# Hands on Kubernetes Container Orchestration In Cluster and Grid Computing


By: Amir Hossein Sorouri

Cluster & Grid Computing

Instructor: Dr. Mohsen Sharifi

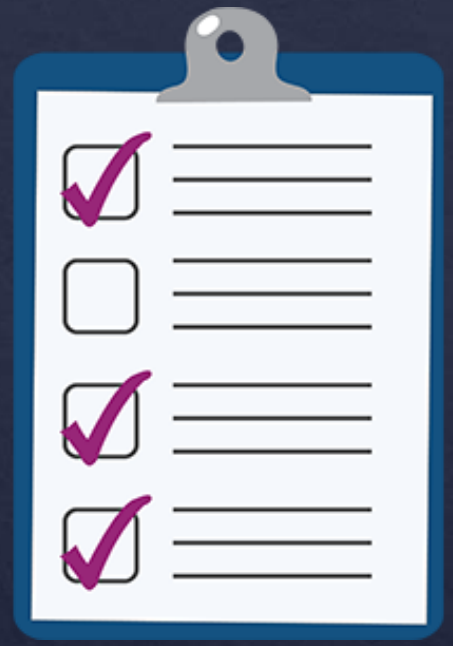

IU ST

Date: 2019-May

1. Cluster Review
2. Grid Review
3. Historical Review on Deployment
4. Historical Review on Softwares Architectures

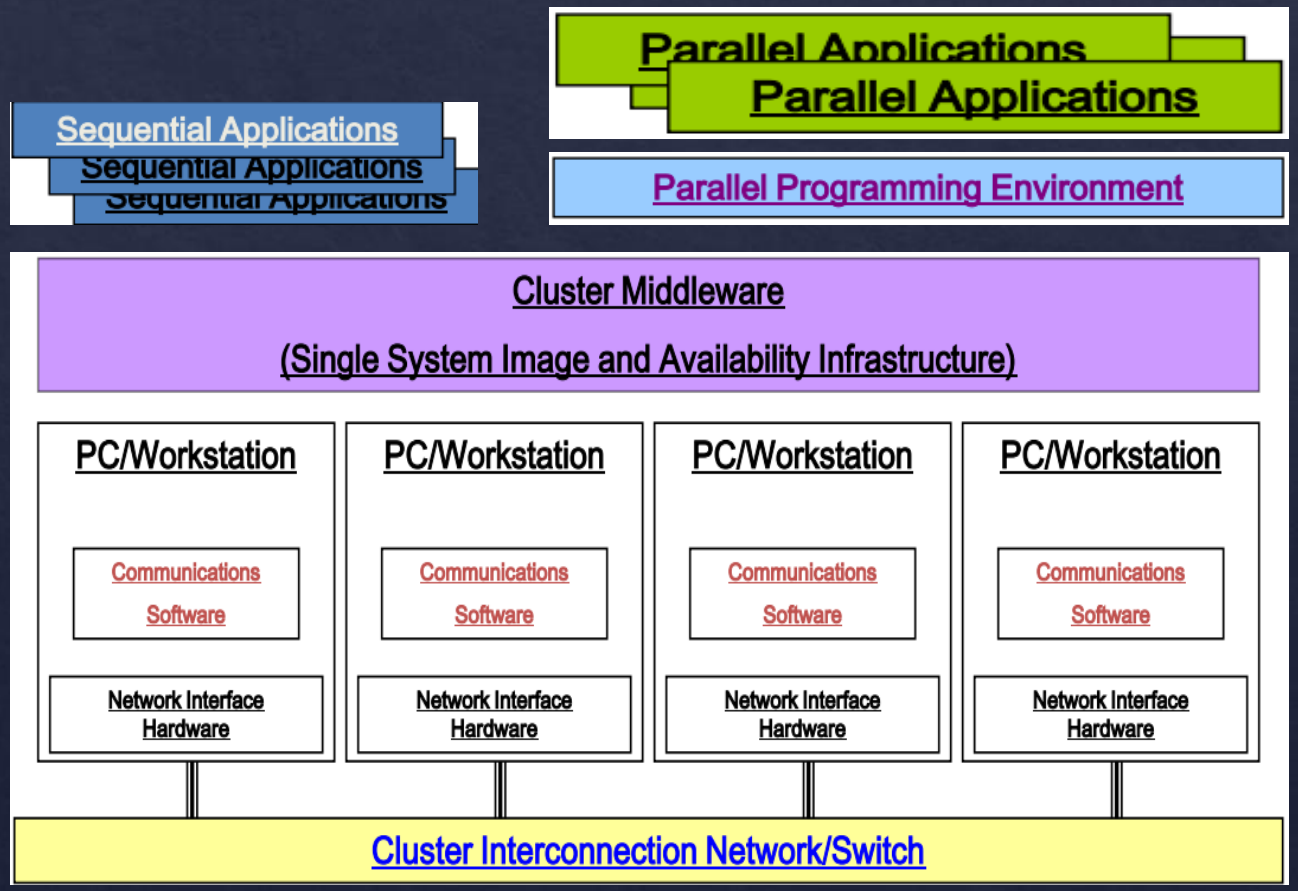
Packed up for the Journey…

5. Container Concepts and Fundamentals
6. Hands On Docker
7. Hands On Kubernetes



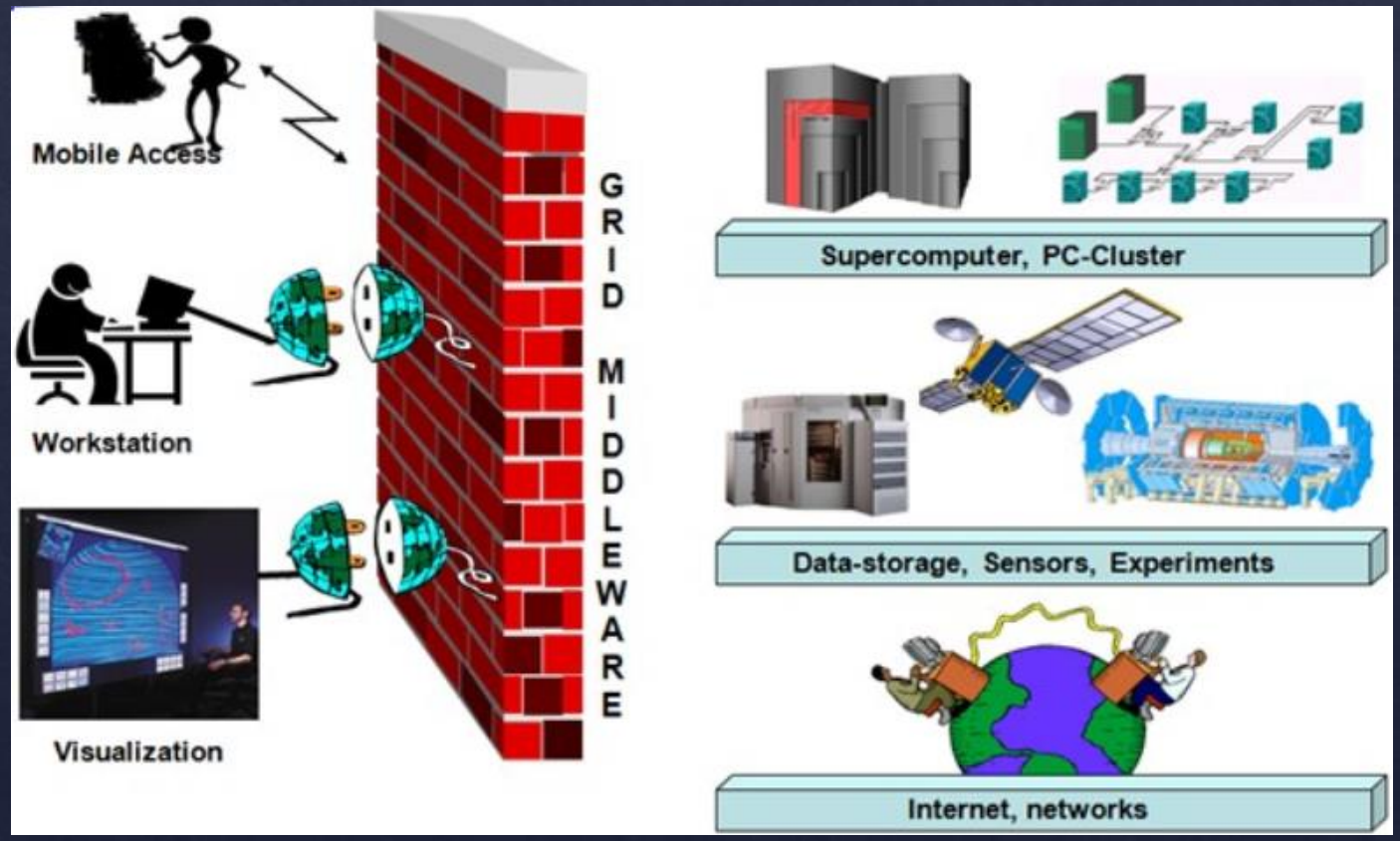
- Main Issues in Cluster Environment
  - Resource Management Scheduling
  - Fault Handling
  - Migration
  - Load Balancing

**All Applications**



# Three Main Issues in Grid Environment

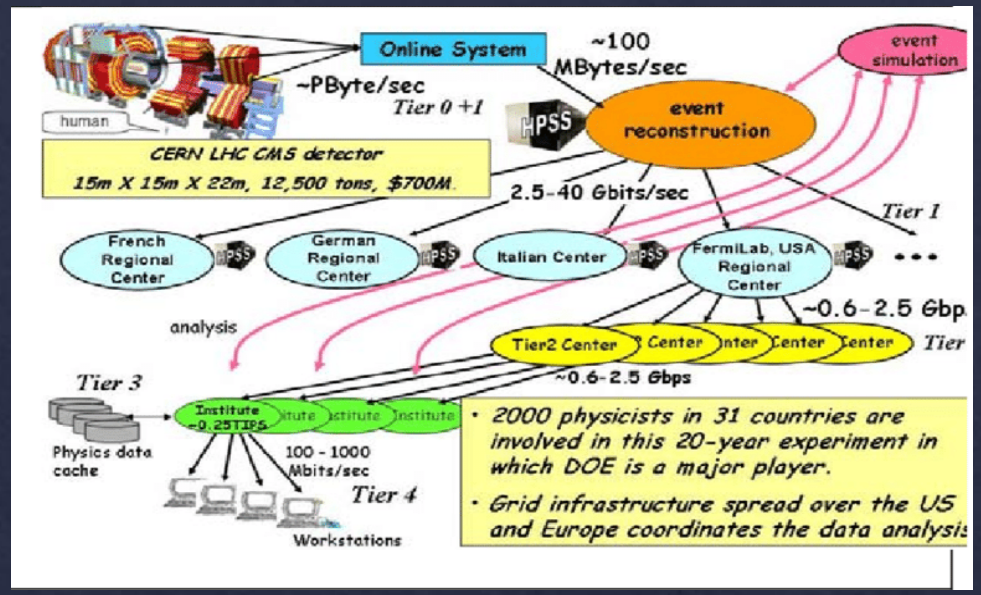
- Heterogeneity
- Scalability
- Adaptability




**All Applications**

# Grid Software/Hardware Stack

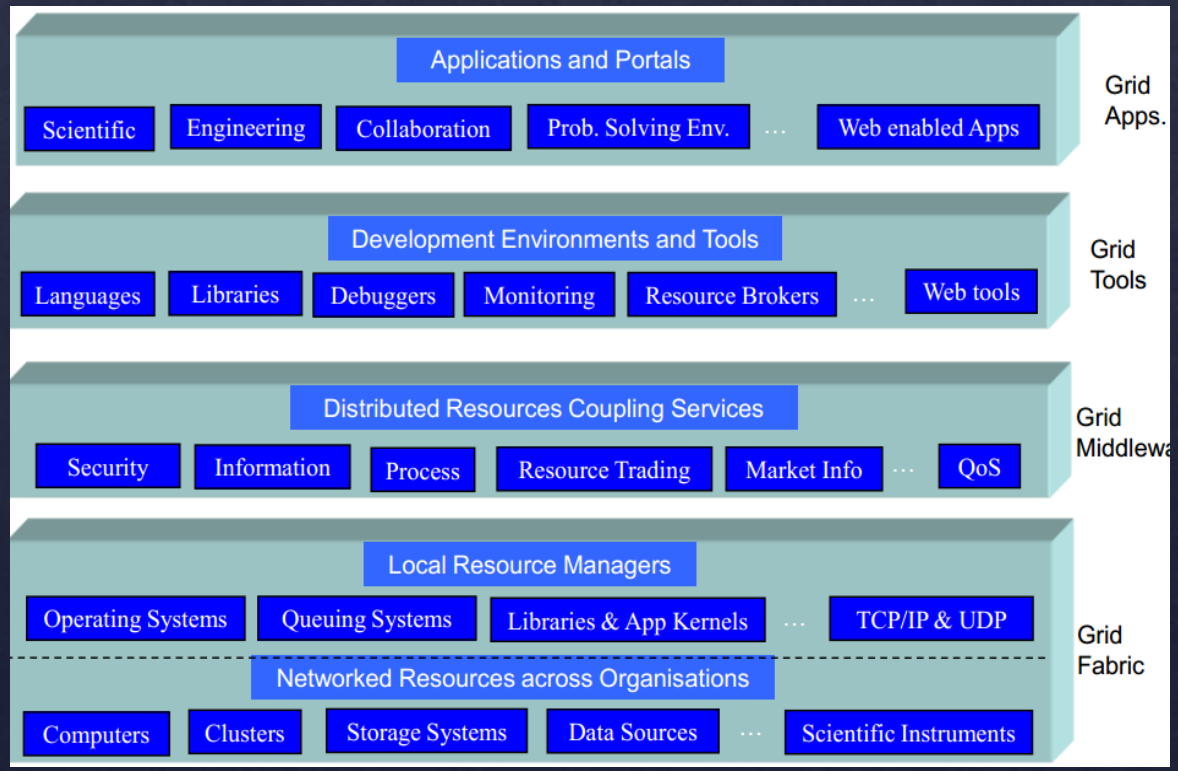
App




Tools

Monitoring

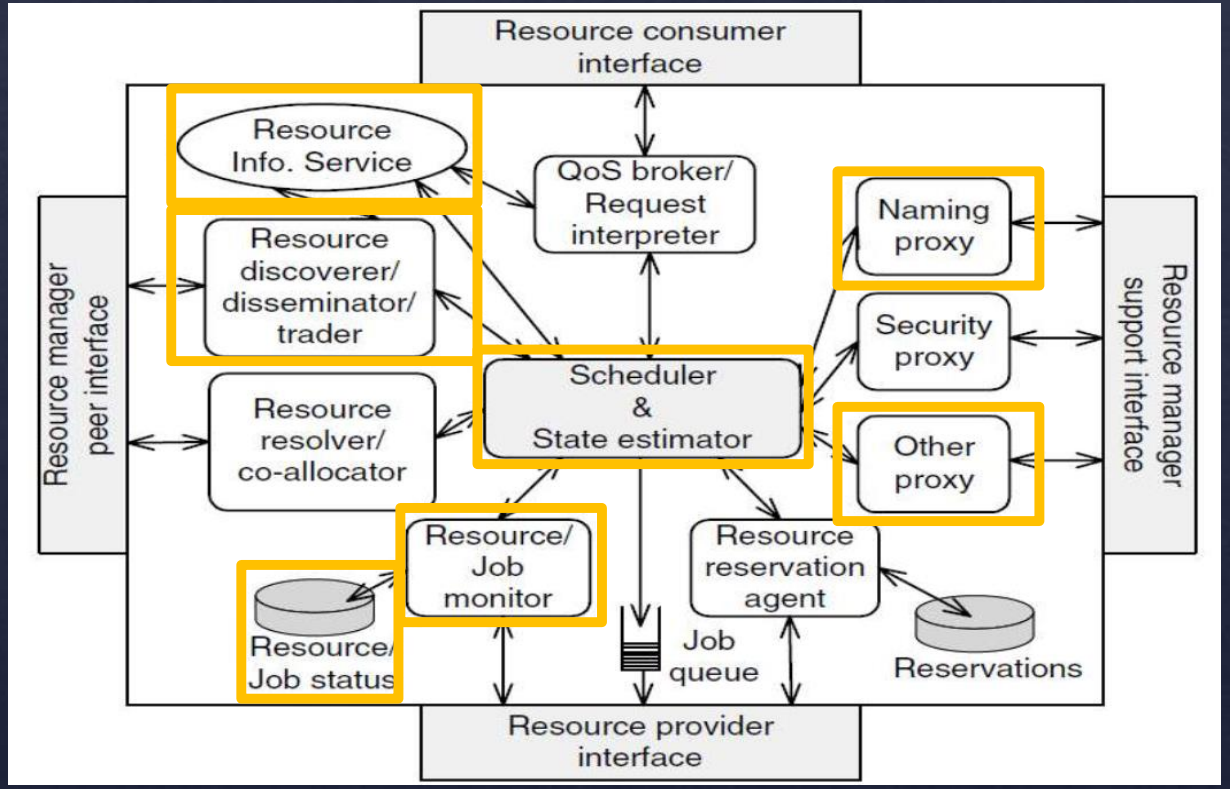
Fabric



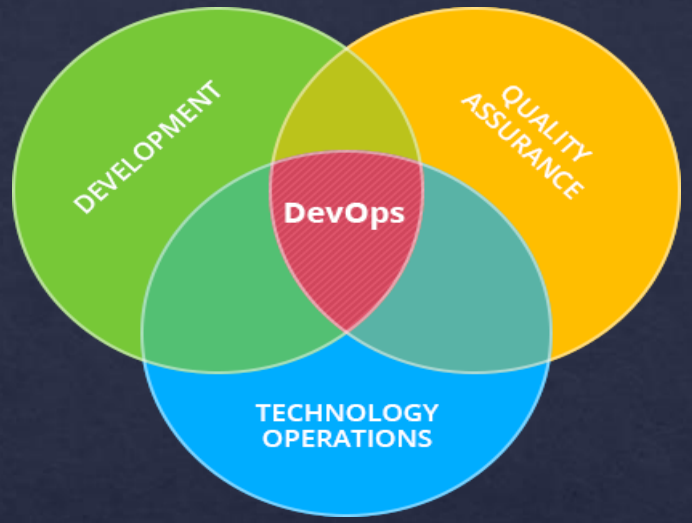
# Resource Management System(RMS)

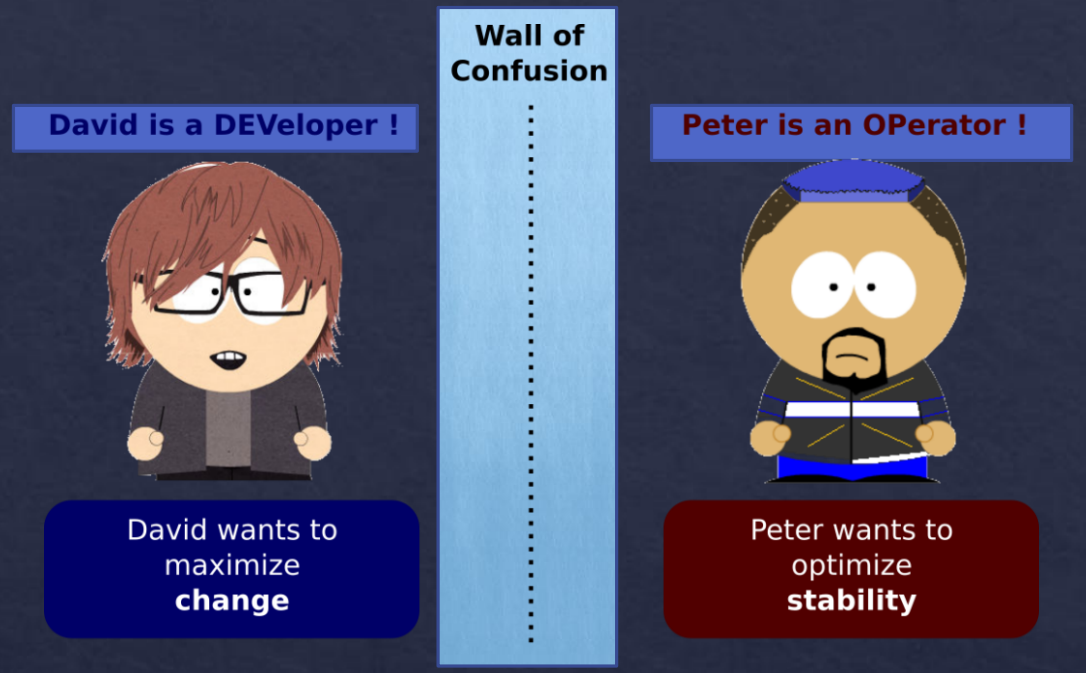
- Resource Dissemination and Discovery Protocols
  - Resource Dissemination Protocol
    - Information about resources
  - Discovery Protocol
    - Resource Discovery

- Resource Resolution and co-allocation Protocols
  - To
    - Schedule the job at the remote resources
    - To Simultaneously acquire multiple resources

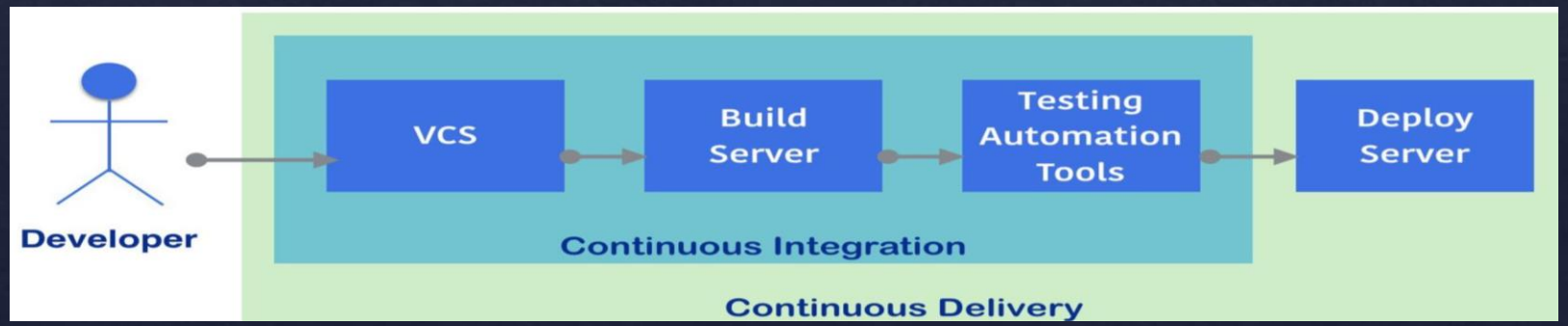


Problem:




Solution: Continuous Integration/Continuous Development (CI/CD)



# CI Combination of Tools: → Continous Delivery

- Version Control Systems: Helps developers to maintain program source code onto the server.
  - Git

- Build Server: Retrieve source code periodically/automatically when the developer updates the code to VCS, and then trigger a new build
  - Jenkins
  - Travis CI



- Testing Automation Tools: Invokes the unit test program after the build succeeds, then notifies the result to the developer and QA.
  - Postman

# Configuration Management (CM):

- Helps to configure an OS Including:
  - The user
  - Group
  - System libraries


- Manages multiple servers keeping them consistent with the desired state or configuration

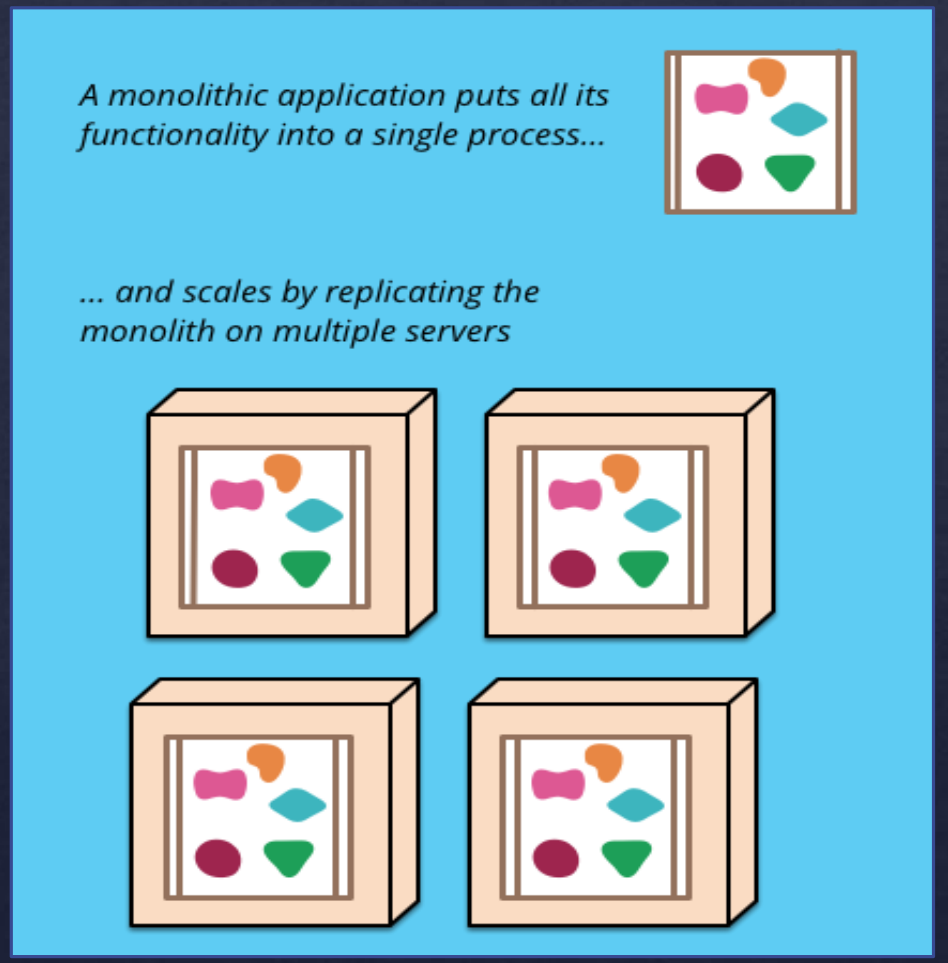
# CM Combination of Tools:

- Ansible
- Puppet

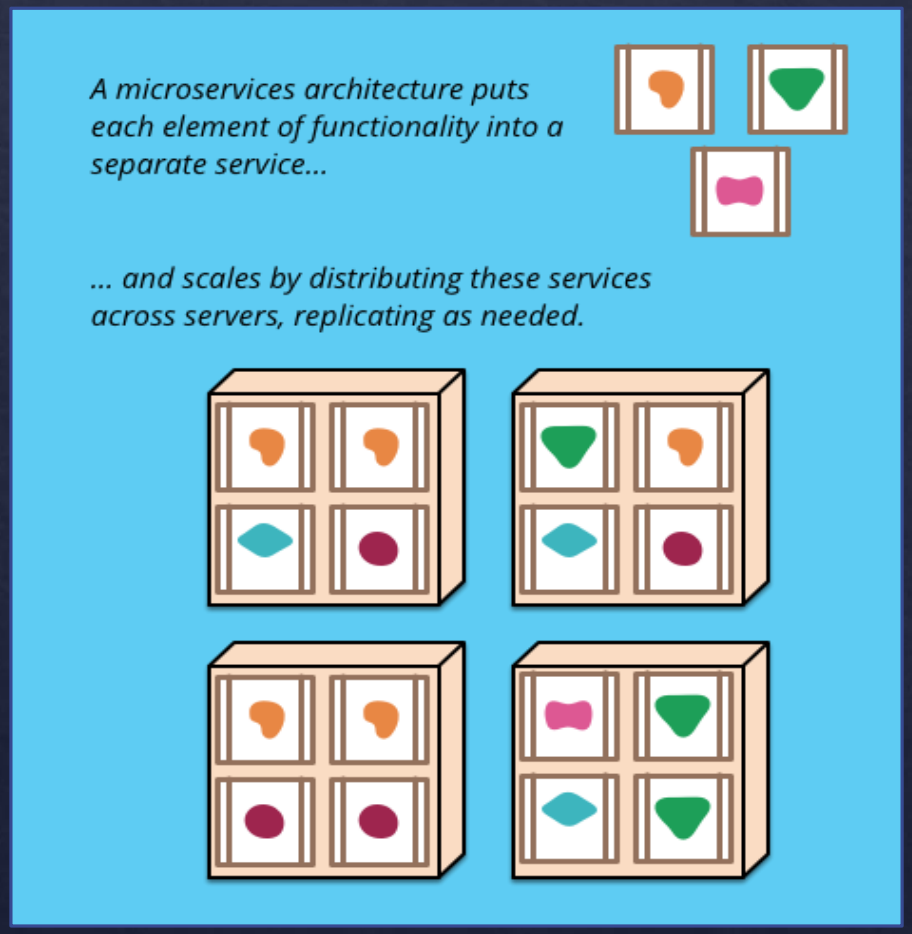

**Too Many Softwares**

# Monolithic Software Architecture:

- No Clear Definition
- But it Used to Have:
  - > than 50 modules or packages
  - > than 50 database tables
- May Takes
  - > than 30 minutes to build
- Cons
  - inflexible
  - unscalable
  - slow development
  - unreliable



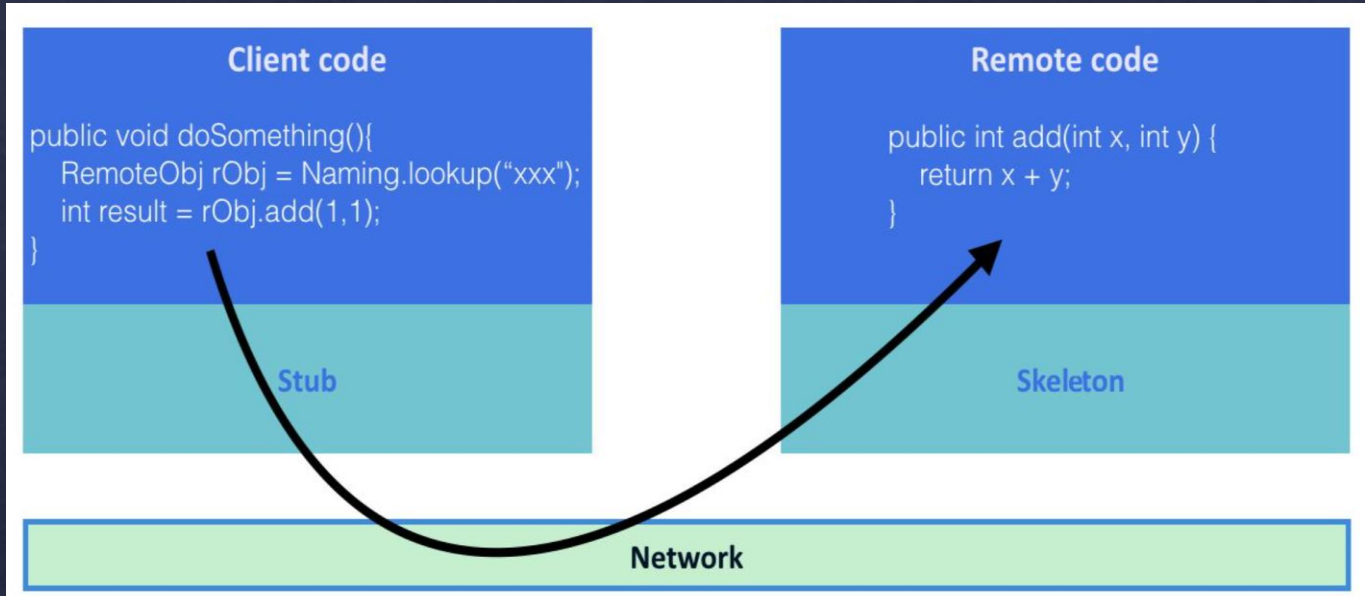
# Microservice Software Architecture:




- Has been attempted back in the 1990s
- Structures an application as a collection of services that are:
  - Highly maintainable
  - Highly testable
  - Loosely coupled
  - Independently deployable

# Remote Procedure Call:

- Use the module remotely (Good for Microservices)

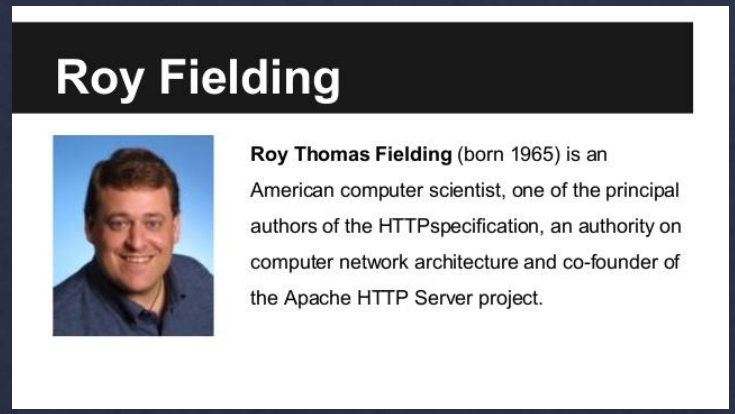


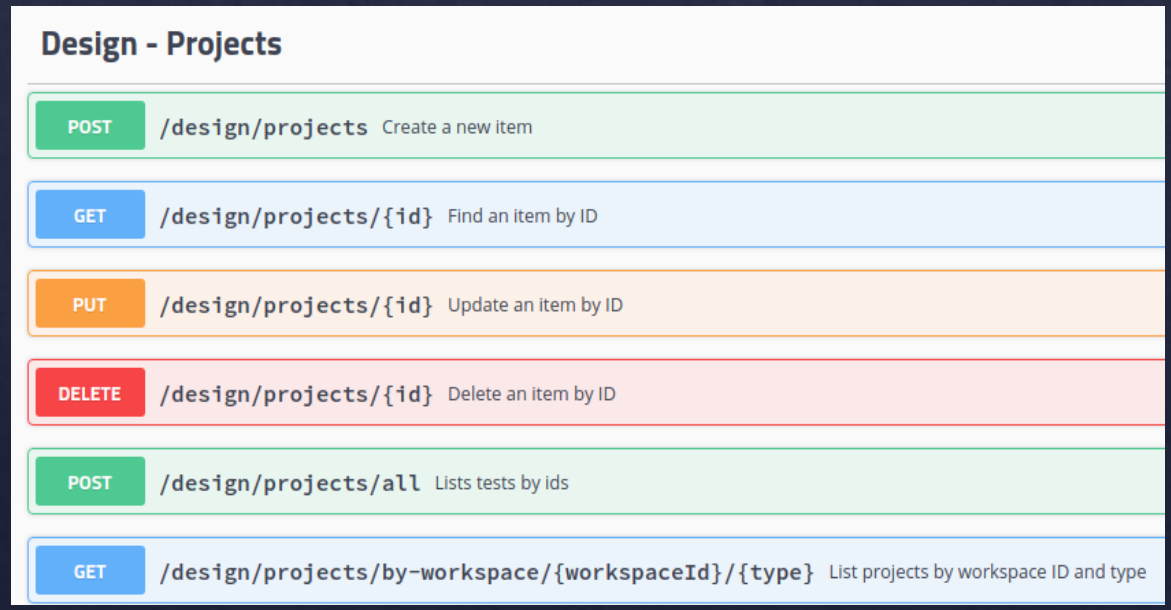

**stub** and **skeleton** are counterparts in a web service setup

- Cons :
  - Requires the same programming language
  - Less security consideration
- Have the benefit :
  - Divide application into multiple processes/programs
  - Programs can have separate source code repositories

# RESTful design:

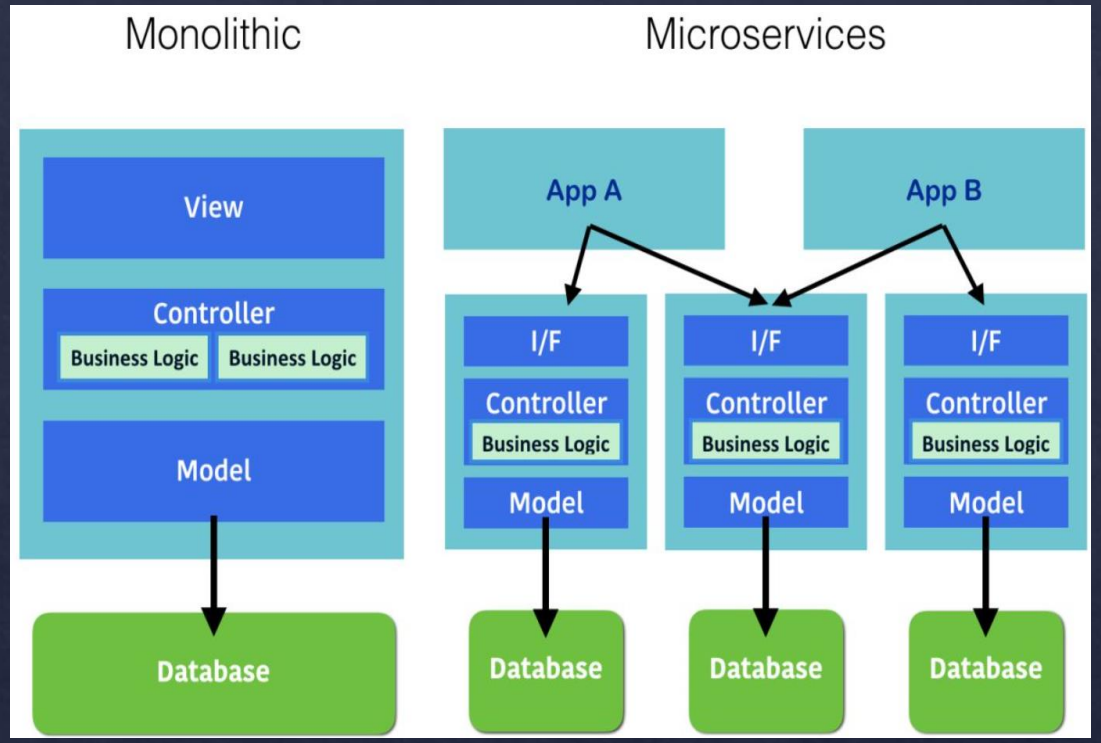
- Invented by Roy Fielding
- Defined in his PHD thesis in 2000
- Widely accepted by many developers
- De facto standard of distributed applications
- RESTful application allows any programming language as it is HTTP-based



# Microservices:

- Should care about the following topics:
  - **Stateless**:
    - Doesn't store user session
    - Helps to scale out easier.
  - **Versioning and compatibility**:
    - May change and update the API
    - But should define a version
    - Should have backward compatibility
  - **No shared datastore**:
    - The microservice owns the datastore such as database.
  - **Integrate CI/CD**:
    - Should adopt CI/CD process to eliminate management effort.

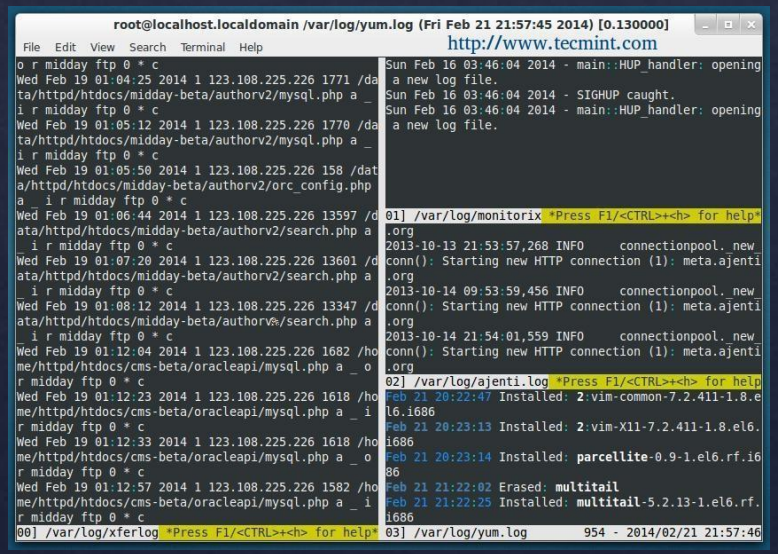


# Contents Cont.

5. Container Concepts and Fundamentals
6. Docker Hands On
7. Kubernetes Hands On

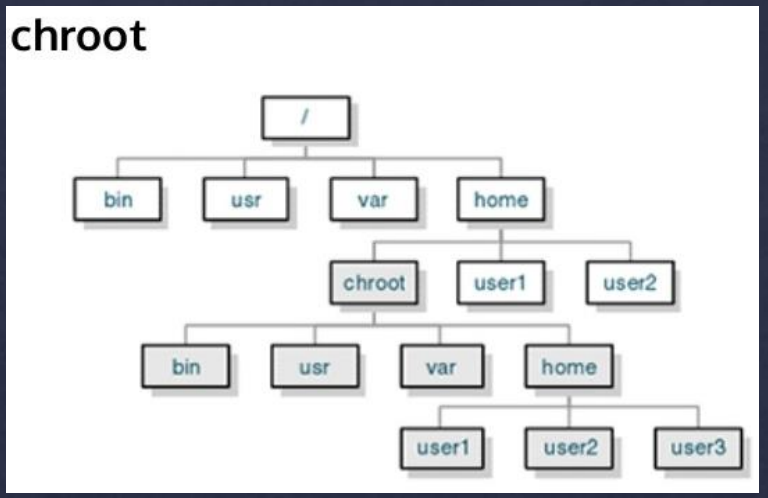
# Development of Virtualization

- They say necessity is the mother of invention, and the history of computers is no exception
- Back in the ’60s, computers were a rare commodity
  - To rent one cost well over a thousand dollars a month
    - which is why in the 1960s and through the 1970s, we saw the development of virtualization
- They made it possible to control all processing from a single location
  - Multiple computer terminals were connected to a single mainframe
- Disadvantages
  - If the user were to crash the central computer
    - The system would go down for everyone

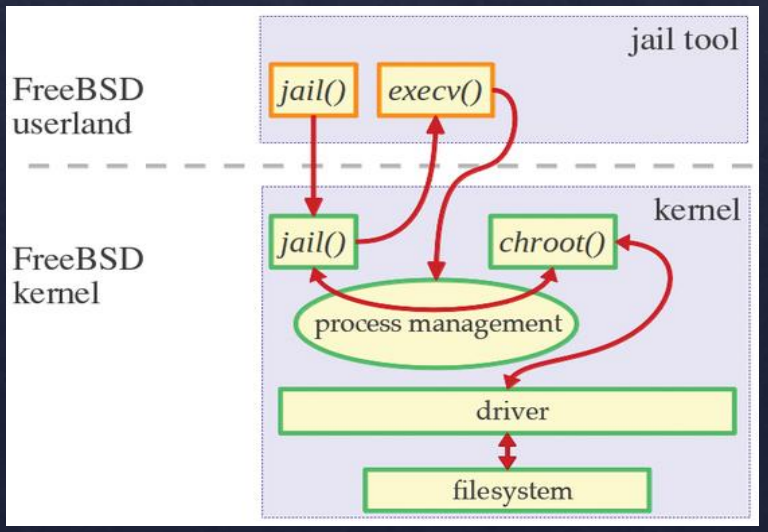


# Development of Virtualization

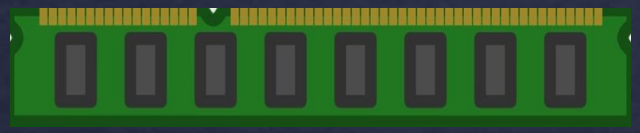
- We took another step towards creating shared, yet isolated, in 1979
  - Development of the chroot (change root) command
    - Isolates system processes into their own segregated filesystems




- Skip forward a bit in time to the **1990s**
  - Bill Cheswick, His solution was to make modifications to a chrooted environment
    - Result of his studies → Linux jail command.
    - Process sandboxing features:
      - ✓ File systems
      - ✓ Users
      - ✓ Networks
      - ✓ etc.



# The key feature of container is isolation


- Strongest network isolation
- Strongest data isolation
- Strongest metadata isolation

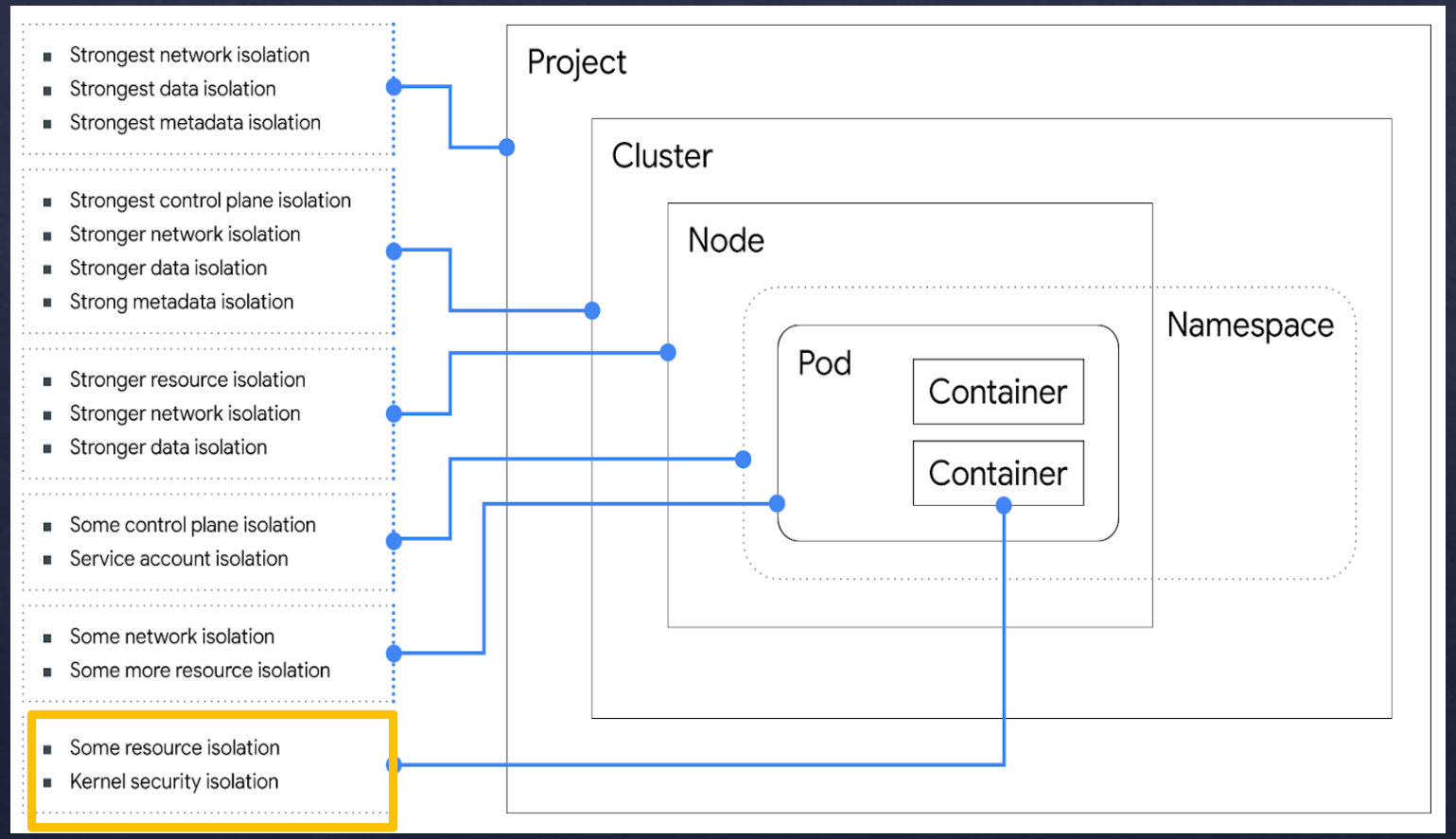
- Strongest control plane isolation
- Stronger network isolation
- Stronger data isolation
- Strong metadata isolation

- Stronger resource isolation
- Stronger network isolation
- Stronger data isolation

- Some control plane isolation
- Service account isolation

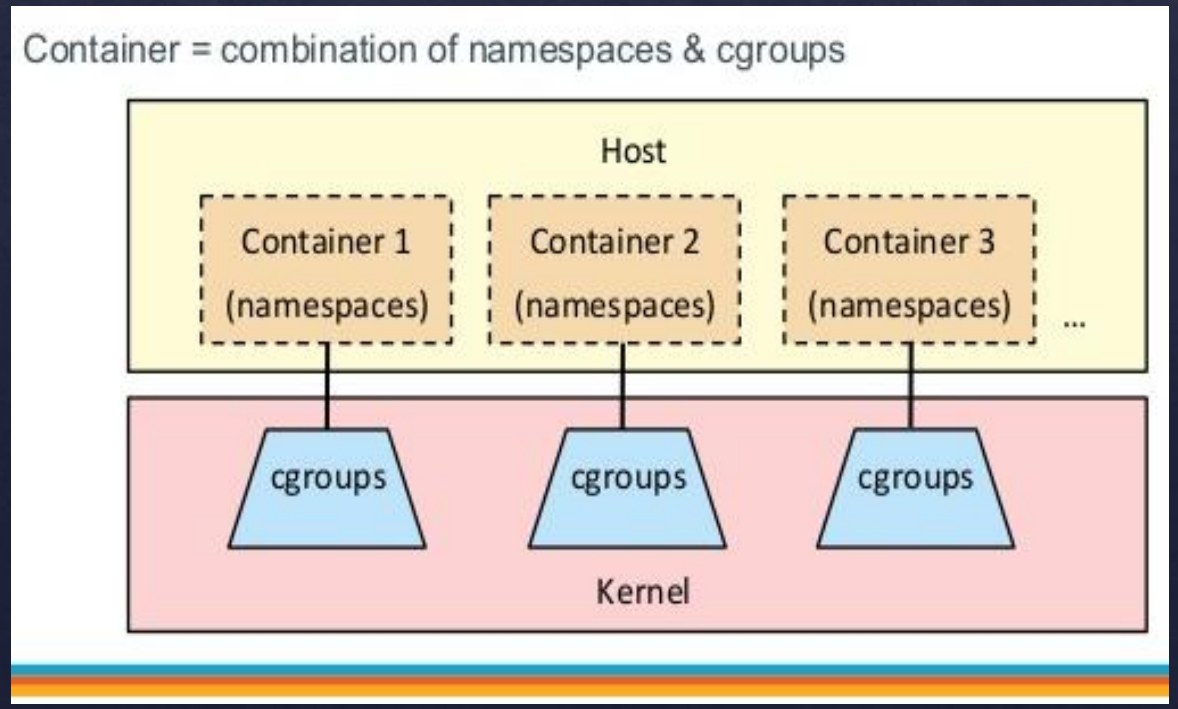
- Some network isolation
- Some more resource isolation

- Some resource isolation
- Kernel security isolation



# Linux Namespaces

- Provide logical partitions of system resources
  - mounting point (mnt)
  - process ID (PID)
  - network (net)




```
$ ps axf
  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:00 [kthreadd]
    3 ?        S      0:42  \_ [ksoftirqd/0]
    5 ?        S<     0:00  \_ [kworker/0:0H]
    7 ?        S      8:14  \_ [rcu_sched]
    8 ?        S      0:00  \_ [rcu_bh]
```

1

```
$ sudo unshare --fork --pid --mount-proc=/proc /bin/sh
$ ps axf
  PID TTY      STAT   TIME COMMAND
    1 pts/0    S      0:00 /bin/sh
    2 pts/0    R+     0:00 ps axf
```

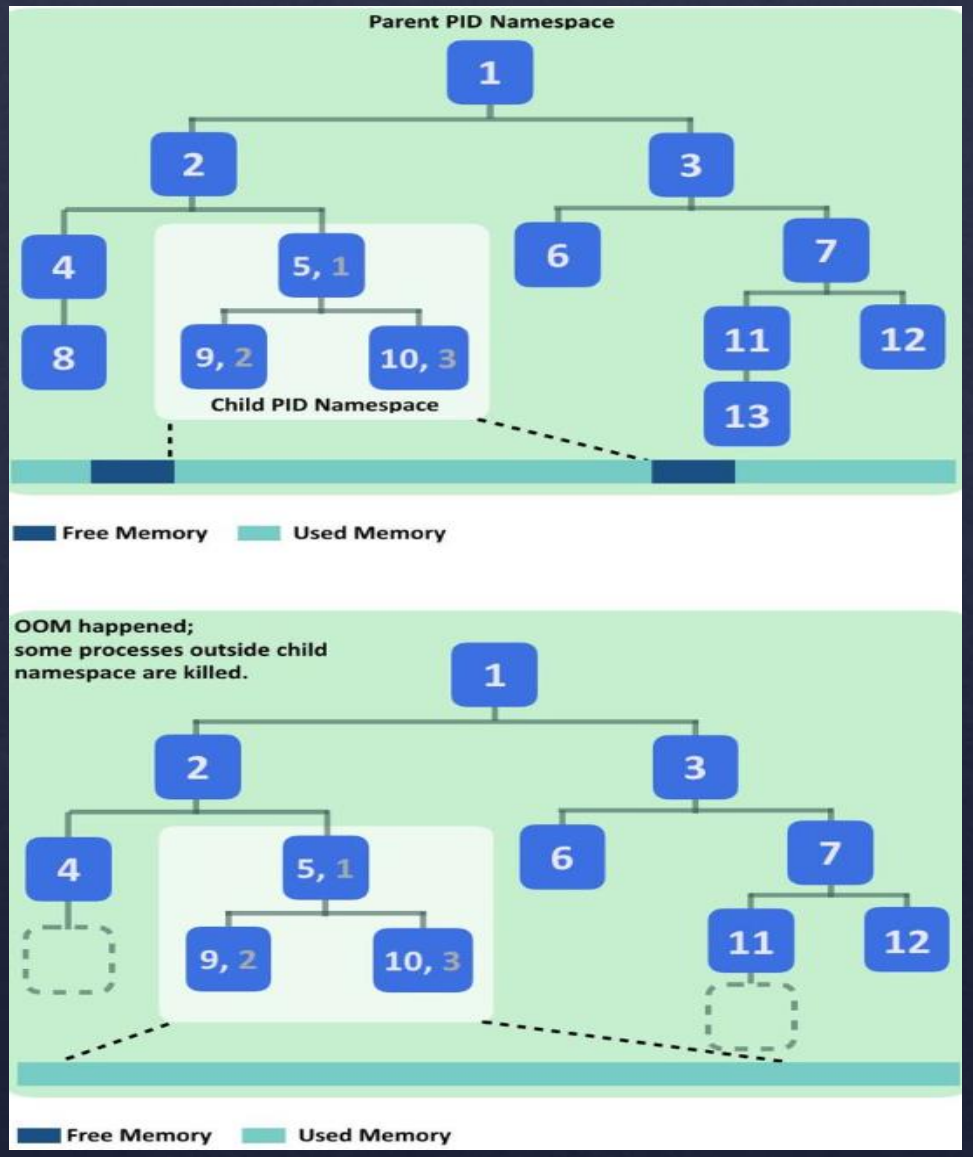
2

```
$ ps axf // from another terminal
  PID TTY   COMMAND
  ...
  25744 pts/0 \_ unshare --fork --pid --mount-proc=/proc
  /bin/sh
  25745 pts/0    \_ /bin/sh
  3305  ?     /sbin/rpcbind -f -w
  6894  ?     /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u
```

3

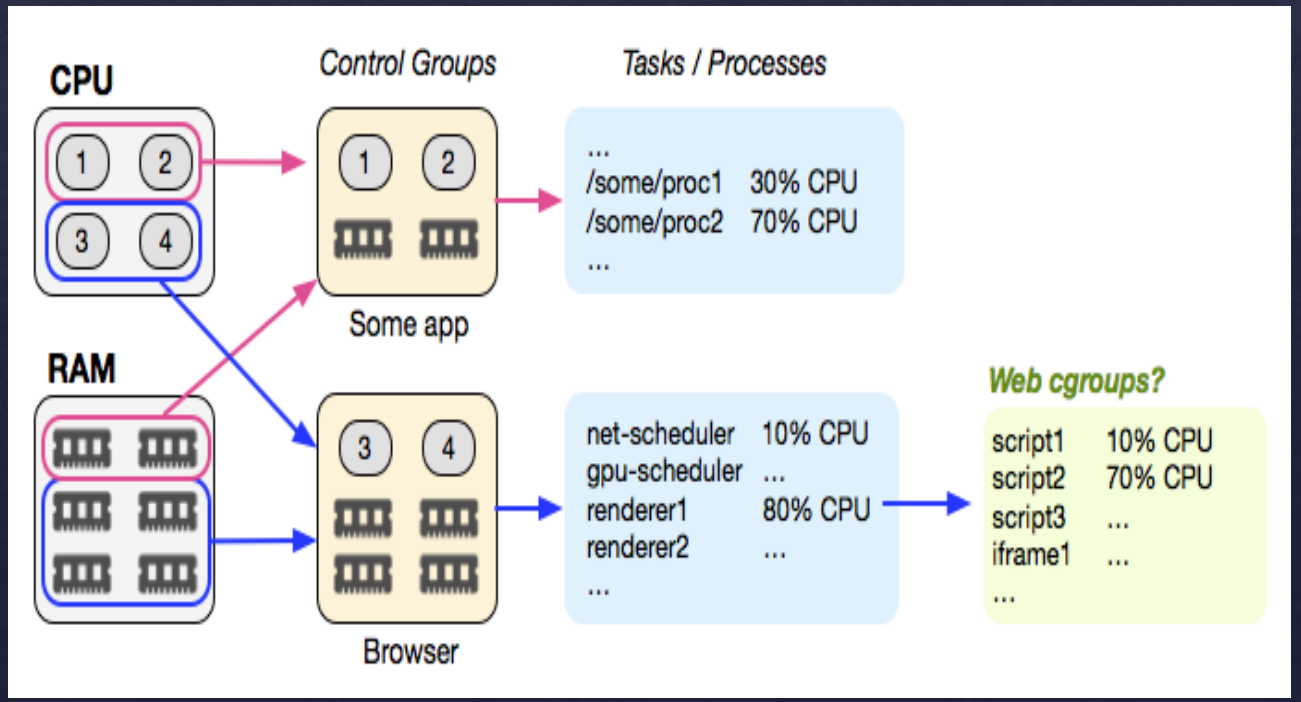
# Out of Memory

The following diagram illustrates the PID namespaces and how an out-of-memory (OOM) event can affect other processes outside a child namespace

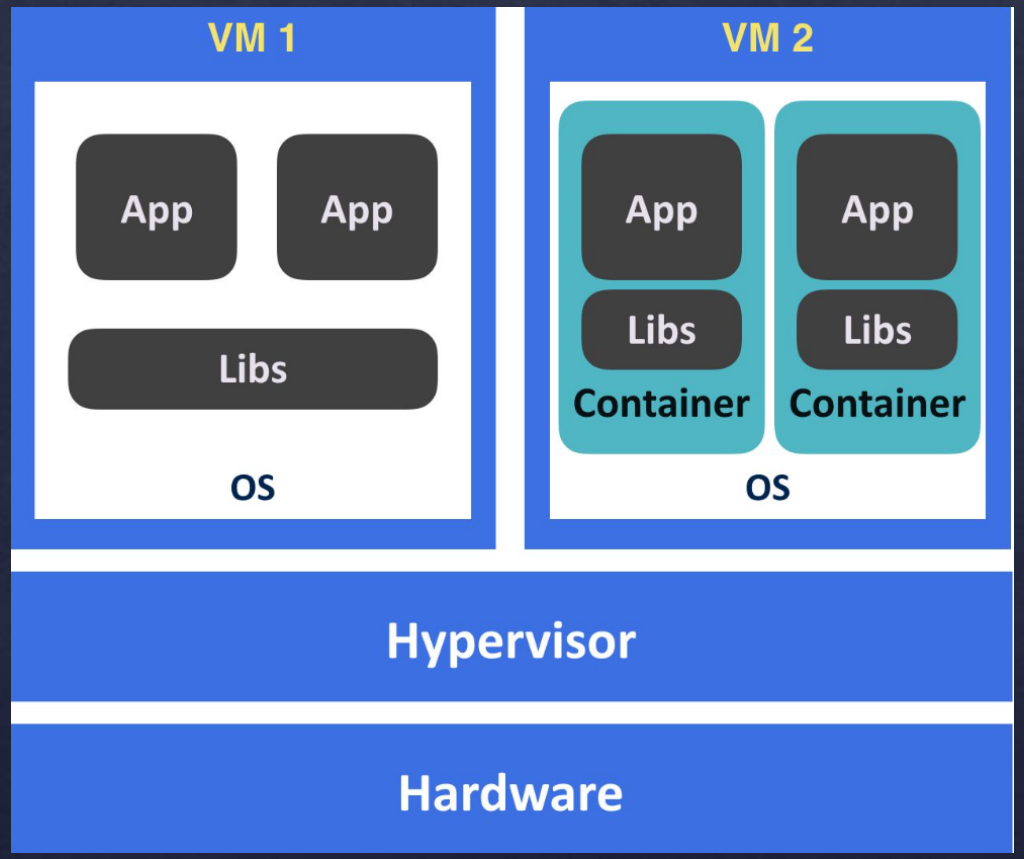


# Linux Control Groups(cgroup)

- Uses in order to
  - Limit resource usage by
    - Setting constraint on different kinds of system resources



# These Days of Virtualization

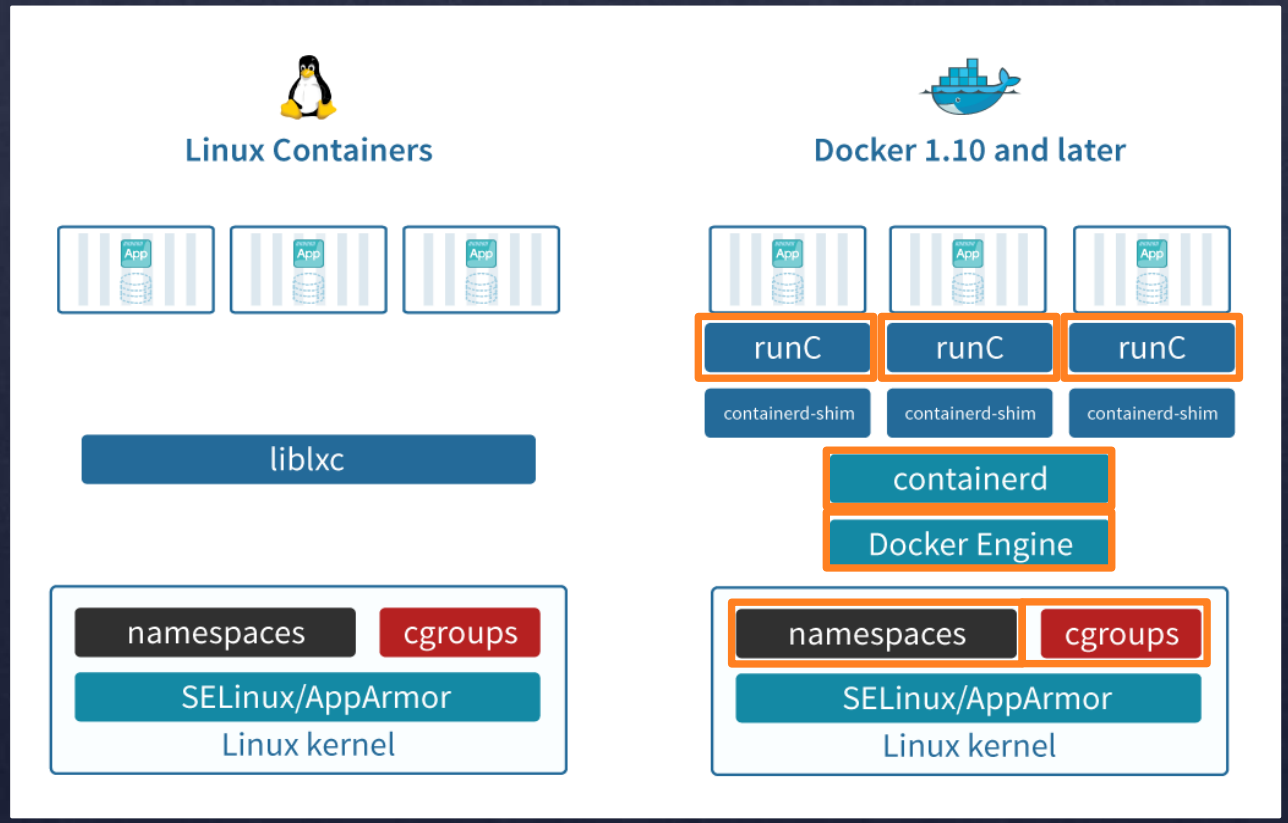



## Terms:

- Virtualization
- Containerization

# Linux Container

- Two most important building blocks
  - **namespaces**
  - **cgroups (control groups)**



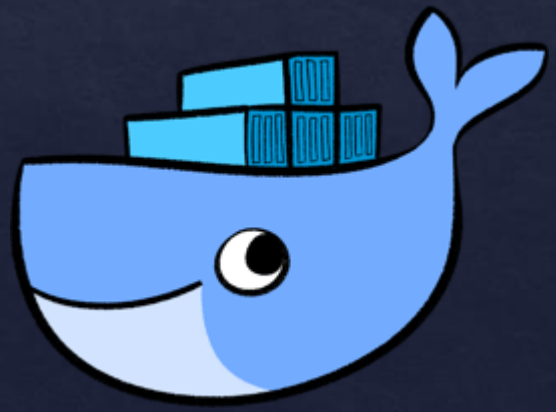
# Contents Cont.

5. Container Concepts and Fundamentals
6. Docker Hands On
7. Kubernetes Hands On

# Docker Fundamental

- An Open-Source GO framework
- Solomon Hykes starts Docker as an internal project within dotCloud in 2013

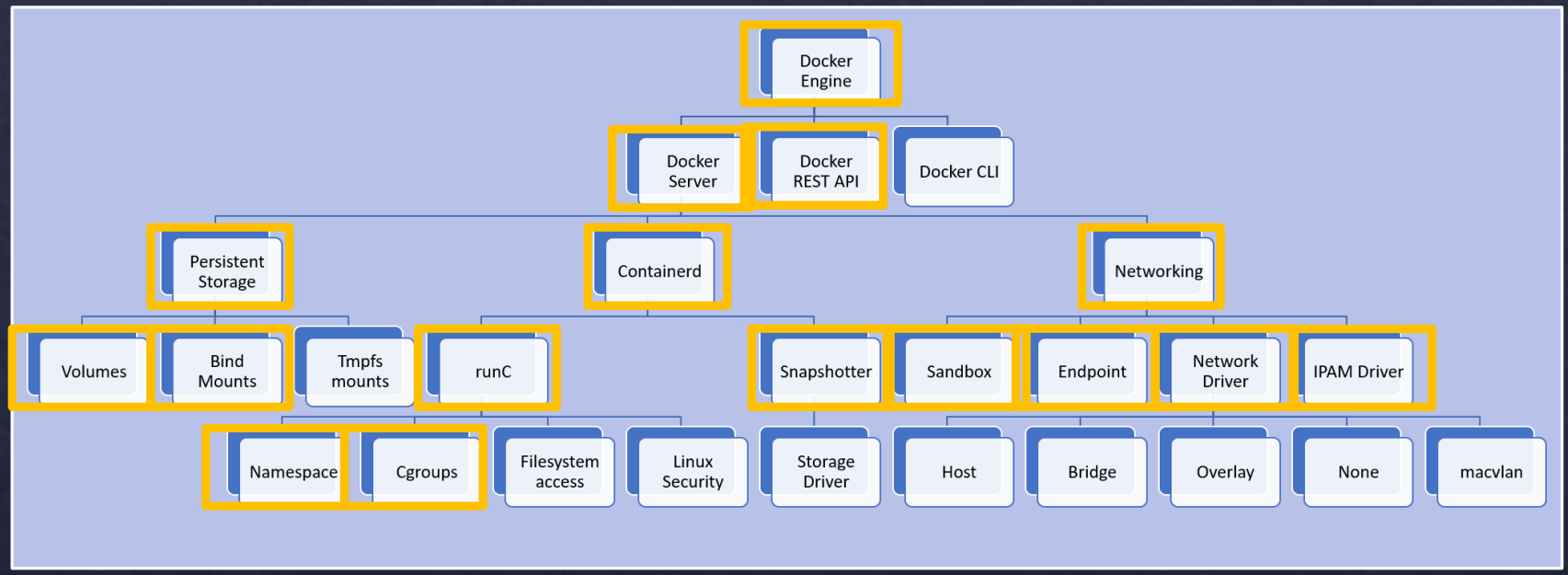

- Whale is the one who won the contest



# Why GO

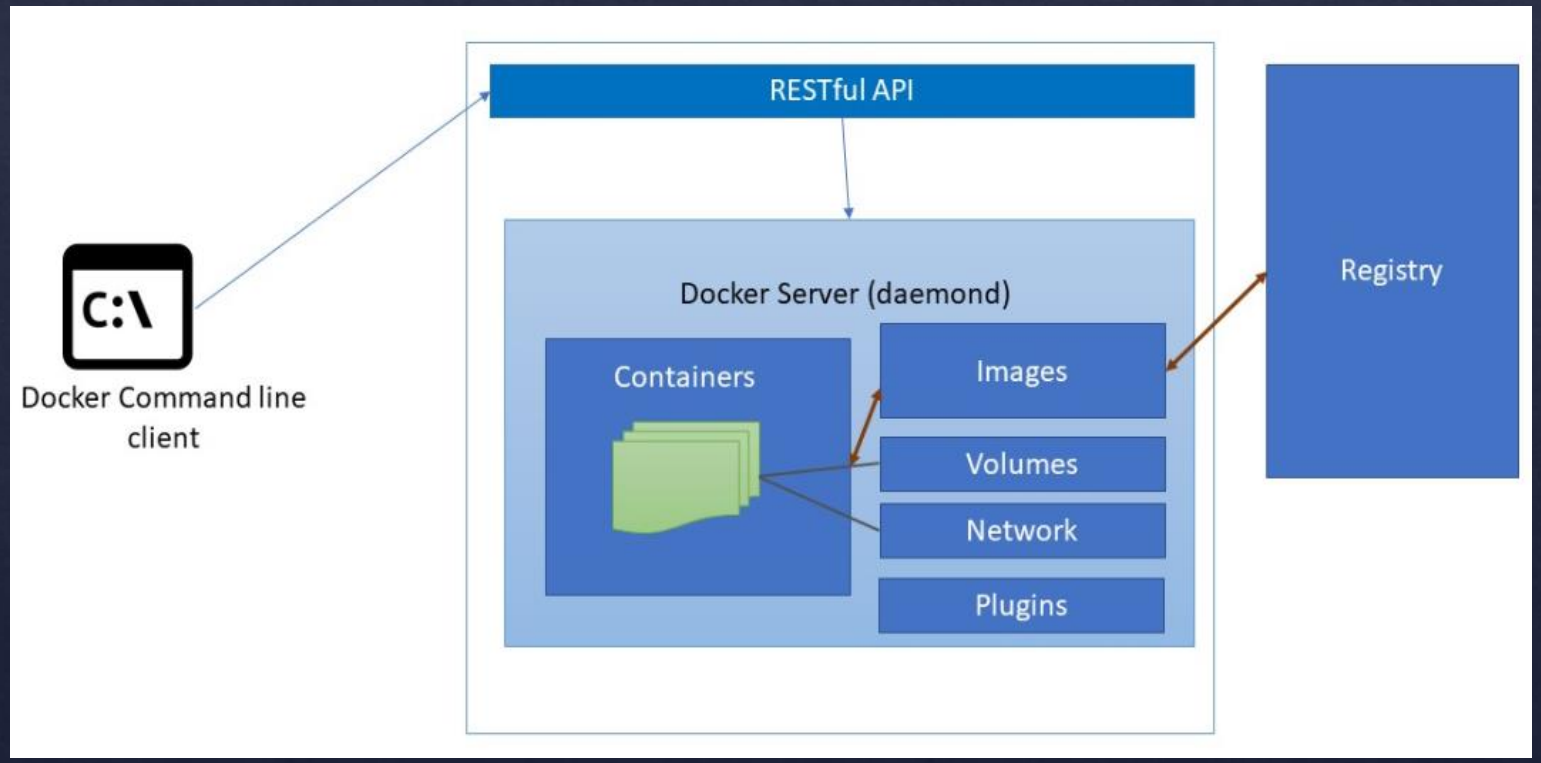

- Simplicity is Complex
- Medium By Example

# Overall Hierarchy of Docker



# Docker Engine

- The heart of Docker is the docker engine
- Constists of:
  1. Docker Server
  2. RESTFul API
  3. Docker CLI



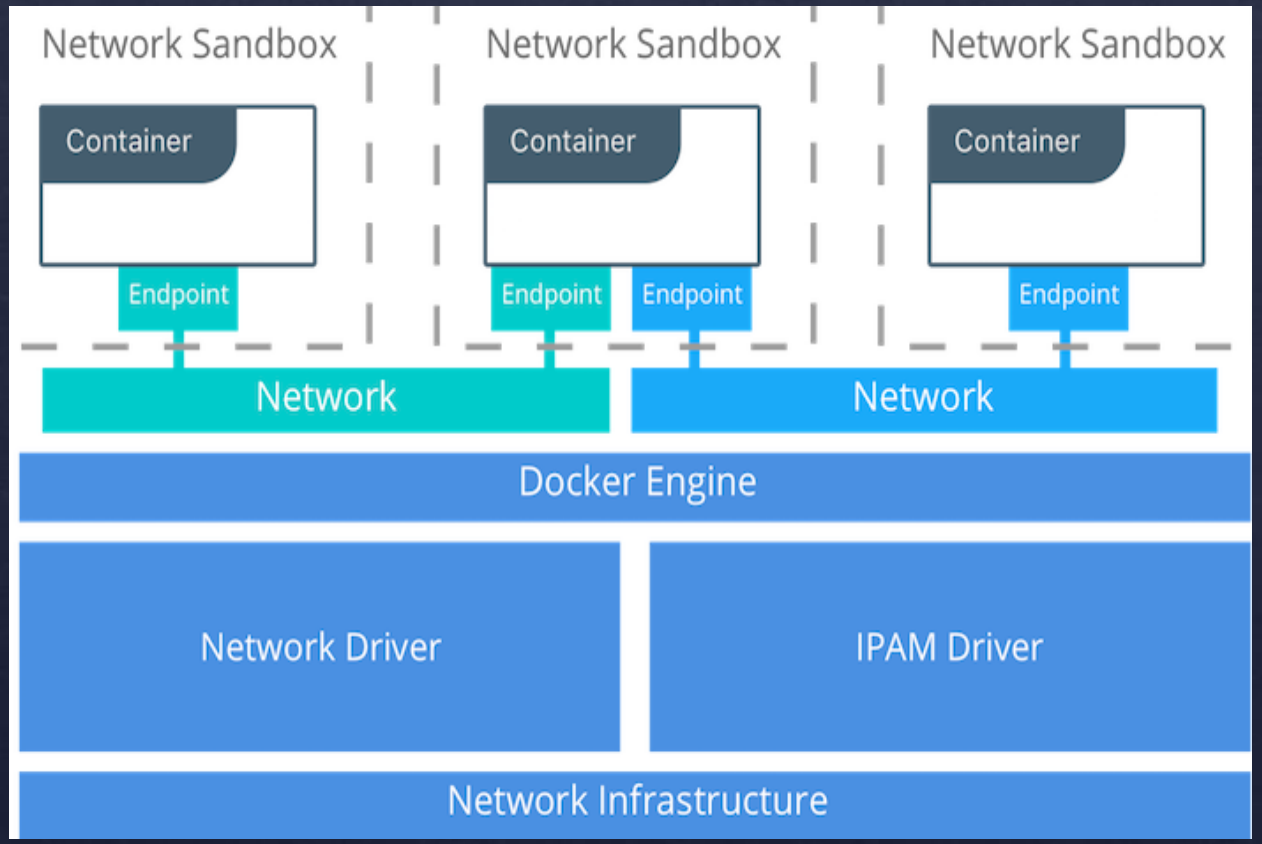
# Persistent Storage

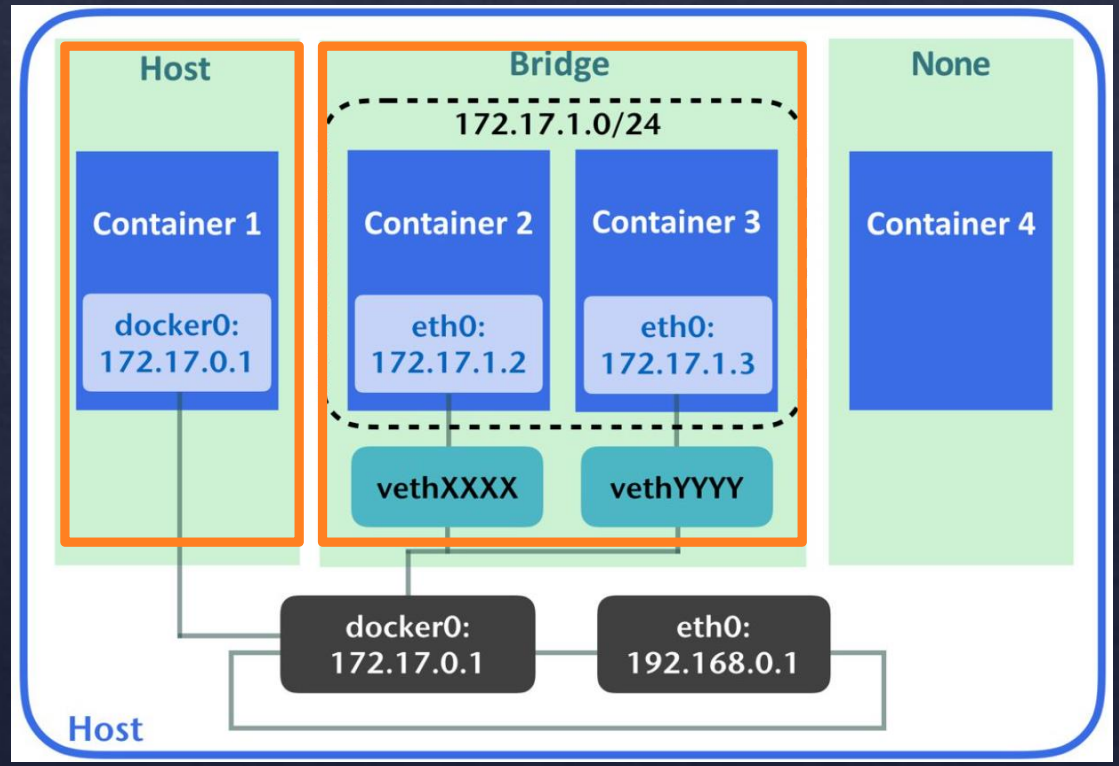
- It is used for making data persistent outside the container
- Constists of:
  - Volumes
    - Maintaining volumes
      - ✓ They are managed using the API
      - ✓ They can be shared amongst multiple containers
      - ✓ They can work on both linux, windows or event remote hosts or cloud providers
  - Bind Mounts
    - You can mount a file or directory from the host machine to the docker container using bind mounts
  - tmpfs Mounts
    - This can be used to store temporary data
    - Data is kept on the temporary storage area of the host machine.

# Networking

- Constists of:
  - Sandbox: It manages the container…
    - Routing tables
    - Interfaces
    - DNS
  - Endpoint
    - Endpoints Join a Sandbox to a network.
  - Network Driver: used by the docker engine to connect to the actual network infrastructure
    - Remote: Container uses Host networking
    - Native: Host, Bridge, Overlay, None …
  - IPAM Driver: Manages IP addresses for Docker

# Networking


Network Sandbox
Network Sandbox
Network Sandbox
Container
Container
Container
Endpoint
Endpoint
Endpoint
Endpoint
Network
Network
Docker Engine
Network Driver
IPAM Driver
Network Infrastructure


Host
Bridge
None
172.17.1.0/24
Container 1
Container 2
Container 3
Container 4
docker0:
172.17.0.1
eth0:
172.17.1.2
eth0:
172.17.1.3
vethXXXX
vethYYYY
docker0:
172.17.0.1
eth0:
192.168.0.1
Host

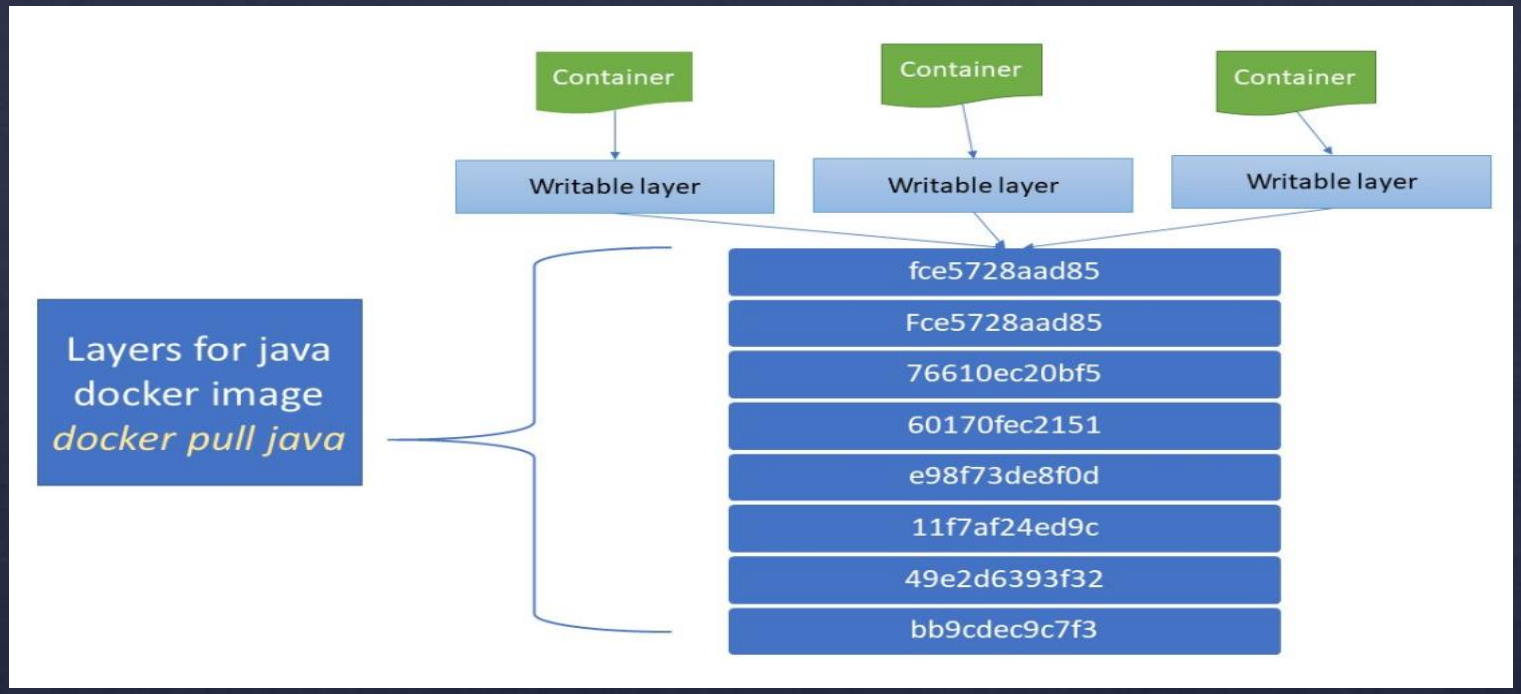
# Containerd: The Heart of Container System

- Docker Engine uses it to create and manage Containers
- It abstracts calls to system or OS specific functionality to run containers on windows, solaris and other Oss
- The scope of containerd includes:
  - Create, start, stop, pause, delete a container, …
  - Functionality for copy on write file systems for containers (Copy on write)
  - Build, push and pull images and management of images
  - Persisting container logs

# Containerd: The Heart of Container System CONT.

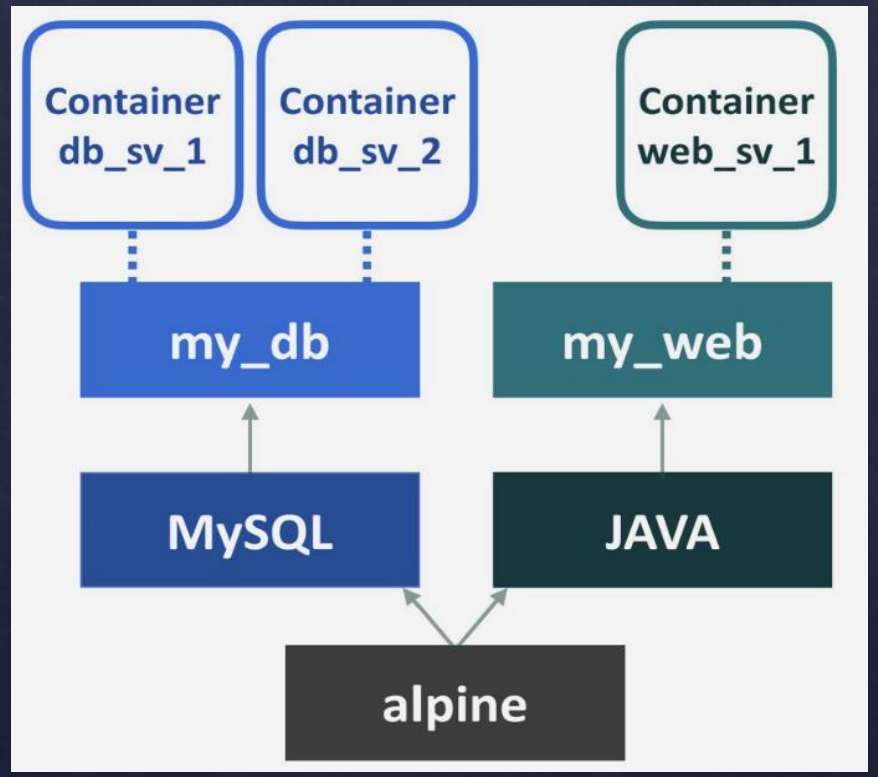
- Docker Engine uses it to create and manage Containers
- Consists of:
  - Container runtime – runC
    - runC is a cli tool that follows the open container initiative(OCI)
    - runC provides GO implementation to create containers using
      - ✓ Namespace
      - ✓ Cgroups
      - ✓ Filesystem access controls
      - ✓ Linux security capabilities
  - Snapshotter
    - A snapshot is a filesystem state
    - Docker containers use a system known as layers
    - Layers allow making modifications to a file system and storing them as a changeset on the top of a base layer
    - A layer is the diff between the snapshots

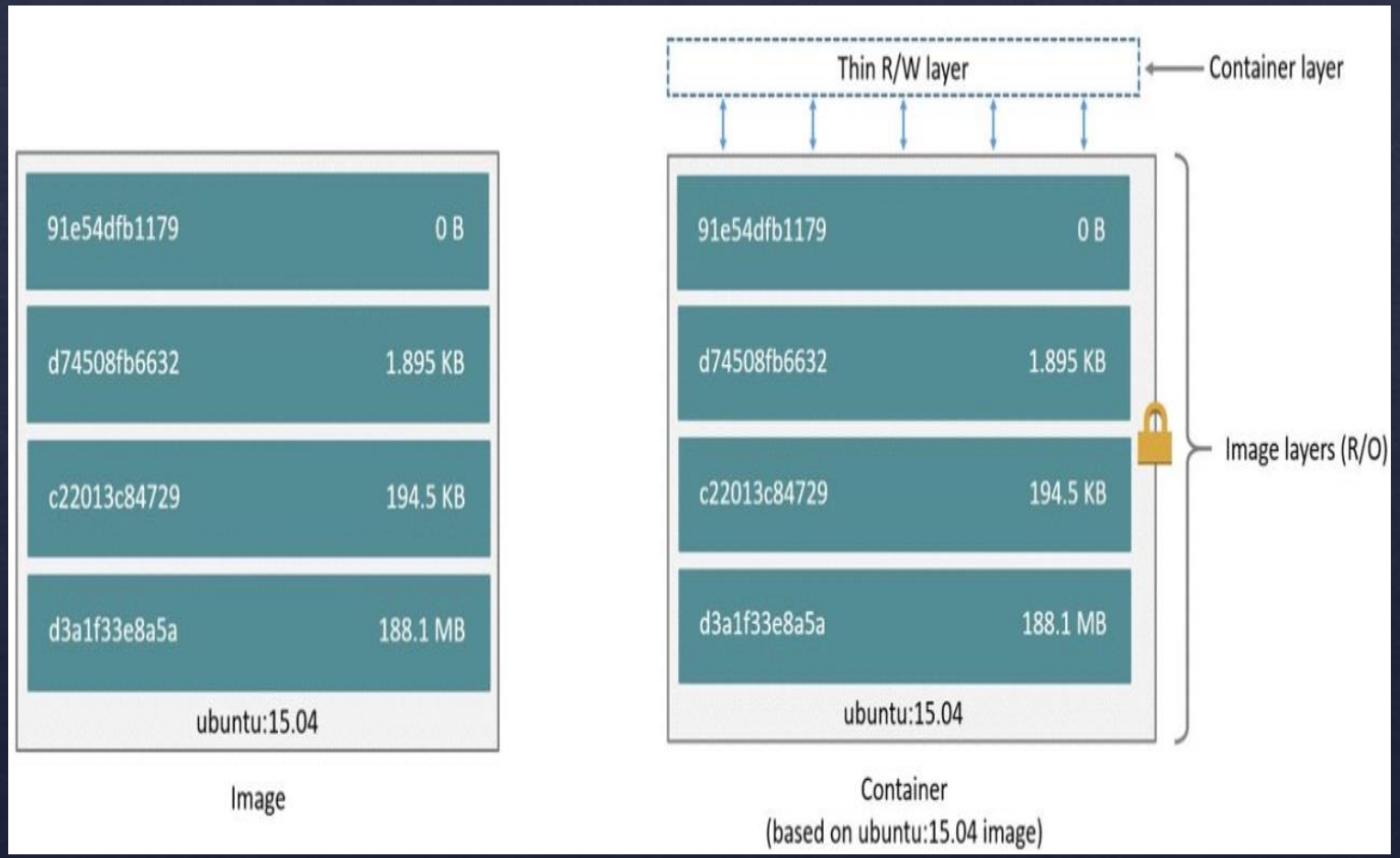
# Snapshotter




- When a container is created
  - Adds a writable layer(Copy on Write) on top of all the layers
    - ➢ All changes are written to this writable layer
  - This writable layer is what differentiates a container from an image.

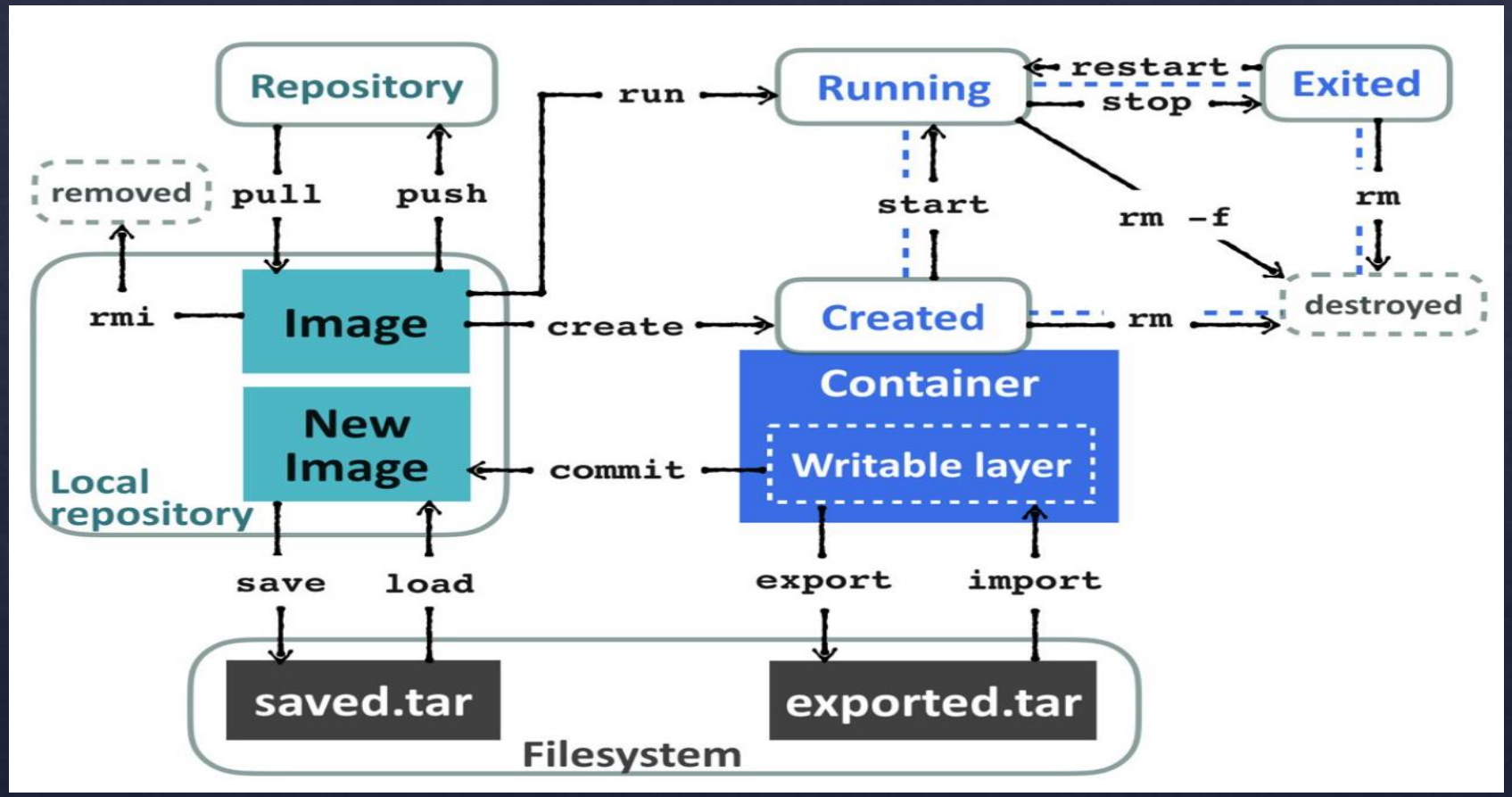
# The Layers


Container
db_sv_1
Container
db_sv_2
Container
web_sv_1
my_db
my_web
MySQL
JAVA
alpine


Thin R/W layer
Container layer
91e54dfb1179 0 B
d74508fb6632 1.895 KB
c22013c84729 194.5 KB
d3a1f33e8a5a 188.1 MB
ubuntu:15.04
Image layers (R/O)
Image
Container
(based on ubuntu:15.04 image)

# Image Vs Container

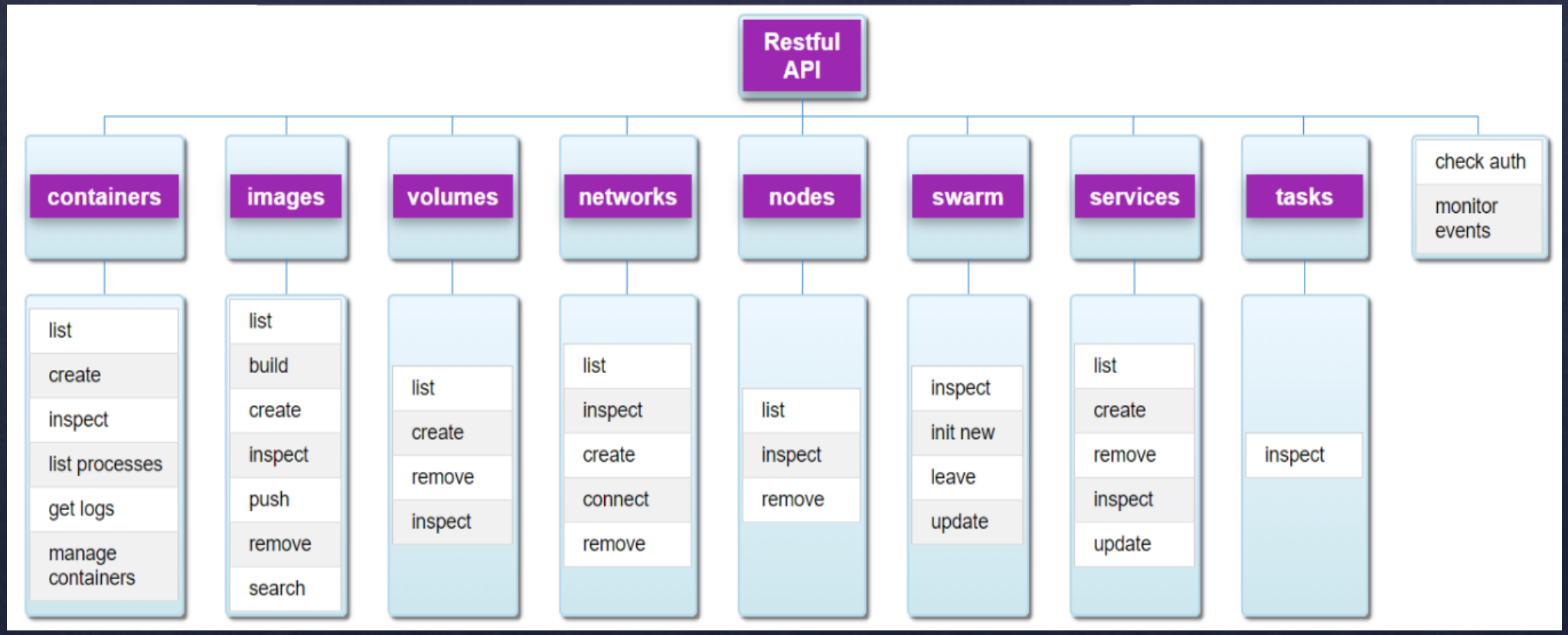
The following diagram depicts the relationship of states between container and images. The captions on the arrows are the corresponding sub-commands of Docker



# The Snapshotter Commands

- `docker commit [CONTAINER]`: Commits the changes of the container layer into a new image
- `docker save --output [filename] IMAGE1 IMAGE2 ...`: Saves one or more images to a TAR archive
- `docker load -i [filename]`: Loads a `tarball` image into the local repository
- `docker export --output [filename] [CONTAINER]`: Exports a container's filesystem as a TAR archive
- `docker import --output [filename] IMAGE1 IMAGE2`: Imports a filesystem `tarball`

# Docker RESTful API



# Glossary

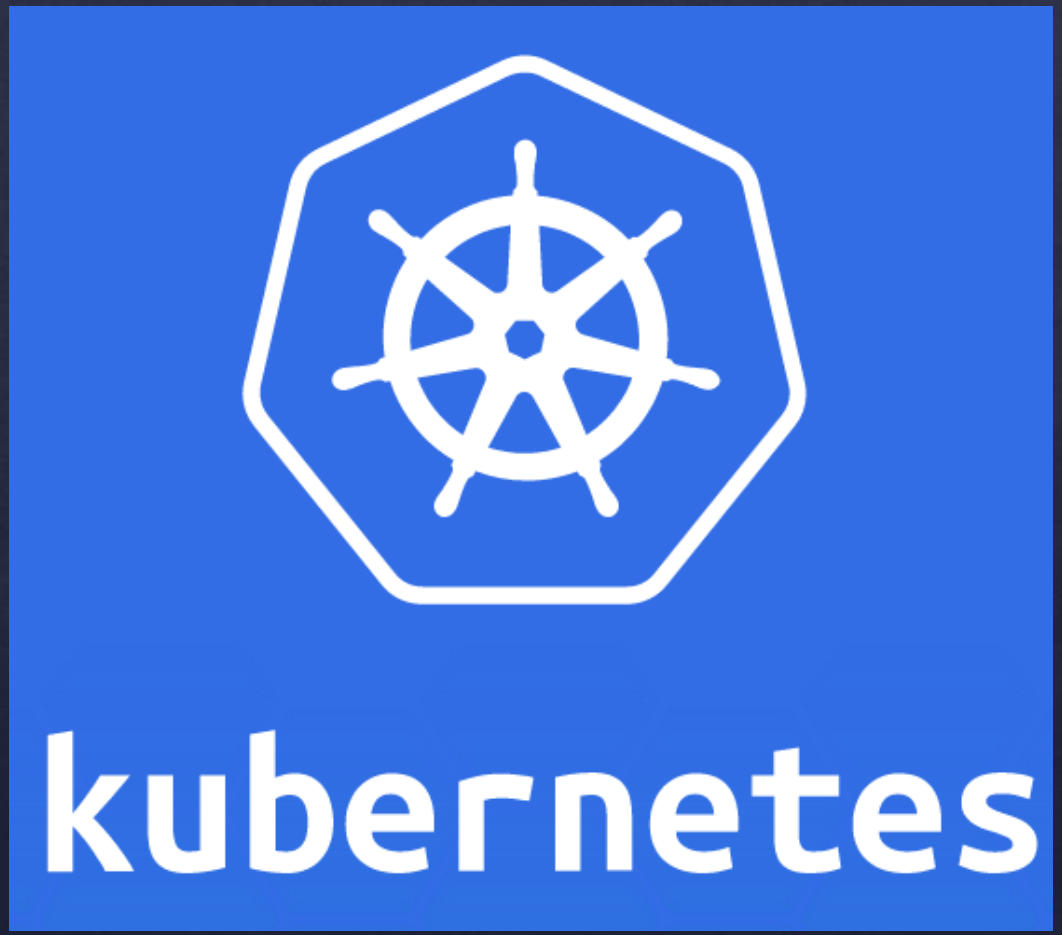
- Docker Client: the binary interacting with the Docker Engine.
- Docker Image: A filesystem (read-only template) used to create a Container (think “the binary”)
- Docker Container: a running image providing a service (think “the process”)
- Host: The computer running the Docker Engine
- Docker Registery: A private or public (Docker Hub) collection of Docker Images
- Docker Hub: A cloud-based repository in which Docker users and partners create, test, store and distribute Docker Images.
- Docker Machine: Provision hosts and install Docker on them
- Docker Compose: Create and manage multi-container architectures
- *Docker Swarm: Orchestrating tool to provision and schedule containers

# Contents Cont.

5. Container Concepts and Fundamentals
6. Docker Hands On
7. Kubernetes Hands On

# Kubernetes Fundamental

- An Open-Source GO framework
- Kubernetes ("koo-burr-NET-eez") is
  - Conventional pronunciation of a Greek word
    - ✓ "κυβερνήτης" == Helmsman/Pilot



# Getting Started with Kubernetes

- Questions:

1. Is there a way to build services across multiple machines without dealing with cumbersome network and storage settings ?

2. Is there any other easy way to manage and roll out our microservices by different service cycle ?

# Understanding Kubernetes

- Kubernetes is a **platform** for managing application containers across multiple hosts.
- Same as the nature of containers: It's designed to run anywhere
  - Clusters
  - Grids
  - Cloud (public, private, hybrid)

# Understanding Kubernetes

- Kubernetes considers most of the operational needs for application containers
  - Auto-scaling
  - Container deployment
  - Persistent storage
  - Container health monitoring
  - Compute resource management
  - High availability

# Understanding Kubernetes

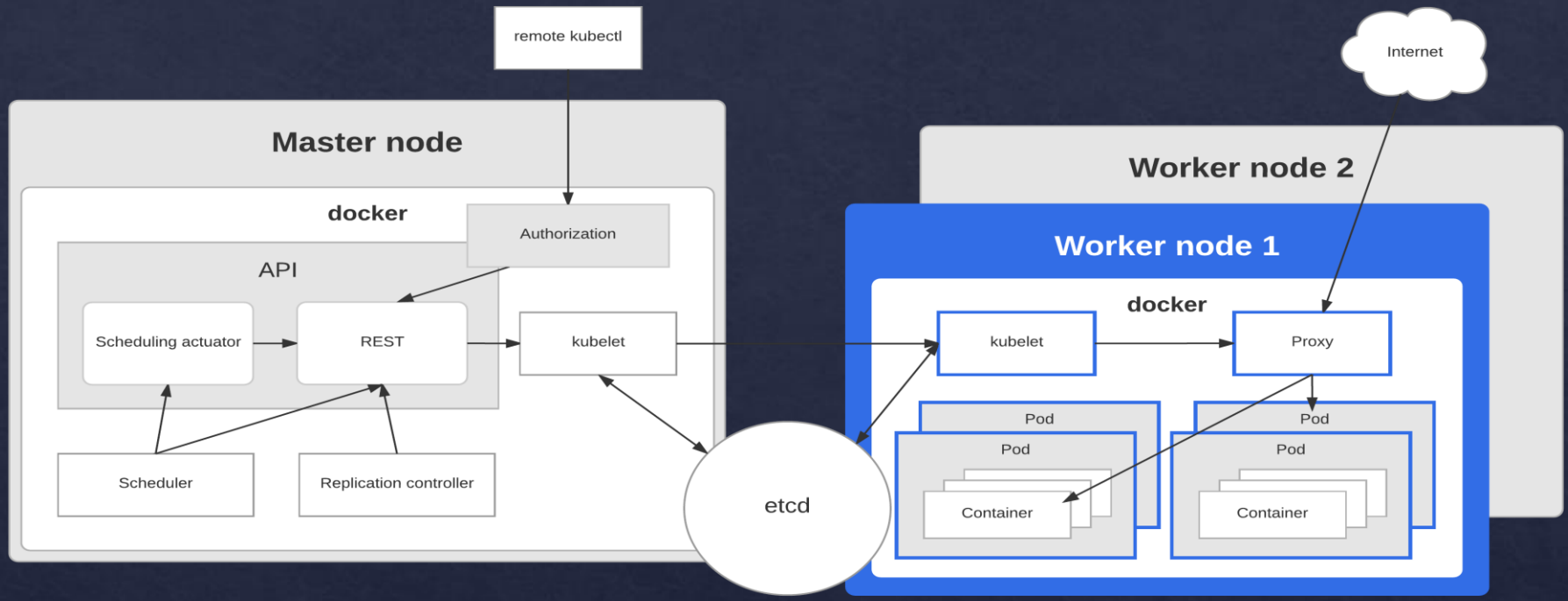
- Kubernetes is a perfect match for microservices
- With Kubernetes, we can create a Deployment to:
  - rollout
  - roll back

  Selected containers…

- Kubernetes provides an optional horizontal pod auto-scaling feature
  - by resource
  - custom metrics
    - ➢ operates on the ratio between desired metric value and current metric value

# Kubernetes Components

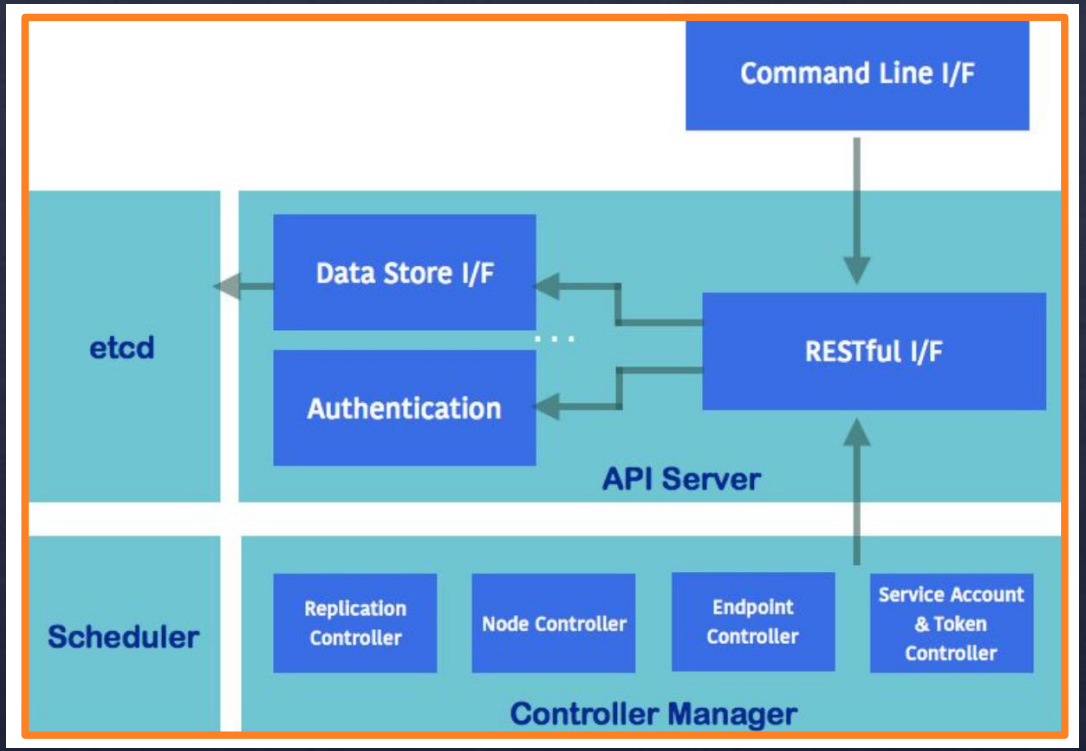
- Kubernetes includes 2 major players




- Masters
  - Is the heart of Kubernetes
  - Controls and schedules all the activities in the cluster
- Nodes
  - Nodes are the workers that run our containers

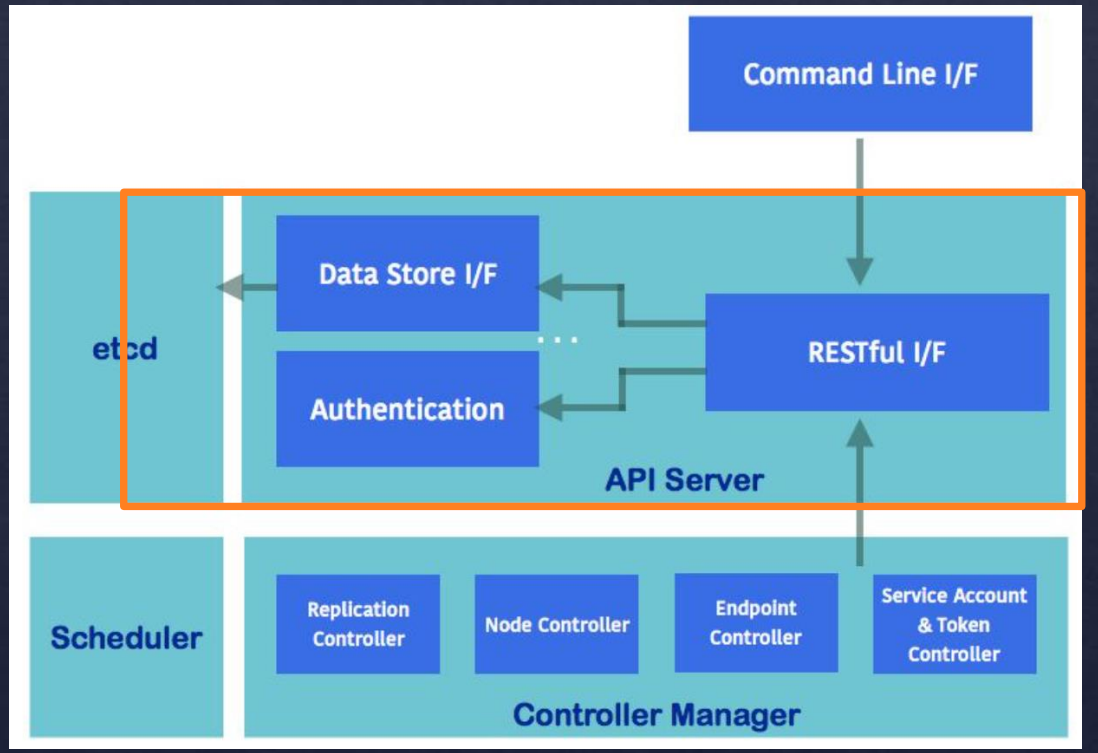
# Master Components

- Includes
  1. API server
  2. Controller Manager
  3. Scheduler
  4. Etcd
- All components can run on different hosts with clustering
  - However, from a learning perspective, we'll make all the components run on the same node.



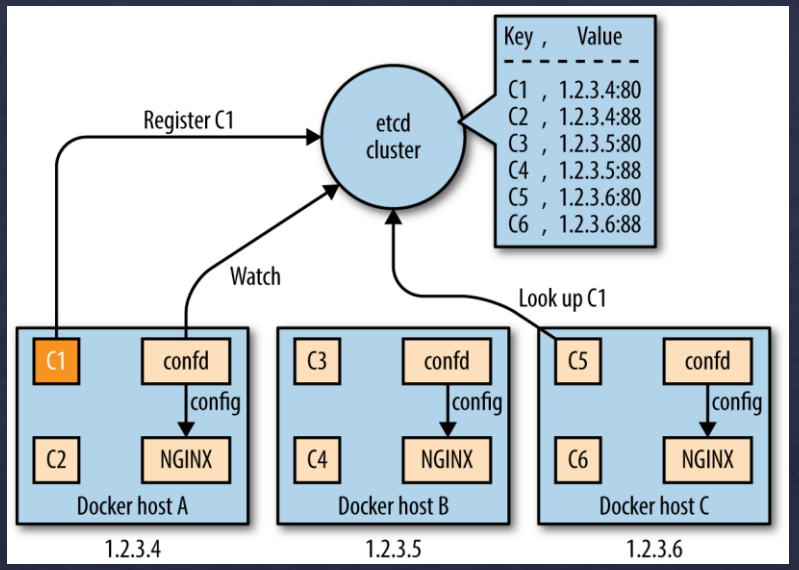
# API server (kube-apiserver)

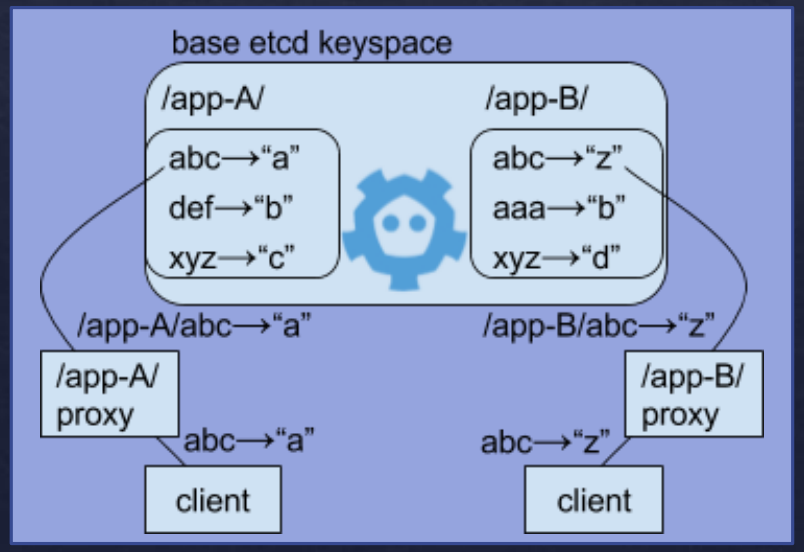
- Provides an HTTP/HTTPS server
  - provides a RESTful API for all the components in the Kubernetes master
- API server reads and updates etcd



# ETCD

- etcd is an open source distributed key-value store
- Kubernetes is distributed
  - So, it needs a distributed database

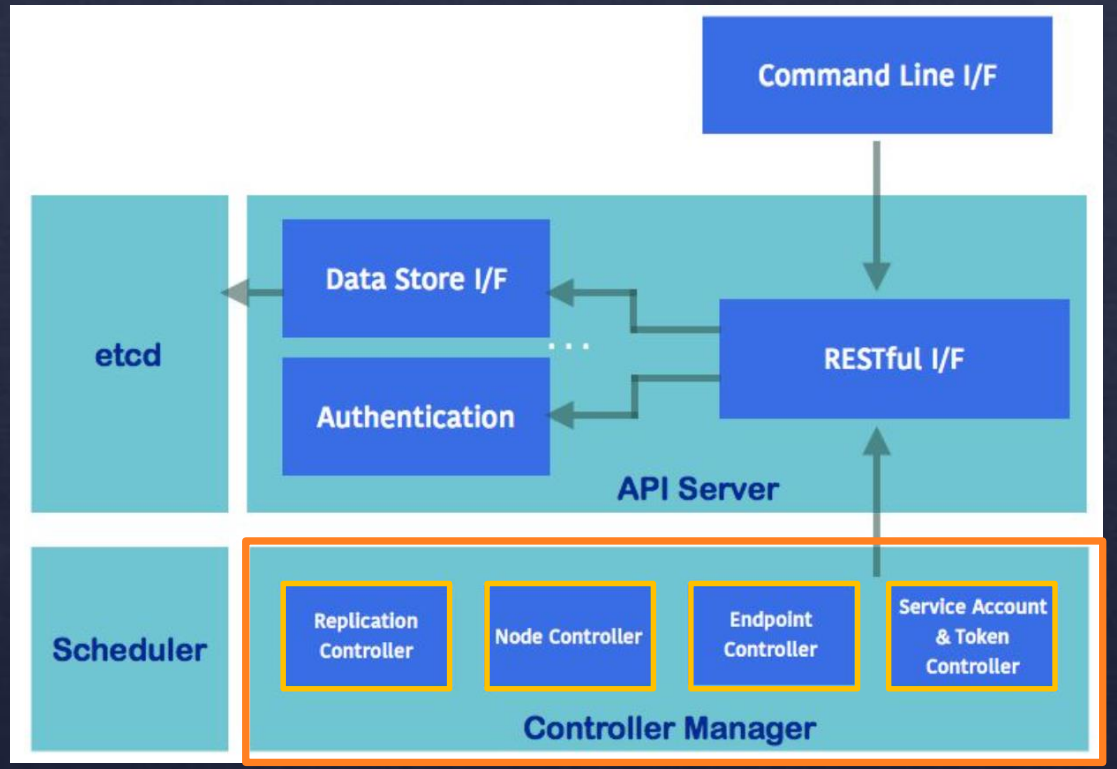



- Kubernetes uses etcd as a key-value database store for
  1. Storing the configuration of the Kubernetes cluster in etcd.
  2. Storing the *actual* state of the system and the *desired* state
- Any node crashing or process dying causes values in etcd to be changed.

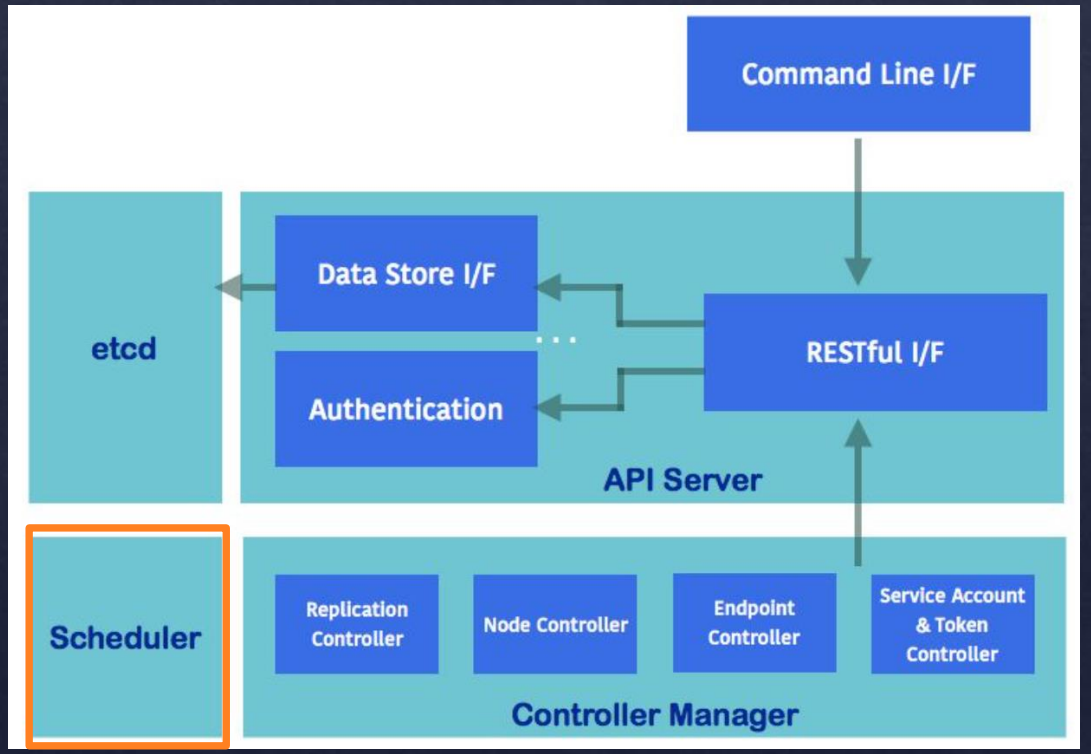
# Controller Manager (kube-controller-manager)

1. Ensures all the ReplicationControllers == Desired container amount
2. Node Controller Manager responds when the nodes go down, it will then evict the pods
3. Endpoint Controller is used to associate the relationship between services and pods
4. Service Account and Token Controller are used to control default account and API access tokens.



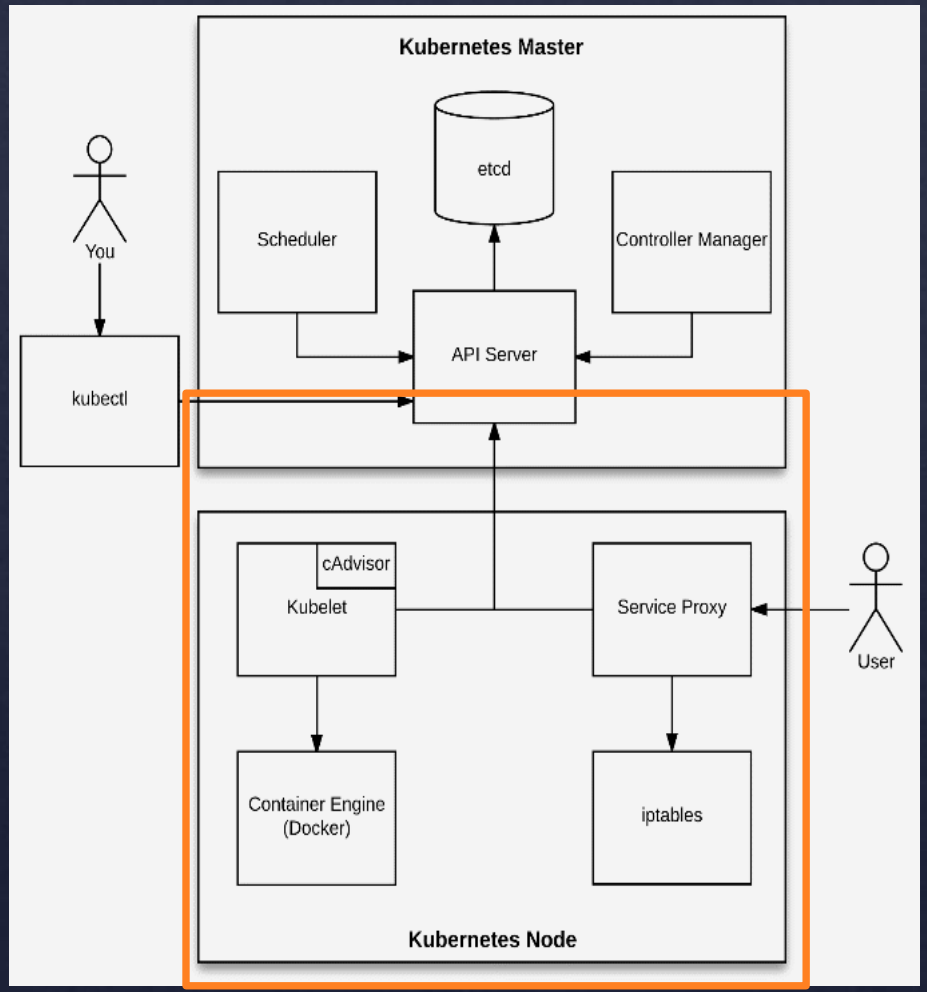
# Scheduler (kube-scheduler)

- Decides which node is suitable for pods to run on according to
  - resource capacity
  - balance of the resource utilization on the node
- It also considers spreading the pods in the same set to different nodes.



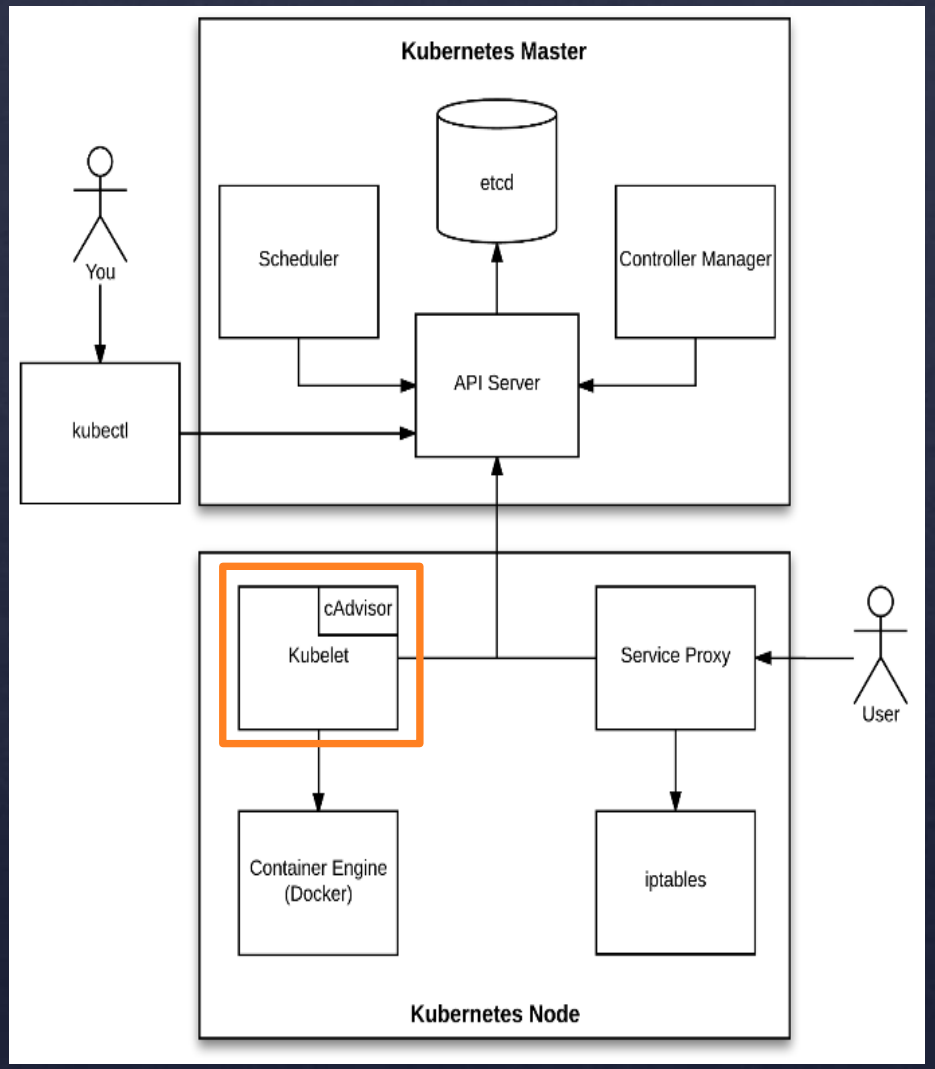
# Node Components

- Node components need to be provisioned and run on every node, in order to:
  - report the runtime status of the pod to the master.



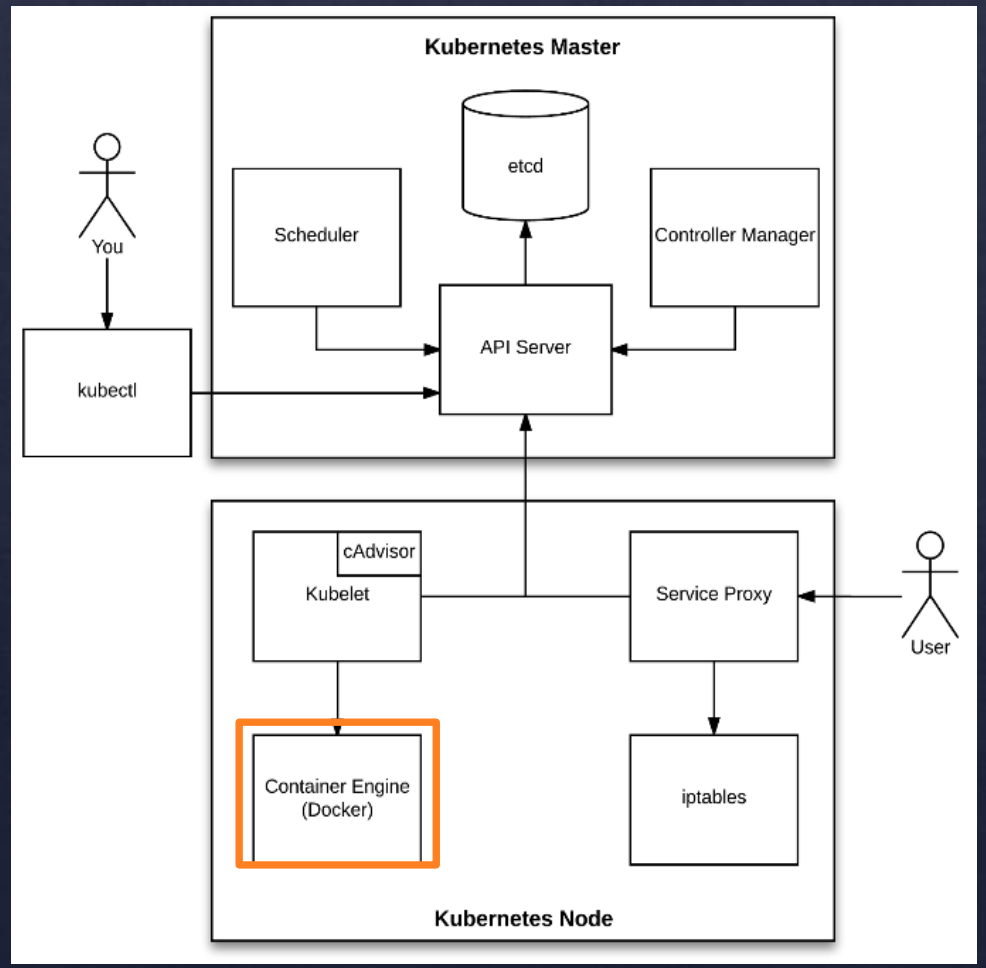
# Kubelet

- Is a major process in the nodes
  - Reports node activities to kube-apiserver periodically, such as:
    - pod health
    - node health
- It runs containers via container runtimes, such as Docker or rkt.



# Docker Engine

- Kubernetes uses Docker as a default container engine.



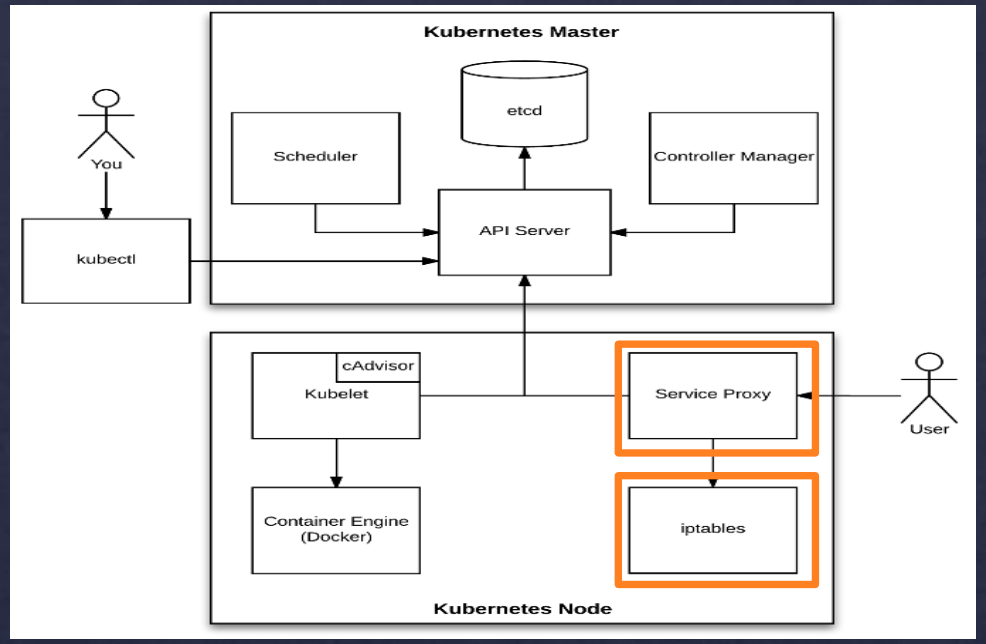


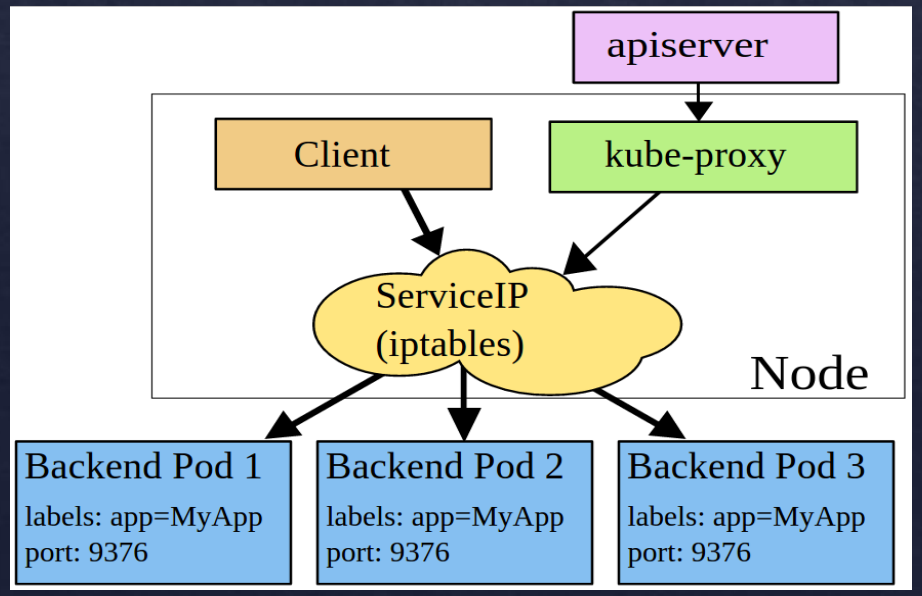
# CAdvisor

- The Kubelet ships with built-in support for cAdvisor
- It
  - Collects Processes
  - Aggregates Processes
  - Exports metrics
    - CPU
    - Memory
    - File
    - Network Usage

# Proxy (kube-proxy)

- Handles the routing between pod load-balancer and pods
  - (a.k.a. service)
- It also provides the routing from outside to service



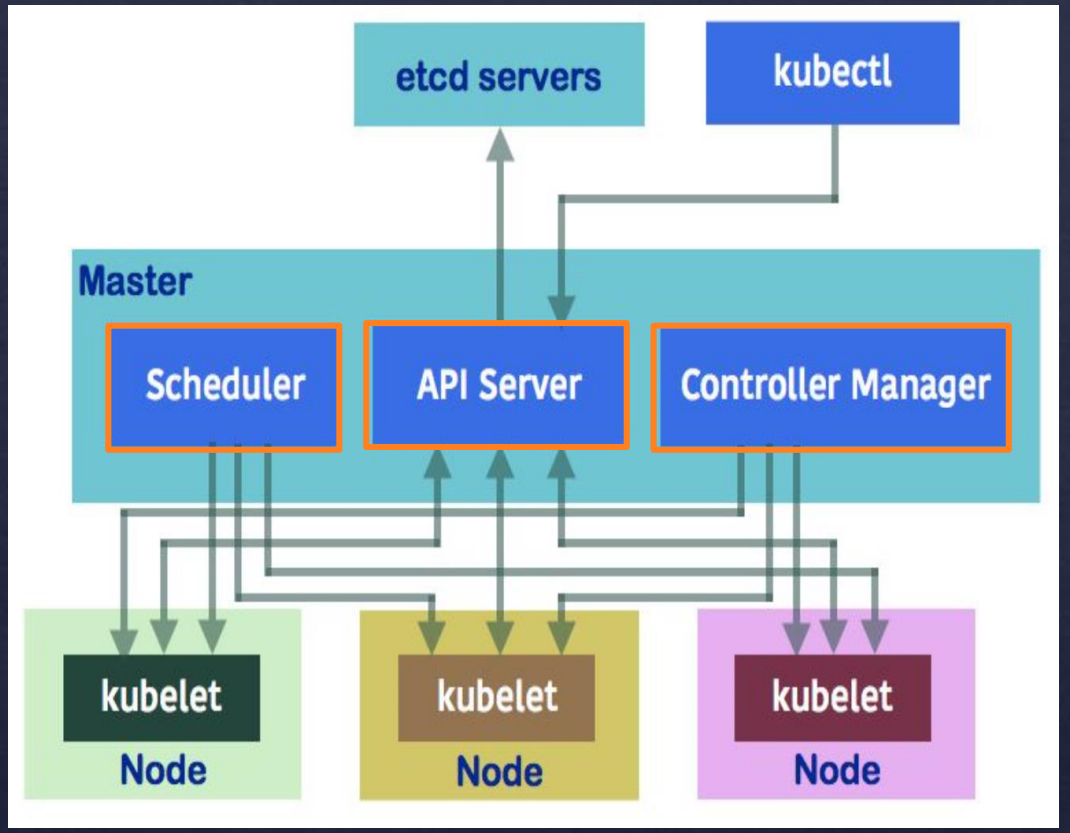

- There are two proxy modes
  - Userspace
    - Large overhead by switching kernel space and user space
  - Iptables
    - Default proxy mode
    - It changes iptables **NAT** in Linux to achieve routing TCP and UDP packets across all containers.

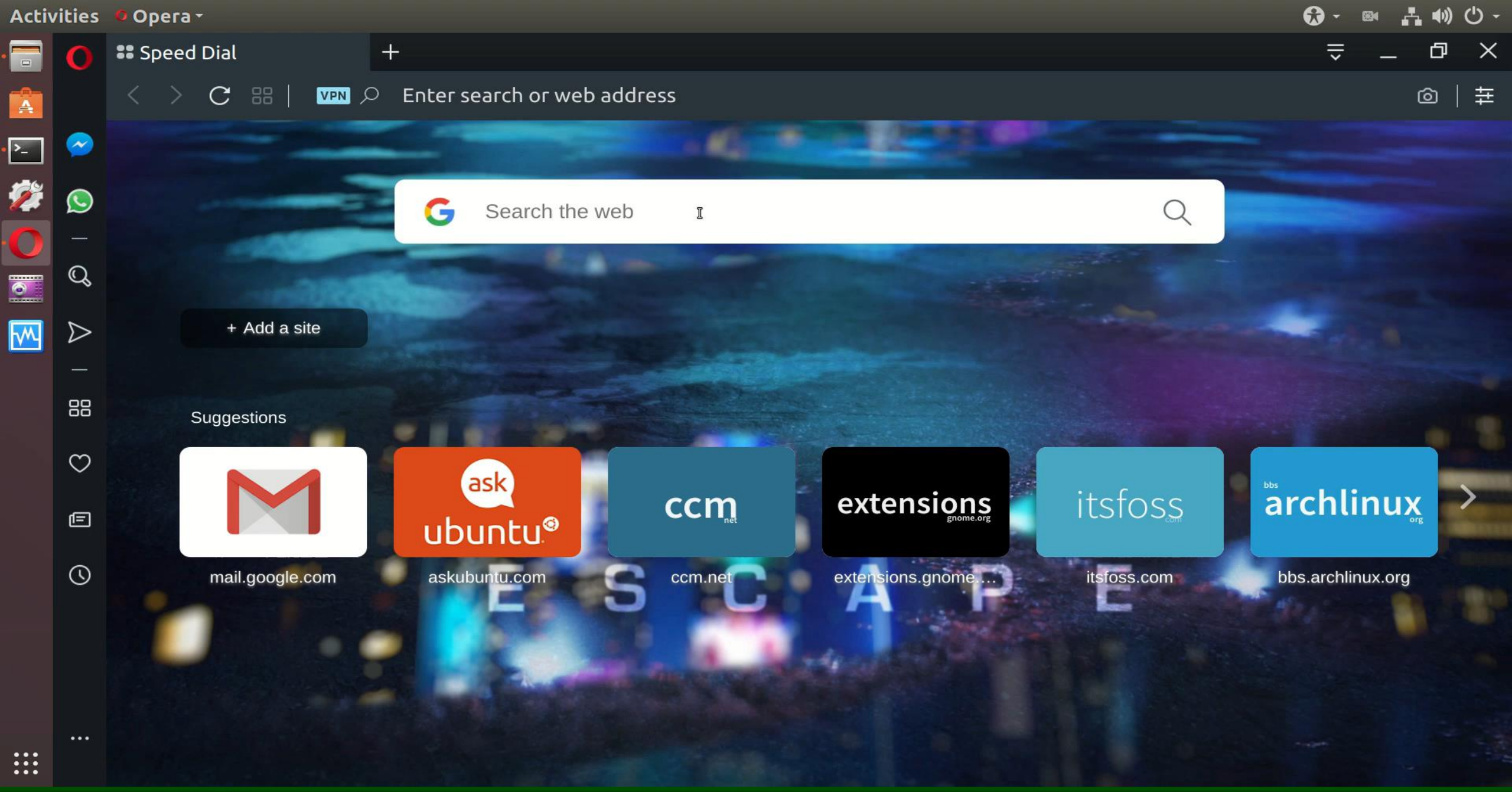
# Interaction between Kubernetes master and nodes

- **Scheduler** determines which node should be assigned in order to :
  - Run pods

- **Controller Manager** monitors the running tasks and responds if any undesired state occurs

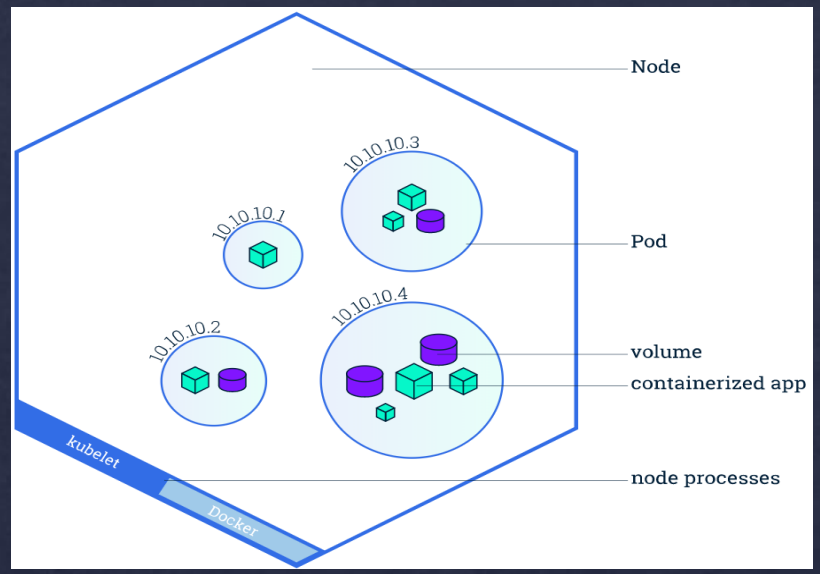
- The **API server** responds to the request after:
  - The client uses **kubectl** to send requests to the API server
  - fetches the logs from pods by kubelet
  - pushes the object information from etcd
  - pulls the object information from etcd



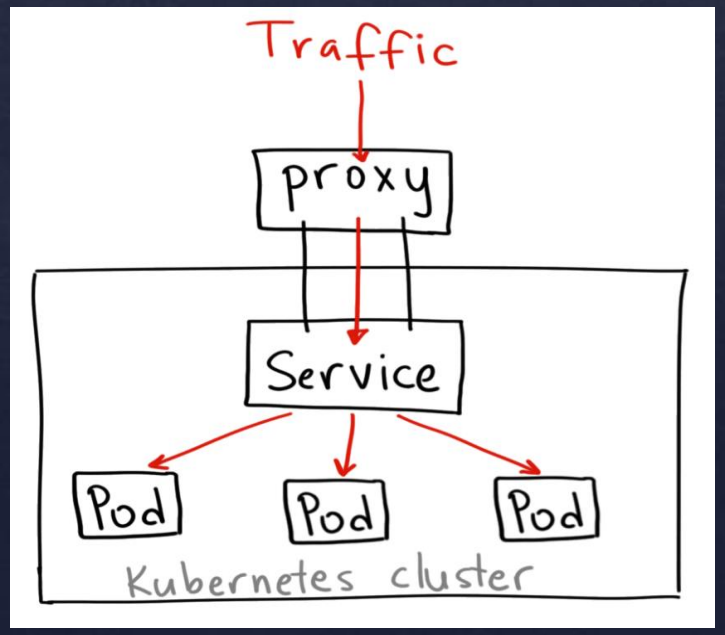

Activities
Opera
Speed Dial
VPN
Enter search or web address
Search the web
+ Add a site
Suggestions
mail.google.com
askubuntu.com
ccm.net
extensions.gnome....
itsfoss.com
bbs.archlinux.org

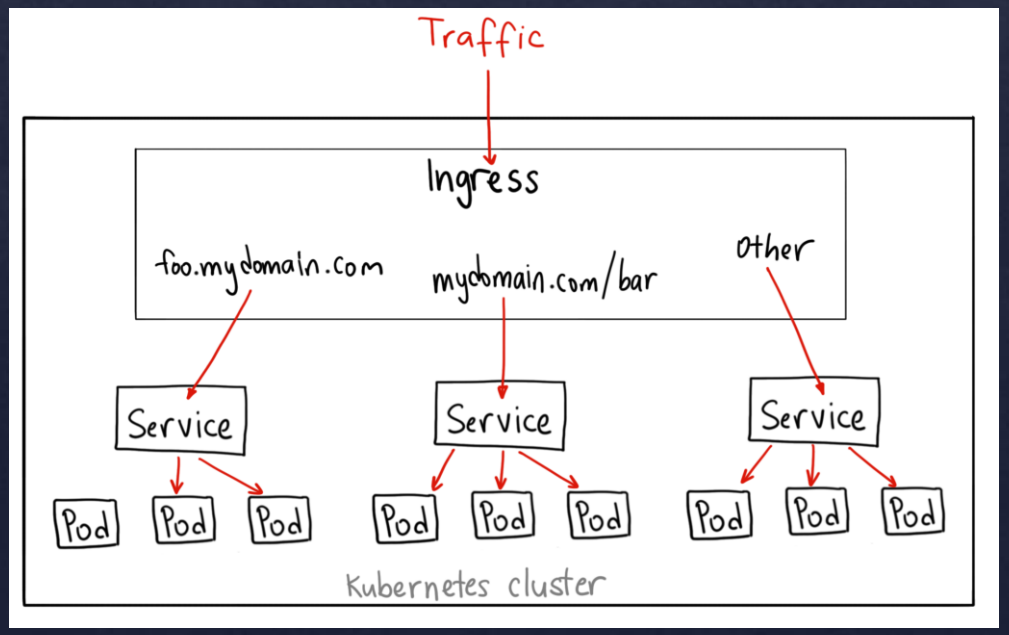
# Objects and Workloads

- Nods
  - A node is a worker machine in Kubernetes
    - ➢ may be a VM/physical machine
      - ✓ depending on the cluster
  - Each node contains the services necessary to run pods
  - is managed by the master components



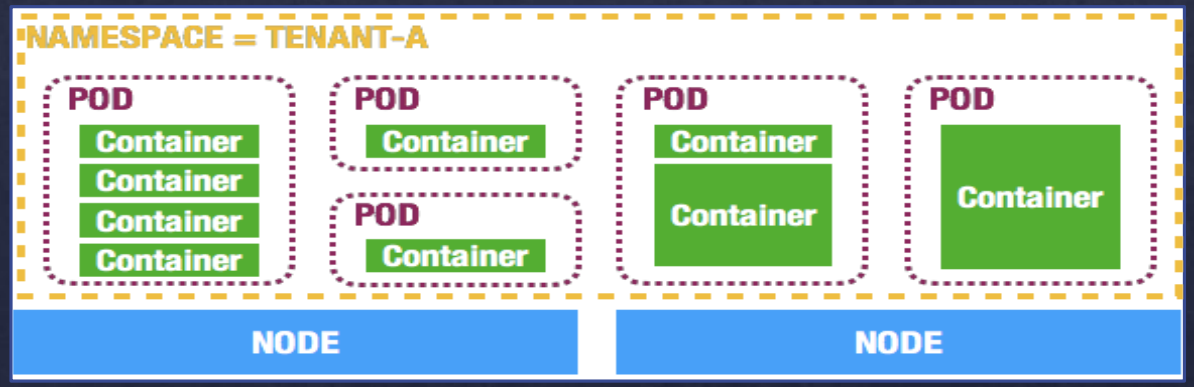

- Services
  - Container runtime
  - Kubelet
  - Kube-Proxy

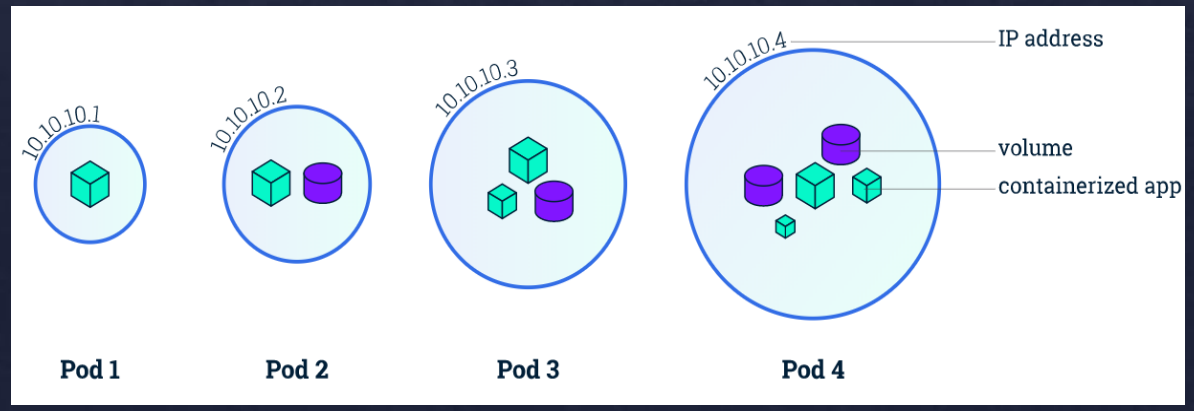


# Objects and Workloads

- Pod
  - Most basic unit
  - One or more Tightly Coupled Containers are encapsulated in an object called a Pod.
  - Works as a single, monolithic application to best conceptualize how the cluster will manage the pod's resources and scheduling.
  - Features
    - Share a life cycle
    - Should always be scheduled on the same node
    - managed entirely as a unit
    - share their
      - ✓ Environment
      - ✓ Volumes
      - ✓ IP Space



# Objects and Workloads

- Jobs
  - Creates one or more Pods and ensures that a specified number of them successfully terminate
    - ➢ A simple case is to create one Job object in order to reliably run one Pod to completion

controllers/job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

```
kubectl apply -f https://k8s.io/examples/controllers/job.yaml
```

```
job "pi" created
```

```
$ kubectl logs $pods
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862
```

# Jobs

controllers/job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

```
kubectl apply -f https://k8s.io/examples/controllers/job.yaml
```

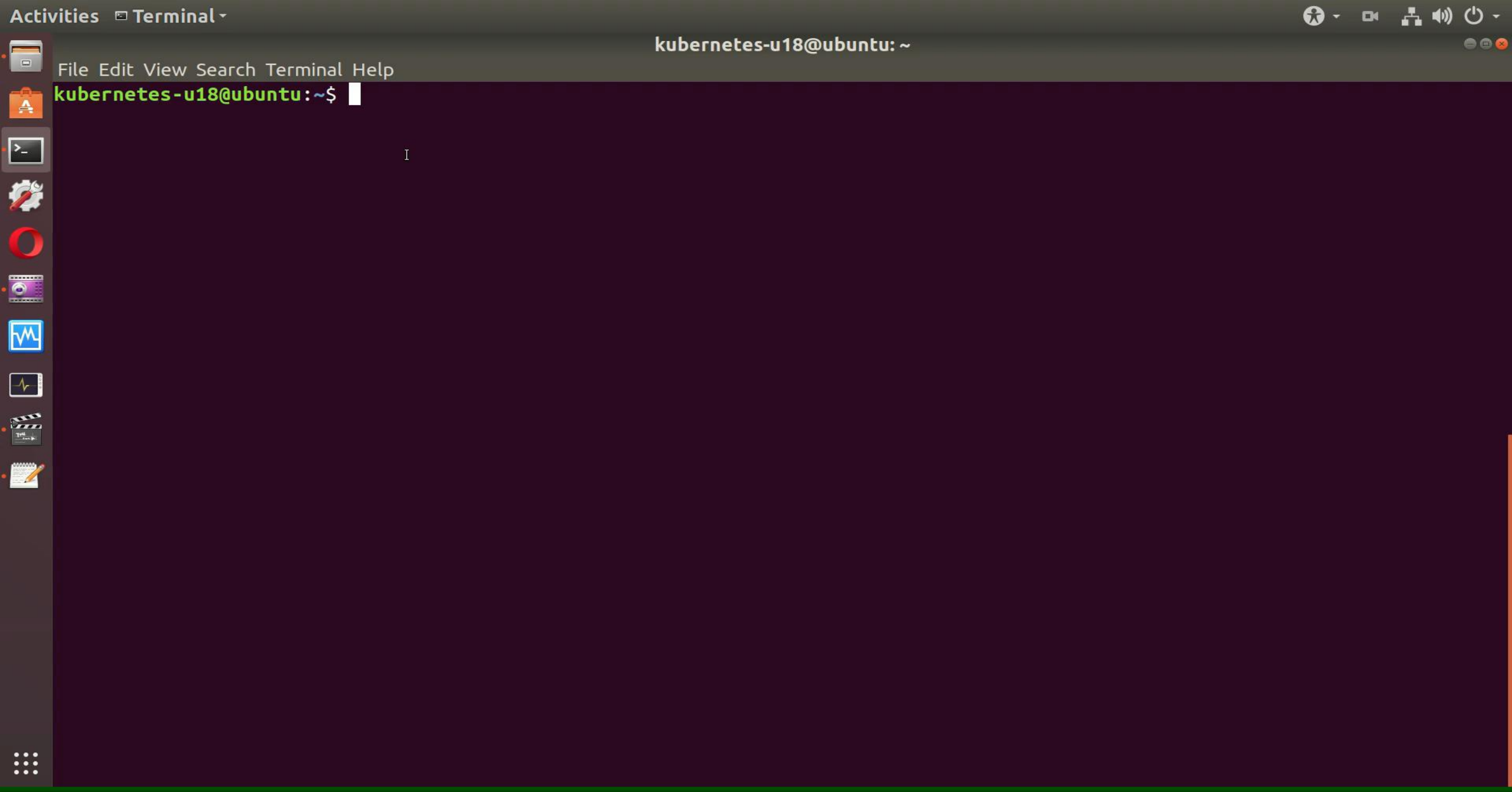
```
job "pi" created
```

```
kubectl describe jobs/pi
```

```
Name:           pi
Namespace:      default
Selector:       controller-uid=b1db589a-2c8d-11e6-b324-0209dc45a495
Labels:         controller-uid=b1db589a-2c8d-11e6-b324-0209dc45a495
                job-name=pi
Annotations:    <none>
Parallelism:    1
Completions:    1
Start Time:     Tue, 07 Jun 2016 10:56:16 +0200
Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
Pod Template:
  Labels:       controller-uid=b1db589a-2c8d-11e6-b324-0209dc45a495
                job-name=pi
  Containers:
   pi:
    Image:      perl
    Port:
    Command:
      perl
      -Mbignum=bpi
      -wle
      print bpi(2000)
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Events:
  FirstSeen    LastSeen    Count    From            SubobjectPath    Type        Reason
  ---------    --------    -----    ----            -------------    --------    ------
  1m           1m          1        {job-controller }                Normal      SuccessfulCreate
```

```
pods=$(kubectl get pods --selector=job-name=pi --output=jsonpath='{.items[*].metadata.name}')
echo $pods
```

```
$ kubectl logs $pods
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862
```


Activities
Terminal
kubernetes-u18@ubuntu: ~
File Edit View Search Terminal Help
kubernetes-u18@ubuntu:~$

◈ Jobs

# Objects and Workloads

- Replication Controllers
  - Is an object that defines a pod template
  - An easy way to distribute load
  - An easy way to increase availability
  - Control parameters to scale identical replicas of a pod horizontally
    - **EX.** increase the desired instance count from 3 to 4
  - Ensures that the number of pods deployed in the cluster == The number of pods in its configuration
    - If a pod or underlying host fails, the controller will start new pods to compensate
      - ✓ Each replication controller has a **desired state**
  - Knows how to create new pods as needed
    - A template that resembles a pod definition is embedded within the replication controller configuration
  - Replication controllers can also perform rolling updates to roll over a set of pods to a new version

# Replication Controllers

- Makes use of label queries

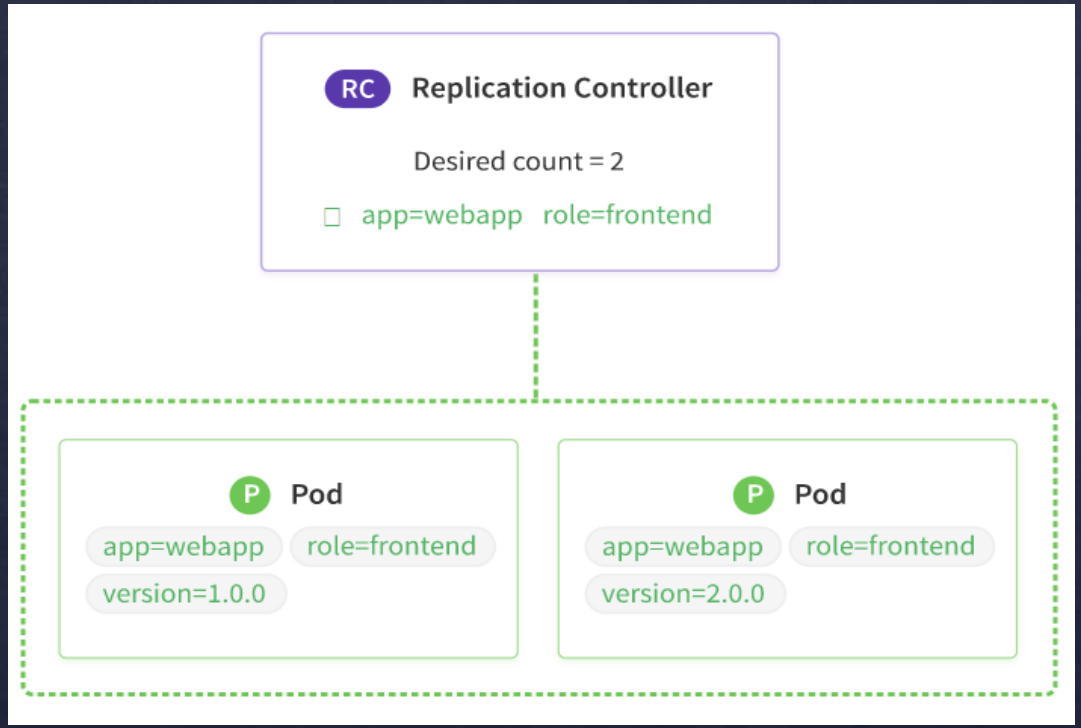
controllers/replication.yaml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

```
kubectl apply -f https://k8s.io/examples/controllers/replication.yaml
```

```
replicationcontroller/nginx created
```

```
kubectl describe replicationcontrollers/nginx
```



# Replication Sets

- Ensures that a specified number of pod replicas are running at any given time
- The link a ReplicaSet has to its Pods is via the Pods' metadata.ownerReferences field
  - Specifies what resources the current object is owned by
  - All Pods acquired by a ReplicaSet have their owning ReplicaSet's identifying information within their ownerReferences field

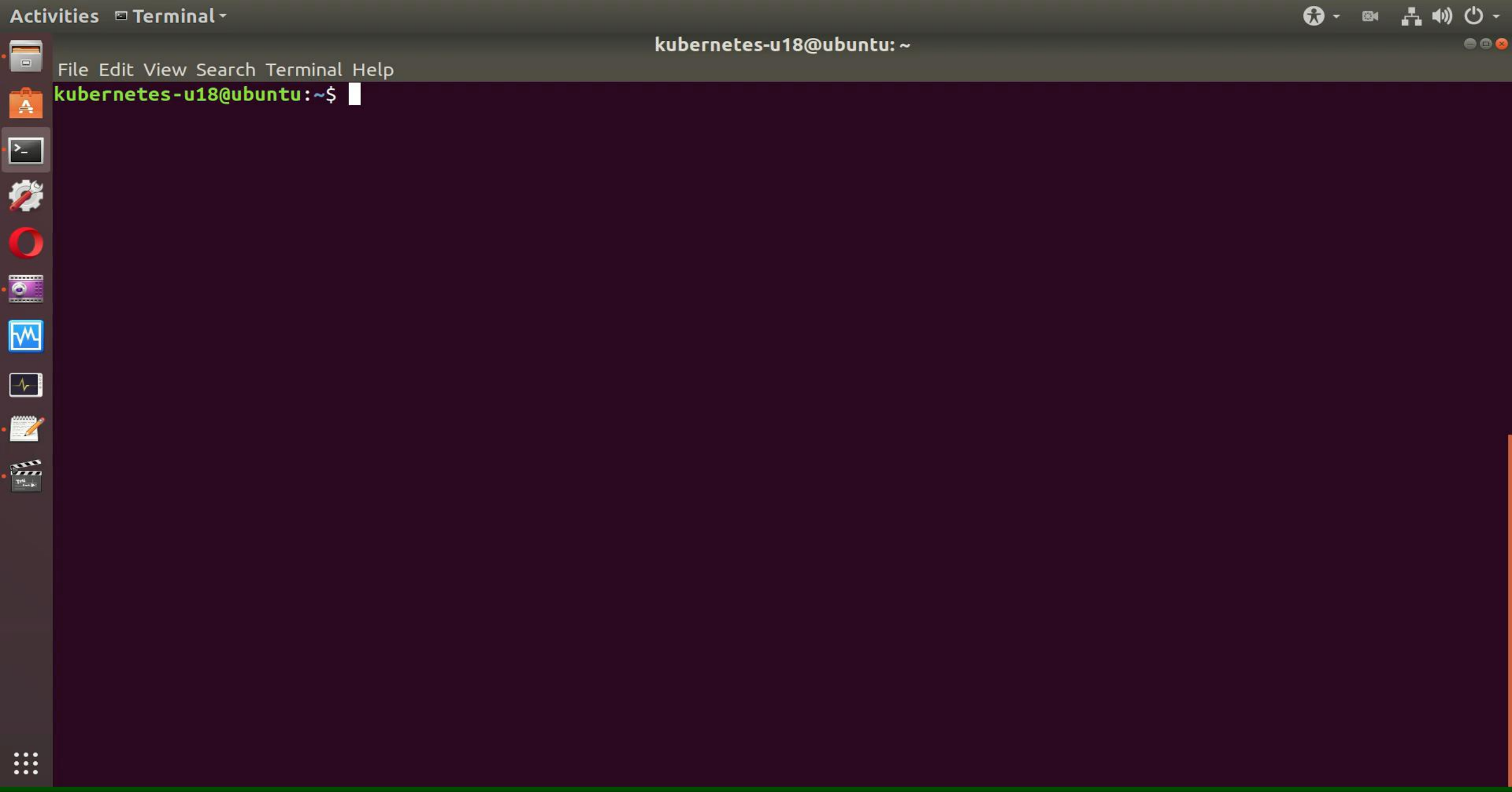
```
kubectl apply -f https://kubernetes.io/examples/controllers/frontend.yaml
```

```
kubectl get rs
```

```
NAME       DESIRED   CURRENT   READY   AGE
frontend   3         3         3       6s
```

controllers/frontend.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```




- ReplicationSets

# Objects and Workloads

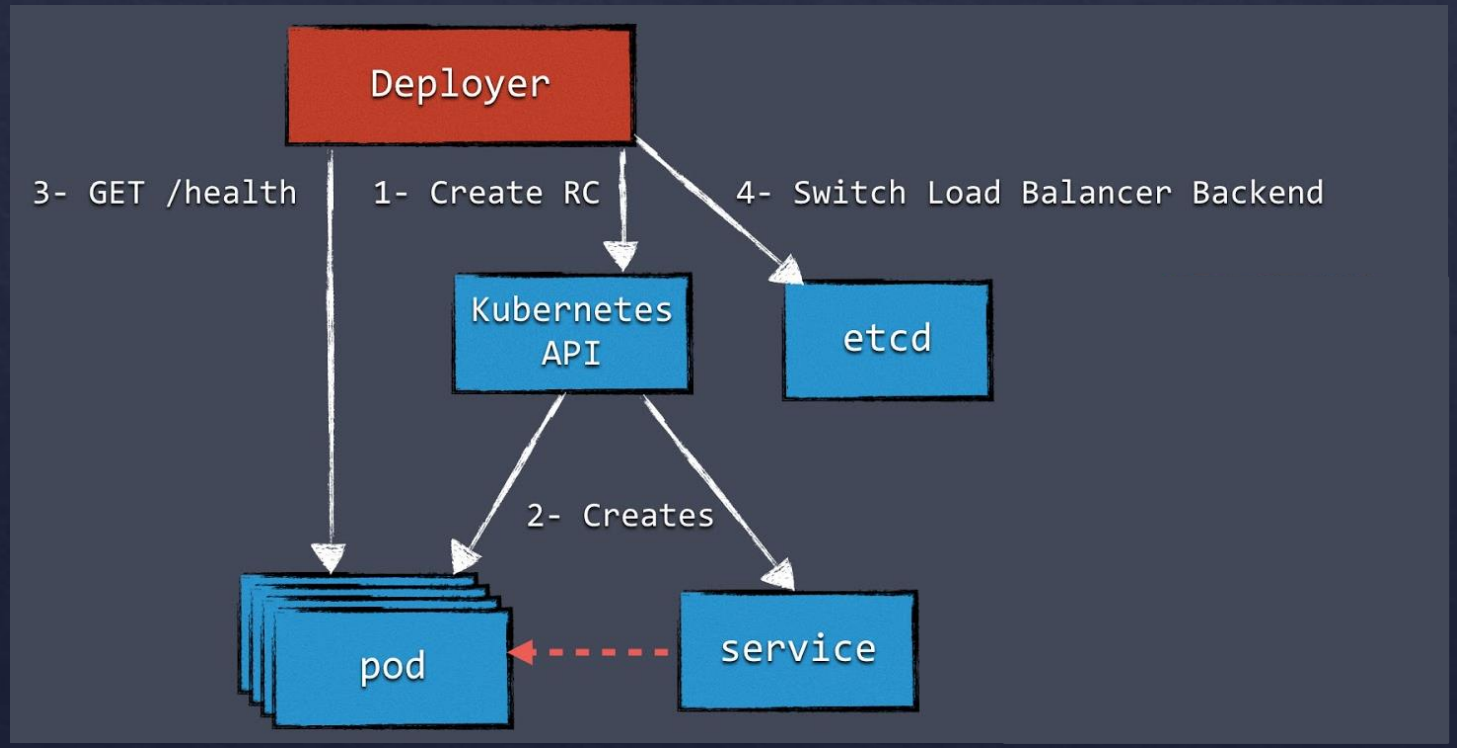
- Replication Controllers and Replication Sets

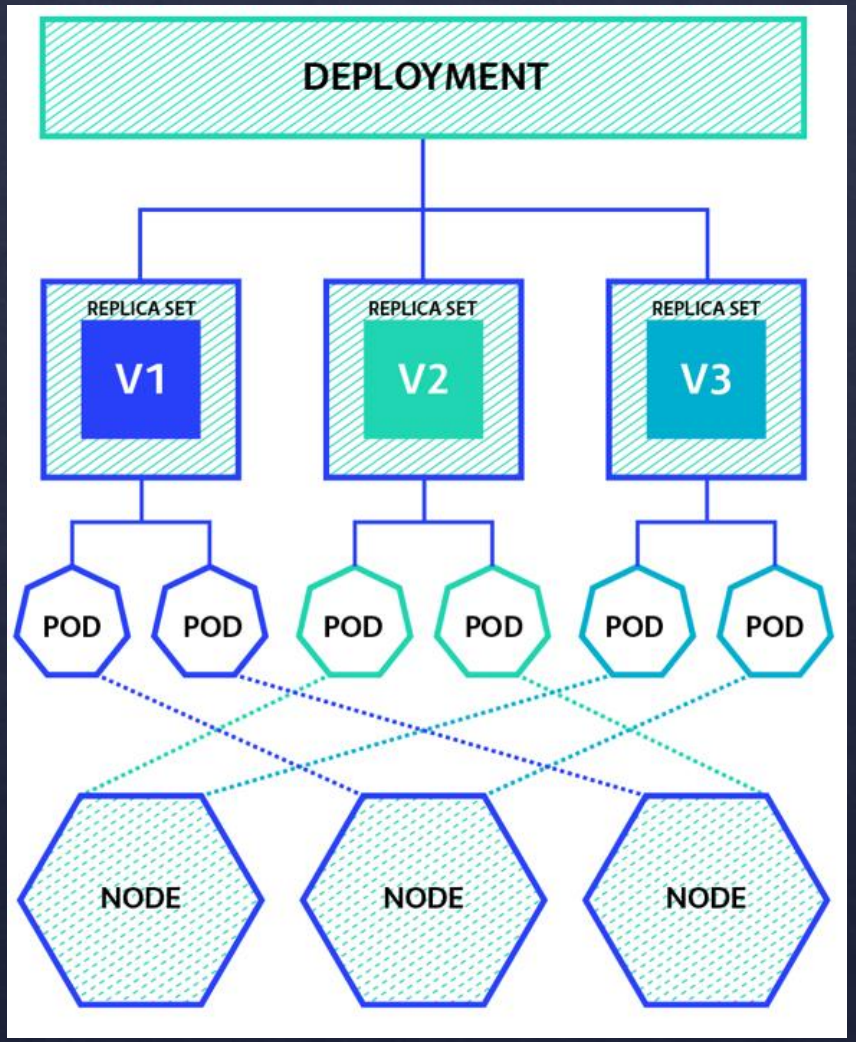
- Managing groups of identical, replicated pods instead of single pods
- These are created from pod templates

- Are the same except the "Roll Over" Feature
- Lack some fine grained life cycle management capabilities found in more complex objects.

# Objects and Workloads

- Deployments
  - One of the most common workloads to directly create and manage.
  - Use replication sets as a building block.



# Deployments

- One of the most common workloads to directly create and manage.
- Use replication sets as a building block.

```
kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml
```

```
kubectl get deployments
```

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

```
kubectl rollout status deployment.v1.apps/nginx-deployment
```

```
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
deployment.apps/nginx-deployment successfully rolled out
```

```
kubectl get deployments
```

```
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            3           18s
```

controllers/nginx-deployment.yaml

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

```
kubectl --record deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1
```

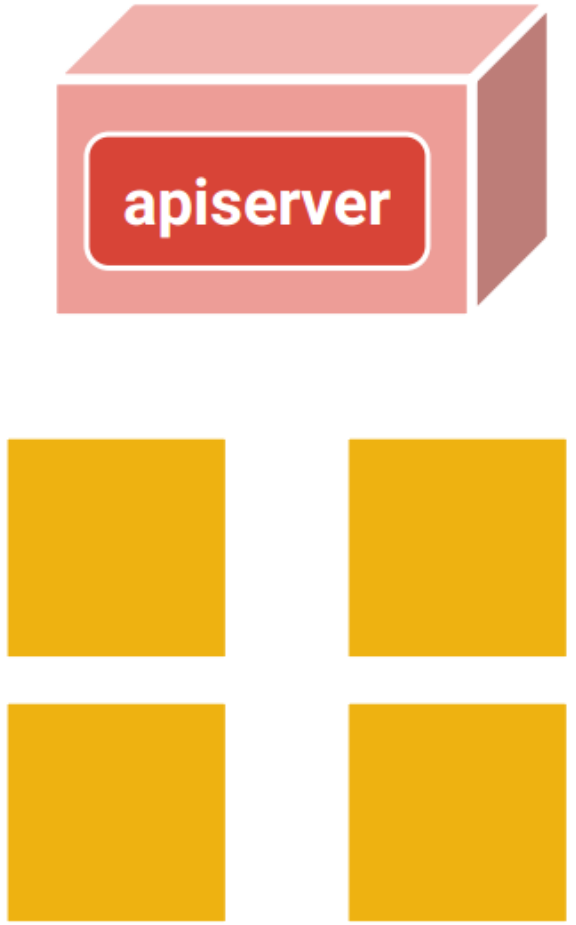
```
image updated
```

# Deployment

Problem: Manual rollout of applications (and changes)

Today:

- kubectl rolling-update
- sed (?!?)



# Deployment API

## Solution: Deployment API

$ kubectl create -f Deployment-1.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.7
        ports:
        - containerPort: 80
```

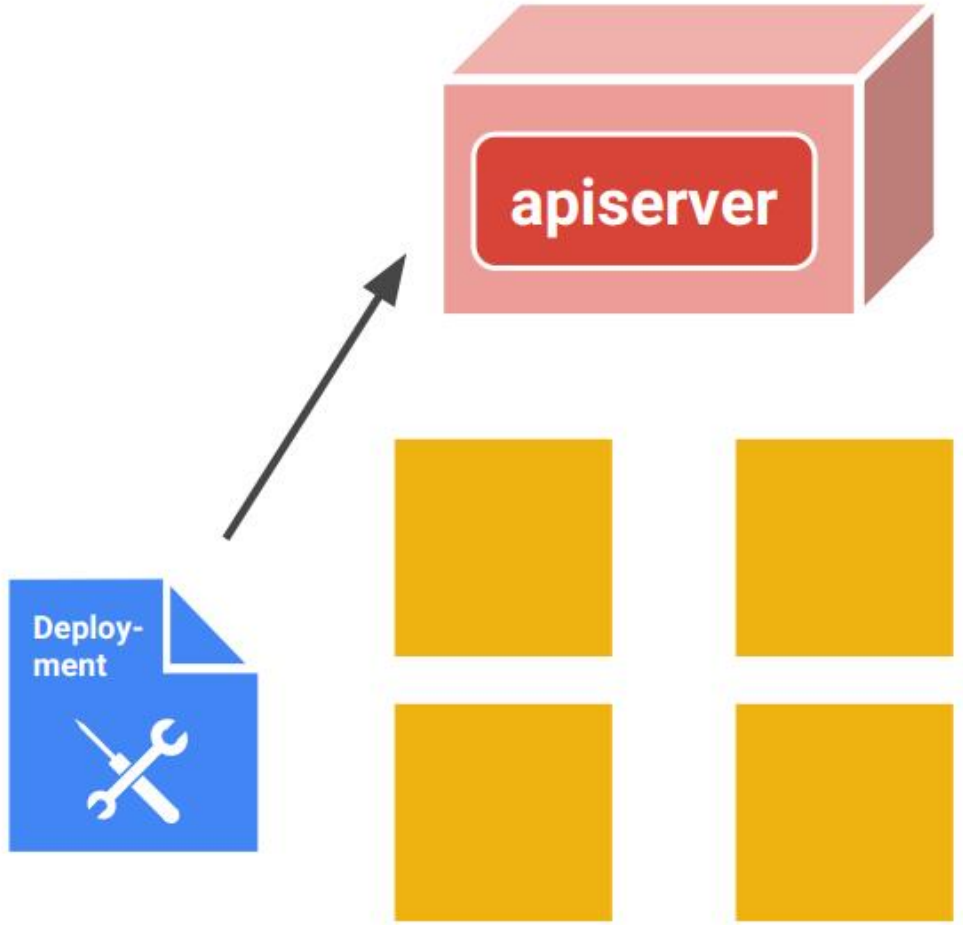


# Deployment API

## Solution: Deployment API

$ kubectl create -f Deployment-1.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.7
        ports:
        - containerPort: 80
```

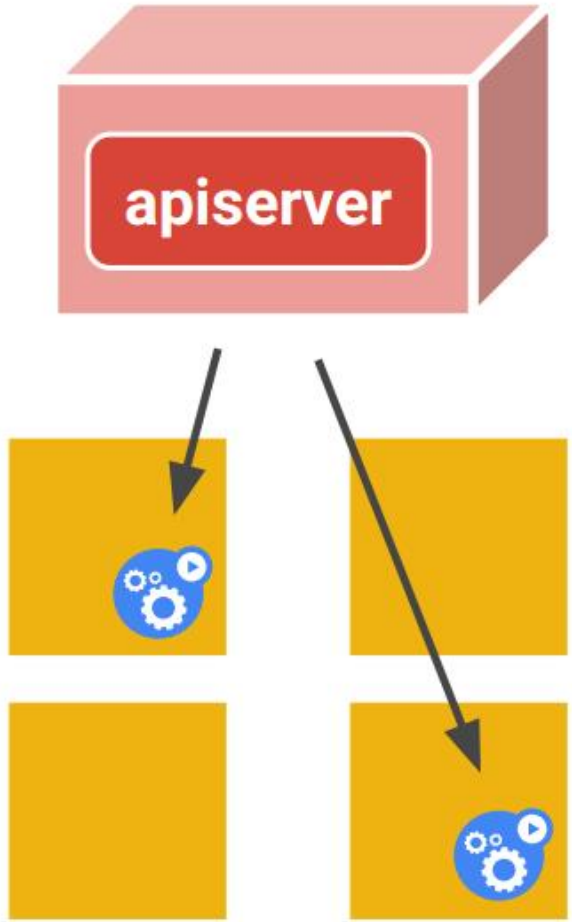


# Deployment API

## Solution: Deployment API

$ kubectl create -f Deployment-1.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.7
        ports:
        - containerPort: 80
```



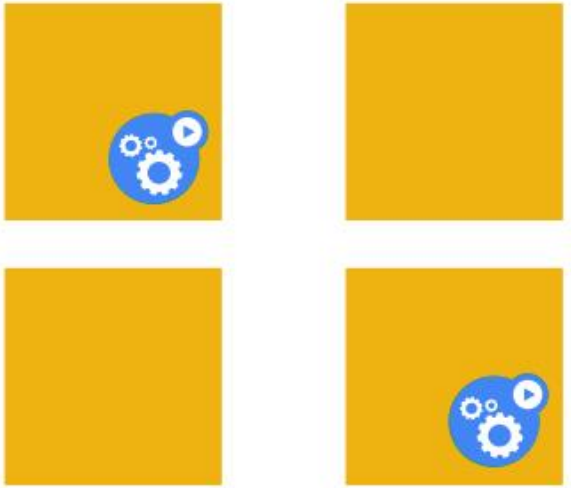
# Deployment API

## Solution: Deployment API!

$ kubectl create -f Deployment-2.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.8
        ports:
        - containerPort: 80
```



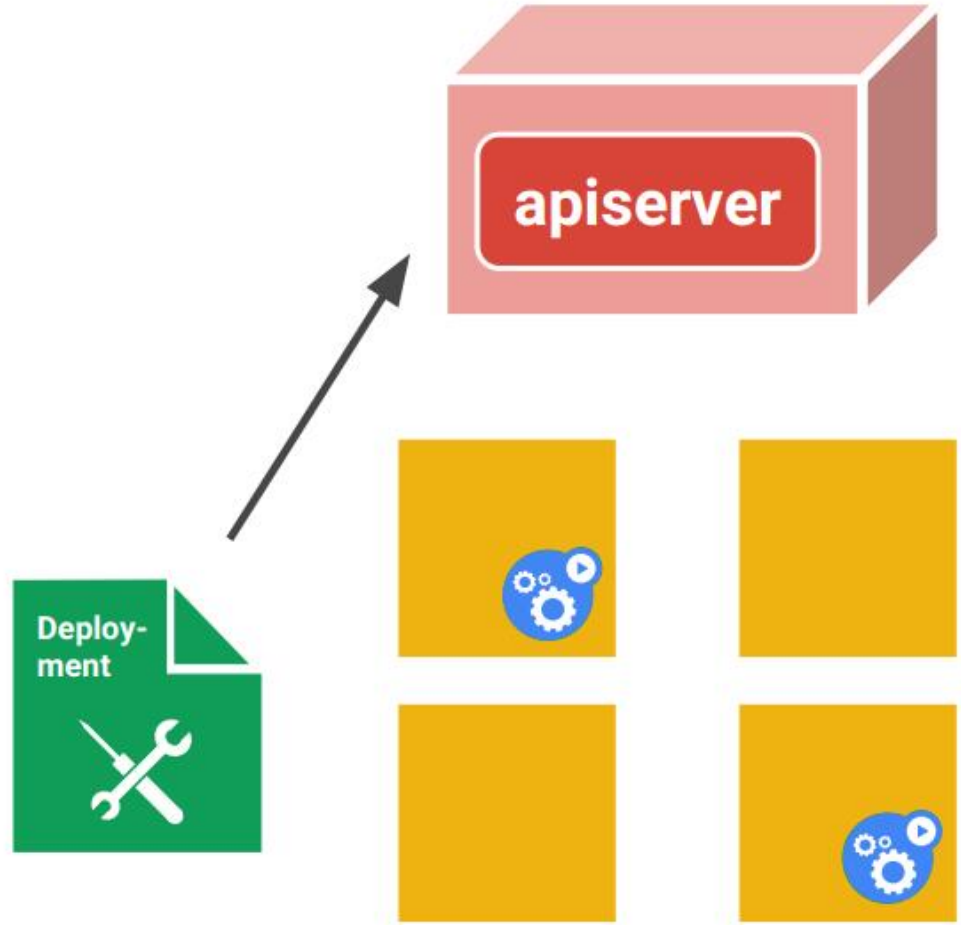


# Deployment API

## Solution: Deployment API!

$ kubectl create -f Deployment-2.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.8
        ports:
        - containerPort: 80
```

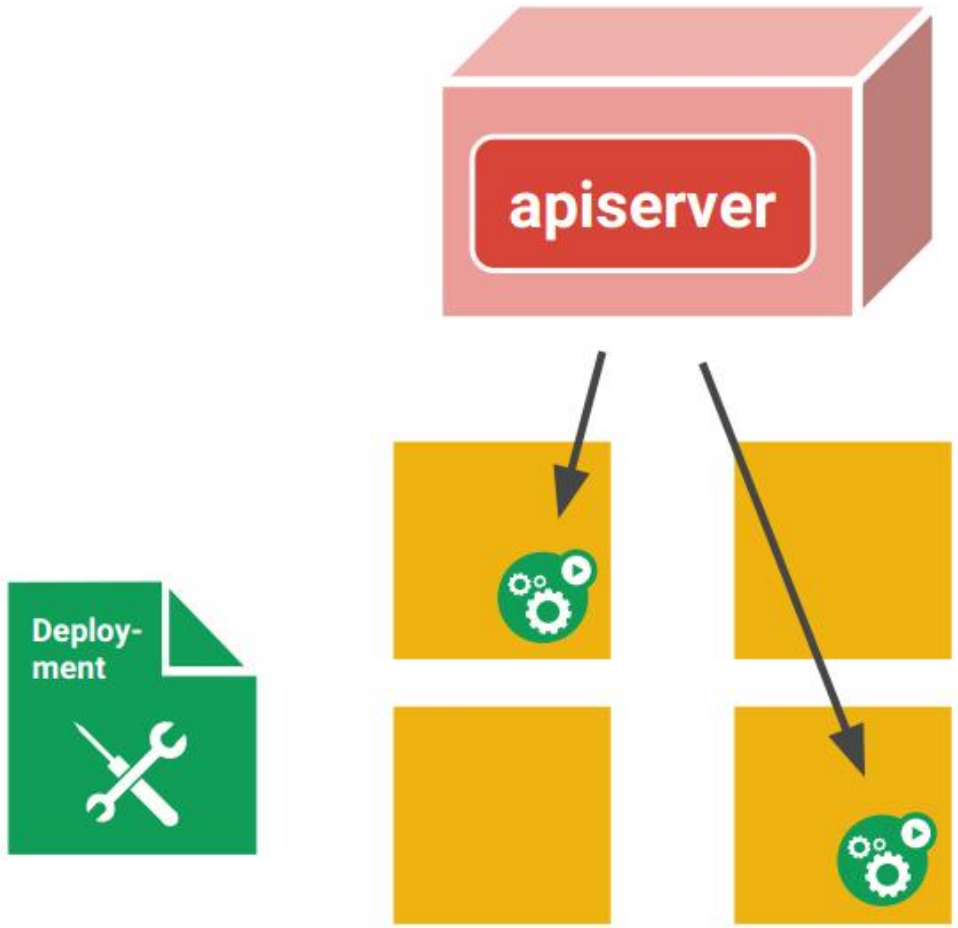


# Deployment API

## Solution: Deployment API!

$ kubectl create -f Deployment-2.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.8
        ports:
        - containerPort: 80
```

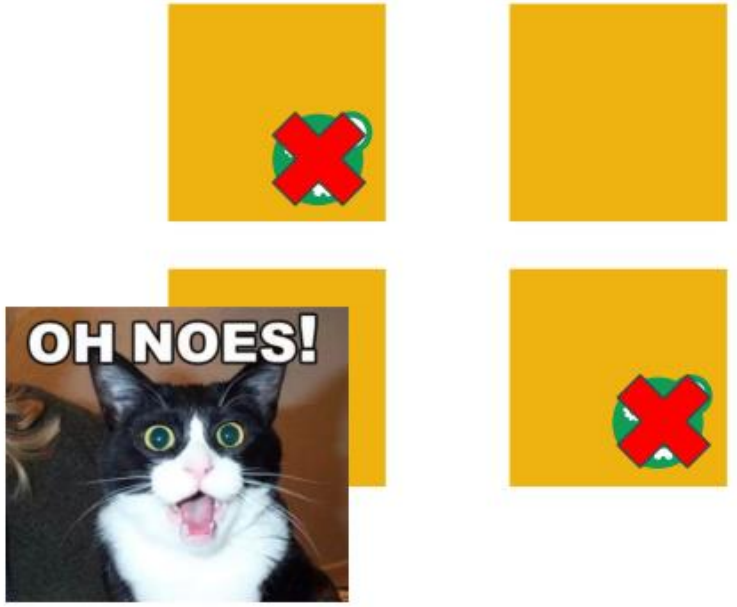


# Deployment API

## Solution: Deployment API!

$ kubectl create -f Deployment-2.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.8
        ports:
        - containerPort: 80
```

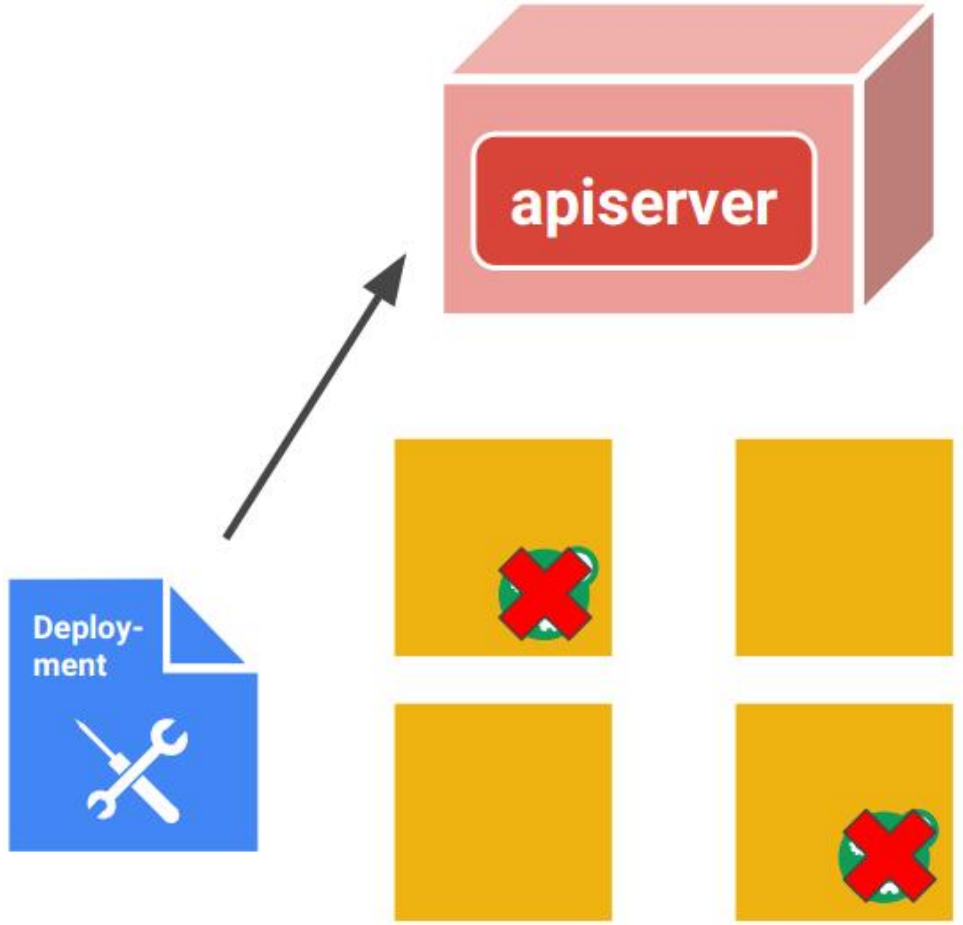



# Deployment API

## Solution: Deployment API!

$ kubectl create -f Deployment-1.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.7
        ports:
        - containerPort: 80
```

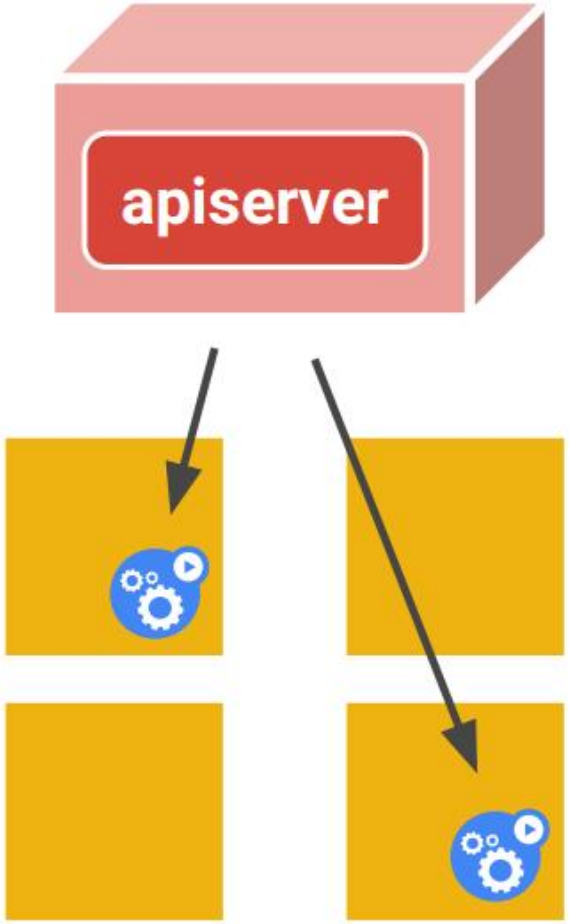


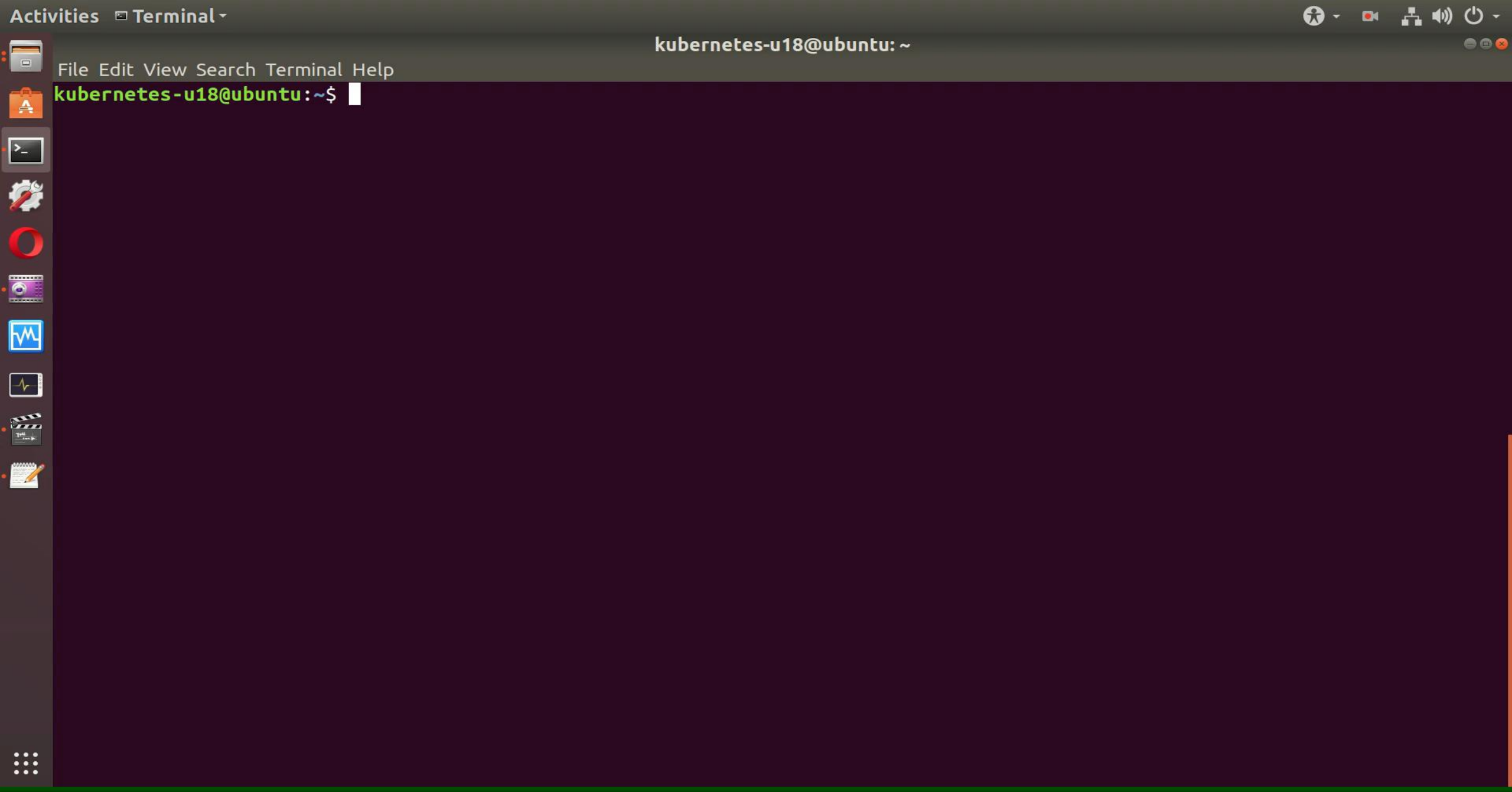
# Deployment API

## Solution: Deployment API!

$ kubectl create -f Deployment-1.yml

```
metadata:
  name: sweet-webapp
spec:
  replicas: 42
  template:
    spec:
      containers:
      - name: webapp
        image: webapp:2.1.7
        ports:
        - containerPort: 80
```

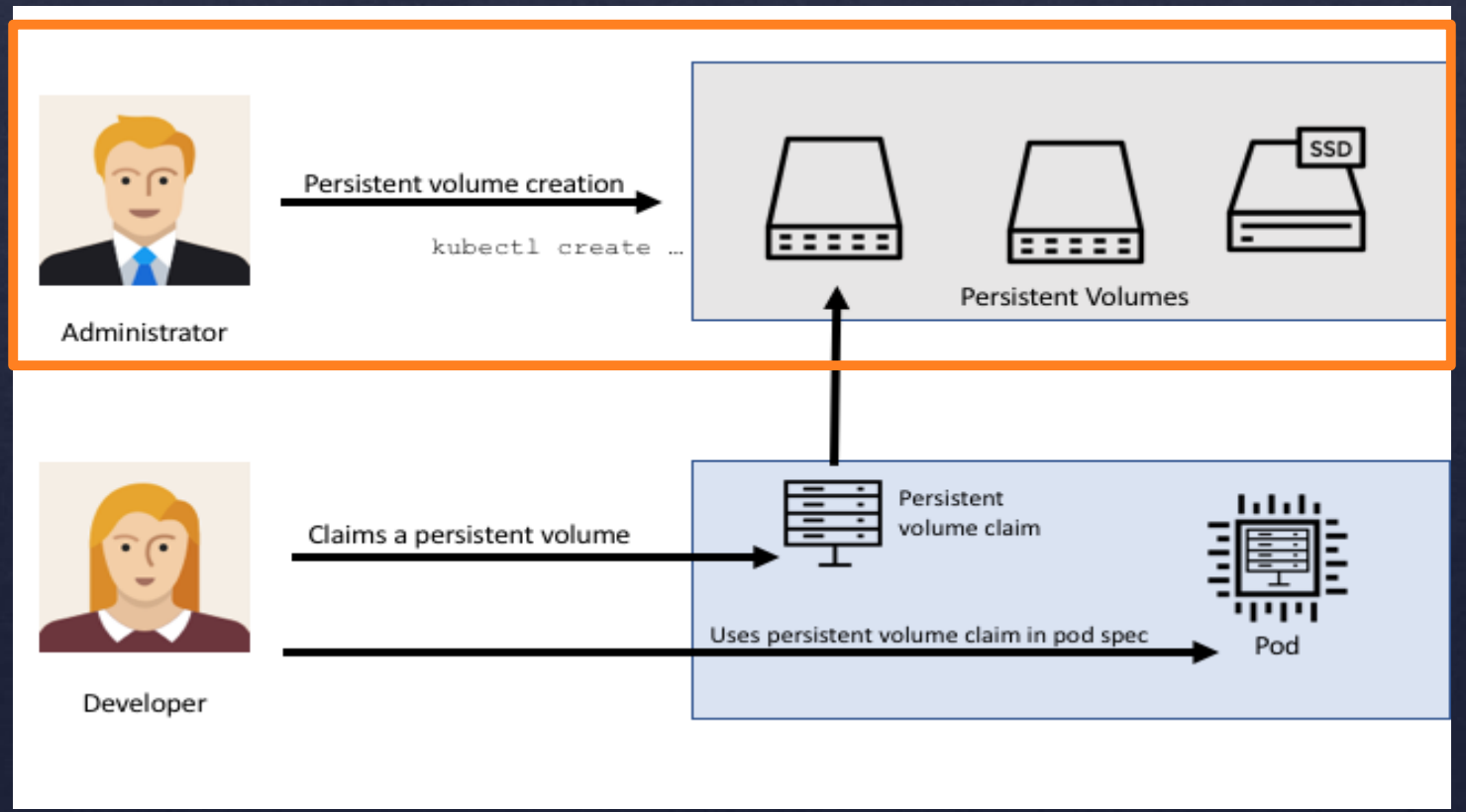



◈ Deployment/HelloWorld

# Objects and Workloads

- Persistent Volume(PV)
  - Piece of storage in the cluster that has been provisioned by an administrator
  - This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

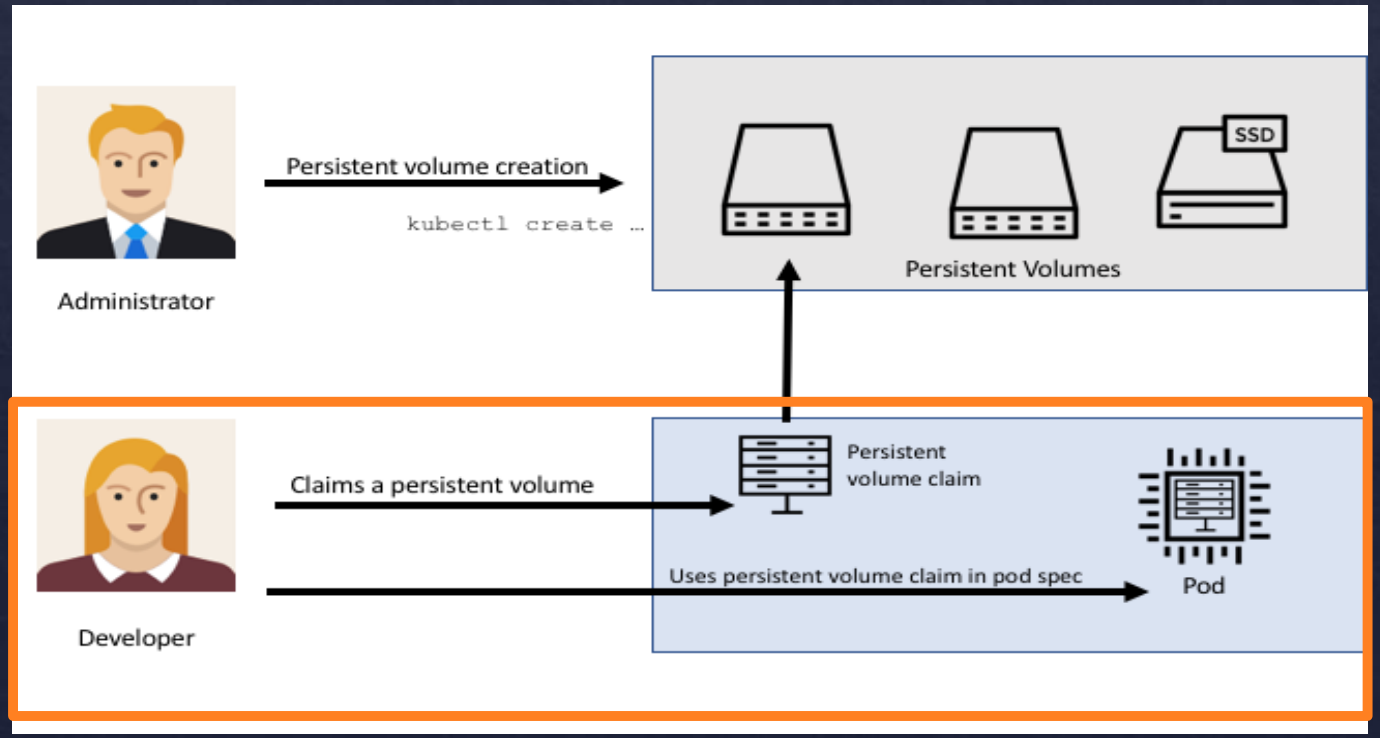
```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```



# Objects and Workloads

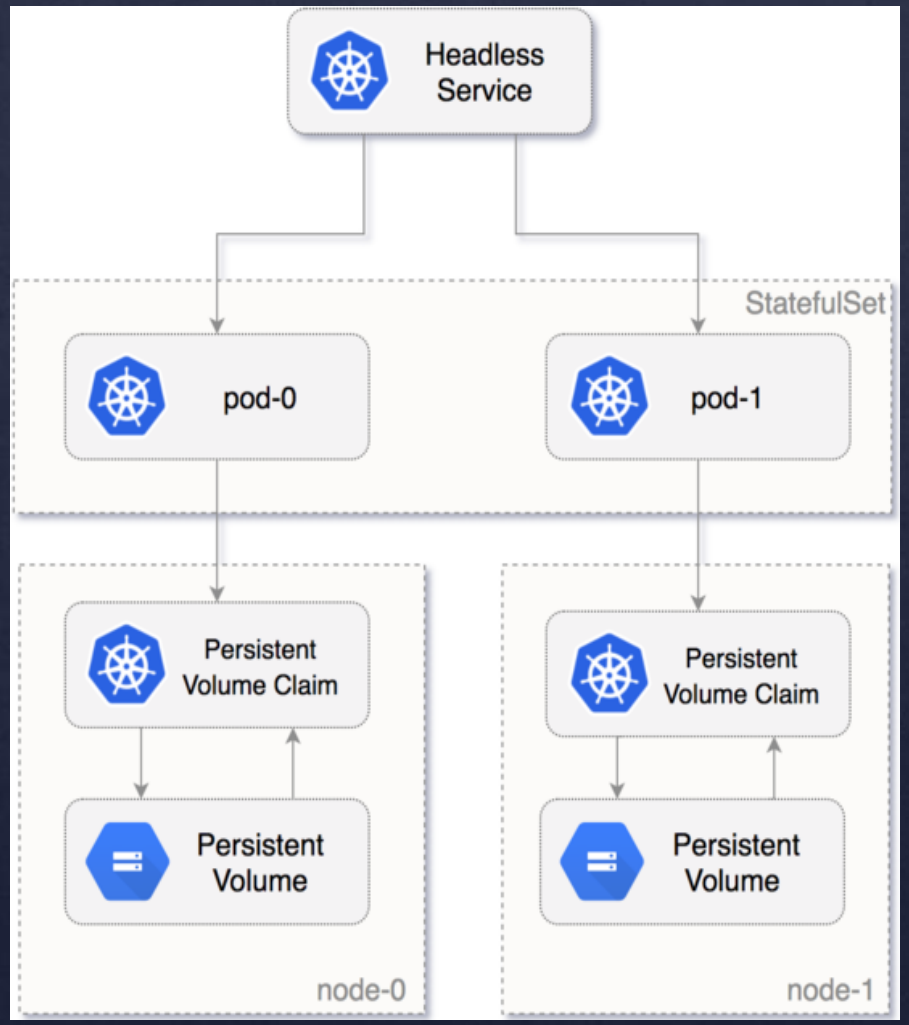
- Persistent Volume Claim(PVC)
  - It is similar to a pod Except
    1. Pods consume node resources While PVCs consume PV resources
    2. Pods can request specific levels of resources (CPU and Memory)(Namespace).
    3. Claims can request specific size and access modes (can be mounted once read/write or many times read-only).

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: environment, operator: In, values: [dev]}
```



# Objects and Workloads

- StatefulSets
  - Are specialized pod controllers
  - Often associated with data-oriented applications
    - ➢ Like databases,
      - ✓ which need access to the same volumes even if rescheduled to a new node.
  - Gives fine-grained control when you have special requirements related to :
    - ➢ Stable, unique network identifiers.
    - ➢ Stable, persistent storage.
    - ➢ Ordered, graceful deployment and scaling.
    - ➢ Ordered, automated rolling updates.



# StatefulSets

- Update Strategies
  - `.spec.updateStrategy.type` is set to `RollingUpdate`
  - `.spec.updateStrategy.type` is set to `OnDelete`

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

# Objects and Workloads

- Daemon Sets
  - Another specialized form of pod controller
  - Run a copy of a pod on each node in the cluster (or a subset, if specified)
  - Useful when deploying pods that help {perform maintenance/provide services} for the nodes
  - Example:
    - ➢ 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
      - ✓ SignalFx Agent

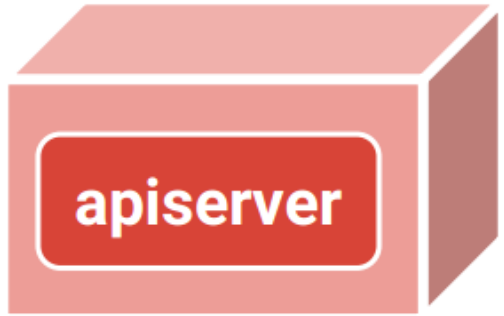
# Daemon Sets

```
kubectl apply -f https://k8s.io/examples/controllers/daemonset.yaml
```

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: gcr.io/fluentd-elasticsearch/fluentd:v2.5.1
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

# DaemonSets

Problem: Run one (and only one) pod on every node (or a subset of nodes)




Today:

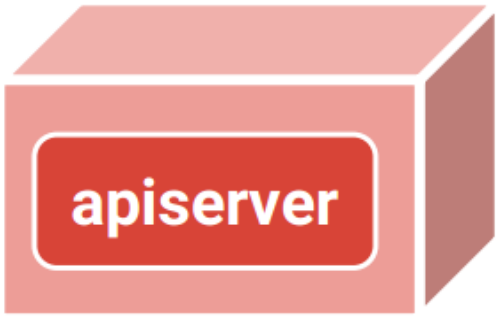
- Over-replicate
- Static pods
- Grab unique resource per node
- Run outside of Kubernetes

# DaemonSets

## Solution: Daemon Sets!

$ kubectl create -f daemon-sharder.yml

```
apiVersion: extensions/v1
kind: DaemonSet
spec:
  template:
    spec:
      nodeSelector:
        app: datastore-node
      containers:
        image: kubernetes/sharded
        ports:
          - containerPort: 9042
```



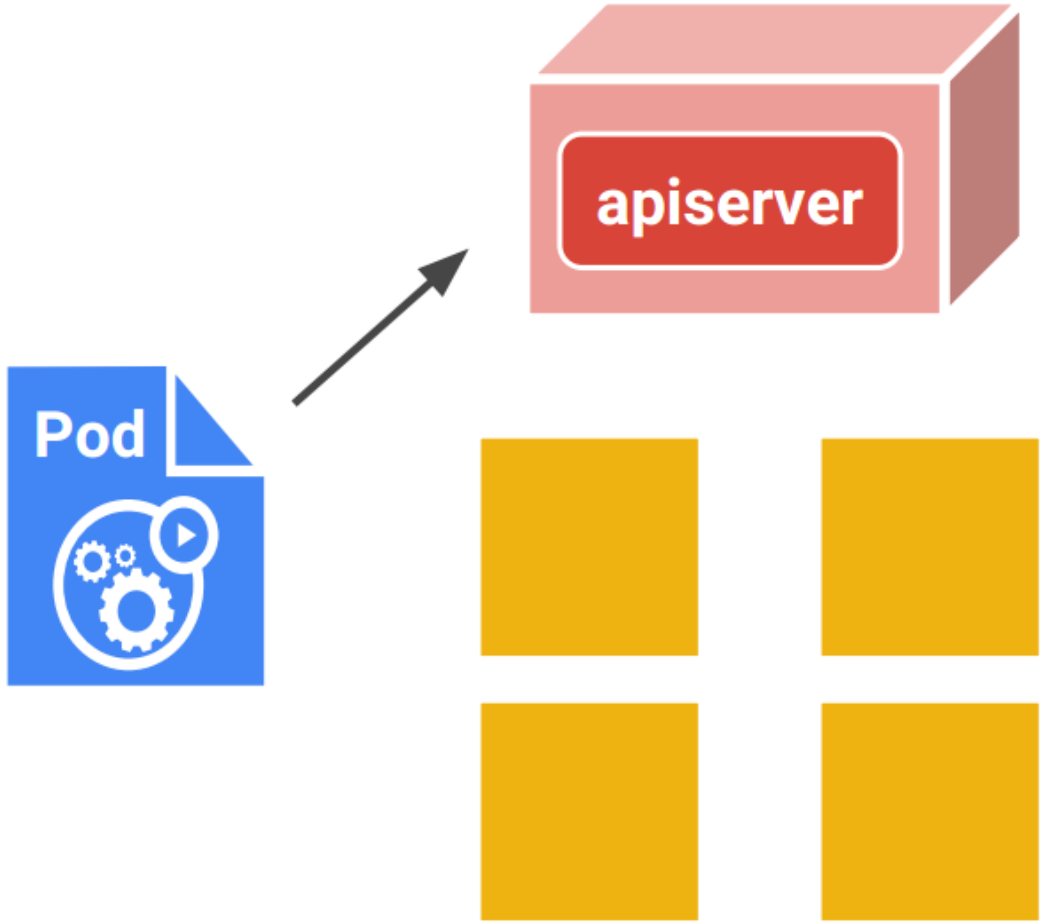


# DaemonSets

## Solution: Daemon Sets!

$ kubectl create -f daemon-sharder.yml

```
apiVersion: extensions/v1
kind: DaemonSet
spec:
  template:
    spec:
      nodeSelector:
        app: datastore-node
      containers:
        image: kubernetes/sharded
        ports:
          - containerPort: 9042
```

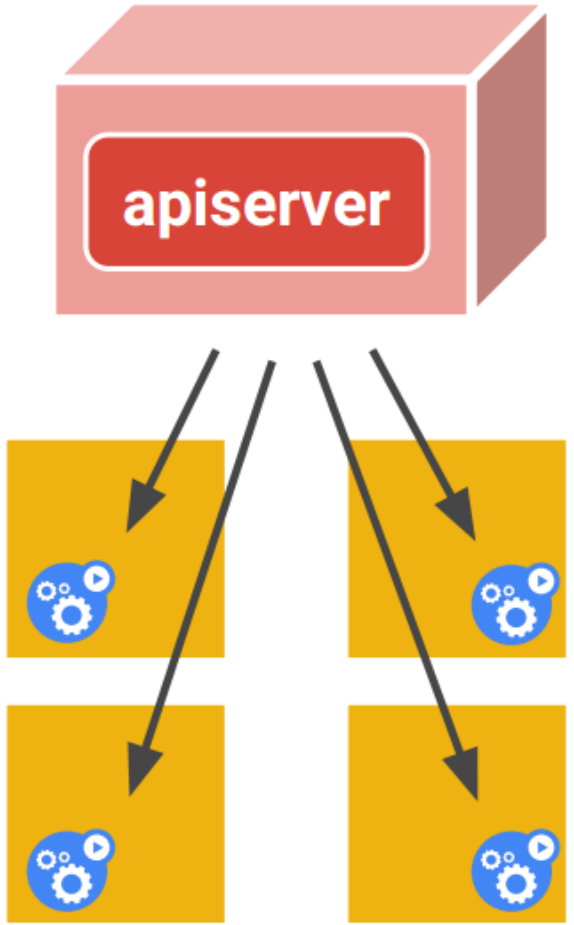


# DaemonSets

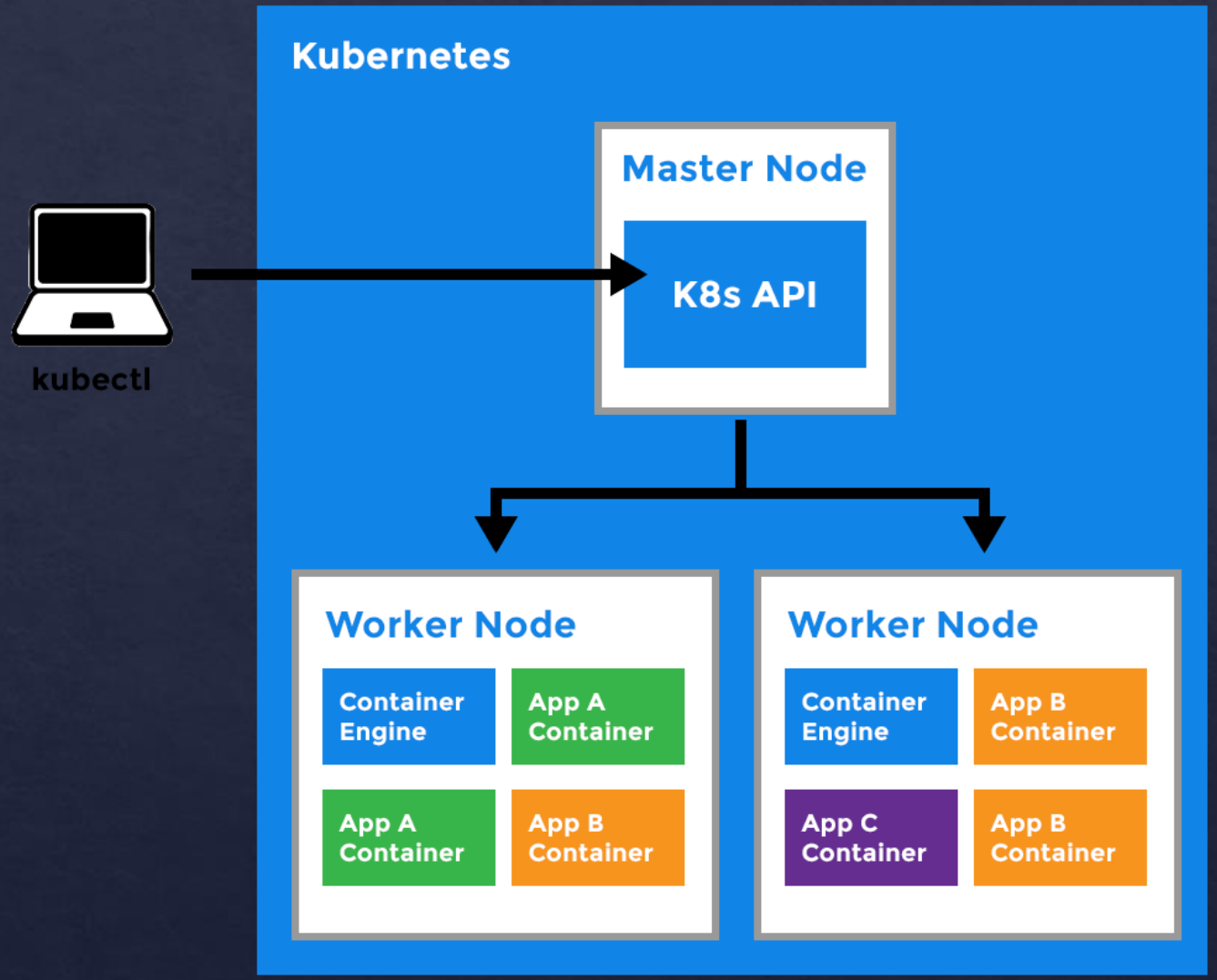
## Solution: Daemon Sets!

$ kubectl create -f daemon-sharder.yml

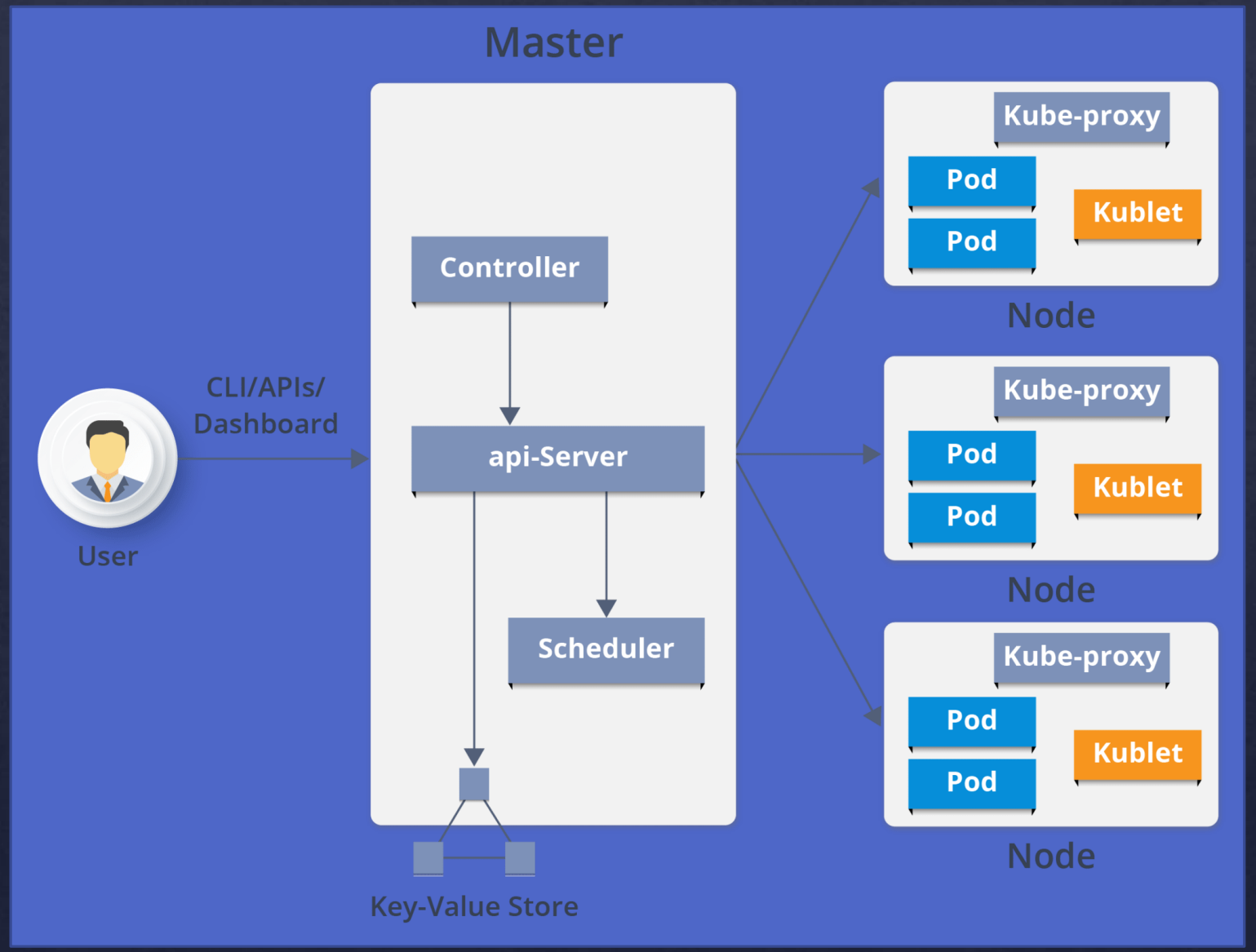
```
apiVersion: extensions/v1
kind: DaemonSet
spec:
  template:
    spec:
      nodeSelector:
        app: datastore-node
      containers:
        image: kubernetes/sharded
        ports:
          - containerPort: 9042
```

a CLI tool for Kubernetes



To be continued…



# What is Kubernetes?

At its basic level: is a

…1… for …2… , …3… across …4…

# What is Kubernetes?

At its basic level: is a

…1… for …2… , …3… across …4…

1. System

# What is Kubernetes?

At its basic level: is a

System for …2… , …3… across …4…

1. System
2. Running and Coordinating

# What is Kubernetes?

At its basic level: is a

System for Running and Coordinating , …3… across …4…

1. System
2. Running and Coordinating
3. Containerized Applications

# What is Kubernetes?

At its basic level: is a

System for Running and Coordinating , Containerized

Applications across …4…

1. System
2. Running and Coordinating
3. Containerized Applications
4. Clusters of Machines

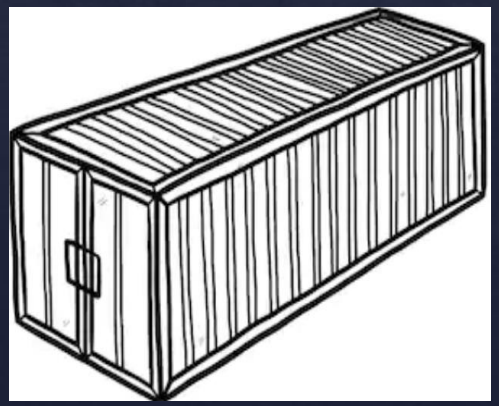
# What is Kubernetes?

## At its basic level: is a …

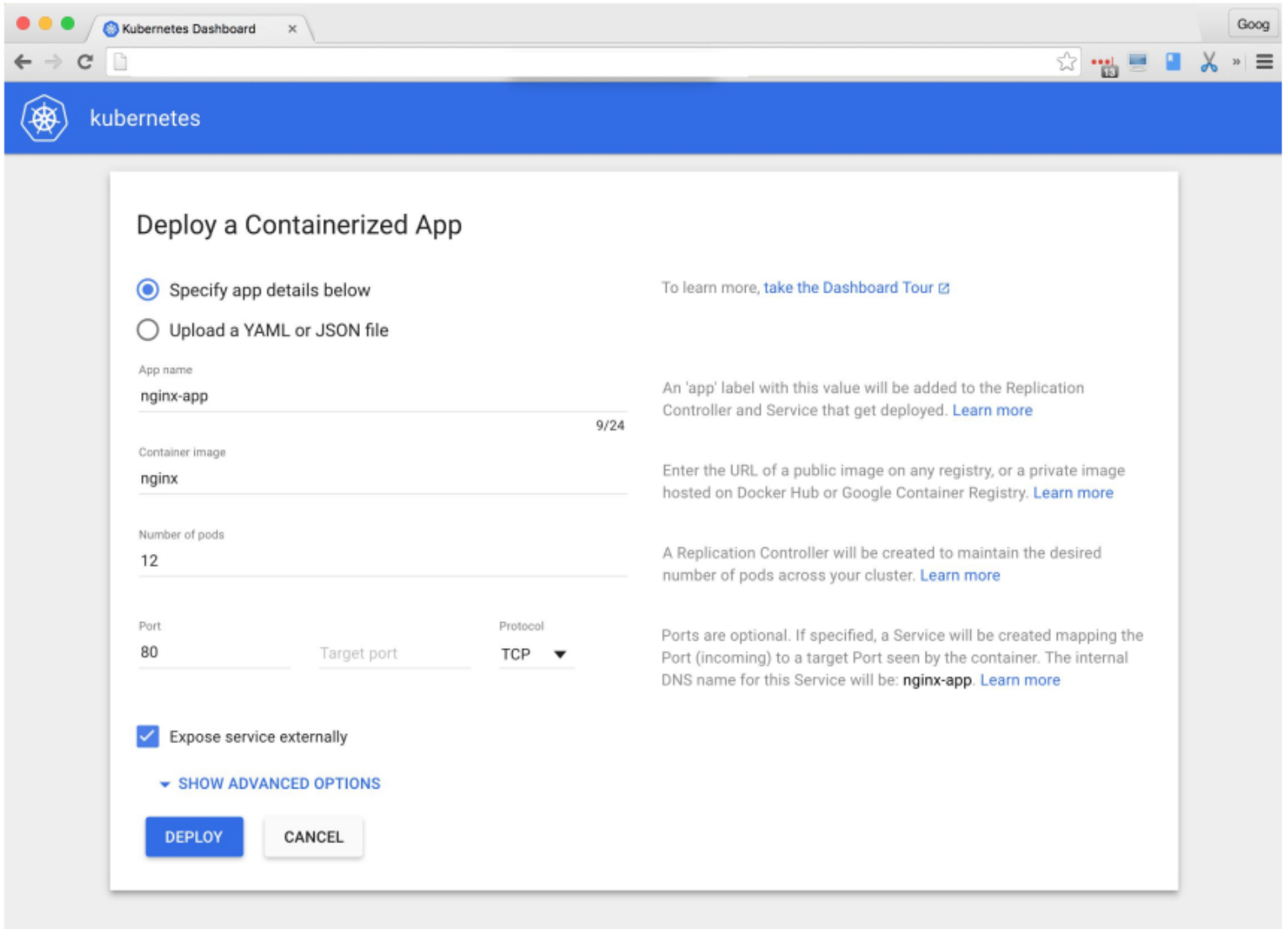
System for Running and Coordinating , Containerized

Applications across Clusters of Machines

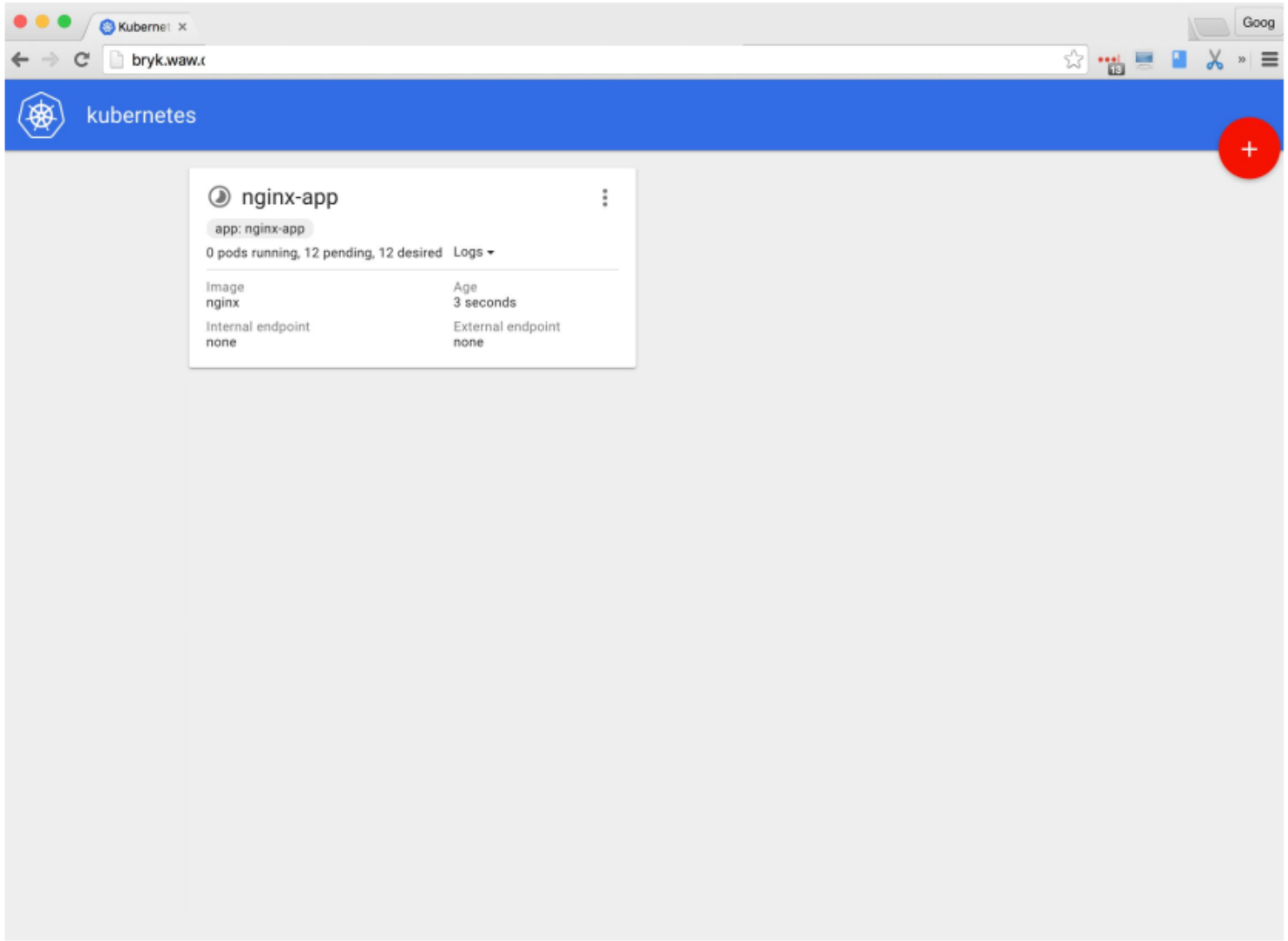




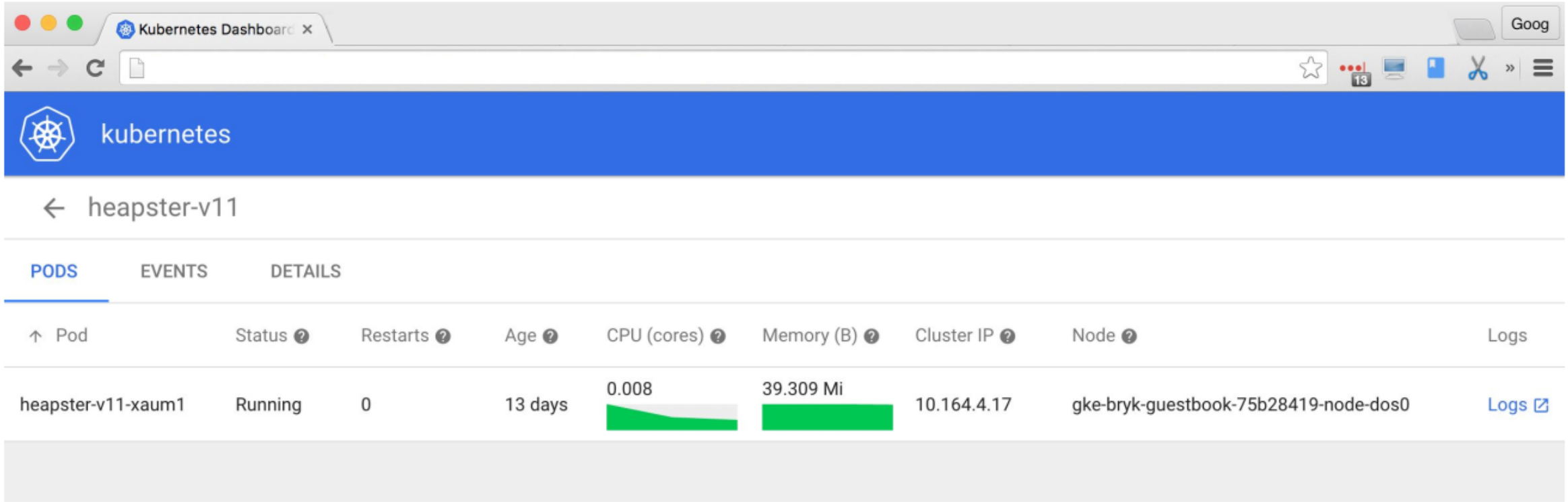
# UI

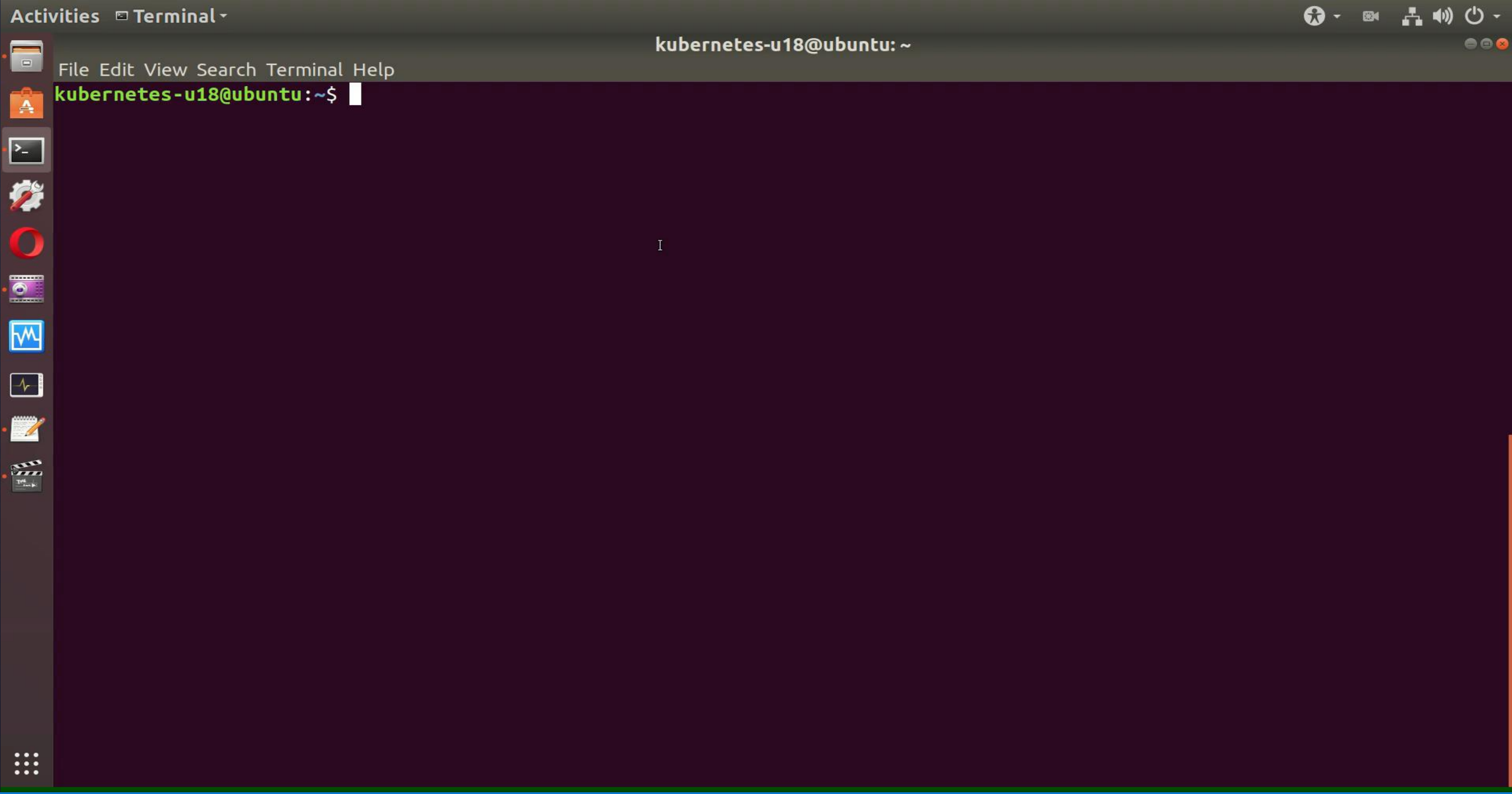

Kubernetes Dashboard
Goog
kubernetes
Deploy a Containerized App
Specify app details below
Upload a YAML or JSON file
To learn more, take the Dashboard Tour
App name
nginx-app
9/24
An 'app' label with this value will be added to the Replication Controller and Service that get deployed. Learn more
Container image
nginx
Enter the URL of a public image on any registry, or a private image hosted on Docker Hub or Google Container Registry. Learn more
Number of pods
12
A Replication Controller will be created to maintain the desired number of pods across your cluster. Learn more
Port
80
Target port
Protocol
TCP
Ports are optional. If specified, a Service will be created mapping the Port (incoming) to a target Port seen by the container. The internal DNS name for this Service will be: nginx-app. Learn more
Expose service externally
SHOW ADVANCED OPTIONS
DEPLOY
CANCEL

# UI


Kuberne
bryk.waw.
Goog
kubernetes
nginx-app
app: nginx-app
0 pods running, 12 pending, 12 desired
Logs
Image
nginx
Age
3 seconds
Internal endpoint
none
External endpoint
none

# UI


Kubernetes Dashboard
Goog
kubernetes
heapster-v11
PODS
EVENTS
DETAILS
Pod
Status
Restarts
Age
CPU (cores)
Memory (B)
Cluster IP
Node
Logs
heapster-v11-xaum1
Running
0
13 days
0.008
39.309 Mi
10.164.4.17
gke-bryk-guestbook-75b28419-node-dos0
Logs




◈ Minikube/Dashboard

# Velocity

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**

# Resources

- https://slideplayer.com/slide/7483962/
- https://www.slideshare.net/chhattanshah/cluster-and-grid-computing
- https://martinfowler.com/articles/microservices.html
- http://energystorage.org/membership/members
- https://www.slideshare.net/Docker/introduction-to-docker-2017
- https://www.cloudbees.com/blog/clustering-jenkins-kubernetes-google-container-engine

# Resources

- https://en.wikipedia.org/wiki/Representational_state_transfer#History
- http://www.studytrails.com/devops/docker-architecture-engine-containerd-runc/
- https://dev.to/jibinliu/how-to-persist-data-in-docker-container-2m72
- https://docs.docker.com/storage/volumes/
- https://www.slideshare.net/GiacomoVacca/docker-from-scratch
- https://www.ianlewis.org/en/how-kubeadm-initializes-your-kubernetes-master
- https://thenewstack.io/implementing-advanced-scheduling-techniques-with-kubernetes/
- https://medium.com/jorgeacetozi/kubernetes-node-components-service-proxy-kubelet-and-cadvisor-dcc6928ef58c

# Resources

- https://blog.risingstack.com/what-is-kubernetes-how-to-get-started/
- https://linuxacademy.com/blog/containers/history-of-container-technology/
- https://blog.risingstack.com/what-is-kubernetes-how-to-get-started/
- https://coreos.com/kubernetes/docs/latest/replication-controller.html
- https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
- https://www.slideshare.net/kubecon/kubecon-eu-2016-keynote-kubernetes-state-of-the-union
- DevOps With Kubernetes Book