Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Kubernetes

Dr. Assane Wade
Francisco Mendonca

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

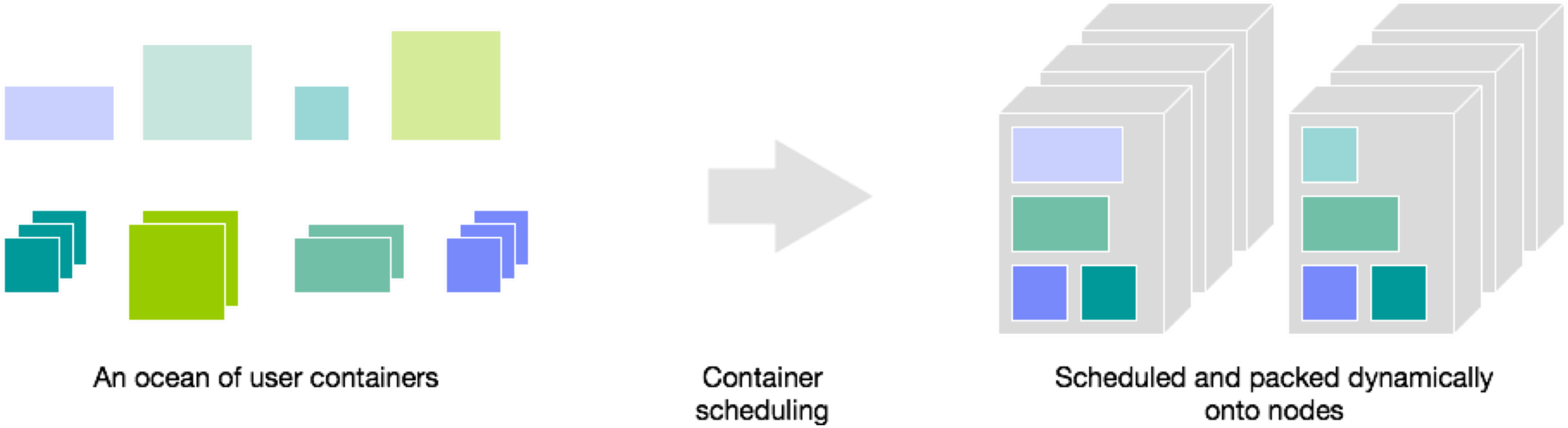
Container cluster management

- Container management refers to a set of practices that govern and maintain containerization software.
- Container management tools automate the creation, deployment, destruction and scaling of application or systems containers.

Container cluster management

- Need to create containers and connect them between each other.
- Putting all containers on a single host is not robust: when the host fails all containers fail.
- Deploy on many hosts with communication between the containers.
- Deploy a new version without service interruption.
  - Continuous integration
- Take a host down for maintenance without service interruption.
- Be sure that containers are healthy at all times
- Need to monitor containers for good health and relaunch if failing: container failure management

Container cluster management



- The placement of application containers on cluster nodes is called *scheduling*
  - Containers have different resource requirements (CPU, memory, disk, …)
  - Containers that cooperate tightly need to be placed on the same node *(affinity)*
  - Containers of an app that uses redundancy need to be placed on different nodes *(anti-affinity)*
- Goals:
  - Increase cluster utilization
  - Still meet the application constraints

Kubernetes - Introduction

- Platform for automating deployment, scaling and management of containerized applications.
- Initially written and designed by Google
  - First announced (open-sourced by Google) 2014-09
  - Kubernetes v. 1.0 released 2015-07-21
  - At the same time Google and the Linux Foundation create the Cloud Native Computing Foundation (CNCF) and transfer Kubernetes to it
- Borrows heavily from Google's long experience with managing containers, namely Google's Borg System
- Adopted by RedHat for OpenShift, CoreOs for Tectonic, Rancher labs for Rancher, …
- Available in public clouds: Google Container Engine, Rackspace, Azure, OpenShift, Bluemix, OpenStack, etc.

Kubernetes (κυβερνήτης): Greek for "helmsman" or "pilot", pronounced 'koo-ber-nay'-tace'

Kubernetes - Main concepts

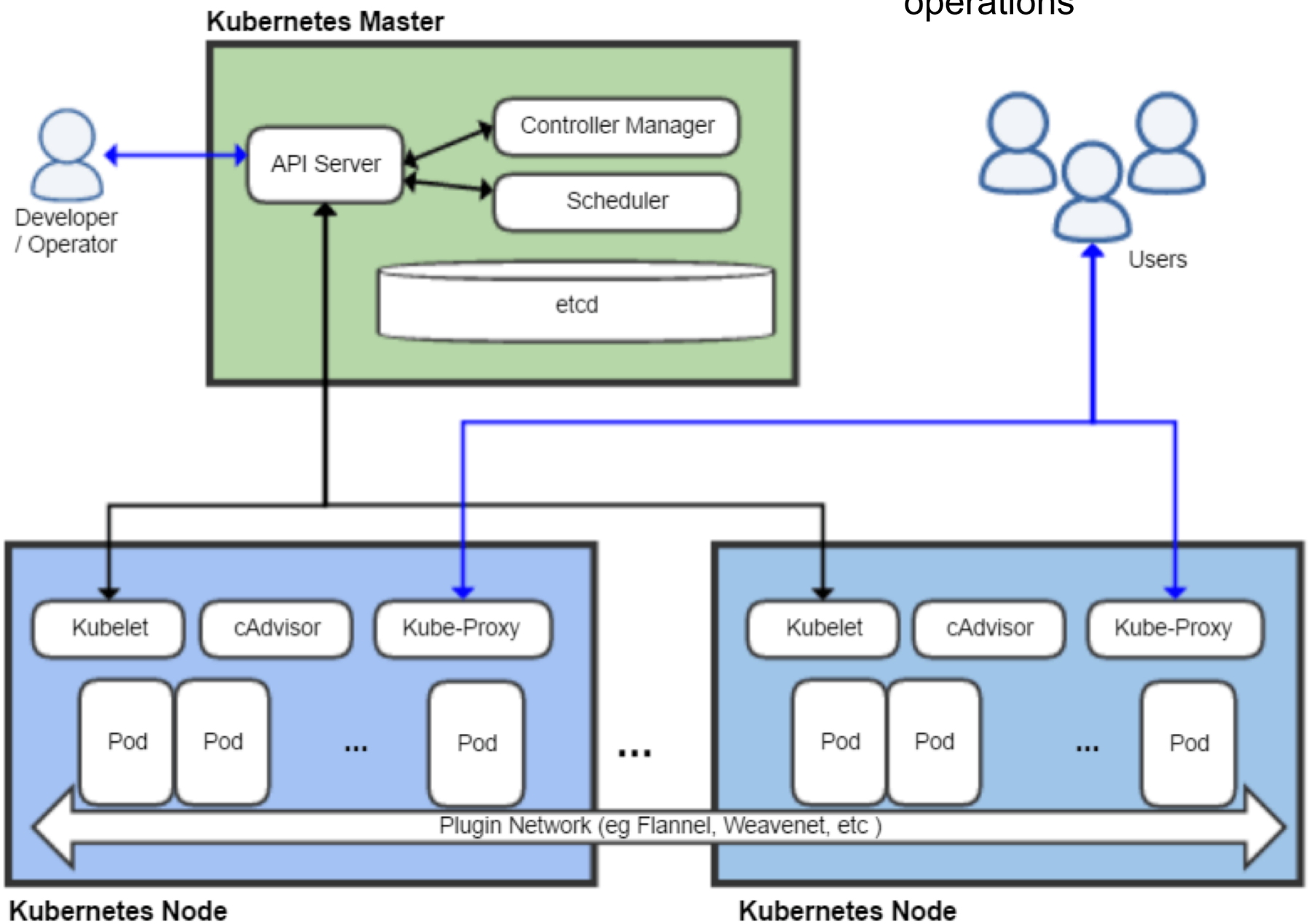
- **Cluster** — Set of machines (physical or virtual) where pods are deployed, managed and scaled.
- **Pod** — A pod consists of one or more containers that are guaranteed to be co-located on the same machine.
- **Controller** — A controller is a reconciliation loop that drives actual cluster state toward the desired cluster state.
- **Replication Controller** — Handles replication and scaling by running a specified number of copies of a pod across the cluster.
- **Service** — Set of pods that work together, such as one tier of a multi-tier application. Kubernetes provides:
  - **Label** — The user can assign key-value pairs (called labels) to any API object in the system (e.g., pods, nodes).
  - **Label selector** — A query against a label that returns matching objects.

Kubernetes - Main concepts

- A cluster is a set of nodes (physical or virtual machines) running Kubernetes agents, managed by the control plane. Kubernetes v1.28 supports clusters with up to 5,000 nodes. More specifically, Kubernetes is designed to accommodate configurations that meet *all* of the following criteria:
- No more than 110 pods per node
- No more than 5,000 nodes
- No more than 150,000 total pods
- No more than 300,000 total containers

Kubernetes - Anatomy of a cluster

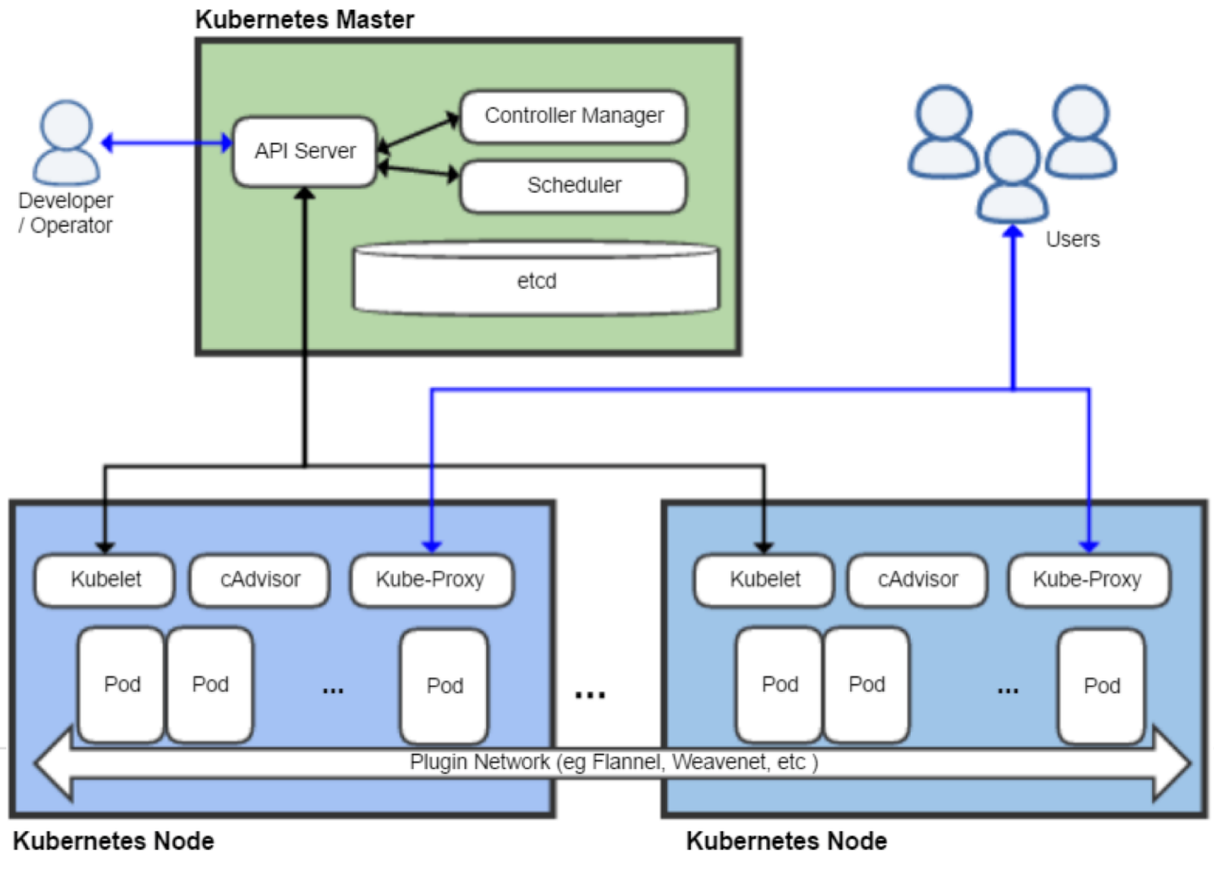
etcd (**Cluster Data Store)** provides a REST API for CRUD (Create Read Update Delete) operations



Source: Wikipedia

Kubernetes - Anatomy of a cluster

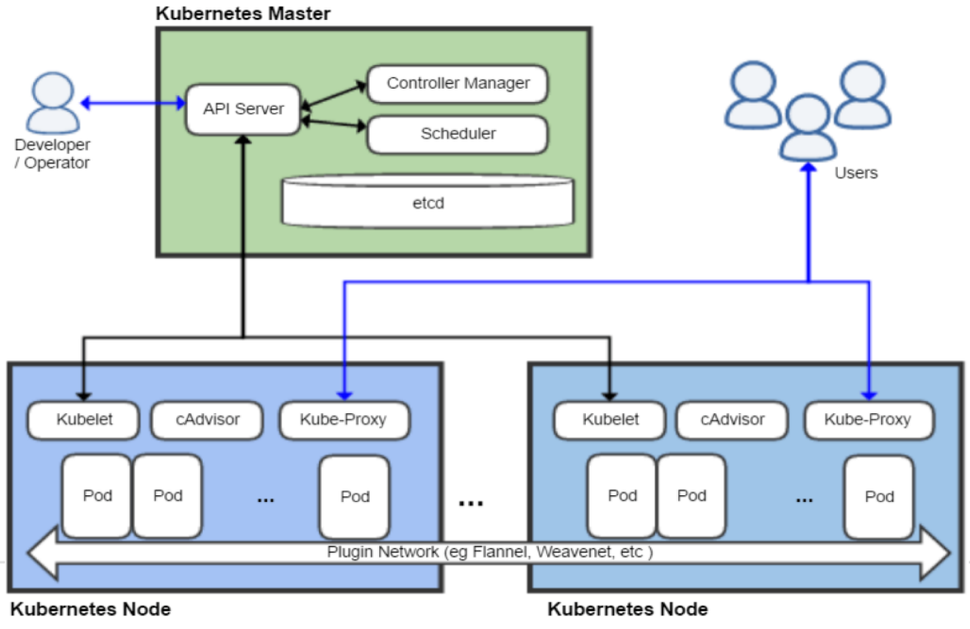
- Components in the **Master node** (aka Control Plane)
  - **etcd** — Key/value store keeping the configuration data of the cluster, representing the overall state of the cluster at any given point of time.
  - **API Server** — Serves the K8s API using JSON/HTTP (external and internal).
  - **Scheduler** — Selects which node an unscheduled pod should run on, based on resource availability.
  - **Controller Manager** — a process that runs core Kubernetes controllers. The controllers communicate with the API server to create, update, and delete the resources they manage (pods, service endpoints, etc.).



Kubernetes - Anatomy of a cluster

Components in the **Worker nodes** aka Kubernetes nodes, Minions

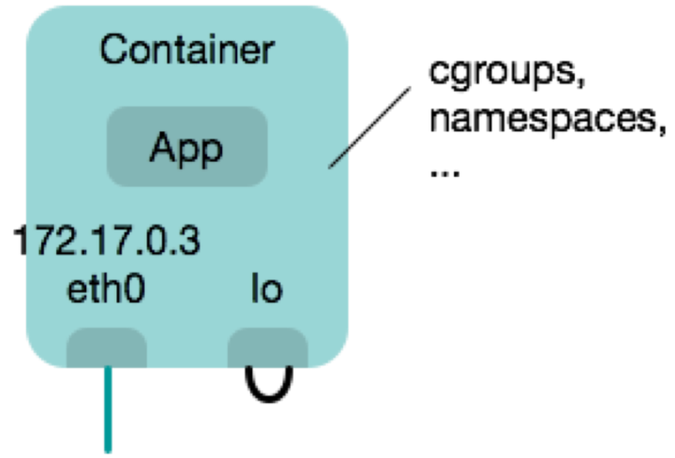
- **Kubelet** — Responsible for the running state of each node, i.e. ensuring that all containers on the node are healthy. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.
- **Kube-proxy** — A network proxy and load balancer. Responsible for routing traffic to the appropriate container based on IP and port number of the incoming request.
- **cAdvisor** — Agent that monitors and gathers resource usage and performance metrics of the containers.
- **Overlay network** — Responsible to connect containers on different nodes on a flat network. Not part of K8s, pluggable.



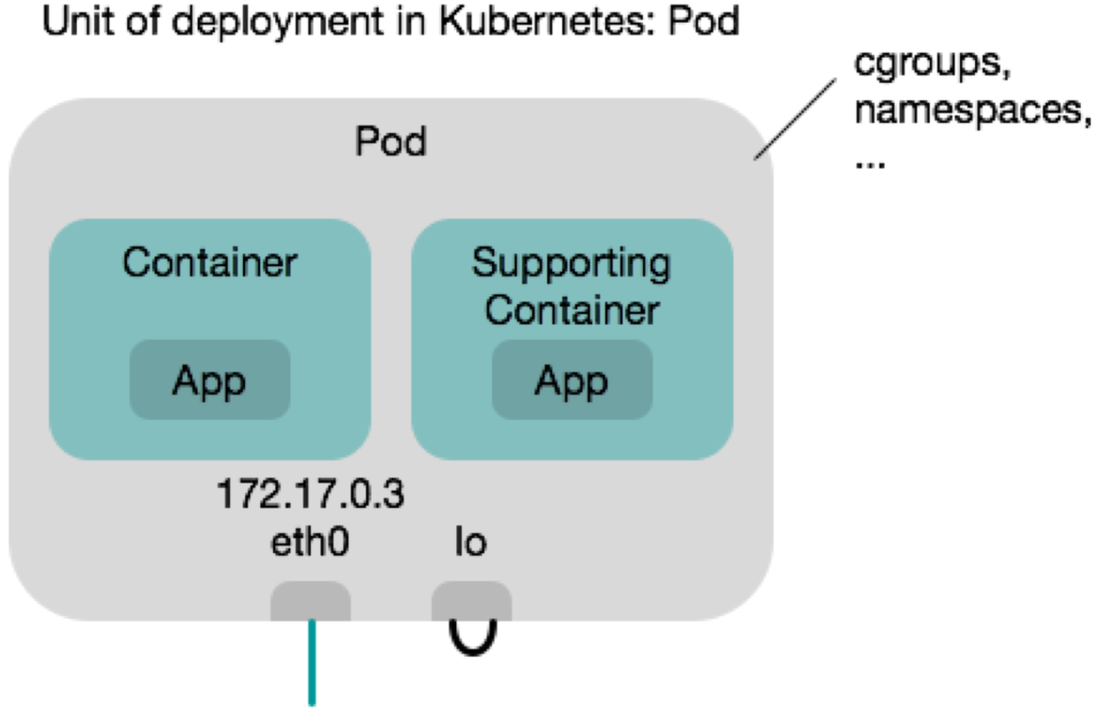
Kubernetes - Pods

- The atomic unit of deployment in Kubernetes is the **Pod**.
- A Pod contains one or more containers. The common case is a single container.
- If a Pod has multiple containers
  - Kubernetes guarantees that they are scheduled on the **same** cluster node.
  - The containers **share** the same Pod environment
    - IPC namespace, shared memory, storage volumes, network stack, etc.
    - IP address
- If containers need to talk to each other **within** the Pod, they can simply use the *localhost* interface.

Unit of deployment in Docker: Container



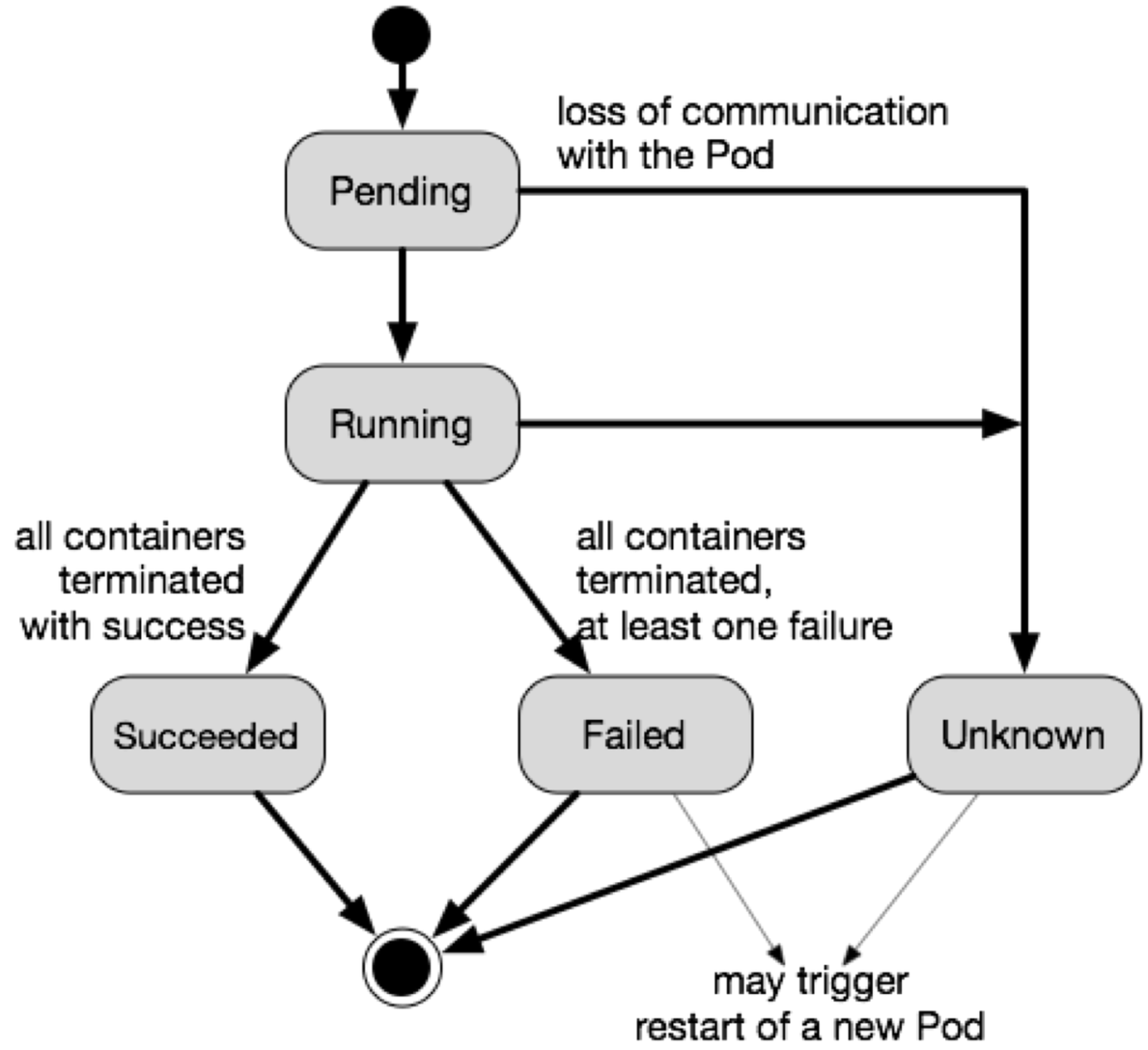
Unit of deployment in Kubernetes: Pod



Kubernetes - Pods

- Pod deployment **atomicity**
  - Pods are the minimum unit of scaling.
  - The deployment of Pod is all or nothing: Either the entire Pod comes up and gets put into service, or it doesn't and fails.

- Pod **lifecycle**
  - When a Pod dies (e.g., a container of the Pod crashes or the cluster node containing the Pod crashes) one does not bother to bring it back to life.
    - Instead Kubernetes starts another one in its place (new Pod ID and IP address).
    - In the pets vs. cattle model, Pods are treated as cattle.



Kubernetes - Common concepts

- Kubernetes adopts a consistent **object API**
- Every Kubernetes object has three basic fields in its description: Object Metadata, Specification (or Spec), and Status (once it has been created).
- The **Object Metadata** is the same for all objects in the system
  - it contains information such as the object's name, UID (unique identifier), an object version number, and labels (key-value pair).
- **Spec** is used to describe the **desired state** of the object. It is a read-only information about the **current state** of the object

```
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    component: redis
    app: todo
  resourceVersion: "439780"
  uid: 145f56fd-ad1b-11e7-9[…]
spec:
  containers:
  - name: redis
    image: redis
    ports:
    - containerPort: 6379
    resources:
      limits:
        cpu: 100m
    args:
    - redis-server
    - --requirepass ccp2
    - --appendonly yes
status:
  hostIP: 172.20.52.100
  phase: Running
  podIP: 100.96.3.9
  qosClass: Burstable
```

Specified by the user

Provided by kubernetes

Kubernetes – Yaml Definition of Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: www
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: /srv/www
      name: www-data
      readOnly: true
  - name: git-monitor
    image: kubernetes/git-monitor
    env:
    - name: GIT_REPO
      value: http://github.com/some/repo.git
    volumeMounts:
    - mountPath: /data
      name: www-data
volumes:
- name: www-data
  emptyDir: {}
```

Name of Pod

Internal name of the container in the Pod

Container image name

Environment variable name and value for the container (docker run -e GIT_REPO=... ...)

Volume name and type

Kubernetes - Manifest files

- Manifest files.
  - kubectl create -f file.yaml
  - File format JSON, which can also be written as YAML
- **kind**: a string that identifies the schema this object should have
- **apiVersion**: a string that identifies the version of the schema the object should have
- **metadata**: metadata associated with object
  - **name**: uniquely identifies this object within the current namespace
  - **labels**: a map of string keys and values that can be used to organize and categorize objects
- **spec**: specification of object's desired state

Manifest for a Pod

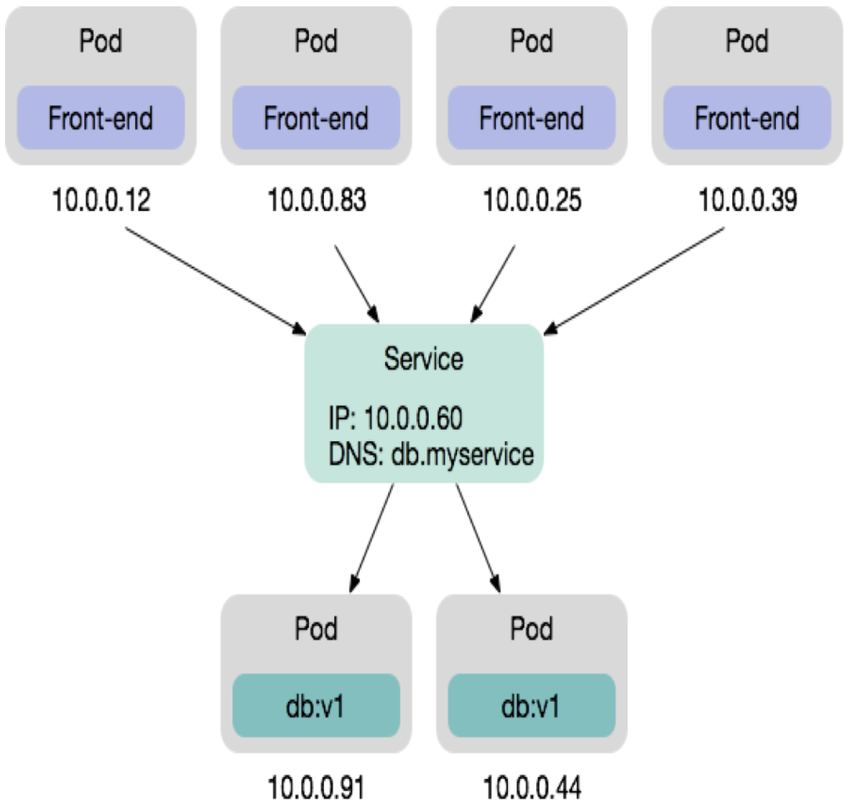
```
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    component: redis
    app: todo
spec:
  containers:
  - name: redis
    image: redis
    ports:
    - containerPort: 6379
    resources:
      limits:
        cpu: 100m
    args:
    - redis-server
    - --requirepass ccp2
    - --appendonly yes
```

Pod usage examples

- Create a Pod declaratively
  - `kubectl create -f pod-nginx.yaml`
- List all Pods
  - `kubectl get pods`
- Delete Pod by name
  - `kubectl delete pod nginx`

Kubernetes - Service

- When replacing, scaling or upgrading Pods services receive new IP addresses every time.
  - Suppose a two-tier app with a front-end Pods talking to back-end Pods. The front-end Pods cannot rely on the IP addresses of the back-end Pods.
- **Services** provide a reliable networking endpoint for a set of Pods.



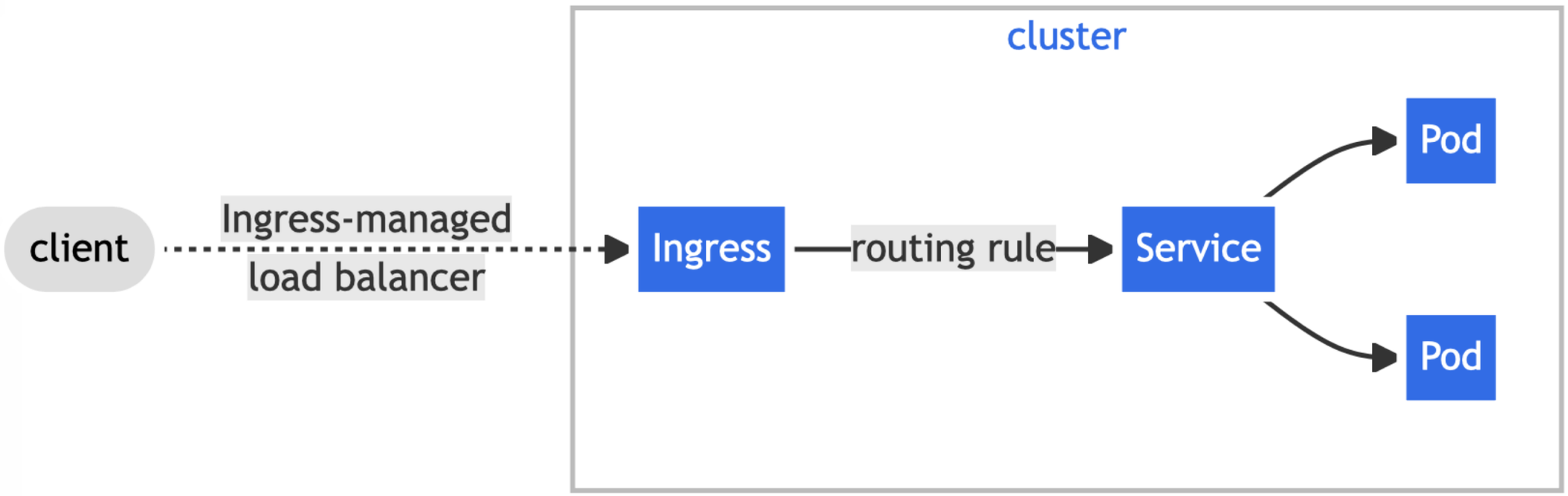
Kubernetes - Service definition

- **Service Type** can be:
  - **ClusterIP**: use a cluster-internal IP only - this is the default. Choosing this value means that you want this service to be reachable only from inside of the cluster.
  - **NodePort**: on top of having a cluster-internal IP, expose the service on a port on each node of the cluster (the same port on each node). You'll be able to contact the service on any <NodeIP>:NodePort address.
  - **LoadBalancer**: on top of having a cluster-internal IP and exposing service on a NodePort also, ask the cloud provider for a load balancer which forwards to the Service exposed as a <NodeIP>:NodePort for each Node.
  - **ExternalName** : Maps the Service to the contents of the externalName field (for example, to the hostname api.foo.bar.example). The mapping configures your cluster's DNS server to return a CNAME record with that external hostname value. No proxying of any kind is set up.

Ingress

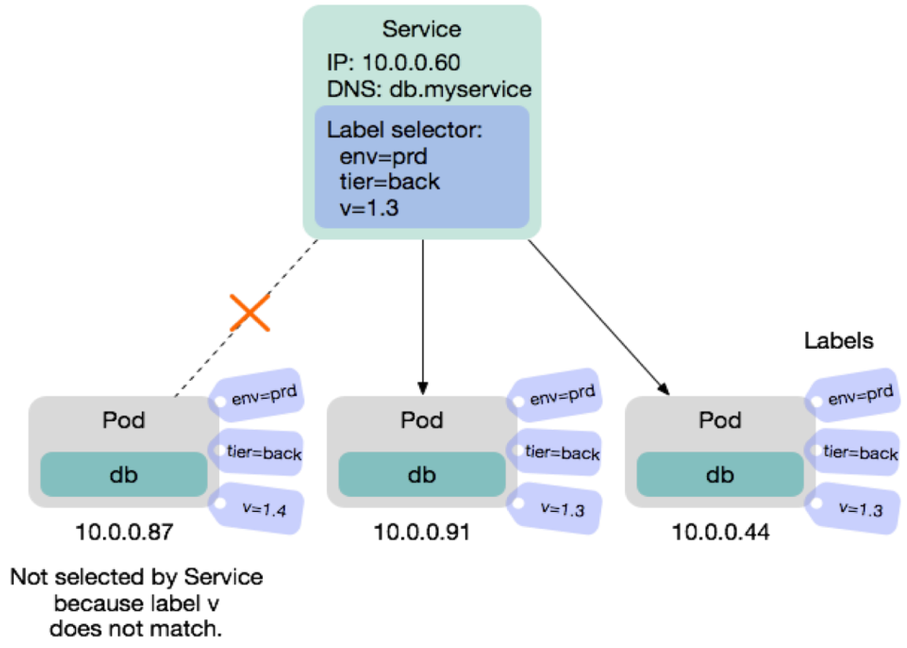
Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

An Ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type=NodePort or Service.Type=LoadBalancer.



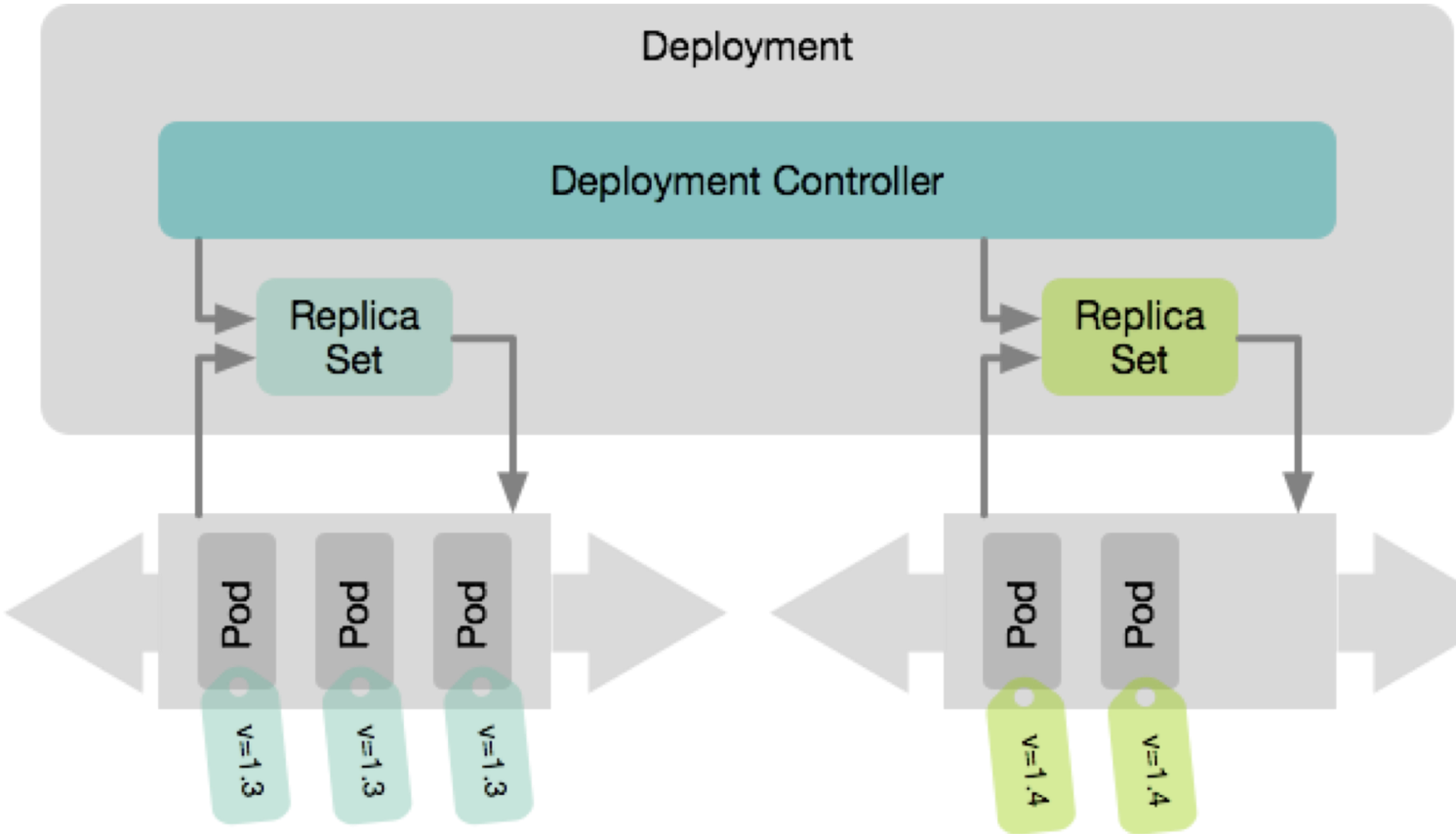
Kubernetes - Connecting Pods to a Service via Labels

- Labels are key-value pairs attached to a Kubernetes object. (Ex. "environment" : "dev", "environment" : "qa", "environment" : "production")
- Key and value can be freely chosen.
- An object may have several labels. The same label may be attached to several objects.
- When defining the Service one specifies a **Label Selector** which is a set of conditions on the label key-values.
  - Pods matched by the Label Selector are connected to the Service.



Kubernetes - Deployments

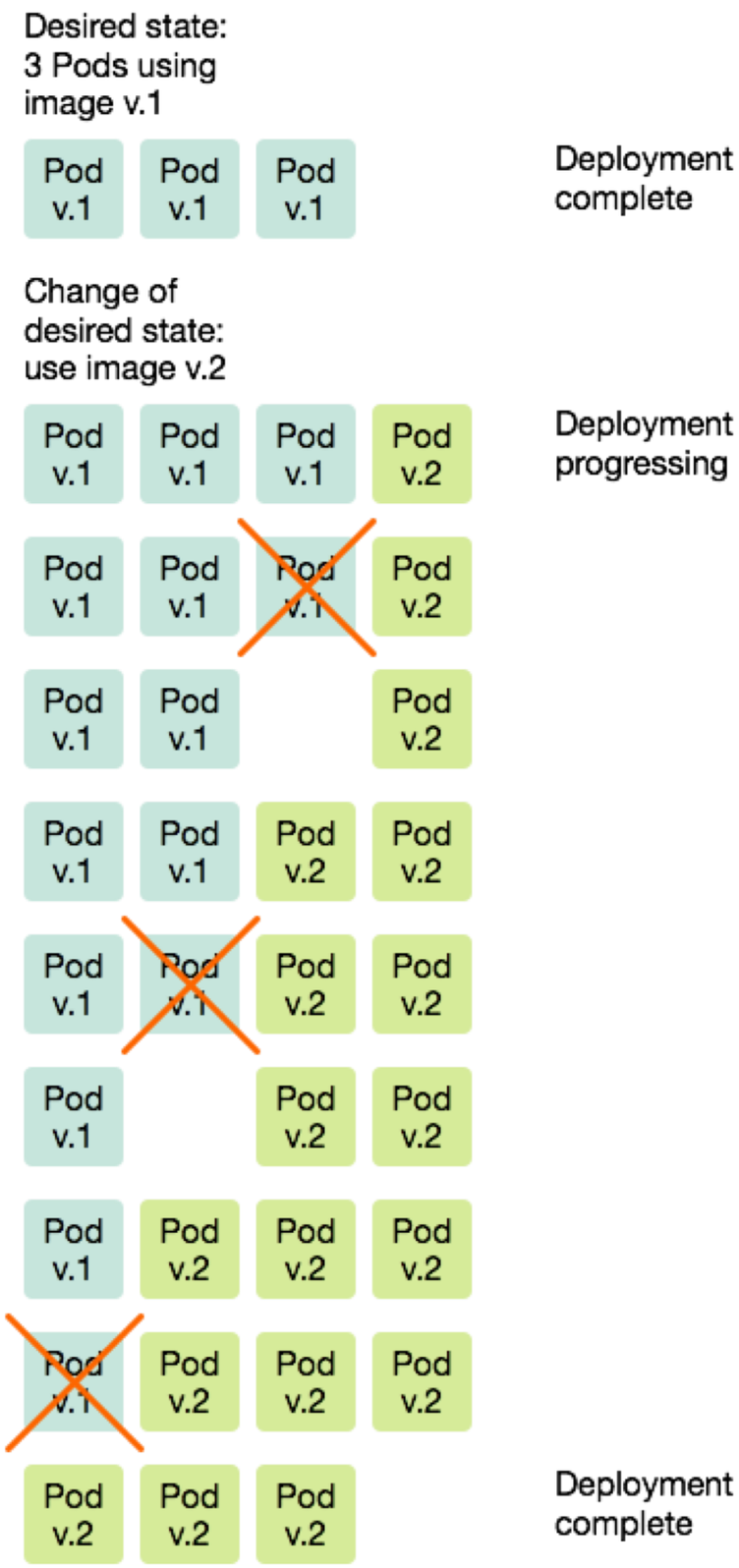
- **Deployments** add features and functionality used to enable updating the deployed software without interrupting the service.
  - Rolling updates
  - Rollbacks
- Deployments make use of **Replica Sets**
  - A Deployment may have several Replica Sets active at the same time when performing updates or rollbacks.



- Deployments work according to the principle of *Desired State Configuration*.
  - The user describes the desired state in a Deployment object.
  - The Deployment Controller compares the desired state to the actual state.
  - The Deployment Controller changes the actual state towards the desired state at a **controlled rate**.
- When the deployment of a new version fails it is very simple to rollback to a previous working version.

Kubernetes - Deployments

- In a **rolling update** the service is never interrupted.
- To trigger a rolling update the user simply updates the desired state of the deployment.
  - For example specify a different version of the Pod image.
  - The Deployment creates a new ReplicaSet.
  - The Deployment Controller creates a new Pod in the new ReplicaSet, and when successful terminates a Pod in the old ReplicaSet.
  - This is repeated until no old Pods are left.



Deployment usage examples

- View deployment:
  - `kubectl get deployments`
  - Output:

Number of total replicas running

Number of updated replicas running (latest spec)

Number of replicas available to users

```
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         0         0            0           1s
```

- Check rollout status:
  - `kubectl rollout status deployment/nginx-deployment`
- Update by setting a new image version:
  - `kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1`
- Update by editing the deployment definition in an editor:
  - `kubectl edit deployment/nginx-deployment`

ConfigMaps and Secrets

- Both are used to propagate environmental variables through the cluster
- ConfigMaps are used for:
  - Non-Confidential Environmental Variables (No Passwords or API Keys)
  - Command-line arguments
  - Configuration files in volumes
- ConfigMaps are meant to be dynamic

ConfigMaps and Secrets

- Secrets are used to propagate **confidential** data through the cluster
  - Passwords
  - API Keys
  - ...
- Secretes are also Key/Value pairs
- Values are stored in base64

ConfigMaps - Code

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: api-pod-config
data:
redis_endpoint: redis-svc
redis_pwd: ccp2
```

Pod File with ConfigMaps

```
env:
  - name: REDIS_ENDPOINT
    valueFrom:
      configMapKeyRef:
        name: api-pod-config
        key: redis_endpoint
- name: REDIS_PWD
  valueFrom:
    configMapKeyRef:
      name: api-pod-config
      key: redis_pwd
```

Original Pod File

```
 env:
- name: REDIS_ENDPOINT
  value: redis-svc
- name: REDIS_PWD
  value: ccp2
```

Secrets - Code

```
apiVersion: v1

kind: Secret

metadata:

  name: api-secret

type: Opaque

data:

  redis_endpoint:
cmVkaXMtc3ZjCg==

  redis_pwd: Y2NwMg==
```

Pod File with Secrets

```
env:

  - name: REDIS_ENDPOINT

    valueFrom:

      secretMapKeyRef:

        name: api-pod-config

        key: redis_endpoint


- name: REDIS_PWD

  valueFrom:

    secretMapKeyRef:

      name: api-pod-config

      key: redis_pwd
```

Original Pod File

```
 env:

- name: REDIS_ENDPOINT

  value: redis-svc

- name: REDIS_PWD

  value: ccp2
```

References

- https://kubernetes.io/docs/concepts/configuration/secret/
- https://kubernetes.io/docs/concepts/configuration/configmap/