RX-M Cloud Native Consulting

Building Cloud Native Applications on Cloud Foundry

An in depth look at the microservices architecture pattern, containers and Cloud Foundry

10: CFCR/PKS and Kubernetes

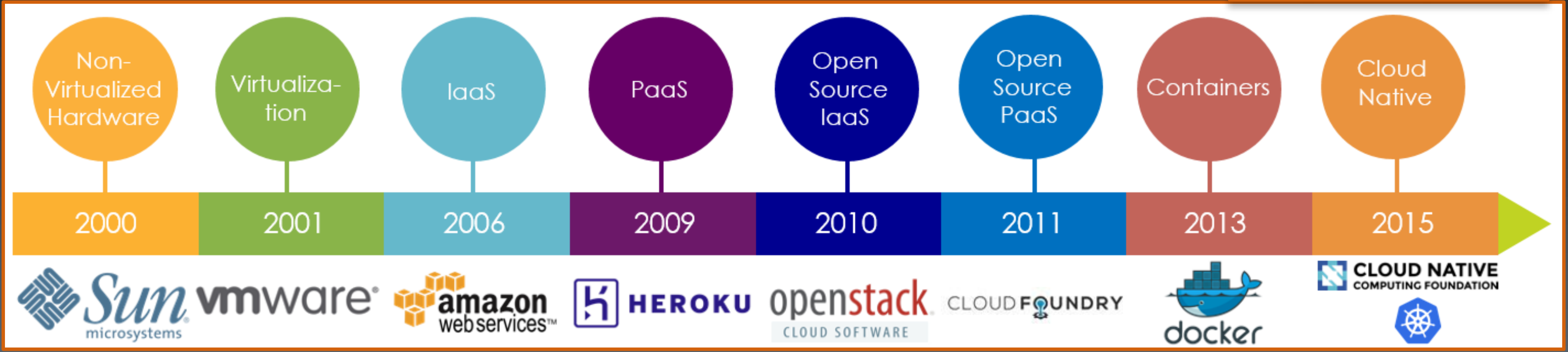
Objectives

- Describe the goals of the Kubernetes system
- Examine the design principles of Kubernetes
- Explore the Kubernetes architecture
- List the Kubernetes service components

Platform Evolution

- The platform underlying rapidly growing Internet applications has changed significantly over the years
  - .com era – bare metal servers
  - Web 2.0 – IaaS
  - APIs – PaaS
  - Microservices – Cloud Native

Image courtesy CNCF



Important PaaS

- Public Cloud based
  - AppEngine
    - Because it is Google, first major PaaS offering
  - Heroku/SalesForce
    - The most popular web tech PaaS (Ruby on Rails, Node.js Express, Python Django)
  - Elastic Beanstalk
    - Because it is Amazon and linked to the services of the largest cloud
  - Azure
    - Because it is Microsoft and the cornerstone of .net in the cloud

- Open Source PaaS
  - Cloud Foundry
    - The Java Spring community centric PaaS
  - OpenShift
    - The Java EE community centric PaaS

The Cloud Foundry Foundation

PLATINUM

CISCO, DELL EMC, IBM, Pivotal, SAP, SUSE, vmware

GOLD

accenture High performance. Delivered., Allianz, Allstate You're in good hands., Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING, Cognizant, GE Digital, Google, HUAWEI, Microsoft, NTT, sas, swisscom, VOLKSWAGEN AKTIENGESELLSCHAFT

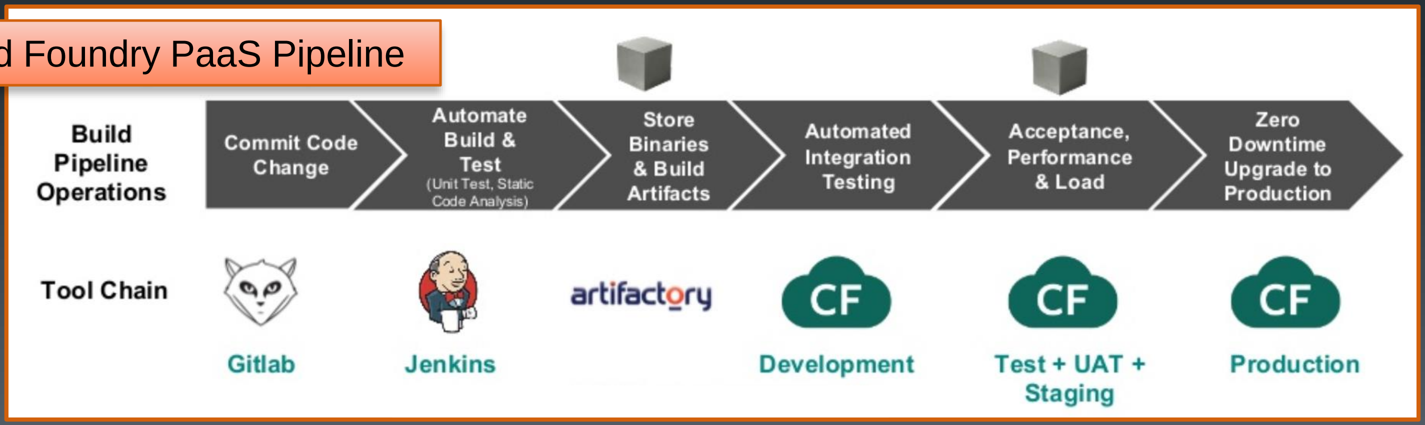
SILVER

acetti, ALTOROS, anynines, armakuni, Atos, BIARCA, Bloomberg, BNY MELLON, BOSCH, ca technologies, cloudsoft, CODENVY, COMCAST, docker, ECS, ENGINEER BETTER, evoila, Fidelity INTERNATIONAL, FUJITSU, gemalto security to be free, grape up, hazelcast, Hewlett Packard Enterprise, Hexad, HITACHI Inspire the Next, Honeywell, intel, JPMorgan Chase & Co., mendix, MIMACOM, MINIO, ANCHORA 魔泊云 MoPaaS, NIA, orange, PROXIMETRY, QiO, resilientscale, RBC, redislabs, snyk, STARK & WAYNE, TOSHIBA

PaaS - CFAR

- Platform as a Service
  - Push code
  - PaaS supports only applications fitting one of it’s buildpack models
  - The PaaS builds, tests, packages and deploys
  - Platform containerizes the artifacts
    - Perhaps JARs, a custom format, Docker Images or other
- Cloud Foundry Application Runtime [CFAR] is the Cloud Foundry PaaS

Cloud Foundry PaaS Pipeline



Build Pipeline Operations

Commit Code Change

Automate Build & Test (Unit Test, Static Code Analysis)

Store Binaries & Build Artifacts

Automated Integration Testing

Acceptance, Performance & Load

Zero Downtime Upgrade to Production

Tool Chain

Gitlab

Jenkins

artifactory

CF

Development

CF

Test + UAT + Staging

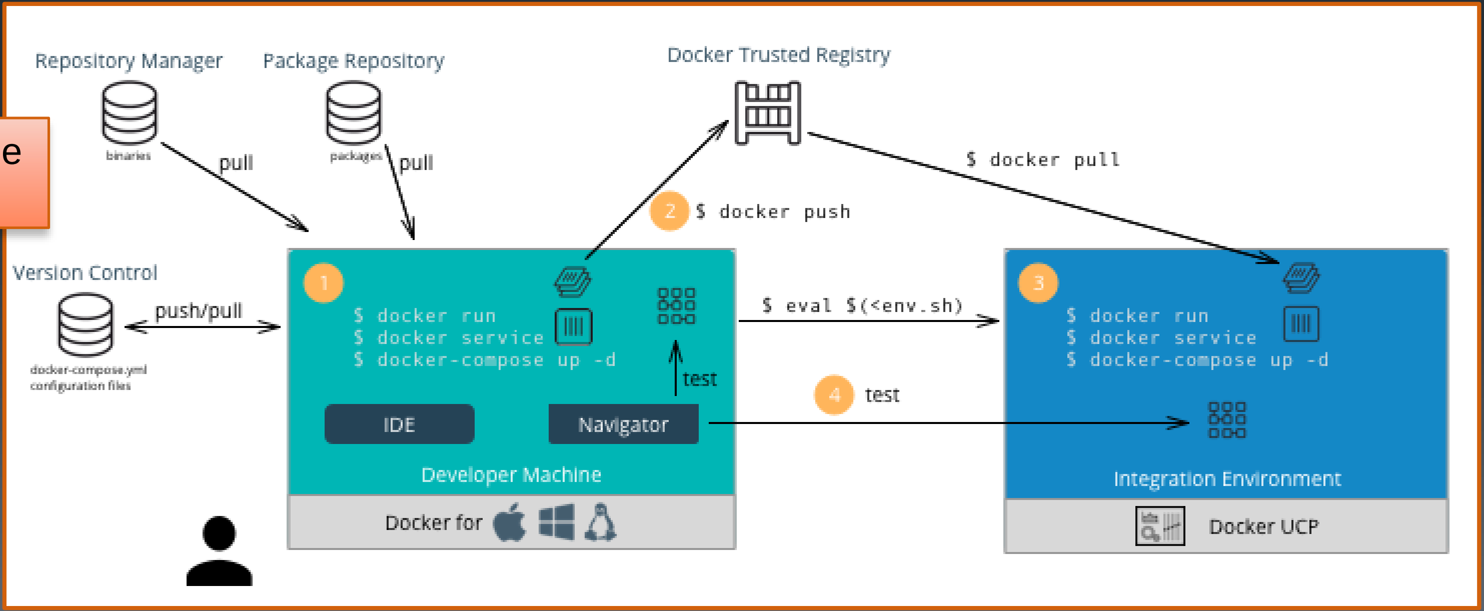
CF

Production

CaaS - CFCR

- Containers as a Service
  - Push container Images
  - The CaaS deploys and orchestrates
  - Containers are created earlier in the pipeline reducing variability
- Hybrid
  - Most popular PaaS today use container technology
    - Cloud Foundry uses containers and its own OCI runc based Guardian container manager (Greenhouse on Windows)
    - RedHat OpenShift uses Docker as the container manager and Kubernetes as the orchestration engine
- Cloud Foundry Container Runtime [CFCR] is Cloud Foundry's Kubernetes solution

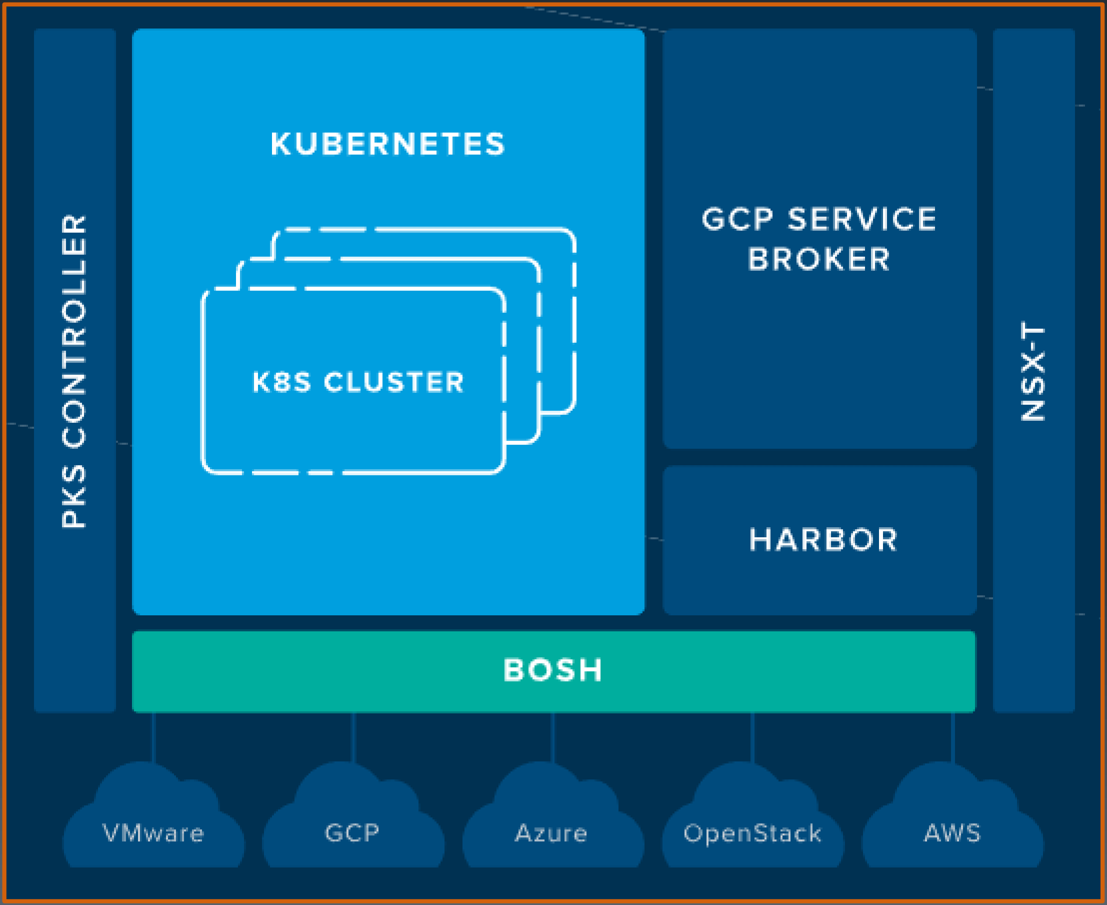
Docker Dev Pipeline Reference Arch



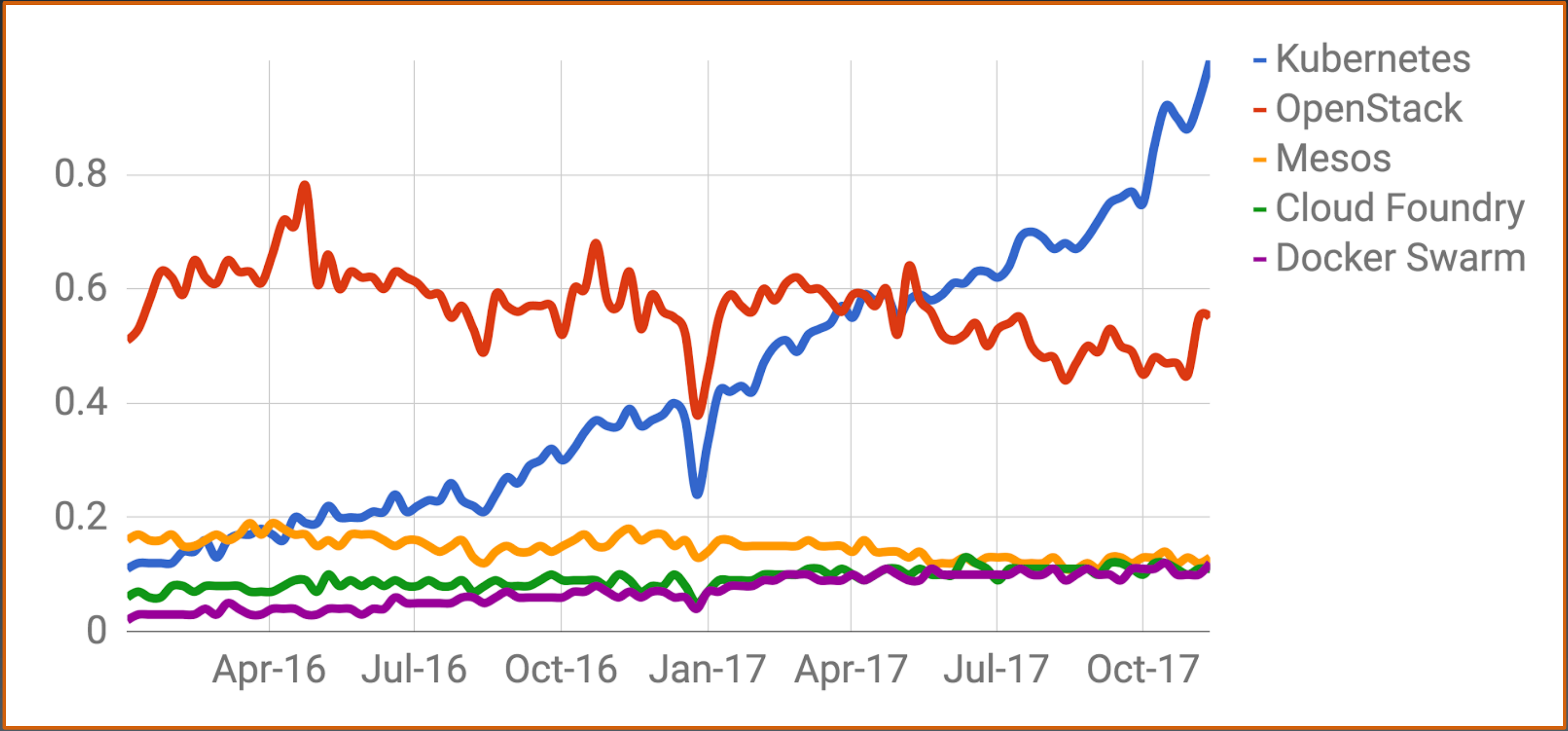
```
$ docker push
$ docker pull
$ docker run
$ docker service
$ docker-compose up -d
$ eval $(<env.sh)
$ docker run
$ docker service
$ docker-compose up -d
```

Pivotal Container Service [PKS]

- Kubernetes
  - Current stable release of Kubernetes, deployed and managed by BOSH
  - No proprietary extensions
- PKS Controller
  - Creates and scales Kubernetes clusters from the command line and API
- GCP Service Broker
  - Allows apps to access Google Cloud APIs
  - Provides Container Engine (GKE) portability
- Harbor
  - A docker registry server with vulnerability scanning, identity management, and support for multiple registries
- NSX-T
  - VMware NSX-T secure SDN
- BOSH
  - BOSH deploys, scales, monitors and heals Kubernetes clusters



Google Trends

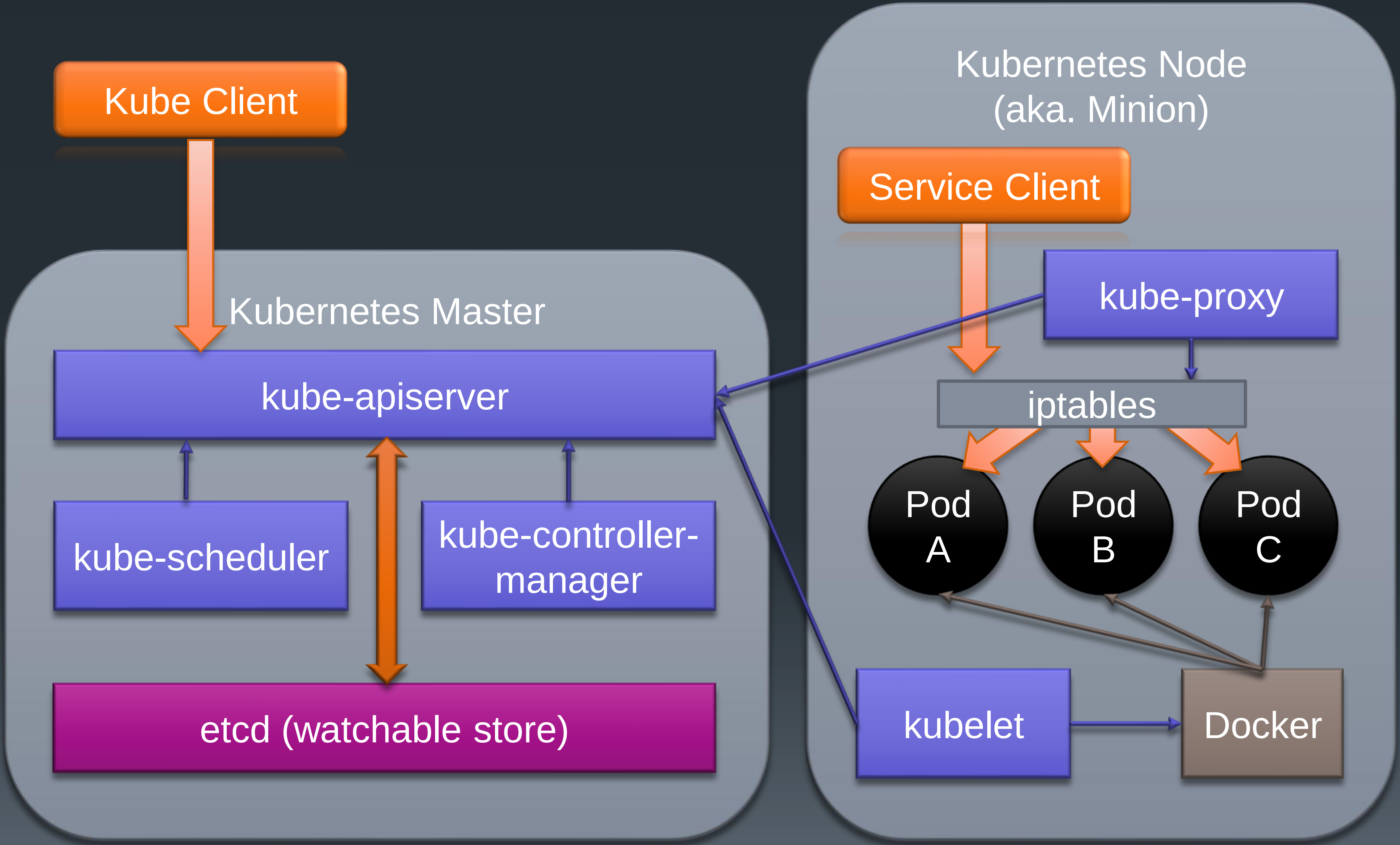


Kubernetes Design Overview

- Kubernetes is a system for managing containerized applications across multiple hosts
  - Deployment
  - Maintenance
  - Scaling
- Kubernetes establishes robust declarative primitives for maintaining the desired state requested by the user
  - Main value add of Kubernetes, declarative target state
- Kubernetes is primarily targeted at applications composed of multiple containers
  - Elastic distributed micro-services
- Also designed to facilitate migration of non-containerized application stacks to Kubernetes
  - Includes abstractions for grouping containers in both loosely coupled and tightly coupled formations
  - Provides ways for containers to find and communicate with each other in relatively familiar ways
- Kubernetes enables users to ask a cluster to run a set of containers
  - The system automatically chooses hosts to run those containers on
  - Scheduling is a policy-rich, topology-aware, workload-specific function that significantly impacts:
    - Availability
    - Performance
    - Capacity
  - The scheduler takes into account:
    - individual and collective resource requirements
    - quality of service requirements
    - hardware/software/policy constraints
    - affinity and anti-affinity specifications
    - data locality
    - inter-workload interference
    - Deadlines
  - Workload-specific requirements exposed through the API as necessary

- Self-healing mechanisms
  - Auto-restart
  - Re-schedule
  - Replication
- Kubernetes runs on a number of cloud providers, as well as on physical hosts
- A single Kubernetes cluster is not intended to span multiple availability zones
  - Recommend building a higher-level layer to replicate complete deployments of highly available applications across multiple zones
- Kubernetes aspires to be an extensible, pluggable, building-block OSS platform and toolkit
  - Kubernetes is built as a collection of pluggable components and layers
  - The ability to use alternative schedulers, controllers, storage systems, and distribution mechanisms
  - Others are building higher-level PaaS functionality and multi-cluster layers, without modification of core Kubernetes source
  - API isn't just (or even necessarily mainly) targeted at end users, but at tool and extension developers
  - All APIs are visible and available, including the APIs used by the scheduler, the node controller, the replication-controller manager, etc.

Kubernetes Architecture

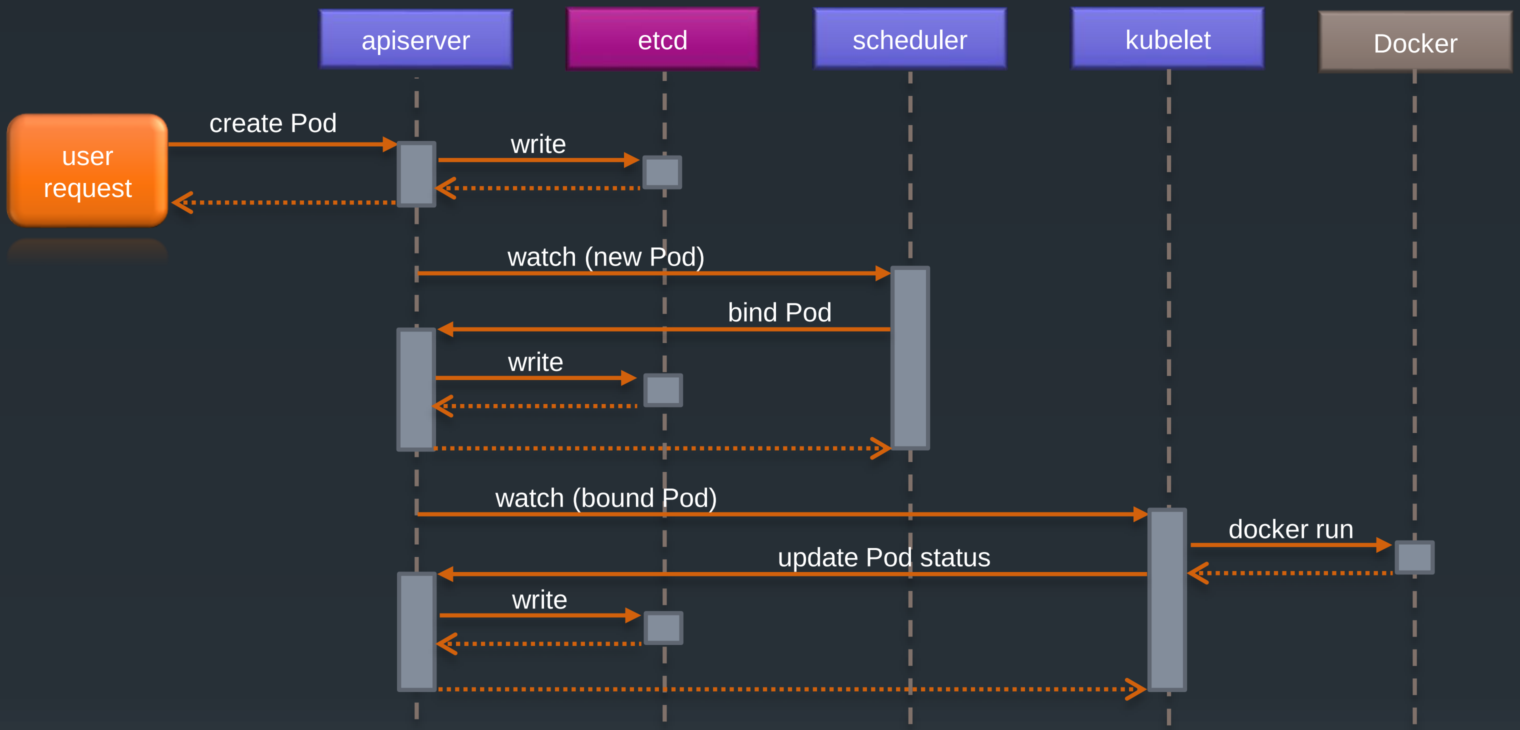


Master Components

- The Kubernetes Control Plane
  - The Kubernetes control plane is split into a set of components which currently all run on a master node
  - These components work together to provide a unified view of the cluster
- etcd
  - All persistent master state is stored in an instance of etcd
  - This provides a way to store configuration data reliably
  - Watch support allows coordinating components to be notified of changes
- API Server
  - The apiserver serves the Kubernetes API
  - A CRUD-y server, with most/all business logic implemented in separate components or in plug-ins
  - Mainly processes REST operations, validates them, and updates the corresponding objects in etcd
- Scheduler
  - Binds unscheduled pods to nodes via the /binding API
  - The scheduler is pluggable
- Controller Manager
  - All other cluster-level functions are currently performed by the Controller Manager
  - Endpoint objects are created and updated by the endpoints controller
  - Nodes are discovered, managed, and monitored by the node controller
  - Replication is managed by Deployments & Replica Sets
  - Could eventually be split into separate components to make them independently pluggable

- Every resource in Kubernetes, such as a pod, is identified by a URI and has a UID
- The URI includes
  - kind of object (e.g. pod)
  - object name
  - Namespace
- For a certain object kind, every name is unique within its namespace
- In contexts where an object name is provided without a namespace, it is assumed to be in the default namespace

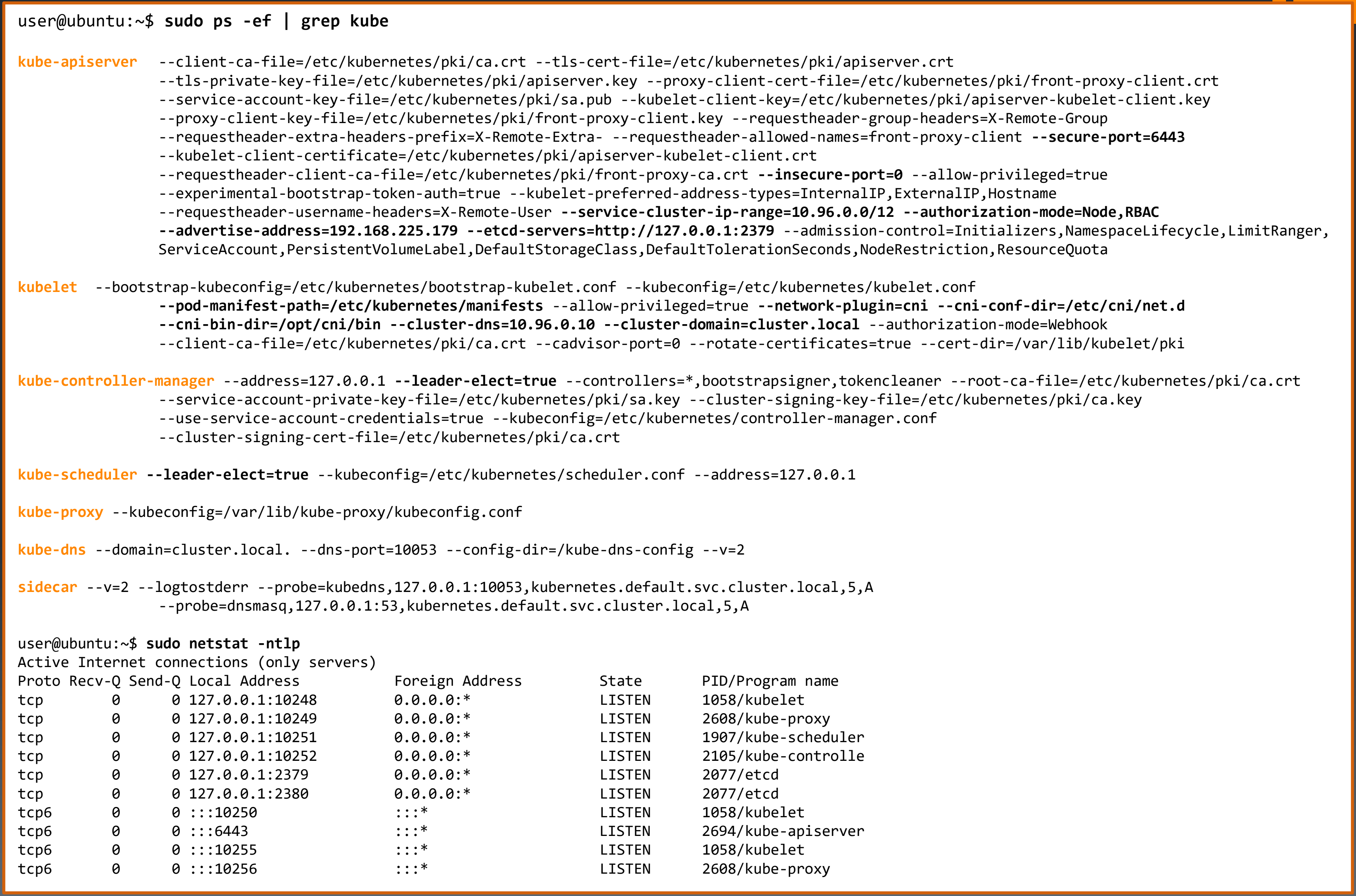
Pod Creation Flow



- The user creates a Pod via the API Server and the API server writes it to etcd
- The scheduler notices an “unbound” Pod and decides which node to run that Pod on
  - It writes that binding back to the API Server
- The Kubelet notices a change in the set of Pods that are bound to its node
  - It, in turn, runs the container via the container runtime (Docker)
- The Kubelet monitors the status of the Pod via the container runtime
  - As things change, the Kubelet will reflect the current status back to the API Server

Master Services

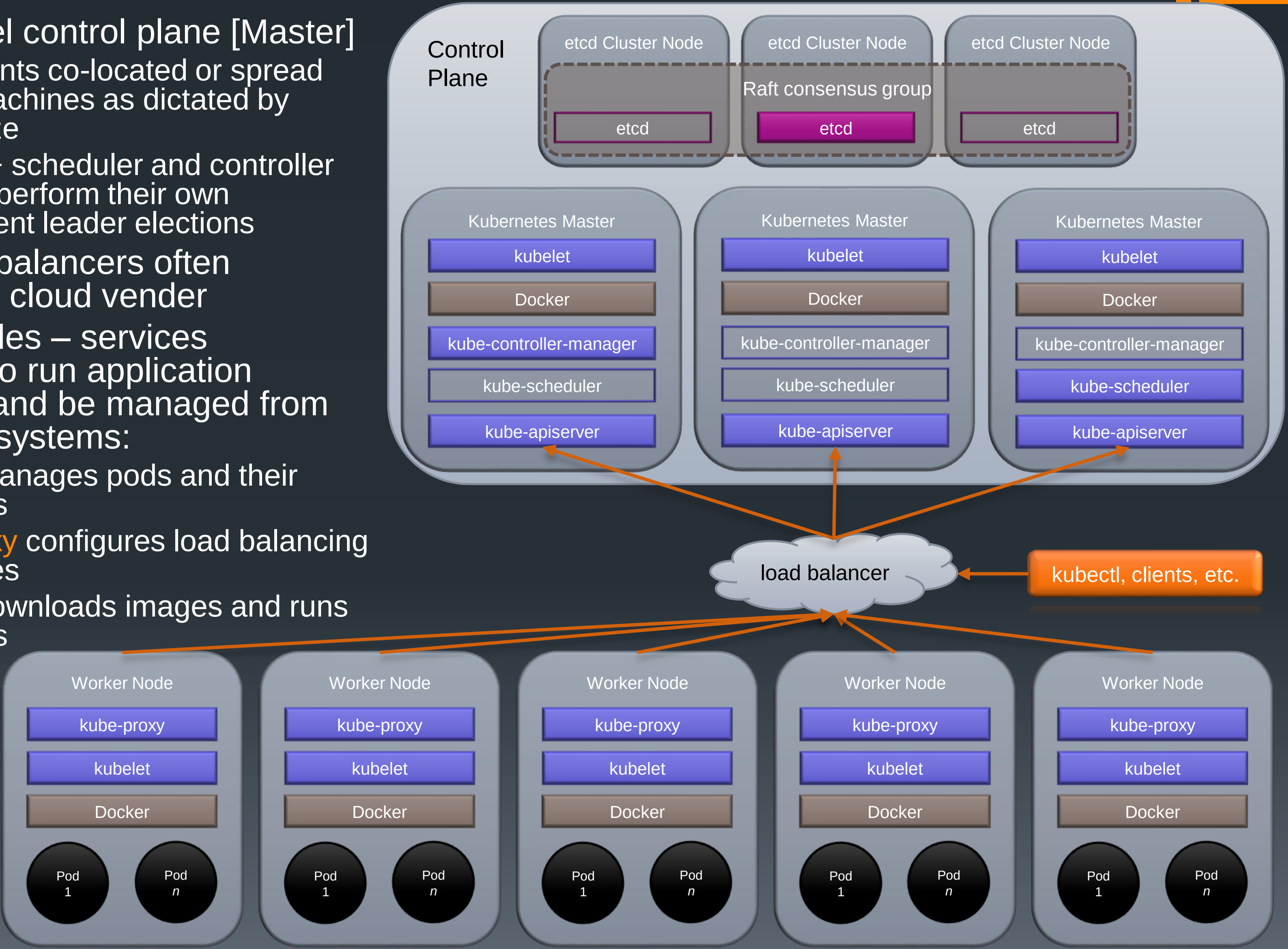
- Service listing from a standalone Kubernetes cluster:



```
user@ubuntu:~$ sudo ps -ef | grep kube

kube-apiserver  --client-ca-file=/etc/kubernetes/pki/ca.crt --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
                --tls-private-key-file=/etc/kubernetes/pki/apiserver.key --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
                --service-account-key-file=/etc/kubernetes/pki/sa.pub --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
                --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key --requestheader-group-headers=X-Remote-Group
                --requestheader-extra-headers-prefix=X-Remote-Extra- --requestheader-allowed-names=front-proxy-client --secure-port=6443
                --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
                --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt --insecure-port=0 --allow-privileged=true
                --experimental-bootstrap-token-auth=true --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
                --requestheader-username-headers=X-Remote-User --service-cluster-ip-range=10.96.0.0/12 --authorization-mode=Node,RBAC
                --advertise-address=192.168.225.179 --etcd-servers=http://127.0.0.1:2379 --admission-control=Initializers,NamespaceLifecycle,LimitRanger,
                ServiceAccount,PersistentVolumeLabel,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,ResourceQuota

kubelet  --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf
                --pod-manifest-path=/etc/kubernetes/manifests --allow-privileged=true --network-plugin=cni --cni-conf-dir=/etc/cni/net.d
                --cni-bin-dir=/opt/cni/bin --cluster-dns=10.96.0.10 --cluster-domain=cluster.local --authorization-mode=Webhook
                --client-ca-file=/etc/kubernetes/pki/ca.crt --cadvisor-port=0 --rotate-certificates=true --cert-dir=/var/lib/kubelet/pki

kube-controller-manager --address=127.0.0.1 --leader-elect=true --controllers=*,bootstrapsigner,tokencleaner --root-ca-file=/etc/kubernetes/pki/ca.crt
                --service-account-private-key-file=/etc/kubernetes/pki/sa.key --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
                --use-service-account-credentials=true --kubeconfig=/etc/kubernetes/controller-manager.conf
                --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt

kube-scheduler --leader-elect=true --kubeconfig=/etc/kubernetes/scheduler.conf --address=127.0.0.1

kube-proxy --kubeconfig=/var/lib/kube-proxy/kubeconfig.conf

kube-dns --domain=cluster.local. --dns-port=10053 --config-dir=/kube-dns-config --v=2

sidecar --v=2 --logtostderr --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local,5,A
                --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local,5,A

user@ubuntu:~$ sudo netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:10248         0.0.0.0:*               LISTEN      1058/kubelet
tcp        0      0 127.0.0.1:10249         0.0.0.0:*               LISTEN      2608/kube-proxy
tcp        0      0 127.0.0.1:10251         0.0.0.0:*               LISTEN      1907/kube-scheduler
tcp        0      0 127.0.0.1:10252         0.0.0.0:*               LISTEN      2105/kube-controlle
tcp        0      0 127.0.0.1:2379          0.0.0.0:*               LISTEN      2077/etcd
tcp        0      0 127.0.0.1:2380          0.0.0.0:*               LISTEN      2077/etcd
tcp6       0      0 :::10250                :::*                    LISTEN      1058/kubelet
tcp6       0      0 :::6443                 :::*                    LISTEN      2694/kube-apiserver
tcp6       0      0 :::10255                :::*                    LISTEN      1058/kubelet
tcp6       0      0 :::10256                :::*                    LISTEN      2608/kube-proxy
```

Scaled Architecture & HA Model

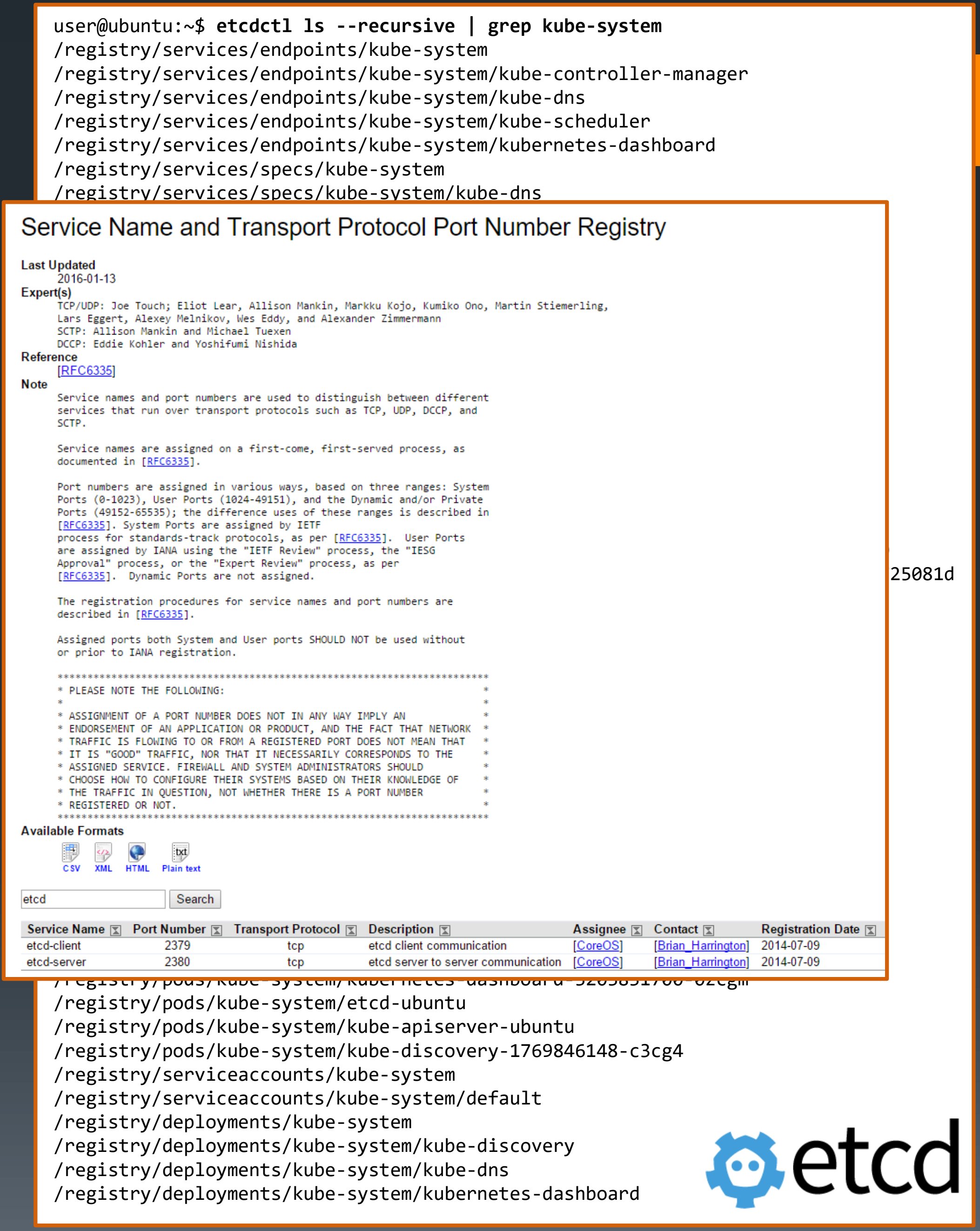
- Distributed watchable storage via etcd & Raft
- Cluster-level control plane [Master]
  - Components co-located or spread across machines as dictated by cluster size
  - K8s v1.2+ scheduler and controller manager perform their own independent leader elections
- Proxy load balancers often provided by cloud vender
- Worker Nodes – services necessary to run application containers and be managed from the master systems:
  - kubelet manages pods and their containers
  - kube-proxy configures load balancing via iptables
  - Docker downloads images and runs containers



etcd

- OpenSource distributed consistent key-value store for shared configuration and service discovery from CoreOS
  - Originally created as a distributed configuration information tool for CoreOS Linux clusters
    - Placed all of the configuration information for cluster nodes into in the key-value store that could be rapidly searched
  - Named for the /etc directory in Linux (which stores all of the config info about a server) and "d" for distributed systems
- etcd focuses on :
  - Simple: curl'able user-facing API (v3 uses gRPC)
  - Secure: automatic TLS w/ optional client cert auth
  - Fast: benchmarked 1000s of writes/s per instance
  - Resilient: properly distributed using Raft
- etcd is written in Go
- etcd uses the Raft consensus algorithm to manage a highly-available replicated log
  - etcd can recover from hardware failure and network partitions
- etcdctl is a simple command line client that can be used in scripts or to explore an etcd cluster
- Kubernetes uses etcd for storing and replicating data across the entire cluster
  - Cloud Foundry also uses etcd as their distributed key-value store
  - Docker supports etcd for and multi-host networking
- Used to track pod deployment, execution status, labels, endpoints, etc.

Over 500 projects on GitHub are making use of etcd in one form or another: https://github.com/coreos/etcd



```
user@ubuntu:~$ etcdctl ls --recursive | grep kube-system
/registry/services/endpoints/kube-system
/registry/services/endpoints/kube-system/kube-controller-manager
/registry/services/endpoints/kube-system/kube-dns
/registry/services/endpoints/kube-system/kube-scheduler
/registry/services/endpoints/kube-system/kubernetes-dashboard
/registry/services/specs/kube-system
/registry/services/specs/kube-system/kube-dns
```

Service Name and Transport Protocol Port Number Registry

Last Updated
2016-01-13

Expert(s)

```
TCP/UDP: Joe Touch; Eliot Lear, Allison Mankin, Markku Kojo, Kumiko Ono, Martin Stiemerling,
Lars Eggert, Alexey Melnikov, Wes Eddy, and Alexander Zimmermann
SCTP: Allison Mankin and Michael Tuexen
DCCP: Eddie Kohler and Yoshifumi Nishida
```

Reference
[RFC6335]

Note

```
Service names and port numbers are used to distinguish between different
services that run over transport protocols such as TCP, UDP, DCCP, and
SCTP.

Service names are assigned on a first-come, first-served process, as
documented in [RFC6335].

Port numbers are assigned in various ways, based on three ranges: System
Ports (0-1023), User Ports (1024-49151), and the Dynamic and/or Private
Ports (49152-65535); the difference uses of these ranges is described in
[RFC6335]. System Ports are assigned by IETF
process for standards-track protocols, as per [RFC6335].  User Ports
are assigned by IANA using the "IETF Review" process, the "IESG
Approval" process, or the "Expert Review" process, as per
[RFC6335].  Dynamic Ports are not assigned.

The registration procedures for service names and port numbers are
described in [RFC6335].

Assigned ports both System and User ports SHOULD NOT be used without
or prior to IANA registration.

************************************************************************
* PLEASE NOTE THE FOLLOWING:                                           *
*                                                                      *
* ASSIGNMENT OF A PORT NUMBER DOES NOT IN ANY WAY IMPLY AN             *
* ENDORSEMENT OF AN APPLICATION OR PRODUCT, AND THE FACT THAT NETWORK  *
* TRAFFIC IS FLOWING TO OR FROM A REGISTERED PORT DOES NOT MEAN THAT   *
* IT IS "GOOD" TRAFFIC, NOR THAT IT NECESSARILY CORRESPONDS TO THE     *
* ASSIGNED SERVICE. FIREWALL AND SYSTEM ADMINISTRATORS SHOULD          *
* CHOOSE HOW TO CONFIGURE THEIR SYSTEMS BASED ON THEIR KNOWLEDGE OF    *
* THE TRAFFIC IN QUESTION, NOT WHETHER THERE IS A PORT NUMBER          *
* REGISTERED OR NOT.                                                   *
************************************************************************
```

Available Formats

etcd | Search

Service Name	Port Number	Transport Protocol	Description	Assignee	Contact	Registration Date
etcd-client	2379	tcp	etcd client communication	[CoreOS]	[Brian_Harrington]	2014-07-09
etcd-server	2380	tcp	etcd server to server communication	[CoreOS]	[Brian_Harrington]	2014-07-09

```
25081d
/registry/pods/kube-system/kubernetes-dashboard-3203831700-02cgm
/registry/pods/kube-system/etcd-ubuntu
/registry/pods/kube-system/kube-apiserver-ubuntu
/registry/pods/kube-system/kube-discovery-1769846148-c3cg4
/registry/serviceaccounts/kube-system
/registry/serviceaccounts/kube-system/default
/registry/deployments/kube-system
/registry/deployments/kube-system/kube-discovery
/registry/deployments/kube-system/kube-dns
/registry/deployments/kube-system/kubernetes-dashboard
```

Google Kubernetes Engine Example Part 1 [docker/gcloud]

1. Create an application container
2. Push the container to the Google Container Registry
3. Create a Google Compute cluster to run containers on

```
user@ubuntu:~$ mkdir kubeex
user@ubuntu:~$ cd kubeex/
user@ubuntu:~/kubeex$ vim server.js
user@ubuntu:~/kubeex$ cat server.js
var http = require('http');

var handleRequest = function(request, response) {
  response.writeHead(200);
  response.end("Hello World!");
}

var www = http.createServer(handleRequest);
www.listen(8080);
user@ubuntu:~/kubeex$ vim Dockerfile
user@ubuntu:~/kubeex$ cat Dockerfile
FROM node:latest
COPY server.js /server.js
CMD node server.js
user@ubuntu:~/kubeex$ docker image build -t gcr.io/kubeex-160118/hello-node .
Sending build context to Docker daemon 3.072 kB
Sending build context to Docker daemon
Step 0 : FROM node:latest
...
 ---> 7b7b1aed5fed
Step 1 : COPY server.js /server.js
...
 ---> 578bc181cf79
Step 2 : CMD node server.js
...
 ---> b75c5d346af0
Successfully built b75c5d346af0
$ gcloud docker -- image push gcr.io/kubeex-160118/hello-node
The push refers to a repository [gcr.io/kubeex-160118/hello-node]
5194b882e159: Pushed
06ddfcaca7da: Layer already exists
...
a2ae92ffcd29: Mounted from google_containers/hyperkube-amd64
latest: digest: sha256:4c5722d9360a4b0450b46df06dc50fcf5e137944c10a38d9f4b84c88ca88e877 size: 2003
$ gcloud container clusters create hello-world --num-nodes 1 --machine-type g1-small
Creating cluster hello-world...done.
Created [https://container.googleapis.com/v1/projects/kubeex-160118/zones/us-west1-b/clusters/hello-world].
kubeconfig entry generated for hello-world.
NAME         ZONE        MASTER_VERSION  MASTER_IP        MACHINE_TYPE  NODE_VERSION  NUM_NODES  STATUS
hello-world  us-west1-b  1.7.6           104.196.246.216  g1-small      1.7.6         1          RUNNING
$ gcloud compute instances list
NAME                                        ZONE        MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP    STATUS
gke-hello-world-default-pool-e7535bd3-pskk  us-west1-b  g1-small                   10.138.0.2   35.185.199.37  RUNNING
```

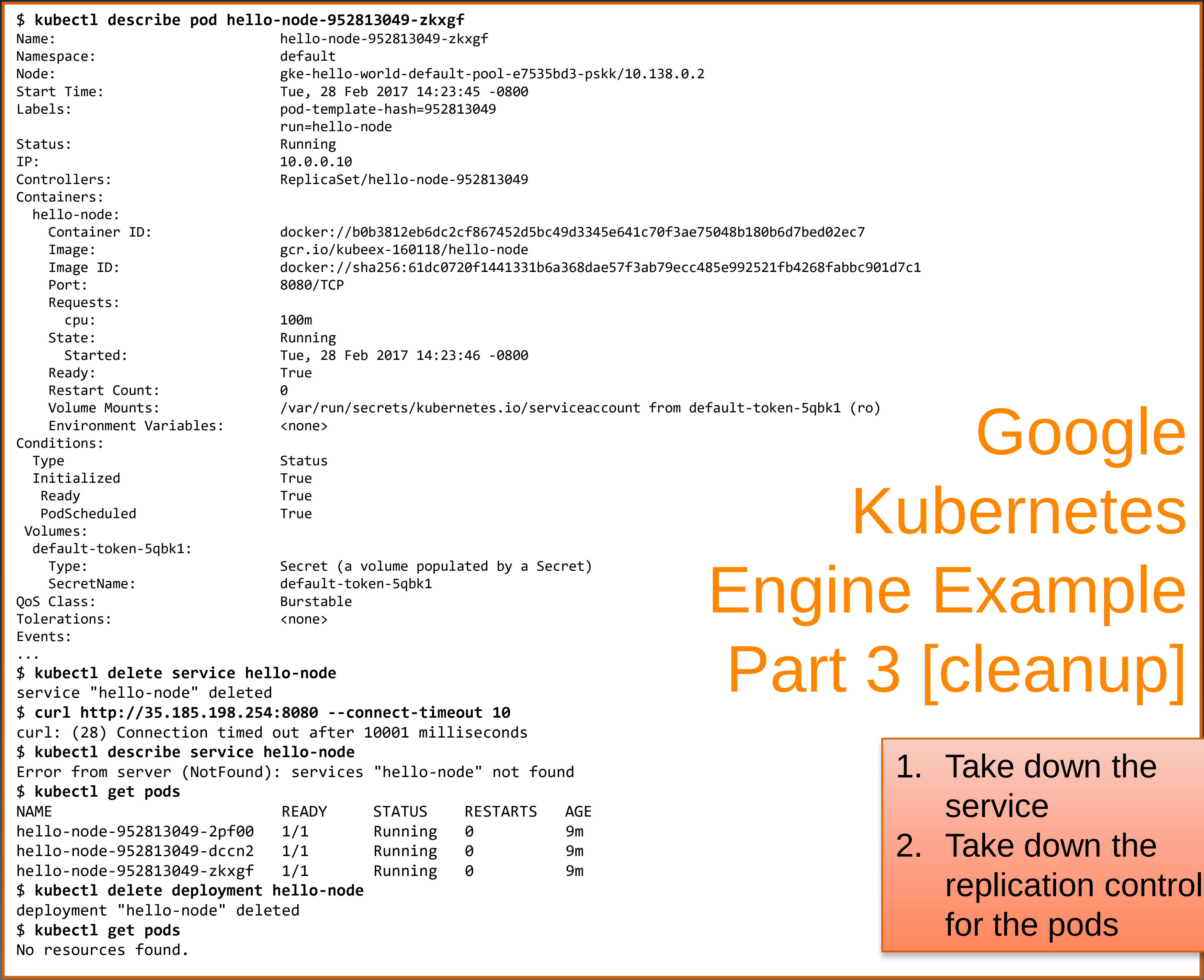
N.B. The gcloud command must be installed and configured to use a valid Google Cloud account

Google Kubernetes Engine Example Part 2 [kubectl]

```
$ kubectl run hello-node --image=gcr.io/kubeex-160118/hello-node --port=8080
deployment "hello-node" created

$ kubectl get deploy
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-node   1         1         1            0           14s

$ kubectl expose deployment hello-node --type="LoadBalancer"
service "hello-node" exposed

$ kubectl get services hello-node
NAME         CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
hello-node   10.3.254.37    35.185.198.254   8080:31487/TCP   1m

$ kubectl scale deployment hello-node --replicas=3
deployment "hello-node" scaled

$ kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
hello-node-952813049-2pf00    1/1       Running   0          49s
hello-node-952813049-dccn2    1/1       Running   0          49s
hello-node-952813049-zkxgf    1/1       Running   0          49s

$ kubectl describe service hello-node
Name:                      hello-node
Namespace:                 default
Labels:                    run=hello-node
Selector:                  run=hello-node
Type:                      LoadBalancer
IP:                        10.3.254.37
LoadBalancer Ingress:      35.185.198.254
Port:                      <unset>      8080/TCP
NodePort:                  <unset>      31487/TCP
Endpoints:                 10.0.0.11:8080,10.0.0.10:8080,10.0.0.9:8080
Session Affinity:          None
Events:
  FirstSeen    LastSeen     Count   From                  SubObjectPath   Type       Reason                Message
  ---------    --------     -----   ----                  -------------   --------   ------                -------
  12m          12m          1       {service-controller }                 Normal     CreatingLoadBalancer   Creating load balancer
  11m          11m          1       {service-controller }                 Normal     CreatedLoadBalancer    Created load balancer

$ curl http://35.185.198.254:8080
Hello World!
```

1. Run a one container pod
2. Expose the pod as a load balanced service
3. Dynamically scale up the pod count in the service

Google Kubernetes Engine Example Part 3 [cleanup]



```
$ kubectl describe pod hello-node-952813049-zkxgf
Name:                            hello-node-952813049-zkxgf
Namespace:                       default
Node:                            gke-hello-world-default-pool-e7535bd3-pskk/10.138.0.2
Start Time:                      Tue, 28 Feb 2017 14:23:45 -0800
Labels:                          pod-template-hash=952813049
                                 run=hello-node
Status:                          Running
IP:                              10.0.0.10
Controllers:                     ReplicaSet/hello-node-952813049
Containers:
  hello-node:
    Container ID:                docker://b0b3812eb6dc2cf867452d5bc49d3345e641c70f3ae75048b180b6d7bed02ec7
    Image:                       gcr.io/kubeex-160118/hello-node
    Image ID:                    docker://sha256:61dc0720f1441331b6a368dae57f3ab79ecc485e992521fb4268fabbc901d7c1
    Port:                        8080/TCP
    Requests:
      cpu:                       100m
    State:                       Running
      Started:                   Tue, 28 Feb 2017 14:23:46 -0800
    Ready:                       True
    Restart Count:               0
    Volume Mounts:               /var/run/secrets/kubernetes.io/serviceaccount from default-token-5qbk1 (ro)
    Environment Variables:       <none>
Conditions:
  Type                           Status
  Initialized                    True
   Ready                         True
   PodScheduled                  True
 Volumes:
  default-token-5qbk1:
    Type:                        Secret (a volume populated by a Secret)
    SecretName:                  default-token-5qbk1
QoS Class:                       Burstable
Tolerations:                     <none>
Events:
...
$ kubectl delete service hello-node
service "hello-node" deleted
$ curl http://35.185.198.254:8080 --connect-timeout 10
curl: (28) Connection timed out after 10001 milliseconds
$ kubectl describe service hello-node
Error from server (NotFound): services "hello-node" not found
$ kubectl get pods
NAME                         READY     STATUS    RESTARTS   AGE
hello-node-952813049-2pf00   1/1       Running   0          9m
hello-node-952813049-dccn2   1/1       Running   0          9m
hello-node-952813049-zkxgf   1/1       Running   0          9m
$ kubectl delete deployment hello-node
deployment "hello-node" deleted
$ kubectl get pods
No resources found.
```

1. Take down the service
2. Take down the replication controller for the pods

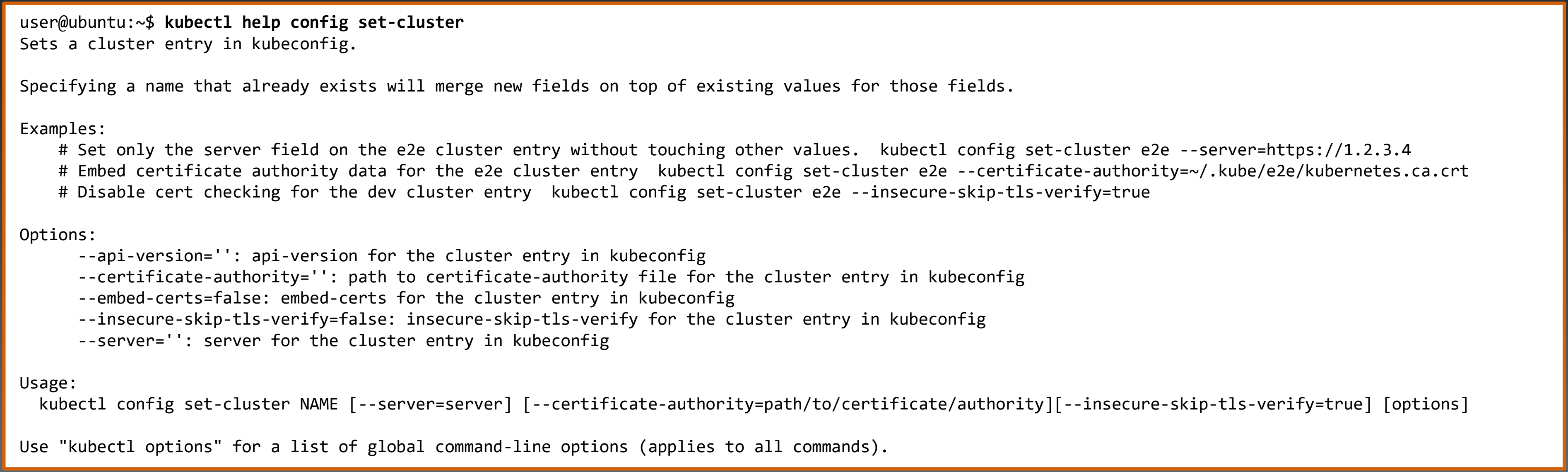
kubectl Setup

- The kubectl command line tool is the generic front end for Kubernetes
  - Much like `docker` is to Docker
- Written in Go
  - Binaries for download on OS X, Linux and Windows
  - Build from Go source
  - Download as part of the Google Cloud SDK
  - Install with brew on OS X
- Reads config from `~/.kube/config`
  - Configuration controlled by the `config` subcommand:
    - `$ kubectl config view`
    - `$ kubectl config set-cluster`
    - `$ kubectl config set-context`

```
user@ubuntu:~$ wget https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
--2017-09-27 08:47:17--  https://storage.googleapis.com/kubernetes-release/release/v1.8.1/bin/linux/amd64/kubectl
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.5.112, 2607:f8b0:4005:808::2010
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.5.112|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 50359943 (48M) [application/octet-stream]
Saving to: ‘kubectl’

kubectl              100%[====================================================>]  48.03M  297KB/s    in 1m 56s
2017-07-27 08:59:55 (266.8 KB/s) - ‘kubectl’ saved [50359943/50359943]

user@ubuntu:~$ chmod a+x kubectl
user@ubuntu:~$ ./kubectl get nodes
NAME        STATUS    AGE
127.0.0.1   Ready     48m
```

kubectl Config: Cluster and Context

- A kubectl cluster is a Kubernetes API endpoint for a given Kubernetes cluster
  - kubectl can be configured with multiple clusters
  - Only one context is active at a time and each context points to precisely one cluster
  - `kubectl config set-cluster` sets a cluster entry in kubeconfig [`~.kube/config`]
  - `kubectl config set-cluster NAME [--server=server] [--api-version=apiversion] [--certificate-authority=path/to/certficate/authority] [--insecure-skip-tls-verify=true]`
    - Example
      - `$ kubectl config set-cluster e2e --server=https://15.22.31.4`
- A kubectl context defines the cluster connection metadata to be used with kubectl commands
- Set with: `kubectl config set-context`
  - Sets a context entry in kubeconfig [`~.kube/config`]
  - Specifying a name that already exists will merge new fields on top of existing values for those fields
  - `kubectl config set-context NAME [--cluster=cluster_nickname] [--user=user_nickname] [--namespace=namespace]`
    - Example
      - `$ kubectl config set-context gce --user=cluster-admin`



```
user@ubuntu:~$ kubectl help config set-cluster
Sets a cluster entry in kubeconfig.

Specifying a name that already exists will merge new fields on top of existing values for those fields.

Examples:
    # Set only the server field on the e2e cluster entry without touching other values.  kubectl config set-cluster e2e --server=https://1.2.3.4
    # Embed certificate authority data for the e2e cluster entry  kubectl config set-cluster e2e --certificate-authority=~/.kube/e2e/kubernetes.ca.crt
    # Disable cert checking for the dev cluster entry  kubectl config set-cluster e2e --insecure-skip-tls-verify=true

Options:
      --api-version='': api-version for the cluster entry in kubeconfig
      --certificate-authority='': path to certificate-authority file for the cluster entry in kubeconfig
      --embed-certs=false: embed-certs for the cluster entry in kubeconfig
      --insecure-skip-tls-verify=false: insecure-skip-tls-verify for the cluster entry in kubeconfig
      --server='': server for the cluster entry in kubeconfig

Usage:
  kubectl config set-cluster NAME [--server=server] [--certificate-authority=path/to/certificate/authority][--insecure-skip-tls-verify=true] [options]

Use "kubectl options" for a list of global command-line options (applies to all commands).
```

kubectl Use

- Syntax
  - `kubectl [command] [TYPE] [NAME] [flags]`
    - `command` – the operation to perform on the named resource(s)
    - `TYPE` – specifies the resource type
      - Specify singular, plural or abbreviated forms (`deployment`, `deployments`, `deploy`)
    - `NAME` – specifies the name of the resource
    - `flags` – optional flags
- Basic Resource Commands:
  - `get`
    - Display one or many resources
  - `describe`
    - Show details of a specific resource or group of resources
  - `create` (covered later)
    - Create a resource by filename or stdin
  - `delete`
    - Delete resources by filenames, stdin, resources and names, or by resources and label selector.
  - `logs`
    - Print the logs for a container in a pod
  - `attach`
    - Attach to a running container
  - `exec`
    - Execute a command in a container
- kubectl includes autocompletion support
  - Add it to your current shell
    - `source <(kubectl completion bash)`
  - To add to your profile so it is automatically loaded in future shells :
    - `echo "source <(kubectl completion bash)" >> ~/.bashrc`

Summary

- Kubernetes is a system for managing containerized applications across multiple hosts
- Kubernetes is itself a microservice based system and some or all of it may be executed via containers
- Kubernetes relies on etcd to maintain distributed watchable state
- Kubernetes can be deployed in a wide range of environments including bare metal OS, VM and cloud
- Kubernetes has been production ready for less than a year and is still changing and developing in significant ways

Lab 10

- Using Kubernetes with kubectl

11: Pods and Configs

Objectives

- Understand Kubernetes resource types
- Learn how to define resources with YAML files
- Define Pod and the operation of Pods on Kubernetes
- Understand the relationship between images, containers and Pods
- Start, Stop and monitor Pods

Kubernetes Concepts

- Cluster
  - A cluster is a set of physical or virtual machines and other infrastructure resources used by Kubernetes to run your applications
- Node
  - A node is a physical or virtual machine running Kubernetes, onto which pods can be scheduled
- Pod
  - Pods are colocated groups of application containers
  - The smallest deployable units that can be created, scheduled, and managed with Kubernetes
  - Pods can be created individually, but it's recommended that you use a replication controller even if creating a single pod
- Replica Set
  - Replica Sets manage the lifecycle of pods
  - They ensure that a specified number of pods are running at any given time, by creating or killing pods as required
- Deployment
  - Allow initial deployment of pods through replica sets and rolling upgrades
- Service
  - Services provide a single, stable name and address for a set of pods
  - Act as basic load balancers
  - Services select the Pods to address using labels
- Label
  - Labels are key:value pairs used to organize and select groups of objects
- Namespace
  - A mechanism to partition resources created by users into a logically named group
  - Allows separate user communities to work in isolation

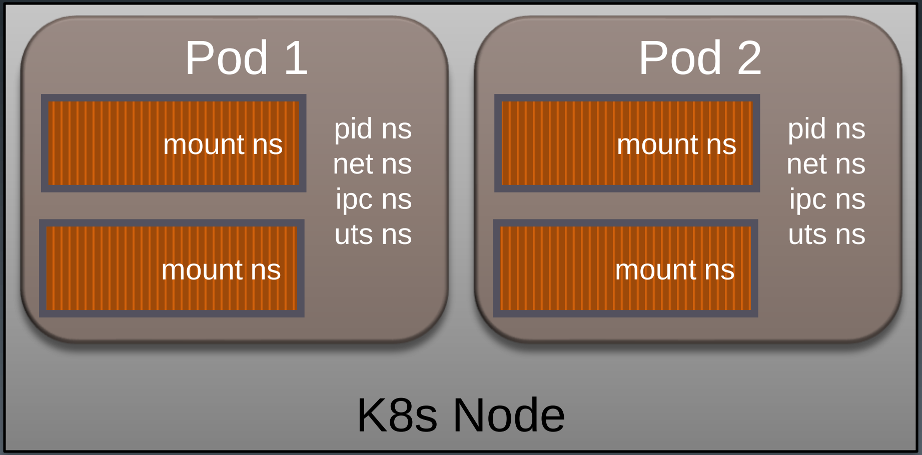
Kubernetes Process Control & Scheduling

- Desired State vs. Actual State
  - You tell Kubernetes the state you desire (run three copies of the nginx container)
    - Spec
  - Kubernetes works to bring the actual state inline with your desired state
    - Status
    - Replica Sets restart failed replicas, etc.
  - This is the eventual consistency model at work at a higher level of abstraction
- The Kubernetes scheduler spreads pods across the cluster and uses different nodes for matching pod replicas when possible
- Many ancillary Kubernetes services may be deployed, usually themselves in containers
  - The default deployment model for the kubelet will likely be changed to in-container in the near future
    - Docker installation example from section 2 uses containerized kubelet

Kubernetes Goal:
Make
Status = Spec

Pods

- Many times a set of containers run as a team on the same host
  - A data loader with a data server
  - A log compressor/saver process with a log server
- These containers usually need to be located together
  - You want to ensure that they do not become separated during dynamic placement
- In Kubernetes a pod is a set of containers that are placed and scheduled together as a unit on a Kubernetes Node
  - Kubernetes Node is a worker computer in the cluster
- By working to place a group of pods, Kubernetes can pack lots of work onto a node in a reliable way
- In Kubernetes, Pods are defined using YAML (or, the more recently supported JSON) files



Config File Commonalities

- YAML formatted
- Main parts
  - # Comment
  - apiVesion
    - Kubernetes API version to build with
    - Presently should always be v1
  - kind
    - Kubernetes object type
    - Pod, Deployment, Service, Namespace, etc…
  - metadata
    - Name, labels and other data describing the object
  - spec
    - Instructions for building the object
    - The desired state
- Construction
  - kubectl create -f filename

```yaml
# mypod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
spec:
  containers:
  - name: web-container
    image: bitnami/apache:latest
    ports:
    - containerPort: 80
```

Pod Definition

- The simplest pod definition describes the deployment of a single container
- For example, an nginx web server pod might be defined as such:

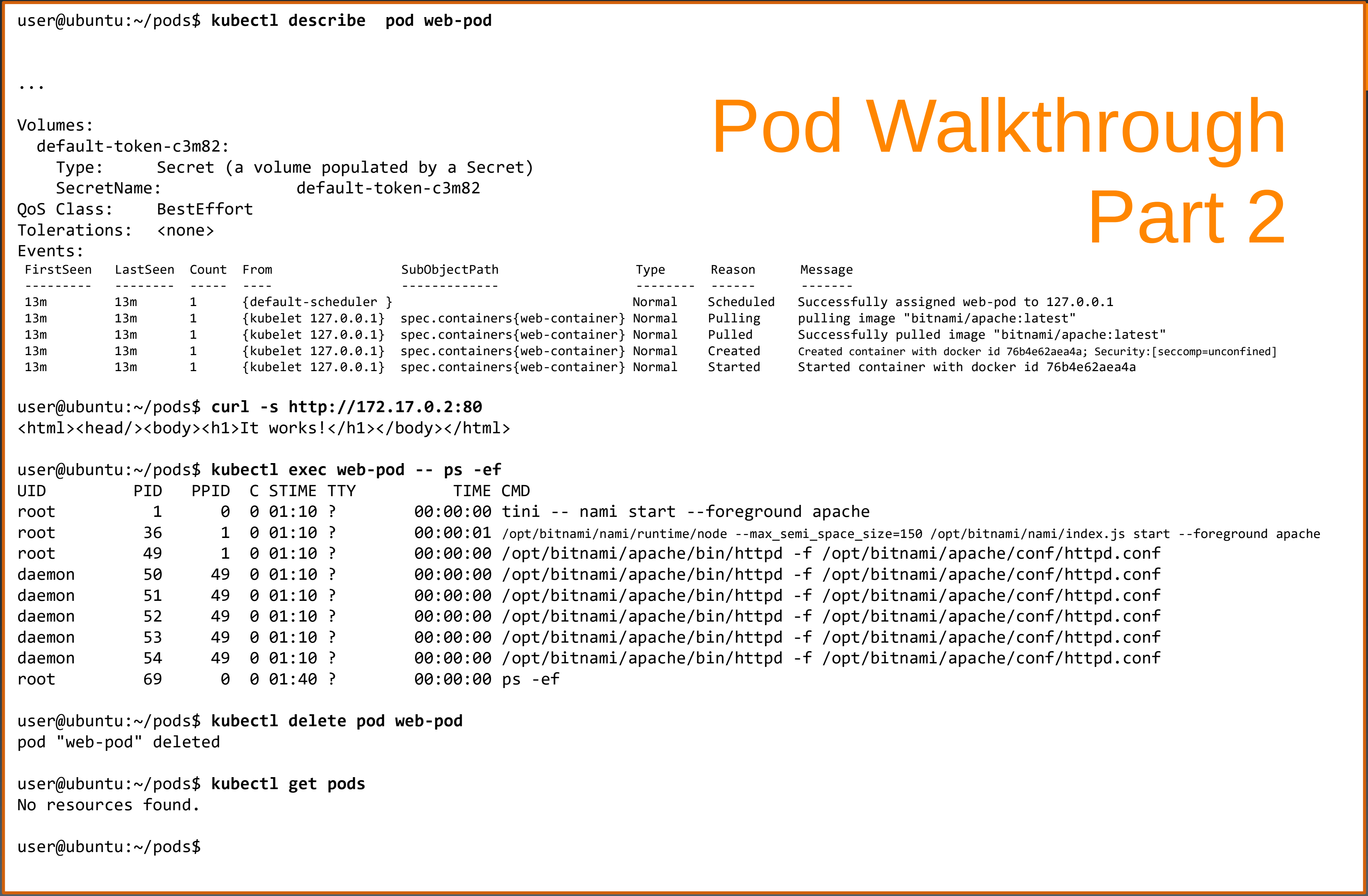
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

- A pod definition is a declaration of a desired state
- Desired state is a very important concept in the Kubernetes model
- Many things present a desired state to the system, and it is Kubernetes' responsibility to make sure that the current state matches the desired state
    - For example, when you create a Pod, you declare that you want the containers in it to be running
    - If the containers happen to not be running (e.g. program failure, ...), Kubernetes will continue to (re-)create them for you in order to drive them to the desired state
        - Spec describes build time (what we want), and Status describes runtime (what we have).
    - This process continues until the Pod is deleted

Pod Walkthrough Part 1

```
user@ubuntu:~/pods$ vim web.yaml
user@ubuntu:~/pods$ cat web.yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
spec:
  containers:
  - name: web-container
    image: bitnami/apache:latest
    ports:
    - containerPort: 80
user@ubuntu:~/pods$ kubectl create -f web.yaml
pod "web-pod" created
user@ubuntu:~/pods$ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
web-pod   0/1       Pending   0          13s
user@ubuntu:~/pods$ kubectl describe pod web-pod
Name:         web-pod
Namespace:    default
Image(s):     bitnami/apache:latest
Node:         127.0.0.1/127.0.0.1
Start Time:   Fri, 20 Jan 2017 06:29:30 -0800
Labels:       <none>
Status:       Running
IP:           172.17.0.2
Controllers:  <none>
Containers:
  web-container:
    Container ID:            docker://76b4e62aea4a1e72ed7c26752194541453d8f4e9d99e681b86ef5f2ad969412b
    Image:                   bitnami/apache:latest
    Image ID:                docker-pullable://bitnami/apache@sha256:4b7dc6a2b0e06c787bc6f38472926b270cb97b14fd1e47637a469ab8a56f9682
    Port:                    80/TCP
    State:                   Running
      Started:               Fri, 20 Jan 2017 06:29:55 -0800
    Ready:                   True
    Restart Count:           0
    Volume Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-c3m82 (ro)
    Environment Variables:   <none>
Conditions:
 Type         Status
 Initialized  True
 Ready        True
 PodScheduled True
...
```

Pod Walkthrough Part 2



```
user@ubuntu:~/pods$ kubectl describe  pod web-pod


...

Volumes:
  default-token-c3m82:
    Type:      Secret (a volume populated by a Secret)
    SecretName:              default-token-c3m82
QoS Class:     BestEffort
Tolerations:   <none>
Events:
 FirstSeen   LastSeen   Count   From                     SubObjectPath                   Type      Reason      Message
 ---------   --------   -----   ----                     -------------                   --------  ------      -------
 13m         13m        1       {default-scheduler }                                     Normal    Scheduled   Successfully assigned web-pod to 127.0.0.1
 13m         13m        1       {kubelet 127.0.0.1}      spec.containers{web-container}  Normal    Pulling     pulling image "bitnami/apache:latest"
 13m         13m        1       {kubelet 127.0.0.1}      spec.containers{web-container}  Normal    Pulled      Successfully pulled image "bitnami/apache:latest"
 13m         13m        1       {kubelet 127.0.0.1}      spec.containers{web-container}  Normal    Created     Created container with docker id 76b4e62aea4a; Security:[seccomp=unconfined]
 13m         13m        1       {kubelet 127.0.0.1}      spec.containers{web-container}  Normal    Started     Started container with docker id 76b4e62aea4a

user@ubuntu:~/pods$ curl -s http://172.17.0.2:80
<html><head/><body><h1>It works!</h1></body></html>

user@ubuntu:~/pods$ kubectl exec web-pod -- ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 01:10 ?        00:00:00 tini -- nami start --foreground apache
root          36       1  0 01:10 ?        00:00:01 /opt/bitnami/nami/runtime/node --max_semi_space_size=150 /opt/bitnami/nami/index.js start --foreground apache
root          49       1  0 01:10 ?        00:00:00 /opt/bitnami/apache/bin/httpd -f /opt/bitnami/apache/conf/httpd.conf
daemon        50      49  0 01:10 ?        00:00:00 /opt/bitnami/apache/bin/httpd -f /opt/bitnami/apache/conf/httpd.conf
daemon        51      49  0 01:10 ?        00:00:00 /opt/bitnami/apache/bin/httpd -f /opt/bitnami/apache/conf/httpd.conf
daemon        52      49  0 01:10 ?        00:00:00 /opt/bitnami/apache/bin/httpd -f /opt/bitnami/apache/conf/httpd.conf
daemon        53      49  0 01:10 ?        00:00:00 /opt/bitnami/apache/bin/httpd -f /opt/bitnami/apache/conf/httpd.conf
daemon        54      49  0 01:10 ?        00:00:00 /opt/bitnami/apache/bin/httpd -f /opt/bitnami/apache/conf/httpd.conf
root          69       0  0 01:40 ?        00:00:00 ps -ef

user@ubuntu:~/pods$ kubectl delete pod web-pod
pod "web-pod" deleted

user@ubuntu:~/pods$ kubectl get pods
No resources found.

user@ubuntu:~/pods$
```

What is YAML

- Rhymes with camel
- Stands for YAML Ain't Markup Language
  - Was originally Yet Another Markup Language but the new reverse backronym and antiestablishment attitude is apparently cooler
- Human-readable data serialization format that takes concepts from programming languages and ideas from XML and electronic mail (RFC 2822)
- Designed to be easily mapped to data types common to most high-level languages:
  - List [array/set]
  - Associative array [hash/object/map]
  - Scalar
- Friendly to ad hoc grep/Python/Perl/Ruby operations
- Eschews enclosures (quotation marks, brackets, braces, open/close-tags, etc.)
- Data structure hierarchy is maintained by outline indentation
  - Spaces only, no tabs!

Docker uses YAML in some human targeted file areas but most files are JSON formatted

```
---
receipt:     VMW Invoice
date:        2012-08-06
customer:
    given:   Dorothy
    family:  Gale
```

YAML Quick Reference

- YAML offers an indented and an in-line styles
- Lists
  - Conventional block format uses a hyphen+space to begin a new item in list.

```
--- # Favorite movies
- Casablanca
- North by Northwest
- The Man Who Wasn't There
```

  - Optional inline format is delimited by comma+space and enclosed in brackets (similar to JSON)

```
--- # Shopping list
[milk, pumpkin pie, eggs, juice]
```

- Associative arrays
  - Keys are separated from values by a colon+space. Indented Blocks use new lines to separate key: value pairs; Inline Blocks use comma+space to separate the key: value pairs between braces

```
--- # Indented Block
  name: John Smith
  age: 33
--- # Inline Block
{name: John Smith, age: 33}
```

- Block literals
  - Strings do not require quotation
  - Newlines preserved (By default first line indent and trailing space is stripped though other behavior can be specified)

```
--- |
  There once was a man from Ealing
  Who got on a bus to Darjeeling
      It said on the door
      "Please don't spit on the floor“
  So he carefully spat on the ceiling
```

  - Newlines folded (Folded text converts newlines to spaces and removes leading whitespace)

```
--- >
  Wrapped text
  will be folded
  into a single
  paragraph

  Blank lines  denote
  paragraph breaks
```

- Hierarchical combinations of elements
  - Lists of associative arrays

```
- {name: John Smith, age: 33}
- name: Mary Smith
  age: 27
```

  - Associative arrays of lists

```
men: [John Smith, Bill Jones]
women:
  - Mary Smith
  - Susan Williams
```

Pod Configuration Files

- Pods can be configured using YAML or JSON configuration files
  - Much like Docker Compose
  - kubectl create -f mynewpod.yml
- Fields for Pod configuration include:
  - apiVersion: Currently v1
  - kind: Always Pod
  - metadata: An object containing:
    - name: The name of this pod (required if generateName is not specified)
    - labels: Optional arbitrary key:value pairs used for grouping and targeting by other resources and services
  - spec: The pod specification
    - volumes[]: List of volumes that can be mounted by containers in the pod
    - restartPolicy: applies to all containers in the pod
    - nodeSelector: node label required for pod to run on a given node
    - nodeName: specific node to schedule pod on
    - terminationGracePeriodSeconds: number of seconds node is given to shutdown before kill
    - hostNetwork: run pod in host net namespace
    - hostPID: run pod in host PID namespace
    - hostIPC: run pod in host IPC namespace
    - imagePullSecrets: secrets to use for pulling images (e.g. DockerConfig for private reg access)
    - containers[]: A list of containers belonging to the pod
      - name: Name of the container
      - image: Docker image name
      - command[]: command line (like Docker ENTRYPOINT)
      - args[]: entry point arguments (like Docker CMD)
      - env[]: env vars
      - ports[]: port mappings
        - containerPort: port to expose
        - protocol: port protocol
        - hostIP: host IP to bind to
        - hostPort: host port to bind to
      - volumes[]: volumes which can be mounted
      - restartPolicy: Always, OnFailure, Never
      - workingDir: cwd for container proc

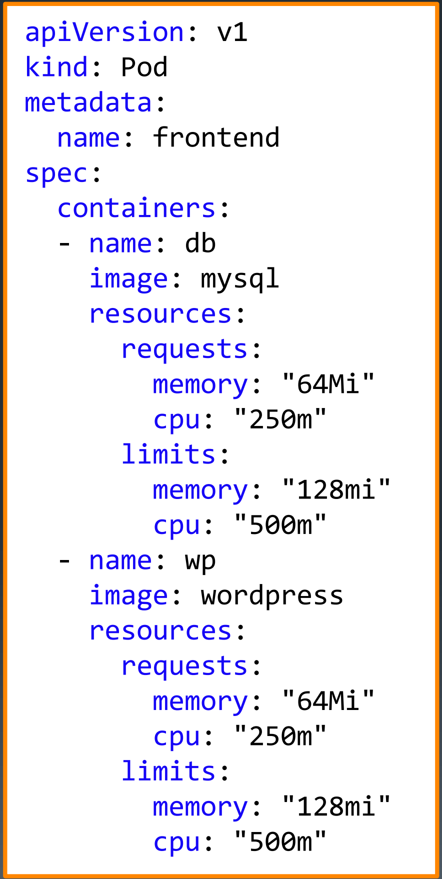
```
user@ubuntu:~/pods$ cat hello.yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
spec:  # specification of the pod’s contents
  restartPolicy: Never
  containers:
  - name: hello
    image: "ubuntu:14.04"
    env:
    - name: MESSAGE
      value: "hello world"
    command: ["/bin/sh","-c"]
    args: ["/bin/echo \"${MESSAGE}\""]

user@ubuntu:~/pods$ kubectl create -f hello.yaml
pod "hello-world" created
user@ubuntu:~/pods$ kubectl get pod
NAME          READY     STATUS     RESTARTS   AGE
hello-world   0/1       Pending    0          15s
user@ubuntu:~/pods$ kubectl logs hello-world
hello world
user@ubuntu:~/pods$ kubectl get pod
user@ubuntu:~/pods$
```

Pod Spec Reference:
https://kubernetes.io/docs/api-reference/v1/definitions/#_v1_podspec

Container Spec Reference:
https://kubernetes.io/docs/api-reference/v1/definitions/#_v1_container

Resource Limits

- You can specify the desired CPU and memory for containers in specs to ensure the scheduler chooses a node with the appropriate resources
- CPU and memory are each a resource type
  - CPU is specified in units of cores
    - spec.container[].resources.requests.cpu
  - Memory is specified in units of bytes
    - spec.container[].resources.requests.memory
- Each container of a Pod can optionally specify constraints
  - spec.container[].resources.limits.cpu
  - spec.container[].resources.limits.memory
- Default values are cluster configured
  - If value of requests is not specified, they are set to be equal to limits by default
  - Resource limits must be greater than or equal to resource requests
- requests/limits can only be specified on individual containers
  - Pod request/limits are simply the sum of their container request/limits
- If the scheduler cannot find any node where a pod can fit, then the pod will remain unscheduled until a place can be found



```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128mi"
        cpu: "500m"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128mi"
        cpu: "500m"
```

Pod Phases

- Phases
  - Describe the macro states in the lifecycle of a Kubernetes resource
  - Not a comprehensive state machine
- Pod Phase
  - Pending
    - The pod has been accepted by the system, but one or more of the container images has not been created
    - Includes time before being scheduled as well as time spent downloading images over the network
  - Running
    - The pod has been bound to a node, and all of the containers have been created
    - At least one container is still running, or is in the process of starting or restarting
  - Succeeded
    - All containers in the pod have terminated in success, and will not be restarted
  - Failed
    - All containers in the pod have terminated, at least one container has terminated in failure (exited with non-zero exit status or was terminated by the system)
  - Unknown
    - For some reason the state of the pod could not be obtained, typically due to an error in communicating with the host of the pod

https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/

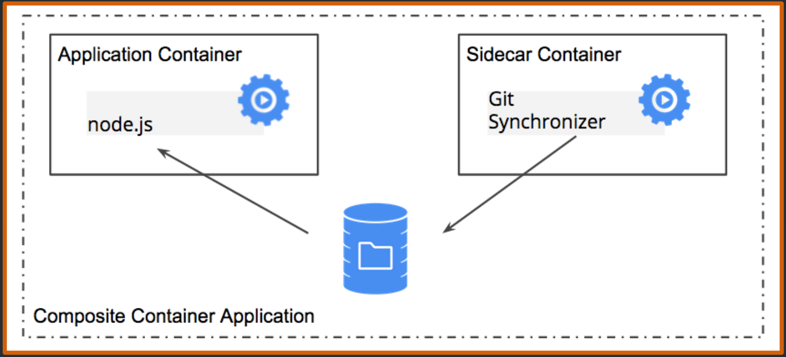
Pause

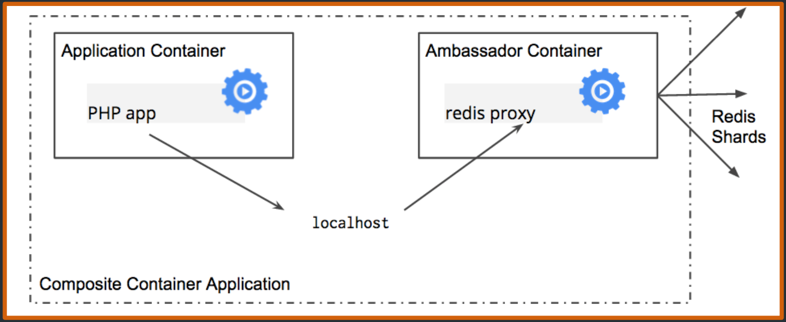
- The Pause container is often referred to as the pod infrastructure container and is used to set up and hold the networking namespace and resource limits for each pod

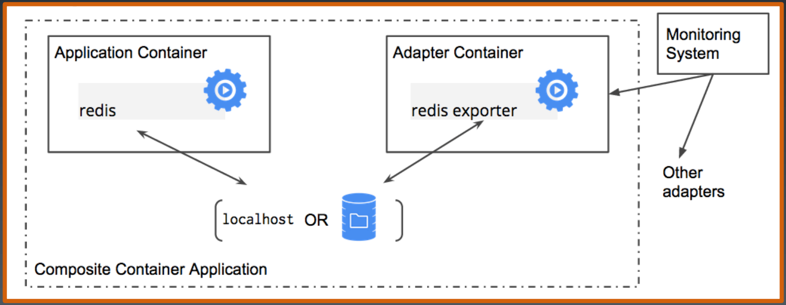
```
user@ubuntu:~/pods$ cat long.yaml
apiVersion: v1
kind: Pod
metadata:
  name: long-running
spec:  # specification of the pod’s contents
  containers:
  - name: long
    image: "ubuntu:14.04"
    command: ["/usr/bin/tail", "-f", "/dev/null"]


user@ubuntu:~/pods$ kubectl create -f long.yaml
pod "long-running" created
user@ubuntu:~/pods$ kubectl get pod
NAME           READY     STATUS    RESTARTS   AGE
long-running   0/1       Running   0          3s
user@ubuntu:~/pods$ sudo docker container ls -f "name=long-running"
CONTAINER ID  IMAGE                                    COMMAND               CREATED  STATUS   NAMES
8fe590c80a12  ubuntu:14.04                             "/usr/bin/tail -f /de" 2m ago   Up 2m   k8s_long.94251326_long...
8bf3dbb786cc  gcr.io/google_containers/pause-amd64:3.0 "/pause"               2m ago   Up 2m   k8s_POD.6d00e006_long...
```

Pod Patterns

- Patterns
  - Sidecar
    - Sidecars extend and enhance the "main" container in the Pod
    - Example: Nginx web server container; add a container that syncs the file system with a git repository, share the file system between the containers and you have built Git push-to-deploy
  - Ambassador
    - Ambassadors proxy a Pod local connection to the world outside
    - Example: Redis cluster with read-replicas and a single write master; create a Pod that groups the main application with a Redis ambassador container which splits reads and writes, sending them on to the appropriate servers
  - Adapter
    - Adapter containers standardize and normalize output
    - Example: A task monitoring N different applications where each application has a different way of exporting monitoring data (e.g. JMX, StatsD, application specific statistics) but every monitoring system expects a consistent and uniform data model for the monitoring data it collects







- Brendan Burns, Distinguished Engineer at Microsoft and former Software Engineer at Google
https://research.google.com/pubs/pub45406.html

Summary

- Kubernetes resource types provide the basic building blocks for Kubernetes based applications
- YAML files are used to specify resources and the metadata associated with them
- Pod are the unit of application deployment in Kubernetes
- Pods are atomic and scheduled together on a single node as a unit
- A Pod may contain one or more user defined containers
- A Pod has an infrastructure container (Pause) used as a place holder for the Pods namespaces and CGroups
- Pods share network and IPC namespaces
- kubectl offers several commands to work with pods
  - create
  - delete
  - describe
  - get
  - logs
- A container engine is used behind the scenes to execute and manage containers (e.g. Docker)
- The container engine is accessible directly for debugging and certain management chores

Lab 11

- Pods and Config Files