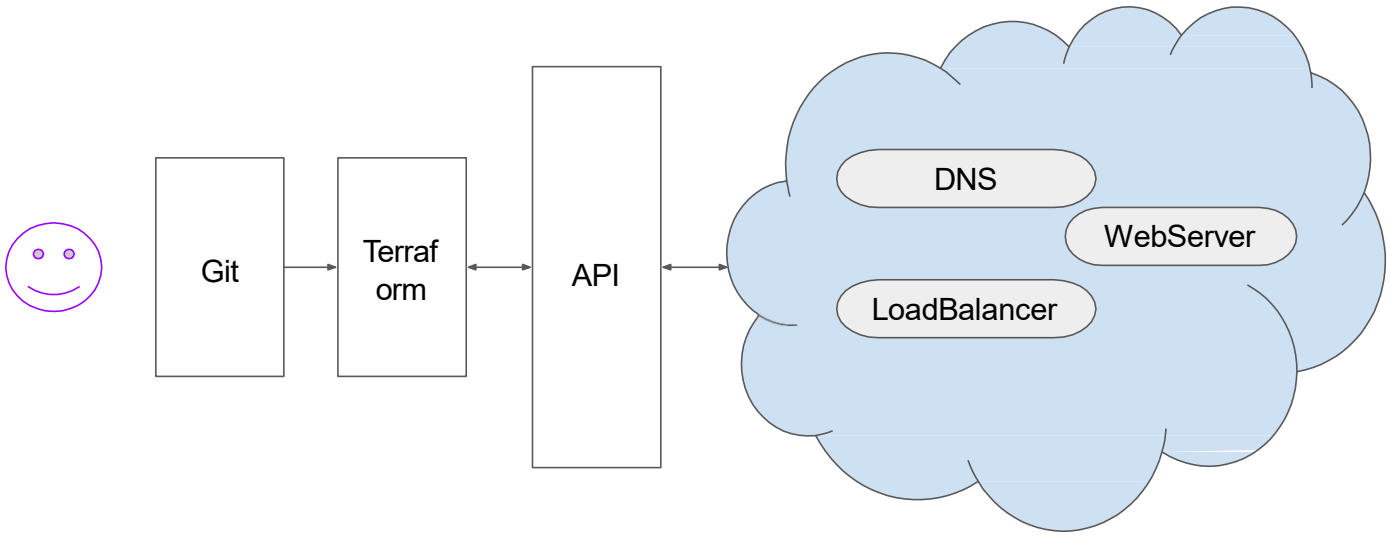
- Manager Container (one)
- Worker Container (multiple)



# Infra Provisioning - Manual

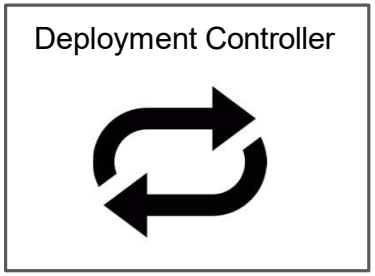


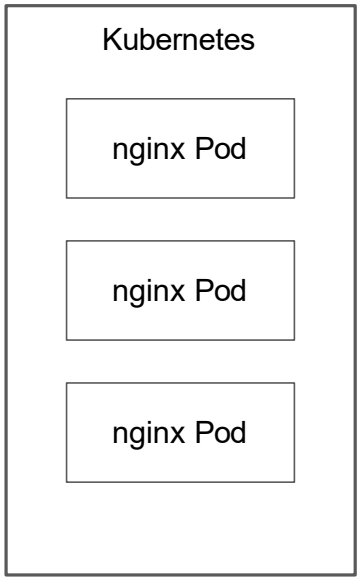
# Infra Provisioning – Configuration Mgmt.



# Kubernetes Controllers

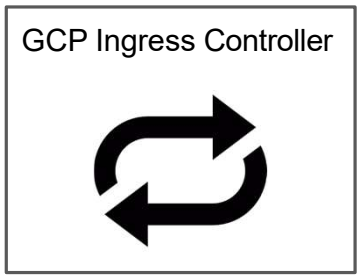
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9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

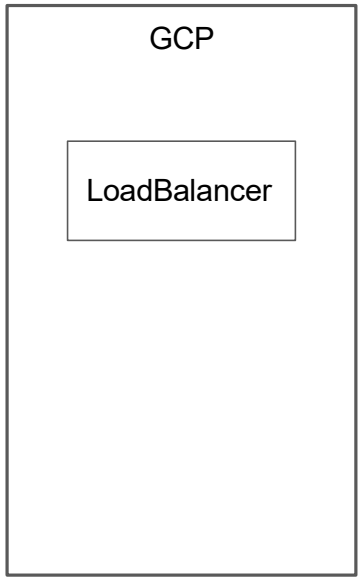


# Kubernetes Controllers

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```



# Kubernetes Custom Resources

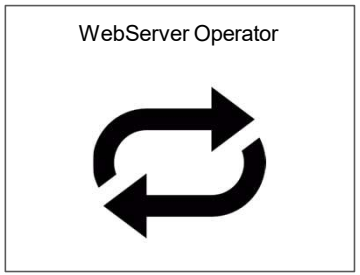
- Extensions of the Kubernetes API
- Define your own “Kind”
- Use capabilities of kubernetes objects. (RBAC, Finalizers, Versioning, Watches,...)
- Kubectl support

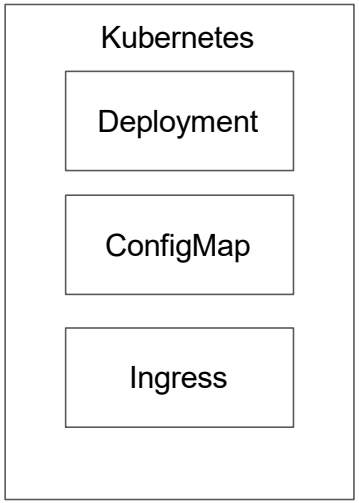
# Kubernetes Custom Resources

```
apiVersion: webser.attila.com/v1
kind: WebServer
metadata:
  name: mywebserver
spec:
  www: <html>hello world</html>
status:
  url: http://34.2.44.5/myweb
```

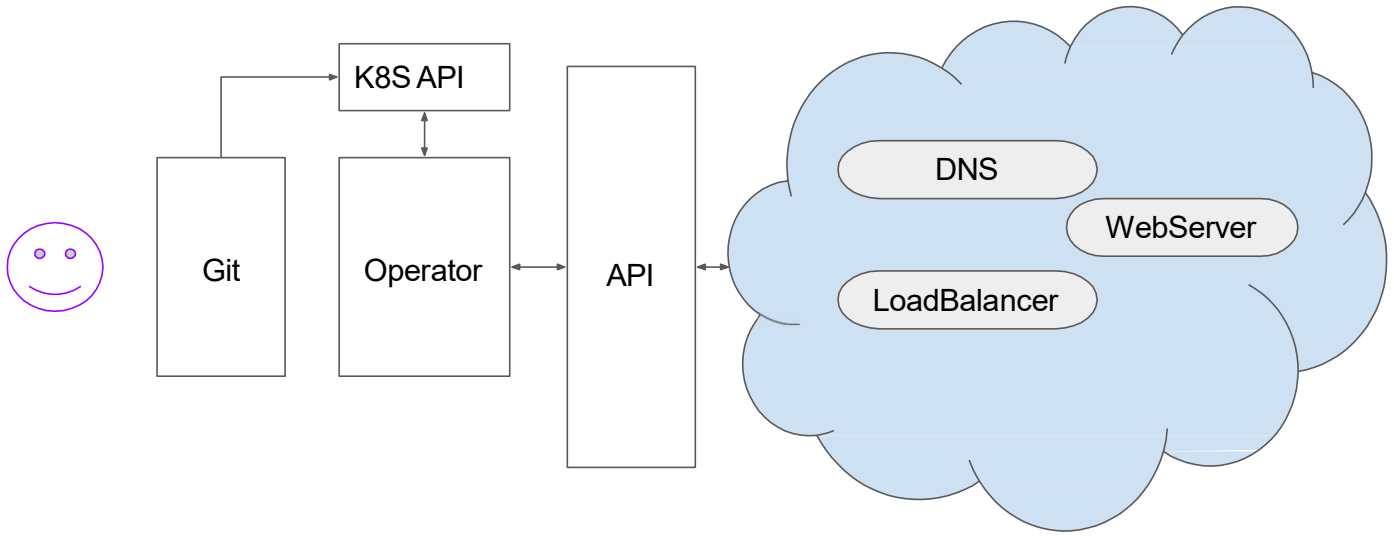
# Kubernetes Custom Resources

```
apiVersion: webser.attila.com/v1
kind: WebServer
metadata:
  name: mywebserver
spec:
  www: <html>hello world</html>
status:
  url: http://34.2.44.5/myweb
```



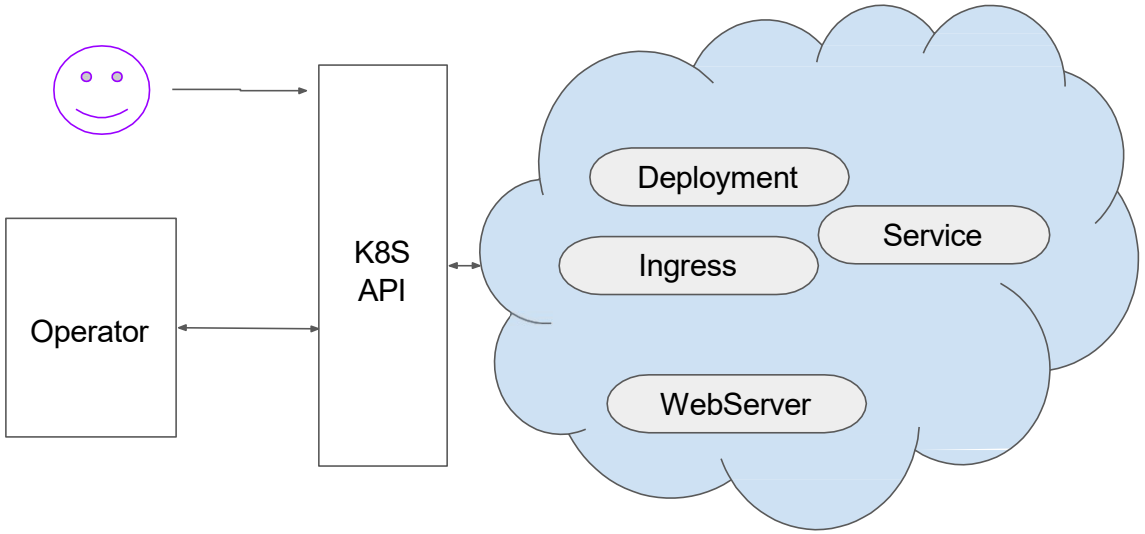
# Infra Provisioning - Operators



# Operators

- Just a pattern - Kubernetes doesn’t know about your Operator
- Just a pattern - not trivial to implement
- It’s how Kubernetes Controllers work
- We have frameworks: operator-sdk (Go), java-operator-sdk (JDK)

# Infra Provisioning - Operators



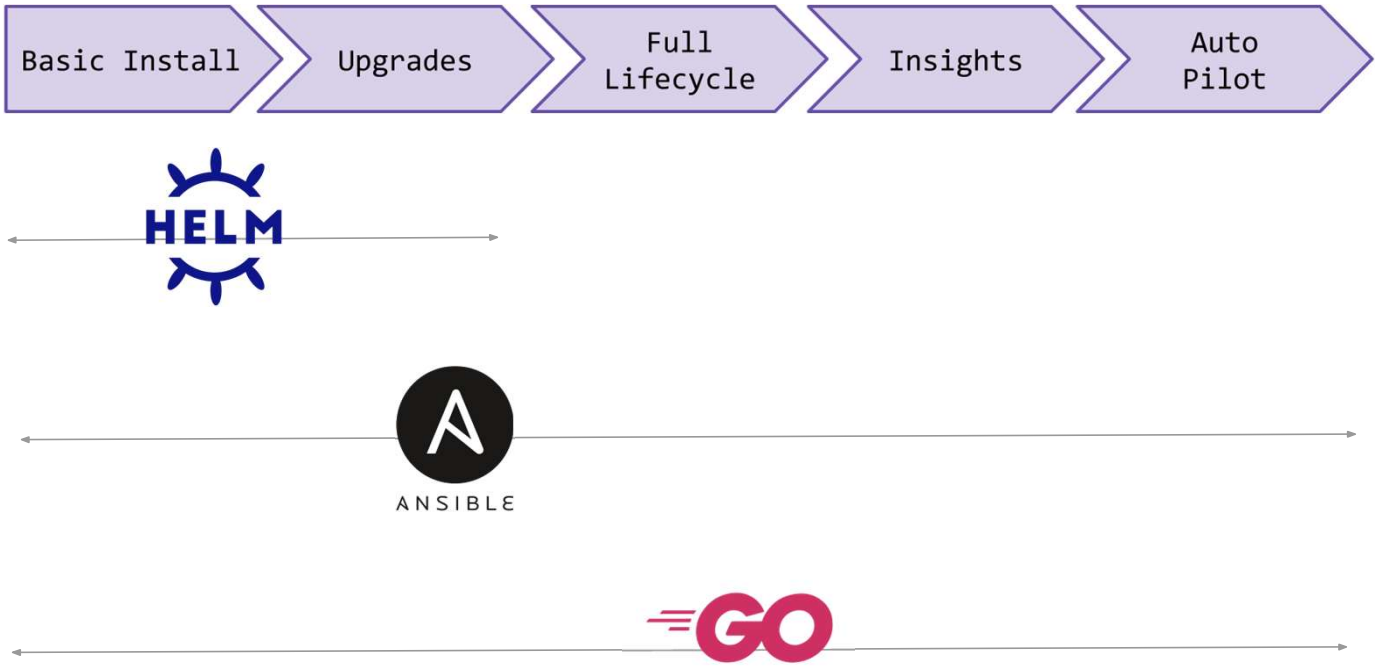
# Operators

- Operators are a very robust and flexible way to write provisioning logic
- Use case 1: Provision stateful resources on Kubernetes
- Use-case 2: Provision complex systems

# Operators

- Deploying an application on Demand
- Taking and Restoring Backups of app State
- Handling Upgrades including DB Schema changes and/or config setting
- Publish Services to Applications that do not support Kubernetes API to discover them
- Simulating Failure in all or parts of your system
- Choosing Leader for distributed apps without election process

# Operators



# Deploying Apps

Deploy Applications with and without Helm

# Q &A

# We're done!

## Thank you for your time and participation.