



Introduction to Kubernetes

and why you should care

by Paris Apostolopoulos (aka javapapo)

About me

I come from Athens Greece, recently moved to Luxembourg.

I love Java, Judo and Apple devices :P

Java EE enthusiast

Java Champion (2007) & JBoss Hero

co-Founder and admin of the first Java User Group in Greece - www.jhug.gr

I hate technical debt and teams that procrastinate



Goals

This is a soft introduction

Targeting mostly developers that now are entering this new world of containers, microservices etc.

Save you some time from endless research and experimentation.

Introduce basic concepts of kubernetes towards system design and architecture.

Validate why kubernetes is the right framework for the job - at least make you consider it!

A lot of words
a lot of words
we are really
at this

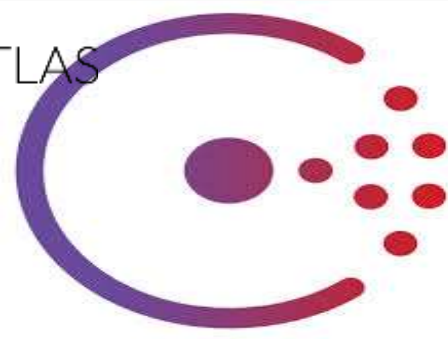


ATLAS

Microservices

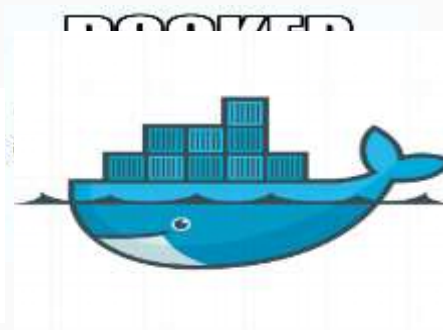
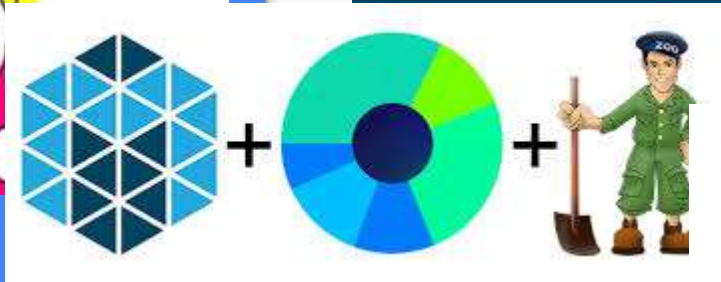


Rocket



EXPERIMENT

OPENSIFT



Some remarks

An new exciting world of technologies.

Containers, microservices and the 'cloud way' are taking over.

There is a demand, sometimes not clearly justified, to move existing and new apps into this new world.

Devops is taking over but a gap is created at the same time, ops vs software developers.

Microservices Containers Cloud

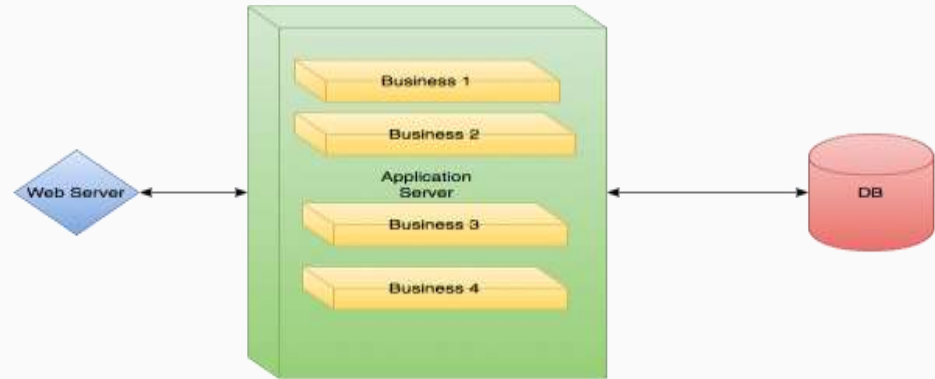
Cloud : your private or a public one, a pool of resources (machines), waiting to execute your code.

Containers : another way of packaging - containing applications (services) and OS dependencies . They are expected to run on top your 'cloud' machines.

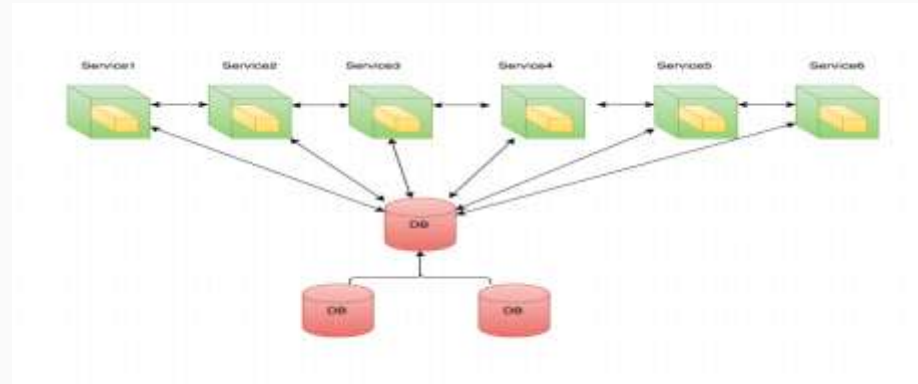
Microservices : A new architectural trend on designing and implementing apps. Separation of concerns, decoupling, modular.

How?

How do we move from this

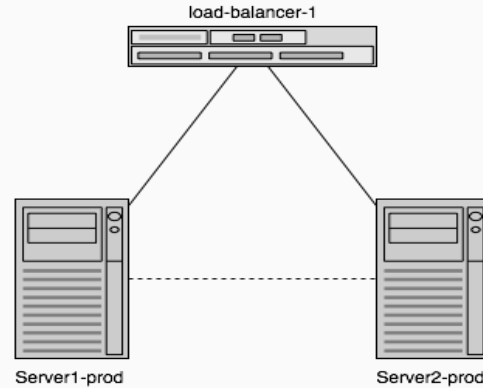


To this

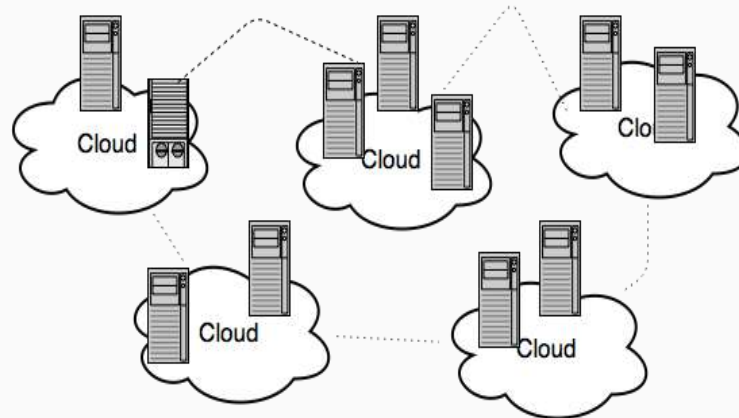


How?

How do we move from this



To this



How?

How do we design applications that can not make any assumptions about their runtime environment? (filesystem/network)

How do we design applications that are consisted of several moving parts, but can still be deployed and orchestrated easily?

How do we design applications can move from one cloud platform to another ?

How can we reuse our existing skills and techniques on application design and architecture without deep diving to other areas?

Common misconceptions



Docker-izing your application does not mean that you do microservices.

2 processes (e.g wars) that exchange JSON are not a microservices platform.



Technical split of your good old monolith, does not mean that you actually moved to a microservices architecture.

Common misconceptions

A lot of technologies / frameworks claim to be 'the operating system' of the cloud.

The reality is that certain tools cover specific areas of the overall problem.

Platforms that abstract the real hardware layer (your good old server cluster).

Platforms that can 'schedule' and orchestrate specific work on a cluster (aka schedulers)

Platforms with mix concerns that promise to act as a 'platform as a service', where you will design, host and operate your own service.

The reality

You need to invest money and skills on separate tools and technologies.

The terms [platform](#) and [infrastructure](#) as a service are sometimes interpreted in a very obscure sense.

You find yourself deep diving on ops and experimenting with OS level services rather than your application

Currently most of the technologies are on their infancy, ever changing and competing.

The question

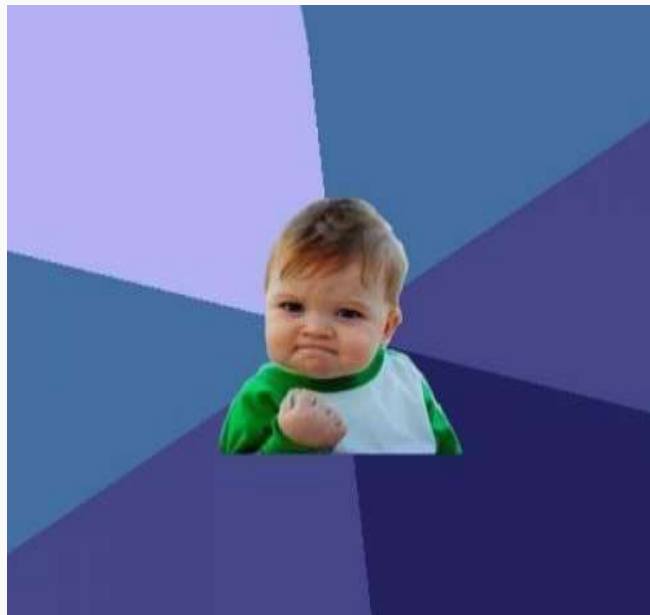


Is there any technology that will abstract most of the complexity imposed by many infrastructure technologies?

Is there any technology that will enable me to think again about application (parts) services and their layout- rather than the specifics of the underlying infrastructure?

Is there any technology that can be easy and powerful enough so that it can be 'setup' by the average Joe Developer in order to be evaluated and then maybe use it actually?

The answer



Yes! Entering Kubernetes

Kubernetes κυβερνήτης

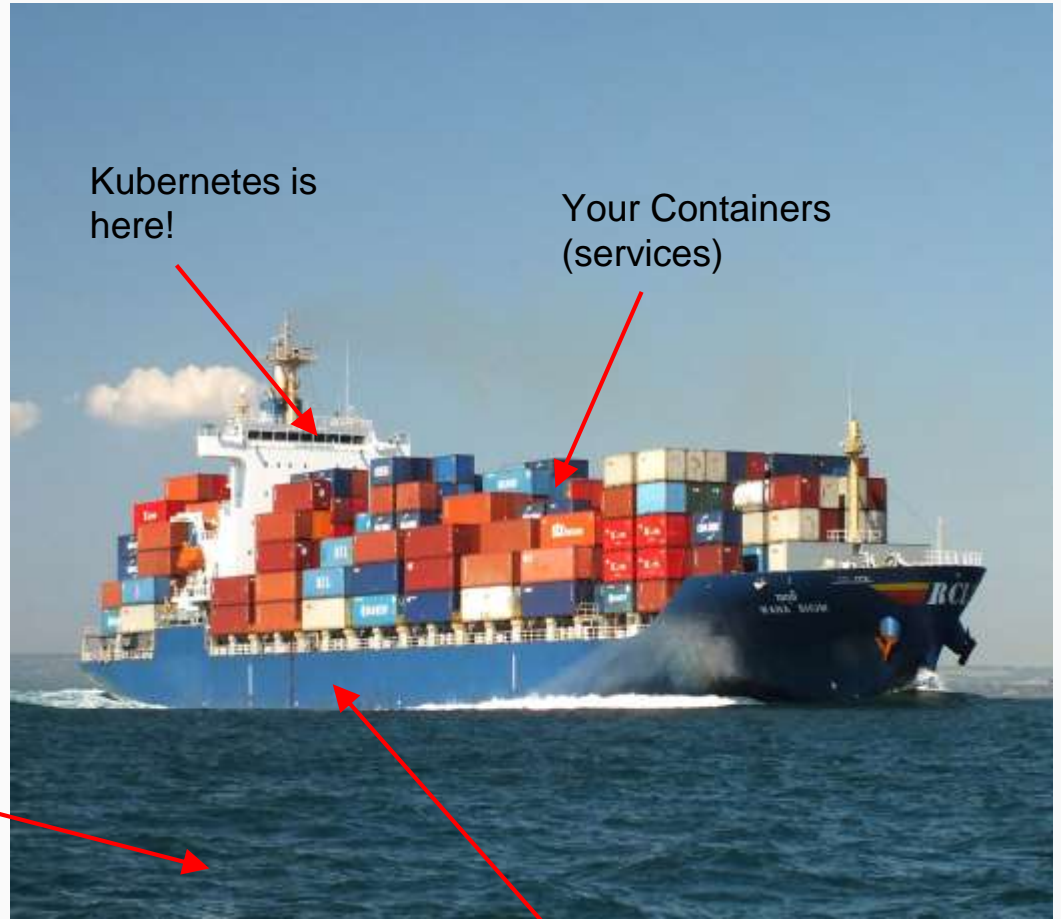
the person controlling a ship (captain)
the person that controls / governs a city

Abstract View (very abstract)

The internet

Kubernetes is
here!

Your Containers
(services)



You Cloud
infrastructure

Kubernetes History

Google project started in 2014

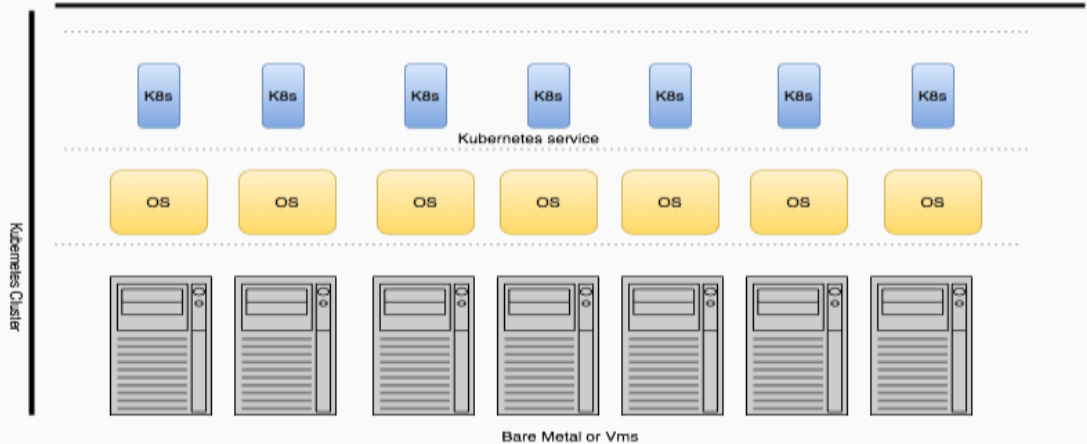
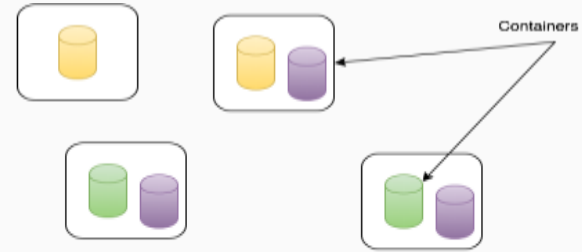
10 year old experience derived from Google internal container platform code name [‘Borg’](#)

Project reached it's 1.0 major release on July 2015.

Written in [Go lang](#).

Heavily relies on [Etcd](#)

A less abstract view



So in plain words either you have a cluster of VMs or a cluster of physical machines, that are all Kubernetes aware, then they form a Kubernetes cluster. Kubernetes then is the 'tool' so that you make use of the underlying docker engine installed in each host (node) . (simplified description).

What is it?

“is an open source system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications”

Is not a [Infrastructure as a Service framework](#), it abstracts the data center resources on a high level.

Is not a [Platform as a Service](#) framework, it can be use as the basis so you can come up with one.

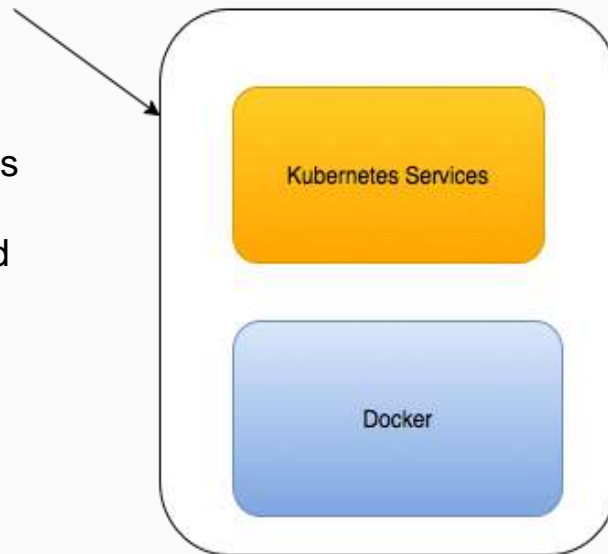
It enables you to deploy, manage and run on top of abstracted pool of resources, your dockerized applications.

It enables you to ‘design’ your **application topology** and spread and interconnect your services with minimal or changes to the actual service

Anatomy of a Kubernetes Node

Each node basically runs
a number of services
(kubernetes related) and
a docker engine

Vm or Physical Machine



We have 2 basic types of nodes

- master
- worker

The Master Node

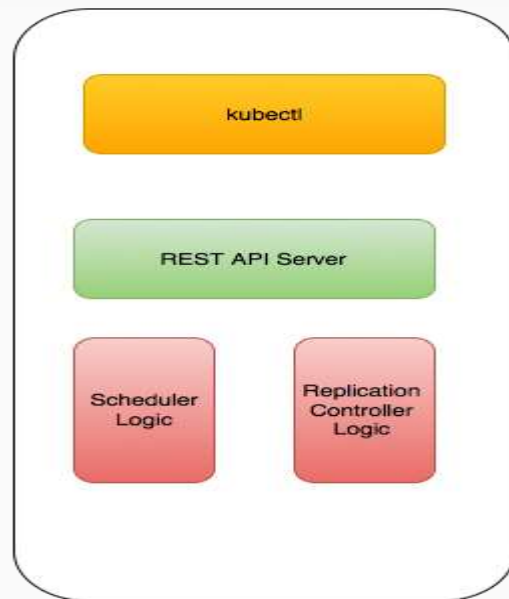
The controller - controls and manages the cluster

Run services of kubernetes

[kubectl](#) : command line client

rest api: rest api for communicating with the workers

[Scheduling](#) and Replication logic



The Worker Node

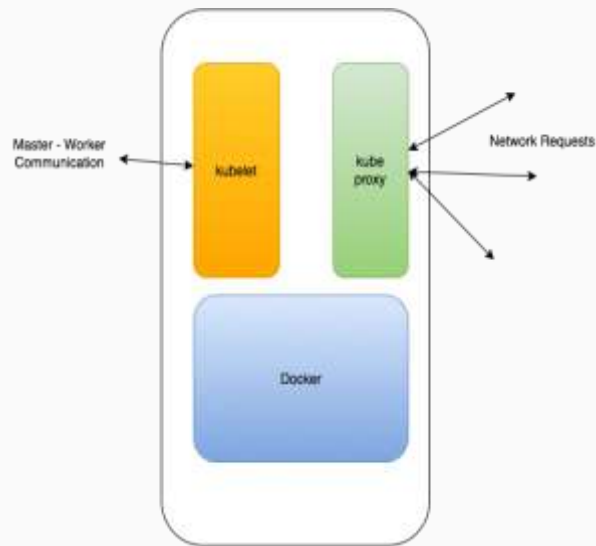
The node that eventually host your services - containers

Run services of kubernetes

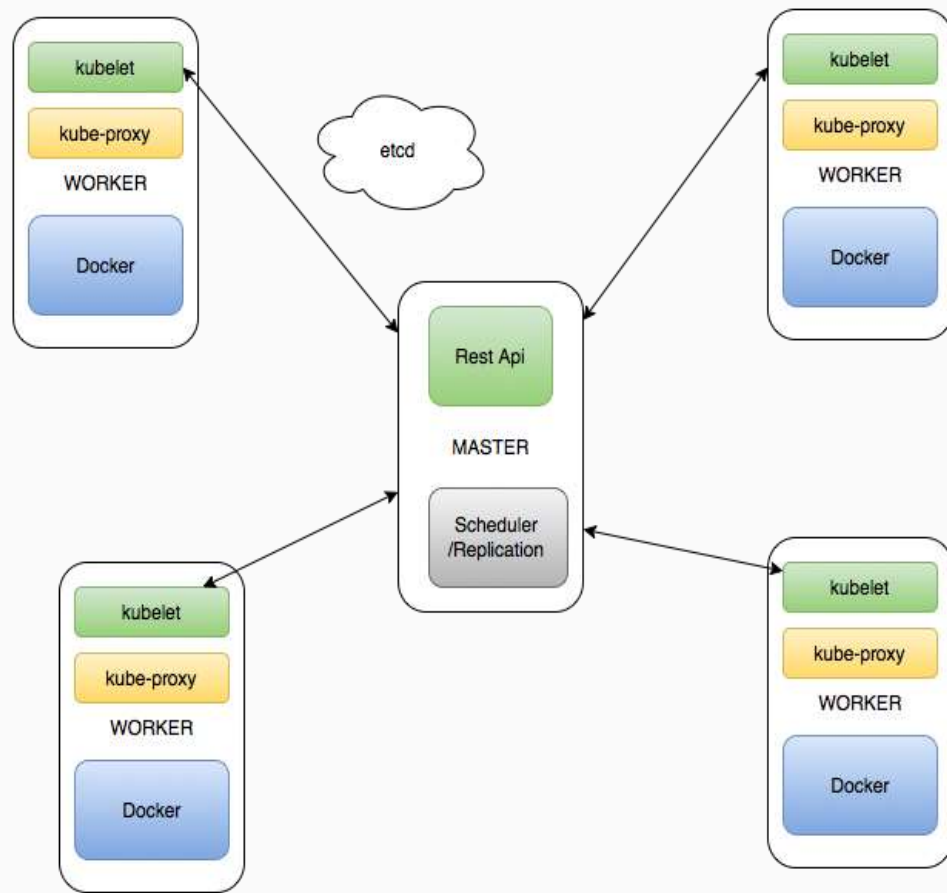
[kubelet](#) : kubernetes agent (accepts commands from master)

[kubeproxy](#): network proxy service on a node level

docker host



All together -
simplified



pod

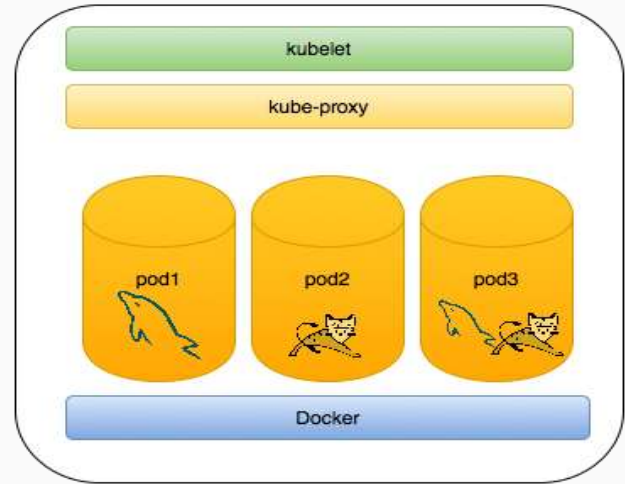
Within the worker node, services will be 'contained' in a [pod](#).

A pod can contain **more than one** services (aka docker containers)

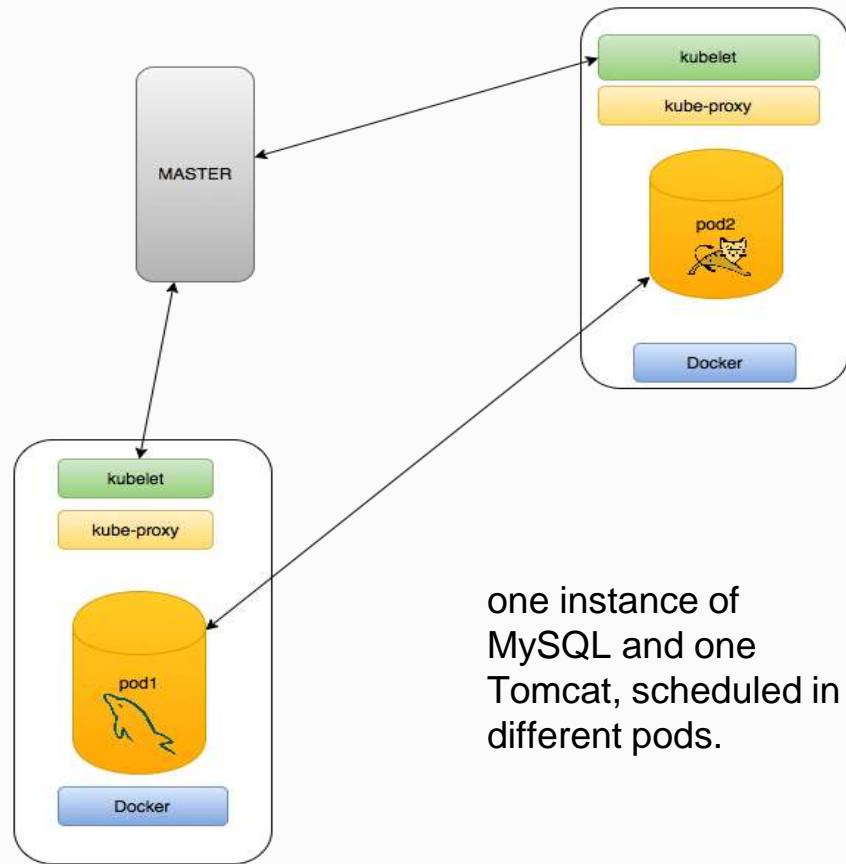
each pod has it's own IP.

it is a logical host .

master schedules, replicates & creates
pod's not containers!



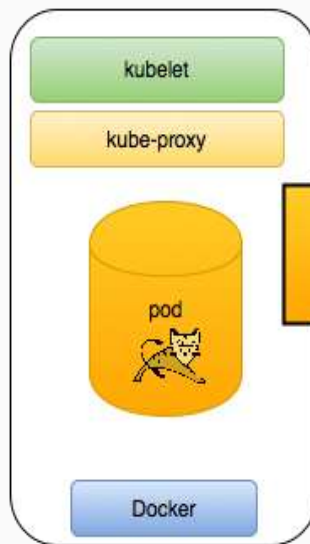
All together pods



one instance of
MySQL and one
Tomcat, scheduled in
different pods.

Labels

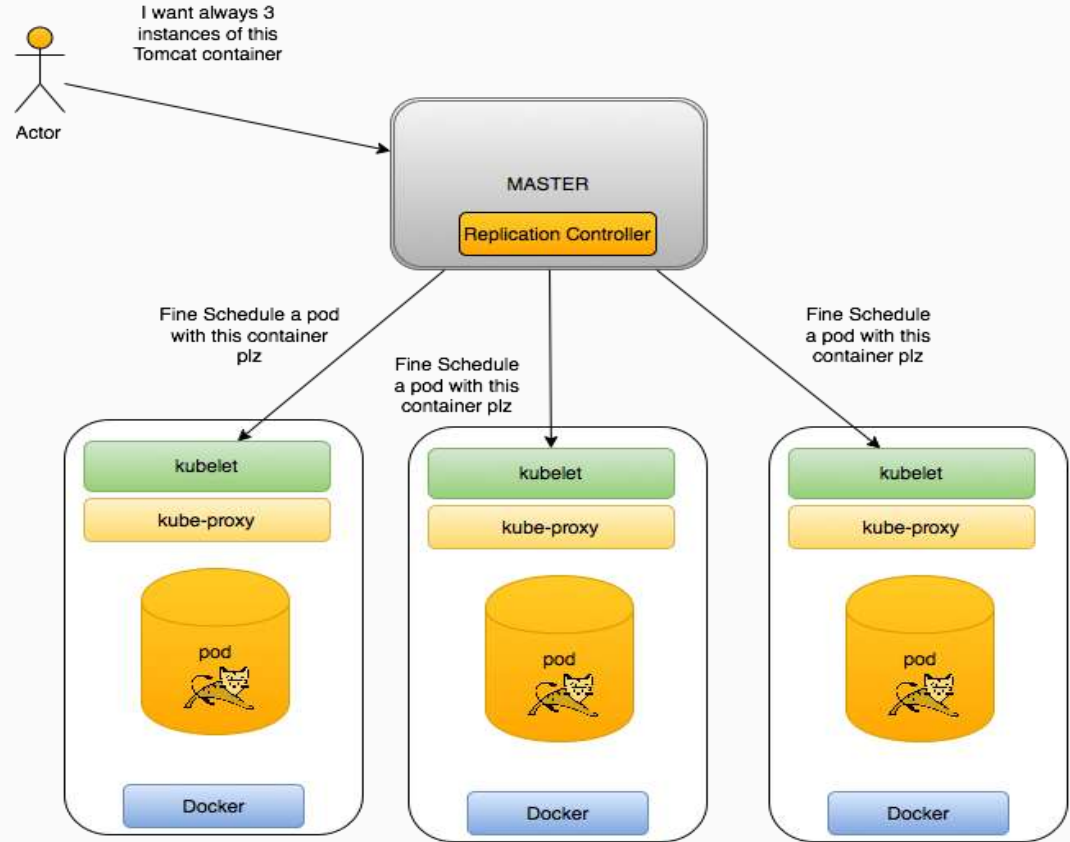
- Labels are metadata we can assign to kubernetes resources - such as pods, services when we create them (spin them).
- They are simple key value pairs.
- Labels are crucial to kubernetes since a lot of core kubernetes functionality relies on 'querying' the cluster for 'resources' that have certain label assigned! (Selectors)



```
"labels": {  
  "appName": "testApp",  
  "container": "tomcat"  
}
```

we can query the kubernetes cluster for all the pods that belong to an application named 'testApp'

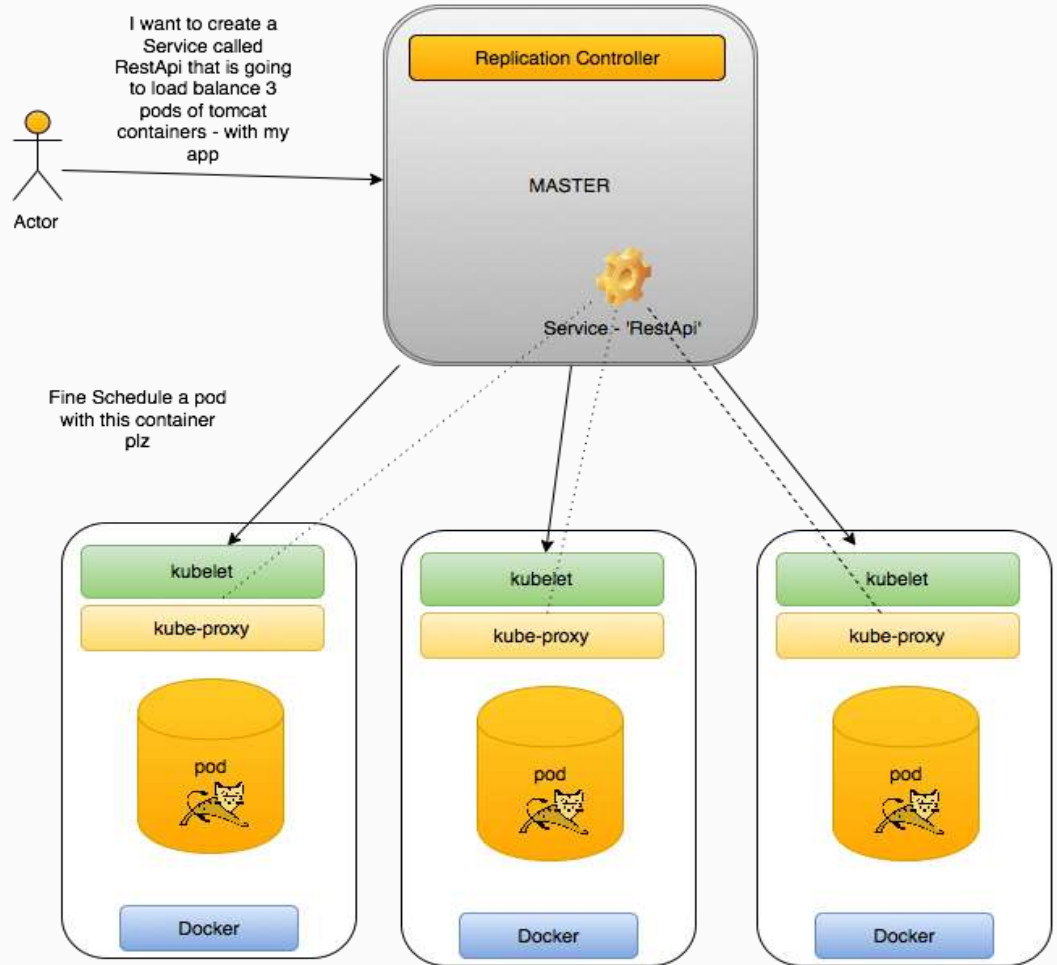
What if I want 3 instances of tomcat running?



The replication Controller

- Is responsible to maintain as many pods as they have been requested by the user.
- It will start or kill pods depending on the replication limit provided.
 - E.g if we ask for 3 pods running a tomcat docker instance with a specific description, then it will kill any attempt to spin a 4th one.
- The replication controller uses a **'template'** which is just a simple descriptor that describes exactly what each pod should contain.
- We can dynamically call the replication controller of the kubernetes cluster in order to scale up or down a specific pod.
 - E.g I want to scale up these 3 tomcats and to make them 6.

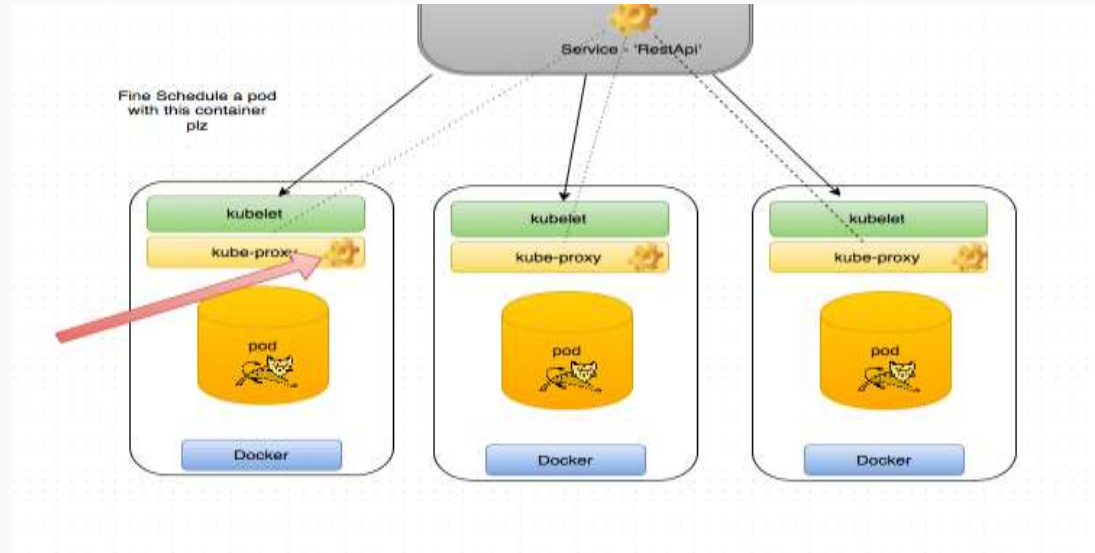
How I can group these 3 pods and load balance them?



The Service

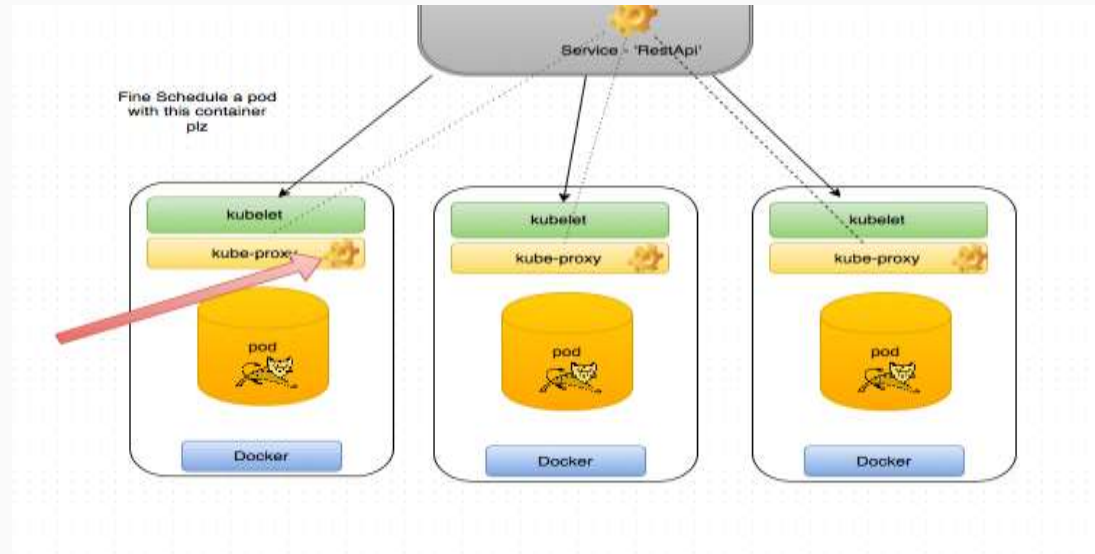
- A [service](#) is not - a running thing - there is no real single load balancer.
- It is information stored in the kubernetes cluster state and networking information propagated to all the nodes.
- It groups a set of pods, providing a single point of access.
- For example if we want to access the 'RestApi' service we no longer need to know each pod's ip address.
- By grouping similar pods into service(s) we eventually solve the discoverability and connectivity between our containers.
- A Service with 3 pods of MySQL and another Service with 3 pods of Wildfly can 'talk' to each by this single 'domain' name - cluster internal IP and on predefined (if we wish ports).

The Service- env variables?



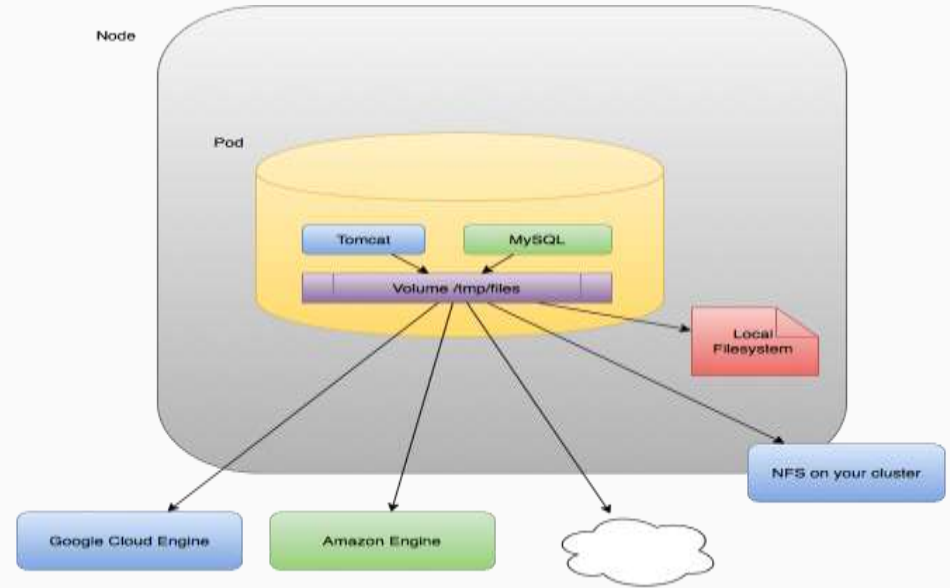
- Once a service is declared then related [Environment Variables](#) are accessible on the node.
- TOMCAT_SERVICE_HOST=10.0.0.11
TOMCAT_SERVICE_HOST_PORT=80
TOMCAT_PORT=tcp://10.0.0.11:6379

The Service- How?



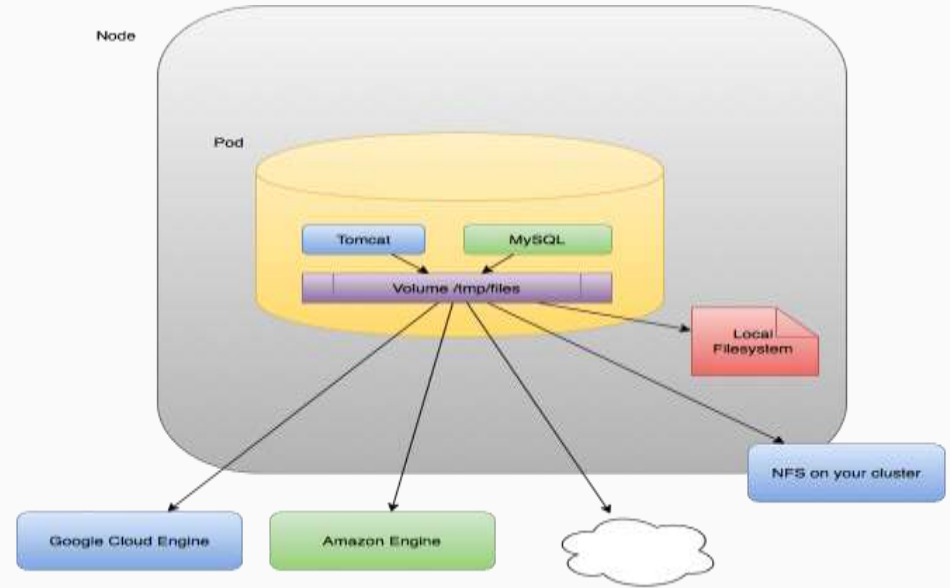
- Every time we create a service, all the nodes of the cluster - get aware of it.
- The kube-proxy service running on all workers allocates a specific port.
- IP-Tables are being populated so that traffic can be redirected to the related services
- A service is eventually a cluster wide 'information' that each node is aware.

Volumes



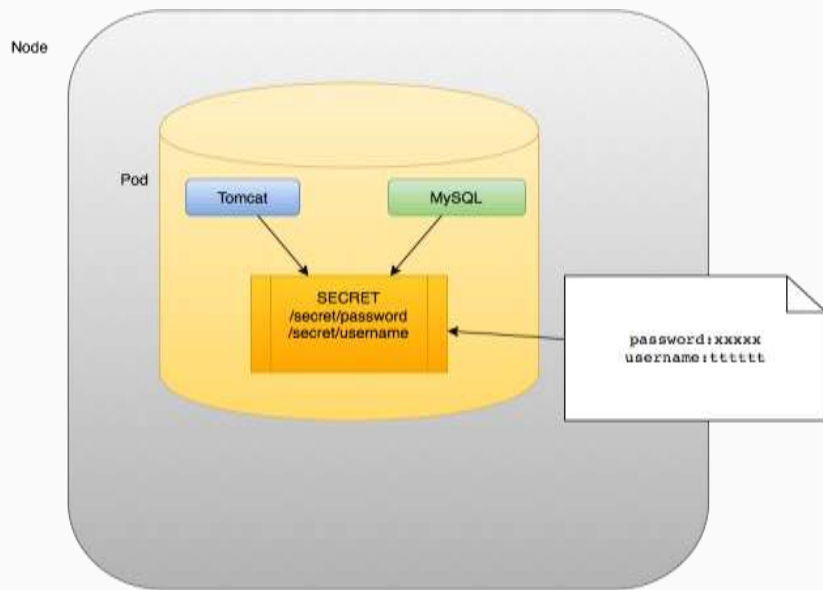
- Another powerful abstraction
- File access for your services.
- It lives as the pod lives!
- A container can die in the pod, the volume still lives.
- The volume can point to the Node (host), pod only or to an external file /storage provider.

Volumes



- The most important thing is to foresee the need of the volume in your topology.
- Once you have selected the type and you have integrated it on your app design, you won't have to worry about, no matter how your pods or containers run or scale.

Secrets



- Sensitive information, that containers need to read or consume.
- Secrets are another kubernetes abstraction
- Technically they end up as special volumes mounted automatically so that your containers can read their contents.
- Each entry has its own path.

So how I Install kubernetes - for the first time?

The easiest way is a local cluster - with vagrant managed vms. But there are several other ways, as documented [here](#).

1. Install [Vagrant](#) on your machine, make sure it works
2. Download the latest release of kubernetes from here ([releases](#)) e.g 1.0.6

3. Unzip the folder to a path of your choice e.g
~\kubernetes_home\

1. Set the following env variables on your environment or just export them beforehand.

```
export KUBERNETES_PROVIDER=vagrant
export VAGRANT_DEFAULT_PROVIDER=virtualbox
export NUM_MINIONS=3
export KUBERNETES_MASTER_MEMORY=1536
export KUBERNETES_MINION_MEMORY=1024
```

1. Execute the following script
~\kubernetes_home\cluster\kube-up.sh

1. Wait a bit (depending on the number of nodes) to complete - you are done!!!

Number of
Nodes



```
Kubernetes cluster is running. The master is running at:
https://10.245.1.2

The user name and password to use is located in ~/.kubernetes_vagrant_auth.

... calling validate-cluster
Found 3 nodes.
  NAME      LABELS                                STATUS
  1  10.245.1.3  kubernetes.io/hostname=10.245.1.3  Ready
  2  10.245.1.4  kubernetes.io/hostname=10.245.1.4  Ready
  3  10.245.1.5  kubernetes.io/hostname=10.245.1.5  Ready

Validate output:
NAME                STATUS  MESSAGE  ERROR
controller-manager  Healthy ok        nil
scheduler            Healthy ok        nil
etcd-0               Healthy {"health": "true"} nil
Cluster validation succeeded
Done, listing cluster services:

Kubernetes master is running at https://10.245.1.2
KubeDNS is running at https://10.245.1.2/api/v1/proxy/namespaces/kube-system/services/kube-dns
KubeUI is running at https://10.245.1.2/api/v1/proxy/namespaces/kube-system/services/kube-ui
```

So how do I issue commands to my cluster?

Kubectl is the command line tool to 'talk' to your cluster master.

- Kubectl is already included in the download archive

```
export KUBECTL=~\kubernetes_home\platforms\xxx\amd64
```

- add KUBECTL to your \$PATH, or always navigate to the above path.

Some kubectl commands

```
#Get the state of your cluster
```

```
$ kubectl cluster-info
```

```
#Get all the nodes of your cluster
```

```
$ kubectl get nodes -o wide
```

```
#Get info about the pods of your cluster
```

```
$ kubectl get pods -o wide
```

```
#Get info about the replication controllers of your cluster
```

```
$ kubectl get rc -o wide
```

```
#Get info about the services of your cluster
```

```
$ kubectl get services
```

```
#Get full config info about a Service
```

```
$ kubectl get service <NAME_OF_SERVICE> -o json
```

```
#Get the IP of a Pod
```

```
$ kubectl get pod <NAME_OF_POD> -template={{.status.podIP}}
```

```
#Delete a Pod
```

```
$ kubectl delete pod NAME
```

```
#Delete a Service
```

```
$ kubectl delete service NAME_OF_THE_SERVICE
```

Schedule a simple pod

```
# official documentation here
```

```
# Schedule to start a pod that will contain the latest
```

```
# Wildfly Image.
```

```
$ kubectl run myjboss --image=jboss/wildfly --port=8080
```

```
# Check our created pod(s).
```


```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS
AGE			
myjboss-jvqob	1/1	Running	0
3m			

Schedule a simple pod - with template

```
{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "anotherJboss",
    "labels": {
      "app": "myapp"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "backend-core-jboss",
        "image": "jboss/wildfly",
        "ports": [
          {
            "containerPort": 8080,
            "protocol": "TCP"
          }
        ]
      }
    ]
  }
}
```

Save it as
single-pod.json




```
$ kubectl create -f ./single-pod.json
```


Schedule a simple service - with template

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "test-service"
  },
  "spec": {
    "selector": {
      "app": "myapp"
    },
    "ports": [
      {
        "protocol": "TCP",
        "port": 80,
        "targetPort": 8080
      }
    ]
  }
}
```

Save it as
single-
service.json

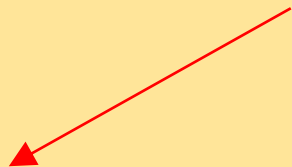


```
$ kubectl create -f ./single-service.json
```

Schedule a replication controller - with template

```
{
  "kind": "ReplicationController",
  "apiVersion": "v1",
  "metadata": {
    "name": "jboss-controller"
  },
  "spec": {
    "replicas": 2,
    "selector": {
      "app": "backend"
    },
    "template": {
      "metadata": {
        "labels": {
          "app": "backend"
        }
      },
      "spec": {
        "volumes": null,
        "containers": [
          {
            "name": "jboss",
            "image": "jboss/wildfly",
            "ports": [
              {
                "containerPort": 8080,
                "protocol": "TCP"
              }
            ]
          }
        ],
        "imagePullPolicy": "IfNotPresent"
      }
    },
    "restartPolicy": "Always",
    "dnsPolicy": "ClusterFirst"
  }
}
```

Save it as
single-rc.json



```
$ kubectl create -f ./single-rc.json
```

Who is embracing Kubernetes?

- [Google Container Engine](#)
 - You can spin a cluster of VMs that run Kubernetes in a matter of minutes
- [OpenShift version 3](#)
 - The new P.a.a.S of RedHat is based on Kubernetes!
 - Available as a service and as private cloud installation.
- [Tectonic](#) - by [CoreOS](#)
 - a new platform
 - CoreOS+Kubernetes
- [Fabric 8](#)
 - O.S platform based on Kubernetes
 - + extra services
- OpenStack Support
 - [Murano](#)
- Apache Mesos
 - [Kubernetes](#)

Available documentation and resources?

Documentation

- Official Documentation [here](#)
- [Code](#)
- Google Container Engine documentation [here](#)
- [Google Group](#) - for questions.
- [kubernetes.slack.com](#)
- [StackOverflow #kubernetes](#)

Books

- [Kubernetes Up and Running](#) O'Reilly (K.H)
- [Scheduling the future at cloud scale](#) - OpenShift
- [Kubernetes-book \(upcoming\)](#)

Articles

- [Kubernetes Design Patterns](#) (Arun Gupta)
- [Kubernetes Key Concepts](#) (Arun Gupta)
- [Recipes for deploying JavaEE Apps](#) (Arun. Gupta)
- [Intro to Kubernetes](#)
- [Kubernetes for Developers](#)
- [The new PAAS](#)

Video

- [Tech overview of Kubernetes](#)

- Container Orchestration using CoreOS and Kubernetes ([1](#), [2](#), [3](#))

Twitter accounts
related with
kubernetes - check
them out!

- #kubernetes (twitter)
- @[kubernetesio](#)

Devs

- @[ibeda](#)
- @[thockin](#)
- @[brendandburns](#)
- @[TallMartin](#)
- @[kelseyhightower](#)
- @[kleban](#)
- @[asynchio](#)
- @[preillyme](#)
- @[KitMerker](#)
- @[jml3on](#)

Other

- @[tectonicStack](#)
- @[thenewstack](#)
- @[kubeweekly](#)
- @[kismatic](#)
- @[googlecloud](#)

Well it was just a start..

This was just a soft introduction, but you need to understand the basic ideas and then project them to your current or future project.

Don't rush into every technology there is a chance you lose your focus and forget about the real problem, which is delivering your app.

The power of kubernetes is that does not distract you 100% from your application design and topology.

It does try not to become yet another technical milestone in order for you to reach your end goal - which is eventually to deploy a scalable and easily maintained application.

Thanks for
your time!

Contact :

@javapapo

javapapo@mac.com

javapapo.blogspot.com

lu.linkedin.com/in/javaneze



**KEEP
CALM
AND
TRY
KUBERNETES**