

# Kubernetes

for

# Developers and Operators



# We are here to help.

Any time during the training:

Not helpful.

You want to know how a specific feature works.

Too confusing.

Too technical.

Not technical enough.

You think we are wrong.

**Please let us know!**

# Meet the Coreos in the Room!

Tony

John

Matt

# Logistics

Start Time: **9am**

End Time: **4pm - 5pm**

Breaks & Lunch

Restrooms

Slack Back Channel

presented by



#### OPEN SOURCE

90+ Projects on GitHub, 1,000+ Contributors



[coreos.com](http://coreos.com)

#### ENTERPRISE

Support plans, training and more



[sales@coreos.com](mailto:sales@coreos.com)



# We are active participants in the open source community

At CoreOS, open source communities are part of our DNA.

We are **BUILDERS** and **LEADERS** in the open source community, contributing projects, code, and standards

- 35+ engineers contributing to upstream projects and leading 5 special interest groups (SIGs)
- Created and built several critical projects, including: etcd, rkt, clair, flannel, bootkube, minikube, and lead of Prometheus project for monitoring
- Active in roadmap contribution across all projects

*...we are producers, not just packagers*



**kubernetes**

---

CONTAINER ORCHESTRATION



CONSENSUS & DISCOVERY



VULNERABILITY ANALYSIS



CONTAINER RUNTIME



NETWORKING



# container linux

- Linux OS optimized for clusters of containerized applications
- Automatic, secure, reliable updates



- Reliable, distributed key-value store for critical data
- Solving the hardest problem in infrastructure today:  
Distributed consensus
- More than 500+ projects on Github using etcd
- Cloud Foundry, Kubernetes and many other projects use  
etcd



- Container runtime
- Designed with security and composability in mind
- Minimal architecture, native ability to work with init systems



- Open source container security analyzer
- Flags vulnerabilities in packaged (RPM/DEB) software in containers



Hosted in the cloud and scales with you.  
Priced by the number of private repositories.



Runs privately anywhere you can run a Docker container.  
Fixed price for unlimited users and repositories

- Container registry service
- Along with storing containers, Quay can also:
  - Build
  - Distribute
  - Perform security analysis using Clair
- Configurable triggers, advanced team permissions, and secure storage
- Quay.io is hosted; Quay Enterprise is self-hosted



# TECTONIC

- Installs and bootstraps Kubernetes clusters
- Enterprise management console on top of Kubernetes
- Manages self-driving Kubernetes
- Supports cloud and on-premise Kubernetes installations



## Representing you in the open source community

*CoreOS helps establish open standards within the community*



### kubernetes

*We pull the most stable branch, and then test & certify*



*We work w/ you to identify new feature requests & security issues*



*With self driving, CoreOS automatically updates your cluster*

# Lab Exercises

<http://sfo829.coreosstrain.me>

# Exercise

Connect to the Lab

# The Introduction App!

*Tell us who you are by writing a simple web application and deploying it to...*



# Kubernetes!

# Exercise

Intro App



# Let's Begin With...

# World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#) , [Policy](#) , November's [W3 news](#) , [Frequently Asked Questions](#) .

## What's out there?

Pointers to the world's online information, [subjects](#) , [W3 servers](#), etc.

## Help

on the browser you are using

## Software Products

A list of W3 project components and their current state. (e.g. [Line Mode](#) ,[X11 Viola](#) ,[NeXTStep](#) ,[Servers](#) ,[Tools](#) ,[Mail robot](#) ,[Library](#) )

## Technical

Details of protocols, formats, program internals etc

## Bibliography

Paper documentation on W3 and references.

## People

A list of some people involved in the project.

## History

A summary of the history of the project.

## How can I help ?

If you would like to support the web..

## Getting code

Getting the code by [anonymous FTP](#) ,etc.

**info.cern.ch**

August 6, 1991

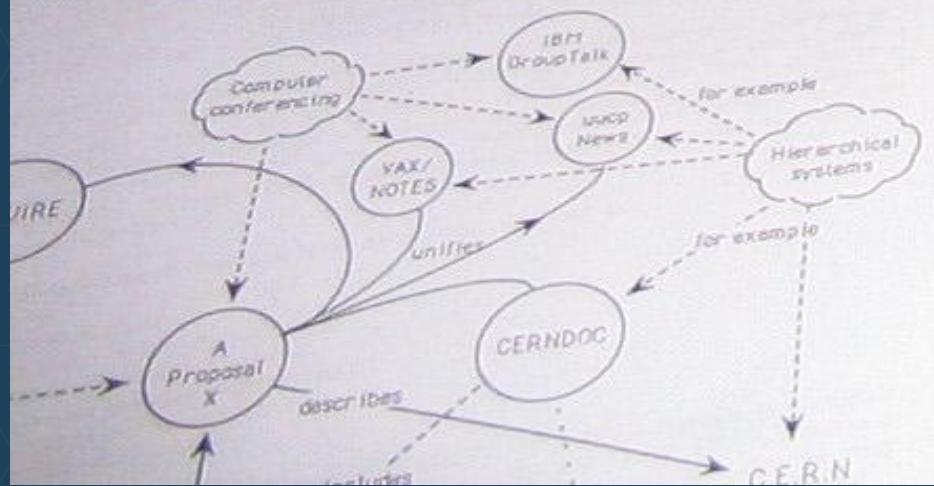


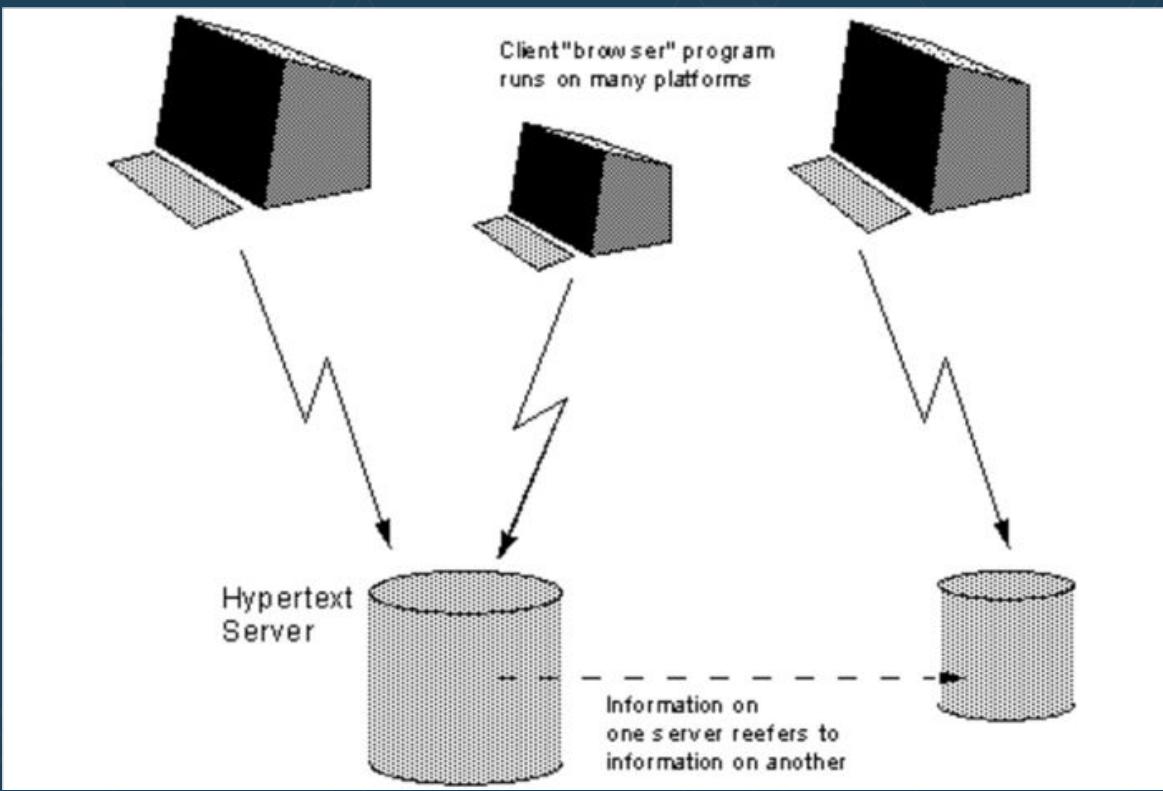
## Information Management: A Proposal

### Abstract

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the possibility of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.

Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project management, CERN, CERNDOC, YAX/NOTES, IBM GroupTalk, UUCP News, Hierarchical systems.





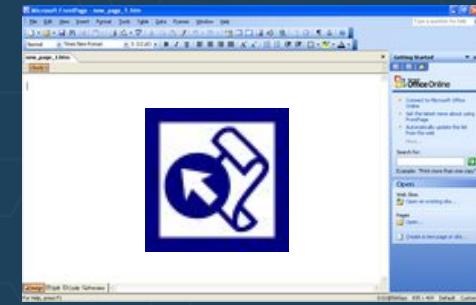
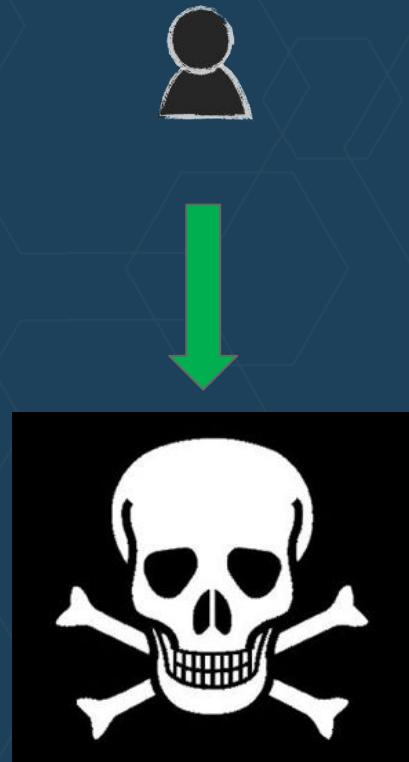
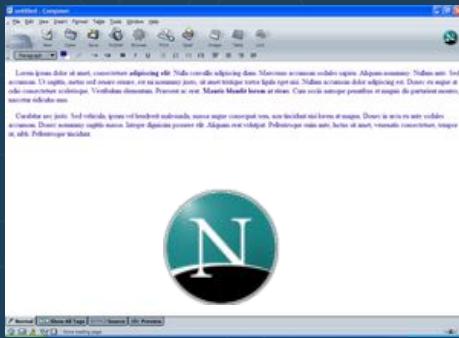
A "universal linked information system" in which a network of documents linked to one another in a simple way that anyone could navigate to find exactly what they need.

# Which led to...



# Which led to...

# “Web” Applications...



*Everything fails...all the time.*

-Werner Vogels (CTO & VP Amazon)

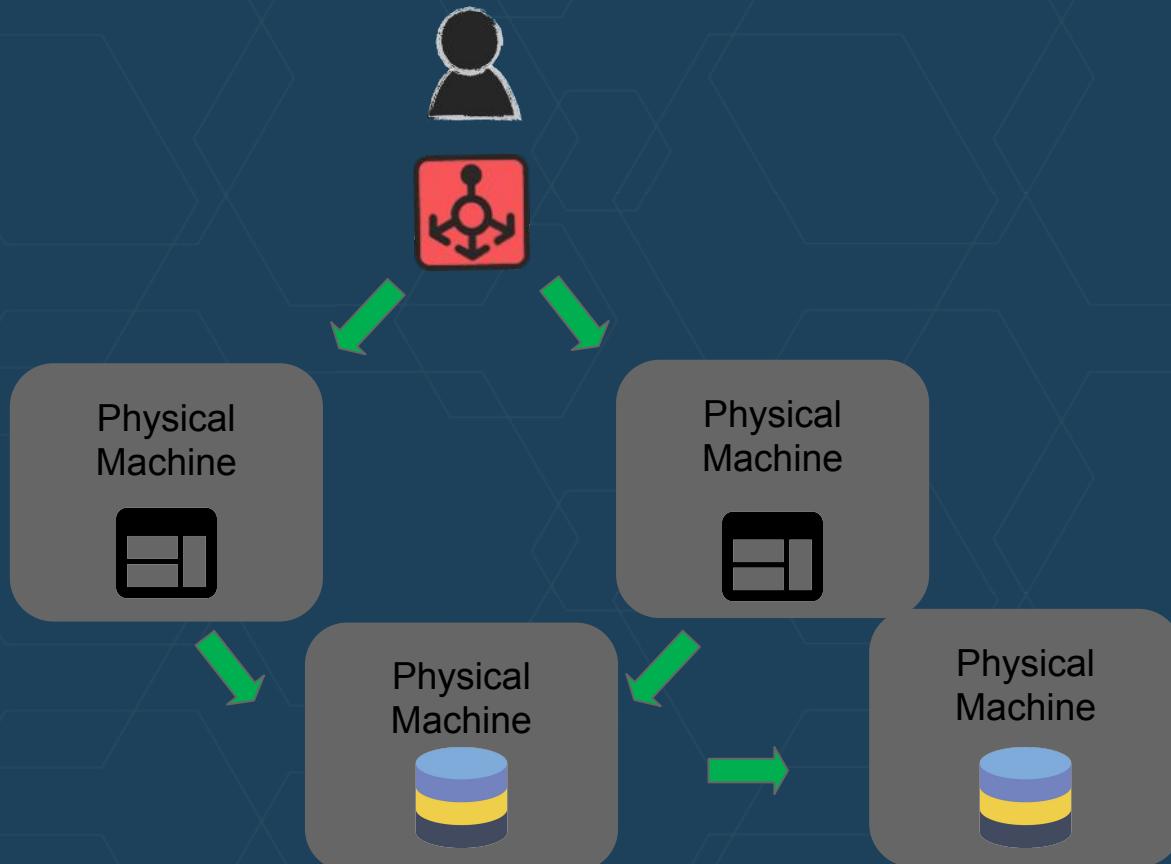
*Hope is not a strategy.*

-Google Site Reliability Engineer Motto

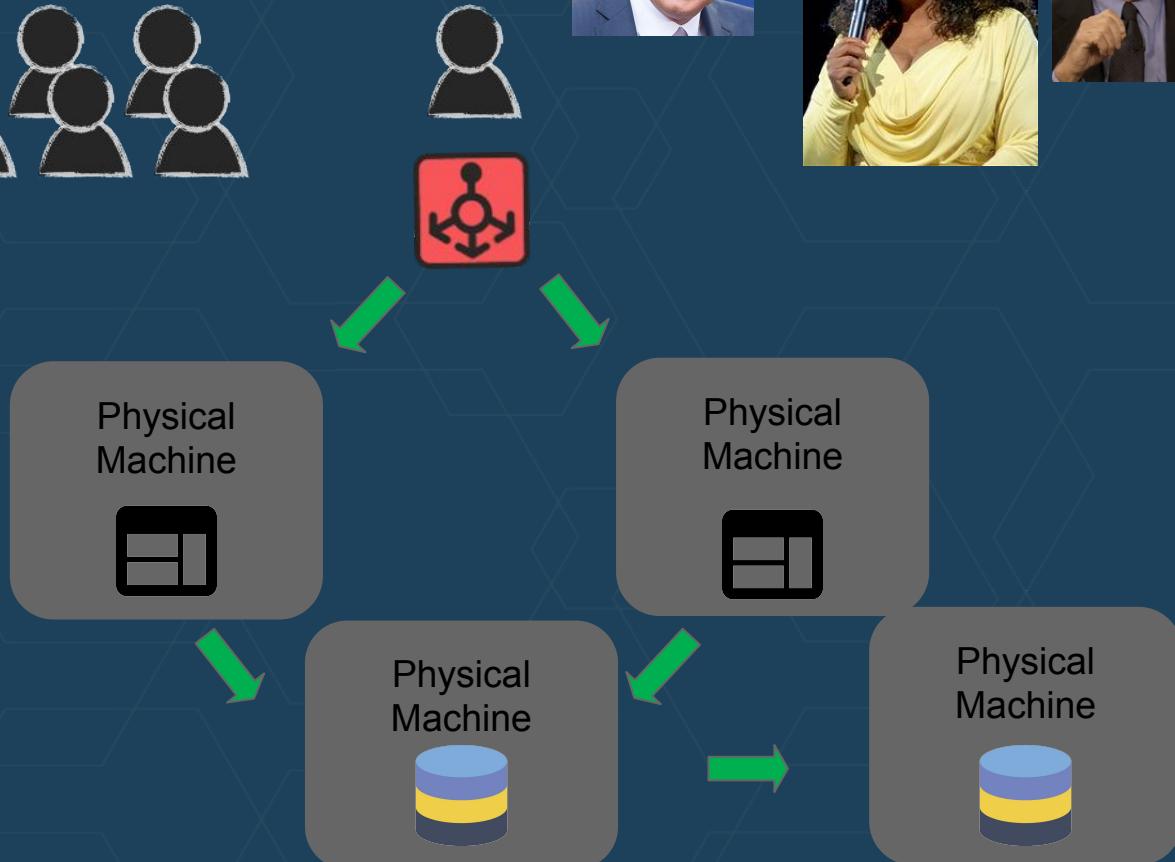
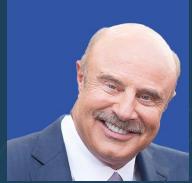
# Web Applications...



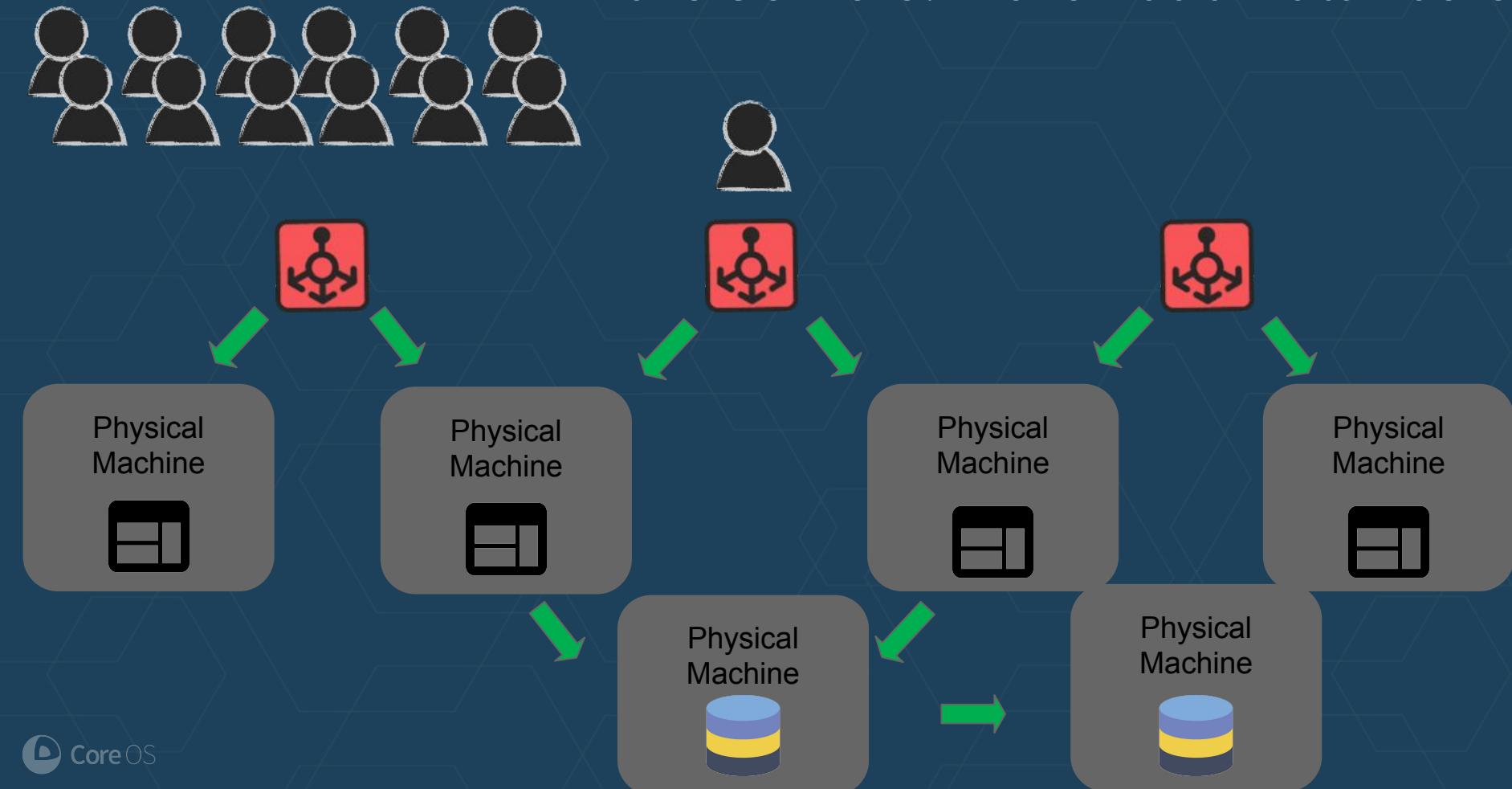
# Web Applications...



# Things are great until....

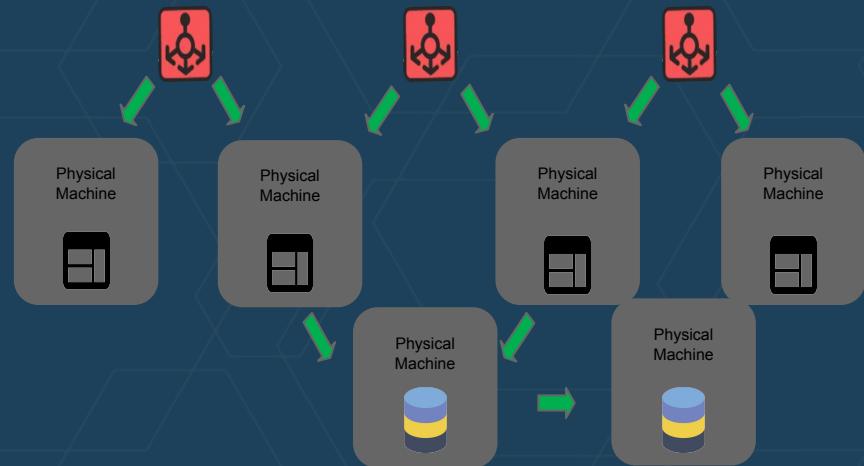


# More Servers! More Load Balancers!

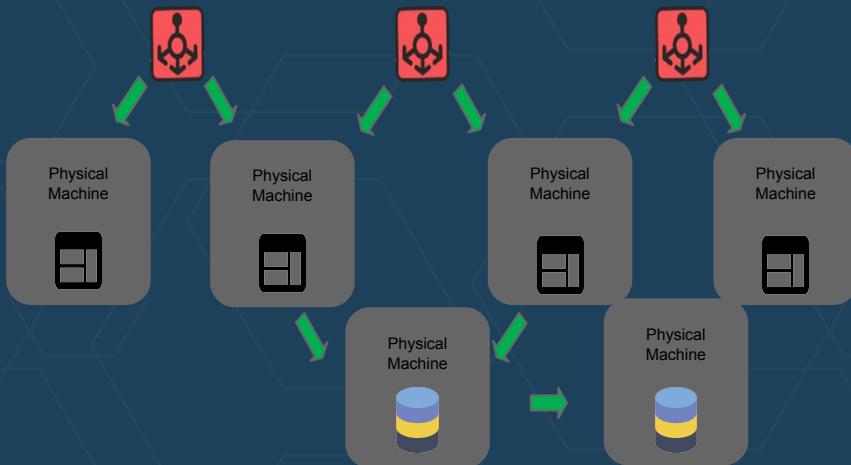


# Disaster Recovery Plan

San Antonio



New York City



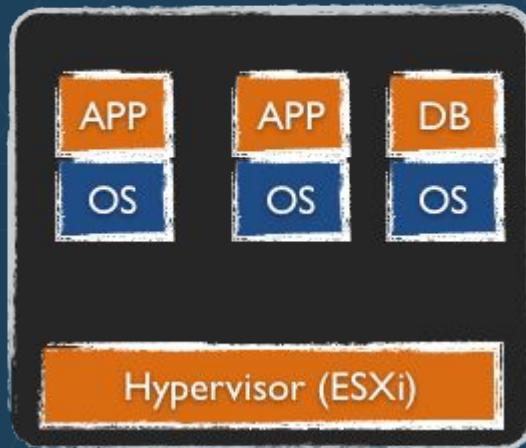
A few years later....in 2003

# VMware vMotion

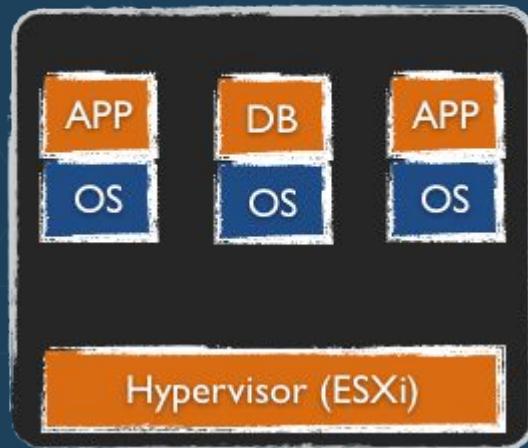
Hypervisor 1



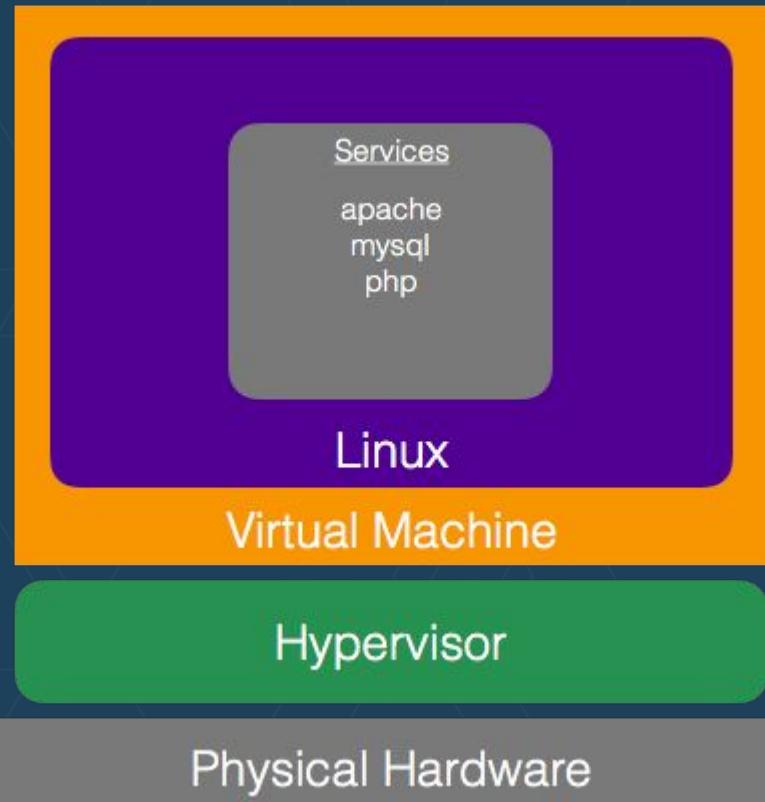
Hypervisor 2



Hypervisor 3



# Monolithic Application in a Virtual Machine



A few years later....in 2006



Posted August 25, 2006 by Jeff Barr (@jeffbarr) of Amazon—Amazon EC2 gives you access to a virtual computing environment. Your applications run on a "virtual CPU", the equivalent of a 1.7 GHz Xeon processor, 1.75 GB of RAM, 160 GB of local disk and 250 Mb/second of network bandwidth. You pay just 10 cents per clock hour (billed to your Amazon Web Services account), and you can get as many virtual CPUs as you need.

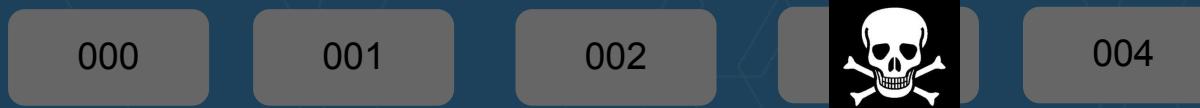
# Provides Power to Developers!

Web Developers build web applications.

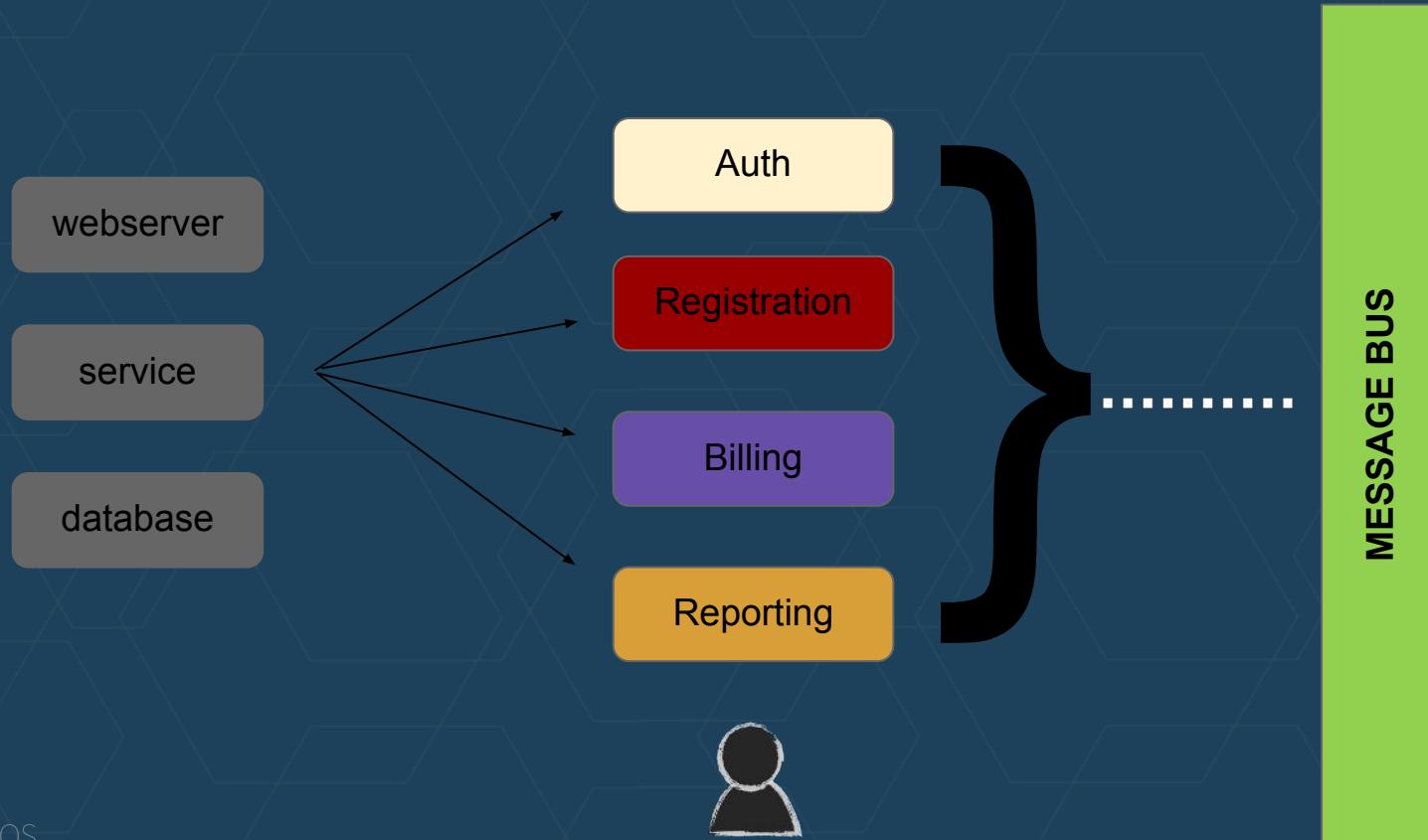
They don't manage the datacenter.

Quickly scale your application during times of success!

Quickly handle failure elegantly (and it will fail eventually)!

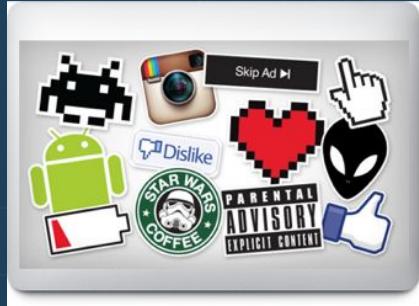


# Microservices & 12-Factor App Methodology





# GitHub



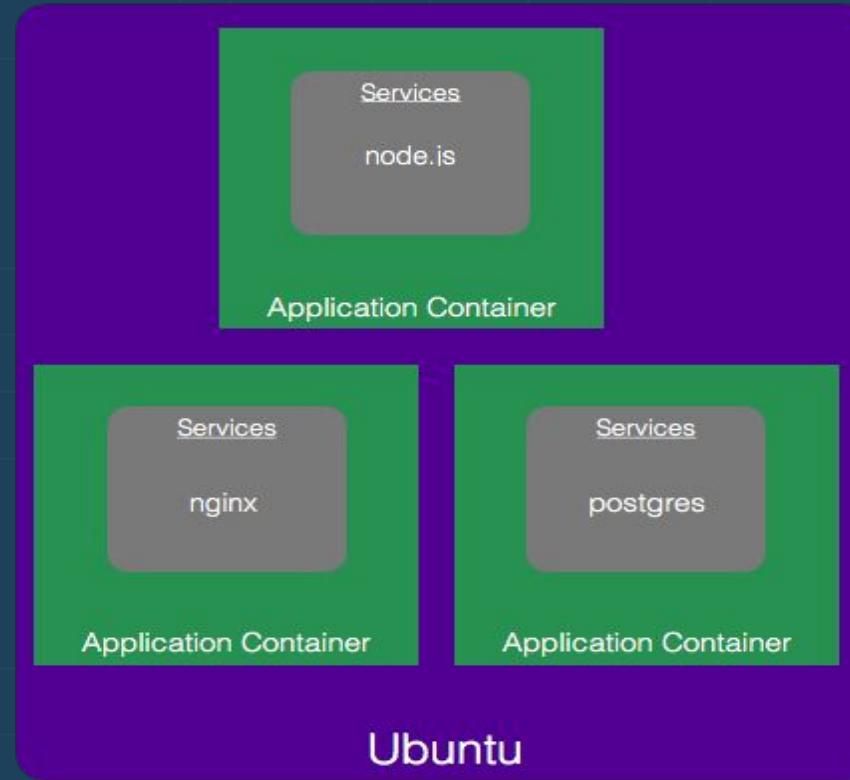
Between 2006 and 2008...

# Linux kernel begins adding “namespace” features

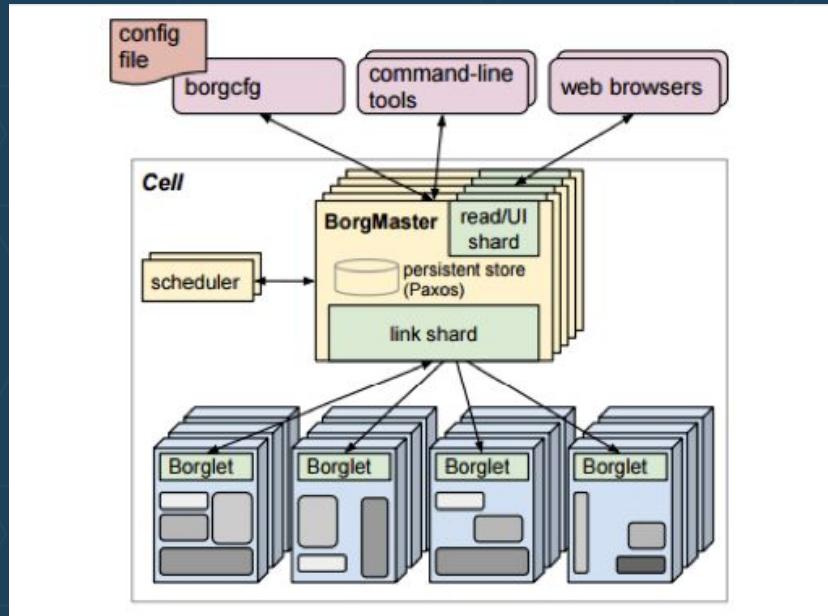
- mnt (mount points, filesystems)
- pid (processes)
- net (network stack)
- ipc (system V IPC)
- uts (hostname)
- user (UIDs)



# Application Containers



# Borg at Google



Google's **Borg** system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to **tens of thousands of machines**.

# And then in 2013





Client

docker build

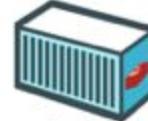
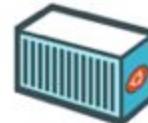
docker pull

docker run

DOCKER\_HOST

Docker daemon

Containers



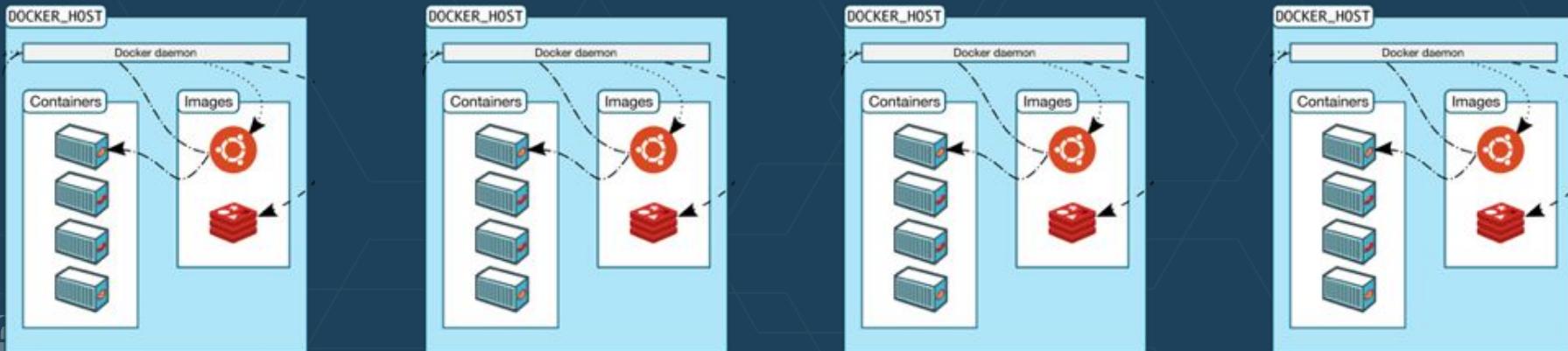
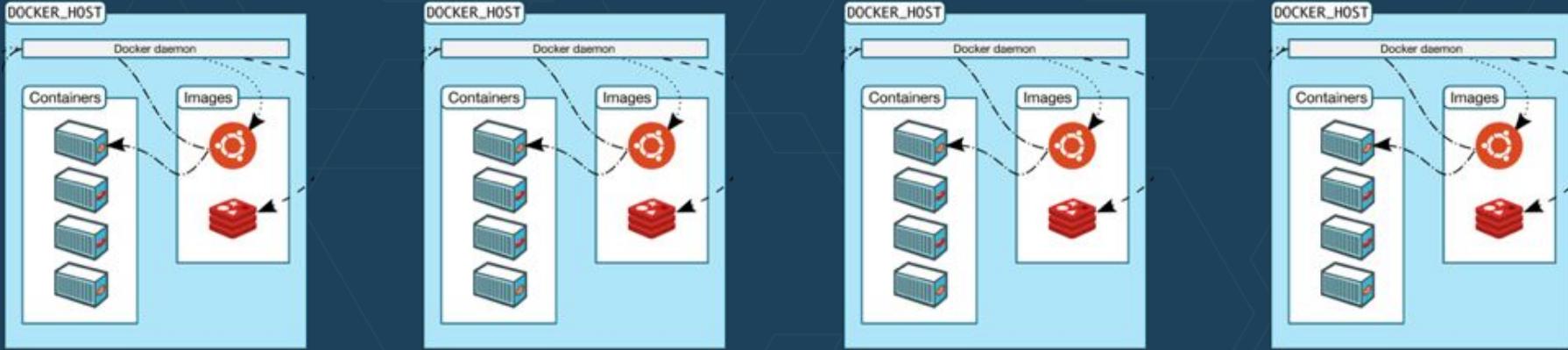
Images



Registry



NGINX



# Recap the Evolution.

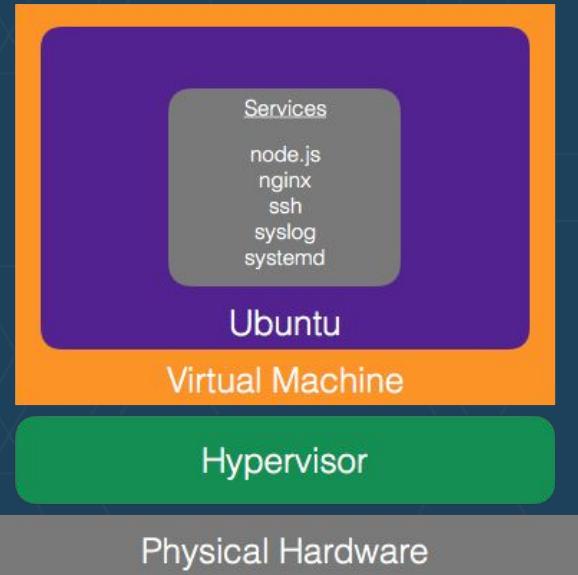
## Virtual Machines

Examples: ESXi, KVM, QEMU-KVM, Xen

Emulation of a computer system.

Executes an entire operating system.

Mutable.



# Recap the Evolution.

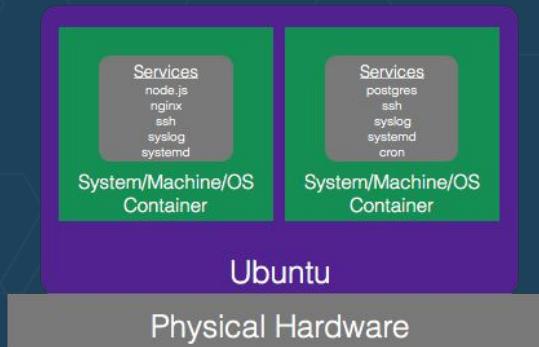
## System/Machine/OS Containers

Examples: LXC, LXD, OpenVZ, BSD Jails,  
Solaris Zones

Non-layered file-systems (i.e. ext4)

Seek to mimic virtual machines.

Mutable.



# Recap the Evolution.

## Application Containers

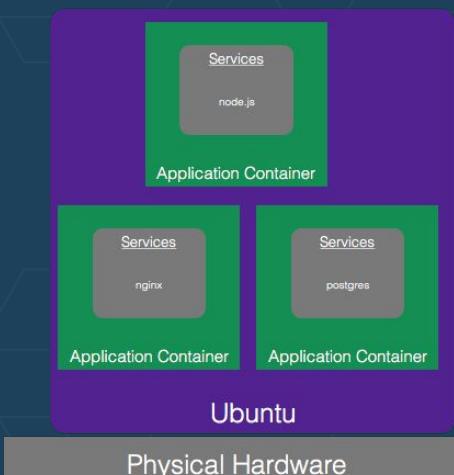
Examples: Docker and RKT.

Run a single application.

Layered Filesystems - overlay2, aufs, etc.

The perfect level of microservice-ness for creating cloud-native applications.

Immutable.



2014!

Kubernetes is Open Sourced!



# What is a “Kubernetes” ???

**Kubernetes** originates from Greek,  
meaning helmsman or pilot.

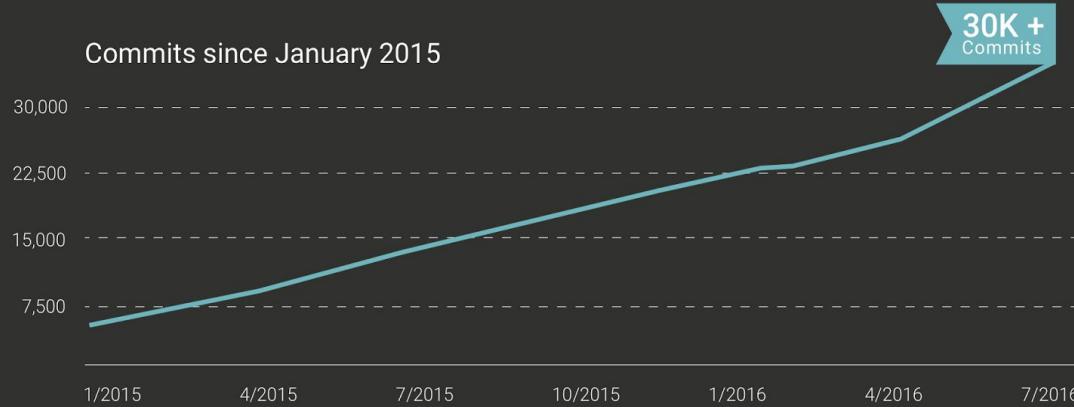
# What About K8s ???

**K8s** is an abbreviation derived by replacing the 8 letters “*ubernetes*” with “8”.

# Velocity of Kubernetes



kubernetes



**230+ years**  
of effort by the community



**800+**  
Contributors

<http://blog.kubernetes.io/2016/07/>

5k Commits  
in 1.2

+50% Unique  
Contributors

Top 0.01% of  
all Github  
Projects

1200+ External  
Projects Based  
on K8s

**Companies  
Contributing**



**Companies  
Using**



# 1. The Project



[github.com/kubernetes](https://github.com/kubernetes)

# 2. Community Deployment

kops

kubeadm

minikube

# 3. Product

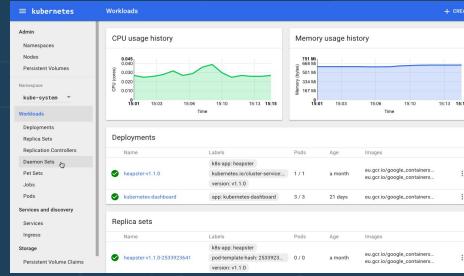


# 4. Service



# Manual

## 1. Dashboard



## 2. Command Line Interface

```
$ kubectl run -f my-new-deployment.yaml
```

## 3. SDK

```
pod, err :=  
c.Pods(v1.NamespaceDefault).Get("my-pod")  
  
if err != nil {  
    fmt.Println(err)  
  
return  
}
```

## 4. Helm

```
$ helm install stable/mariadb
```

Automation

API

# Kubernetes

- Production-grade container orchestration
  - Automated container deployment, scaling and management
  - Application-centered management
  - Decoupled/modular architecture
  - Open-sourced by Google with large community involvement
  - Based on Google's container-centric production infrastructure
  - Top 0.01% of all GitHub projects | 1000+ contributors | 43K+ commits
- Run anywhere
  - On-premise
  - Public cloud
  - Hybrid
  - Virtual machine or bare metal



# Kubernetes Cloud Providers

<https://github.com/kubernetes/kubernetes/tree/master/pkg/cloudprovider/providers>

**AWS**

**Azure**

**Cloudstack**

**GCE**

**Mesos**

**Openstack**

**Ovirt**

**Photon**

**Rackspace**

**Vsphere**

# What Does This Mean For Me?

# Faster Development!

## Increased Reliability

## More Compute (Less Money)

# Faster Development?

# CI/CD Pipeline



Code/Config Changes



Build

Test

Push Docker Image

Staging/QA Development

Production Deployment

# Kubernetes Quick Start

## Module 1

# kubectl

# kubectl

Command line interface for running commands against  
Kubernetes clusters.

```
kubectl [command] [TYPE] [NAME] [flags]
```

# kubectl explain

Get documentation of various resources. For instance  
pods, nodes, services, etc.

```
kubectl explain [--include-extended-apis=true] [--recursive=false] [flags]
```

# Exercise

kubectl

# kubectl run

Run a specified image on the cluster.

```
kubectl run NAME --image=image [--env="key=value"] [--port=port]
[--replicas=replicas] [--dry-run=bool] [--overrides=inline-json]
[flags]
```

# Deployments with kubectl run

- A deployment results in a running and managed application on Kubernetes
- The **kubectl run** command deploys pod(s) with a single container to the cluster

```
$ kubectl run hello-kube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
$ kubectl get deployments
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-kube    1         1         1            1           21h
$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-kube-2825702551-0bw69  1/1     Running   0          21h
```

# Exercise

`kubectl run`

# kubectl exec

Execute a command against a container in a pod.

```
kubectl exec POD [-c CONTAINER] [-i] [-t] [flags] [-- COMMAND [args...]]
```

# Executing Commands in Containers using kubectl exec

- You can execute commands in running containers

```
kubectl exec POD [-c CONTAINER] -- COMMAND [args...]
```

- This may be useful for troubleshooting
- For example, get a shell to a container in a pod and read the environment variables:

```
$ kubectl exec -it hello-kube-2825702551-fj92f -- /bin/bash
root@hello-kube-2825702551-fj92f:/# printenv
...
HELLO=WORLD
```

# Exercise

`kubectl exec`

# Exercise

## Blog

# Kubernetes Concepts

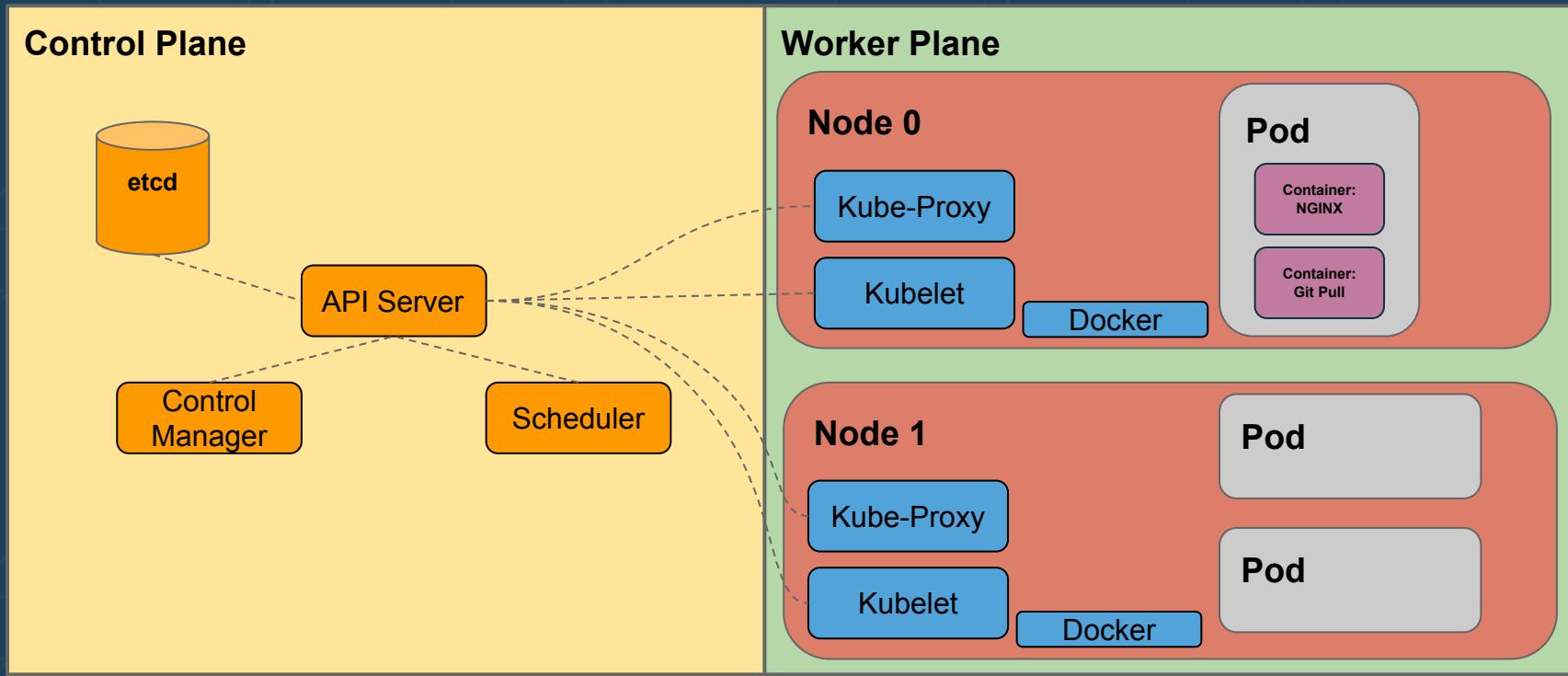
# Cluster

A group of nodes configured to run a functioning Kubernetes system.

# Kubernetes Cluster

<b>Master 0</b>	<b>Master 1</b>	<b>Master 2</b>	
<b>Worker 0</b>	<b>Worker 1</b>	<b>Worker 2</b>	<b>Worker 3</b>

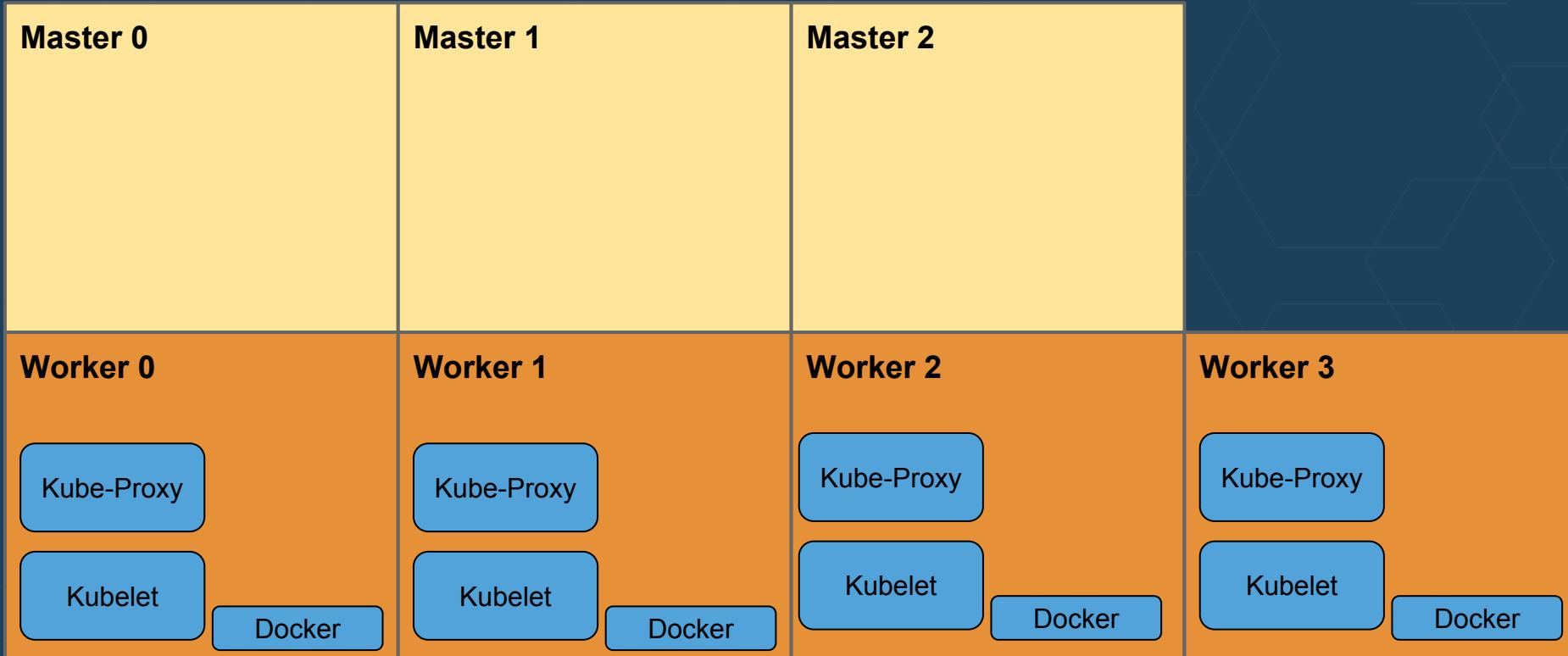
# Kubernetes Cluster



# Node

A worker machine (previously known as a *minion*). Contains the services necessary to run pods.

# Kubernetes Node

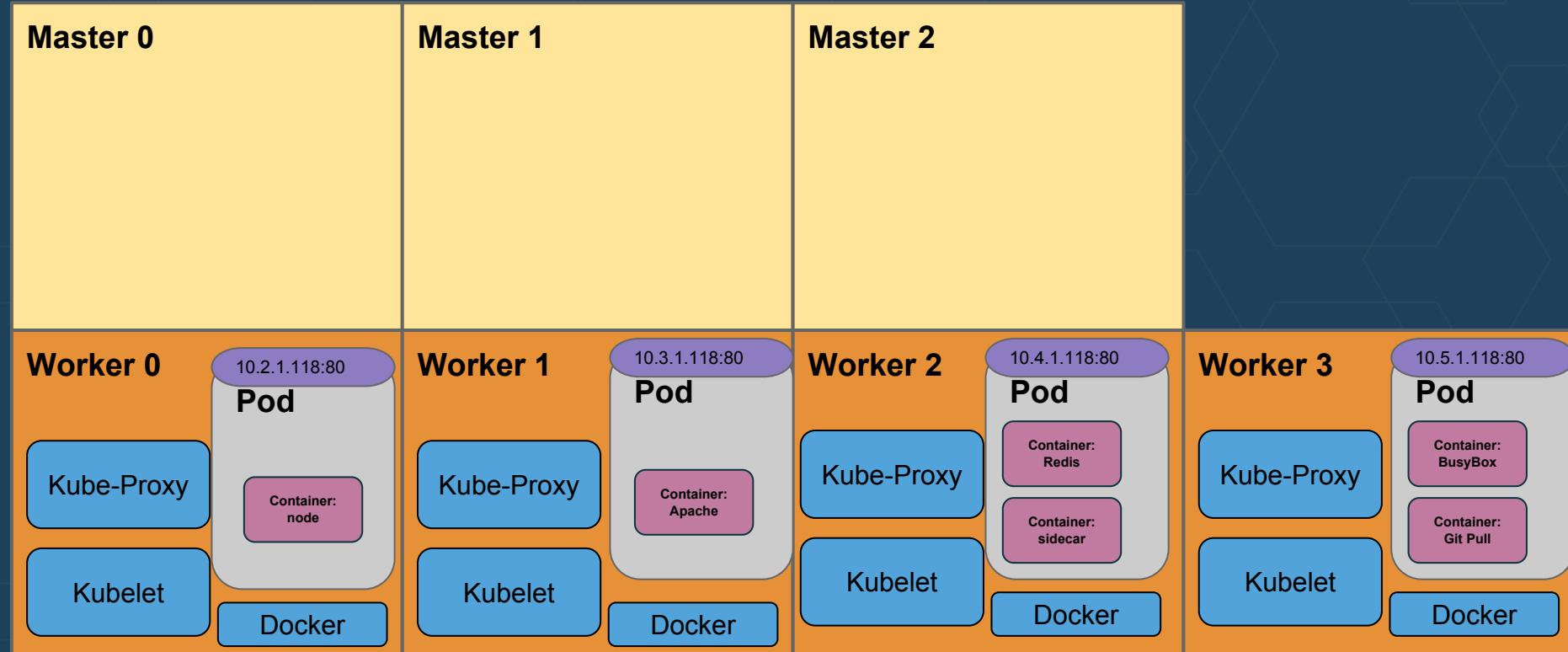


# Pod

A group of one or more containers  
running on a single node.

```
kubectl run my-first-pod --image=node:latest --restart=Never --port=80
```

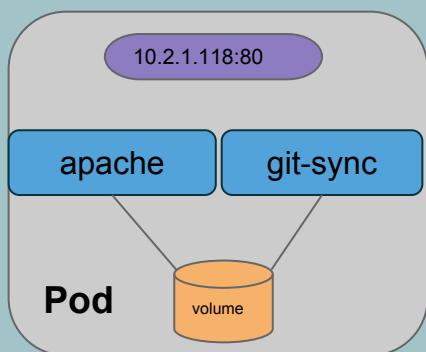
# Kubernetes Pod



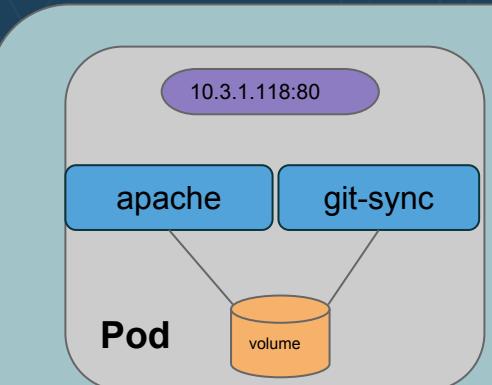
# Deployment

A declarative template for  
creating and scaling pods.

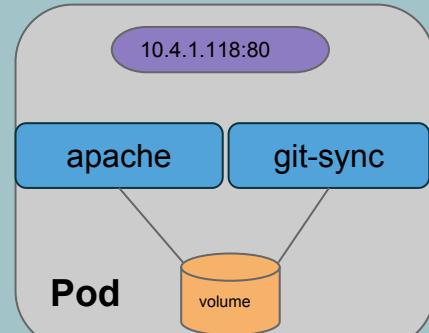
# Deployment



Node 1



Node 2



Node 3

# ReplicationSet

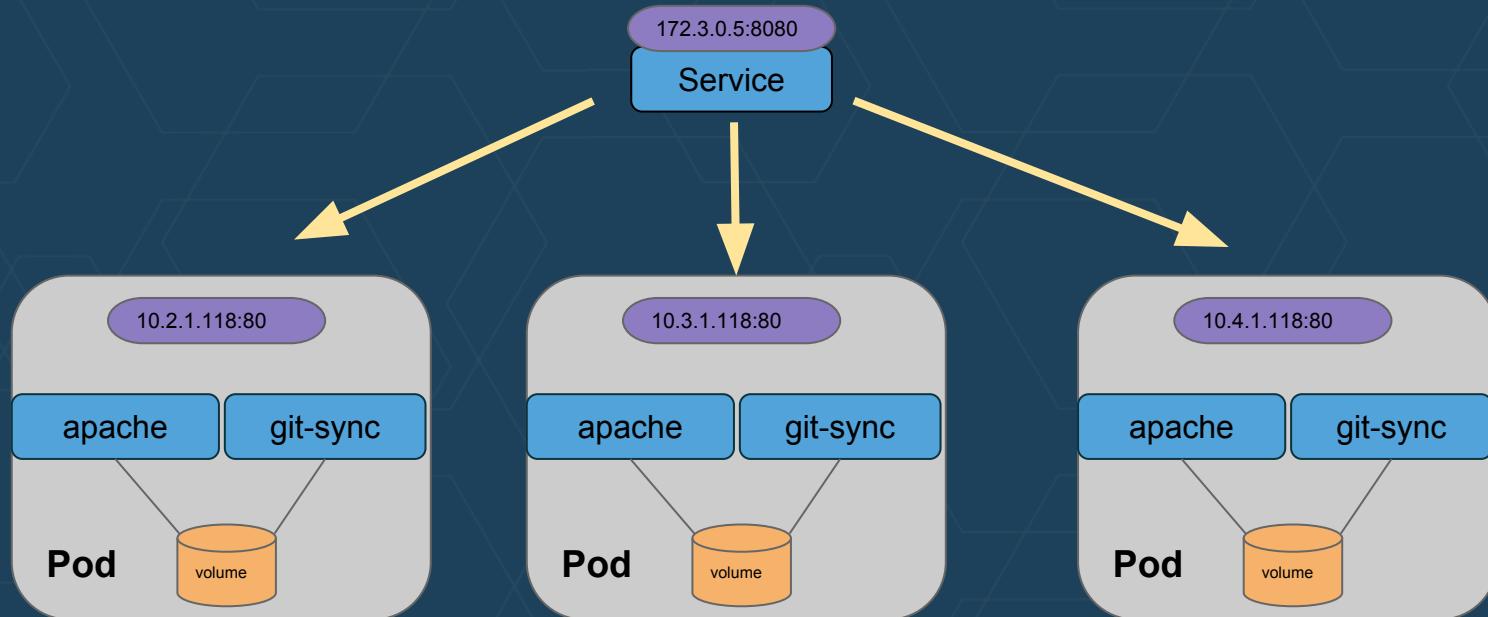
Coordinates pod creation, deletion, and updates during deployments.

# Service

A networking abstraction that defines rules to access pods determined by the selector.

# Service

- Persistent IPs for Pods
- Load Balances Between Replicas



# Labels

Key/value pairs attached to resources.

# Selector

A set of rules to match resources  
based on metadata.

# Labels and Selectors

- Kubernetes objects are organized with labels
  - Labels are key-value pairs attached to each object
  - More flexible than a rigid hierarchy
- Selectors can be passed to the API Server to retrieve a list of objects whose labels match that selector

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        tier: backend
  spec:
    containers:
      - name: nginx
        image: nginx
        ports:
          - containerPort: 80
```

```
$ kubectl get pods -l app=nginx
```

# Resource

Any individual Kubernetes item such as a deployment, pod, service, or secret, etc.

# Kubernetes Resources

- **Nodes**
- **Namespaces**
- **Pods**
- **Endpoints**
- **Services**
- **Deployments**
- **ReplicaSets**
- **Persistent Volumes**
- **PersistentVolumeClaims**
- **ConfigMaps**
- **DaemonSets**
- **StatefulSets**
- **Events**
- **PodDisruptionBudgets**
- **PodSecurityPolicies**
- **ResourceQuotas**
- **Service Accounts**
- **HorizontalPodAutoScalers**

# Name

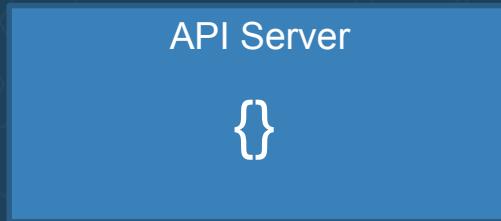
A unique identifier for a particular resource.

# Annotation

Non-identifying metadata.

# Behind the Scenes- The Kubernetes Architecture in a Nutshell

- At the heart of Kubernetes is the API Server
- The API Server contains JSON objects representing resources in Kubernetes
- The JSON objects are worked on individually to simplify and allow for scaling of the infrastructure



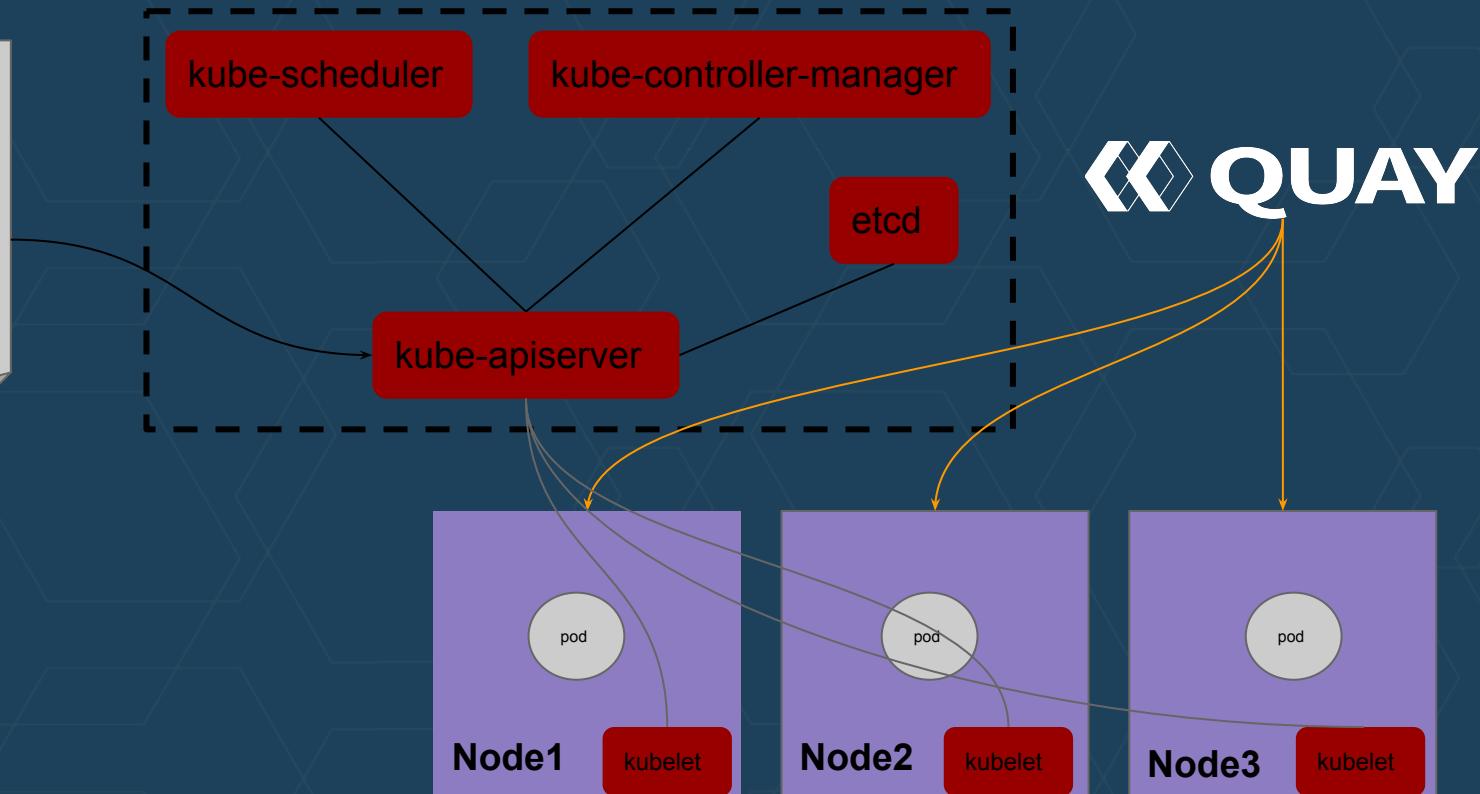
# Kubernetes API

Determines maturity of the API:

- **Alpha**: under active development.
- **Beta**: compatibility/upgradability guarantees.
- **Stable**: stable.

# Declarative Configuration Model

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        name: nginx
        role: frontend
        track: testing
    spec:
      containers:
        - name: quay.io/aptible/nginx
          image: nginx:1.12
          ports:
            - containerPort: 80
```



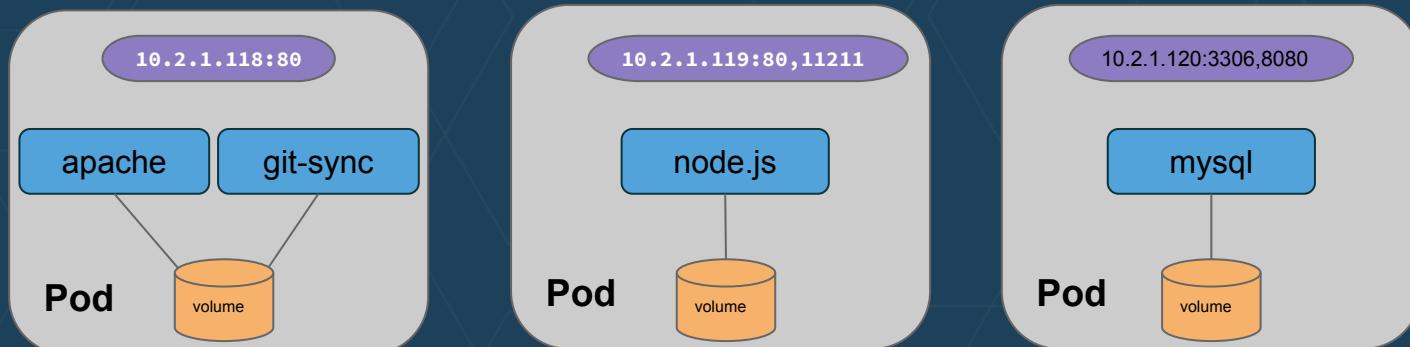
# Deployments Exposed

Module 2

# Pods

# Pods

- Smallest deployable unit in Kubernetes.
- Represents a running process on the cluster.
- A grouping of:
  - One or more containers.
  - Shared storage for the containers.
  - Unique network IP.
  - Options for running the container(s).



# Using Labels

Labels can be displayed as a column in output with -L option:

```
$ kubectl get pods -L tier
```

NAME	READY	STATUS	RESTARTS	AGE	TIER
my-nginx-3800858182-1v53o	1/1	Running	0	46s	backend
my-nginx-3800858182-2ds1q	1/1	Running	0	46s	backend

## Deployment's labels

```
$ kubectl get deployment/my-nginx -L tier
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE	TIER
my-nginx	2	2	2	2	2m	backend

<http://kubernetes.io/docs/user-guide/labels/>

# Using Labels Effectively

Examples of multiple labels for app, tier and role:

```
labels:  
  app: guestbook  
  tier: frontend
```

```
labels:  
  app: guestbook  
  tier: backend  
  role: master
```

```
labels:  
  app: guestbook  
  tier: backend  
  role: slave
```

Other example labels:

- "release" : "stable" or "canary"
- "partition" : "customerA" or "customerB"
- "track" : "daily" or "weekly"

# Updating Labels

Sometimes existing pods and other resources need to be relabeled before creating new resources

```
$ kubectl label pods -l app=nginx,tier=fe # select by label app=nginx, apply tier=fe
pod "my-nginx-v4-9gw19" labeled
pod "my-nginx-v4-hayza" labeled
pod "my-nginx-v4-mde6m" labeled
pod "my-nginx-v4-sh6m8" labeled
pod "my-nginx-v4-wf0f4" labeled
```

```
$ kubectl get pods -l app=nginx -L tier
NAME          READY   STATUS    RESTARTS   AGE   TIER
my-nginx-v4-9gw19  1/1    Running   0          15m   fe
my-nginx-v4-hayza  1/1    Running   0          14m   fe
my-nginx-v4-mde6m  1/1    Running   0          18m   fe
my-nginx-v4-sh6m8  1/1    Running   0          19m   fe
my-nginx-v4-wf0f4  1/1    Running   0          16m   fe
```

# spec.template.metadata.labels

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        deployer: coreos-training
    spec:
      containers:
        - name: nginx
          image: nginx:latest
```

# Guestbook Example Labels Usage

Examples of labels used in guestbook example:

```
$ kubectl get pods -L app -L tier -L role
NAME          READY   STATUS    RESTARTS   AGE     APP      TIER      ROLE
guestbook-fe-4nlpb   1/1    Running   0          1m     guestbook  frontend  <none>
guestbook-fe-ght6d   1/1    Running   0          1m     guestbook  frontend  <none>
guestbook-fe-jpy62   1/1    Running   0          1m     guestbook  frontend  <none>
guestbook-redis-master-5pg3b  1/1    Running   0          1m     guestbook  backend   master
guestbook-redis-slave-2q2yf   1/1    Running   0          1m     guestbook  backend   slave
guestbook-redis-slave-qgazl   1/1    Running   0          1m     guestbook  backend   slave
my-nginx-divi2        1/1    Running   0          29m    nginx     <none>   <none>
my-nginx-o0ef1        1/1    Running   0          29m    nginx     <none>   <none>
$ kubectl get pods -l app=guestbook,role=slave
NAME          READY   STATUS    RESTARTS   AGE
guestbook-redis-slave-2q2yf   1/1    Running   0          3m
guestbook-redis-slave-qgazl   1/1    Running   0          3m
```

# Exercise

Pods

# Manifests

# Pod Manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    environment: dev
    tier: frontend
spec:
  containers:
    - name: server
      image: nginx:1.13
      ports:
        - containerPort: 80
          protocol: TCP
```

# kubectl create

Create one or more resources from a file or stdin.

```
kubectl create -f FILENAME [flags]
```

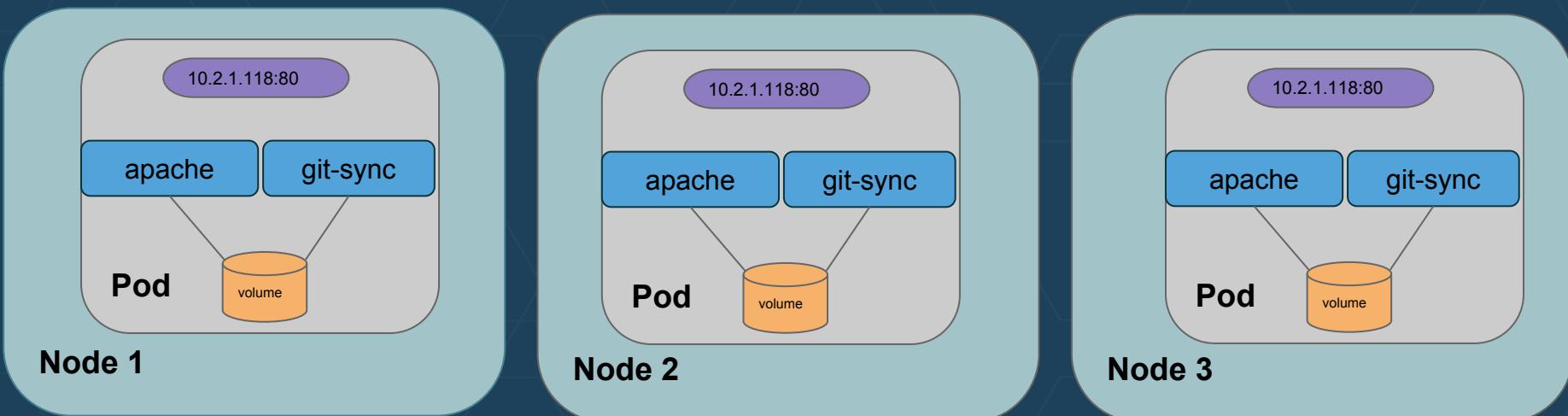
# Exercise

## Manifests

# Deployments

# Deployments

- Drive current state toward desired state.



# The Deployment Object

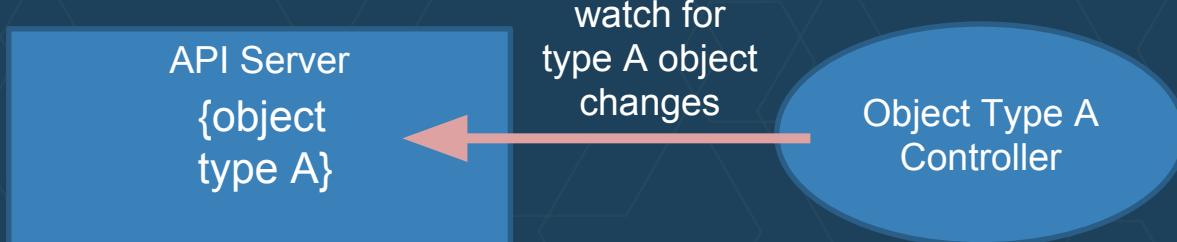
- The parameters of the `kubectl run` command provides the initial contents of a deployment object in the API Server

```
$ kubectl run hello-kube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
```

API Server  
{deployment  
object}

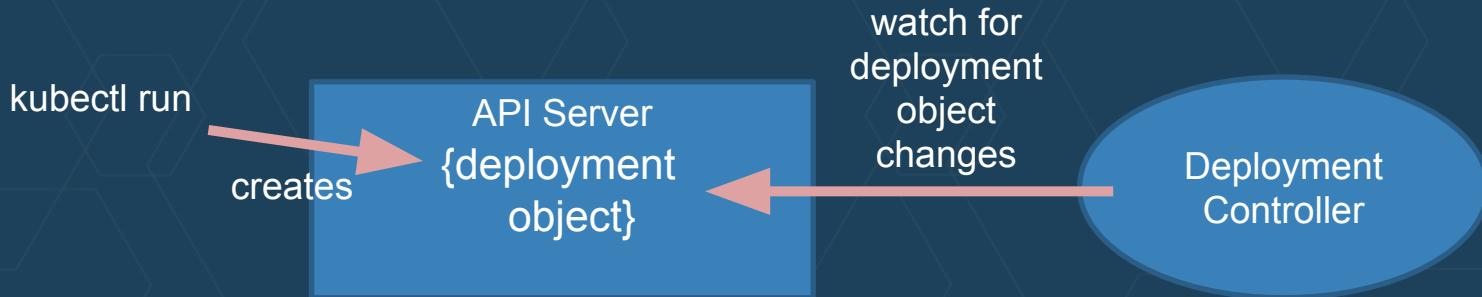
# Controllers

- Controllers are a basic pattern in Kubernetes
- Controllers are code that watches the API Server for changes to object types of interest, and takes action based on those changes
  - Controllers read the desired state, evaluate the current state and reconcile the differences



# Deployment Controller

- The deployment controller watches for new or modified deployment objects
- The deployment controller is responsible for creating and managing the deployment



# Desired State and Actual State

- The deployment object contains a desired state object (spec) from the user and the actual state from the deployment controller (status)



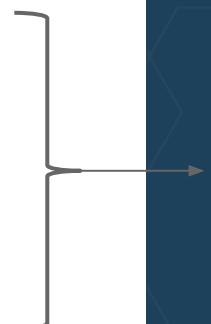
# Deployments using a Configuration File

- Instead of using the **kubectl run** command to create a deployment object, you can specify a deployment YAML file
  - kubectl converts the YAML to JSON before sending it to the API Server
- Advantages over **kubectl run**:
  - Declarative (what to do) instead of imperative (how to do it)
  - Can track your changes in Git
  - Can have multiple containers in a pod
- Other Kubernetes objects (services, pods, etc.) can also be configured with YAML or JSON files

# Deployment Example (1/2)

Run three replicas (copies) of pods containing an nginx web server/proxy/load balancer

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        tier: backend
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```



Pod spec template defines the 'cookie cutter' used for creating new pods when necessary

# Deployment Example (2/2)

```
$ kubectl create -f nginx-deployment.yaml
deployment "nginx-deployment" created

$ kubectl describe deployments/nginx-deployment
Name:           nginx-deployment
Namespace:      default
CreationTimestamp:  Wed, 24 Aug 2016 13:17:36 -0400
Labels:         app=nginx
                tier=backend
Selector:       app=nginx,tier=backend
Replicas:       3 updated | 3 total | 3 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 1 max surge
OldReplicaSets: <none>
NewReplicaSet:  nginx-deployment-4253660899 (3/3 replicas created)
Events:
[...]
```

You can also use '`kubectl get deployment nginx-deployment -o json`' to view the JSON

# Deleting Deployments

kubectl delete deployment

```
$ kubectl delete deployment/my-nginx  
deployment "my-nginx" deleted
```

If you try to delete the pods or ReplicaSets before deleting the deployments, Kubernetes will just replace them

# Summary: Deployments, ReplicaSets and Pods



- A Deployment object results in the creation of a ReplicaSet (rs) object, which results in a number of Pod objects

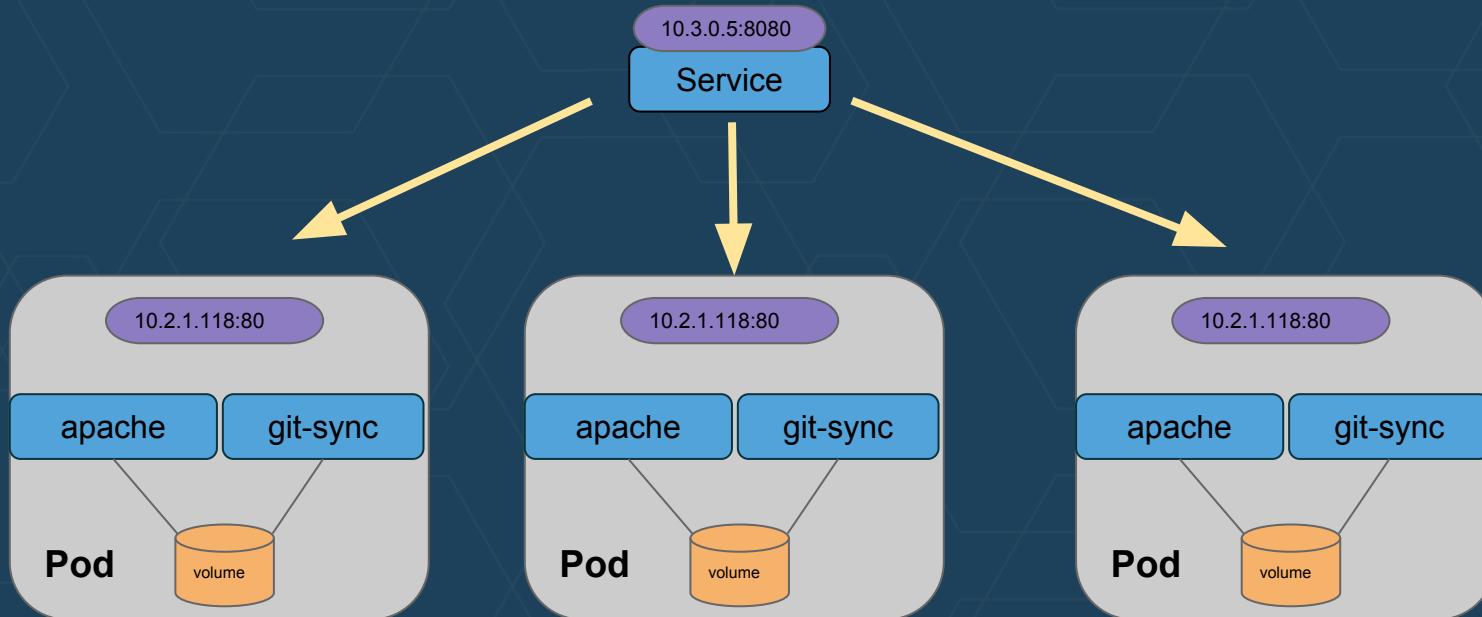
# Exercise

## Deployments

# Services

# Services

- Persistent IPs for Pods
- Load Balances Between Replicas

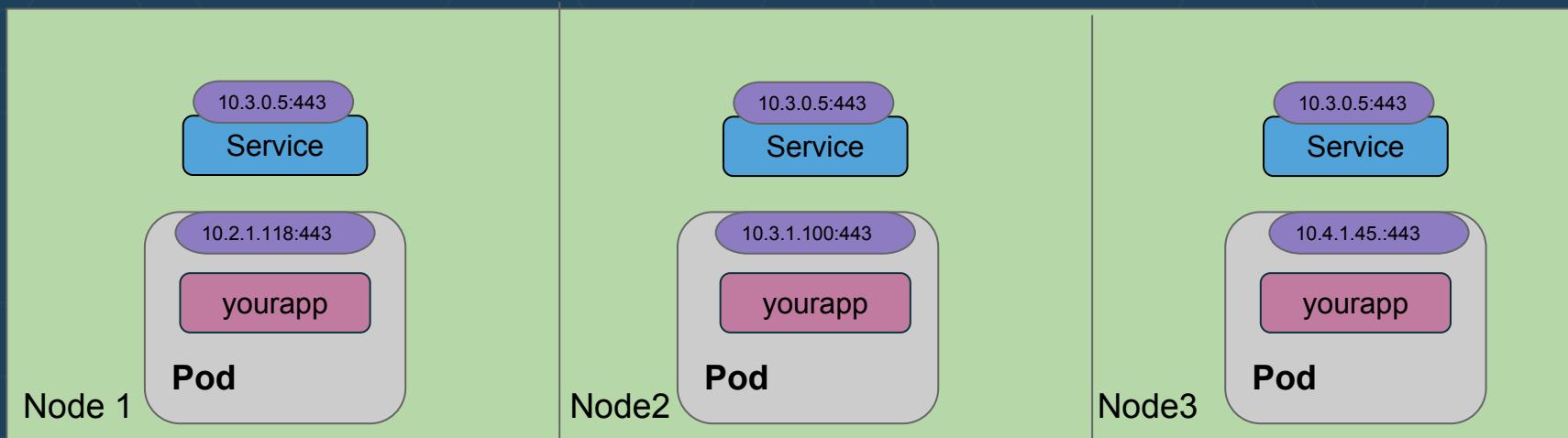


# Service Types

- ClusterIP (shown previously)
  - Exposes a service using an internal IP accessible only in the Kubernetes cluster
- NodePort
  - Exposes a service at every node <NODE\_IP>:<NODE\_PORT>
- LoadBalancer
  - Works with a cloud provider to create a load balancer and rules to expose the service as a layer on top of NodePort

# ClusterIP

```
Kubectl create -f deployment.yaml  
Kubectl create -f myservice.yaml
```



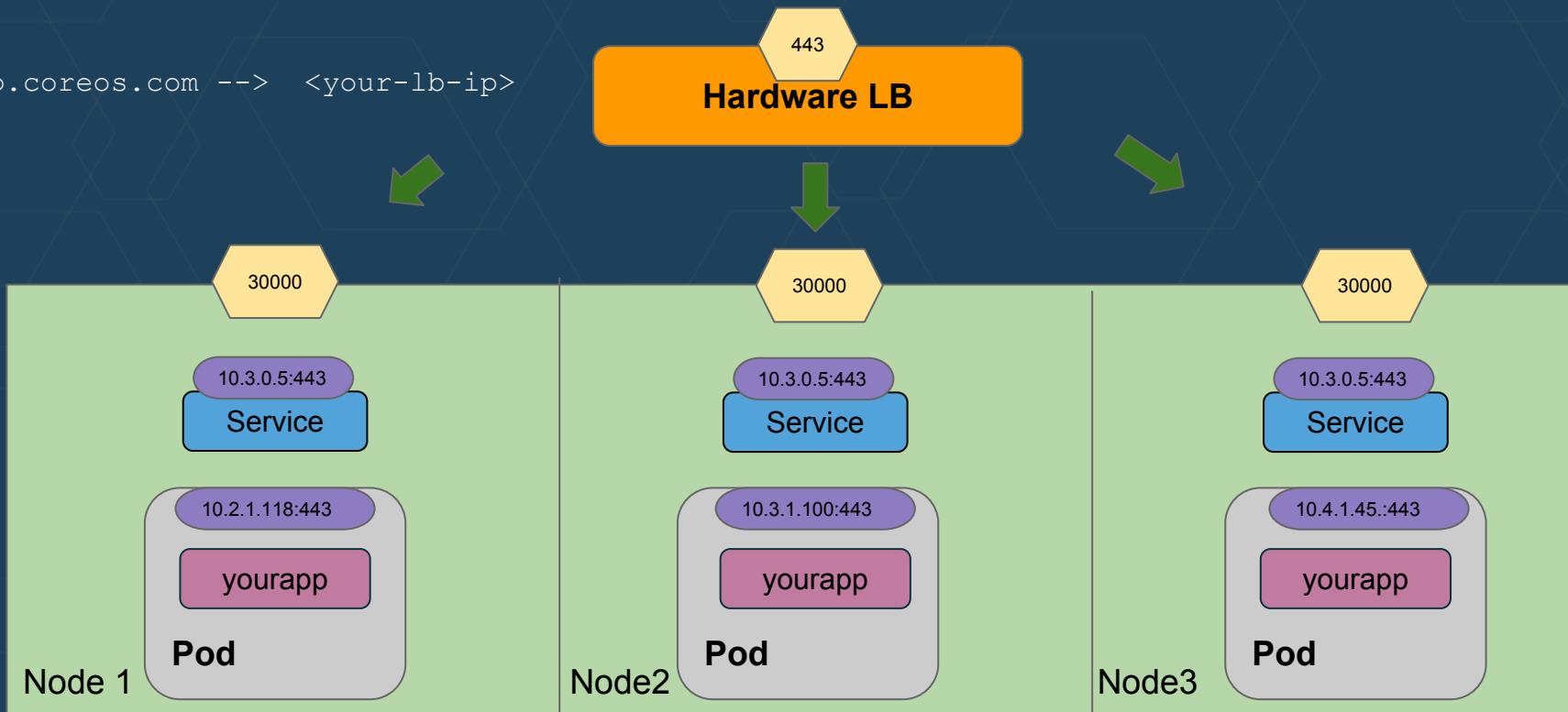
# Exercise

## ClusterIP Services

# NodePort

```
Kubectl create -f deployment.yaml  
Kubectl create -f myservice.yaml
```

myapp.coreos.com --> <your-lb-ip>



# Exercise

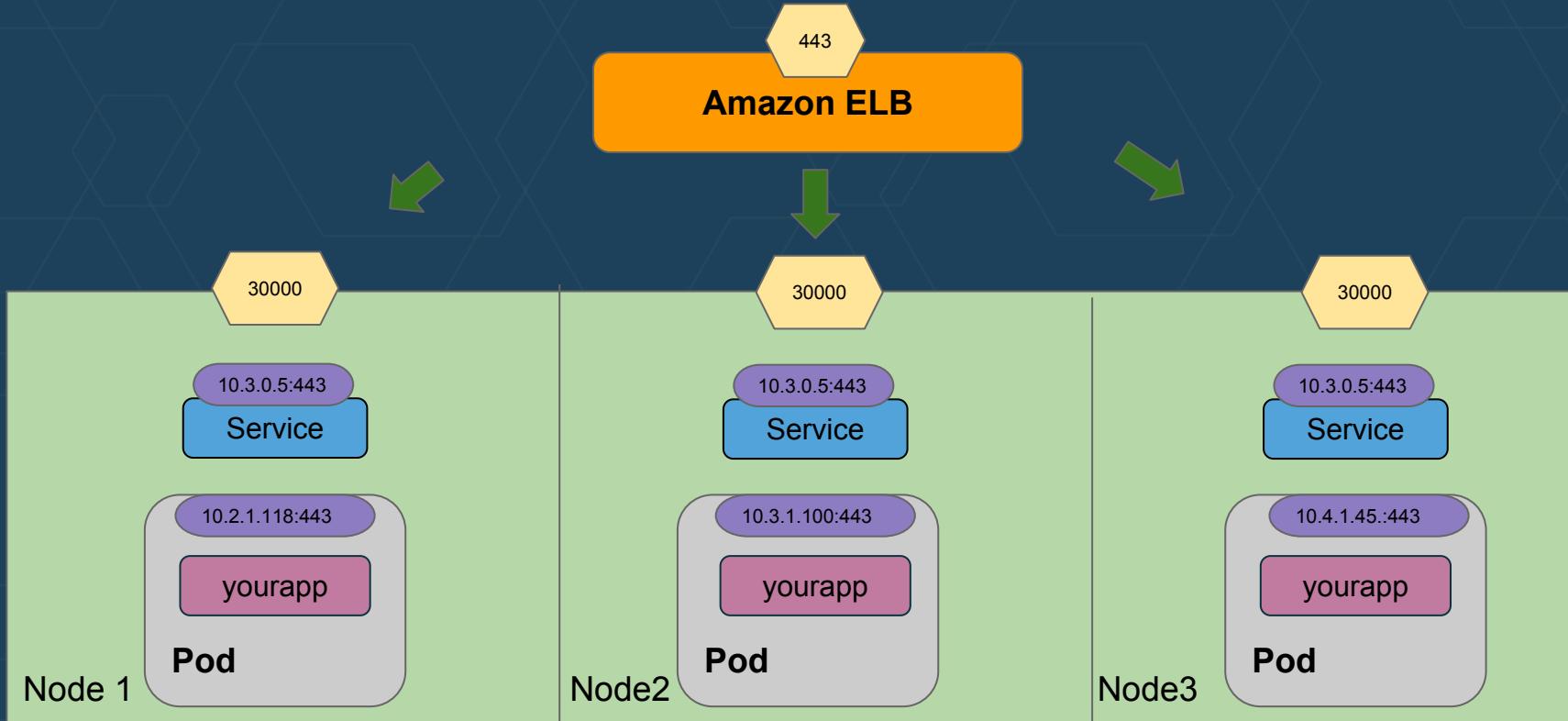
## NodePort Services

# Exercise

## ExternalName Services

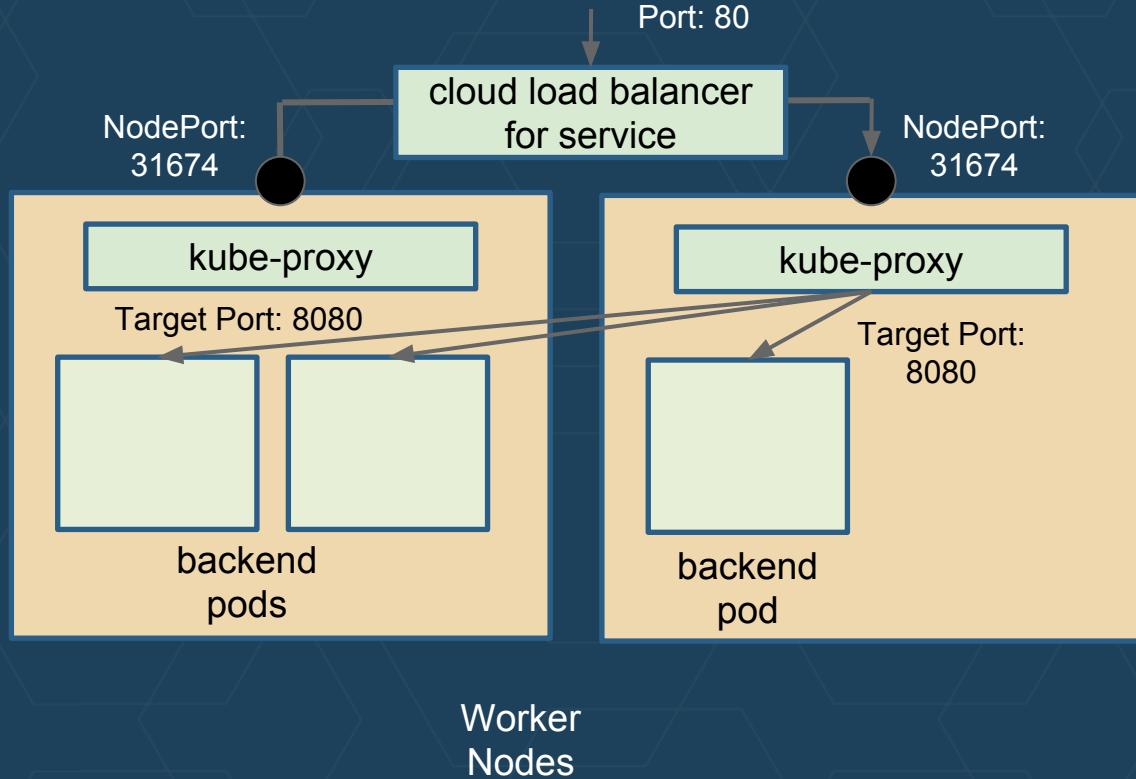
# LoadBalancer

```
Kubectl create -f deployment.yaml  
Kubectl expose... --port 443 --port  
443
```



# Example Service with LoadBalancer

- A cloud load balancer is created for every service
  - `myservice.mydomain.com`
- The load balancer picks a worker node and sends the request at NodePort 31674
  - This is configured in the load balancer
- `kube-proxy` identifies the service by port and load balances among all service pods



# Exercise

Load Balancer Service

# Running Applications

Module 3

# kubectl run

- ‘Off the cuff’ command to create **pods, deployments, or jobs**.
- Great for testing and troubleshooting.
- Similar to “docker run” in usage.

# Passing Environment Variables to Containers

- Environment variables can be passed to containers using `env`
- Applications can then read these values and act accordingly

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        tier: backend
    spec:
      containers:
        - name: nginx
          image: nginx
          env:
            - name: HELLO
              value: WORLD
      ports:
        - containerPort: 80
```

# Application Scaling

Module 4

# ReplicaSets

Behind the scenes, replicas (pod copies) in a deployment are represented and managed by ReplicaSets (RS)

- RS's job is to create/recreate/destroy pods when needed from a template in the manifest
- Results in self-healing and application high availability

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
        tier: backend
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
```

```
$ kubectl get replicaset
NAME           DESIRED   CURRENT   AGE
nginx-684635458   2         2        31m

$ kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
nginx-684635458-9p0kg   1/1     Running   0          31m
nginx-684635458-ibp6q   1/1     Running   0          31m
```

# Scaling your Application: Replicas

- Scale with kubectl from two to three replicas

```
$ kubectl scale deployment my-nginx --replicas=3
deployment "my-nginx" scaled
$ kubectl get pods -l app=nginx
NAME          READY     STATUS    RESTARTS   AGE
my-nginx-1jgkf 1/1      Running   0          3m
my-nginx-divi2 1/1      Running   0          1h
my-nginx-o0ef1 1/1      Running   0          1h
```

- You also could update the deployment config file

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        tier: backend
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

# Scaling your Application: Autoscale

Kubernetes automatically chooses the number of nginx replicas as needed, from one to three in this example

```
$ kubectl autoscale deployment my-nginx --cpu-percent=80 --min=1 --max=3
deployment "my-nginx" autoscaled
$ kubectl get pods -lapp=nginx
NAME        READY     STATUS    RESTARTS   AGE
my-nginx-1jgkf  1/1      Running   0          3m
my-nginx-divi2  1/1      Running   0          3m
$ kubectl get horizontalpodautoscaler
NAME        REFERENCE                               TARGET     CURRENT   MINPODS   MAXPODS   AGE
nginx      Deployment/my-nginx                   80%       39%      1          3          1m
```

# spec.replicas

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
```

# Scale Existing Deployment

Update existing deployment:

- `kubectl scale --replicas=3 deployment nginx-deployment`

Reference deployment specification file:

- `kubectl scale --replicas=3 -f my-deployment-spec.yml`

# Deployment Definition

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.12
          ports:
            - containerPort: 80
```

```
$ kubectl create -f nginx-deployment.yaml --record
deployment "nginx-deployment" created
```

```
$ kubectl get deployments
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	3	3	3	3	27s

```
$ kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
deployment "nginx-deployment" created
```

# kubectl scale

Update the size of the specified replication controller.

```
kubectl scale (-f FILENAME | TYPE NAME | TYPE/NAME) --replicas=COUNT
[--resource-version=version] [--current-replicas=count] [flags]
```

# Exercise

Scale the World

# Auto Scale

- Built-in support for CPU based autoscaling.

# kubectl autoscale

Automatically scale the set of pods that are managed by a replication controller.

```
kubectl autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS]  
--max=MAXPODS [--cpu-percent=CPU] [flags]
```

# Exercise

Autoscale

# Application Updates

Module 5

# Updating an Application

# Features of Rolling Deployments

- Updates are versioned
- Deployment has a version history for rollbacks
- Can specify the number of pods created at one time (default=1)
  - Can specify with number or percentage
- Check the rollout status with kubectl rollout status

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
        tier: backend
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

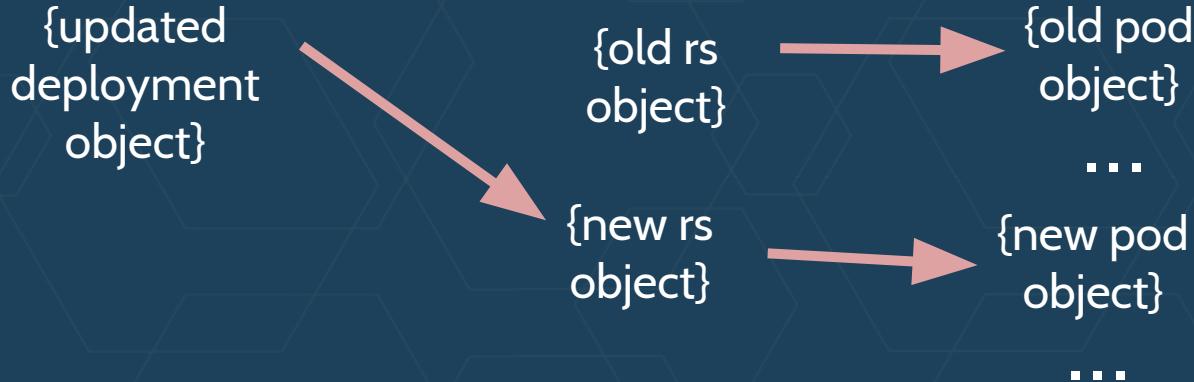
# kubectl apply

- Use **kubectl apply** to create or update a resource in Kubernetes
- Updating any part of the podSpec template of a deployment creates a rolling update
  - You can then watch the rolling update create and terminate pods

```
$ kubectl apply -f hello-kube.yaml
deployment "hello-kube" configured

$ kubectl get pods --watch
```

# Summary: Rolling Updates



- Updating a deployment results in a new rs object
- Pods are slowly added to the new rs object and removed from the old rs object
- When the old rs object no longer has pods to manage, it is destroyed

# Update Strategies

## Recreate

All existing pods are killed before new ones are created.

## Rolling Update

Specify **maxUnavailable** and **maxSurge** to control the rolling update process.

# Exercise

Update the World

# Configuration Files & Specs

Module 6

# Init Containers

# Init Containers

- Provides an easy way to block or delay startup of an application container until some condition is met.
- Can contain utilities that are too (insecure, large, insignificant, etc.) to run within the primary application container.
- They run **EVERY** time a pod is created.

# Exercise

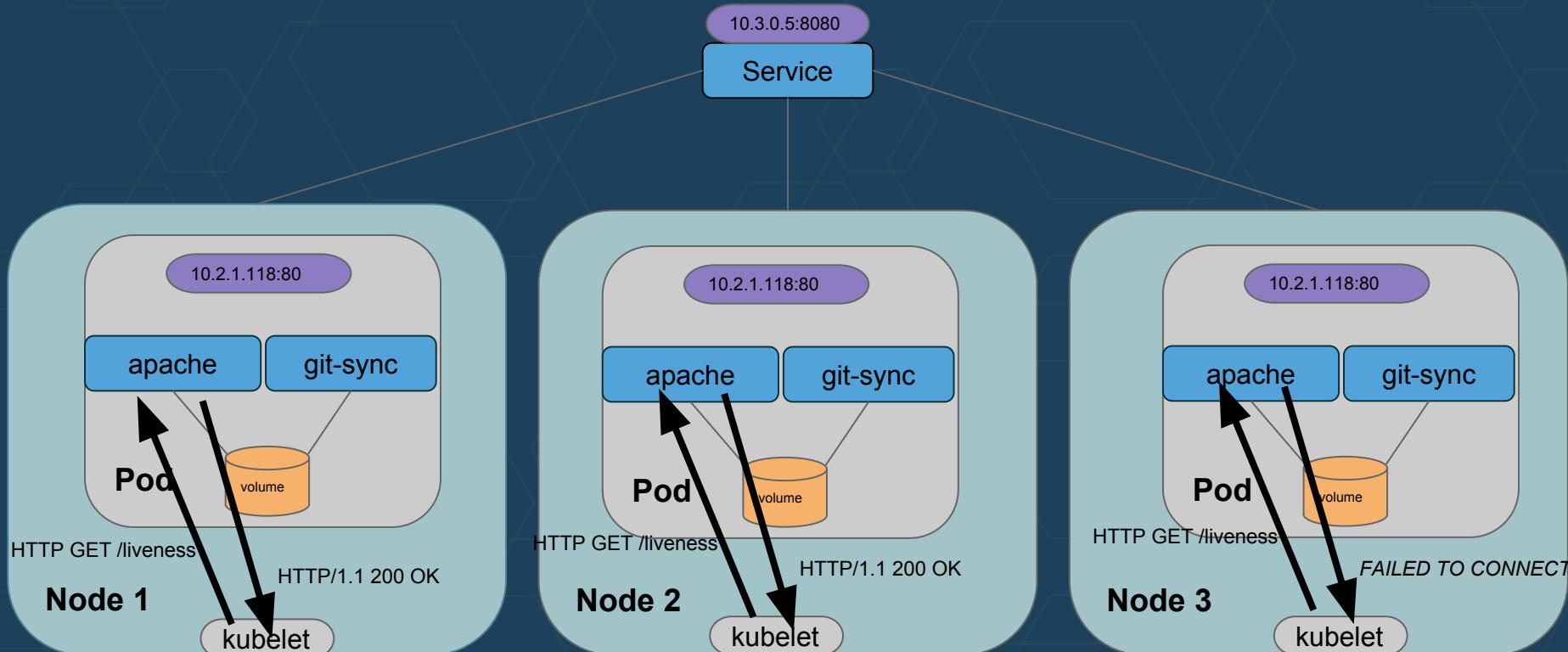
## Init Containers

# Liveness & Readiness Probes

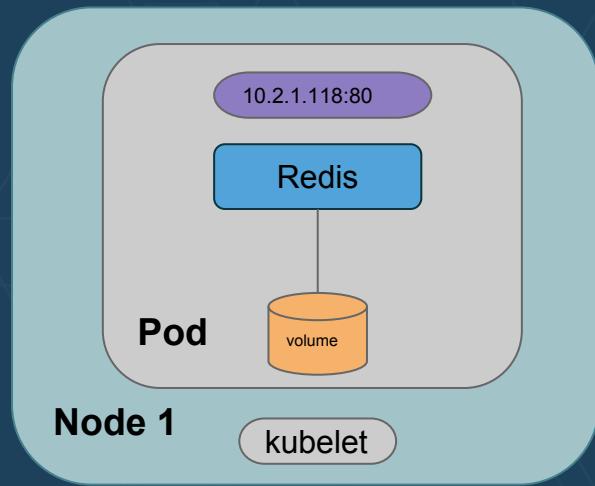
# Probes are Health Checks

- Used to monitor containers **AFTER** they start.
- Supported Types: HTTP Get, TCP socket, custom command.
- **Liveness:** tests if the container should be restarted.
  - Ideal for *broken* containers.
- **Readiness:** tests if the container should be taken out of service.
  - Ideal for containers that *aren't ready yet*.

# Probes in Action



# Examples!



Liveness  
TCP socket check

Readiness  
redis-cli ping

# Probe Lifecycle

1. Readiness Probe Fails
2. Kubernetes stops routing traffic to the pod
3. Liveness probe fails
4. Kubernetes restarts the pod
5. Readiness probe succeeds
6. Kubernetes starts routing traffic to the pod again

# Exercise

## Probes

# Dealing with Storage

Module 7

# Volumes

- Container filesystems are ephemeral!
- Often necessary to share files between containers in a Pod.
- Volumes abstraction solves these problems.

# Volumes

- Container filesystems are ephemeral!
- Often necessary to share files between containers in a Pod.
- Volumes abstraction solves these problems.

# Volume Types

## Non-Persistent

- downwardAPI
- emptyDir
- gitRepo
- hostPath
- projected
- secret

## Persistent

- awsElasticBlockStore
- azureDisk
- azureFileVolume
- gcePersistentDisk
- nfs
- etc...

# Using Volumes

- Provide the volumes to the Pod:
  - **spec.volumes**
- Provide the mount points in the container(s):
  - **spec.containers.volumeMounts**

# spec.volumes & spec.containers.volumeMounts

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: sample-deployment
spec:
  template:
    metadata:
      labels:
        app: volume-empty
    spec:
      containers:
        - name: volume-empty
          image: busybox
          stdin: true
          tty: true
          volumeMounts:
            - mountPath: /data
              name: volume-data
      volumes:
        - name: volume-data
          emptyDir: {}
```

# Persistent Volume

- Volume whose lifecycle is independent of any individual Pod.

# Application Configuration

Module 8

# ConfigMaps

# Consider the following...

```
$ cat /etc/mysql/my.cnf
[mysqld]
temp-pool
key_buffer_size=$buffer_size
datadir=/my/my
loose-innodb_file_per_table

[mariadb]
datadir=$my_data_dir
default-storage-engine=aria
loose-mutex-deadlock-detector
max-connections=$max_connections
```

# Why ConfigMaps?

Separates application **code** from  
application **configuration**.

# Three Ways to Consume

Command line arguments

Environment variables

Files in a volume

# Exercise

## ConfigMaps



# Exercise

## ConfigMap Volumes



# Secrets

# Secrets

- Secrets are intended to hold sensitive information, such as passwords, OAuth tokens, and ssh keys.
- Putting this information in a secret is safer and more flexible than putting it verbatim in a pod definition or in a docker image.

# Secrets

A secret can be used with a pod in two ways:

- As files in a volume mounted on one or more of its containers.
- Used by kubelet when pulling images for the pod.

# Viewing Secrets

- Secrets are stored in a Base64 encoded format.
- Must decode to view the actual secret value

```
$ kubectl get secret mysecret -o yaml
apiVersion: v1
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
kind: Secret
metadata:
  creationTimestamp: 2016-01-22T18:41:56Z
  name: mysecret
  namespace: default
  resourceVersion: "164619"
  selfLink: ...
type: Opaque
```

```
$ echo "MWYyZDFlMmU2N2Rm" | base64 --decode
1f2d1e2e67df
```

# Exercise

## Secrets



# Exercise

## Secret Volumes



# Exercise

## TLS Secrets

# Jobs & DaemonSets

Module 9

# Jobs

# Jobs

- A job creates one or more pods and ensures that a specified number of them successfully terminate.
- As pods successfully complete, the job tracks the successful completions.
- When a specified number of successful completions is reached, the job itself is complete.
- Deleting a Job will cleanup the pods it created.

# Exercise

Jobs

# DaemonSets

# DaemonSet

- A DaemonSet ensures that all (or some) nodes run a copy of a pod.
- As nodes are added to the cluster, pods are added to them.
- As nodes are removed from the cluster, those pods are garbage collected.
- Deleting a DaemonSet will clean up the pods it created.

# DaemonSet Typical Uses

- Running a cluster storage daemon, such as glusterd, ceph, on each node.
- Running a logs collection daemon on every node, such as fluentd or logstash.
- Running a node monitoring daemon on every node, such as Prometheus Node Exporter, collectd, Datadog agent, New Relic agent, or Ganglia gmond.

# Exercise

## DaemonSets

# Stateful Applications

Module 10

# StatefulSets

# StatefulSets

- A StatefulSet is a Controller that provides a unique identity to its Pods.
- It provides guarantees about the ordering of deployment and scaling.

# Stateful Sets

Valuable for applications that require one or more of the following.

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, graceful deletion and termination.
- Ordered, automated rolling updates.

# Limitations

- StatefulSet is a beta resource, not available in any Kubernetes release prior to 1.5.
- As with all alpha/beta resources, you can disable StatefulSet through the --runtime-config option passed to the apiserver.
- The storage for a given Pod must either be provisioned by a PersistentVolume Provisioner based on the requested storage class, or pre-provisioned by an admin.
- Deleting and/or scaling a StatefulSet down 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.
- StatefulSets currently require a Headless Service to be responsible for the network identity of the Pods. You are responsible for creating this Service.

# Ingress

## Module 11

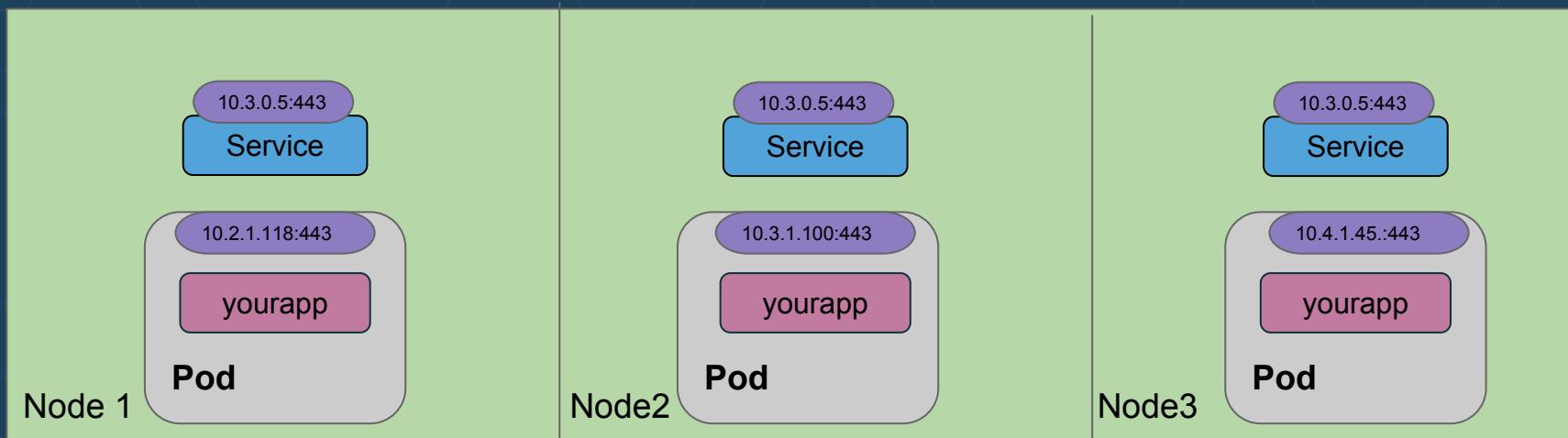
# Why INGRESS?

**...Because load-balancing provided  
by services is too simple!**

**We want rules on how to access a service.**

# ClusterIP

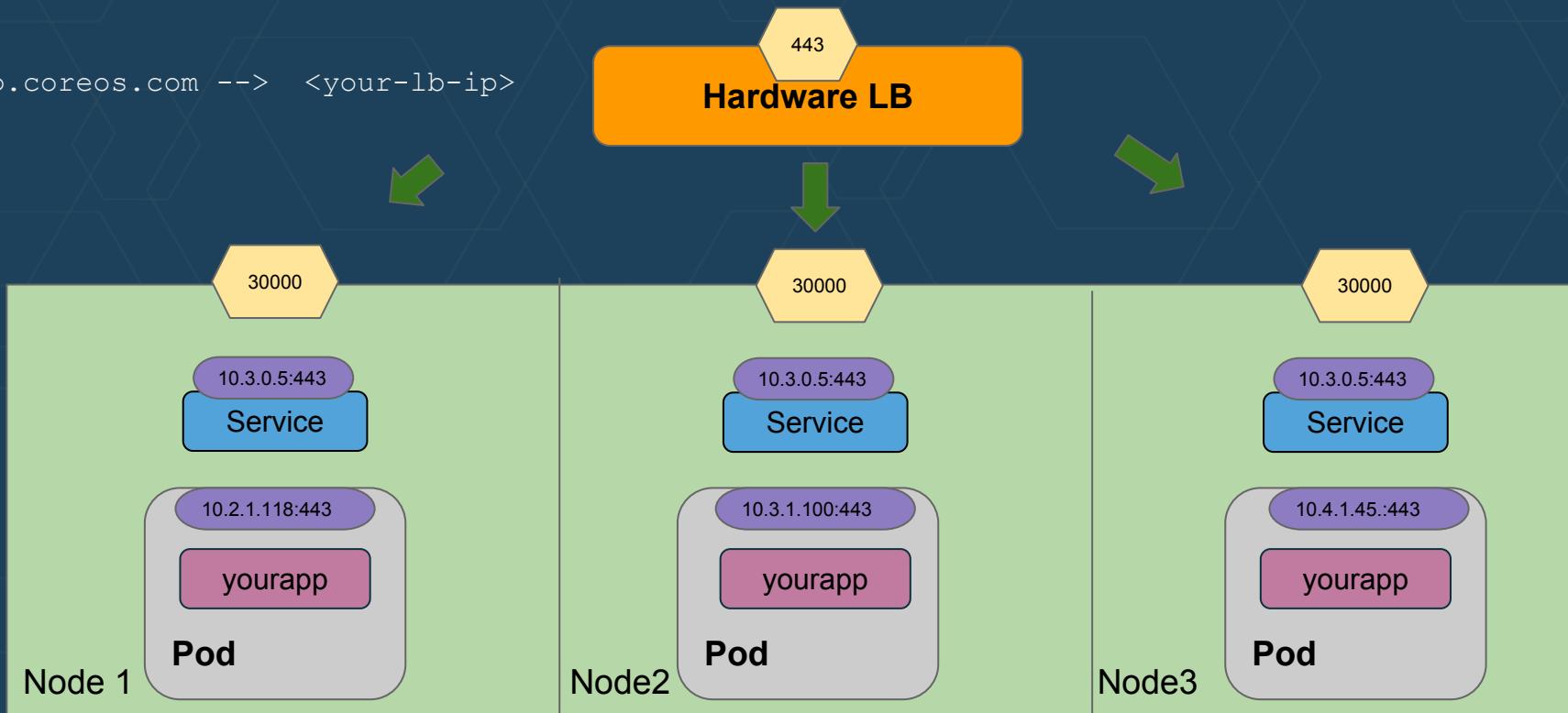
```
Kubectl create -f deployment.yaml  
Kubectl create -f myservice.yaml
```



# NodePort

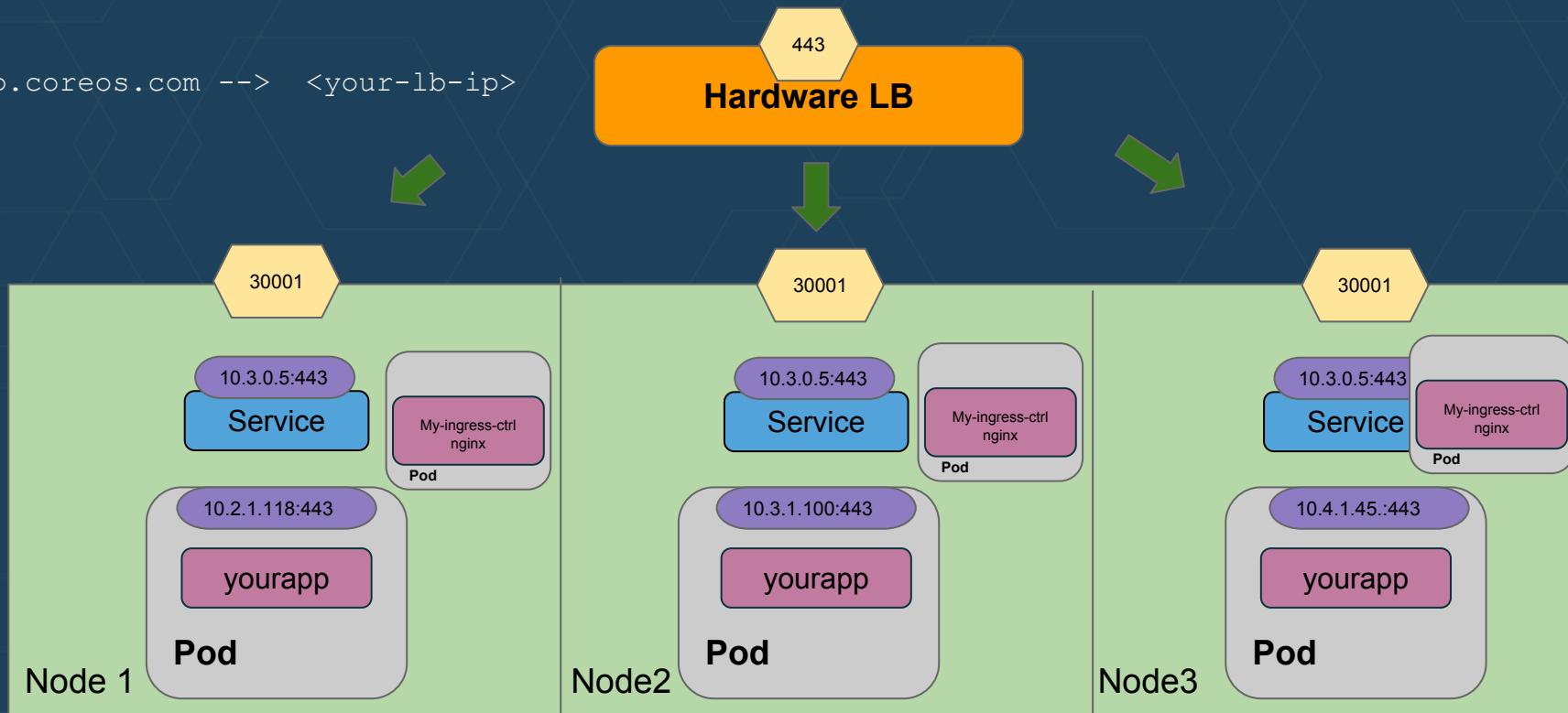
```
Kubectl create -f deployment.yaml  
Kubectl create -f myservice.yaml
```

myapp.coreos.com --> <your-lb-ip>

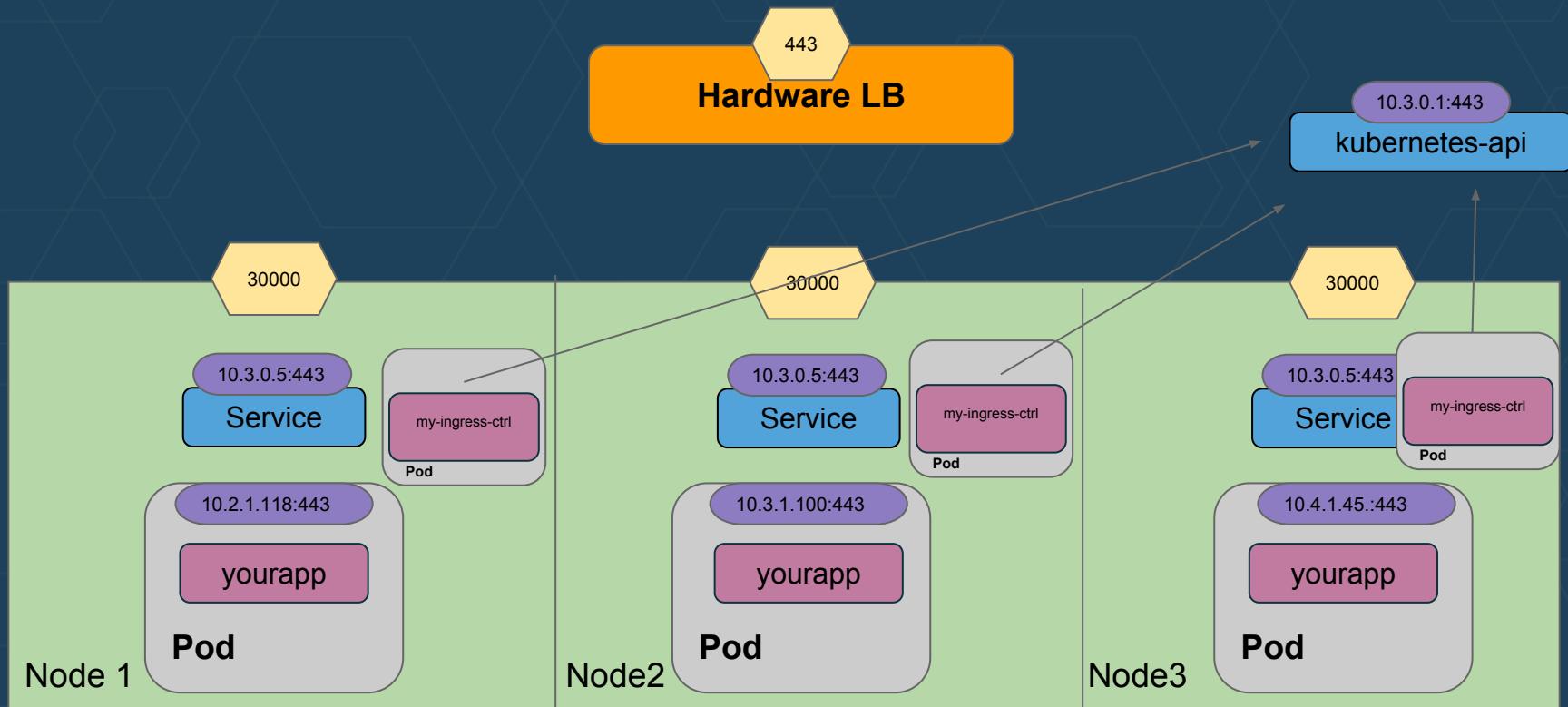


# Ingress Controller

myapp.coreos.com --> <your-lb-ip>



# Ingress Controller



# Ingress Object

Create Ingress object!

myapp1.coreos.com --> <your-lb-ip>  
myapp2.coreos.com --> <your-lb-ip>  
myapp3.coreos.com --> <your-lb-ip>

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

  name: demo-ingress

spec:

  rules:

    - host: myapp1.coreos.com

      http:

        paths:

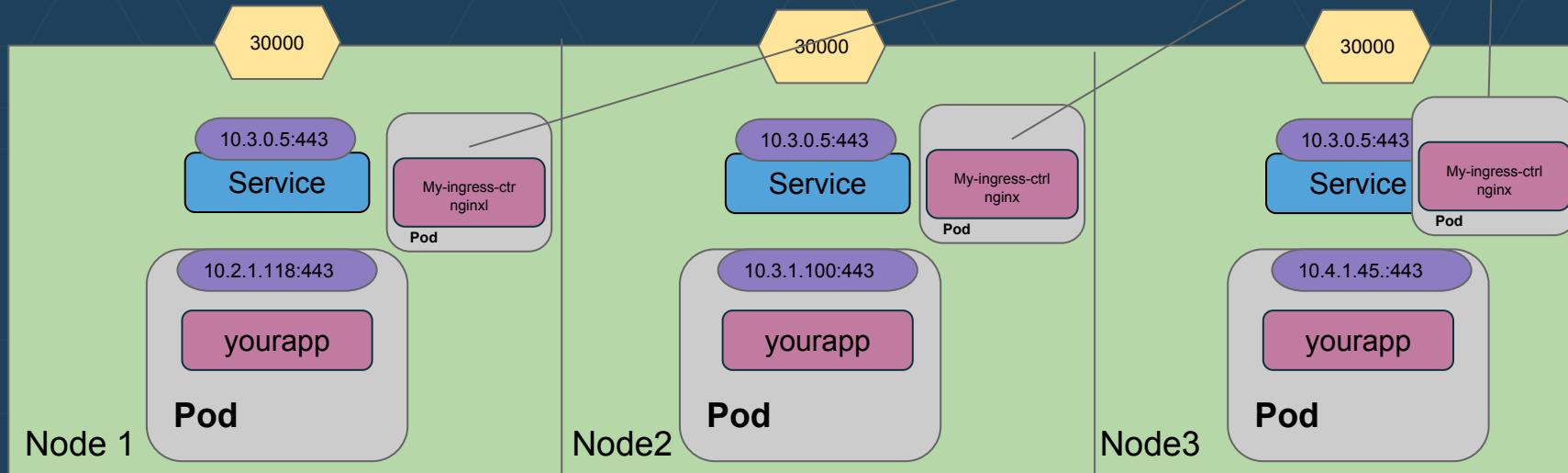
          - backend:

            serviceName: nginx1

            servicePort: 80

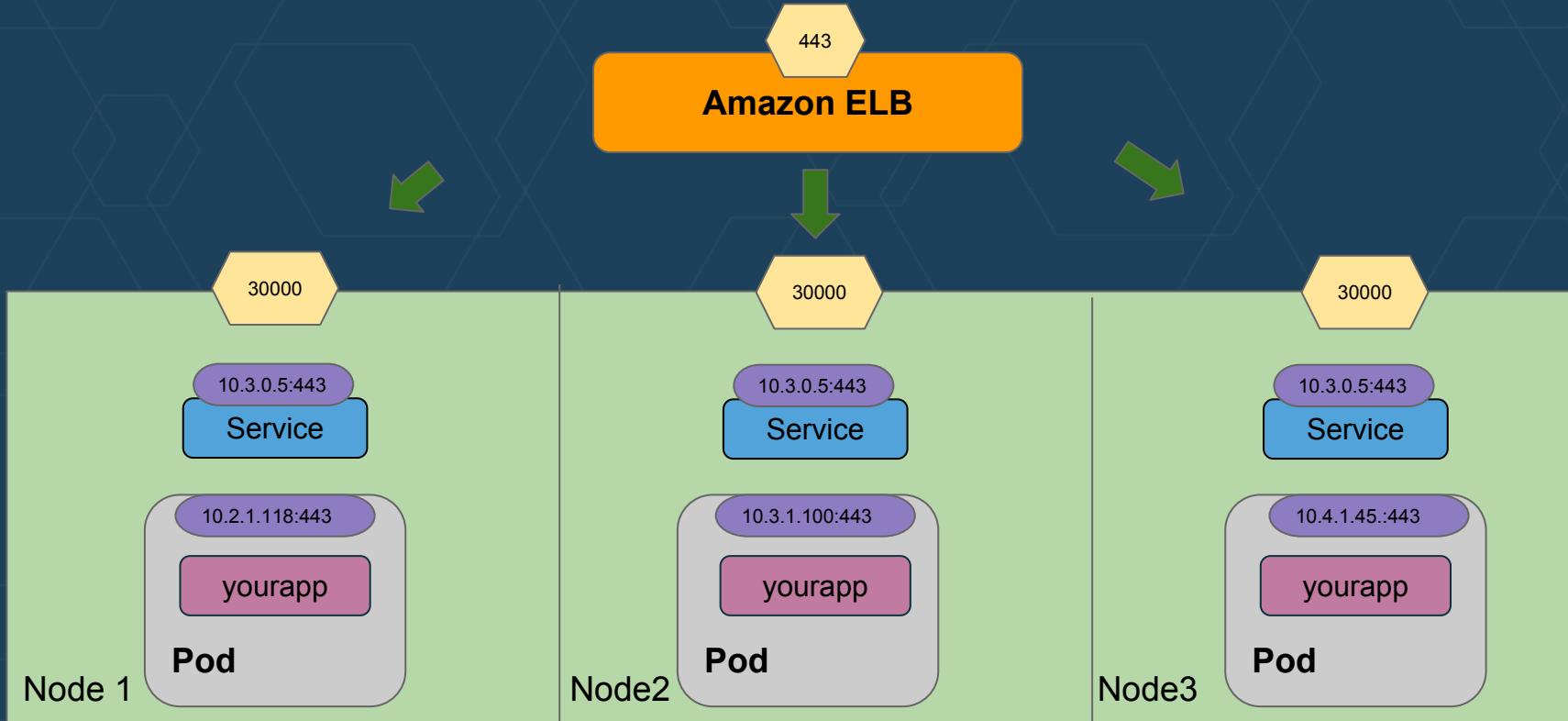
            10.3.0.1:443

kubernetes-api

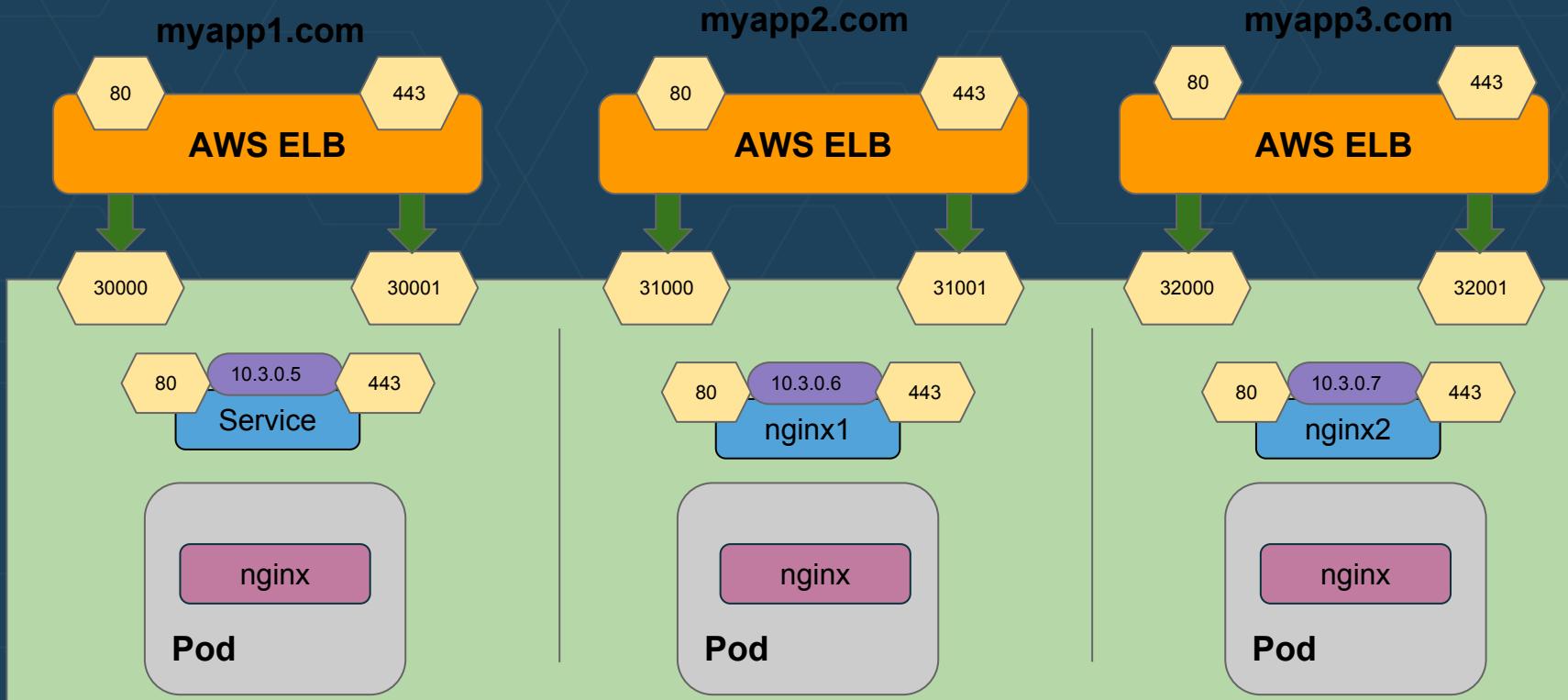


# LoadBalancer

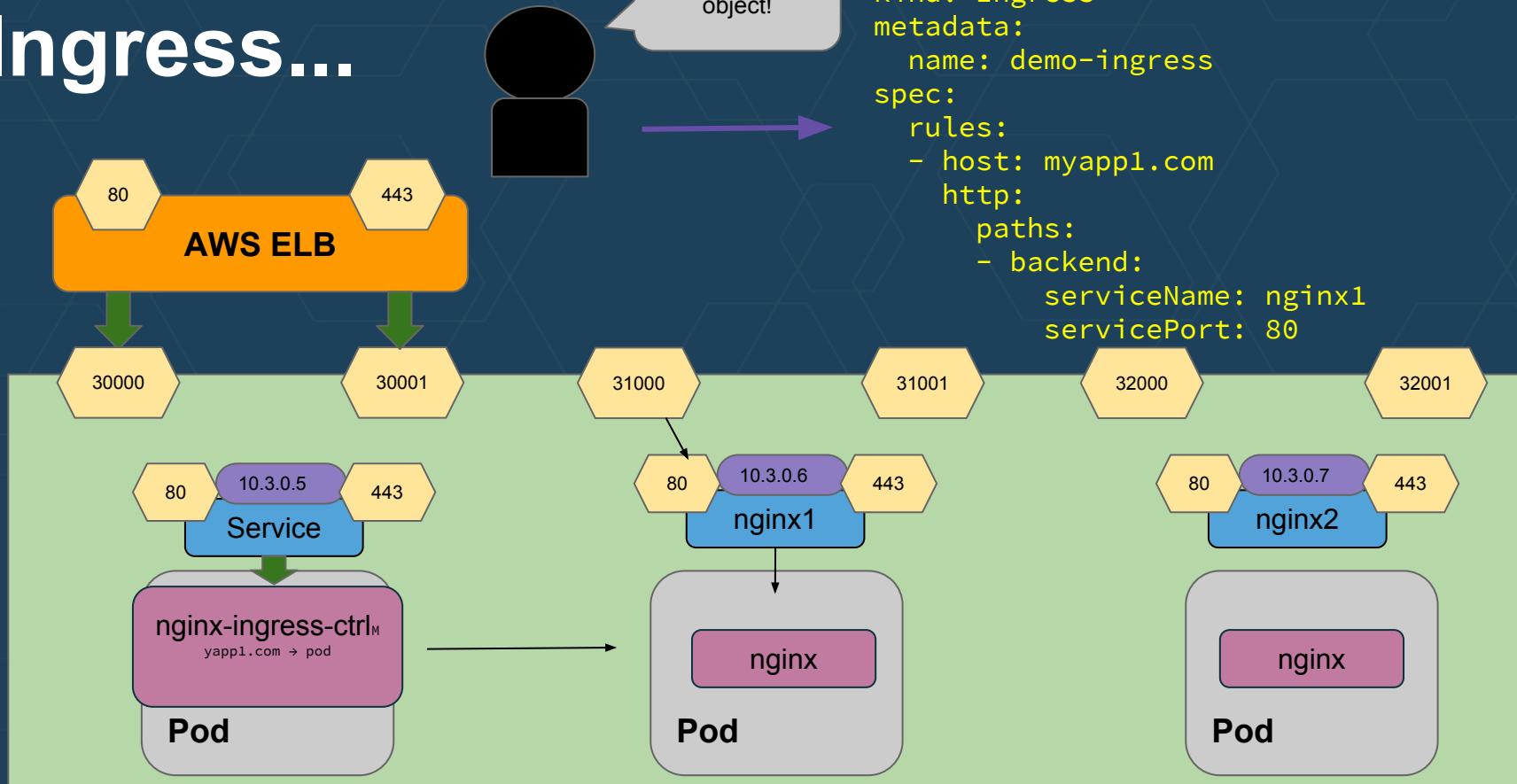
```
Kubectl create -f deployment.yaml  
Kubectl expose... --port 443 --port  
443
```



# Consider this...



# Ingress...



# Ingress

- Logical mapping of a hostname/path to service.
- SSL termination, rules on L7 (vs L4 with regular K8s service)
- Traffic Routing Rules.

# Ingress

- Path-based (L7)
- TCP/UDP-based (L4)
- Affinity/Sticky Sessions
- Multi-TLS (terminate SSL for multiple hostnames)

# BYO Ingress Controller - 5 Easy Steps

1. Create Backend Deployment.
2. Expose Backend as a Service.
3. Create Ingress Controller Deployment.
4. Expose Ingress Controller as a Service.
5. Create the Ingress Object.

# Exercise

## Ingress

# Exercise

BYO Ingress

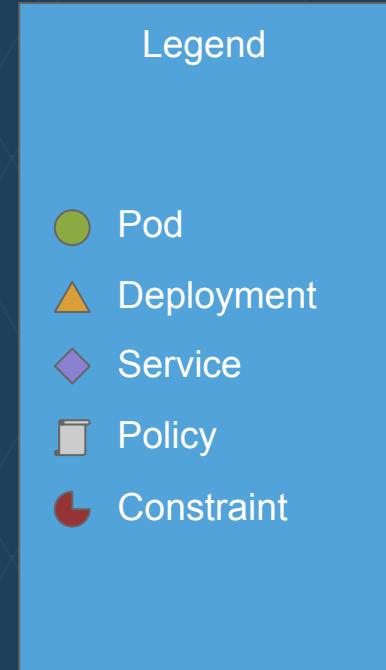
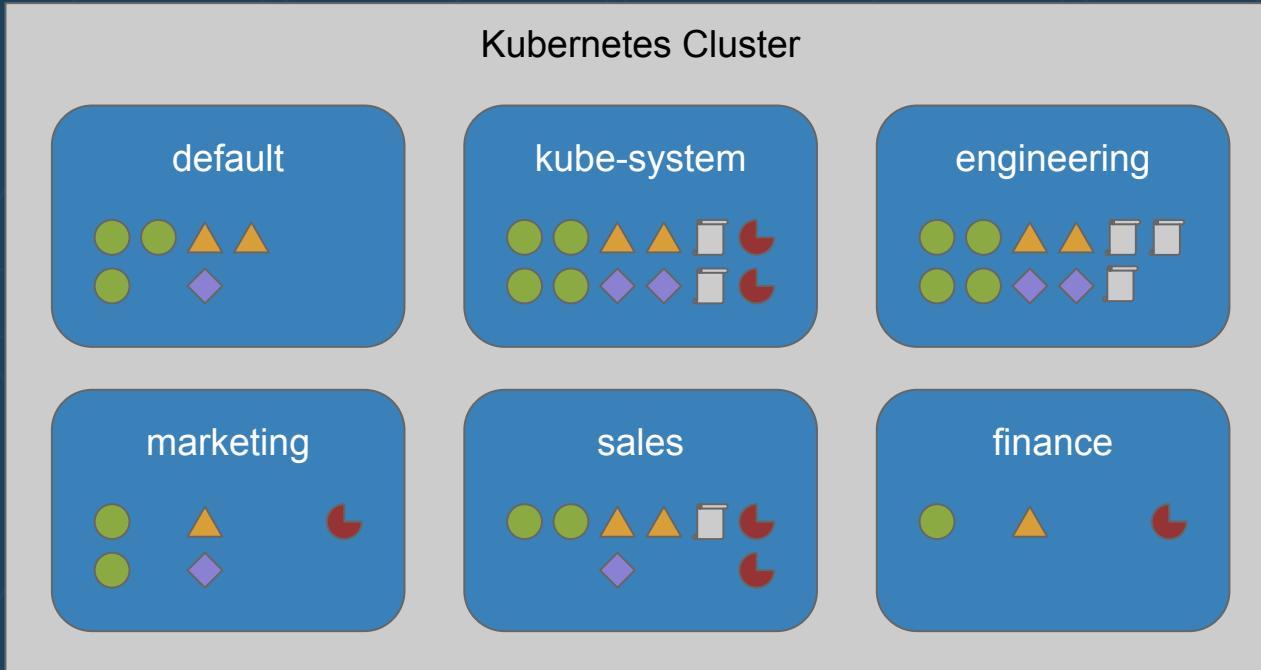
# Namespaces

Module 12

# Namespaces

- Allow grouping of cluster resources into logical groups.
- Divides a physical cluster into one or more virtual clusters
- Primarily designed for clusters with many users spread across multiple teams or projects

# Namespaces Example



# Namespaces

- Resource names must be unique
- Namespaces provide a scope for resources to prevent name collisions
- Resource Quotas allow sharing of cluster resources across teams by Namespace

# Namespaces

- The **default** namespace is used for all resources, unless specified otherwise
- The **kube-system** namespace is used for resources created by the Kubernetes system
- Additional namespaces can be defined as needed

# Exercise

## Namespaces

# Contexts

# Anatomy of a kubeconfig

Clusters

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: REDACTED
  server: https://192.168.56.60:6443
  name: my-first-cluster
contexts:
- context:
  cluster: my-first-cluster
  user: kubelet
  name: default
current-context: default
kind: Config
preferences: {}
users:
- name: kubelet
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

Context

Current

Users

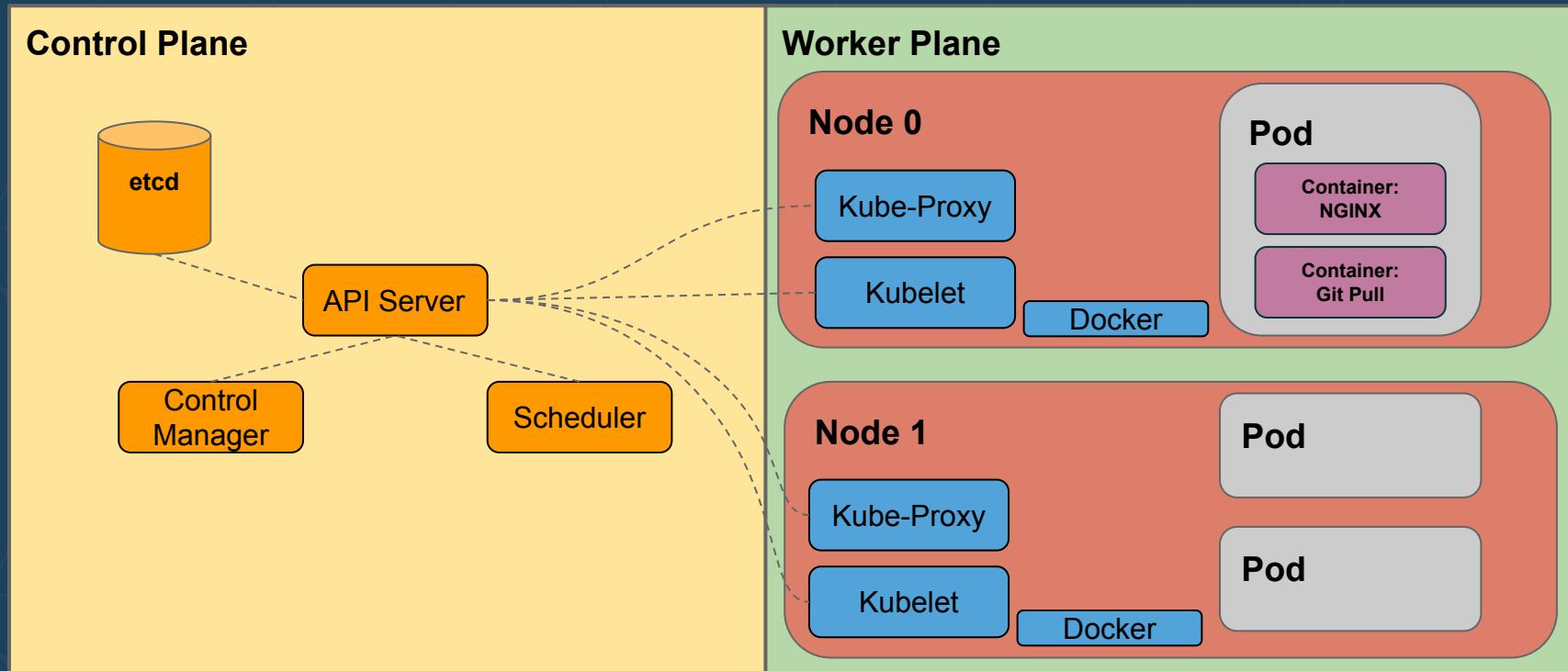
# Exercise

## Contexts

# Architecture Overview

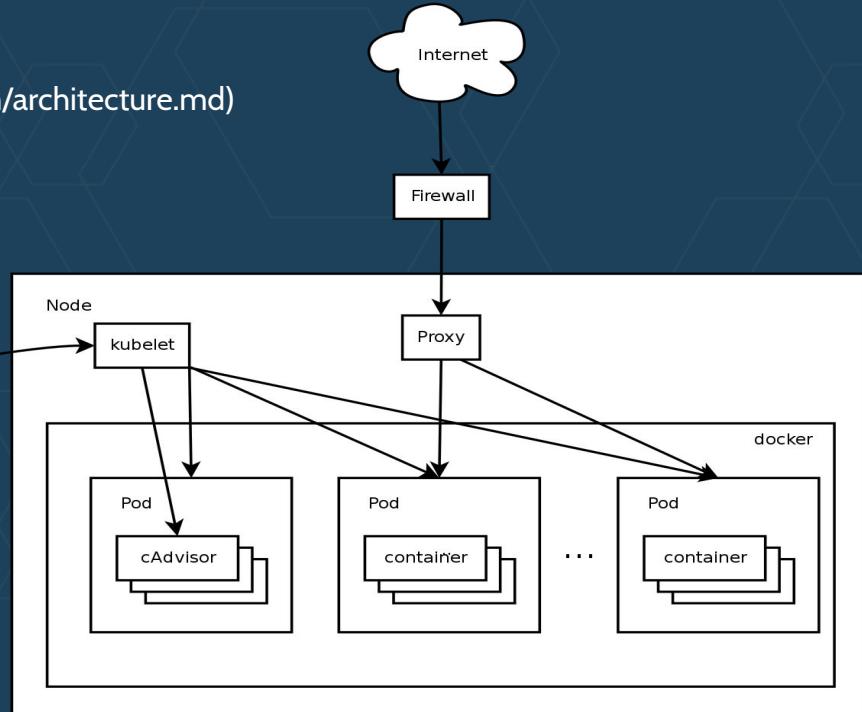
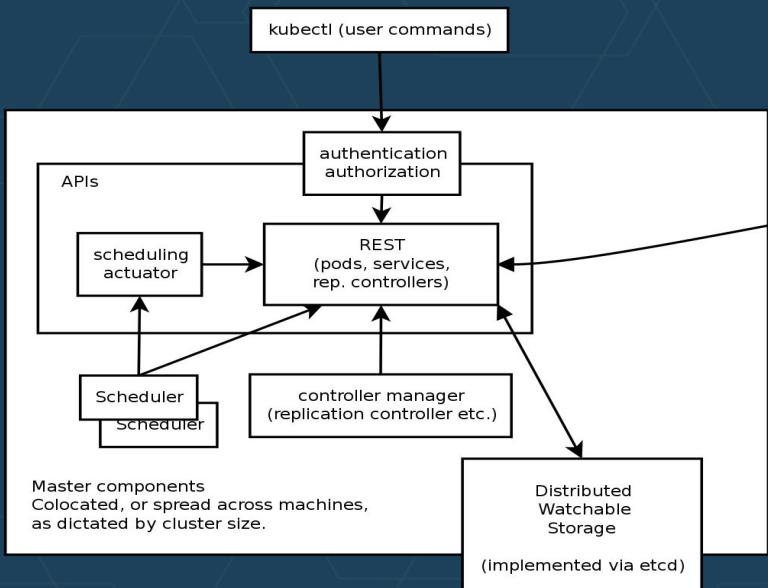
Module 13

# Kubernetes Architecture



# Kubernetes Architecture

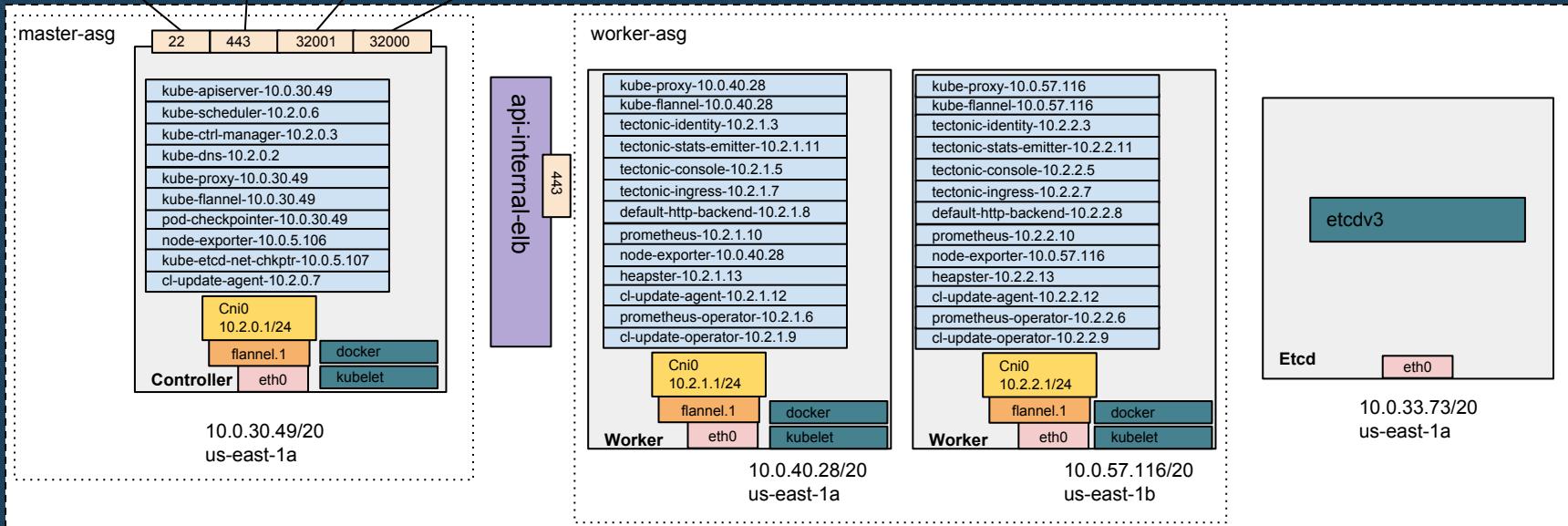
Image credit: adapted from the Kubernetes project  
(<https://github.com/kubernetes/kubernetes/blob/master/docs/design/architecture.md>)





- A persistent distributed key/value store
- Stores the state/objects of the Kubernetes cluster
- Includes the ability to watch for key changes
  - Enables the API Server to provide watch services to other components
- Three to seven node clusters are recommended for high availability
- For security and performance reasons, it is recommended not to use for your applications
  - Create a separate cluster for your applications instead
- Should be backed up regularly





10.0.32.1/20

10.0.16.1/20      10.0.48.1/20



# <kubelet reading local manifest>

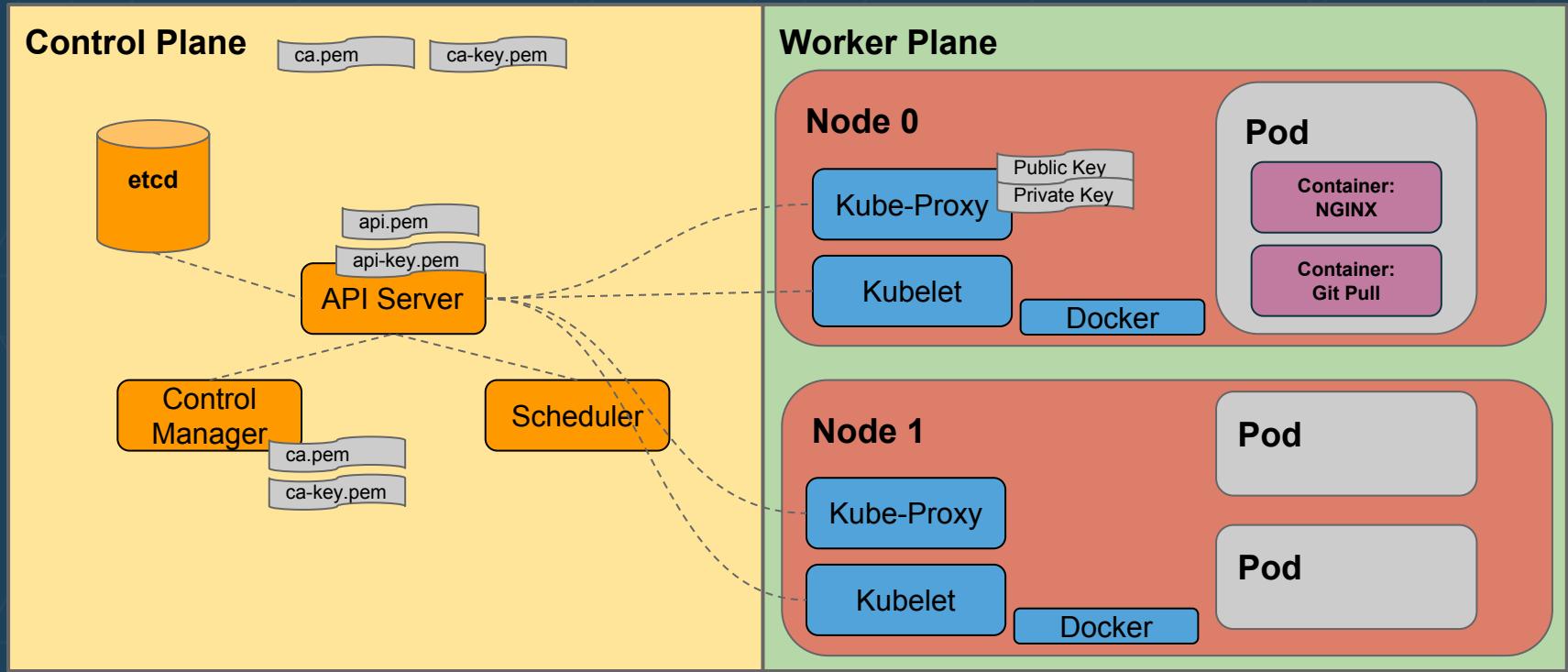
<setting up kube-api - running a  
pod submitted to the api>

<setting up kube-scheduler -  
watch scheduler modify manifest  
by selecting a node.>

# <setting up kube-controller-mgr>

# TLS Certificates in the Kubernetes Cluster

# PKI in the Kubernetes Cluster



# Providing CA Certificate and Key

1. Choose Cluster Type  
2. Define Cluster  
  
AWS Credentials  
Cluster Info  
**Certificate Authority**  
SSH Key  
Define Nodes  
Networking  
Console Login  
Submit  
3. Boot Cluster

**Certificate Authority** Save progress

A certificate authority (CA) is needed so we can generate certificates for cluster components.

Generate a CA certificate and key for me. (default)  
Component certificates will not be trusted in web browsers without additional configuration.

I'll provide a CA certificate and key in PEM format.  
Your CA will be used to issue certificates for cluster components.

**Upload CA Certificate**  
Paste your certificate here. It should start with:  
-----BEGIN CERTIFICATE-----  
  
It should end with:  
-----END CERTIFICATE-----

**Upload CA Private Key**  
Paste your private key here. It should start with:  
-----BEGIN RSA PRIVATE KEY-----  
  
It should end with:  
-----END RSA PRIVATE KEY-----

[Previous Step](#) [Next Step](#)

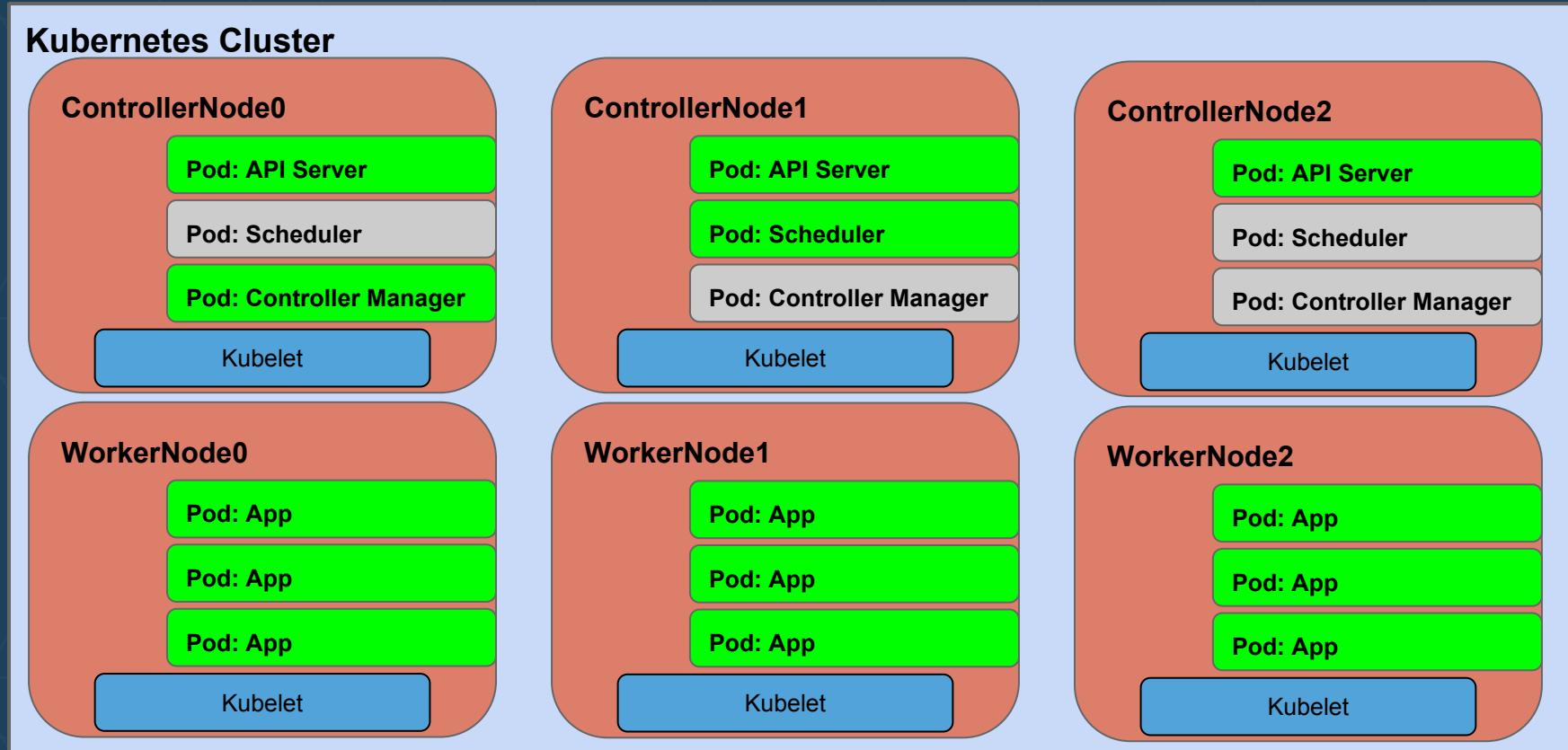


# Self-Hosted Infrastructure

# Self-Hosted Kubernetes

Active  
Passive/Standby

## Kubernetes Cluster



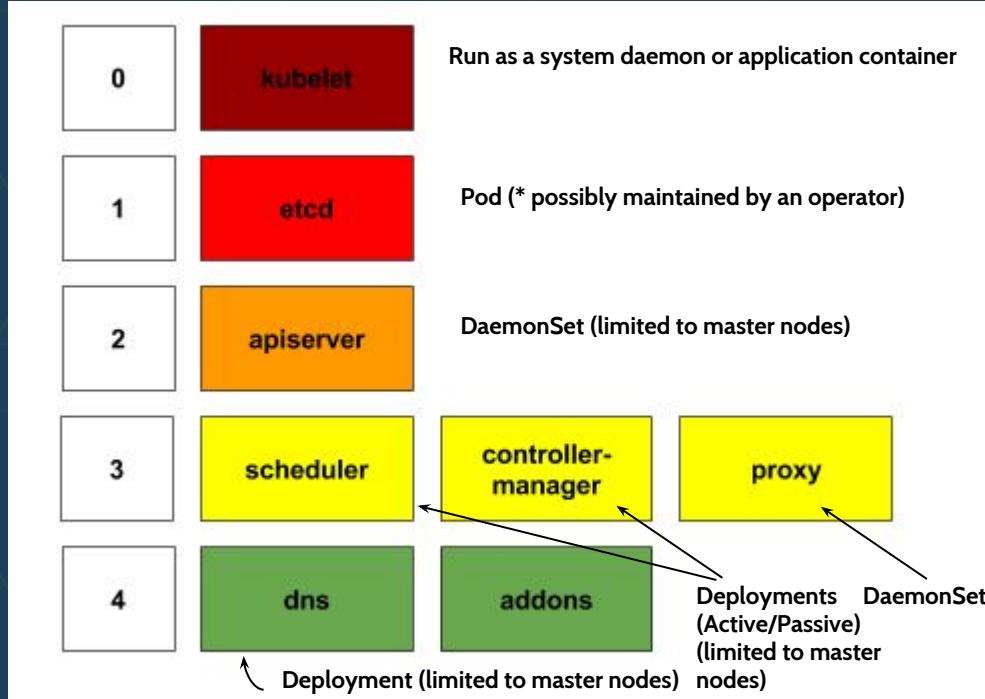
# Advantages of Self-Hosted

- **Small Dependencies**
  - Reduces the number of components required for a Kubernetes cluster to be deployed.
- **Deployment Consistency**
  - Reduces the number of moving parts. No need to worry about using config management tools or writing files to disk.
- **Introspection**
  - Components can be debugged via Kubernetes API (i.e. kubectl logs)
- **Cluster Upgrades**
  - Easy upgrades via API.
- **HA Configurations**
  - Easier to scale up and monitor HA environment without external tooling.

Original Proposal: <https://github.com/kubernetes/kubernetes/issues/246#issuecomment-64533959>

Updated Proposal: <https://github.com/kubernetes/community/blob/master/contributors/design-proposals/self-hosted-kubernetes.md>

# Self-Hosting Layers 1-4



# Self-Hosting Layers 1-4

```
kubectl get pods -n kube-system -l tier=control-plane
```

NAME	READY	STATUS	RESTARTS	AGE
kube-apiserver-c4fvk	1/1	Running	4	1d
kube-controller-manager-472801289-65wcv	1/1	Running	0	1d
kube-controller-manager-472801289-7rt18	1/1	Running	1	1d
kube-etcd-network-checkpointer-qr6dh	1/1	Running	0	1d
kube-scheduler-1310662694-15k6q	1/1	Running	0	1d
kube-scheduler-1310662694-wr83q	1/1	Running	1	1d
pod-checkpointer-k1mnp	1/1	Running	0	1d
pod-checkpointer-k1mnp-ip-10-0-20-162.us-west-1.compute.internal	1/1	Running	0	1d

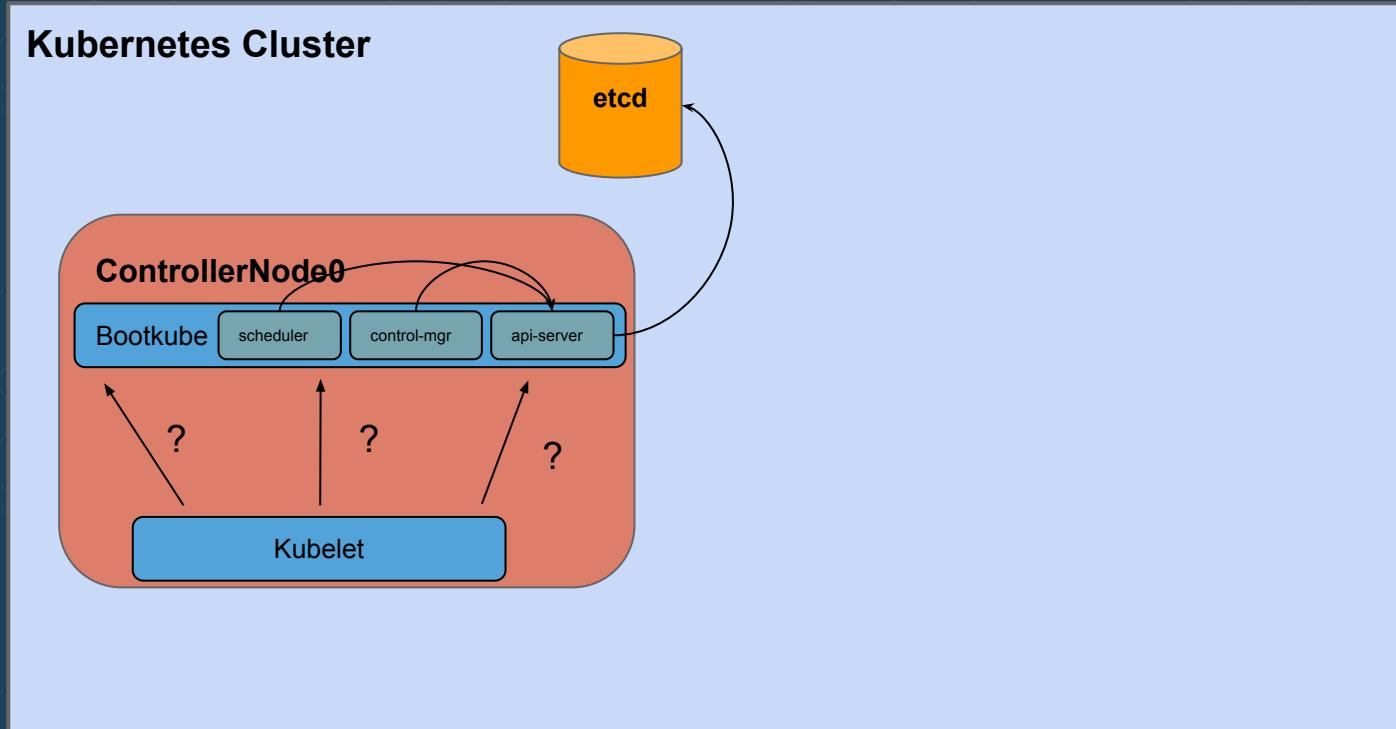
# How do we bootstrap a self-hosted system?

# Bootkube!

<https://github.com/kubernetes-incubator/bootkube>

# How does it work?

# A kubelet is born... And looks for an K8s API server....



# <bootkube-demo>

core@localhost:~

```
Every 0.1s: docker ps                               Mon Jun 26 15:52:14 2017
CONTAINER ID        IMAGE               COMMAND      CREATED
STATUS              PORTS              NAMES

```

core@localhost:~/cluster

```
ubelet/kubelet.go:390: Failed to list *v1.Node: Get https://172.17.4.100:443/api/v1/nodes?fieldSelector=metadata.name%3D172.17.4.100&resourceVersion=0: dial tcp 172.17.4.100:443: getsockopt: connection refused
Jun 26 15:52:14 localhost kubelet-wrapper[1377]: E0626 15:52:14.075990 1377 reflector.go:190] k8s.io/kubernetes/pkg/kubelet/config/apiserver.go:46: Failed to list *v1.Pod: Get https://172.17.4.100:443/api/v1/pods?fieldSelector=spec.nodeName%3D172.17.4.100&resourceVersion=0: dial tcp 172.17.4.100:443: getsockopt: connection refused
Jun 26 15:52:14 localhost kubelet-wrapper[1377]: E0626 15:52:14.143917 1377 reflector.go:190] k8s.io/kubernetes/pkg/kubelet/kubelet.go:382: Failed to list *v1.Service: Get https://172.17.4.100:443/api/v1/services?resourceVersion=0: dial tcp 172.17.4.100:443: getsockopt: connection refused
Jun 26 15:52:14 localhost kubelet-wrapper[1377]: W0626 15:52:14.542177 1377 cni.go:157] Unable to update cni config: No networks found in /etc/kubernetes/cni/net.d
Jun 26 15:52:14 localhost kubelet-wrapper[1377]: E0626 15:52:14.542560 1377 kubelet.go:2067] Container runtime network not ready: NetworkReady=false reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config uninitialized
Jun 26 15:52:14 localhost kubelet-wrapper[1377]: I0626 15:52:14.607036 1377 kubelet_node_status.go:230] Setting node annotation to enable volume controller attach/detach
Jun 26 15:52:14 localhost kubelet-wrapper[1377]: I0626 15:52:14.608099 1377 kubelet_node_status.go:77] Attempting to register node 172.17.4.100
Jun 26 15:52:14 localhost kubelet-wrapper[1377]: E0626 15:52:14.609251 1377 kubelet_node_status.go:101] Unable to register node "172.17.4.100" with API server: Post https://172.17.4.100:443/api/v1/nodes: dial tcp 172.17.4.100:443: getsockopt: connection refused
Jun 26 15:52:14 localhost kubelet-wrapper[1377]: E0626 15:52:14.762998 1377 reflector.go:190] k8s.io/kubernetes/pkg/kubelet/kubelet.go:390: Failed to list *v1.Node: Get https://172.17.4.100:443/api/v1/nodes?fieldSelector=metadata.name%3D172.17.4.100&resourceVersion=0: dial tcp 172.17.4.100:443: getsockopt: connection refused
```

root@localhost:~

```
mdmpro:single-node madorn$ vagrant ssh
Last login: Mon Jun 26 15:41:33 UTC 2017 from 10.0.2.2 on pts/2
Container Linux by CoreOS stable (1298.6.0)
core@localhost ~ $ 
core@localhost ~ $ sudo su -
localhost ~ #
localhost ~ # ls
localhost ~ # tail -f /home/core/bootkube.log
tail: cannot open '/home/core/bootkube.log' for reading: No such file or directory
tail: no files remaining
localhost ~ # tail -f /home/core/bootkube.log
tail: cannot open '/home/core/bootkube.log' for reading: No such file or directory
tail: no files remaining
localhost ~ # tail -f /home/core/bootkube.log
tail: cannot open '/home/core/bootkube.log' for reading: No such file or directory
tail: no files remaining
localhost ~ # cd ~
```

root@localhost:~

```
localhost ~ # cd /home/core/cluster/
localhost cluster # ls
auth bootstrap-manifests manifests tls user-data
localhost cluster # cd auth/
localhost auth # ls
kubeconfig
localhost auth # ps aux | grep etcd
1416 ? Ssl 0:07 /usr/local/bin/etcd
4084 pts/1 S+ 0:00                                     \_ grep --colour=auto etcd
localhost auth # vim /etc/etcd/etcd
etcd/ ethertypes
localhost auth # vim /etc/etcd/tls/etcd-
etcd-ca.crt etcd-client.crt etcd-client.key etcd-peer.crt etcd-peer.key
localhost auth # vim /etc/etcd/tls/etcd-
etcd-ca.crt etcd-client.crt etcd-client.key etcd-peer.crt etcd-peer.key
localhost auth # vim /etc/etcd/tls/etcd-
etcd-ca.crt etcd-client.crt etcd-client.key etcd-peer.crt etcd-peer.key
localhost auth # cd ~
localhost ~ # ls
localhost ~ # [ ]
```

# </bootkube-demo>

# Cluster Control Plane

Module 14

# Control Plane

- The following components make up the Kubernetes control plane
  - API Server
  - Scheduler
  - Controller Manager
- These components are often deployed on the same machine
- Stateless- should be clustered for high availability

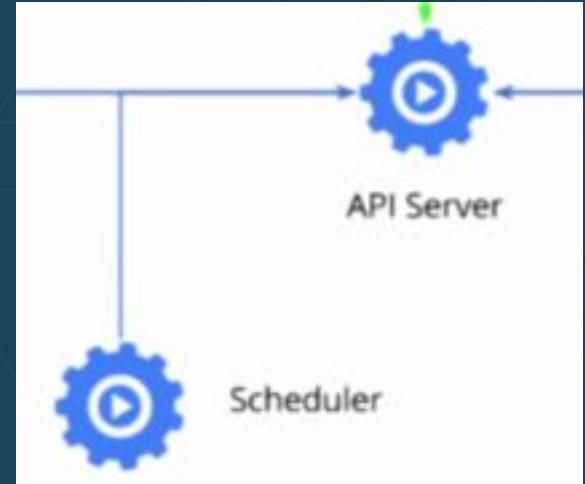
# API Server



- Validates and configures data for Kubernetes objects
- Provides a simple REST API
- All clients and Kubernetes components talk via the API Server
  - Strictly modular- there are no back channels for communication
- Stateless- uses a distributed store (etcd) for state
  - The API Server is the only component that talks to etcd directly

# Scheduler

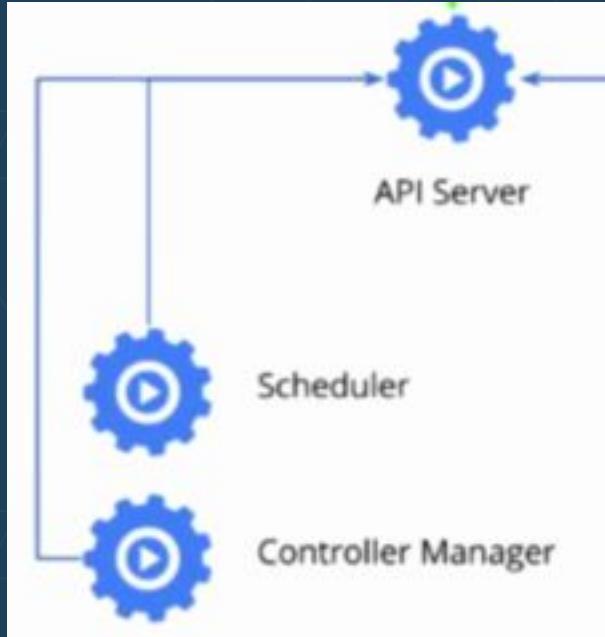
- Interfaces with the API Server
- Watches for pod objects with an empty PodSpec.NodeName
- Decides which node to run the pod and posts this information to the pod object
- Extensible- can choose or add scheduling policies



<https://kubernetes.io/docs/admin/kube-scheduler/>

# Controller Manager

- A daemon that includes the controllers of Kubernetes
  - Node controller, deployment controller, ReplicaSet, etc.
- Controller- a control loop that watches the API server
  - Continuously moves the actual state toward the desired state



<https://kubernetes.io/docs/admin/kube-controller-manager/>

# Exercise

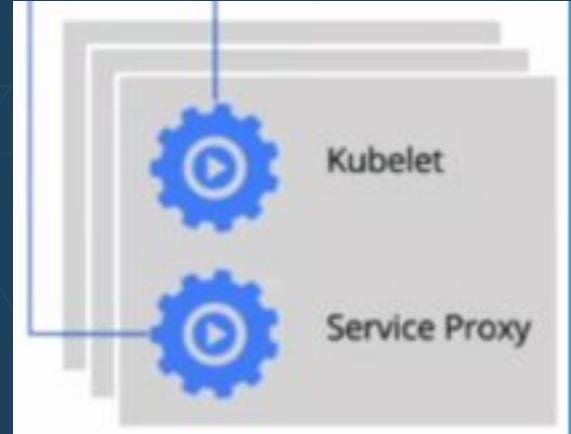
## Node Management

# Worker Nodes

Module 15

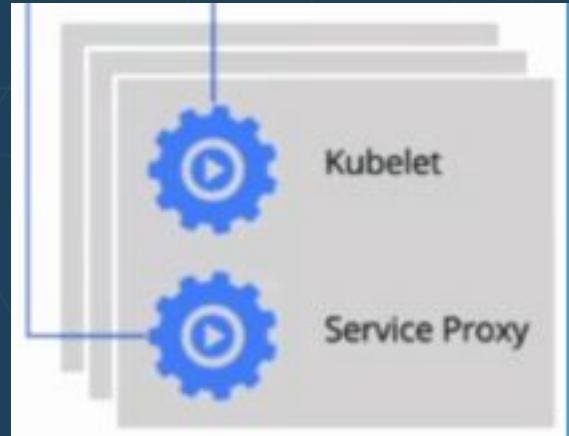
# Kubelet

- Each worker node has a “node agent” process called a Kubelet
- Watches the API Server and local files for PodSpecs assigned to the node
- Ensures that containers of the pod are running and healthy
- Mounts pod volumes



# Service Proxy (kube-proxy)

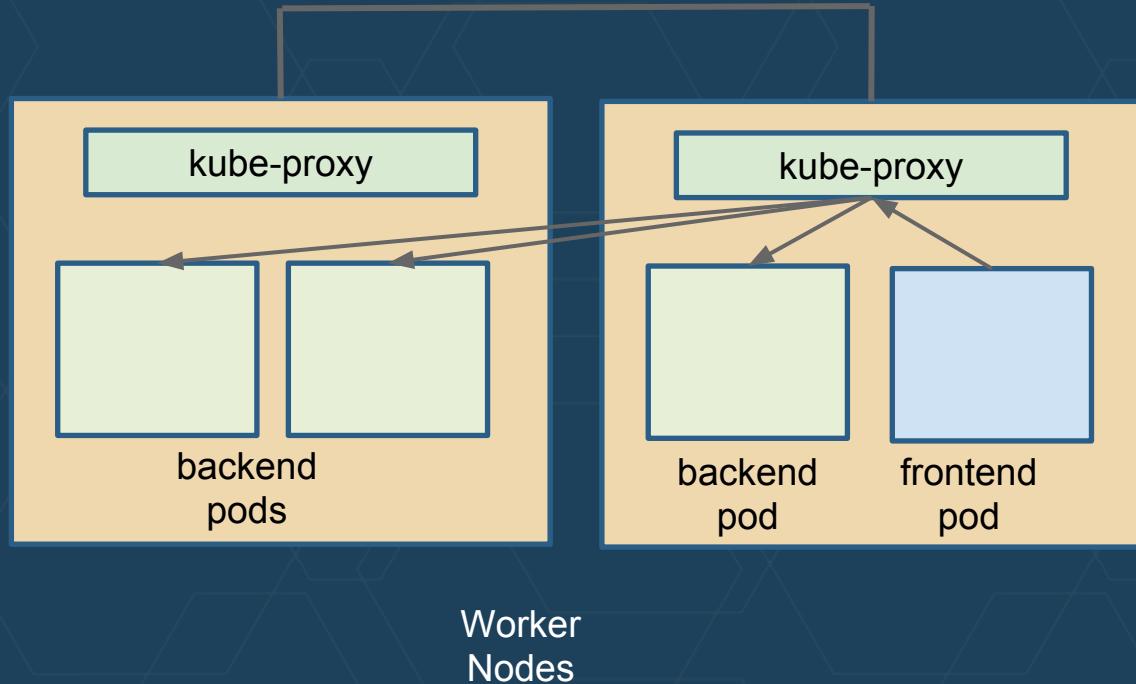
- Reflect the services defined by the Kubernetes API on each node
- Can do TCP and UDP stream forwarding or round robin TCP and UDP forwarding
- The proxy is configured by creating services
  - Services provide clients a consistent IP/DNS name for a (dynamic) set of pods



<https://kubernetes.io/docs/admin/kube-proxy/>

# Kube-proxy Internal Service Discovery (1 of 2)

- Containers in pods can use other services on the cluster
- Each worker node has iptables for the services, allowing for load-balancing
- The service call never has to leave the cluster



# Kube-proxy Internal Service Discovery (2 of 2)

- A container can use another service in the cluster (without leaving the cluster):
  - DNS lookup- provided by Kubernetes DNS service (kube-dns)
    - `curl anotherservice`- returns the response for the service
  - Environment variables- a container in a pod containers connection information for other services in the cluster
    - `ANOTHERSERVICE_SERVICE_PORT=80`
    - `ANOTHERSERVICE_SERVICE_HOST=10.3.0.33`
    - `curl 10.3.0.33:80`- returns the response for the service

# Adding Additional Worker Nodes

# Exercise

Add Worker Node

# Add-Ons

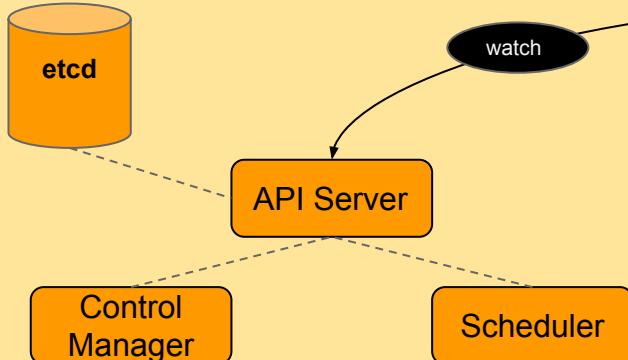
Module 16

# DNS

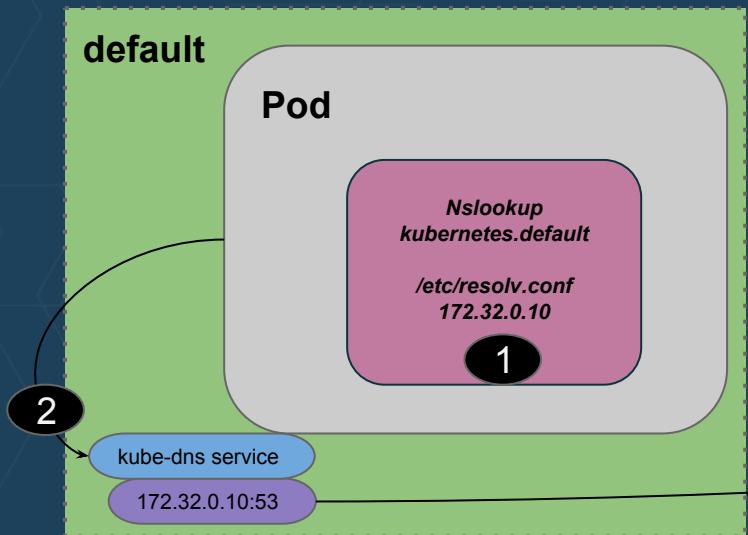
# Kubernetes DNS

- Schedules a DNS Pod and Service on the cluster.
- Configures the kubelets to tell individual containers to use the DNS Service's IP to resolve DNS names.

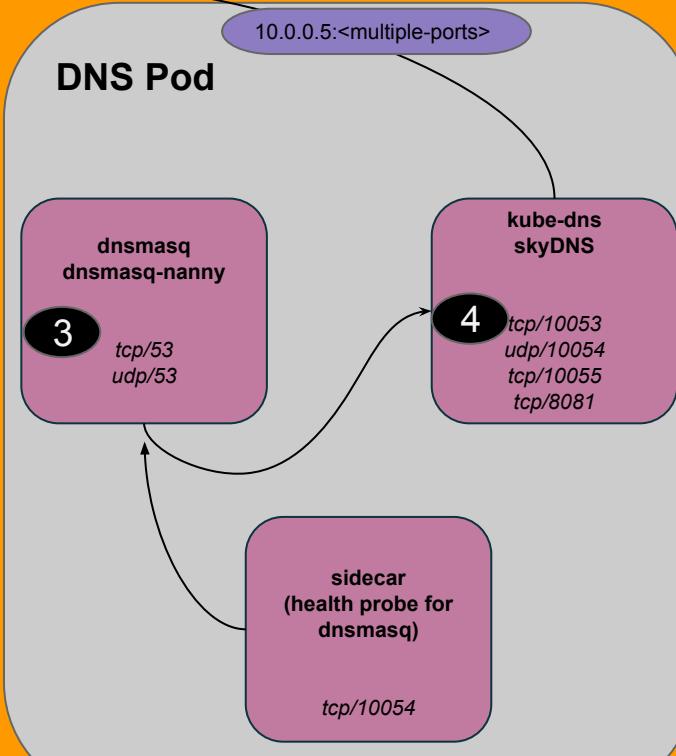
## Control Plane



## default



## kube-system



# Exercise

DNS Service

# Networking

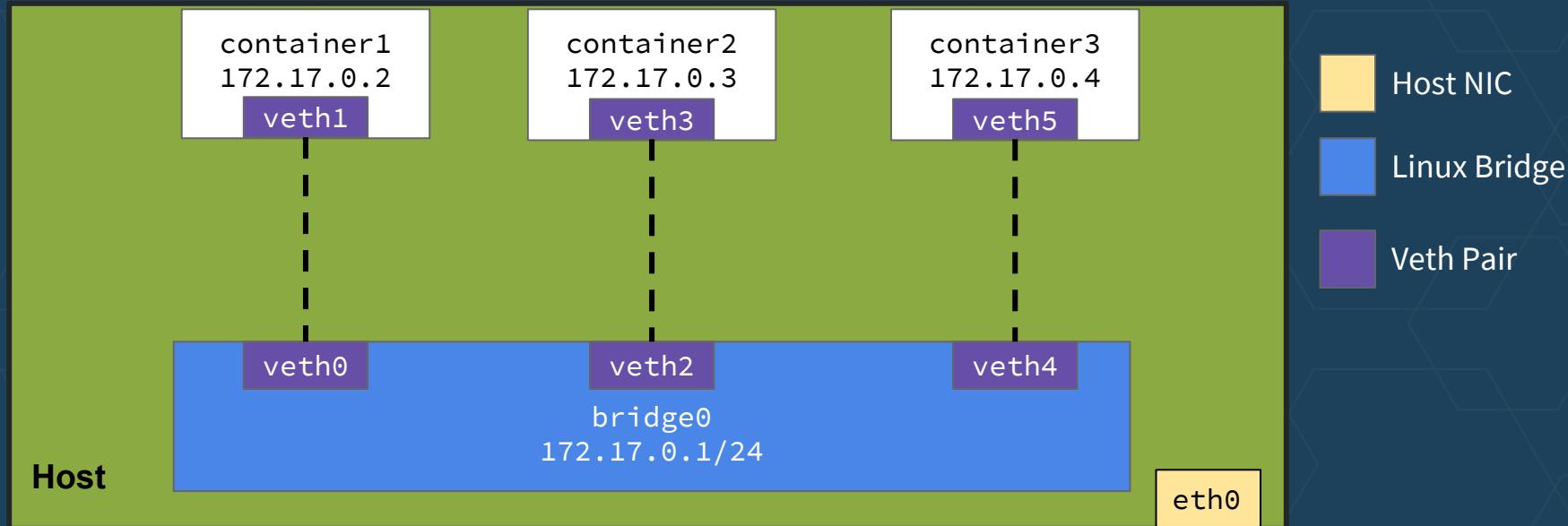
Module 17

# Kubernetes Networking

# Kubernetes Networking Fundamentals

- Pods can communicate with other pods, regardless of which node they land on.
- Each Pod gets it's own IP address.

# Traditional Container Networking



```
$ ip address  
-lo  
-eth0  
-bridge0  
-veth0  
-veth2  
-veth4
```

```
$ brctl show  
Bridge          Interfaces  
bridge0         veth0  
                veth2  
                veth4
```

# Traditional Container Networking: Issues

- Containers can talk to other containers only if they are on the same machine (virtual bridge).
- Containers on different machines can not reach each other (may end up with the exact same network ranges and IP addresses).

# Container Networking 101

# In the beginning...Host Networking!

```
docker run -it -d --net=host nginx:latest
```

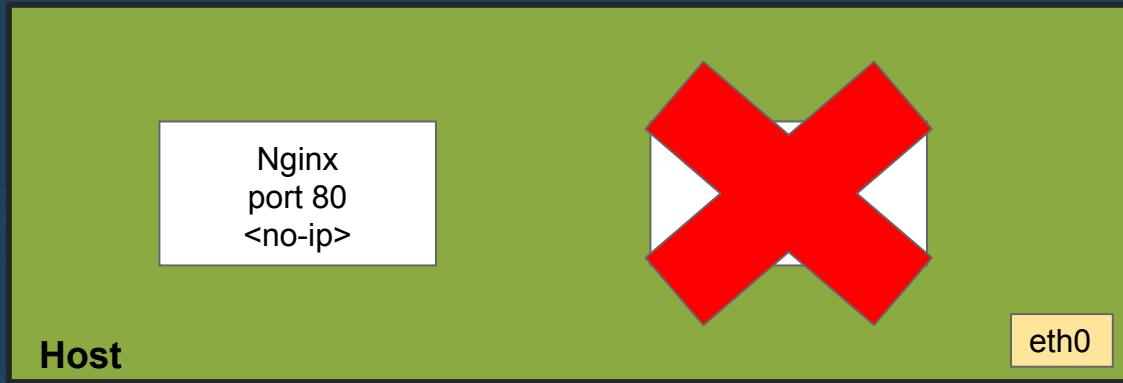


```
$ ss -ntlp
LISTEN *:80 nginx

$ curl localhost:80
WELCOME TO NGINX
```

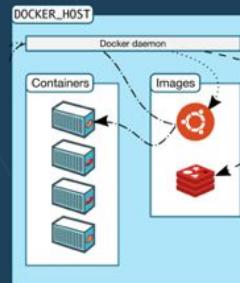
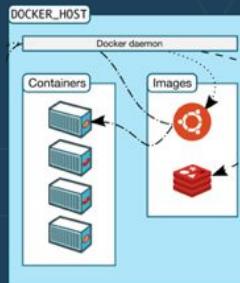
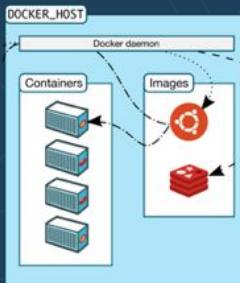
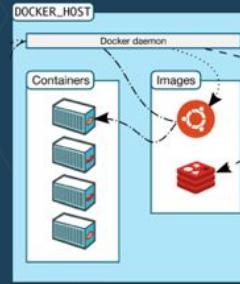
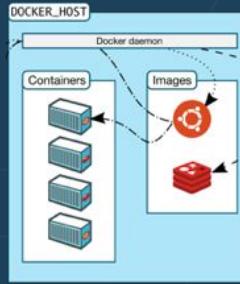
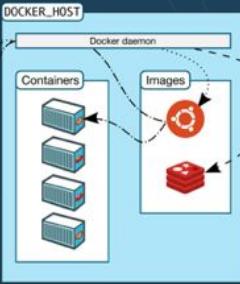
# Multiple Containers? Port Conflicts!

```
docker run -it -d --net=host nginx:latest
```



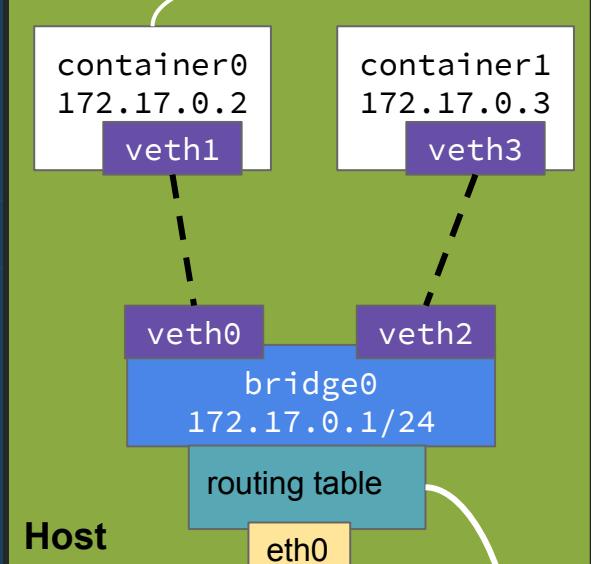
```
$ ss -ntlp  
LISTEN *:80 nginx  
  
$ curl localhost:80  
WELCOME TO NGINX
```

# Multiple Container Hosts Require a Container Networking Solution



# Solution #1 Host-Gateway

172.17.0.2 → 172.17.2.2



172.17.0.2 → 172.17.2.2

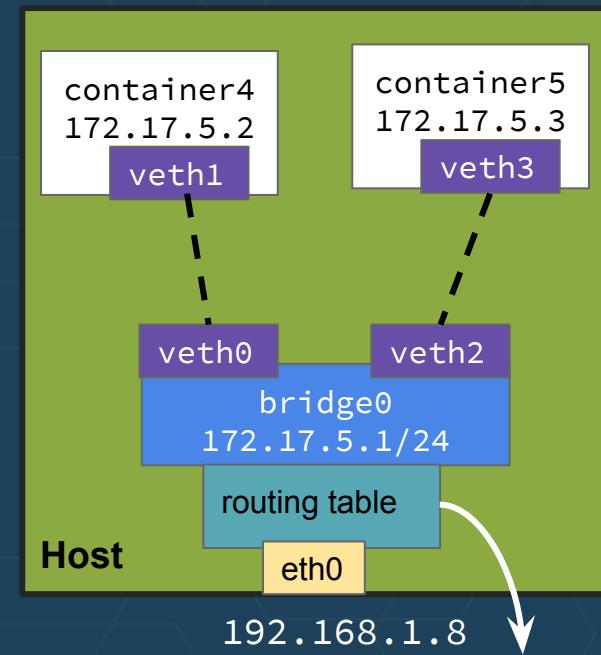
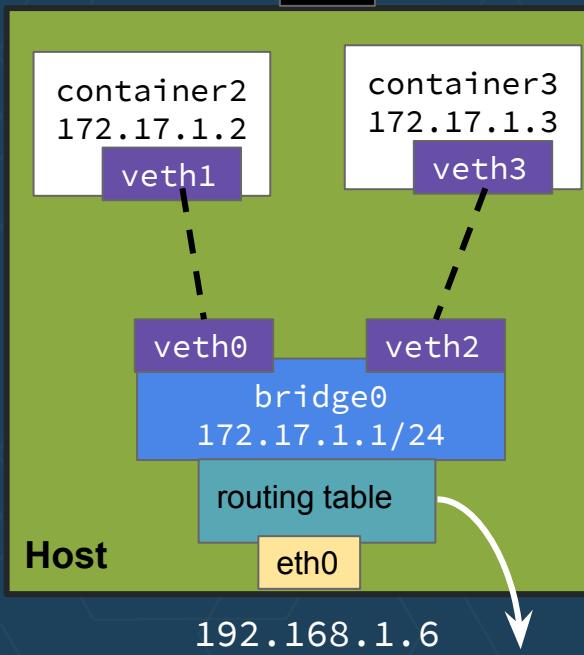
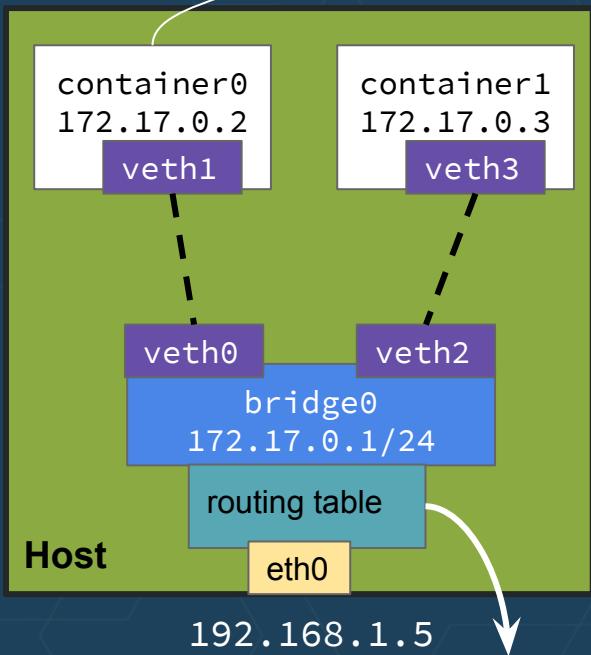
172.17.1.0/24 dev bridge0 via 192.168.1.6 dev eth0  
172.17.2.0/24 dev bridge0 via 192.168.1.7 dev eth0

# Works Great...but....

# Problem #1 with Traditional Host-Gateway

172.17.0.2 → 172.17.5.3

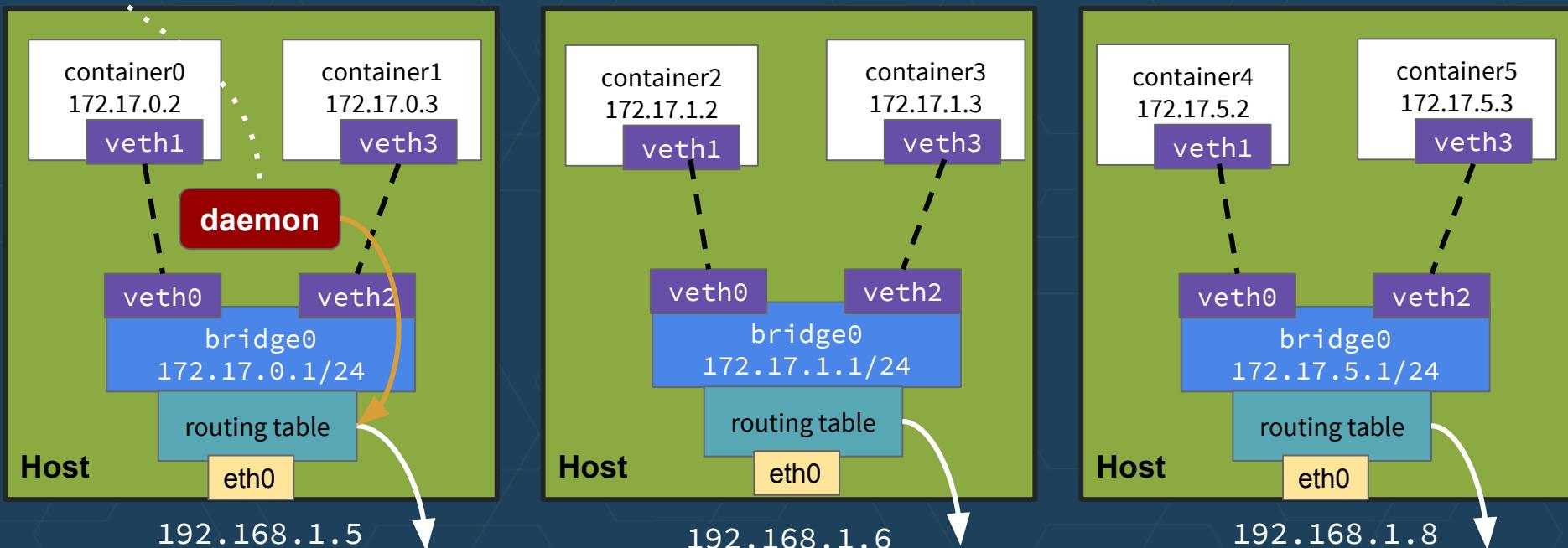
I don't want to manually  
configure routes on every  
node!!!!



# Source of Truth for Networking?



# Consider a process fetching host routes to inject into a node's routing table.



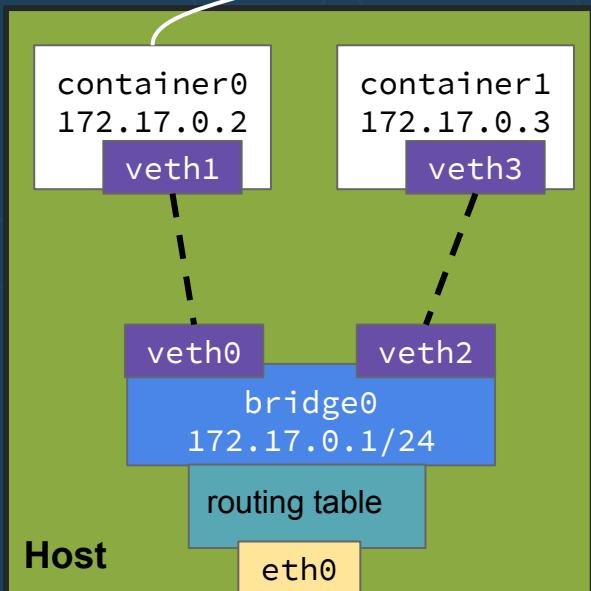
172.17.5.1/24 dev bridge0 via 192.168.1.8 dev eth0

# Problem 1 solved.

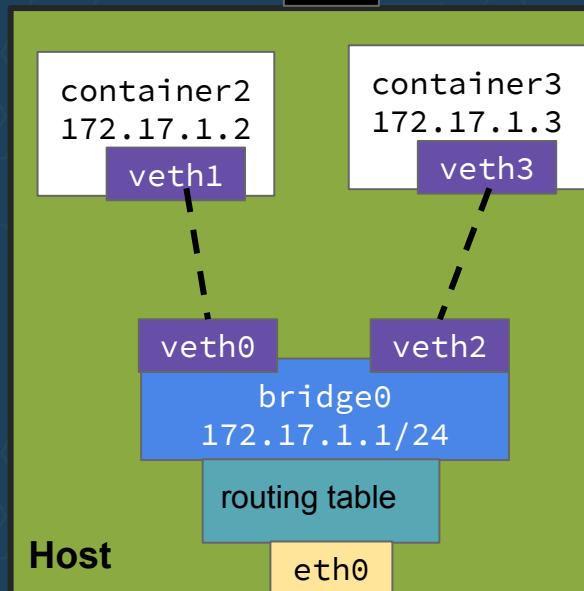
# Problem #2 with Traditional Host-Gateway

172.17.0.2 → 172.17.5.3

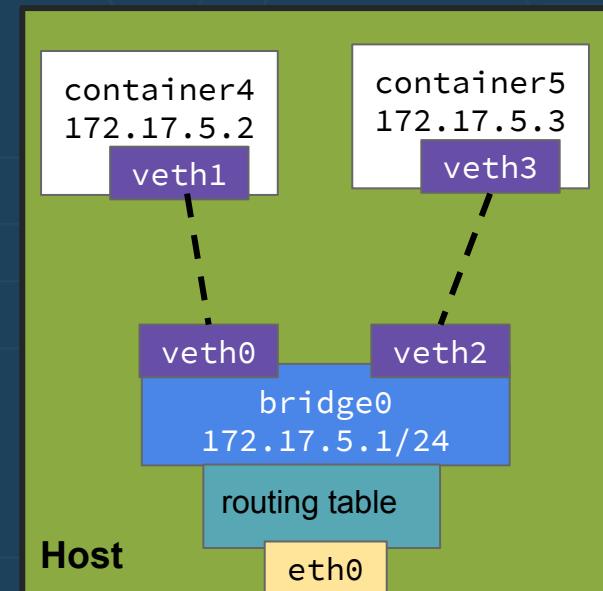
All my nodes need to  
be on the same  
subnet!!!



192.168.1.5



192.168.1.6



192.168.1.8

192.168.2.8

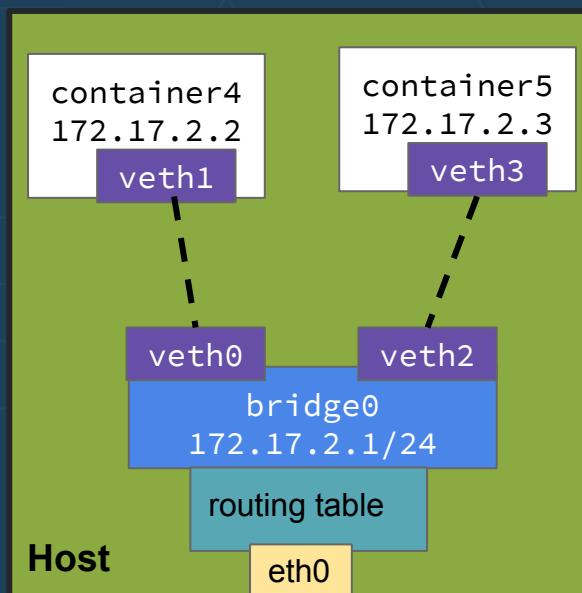
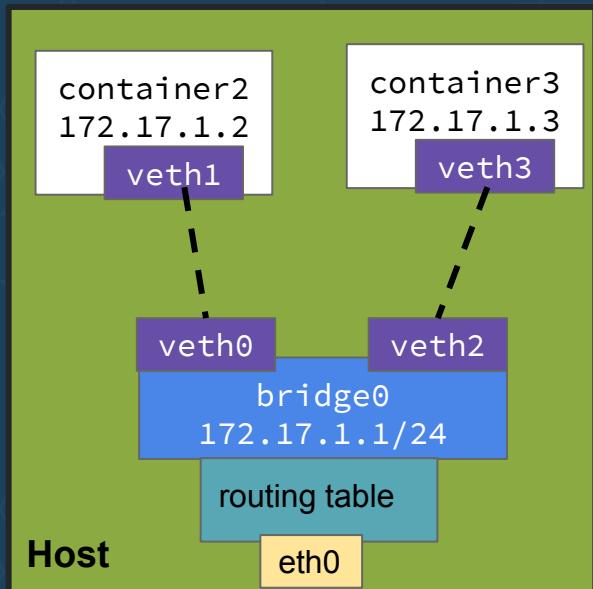
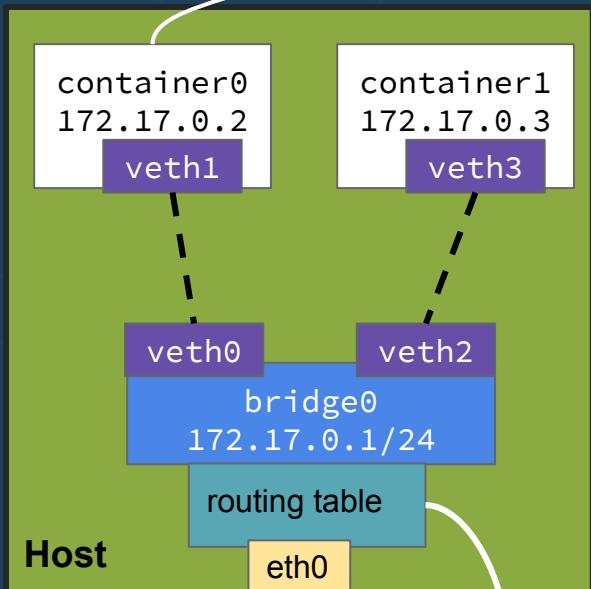
# Router with Static Routes

# Router with Static Routes



172.17.0.2 → 172.17.2.2

172.17.0.2 → 172.17.2.2



default via 192.168.1.1 dev eth0 proto static metric 100

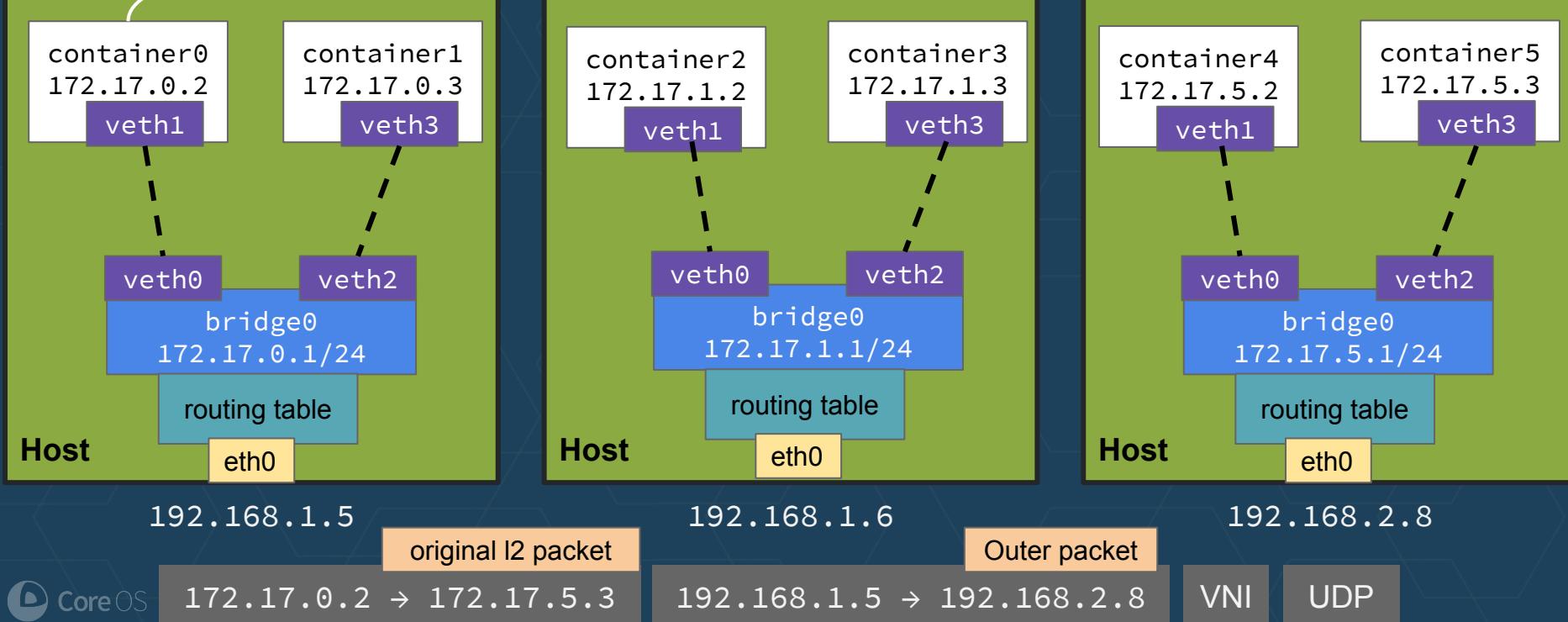
172.17.5.1/24 dev bridge0 via 192.168.2.8 dev

daemon

# Overlay Network (VXLAN)

# Overlay Network (VXLAN)

172.17.0.2 → 172.17.5.3



CoreOS

172.17.0.2 → 172.17.5.3

192.168.1.5 → 192.168.2.8

VNI      UDP





# flannel

# flannel

flannel is a network fabric for containers,  
designed for Kubernetes

# flannel

- Simple and easy to configure layer 3 network fabric designed for Kubernetes.
- Runs a small, single binary agent called **flanneld** on each host.
- Allocates a subnet lease to each host out of a larger, preconfigured address space.

# flannel

- Responsible for providing a layer 3 IPv4 network between multiple nodes in a cluster.
- Uses either the **Kubernetes API** or **etcd** directly to store the network configuration and data.
- Packets forwarded using one of several backend mechanisms including **VXLAN** and **cloud integrations**.

# flannel

- Does not control how containers are networked to the host, only how the traffic is transported between hosts.
- Provides a **CNI** plugin for Kubernetes and integration with Docker.

# flannel Steps

1. Allocate new subnet to host through **etcd**.
2. Create a virtual bridge on host ( `docker0` ).
3. Set up packet forwarding through backend.
  - a. **host-gw**: Use remote machine IPs
  - b. **vxlan**: Create virtual VXLAN interface
  - c. other (aws-vpc, gce, ali-vpc, etc...)

# Networking Plugins



# Container Network Interface (CNI)

- Cloud Native Computing Foundation (CNCF) project.
- Specifications and libraries for writing plugins to configure network interfaces in Linux containers.
- Concerned only with container network connectivity and removing allocated resources when container deleted.

# Container Network Interface (CNI)

- Containers can join multiple networks.
- Network described by JSON configuration.
- CNI Plugins support two actions:
  - Add container to the network
  - Remove container from the network

# Container Runtime (e.g. docker, rkt)

veth

macvlan

ipvlan

OVS

# Container Runtime (e.g. docker, rkt)

## Container Networking Interface (CNI)

veth

macvlan

ipvlan

OVS

# User configures a network

```
$ cat /etc/rkt/net.d/10-mynet.conf
{
  "name": "mynet",
  "type": "bridge",
  "ipam": {
    "type": "host-local",
    "subnet": "10.10.0.0/16"
  }
}
```

# CNI: Step 1

Container runtime creates network namespace and gives it a named handle.

```
$ cd /run  
$ touch myns  
$ unshare -n mount --bind /proc/self/ns/net myns
```

# CNI: Step 2

Container runtime invokes the CNI plugin

```
$ export CNI_COMMAND=ADD  
$ export CNI_NETNS=/run/myns  
$ export CNI_CONTAINERID=5248e9f8-3c91-11e5-...  
$ export CNI_IFNAME=eth0  
  
$ $CNI_PATH/bridge </etc/rkt/net.d/10-mynet.conf
```

# CNI: Step 3

Inside the bridge plugin (1):

```
$ brctl addbr mynet  
$ ip link add veth123 type veth peer name $CNI_IFNAME  
$ brctl addif mynet veth123  
$ ip link set $CNI_IFNAME netns $CNI_IFNAME  
$ ip link set veth123 up
```

# CNI: Step 3

Inside the bridge plugin (2):

```
$ IPAM_PLUGIN=host-local # from network conf
$ echo $IPAM_PLUGIN
{
  "ip4": {
    "ip": "10.10.5.9/16",
    "gateway": "10.10.0.1"
  }
}
```

# CNI: Step 3

Inside the bridge plugin (3):

```
# switch to container namespace  
  
$ ip addr add 10.0.5.9/16 dev $CNI_IFNAME  
  
# Finally, print IPAM result JSON to stdout
```

# flannel CNI plugin

- reads in /run/flannel/subnet.env
- writes out "bridge" + "host-local" conf
- calls out to "bridge"

# flannel CNI plugin

`/run/flannel/subnet.env`

```
FLANNEL_NETWORK=10.1.0.0/16
FLANNEL_SUBNET=10.1.17.1/24
FLANNEL_MTU=1472
FLANNEL_IPMASQ=true
```

CNI Config

```
{
  "name": "mynet",
  "type": "flannel"
}
```

# Kubernetes + CNI + Docker

- k8s starts "pause" container to create netns
- k8s invokes CNI driver
- CNI executes a CNI plugin
- CNI plugin joins "pause" container to network
- Pod containers use "pause" container netns

# Exercise

## Networking

# Network Policies

# Network Policies

- Specification of how groups of pods are allowed to communicate with each other and other network endpoints.
- Kubernetes resource that uses labels to select pods and define rules which specify what traffic is allowed to the selected pods.

# Pod Isolation

- By default, pods are non-isolated and accept traffic from any source.
- Pods become isolated by having a NetworkPolicy that selects them.
- Selected pods will reject any connections that are not allowed by a NetworkPolicy.

# Network Policy Examples

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
  ports:
  - protocol: TCP
    port: 6379
```

Sample Application Network Policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector:
```

Sample Default Policy

# Exercise

## Network Policies

# Canary Deployments

Module 20

# Canary Deployments

original deployment:

```
labels:  
  app: guestbook  
  tier: frontend  
  track: stable
```

canary deployment:

```
labels:  
  app: guestbook  
  tier: frontend  
  track: canary
```

service:

```
selector:  
  app: guestbook  
  tier: frontend
```

- When updating an application, it is common to start with a single "canary" to handle a small fraction of the existing traffic
- If the canary is successful, a full update can proceed
- Create a new deployment for the canary, with unique labels
- The frontend service can span both deployments by using a common label ("guestbook")

# Exercise

Canary Deployment

# Resource Allocation & Quotas

Module 21

# Requests & Limits

# Resource Types

- **CPU** - Measured in units of cores
- **1** Kubernetes cpu equivalent to:
  - **1 AWS vCPU**
  - **1 GCP Core**
  - **1 Azure vCore**
  - **1 bare-metal Hyperthread**

- **Memory** - Measured in units of bytes
- Express as integer or with suffix:
  - **E, P, T, G, M, K**
  - **Ei, Pi, Ti, Gi, Mi, Ki**
- The following represent roughly the same value:
  - **128974848, 129e6, 129M, 123Mi**

# Requests vs Limits

- Requests help the scheduler to make better decisions about which nodes to place Pods on.
- Limits allow contention for resources on a node to be handled in a specified manner.

# Requests vs Limits

- If container exceeds memory **limit**, it might be terminated.  
If restartable, the kubelet will restart it, as with any other type of runtime failure.
- If container exceeds memory **request**, likely to be evicted if/when node runs out of memory.

# Exercise

## Limits



# Quotas

# Resource Quotas

- Kubernetes resource to provide constraints that limit aggregate resource consumption per namespace.
- Limit the quantity of objects that can be created by type.
- Limit the total amount of compute resources that may be consumed.

# Resource Types

- Compute Resource Quota
- Storage Resource Quota
- Object Count Quota

# Quotas vs Capacity

- Quotas are independent of cluster capacity.
- Quotas are expressed in absolute units and do not increase automatically when adding nodes.
- Could write a custom controller to adjust quotas automatically.

# Exercise

## Quotas

# Requests/Limits

Module x

# Pod Scheduling

# 2007



# 2007

## CFS scheduler to appear in Linux kernel 2.6.23

Submitted by [admin](#) on Tue, 2007-07-10 02:42

The Linux kernel process scheduler, as you know it, has been completely ripped out and replaced with a completely new one called Completely Fair Scheduler (CFS). How fair it will be, remains to be seen, but in the meantime here's what its original creator Ingo Molnar has to say on the subject:

*80% of CFS's design can be summed up in a single sentence: CFS basically models an "ideal, precise multi-tasking CPU" on real hardware.*

*"Ideal multi-tasking CPU" is a (non-existent :-) CPU that has 100% physical power and which can run each task at precise equal speed, in parallel, each at  $1/nr\_running$  speed. For example: if there are 2 tasks running then it runs each at 50% physical power - totally in parallel.*

# Anonymous Hater

Anonymous (not verified)

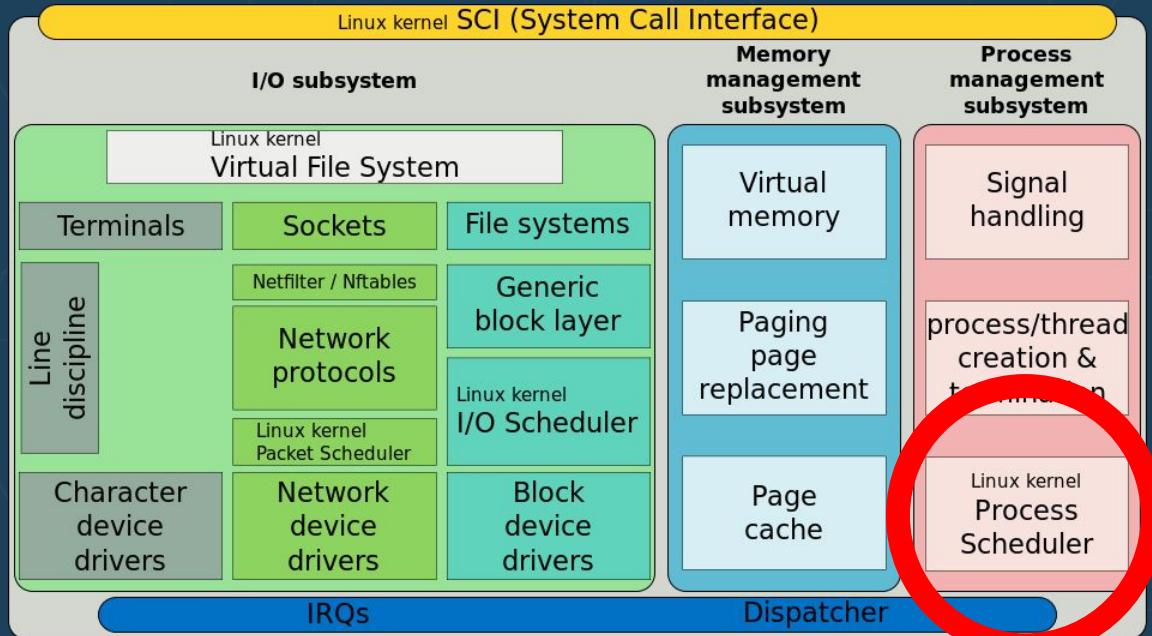
Tue, 2008-07-08 03:23

[Permalink](#)

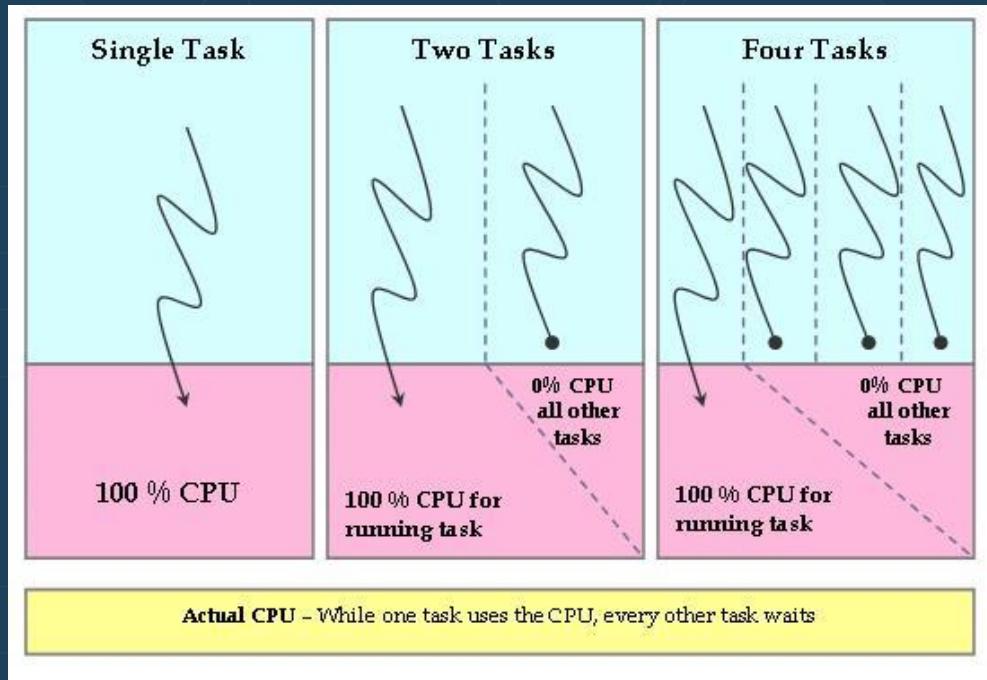
Ingo Molnar's the 'original

Ingo Molnar's the 'original  
creator' of CFS ? Shame, shame,  
shame.

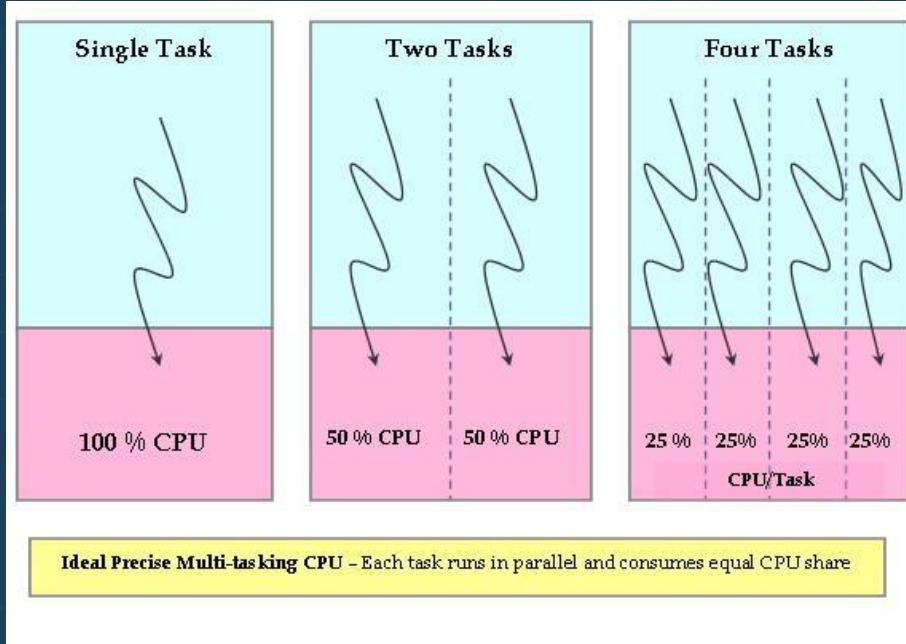
# CFS (Completely Fair Scheduler)



# Unfair!



# Fair



[https://en.wikipedia.org/wiki/Red%2Eblack\\_tree](https://en.wikipedia.org/wiki/Red%2Eblack_tree)

\*self-balancing binary search tree

# CFS is Tunable

```
ls /proc/sys/kernel | grep sched
```

```
sched_autogroup_enabled
sched_cfs_bandwidth_slice_us
sched_child_runs_first
sched_domain
sched_latency_ns
sched_migration_cost_ns
sched_min_granularity_ns
sched_nr_migrate
sched_rr_timeslice_ms
sched_rt_period_us
sched_rt_runtime_us
sched_shares_window_ns
sched_time_avg_ms
sched_tunable_scaling
sched_wakeup_granularity_ns
```

# 2007

## Add group awareness to CFS - v2

**From:** Srivatsa Vaddagiri <vatsa@linux.vnet.ibm.com>  
**To:** Ingo Molnar <mingo@elte.hu>  
**Subject:** [PATCH 0/2] Add group awareness to CFS - v2  
**Date:** Sat, 23 Jun 2007 18:45:45 +0530  
**Cc:** Nick Piggin <nickpiggin@yahoo.com.au>,efault@gmx.de, kernel@kolivas.org, containers@lists.osdl.org, ckrmtch@lists.sourceforge.net, torvalds@linux-foundation.org, akpm@linux-foundation.org, pwil3058@bigpond.net.au, tong.n.li@intel.com, wli@holomorphy.com, linux-kernel@vger.kernel.org, dmitry.adamushko@gmail.com, balbir@in.ibm.com, dev@sw.ru  
**Archive-  
link:**

Hi Ingo,

Here's an update for the group-aware CFS scheduler that I have been working on.

(For those reading these patches for the first time:)

The basic idea is to reuse CFS core and other pieces of scheduler like smpnice-driven load balance for driving fairness between 'schedulable entities' other than tasks, for ex: users or containers.

# Group Aware CFS

Alice

Bob

PID

PID

50%

PID

PID

50%

# Running cGroups

```
$ sudo cgcreate -a bork -g memory:mycoolgroup
```

```
$ ls -l /sys/fs/cgroup/memory/mycoolgroup/
-rw-r--r-- 1 bork root 0 Okt 10 23:16 memory.kmem.limit_in_bytes
-rw-r--r-- 1 bork root 0 Okt 10 23:14 memory.kmem.max_usage_in_bytes
```

```
$ sudo echo 10000000 >
/sys/fs/cgroup/memory/mycoolgroup/memory.kmem.limit_in_bytes
```

```
          $ sudo cgexec -g memory:mycoolgroup bash
```

```
$ ./someprogram
```

Caused by:

Cannot allocate memory (os error 12)

```

root@master1:~          root@master1:~          root@master1:~
nts.
See 'docker exec --help'.

Usage: docker exec [OPTIONS] CONTAINER COMMA
ND [ARG...]

Run a command in a running container
root@master1:~# docker exec 5da07fd14687 /bin
/sh

^X^C
root@master1:~# ^C
root@master1:~# docker exec -it 5da07fd14687
/bin/sh
/ #
/ # dd if=/dev/zero of=/dev/null
]
] ^C
/ # dd if=/dev/zero of=/dev/null
[

caused "exec: \"dd if=/dev/
zero of=/dev/null\": stat
dd if=/dev/zero of=/dev/n
ll: no such file or direct
ory"
docker: Error response fro
m daemon: invalid header f
ield value "oci runtime er
ror: container_linux.go:24
7: starting container proc
ess caused \"exec: \\\\"dd
if=/dev/zero of=/dev/null\"
\\\": stat dd if=/dev/zero
of=/dev/null: no such file
or directory\\n".
root@master1:~# docker run
-it busybox /bin/sh
/ # dd if=/dev/zero of=/de
v/null
[

Every 0.1s: docker stats --no-stream   Mon Aug 28 17:49:44 2017
CONTAINER           CPU %             MEM USAGE / LIMIT      ME
M %
9694f1589fd5      50.35%           92 KiB / 3.765 GiB    0.
00%                648 B / 648 B     0 B / 0 B            2
5da07fd14687      50.19%           164 KiB / 3.765 GiB   0.
00%                1.386 kB / 648 B    0 B / 0 B            4

```

# Pod Placement

Module 22

# Pod Scheduling

# Scheduling Methods

1. nodeSelector

```
kubectl explain deployment.spec.template.spec.nodeSelector
```

2. nodeAffinity

```
kubectl explain deployment.spec.template.spec.affinity.nodeAffinity
```

3. podAffinity

```
kubectl explain deployment.spec.template.spec.affinity.podAffinity
```

4. podAntiAffinity

```
kubectl explain deployment.spec.template.spec.affinity.podAntiAffinity
```

5. Taints

```
kubectl explain node.spec.taints
```

6. Tolerations

```
kubectl explain deployment.spec.template.spec.tolerations
```

# nodeSelector

Use **kubectl** to add labels to nodes...

```
kubectl label nodes worker2 disktype=ssd
```



# nodeSelector

Specify the nodeSelector with kubectl run...

```
kubectl run pod --image nginx --restart Never --overrides '{"apiVersion": "v1", "spec": {"nodeSelector": {"disktype": "ssd"}}}'
```

Specify the nodeSelector in a PodSpec...

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

# nodeAffinity

## required

Must run on specified nodes

```
spec:  
  affinity:  
    nodeAffinity:  
      requiredDuringSchedulingIgnoredDuringExecution:  
        nodeSelectorTerms:  
          - matchExpressions:  
              - key: kubernetes.io/e2e-az-name  
                operator: In  
                values:  
                  - e2e-az1  
                  - e2e-az2
```

## preferred

Try to... if not, schedule elsewhere

```
spec:  
  affinity:  
    nodeAffinity:  
      preferredDuringSchedulingIgnoredDuringExecution:  
        - weight: 1  
          preference:  
            matchExpressions:  
              - key: another-node-label-key  
                operator: In  
                values:  
                  - another-node-label-value
```

# podAffinity

'required' = it must run on specified nodes.

'preferred' = try to... if not, schedule elsewhere.

```
affinity:  
  podAffinity:  
    requiredDuringSchedulingIgnoredDuringExecution:  
    - labelSelector:  
        matchExpressions:  
        - key: service  
          operator: In  
          values: ["S1"]  
    topologyKey: failure-domain.beta.kubernetes.io/zone
```

# Taints & Tolerations

- Allows you to mark (“taint”) a node so that **no** pods can schedule onto it unless a pod **explicitly** “tolerates” the taint.
- Marking nodes instead of pods (as in node affinity/anti-affinity) is particularly useful for situations where most pods in the cluster should avoid scheduling onto the node. For example: master nodes

# Taints & Tolerations

Use kubectl to set taints on nodes...

```
kubectl taint nodes NODE key=value:NoSchedule
```

Add a toleration to the PodSpec to tolerate a taint on a node...

```
tolerations:  
- key: "key"  
  operator: "Equal"  
  value: "value"  
  effect: "NoSchedule"
```

# Exercise

## Pod Scheduling

# Node Management

Module 23

# Maintenance

- Use `kubectl drain` to gracefully terminate all pods on the node while marking the node as unschedulable.
- Perform maintenance work on the node, reboot, etc.
- Use `kubectl uncordon` to make the node schedulable again.

# Drain

# kubectl drain

Drain node in preparation for maintenance.

```
kubectl drain NODE [flags]
```

# Exercise

Drain

# Cordon

# Unschedulable Node

- Marking a node as unschedulable will prevent new pods from being scheduled to that node, but will not affect any existing pods on the node.
- This is useful as a preparatory step before a node reboot, etc.

# kubectl cordon

Mark node as unschedulable.

```
kubectl cordon NODE [flags]
```

# kubectl uncordon

Mark node as schedulable.

```
kubectl uncordon NODE [flags]
```

# Exercise

Cordon

# Authentication

Module 24

# Authentication

The act of confirming the identity of a specific user. In other words, proving that a user is who she or he claims to be.

# Authorization

The function of determining access rights for that specific user.

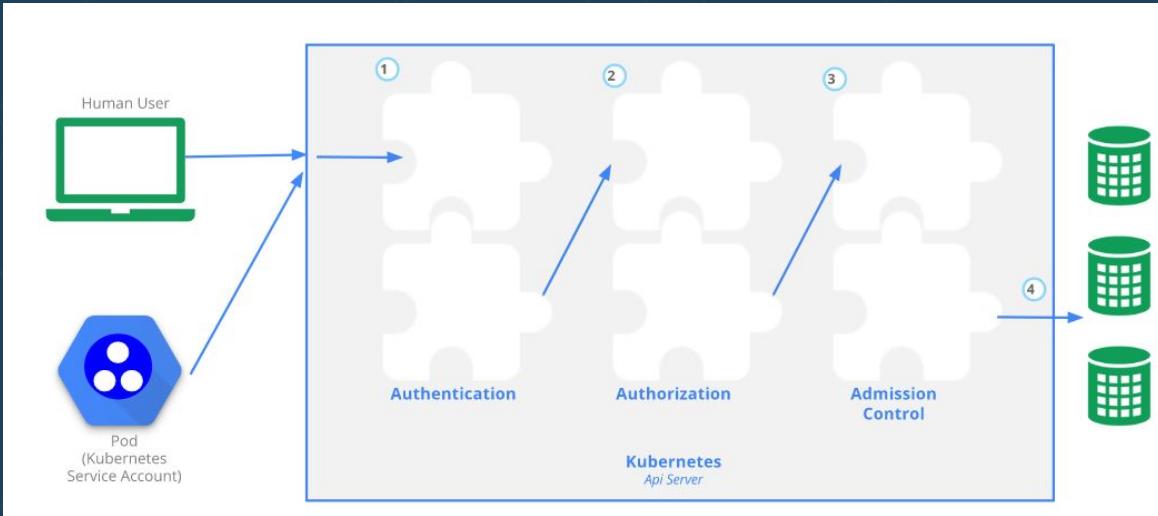
# Authentication

2 Categories of Kubernetes Users

1) User Accounts

2) Service Accounts

# Access to the K8s API



Step 1. Authentication: Examines client certificates, password, plain token, bootstrap tokens, OIDC, etc. If multiple authenticators specified, each one can be tried in sequence.

Step 2. Authorization: After the request is authenticated from a specific user, group, or serviceaccount, the request must be authorized. Kube-API can be set to utilize RBAC, ABAC, Webhook, etc.

Step 3. Last step. Accepts a variety of plugins. i.e. LimitRanger (ensure request meets any constraints), NamespaceLifecycle (can't delete default, kube-system, and kube-public), ServiceAccounts ( Sets ServiceAccount), more available at <https://kubernetes.io/docs/admin/admission-controllers/#namespacelifecycle>

Step 4. It's written to etcd!!!!

# User vs. Service Accounts

User Accounts	Service Accounts
For humans.	For processes running in pods.
Globally unique.	Only unique within a namespace.
Possibly synced to a corporate database via LDAP or Active Directory.	Created for specific tasks.

# Service Accounts

- Users managed by Kubernetes API.
- Bound to specific namespaces and created via API calls.
- Credentials are stored as **Secrets** which are mounted into pods allowing cluster processes to talk to Kubernetes API.

# Service Accounts - Admission Controller

- **Admission Controller Plugin (Part of the KubeAPI).**
  - Responsible for setting default ServiceAccount for all pods to `default` and adding the ServiceAccount's `ImagePullSecrets`.
  - Adds a volume to the pod that contains token for API access.
  - Adds volume source to each container in the pod mounted at `/tmp/secrets/kubernetes.io/serviceaccount`.

# Service Accounts - Token Controller

- **Token Controller (Part of the Controller-Manager).**
  - Creates secrets for new service accounts.
  - Deletes all secrets when a service account gets deleted.
  - Observes secret additions.
  - Observes secret deletions.

# Service Accounts - Controller

- **Service Account Controller (Part of the Controller-Manager).**
  - Manages ServiceAccounts inside namespaces to ensure ServiceAccount named default exists in every active namespace.

# Methods for Authentication

- Client certificates
- Bearer Tokens
- HTTP Basic auth (username/password)

# Federation

- Federation in Beta in Kubernetes 1.6
- Allows you to present multiple Kubernetes clusters under a single federated control plane.

# Exercise

## Authentication

# Authorization

Module 25

# Authentication

The act of confirming the identity of a specific user. In other words, proving that a user is who she or he claims to be.

# Authorization

The function of determining access rights for that specific user.

# RBAC (Role Based Access Control)

- Authorization.
- Fine-grained permissions for ServiceAccounts and Users.

# Logging and Monitoring

Module 26

# Logging

# Kubernetes Logging

- Provides two logging end-points for applications and cluster logs.
  - Stackdriver Logging (Google Cloud Platform)
  - Elasticsearch
- Fluentd logging agent handles log collection, parsing and distribution.

# Basic Logging Layers

- Application
- Node
- Cluster



fluentd

# Fluentd

An open source data collector, which lets you unify the data collection and consumption for a better use and understanding of data.

## Access logs

Apache

## App logs

Frontend

Backend

## System logs

syslogd

## Databases

## Alerting

Nagios

## Analysis

MongoDB

MySQL

Hadoop

## Archiving

Amazon S3



filter / buffer / routing

# Fluentd

- Open-source log collector enabling a ‘Log Everything’ architecture with plugins for 600+ systems.
- Treats logs as JSON.
- Written primarily in C.
- Scalable to over 500,000 servers.

# Exercise

## Logging

# Monitoring

# It starts with cAdvisor...



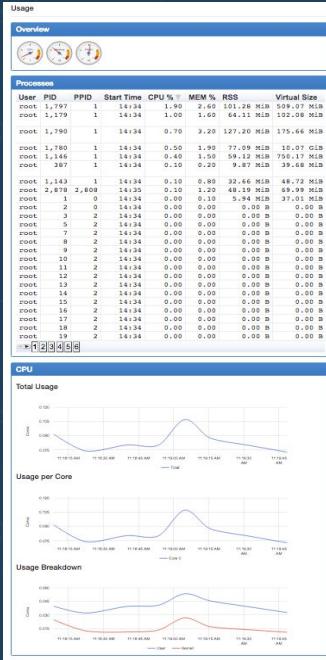
Analyzes resource usage and performance characteristics of running containers.

```
kubelet --help 2>&1 | grep "cAdvisor"
```

```
--cadvisor-port int32  
--container-hints string  
--log-cadvisor-usage  
--storage-driver-db string
```

The port of the localhost cAdvisor endpoint (default 4194)  
location of the container hints file (default "/etc/cadvisor/container\_hints.json")  
Whether to log the usage of the cAdvisor container  
database name (default "cadvisor")

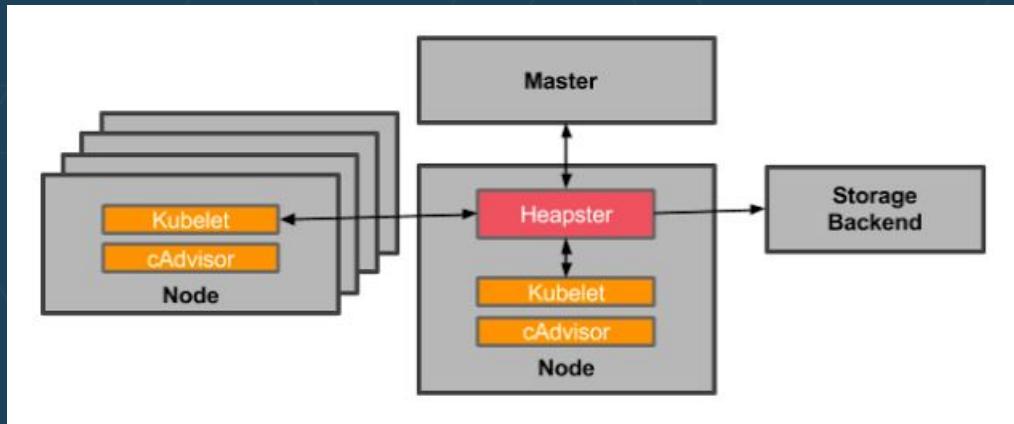
```
curl localhost:4194
```



# CPU Usage Memory Usage Network Throughput Filesystem

# Heapster

Runs as a pod that collects metrics from the kubelet (cadvisor) running on every node in the cluster.



# Heapster Storage Backends

- Log
- InfluxDB
- Google Cloud monitoring
- Google Cloud logging
- Hawkular-Metrics (metrics only)
- OpenTSDB
- Monasca (metrics only)
- Kafka (metrics only)
- Riemann (metrics only)
- Elasticsearch

# Exercise

## Monitoring