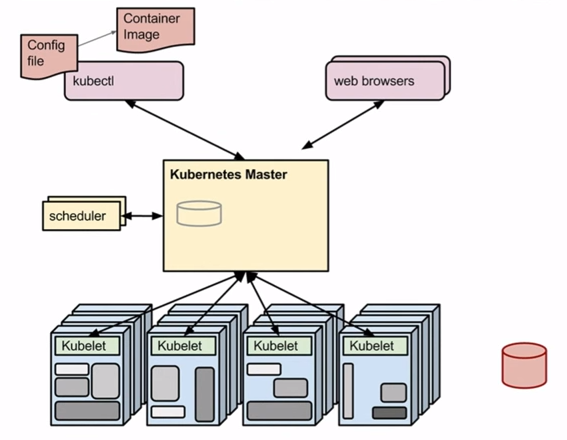
**Kubernetes Notes**

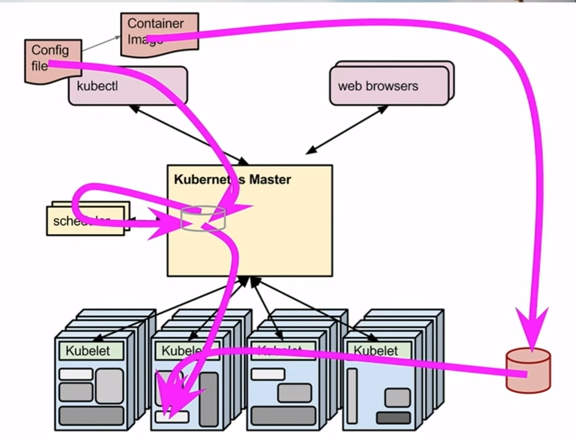
<https://www.youtube.com/watch?v=kT1vmK0r184>

Kubernetes is:

* Google Borg architecture open sourced
* Container orchestrator
* Runs containers
* Supports multiple cloud and bare-metal environments
* Open source, written in Go
* Manage applications, not machines



* Steps
  1. Create a container (docker or rocket) image of what you need to run
  2. Push that to a central repository (dockerhub or private repo)
  3. Push the config file to Kubernetes Master (how the application will be run)
  4. Through the scheduler it figures out which machines it is managing, and schedules the containers to be run



Config file

spec:

containers:

-name: myservice

image: myservice

resources:

limits:

memory: “128Mi”

cpu: “0.1”

ports:

* containerPort: 3306

protocol: TCP

replicas: 10000

When kubernetes spins up instances, each has a unique name/IP address.

Implementation

1. Starting a pod

kubectl run <pod-name> --image=<docker image name including repo name> -l <comma-separated key-value pairs for labels>

1. Scaling a pod

kubectl scale rc <pod-name> --replicas=<#of replicas to scale to>

Looking at pods

kuberctl get pods

1. Creating a service

kuberctl expose rc <service-name> --port=<port#> --type=LoadBalancer

The way the requests are routed are based on labels.

Question: Where are we mapping the labels with the service?

1. Describing a service

kuberctl describe service <service-name>

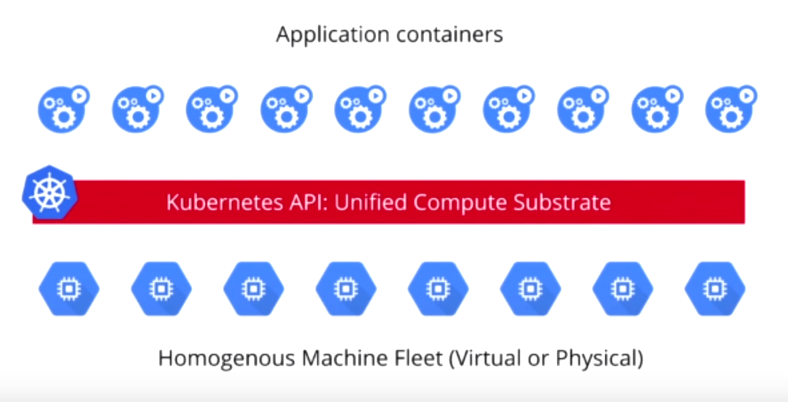
Updating the code

* In kubernetes, rolling updates are easy
* Steps for rolling updates
  1. Update the code, test, push it to repository
  2. Ask kubernetes Master to do a rolling update

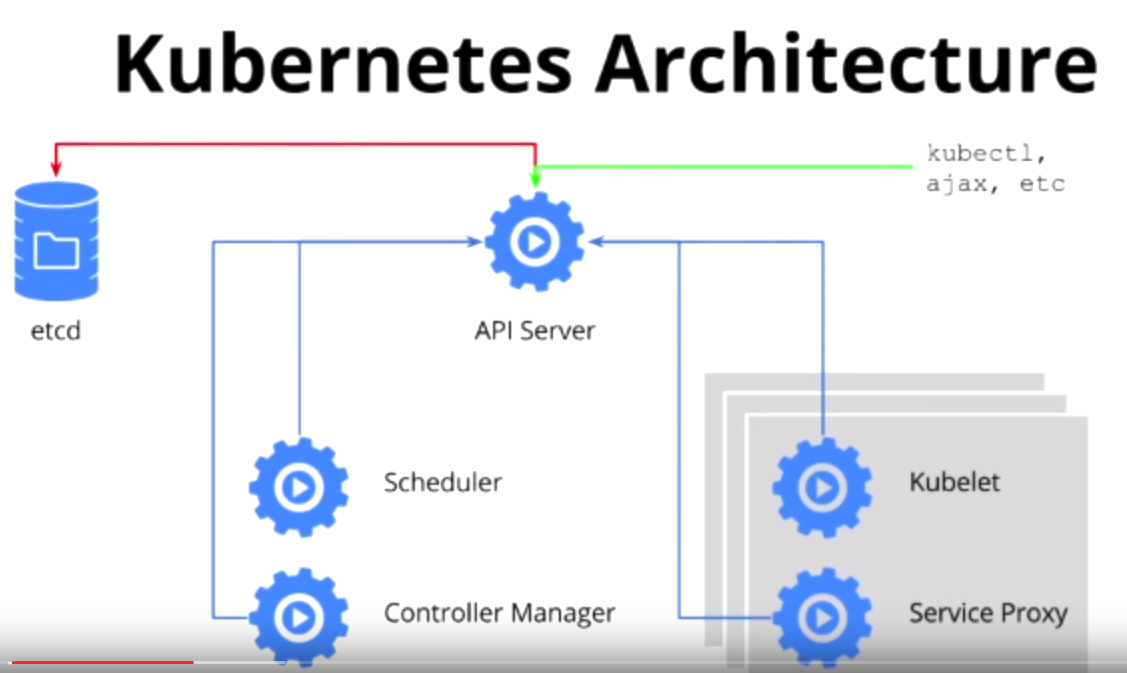
kubectl <service-name> --image=<name of the repo/image to use> --update-period=<time interval for update between instances>

Overview of Kubernetes

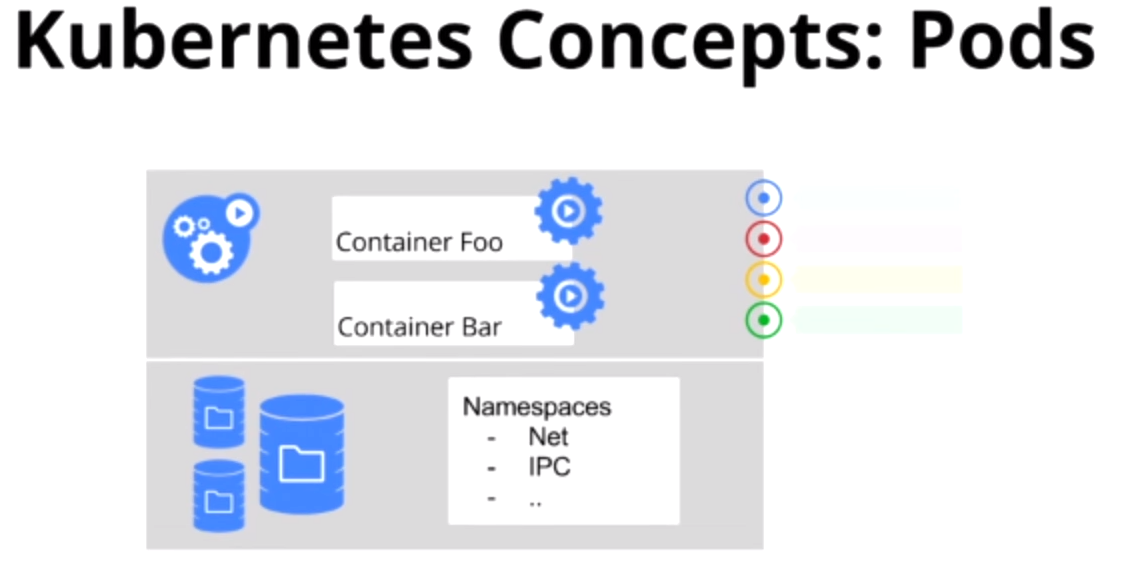
<https://www.youtube.com/watch?v=WwBdNXt6wO4>



* Application containers view the world as a sea of compute.
* Kubernetes takes every machine look exactly the same to the application containers

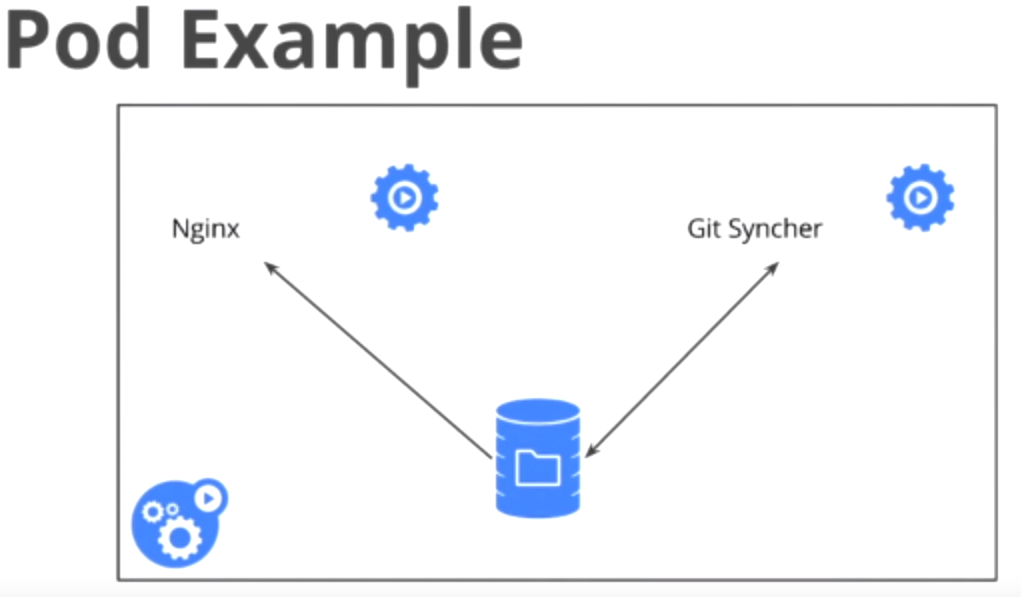


* API Server
  + HTTP restful APIs
  + Swagger definitions
  + Server is stateless, can be replicated, scaled, etc.
  + etcd provides storage behind the scenes
* Scheduler
  + Scheduling pods/containers out on the machine
* Controller Manager
  + Health maintenance
  + Reconciliation and self-healing
  + When something goes wrong, this component
* Kubelet
  + Daemon Managing the host/container itself
* Service proxy
  + Load balancing and separation of conceptual idea of fe and be



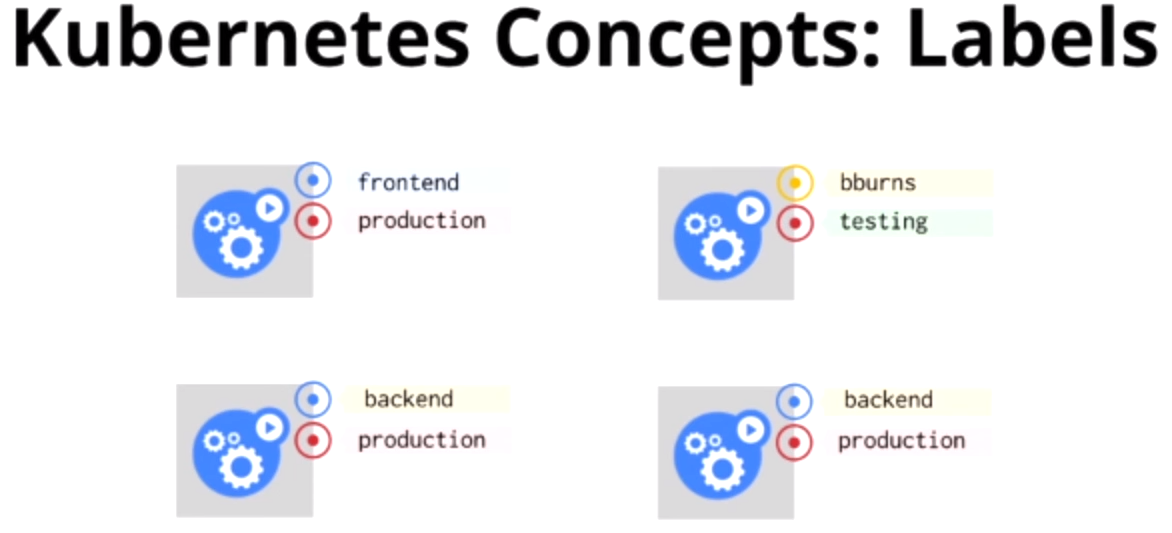
Pods are atomic units of Kubernetes cluster.

* Collection of one or more containers that work together
* Central data volumes (independent of the life of the container)
* Namespaces that these containers shared within a pod
* Labels, key-value pairs



Pod

* is the atomic unit of scheduling
  + does not make sense to keep these containers on different hosts, even though it makes perfect sense to keep them in different containers
    - different container argument validate by the need for separation of concerns, manageability, QoS, decoupling, etc.
* volumes and network namespace are shared
* Different containers in a pod can see each other on the host, as they share the same name-space
* They share other kind of resources, e.g. memory
* When you scale pods,
  + each pod gets a different name/IP address
  + Different instances of a pod can talk to each other using the name/generated IP address
  + these IP addresses are unstable (keeps changing)
  + Hence the concept of Kubernetes Service



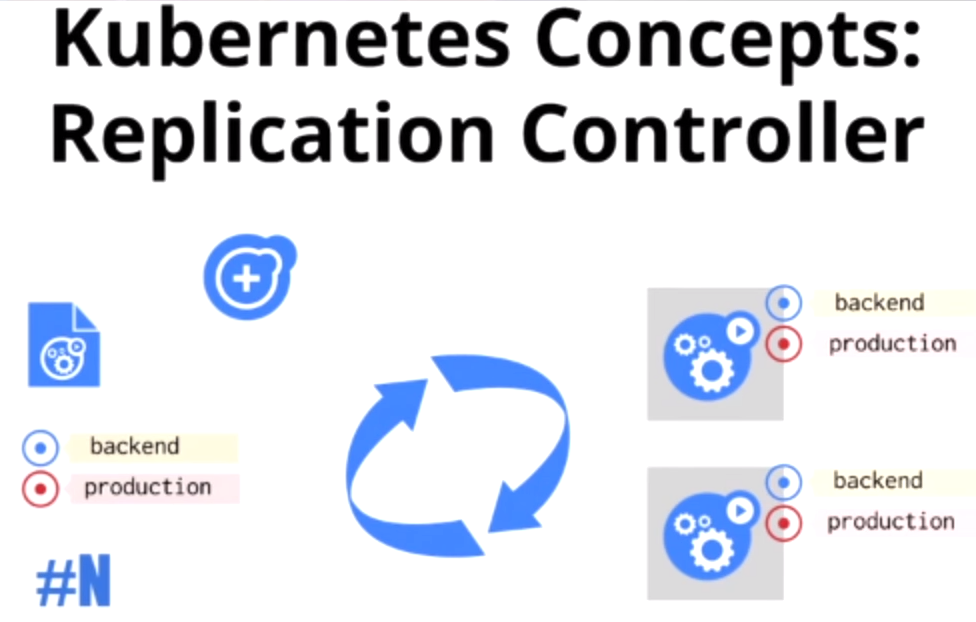
Labels are key-value pairs which can be queried in a very powerful, flexible way.

**Reconciliation**

You have 2 things

* Desired state
* Actual state

Reconciliation loops job is to find the difference and then take action to move the current state towards kdesired state.



Replication Controller is a combination of

* A desired state template
* Some labels
  + Query and get the current state
* And a number
  + To define what the state is



Decoupling of application from infrastructure – we have lost the ability to identify machines. We have to regain the ability to identify

* Services can be seen as a load balancer which has a stable IP address
* Services get a known static fixed IP address
* Service has a name, a port and labels
* Can export the IP address as a DNS name
* The IP address is actually fake – is not entered in machines
  + These IP addresses are not very stable, come and go as you scale out and in.
* Load balancer between every single layer of the application
  + Microservices
  + Facilitates a team growth
* The ability to get a DNS name for a service is important for Service Discovery

Service Discovery