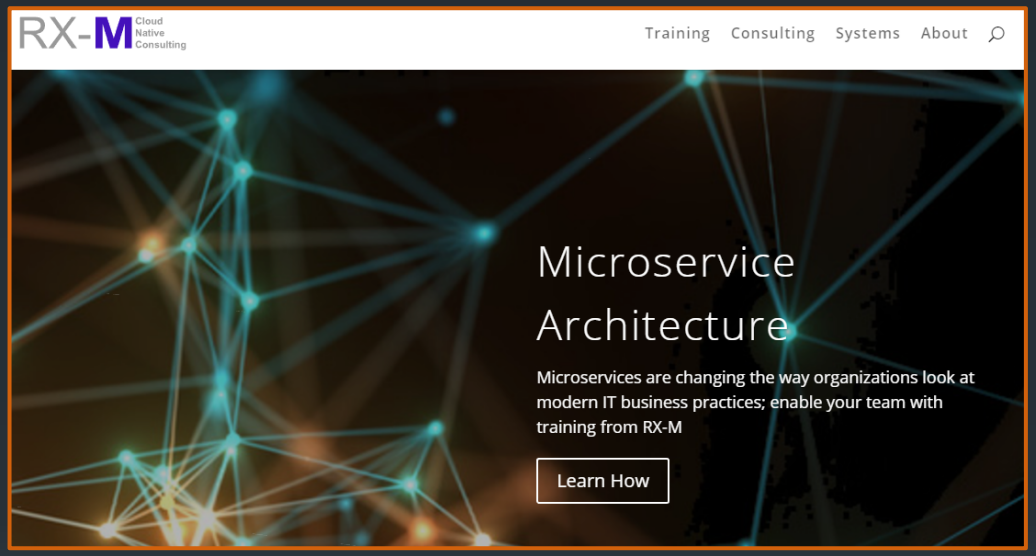
RX-M Cloud Native Consulting

Advanced Kubernetes

Orchestration of Containers in depth

RX-M Cloud Native Advisory, Consulting and Training

- Microservice Oriented
  - Microservices Foundation [3 Day]
  - Building Microservices on AWS [3 Day]
  - Building Microservices with Go [3 Day]
  - Building Microservices with Thrift [2 Day]
  - Building Microservices with gRPC [2 Day]
- Container Packaged
  - Docker Foundation [3 Day]
  - Docker Advanced [2 Day]
  - OCI [2 Day]
  - CNI [2 Day]
  - Containerd [2 Day]
  - Rocket [2 Day]
- Dynamically Managed
  - Cloud Native Container Networking and Cisco ACI [3 Day]
  - Docker Orchestration (Compose/Swarm) [2 Day]
  - Kubernetes Foundation [2 Day]
  - Kubernetes Advanced [3 Day]
  - Mesos Foundation [2 Day]
  - MANTL [2 Day]
  - Nomad [2 Day]



kubernetes CERTIFIED SERVICE PROVIDER

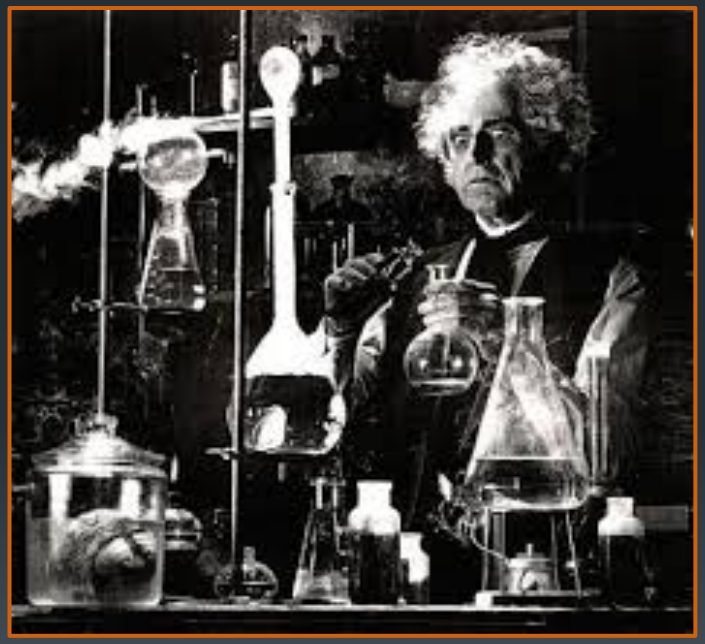
RX-M Cloud Native Consulting

Administrative Info

- Length: 2 Days
- Format: Lecture/Labs/Discussion
- Schedule: 9:00AM – 5:00PM
  15 minute break, AM & PM
  1 hour lunch at noon
  Lab time at the end of each AM and PM session
- Location: Fire exits, Restrooms, Security, other matters
- Attendees: Name/Role/Experience/Goals for the Course

Lecture and Lab

- Our Goals in this class are two fold:
  1. Familiarize you with the general concepts and ecosystem surrounding the course topic
     - This is the primary purpose of the lecture/discussion sessions
  2. Give you practical experience working with the course topic
     - This is the primary purpose of the labs



Overview

Day One

1. Kubernetes Architecture
2. Advanced Pod configs and kubelet
3. Scheduling

Day Two

4. Services and Networking
5. DNS and Service Discovery
6. API and Security

Prerequisites:

RX-M Kubernetes Foundation and Docker Foundation or equivalent experience

Equivalent experience:

- Basic familiarity with Kubernetes cluster components and kubectl client subcommands
- Basic familiarity with all docker client subcommands
- Clear understanding of containers, isolation, container volumes, and port mapping

Day 1

1. Kubernetes Architecture
2. Advanced Pod configs and kubelet
3. Scheduling

1. Kubernetes Architecture

Objectives

- Understand K8s overall design
- Understand how K8s leverages containers
- Review primary container components
- Review primary K8s components
- Overview K8s High Availability and Federation Models

Kubernetes

- Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.
- K8s allows you to:
  - Deploy your applications quickly and predictably
  - Scale your applications on the fly
  - Seamlessly roll out new features
  - Optimize use of your hardware by using only the resources you need
- K8s is portable (public, private, hybrid clouds), extensible, and self-healing.

- The *Old Way* to deploy applications was to install the applications on a host using the operating system package manager. This had the disadvantage of entangling the applications' executables, configuration, libraries, and lifecycles with each other and with the host OS. One could build immutable virtual-machine images in order to achieve predictable rollouts and rollbacks, but VMs are heavyweight and non-portable.

- The *New Way* is to deploy containers based on operating-system-level virtualization rather than hardware virtualization. These containers are isolated from each other and from the host: they have their own filesystems, they can't see each others' processes, and their computational resource usage can be bounded. They are easier to build than VMs, and because they are decoupled from the underlying infrastructure and from the host filesystem, they are portable across clouds and OS distributions.

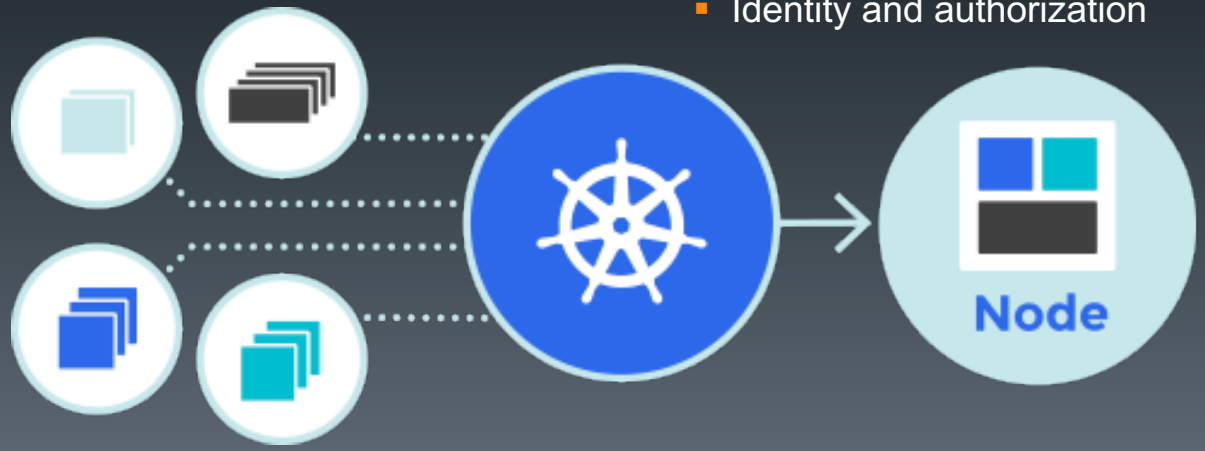
http://kubernetes.io/docs/whatisk8s/

kubernetes

K8s and Containers

- Containers
  - Benefits include:
    - Agile application creation and deployment
    - Continuous development, integration, and deployment
    - Dev and Ops separation of concerns
    - Environmental consistency across development, testing, and production
    - Cloud and OS distribution portability
    - Application-centric management
    - Loosely coupled, distributed, elastic, liberated micro-services
    - Resource isolation
    - Resource utilization

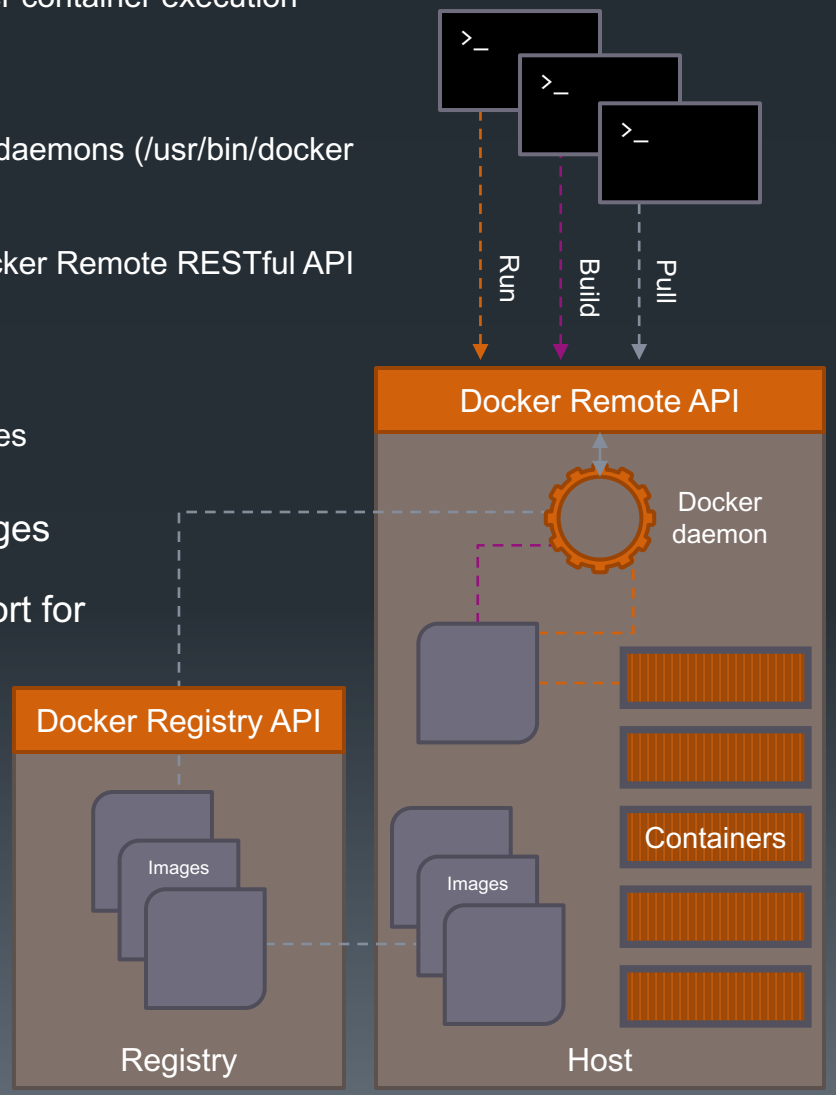
- K8s
  - Benefits include:
    - Co-locating helper processes - Pods
    - Mounting storage systems - Volumes
    - Distributing secrets - Secrets
    - Application health checking - liveness/readiness checks
    - Replicating application instances - Deployment, Replica Sets
    - Horizontal auto-scaling - Horizontal pod autoscaling
    - Naming and discovery - Service + DNS
    - Load balancing - Services, Ingress
    - Rolling updates - Deployments
    - Resource monitoring
    - Log access and ingestion
    - Lntrospection and debugging
    - Identity and authorization - Authorization plugins



K8s + containers = Isolation and orchestration

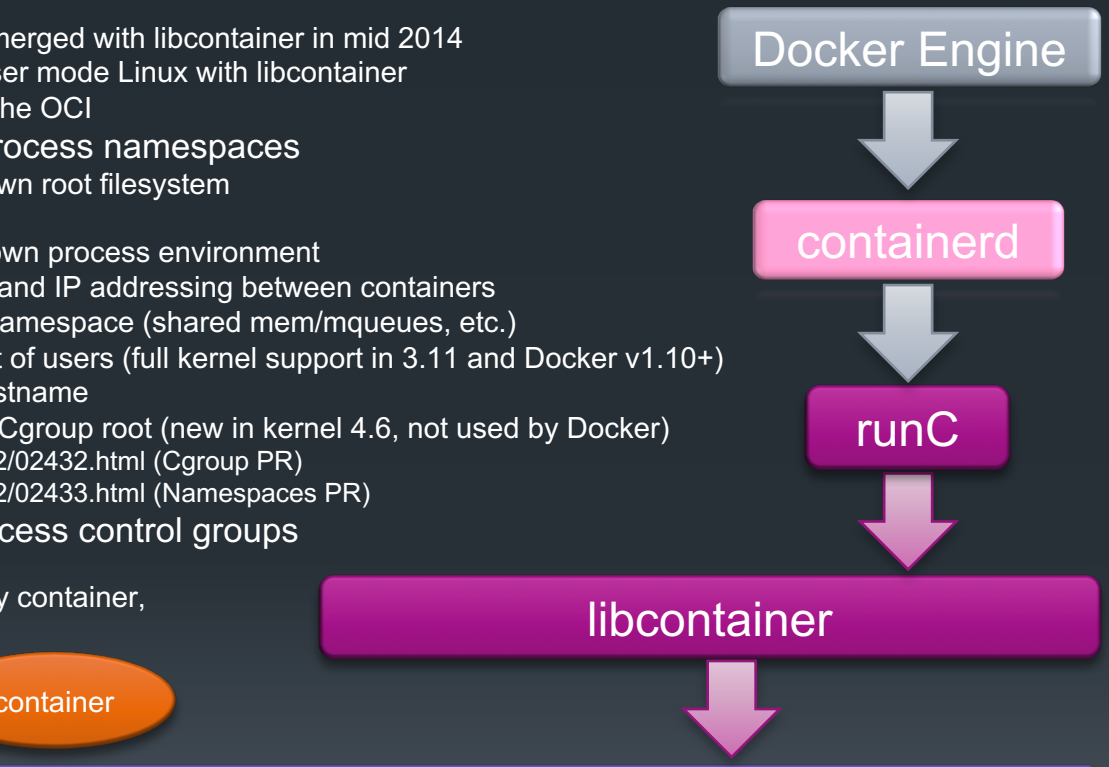
Core Docker Components

- Docker client and server executables
  - $ dockerd
    - Runs Docker as a daemon on the Linux host supporting docker container execution (/usr/bin/dockerd)
    - Docker manages containers, the linux kernel runs containers
  - $ docker
    - The Docker client sends requests to local and remote Docker daemons (/usr/bin/docker or docker.exe)
  - Docker Remote API
    - The Docker client talks to the Docker daemon through the Docker Remote RESTful API
- Docker Images
  - Images are used to generate containers
    - Just as a VM image can create multiple VM instances
    - Just as an executable (/usr/bin/vi) can create multiple processes
- Registries
  - Registries are network services from which Docker Images can be saved and retrieved
  - The Docker Hub is an internet based registry with support for public and private images
  - Private registry servers can be created for an organization's internal use
- Docker Containers
  - A container is a software package generated from an image
  - Said to be running when processes are executing within it
  - Can be stopped and started



Container Enablers

- Docker installations include the following elements:
  - **dockerd** [Docker daemon] – service performing high level container management
    - https://github.com/docker/docker
  - **containerd** [runc manager] – daemon directly managing OCI containers (used by Docker v1.11+)
    - https://github.com/docker/containerd [moving from Docker to OCI/CNCF?]
  - **runc** [launcher] – OCI container launcher (used by Docker v1.11+)
    - https://github.com/opencontainers/runc
  - **libcontainer** [system interface] – OCI kernel container interface (used by Docker v0.9+)
    - https://github.com/opencontainers/runc/tree/master/libcontainer
    - Linux lxc is no longer supported by Docker
    - The Parallels (now Odin) libct interface was merged with libcontainer in mid 2014
    - RedHat's project atomic has containerized user mode Linux with libcontainer
    - Now the standard container library/format of the OCI
  - Namespaces [Isolation] – Linux kernel process namespaces
    - Filesystem isolation: each container has its own root filesystem
      - Copy-on-write (COW) minimizing disk usage
    - Process isolation: each container runs in its own process environment
    - Network isolation: separate virtual interfaces and IP addressing between containers
    - IPC isolation: interprocess communications namespace (shared mem/mqueues, etc.)
    - User isolation: each container has its own set of users (full kernel support in 3.11 and Docker v1.10+)
    - UTS isolation: each container has its own hostname
    - Cgroup isolation: each container has its own Cgroup root (new in kernel 4.6, not used by Docker)
      - http://lkml.iu.edu/hypermail/linux/kernel/1603.2/02432.html (Cgroup PR)
      - http://lkml.iu.edu/hypermail/linux/kernel/1603.2/02433.html (Namespaces PR)
  - CGroups [Constraints] - Linux kernel process control groups
    - Resource constraints: resources like CPU, memory and I/O are constrained, container by container, using kernel control groups




```
user@ubuntu:~$ ps -f $(pgrep docker)
UID          PID    PPID  C STIME TTY      STAT   TIME CMD
root        1388       1  0 Feb28 ?        Ssl    5:06 /usr/bin/dockerd -H fd://
root        1497    1388  0 Feb28 ?        Ssl    2:46 unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout 2m --state-dir
/var/run/docker/libcontainerd/containerd --shim docker-containerd-shim --runtime docker-runc
```

```
user@ubuntu:~$ ls -l $(which docker-runc)
-rwxr-xr-x 1 root root 8199616 Apr  7 22:49 /usr/bin/docker-runc
```

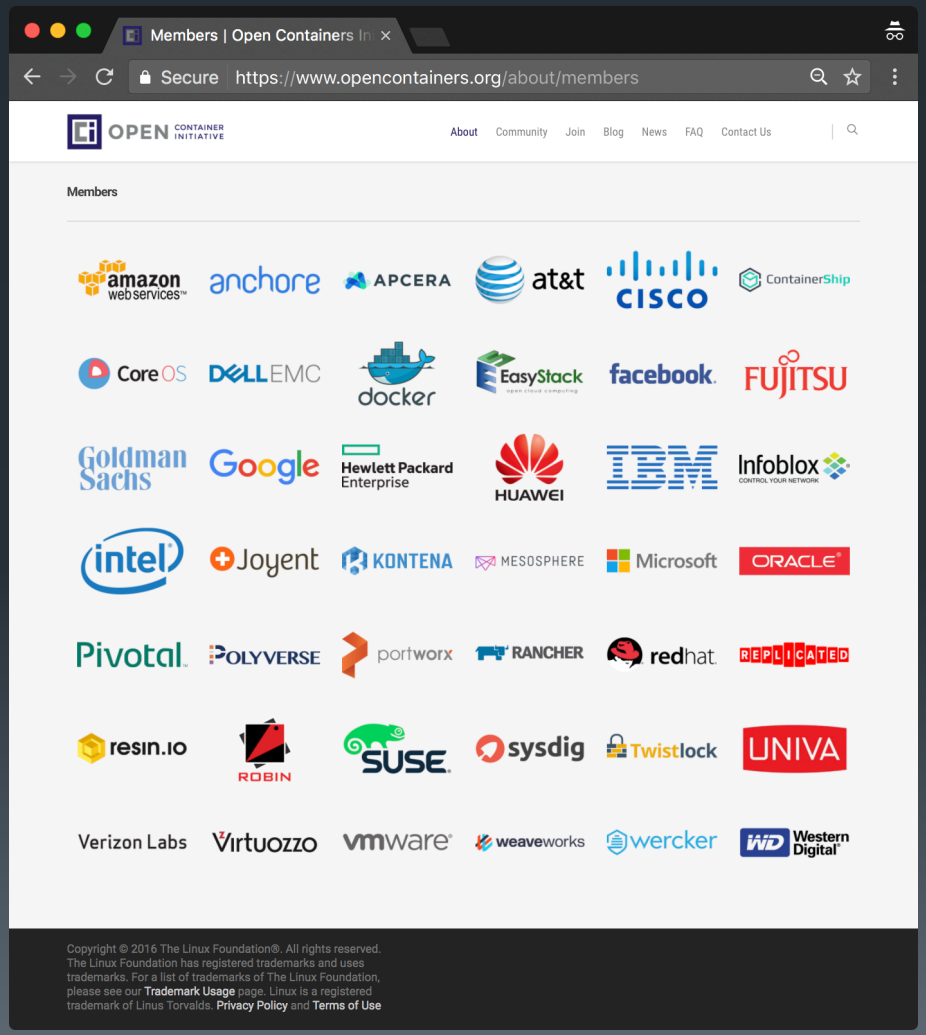
Container & Kubernetes Timeline

- 2004 Google begins work on what becomes Borg
- 2005 OpenVZ (Previously Parallels [SWsoft] Virtuozzo containers) open sourced [Linux 2.6.11]
- 2006 Google (Menage/Seth) add Process Containers to Linux (now called CGroups) [Linux 2.6.15]
- 2007 Google uses cgroups to containerize search [Linux 2.6.20]
- 2008 LXC version 0.1.0 released (basic functions) [Linux 2.6.24]
- 2009 First LTS Linux release with official LXC support [Linux 2.6.32 LTS]
- 2011 Container Unification agreement during kernel summit (In tree unified CGroups and Namespaces)
- 2012 dotCloud develops PaaS framework with a handy tool called Docker
- 2013 Linux supports LXC, Docker, OpenVZ [Linux 3.10 & 3.12 LTS] dotCloud becomes Docker Inc., Google CE adds Docker support
- 2013-05 – CoreOS founded – creaters of CoreOS Linux, the rkt container runtime, etcd distributed key-value store & flannel SDN
  - First release of flannel in October 2013
- 2014-04 – Docker creates Docker Governance Advisory Board (DGAB) and partners with Amazon, Ubuntu, RHEL
- 2014 05 – Linux Distros began enabling user namespaces [Linux 3.14 & 3.18 LTS]
- 2014-06 – Docker v1.0
- 2014-07 – Docker v1.1 released (Docker Engine + Docker Hub + APIs) and Docker acquires Orchard (Fig renamed Compose)
- 2014-08 – Docker v1.2 and Docker VMWare partnership announced
- 2014-09 – Google (Beda/Burns/McLuckie) open source Kubernetes
- 2014-10 – Docker v1.3 and Docker Microsoft partnership announced
- 2014-11 – Amazon announces Docker container support in EC2 Container Service [ECS]
- 2015-01 – etcd v2.0 released; first major stable release
- 2015-02 – Docker v1.5 and orchestration tools: Machine, Swarm and Compose
- 2015-04 – Docker v1.6 with Registry 2.0 and Windows Client Preview
- 2015-06 – Docker v1.7; Linux Foundation - Open Container Initiative [OCI] with libcontainer as base
- 2015-07 – K8s v1.0; Cloud Native Computing Foundation [CNCF] launches with Kubernetes as base
- 2015-08 – K8s v1.1 Deployments/ReplicaSets; Docker 1.8 released with support for pluggable volume backends
- 2015-10 – Docker Inc. acquires DevOps/Container orchestration and management platform TuTum
- 2015-11 – Docker v1.9 released with support for multi-host networking and named volumes
- 2016-01 – Docker acquires Unikernel Systems (folks from the Xen Project)
- 2016-02 – Docker v1.10 released with support for user namespaces, SecComp and DNS container name resolution
  - rkt v1.0 released - first release of rkt recommended for use in production
- 2016-03 – K8s v1.2 HA; Docker Cloud released with automated deployment to AWS, DigitalOcean, SoftLayer and Azure
- 2016-04 – Docker v1.11 released with OCI support (built on containerd and runC)
- 2016-07 – K8s v1.3 RBAC & Federation
- 2016-08 – Docker v1.12 released with integrated cluster management (Swarm Mode)
- 2016-09 – K8s v1.4 kubeadm
- 2016-12 – K8s v1.5 Stateful Sets
- 2017-01 – Docker v1.13 released docker plugin, docker secret, docker stacks with support for Compose on Swarm Mode
- 2017-02 – Docker v17.03.0-ce changed to monthly release cycle (YY.MM) with two release channels (monthly and quarterly)
- 2017-03 – K8s v1.6 improvements to Node affinity, CRI and federation
- 2017-04 – Containerd removed from Docker and contributed to CNCF along with Rocket
- 2017-06 – K8s v1.7 adds PID namespace to PODs, encrypted secrets (alpha), network policy for pod-to-pod communication (networking.k8s.io/v1)
  - Docker v17.06-ce introduces swarm-mode services with node-local networks

Container Standardization

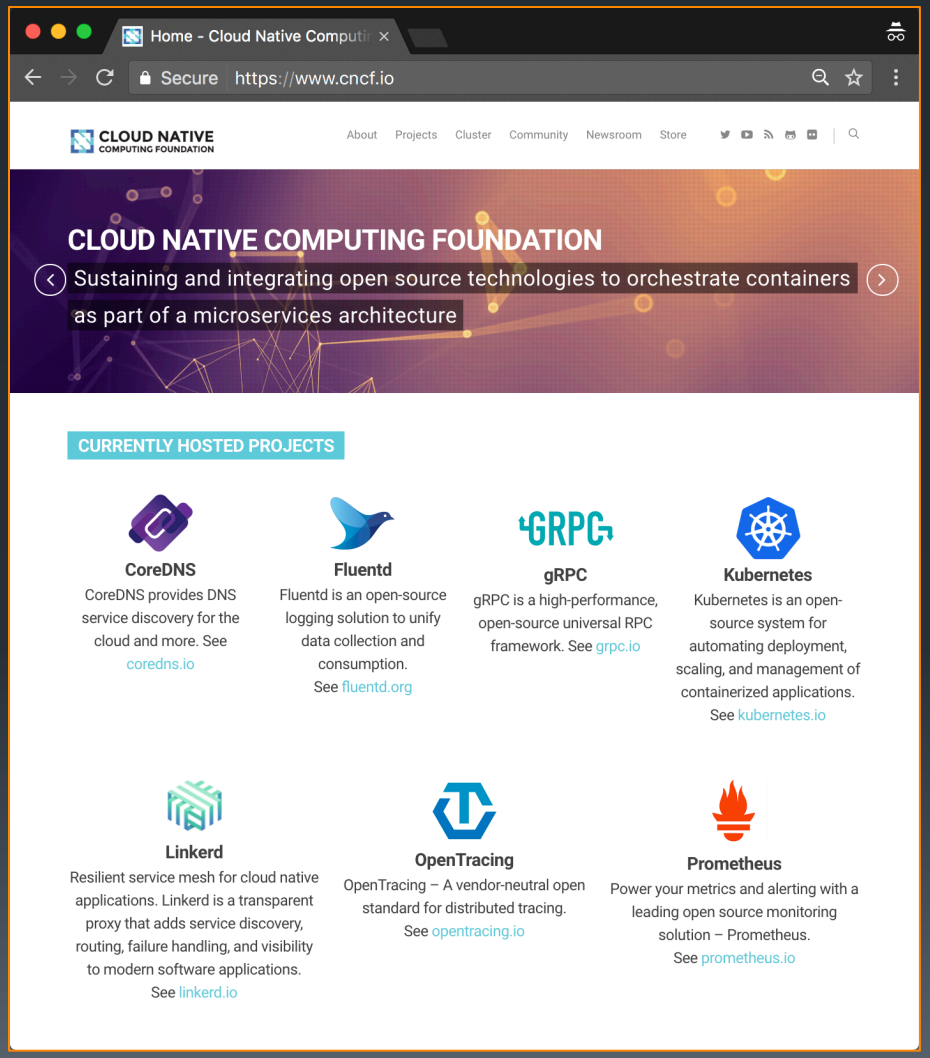
- OCI [circa 6/2015]
  - Open Container Initiative
    - Formerly Open Container Project
  - OCI seeks to create open industry standards around container formats and runtime
    - A standard that ensures any container can run on any machines or cloud computing service
  - Run by the Linux Foundation
    - Non-profit which oversees the Linux open source operating system
  - Docker donated the container format, runtime code (libcontainer) and specifications
  - Members (All of the tech companies in the Fortune 100 except Apple, all of the key container startups):
    - Amazon, Anchore, Apcera, AT&T, Cisco, ContainerShip, CoreOS, DellEMC, Docker, EasyStack, Facebook, Fujitsu Limited, Goldman Sachs, Google, Hewlett Packard Enterprise, Huawei, IBM, Infoblox, Intel, Joyent, Kontena, Mesosphere, Microsoft, Midokura, Nutanix, Oracle, Pivotal, Polyverse, Portworx, Rancher Labs, Red Hat, Replicated, Resin.io, Robin Systems, SUSE, Sysdig, Twistlock, Twitter, Univa, Verizon Labs, Virtuozzo, VMware, Weaveworks, Wercker and WD

The Open Container Initiative (OCI) is a lightweight, open governance structure (project), formed under the auspices of the Linux Foundation, for the express purpose of creating open industry standards around container formats and runtime. The OCI was launched on June 22nd 2015 by Docker, CoreOS and other leaders in the container industry

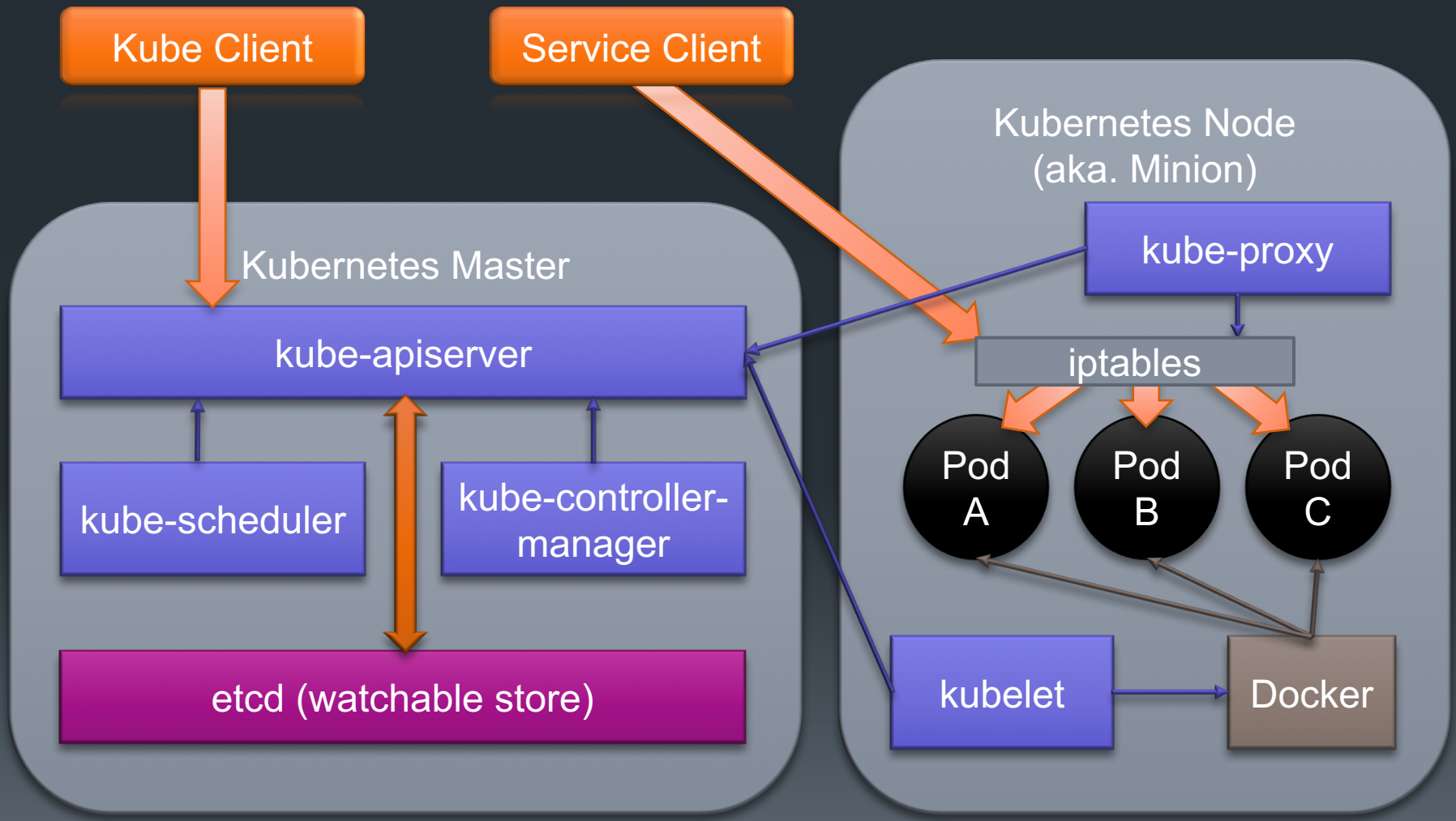


Container Orchestration Initiatives

- CNCF [circa 7/2015]
  - Cloud Native Computing Foundation
    - Hosted by the Linux Foundation
  - Cloud Native Applications are: container packaged, dynamically managed and micro-services oriented
  - Mission: To create and drive the adoption of a new set of common container technologies informed by technical merit and end user value, and inspired by Internet-scale computing
  - Aims to host the reference stack of technologies around container orchestration
    - Google contributed Kubernetes to the kick off the foundation
    - Prometheus, Fluentd, CoreDNS, Linkerd, OpenTracing, gRPC, Containerd, Rocket and CNI projects have also been moved under the foundation
  - Members include:
    - Cisco, CoreOS, Docker, Fujitsu, Google, Huawei, IBM, Intel, Joyent, Mesosphere, Red Hat, Samsung SDS, Supernap, AT&T, NetApp Amihan, Apcera, Apigee Aporeto, Apprenda, AVI Networks, Caicloud, Canonical, Capital One, Centrify, ChaoSuan, Chef, Cloudsoft, Container Solutions, Crunchy, Dao Cloud, Deis, Digital Ocean, Easy Stack, eBay, Eldarion, Exoscale, Galactic Fog, Goldman Sachs, Gronau, Heptio, Iguaz.io, Infoblox, Juniper, Livewyer, Loodse, Minio, Mirantis, NCSoft, NEC, Nginx, Packet, Plexistor, Portworx, Rancher, RX-M, Stack Point Cloud, Storage OS, Sysdig, Tigera, Treasure Data, Twistlock, Twitter, Univa, Virtuozzo, VMware, Weaveworks, Box, Ticketmaster, Zhejiang University

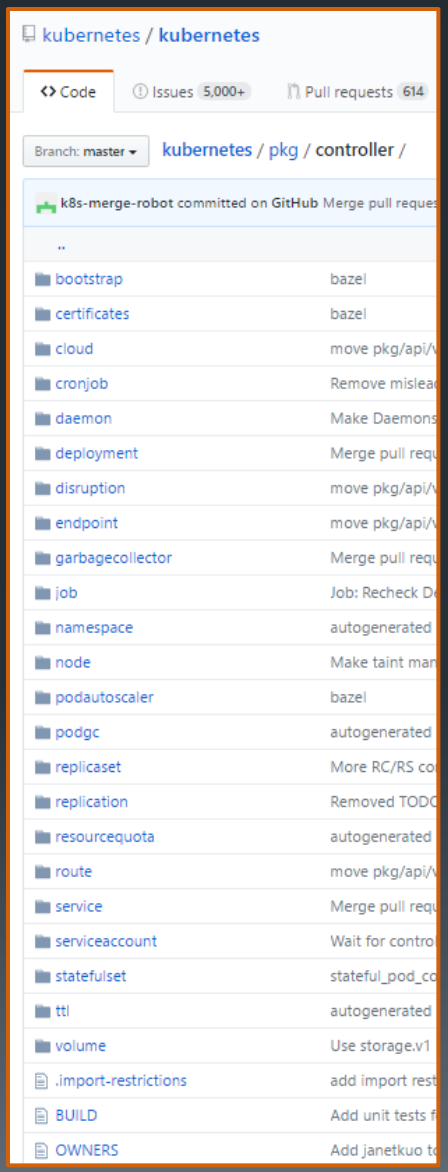


Kubernetes Architecture



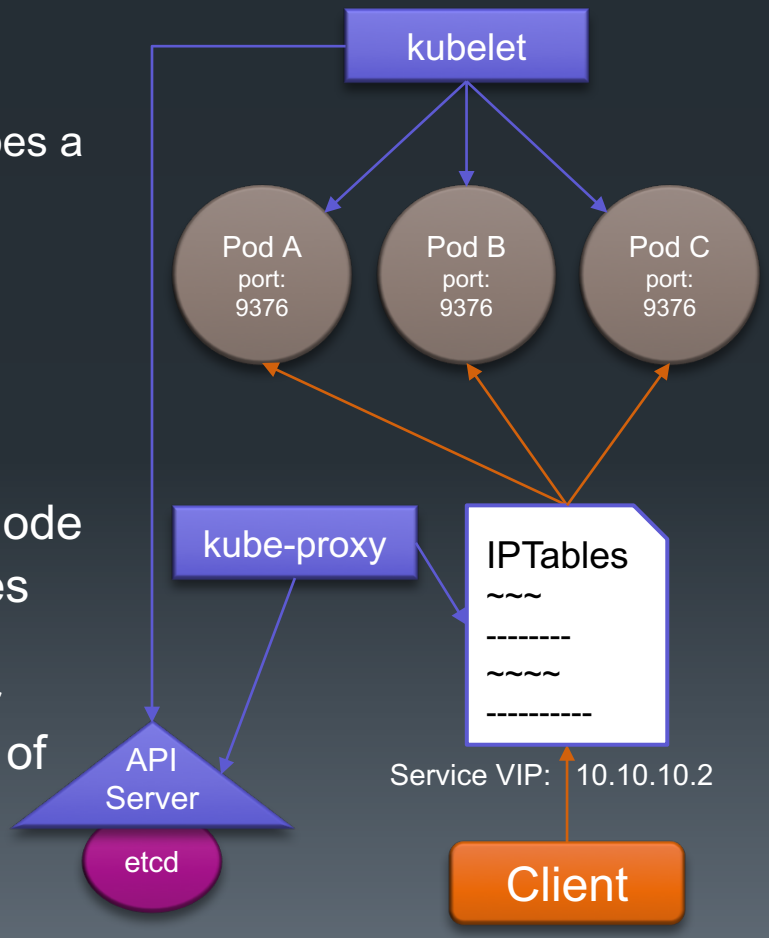
Master Components

- The Kubernetes Control Plane
  - The Kubernetes control plane is split into a set of microservices
  - Often all of these services run on a "Kubernetes Master" node
  - These components work together to provide a unified view of the cluster
- etcd
  - All persistent master state is stored in an etcd cluster
  - Watch support allows coordinating components to be notified of changes
  - etcd 3 moved from HTTP/JSON to gRPC/ProtoBuf
    - Improving performance and scalability
    - gRPC gateway supports etcd2 JSON endpoints for backward compatibility
  - etcd 3 data model offers a flat binary key space in lieu of key hierarchies
- API Server
  - The apiserver serves the Kubernetes API
  - A CRUD-y server, most/all business logic implemented in separate components or plug-ins
  - Processes REST operations, validates them, updates etcd
- Scheduler
  - Binds unscheduled pods to nodes via the /binding API
  - The scheduler is pluggable
- Controller Manager
  - Other cluster-level functions are performed by the Controller Manager
    - Endpoint manages and syncs service endpoints
    - Nodes are discovered, managed, and monitored by the node controller
    - ReplicaSet controller creates and maintains replicasets
    - PodAutoScaler controller scales pods in and out
    - Etc.
  - May eventually be split into separate components



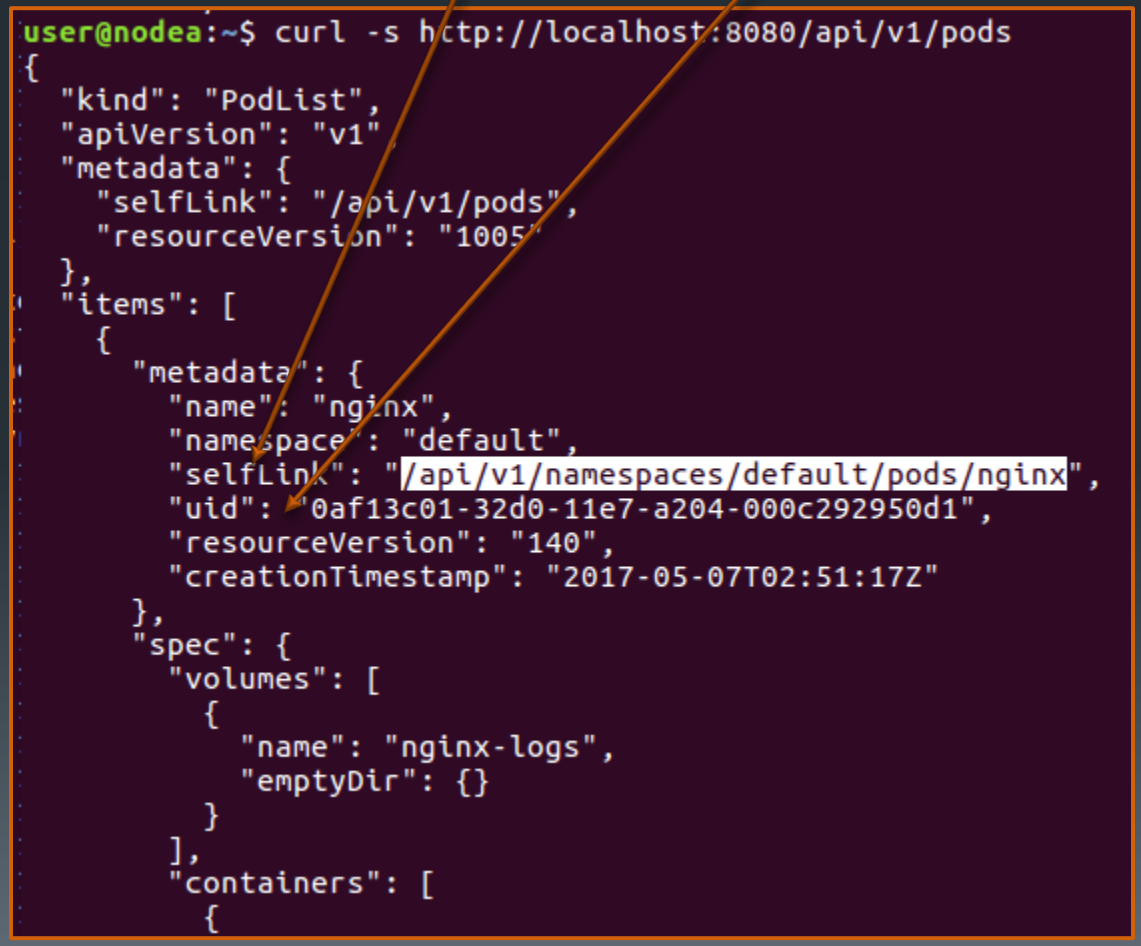
Node Components

- kubelet
  - The kubelet is the primary “node agent” that runs on each node
  - The kubelet manages pods and their containers, their images, their volumes, etc.
  - The kubelet works in terms of a PodSpec
    - A PodSpec is a YAML or JSON object that describes a pod
    - kubelet PodSpecs are provided through various mechanisms (primarily through the apiserver)
    - Kublet ensures that the containers described in PodSpecs are running and healthy
- kube-proxy
  - The Kubernetes network proxy runs on each node
  - Manages the iptables service mesh for services defined in the Kubernetes API
  - Can do simple TCP/UDP stream forwarding or round robin TCP/UDP forwarding across a set of backends in proxy mode



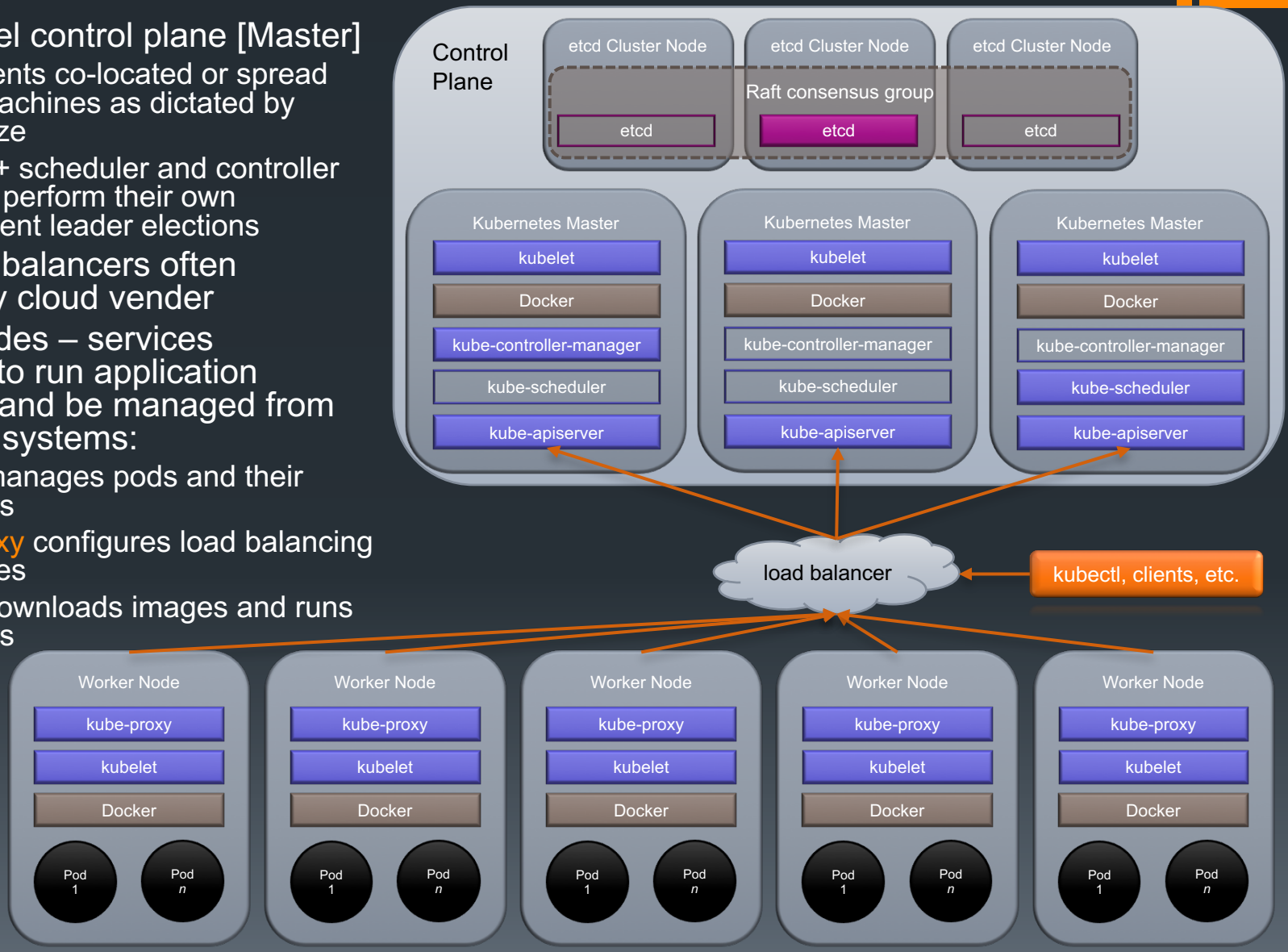
Object Names

- Every resource in Kubernetes is identified by a URI and has a UID
- The URI includes:
  - API version
  - Namespace
  - Object Type
  - Object name
- 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



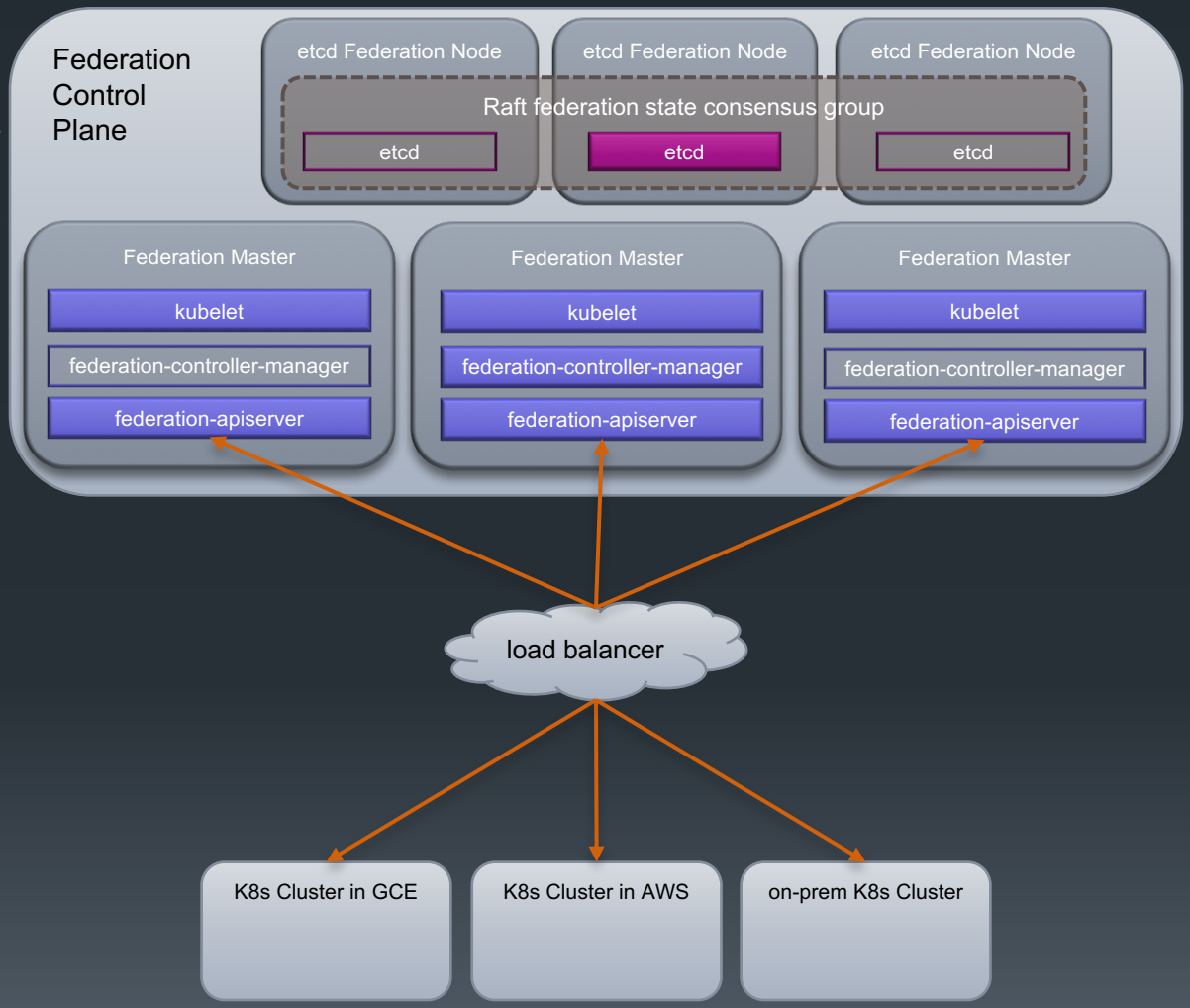
Scaled Architecture & HA Model

- 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



Federated Architecture

- Federation makes it possible to manage multiple K8s clusters
  - Hybrid cloud – have multiple clusters with different cloud providers and on-prem
    - Prevents cloud/cluster provider lock-in
- Resources in multiple clusters are kept in sync
  - Same Deployment resource can exist in multiple clusters
- Cross-cluster service discovery
  - Auto-configure DNS servers & load balancers with backends from all clusters
- Mitigates impact of cluster failure (HA)
- Limitations
  - Increased network bandwidth & cost
    - May be significant if clusters are running in different regions and with different providers
  - Reduced cross-cluster isolation
    - A bug in the federation control plane can impact all clusters
  - Maturity (federation project is relatively new)
    - Many resources are alpha or unavailable



etcd

- OpenSource distributed consistent key-value store for shared configuration and service discovery
  - Created by CoreOS
  - Originally a distributed configuration store 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 config info about a server) and "d" for distributed
- etcd focuses on :
  - Secure: automatic TLS w/ optional client cert auth
  - Fast: benchmarked 1000s of writes/s per instance
  - Resilient: Raft consensus based highly consistent storage model
- Written in Go
- etcdctl is a simple command line client that can be used in scripts or to explore an etcd cluster
  - Must “export ETCDCTL_API=3” to use with v3
- 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 multi-host networking
- Used to track pod deployment, execution status, labels, endpoints, … everything in Kubernetes
- Kubernetes support
  - K8s 1.0 only supports etcd2
  - K8s 1.3 added support for etcd3 with a switch
    - kube-apiserver --storage-backend=etcd3
  - K8s 1.6 defaults to etcd 3 but still supports 2
    - kube-apiserver --storage-backend=etcd2

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

CSV XML HTML Plain text

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

```
l80c60
ebb4a725081d
/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
```

kubeadm (beta)

- Works with VMs, physical servers and cloud servers
- Designed to be part of a large provisioning system
  - Also for easy manual provisioning
- `kubeadm init` bootstraps a K8s cluster:
  - Runs a series of pre-flight checks to validate the system state prior to making changes
    - Generates warnings or errors that cause kubeadm to exit (can be bypassed with `--skip-preflight-checks`)
  - Generates a token that additional nodes can use to register themselves with the master
    - User supplied token supported
  - Generates a self-signed CA using openssl to provision identities for each node in the cluster
    - Also used by the API server to secure communication with clients
  - Outputs a kubeconfig file for the kubelet to use to connect to the API server
    - Creates an additional kubeconfig file for administration
  - Generates Kubernetes resource manifests in `/etc/kubernetes/manifests` for the API server, controller manager and scheduler
  - Installs any add-on components, such as DNS or discovery, via the API server
- Configuring kubeadm with a configuration file instead of command line flags is supported
  - Some advanced features only available as configuration file options

`kubeadm join`

- Uses the token to talk to the API server and securely get the root CA certificate
- Creates a local key pair; prepares a certificate signing request (CSR) and sends that off to the API server for signing
- Configures the local kubelet to connect to the API server

```
$ kubeadm init --help
Run this in order to set up the Kubernetes master

Usage:
  kubeadm init [flags]

Flags:
      --api-advertise-addresses stringSlice   The IP addresses to advertise, in case autodetection fails
      --api-external-dns-names stringSlice    The DNS names to advertise, in case you have configured them yourself
      --api-port int32                        Port for API to bind to (default 6443)
      --cloud-provider cloudprovider          Enable cloud provider features (external load-balancers, storage, etc). Note that you have to configure all kubelets manually
      --config string                         Path to kubeadm config file
      --discovery-port int32                  Port for JWS discovery service to bind to (default 9898)
      --pod-network-cidr string               Specify range of IP addresses for the pod network; if set, the control plane will automatically allocate CIDRs for every node
      --service-cidr string                   Use alternative range of IP address for service VIPs (default "10.96.0.0/12")
      --service-dns-domain string             Use alternative domain for services, e.g. "myorg.internal" (default "cluster.local")
      --skip-preflight-checks                 skip preflight checks normally run before modifying the system
      --token string                          Shared secret used to secure cluster bootstrap; if none is provided, one will be generated for you
      --use-kubernetes-version string         Choose a specific Kubernetes version for the control plane (default "stable")
```

Kubefed (beta in K8s 1.6)

- Kubernetes 1.5 added the kubefed command line tool to help administrate federated clusters
- kubefed lets you
  - Deploy a new Kubernetes cluster federation control plane
  - Add clusters to or remove clusters from an existing federation control plane
- To install:
  - `curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/kubernetes-client-linux-amd64.tar.gz`
  - `tar -xzvf kubernetes-client-linux-amd64.tar.gz`
  - `sudo cp kubernetes/client/bin/kubefed /usr/local/bin`
  - `sudo chmod +x /usr/local/bin/kubefed`
- To configure the federation “fellowship” on the host cluster named "rivendell":

```
$ kubefed init fellowship \
     --host-cluster-context=rivendell \
     --dns-provider="google-clouddns" \
     --dns-zone-name="example.com."
```

Kubernetes Startup script Providers

IaaS Provider	Config. Mgmt	OS	Networking	Docs	Conforms	Support Level
GKE			GCE	docs	'œ"	Commercial
Stackpoint.io		multi-support	multi-support	docs		Commercial
GCE	Saltstack	Debian	GCE	docs	'œ"	Project
Azure	CoreOS	CoreOS	Weave	docs		Community (@errordeveloper, @squillace, @chanezon, @crossorigin)
Azure	CoreOS	CoreOS	flannel	docs		Community (@colemickens)
Docker Single Node	custom	N/A	local	docs		Project (@brendandburns)
Docker Multi Node	custom	N/A	flannel	docs		Project (@brendandburns)
Bare-metal	Ansible	Fedora	flannel	docs		Project
Bare-metal	custom	Fedora	*none*	docs		Project
Bare-metal	custom	Fedora	flannel	docs		Community (@aveshagarwal)
libvirt	custom	Fedora	flannel	docs		Community (@aveshagarwal)
KVM	custom	Fedora	flannel	docs		Community (@aveshagarwal)
Mesos/Docker	custom	Ubuntu	Docker			
Mesos/GCE						
DCOS	Marathon	CoreOS/Alpine	custom			
AWS	CoreOS	CoreOS	flannel			
GCE	CoreOS	CoreOS	flannel			
Vagrant	CoreOS	CoreOS	flannel			
Bare-metal (Offline)	CoreOS	CoreOS	flannel			
Bare-metal	CoreOS	CoreOS	Calico			
CloudStack	Ansible	CoreOS	flannel			
Vmware		Debian	OVS			
Bare-metal	custom	CentOS	*none*	docs		Community (@coolsvap)
AWS	Juju	Ubuntu	flannel	docs		Community (@whit, @matt, @chuck)
OpenStack/HPCloud	Juju	Ubuntu	flannel	docs		Community (@whit, @matt, @chuck)
Joyent	Juju	Ubuntu	flannel	docs		Community (@whit, @matt, @chuck)
AWS	Saltstack	Ubuntu	OVS	docs		Community (@justinsb)
Bare-metal	custom	Ubuntu	Calico	docs		Community (@djosborne)
Bare-metal	custom	Ubuntu	flannel	docs		Community (@resouer, @WIZARD-CXY)
libvirt/KVM	CoreOS	CoreOS	libvirt/KVM	docs		Community (@lhuard1A)
oVirt				docs		Community (@simon3z)
OpenStack Heat	Saltstack	CentOS	Neutron + flannel hostgw	docs		Community (@FujitsuEnablingSoftwareTechnologyGmbH)
Rackspace	CoreOS	CoreOS	flannel	docs		Community (@doublerr)
any	any	any	any	docs		Community (@erictune)

- Kubernetes source includes an installation script to stand up a small cluster in a number of provider environments
  - e.g. To start an AWS cluster:
    - `$ export KUBERNETES_PROVIDER=aws`
    - `$ kube-up.sh  # starts up a small generic cluster`
      - Requires a preconfigured AWS client
      - Copies files to S3 and starts a VPC with 4 nodes for the cluster
    - `$ kube-down.sh  # to take the cluster down`

Kubernetes Operations (kops)

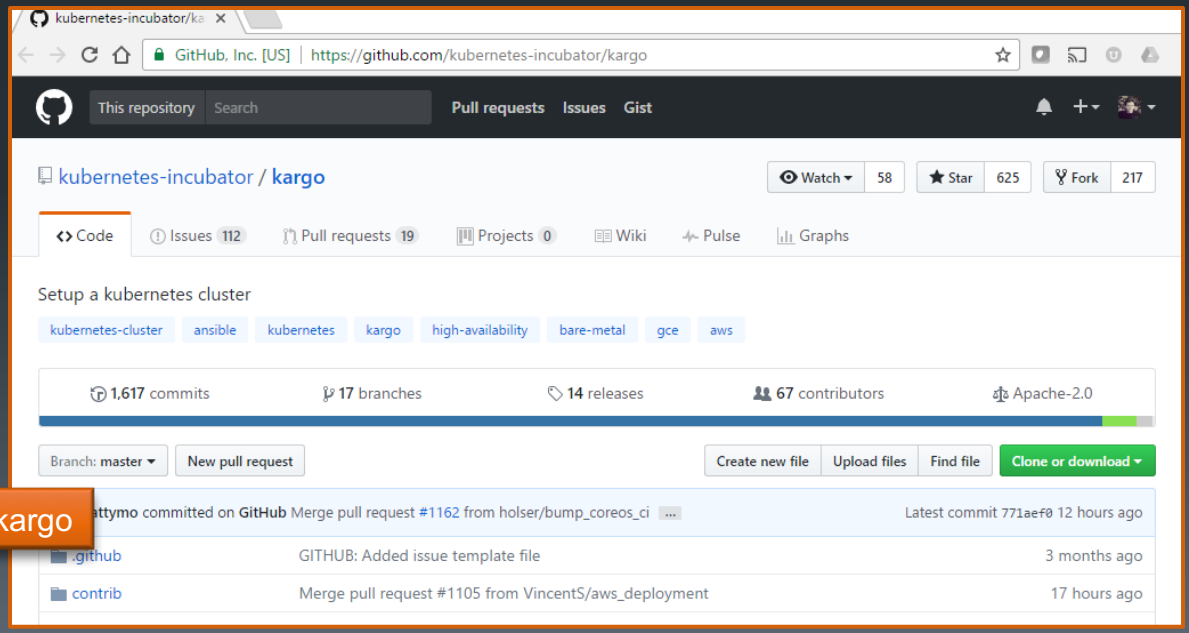
- Create & maintain production-grade K8s clusters from the command line
- Currently only AWS supported
  - GCE & VMware vSphere in alpha
  - Other platforms in roadmap
- Automates provisioning
  - HA K8s Masters & Nodes in multiple Availability Zones & auto-scaling groups
  - Idempotent (remove `--yes` for dry runs of updates/deletes)
  - Can generate AWS CloudFormation & Terraform configurations
  - Supports add-ons

```
# Create an ssh key, used by kops for connecting to AWS instances
$ ssh-keygen -t rsa

# Install AWS CLI tools (req. by kops) & create config file using the kops AWS
# user ID & secret created in the AWS web portal
$ sudo pip install --upgrade --user awscli
$ mkdir ~/.aws && vim ~/.aws/credentials
[default]
aws_access_key_id = <ACCESS_KEY_ID>
aws_secret_access_key = <SECRET_ACCESS_KEY>

# Install kops tool
$ wget https://github.com/kubernetes/kops/releases/download/1.7.0/kops-linux-amd64
$ sudo chmod +x kops-linux-amd64
$ sudo mv kops-linux-amd64 /usr/local/bin/kops

# Install kubectl
$ curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
      https://storage.googleapis.com/kubernetes-
      release/release/stable.txt)/bin/linux/amd64/kubectl
$ sudo chmod +x ./kubectl
$ sudo mv ./kubectl /usr/local/bin/kubectl

# Create a bucket in S3 to store kops configs
$ aws s3api create-bucket --bucket=kops-state-store --region=us-east-1

# Create cluster config
$ kops create cluster \
    --cloud=aws \
    --master-count=3 \
    --master-zones=us-west-1a,us-west-1c \
    --master-size=t2.xlarge \
    --node-count=5 \
    --zones=us-west-1a,us-west-1c \
    --node-size=t2.xlarge \
    --associate-public-ip=false \
    --api-loadbalancer-type=public \
    --dns-zone=<HOSTED_ZONE_ID> \
    --authorization=RBAC \
    --channel=stable \
    --networking=kubenet \
    --state=s3://kops-state-store \
    --name=mycluster

# View/edit the yaml config generated by the create command
$ kops edit cluster mycluster --state=s3://kops-state-store

# Launch/update the cluster
$ kops update cluster mycluster --state=s3://kops-state-store --yes
# This action generates a config for kubectl in .kube/config

# Delete the cluster
$ kops delete cluster mycluster --state=s3://kops-state-store --yes
```

Kargo

- Ansible solution for installing K8s
- Supported Linux distributions
  - Container Linux by CoreOS
  - Debian Jessie
  - Ubuntu 16.04
  - CentOS/RHEL 7
- Note: Upstart/SysV init based OS types are not supported.
- Versions of supported components
  - kubernetes v1.6.4
  - etcd v3.0.17
  - flanneld v0.6.2
  - calicoctl v0.23.0
  - canal (calico/flannel versions)
  - weave v1.8.2
  - docker v1.13.1
  - rkt v1.21.0

https://github.com/kubernetes-incubator/kargo



Getting Involved

- Twitter
  - https://twitter.com/kubernetesio
- Github
  - https://github.com/kubernetes/kubernetes
- Slack
  - http://slack.k8s.io/
- StackOverflow
  - http://stackoverflow.com/questions/tagged/kubernetes
- Google Group
  - https://groups.google.com/forum/#!forum/google-containers
- Calendar
  - https://calendar.google.com/calendar/embed?src=nt2tcnbtbied3l6gi2h29slvc0%40group.calendar.google.com

Summary

- Kubernetes is a fast evolving platform.
- The primary services of a K8s cluster master node include:
  - etcd
  - kube-apiserver
  - kube-controller-manager
  - kube-scheduler
- The primary services of a K8s node include:
  - kublet
  - kube-proxy
- Primary end user tool:
  - kubectl

Lab 1

- Install a basic cluster by hand

2. Kubelet & Advanced Pod Configuration

Objectives

- Kubelet
  - Understand the role of Kubelet process (primary node agent)
  - See how it works in terms of a PodSpec
  - Understand how PodSpecs are supplied to a Kubelet
  - Ensures that the containers described in those PodSpecs are running and healthy
- Pod Configuration (aka PodSpec)
  - Review key components of a PodSpec
  - Understand interaction between single-container and multi-container
- Understand imperative vs declarative controls

Kubelet internals

- The primary "node agent" that runs on each node
- Manages PodSpecs
- Ensures that the containers in its assigned PodSpecs are running and healthy
- Ways a PodSpec manifest can be provided to the Kubelet:
  - apiServer: PodSpecs retrieved from the kube-apiserver
  - File: files found on a passed on the command line
  - HTTP endpoint
  - HTTP server: HTTP API hosted by the kubelet to which PodSpecs can be posted
- The kubelet synchronizes the desired state (defined in the PodSpecs) with the actual state (that reported by the container engine, e.g. Docker) on a regular basis
  - `--sync-frequency` sets the period between synchronizing running containers and PodSpecs
    - Kubernetes 1.0 – 1.3 default to 10 seconds
      - Net Pod startup times in typical Kubernetes installations are about 5 seconds (10/2)
      - Net container failure detection is <= 10 seconds
    - Changing the frequency has a direct impact on performance
      - Checking 100 podSpecs / 10 seconds requires between 0.5-2 vcpus in Kubernetes v1.1
      - Greatly improved in v1.2 cut v1.1 overhead in half and v1.3 cut it in half again
    - Kubernetes 1.5+ defaults 60 seconds however, Docker events are used to sync most activities on an event basis now (no delay)
  - `--max-pods` sets the maximum number of Pods that can run on a kubelet
    - Defaults to 40 in v1.1, 110 in v1.4

The kubelet caches state on disk, to eliminate the backing store:
`sudo rm -rf /var/lib/kubelet`

```
user@nodea:~$ sudo netstat -natp | grep kubelet
[sudo] password for user:
tcp        0      0 127.0.0.1:10248         0.0.0.0:*               LISTEN      17099/kubelet
tcp        0      0 10.2.0.5:45490          151.101.40.133:443      ESTABLISHED 17099/kubelet
tcp6       0      0 :::4194                 :::*                    LISTEN      17099/kubelet
tcp6       0      0 :::10250                :::*                    LISTEN      17099/kubelet
tcp6       0      0 :::10255                :::*                    LISTEN      17099/kubelet
```

Kubelet Ports
10248 – healthz
4194 – cadvisor
10250 – kubelet
10255 – kubelet (ro)

A Few Kubelet Options

- Over 80+ options
- `kubelet --help |& more`

https://kubernetes.io/docs/admin/kubelet/

- `--address=0.0.0.0`
  - The IP address for the Kubelet to serve on (set to 0.0.0.0 for all interfaces)
- `--allow-privileged[=false]`
  - If true, allow containers to request privileged mode [default=false]
- `--api-servers=[]`
  - List of Kubernetes API servers for publishing events, and reading pods and services (ip:port), comma separated
- `--cert-dir="/var/run/kubernetes"`
  - The directory where the TLS certs are located (by default /var/run/kubernetes). If `--tls-cert-file` and `--tls-private-key-file` are provided, this flag will be ignored
- `--cluster-dns=""`
  - IP address for a cluster DNS server; if set, kubelet will configure all containers to use this for DNS resolution in addition to the host's DNS servers
- `--cluster-domain=""`
  - Domain for this cluster; if set, kubelet will configure all containers to search this domain in addition to the host's search domains
- `--kubeconfig="/var/lib/kubelet/kubeconfig"`
  - Path to a kubeconfig file, specifying how to authenticate to API server (the master location is set by the api-servers flag)
  - `--require-kubeconfig` If true the Kubelet will exit if there are configuration errors, and will ignore the value of `--api-servers` in favor of the server defined in the kubeconfig file
- `--master-service-namespace="default"`
  - The namespace from which the kubernetes master services should be injected into pods

Provide PodSpec to Kubelet

- Other than a PodSpec from the apiserver, there are three ways that a container manifest can be provided to the Kubelet:
  - File: files found on the `--pod-manifest-path` passed on the command line
    - Rechecked every 20 seconds (configurable with `--file-check-frequency`)
    - Example: `sudo ~user/k8s/_output/local/bin/linux/amd64/kubelet --pod-manifest-path=./kubelet/pod.yaml`
  - HTTP endpoint: HTTP endpoint passed as a parameter on the command line: `--manifest-url`
    - Rechecked every 20 seconds (configurable with `--http-check-frequency`)
    - Example: `sudo ~user/k8s/_output/local/bin/linux/amd64/kubelet --manifest-url=https://raw.githubusercontent.com/kubernetes/kubernetes/master/examples/pod`
  - HTTP server: The kubelet can also listen for HTTP and respond to a simple API to which PodSpecs can be posted (`--enable-server` defaults to true) to submit a new manifest (underspec'd currently)
    - Ex. `curl --insecure https://localhost:10250/pods`

```
user@nodea:~$ sudo ~user/k8s/_output/bin/kubelet --manifest-url=https://raw.githubusercontent.com/kubernetes/kubernetes/master/examples/pod
I0912 20:14:13.405789  100524 feature_gate.go:144] feature gates: map[]
W0912 20:14:13.406216  100524 server.go:496] No API client: no api servers specifiedI0912 20:14:13.406338  100524 client.go:72] Connecting to docker on
unix:///var/run/docker.sock
I0912 20:14:13.406387  100524 client.go:92] Start docker client with request timeout=2m0s
W0912 20:14:13.408627  100524 cni.go:189] Unable to update cni config: No networks found in /etc/cni/net.d
I0912 20:14:13.416342  100524 manager.go:143] cAdvisor running in container: "/user.slice"
W0912 20:14:13.498900  100524 manager.go:151] unable to connect to Rkt api service: rkt: cannot tcp Dial rkt api service: dial tcp [::1]:15441: getsockopt: connection refused
I0912 20:14:13.532217  100524 fs.go:117] Filesystem partitions: map[/dev/sda1:{mountpoint:/var/lib/docker/aufs major:8 minor:1 fsType:ext4 blockSize:0}]
...
```

PodSpec

- PodSpec is a description of a pod
- A PodSpec is used by top level API object v1.Pod

- Pod is a collection of containers that can run on a host
  - This resource is created by clients and scheduled onto hosts
- A *pod* (as in a pod of whales or pea pod) is a group of one or more containers (such as Docker containers), the shared storage for those containers, and options about how to run the containers
- A pod is:
  - Co-located
  - Co-scheduled
  - Run in a shared context
  - Models an application-specific “logical host” - it contains one or more application containers which are relatively tightly coupled
    - In a pre-container world, they would have executed on the same physical or virtual machine
- The shared context of a pod is a set of Linux namespaces, cgroups, and potentially other facets.

- Containers within a pod share:
  - An IP address
  - Port space,
  - Find each other via localhost
  - Communicate with each other using ipc
    - Containers in different pods have distinct IP addresses and can not communicate by IPC
- Applications within a pod also have access to shared volumes, which are defined as part of a pod and are made available to be mounted into each application’s filesystem

- http://kubernetes.io/docs/api-reference/v1/definitions/#_v1_podspec

v1.Pod

v1.PodSpec

```
# Copy of pod.yaml without file
# extension for test

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

https://raw.githubusercontent.com/kubernetes/kubernetes/master/examples/pod

v1.PodSpec Definition

- PodSpec is a description of a pod
- http://kubernetes.io/docs/api-reference/v1/definitions/#_v1_podspec

- **v1.PodSpec** – (schema)
  - activeDeadlineSeconds – integer
    - Duration in seconds the pod may be active on the node relative to StartTime before the system will actively try to mark it failed and kill associated containers
  - affinity – v1.Affinity array
    - If specified, the pod's scheduling constraints
  - automountServiceAccountToken – boolean
    - Indicates whether a service account token should be automatically mounted
  - containers – v1.Container array
    - List of containers belonging to the pod
      - Containers cannot currently be added or removed
      - Must be at least one container in a Pod
      - Cannot be updated
  - dnsPolicy – string
    - Set DNS policy for containers within the pod
      - *ClusterFirst (default) or Default*
  - hostAliases – v1.HostAlias array
    - Optional list of hosts and IPs that will be injected into the pod's hosts file
      - Only valid for non-hostNetwork pods
  - hostIPC – boolean
    - Use the host's ipc namespace
  - hostNetwork – boolean
    - Use the host's network namespace
      - If this option is set, the ports that will be used must be specified
  - hostPID – boolean
    - Use the host's pid namespace
  - initContainers – v1.Container array
    - List of initialization containers belonging to the pod

- **v1.PodSpec** (continued)
  - imagePullSecrets – v1.LocalObjectReference array
    - List of references to secrets in the same namespace to use for pulling any of the images used by this PodSpec
      - If specified, these secrets will be passed to individual puller implementations for them to use
  - nodeSelector – any
    - NodeSelector is a selector which must be true for the pod to fit on a node
      - Selector which must match a node's labels for the pod to be scheduled on that node
  - nodeName – string
    - Request to schedule this pod onto a specific node
      - Scheduler simply schedules this pod onto that node, assuming that it fits resource requirements
  - restartPolicy – string
    - Restart policy for all containers within the pod
      - Always, OnFailure, Never
  - schedulerName – string
    - If specified, the pod will be dispatched by specified scheduler
  - securityContext – v1.PodSecurityContext
    - SecurityContext holds pod-level security attributes and common container settings
  - serviceAccountName – string
    - Name of the ServiceAccount to use to run this pod
  - subdomain – string
    - If specified, the fully qualified Pod hostname will be "...svc."
      - If not specified, the pod will not have a domainname at all
  - terminationGracePeriodSeconds – integer
    - Duration in seconds the pod needs to terminate gracefully
  - tolerations – v1.Toleration array
    - If specified, the pod's tolerations
  - volumes – v1.Volume array
    - List of volumes that can be mounted by containers belonging to the pod

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

```
$ 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}\""]

$ kubectl create -f hello.yaml
pod "hello-world" created
$ kubectl get pod
NAME          READY     STATUS    RESTARTS   AGE
hello-world   0/1       Pending   0          15s
$ kubectl logs hello-world
hello world
$ kubectl get pod
No resources found.
```

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 (when limits are defined)
  - 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: blogsite
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"
```

Eviction

- Eviction is the removal of a tenant from rental property by the landlord
  - What the Kubelet does to certain pods when resource levels become critical
- Types of eviction:
  - Soft – eviction that occurs only after a threshold is exceeded for a certain period of time
  - Hard – eviction that occurs immediately when a threshold is exceeded
- Kubelet eviction configuration:
  - `--eviction-soft` A set of eviction thresholds (e.g. memory.available<1.5Gi) that if met over a corresponding grace period would trigger a pod eviction
  - `--eviction-soft-grace-period` A set of eviction grace periods (e.g. memory.available=1m30s) that correspond to how long a soft eviction threshold must hold before triggering a pod eviction
  - `--eviction-hard` A set of eviction thresholds (e.g. memory.available<1Gi) that if met would trigger a pod eviction. (default "memory.available<100Mi")
  - `--eviction-max-pod-grace-period` Maximum allowed grace period (in seconds) to use when terminating pods in response to a soft eviction threshold being met
    - If negative, defers to pod specified value
  - `--eviction-minimum-reclaim` A set of minimum reclaims (e.g. imagefs.available=2Gi) that describes the minimum amount of resource the kubelet will reclaim when performing a pod eviction if that resource is under pressure
  - `--eviction-pressure-transition-period` Duration for which the kubelet has to wait before transitioning out of an eviction pressure condition (default 5m0s)
    - protects against oscillation in and out of a threshold

QoS

- Kubernetes uses QoS classes to make decisions about scheduling and evicting Pods
- When Kubernetes creates a Pod it assigns one of these QoS classes to the Pod:
  - Guaranteed – Pods are considered top-priority and are guaranteed to not be killed until they exceed their limits
    - For a Pod to be given a QoS class of Guaranteed:
      - Every Container in the Pod must have a memory limit and a memory request, and they must be the same
      - Every Container in the Pod must have a cpu limit and a cpu request, and they must be the same
  - Burstable – Pods have some form of minimal resource guarantee, but can use more resources when available
    - Under system memory pressure, these containers are more likely to be killed once they exceed their requests and no Best-Effort pods exist
    - A Pod is given a QoS class of Burstable if:
      - At least one Container in the Pod has a memory or cpu request
  - BestEffort – Pods will be treated as lowest priority
  - Processes in these pods are the first to get killed if the system runs out of memory but can use any amount of free memory on the node
    - For a Pod to be given a QoS class of BestEffort, the Containers in the Pod must not have any memory or cpu limits or requests

Pre/Post Actions

- Containers can be assigned lifecycle actions to run after the container starts and before the container stops
  - Convenient for initializing the started container or cleaning up before the container shutdown
- Management of the container blocks until lifecycle actions are complete
  - Unless the container process fails, in which case the action is aborted
- These hooks are called at least once (be prepared for more than one call!)
- Two types: HTTP and exec
- postStart
  - Called immediately after a container is created
    - No guarantee that the hook will execute before the container ENTRYPOINT
  - If the handler fails, the container is terminated and restarted according to its restart policy
- preStop
  - Called immediately before a container is terminated
  - The container is terminated only after the handler completes
  - The reason for termination is passed to the handler
    - "reason" values:
      - Delete - Delete command was issued via kubectl or the API
      - Health - Health check fails
      - Dependency - Dependency failure (e.g. disk mount failure)
  - If the hook hangs during execution, the Pod phase stays in a running state and never reaches failed

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: apache-hook
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: apache-hook
    spec:
      containers:
      - name: apache-hook
        image: bitnami/apache:latest
        ports:
        - containerPort: 80
        lifecycle:
          postStart:
            httpGet:
              path: http://my.regsvr.com/register/
              port: 80
          preStop:
            exec:
              command: ["/usr/bin/apachectl","-k", "graceful- stop"]
```

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

```
$ cat long.yaml

apiVersion: v1
kind: Pod
metadata:
  name: long-running
spec:
  containers:
  - name: long
    image: "ubuntu:14.04"
    command: ["/usr/bin/tail", "-f", "/dev/null"]


$ kubectl create -f long.yaml
pod "long-running" created

$ kubectl get pod
NAME           READY     STATUS    RESTARTS   AGE
long-running   1/1       Running   0          13s

$ 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...
```

v1.Volume Definition

- Volume represents a named volume in a pod that may be accessed by any container in the pod.
- http://kubernetes.io/docs/api-reference/v1/definitions/#_v1_volume

- v1.Volume – schema
  - name - string
  - hostPath - v1.HostPathVolumeSource
  - emptyDir - v1.EmptyDirVolumeSource
  - gcePersistentDisk - v1.GCEPersistentDiskVolumeSource
  - awsElasticBlockStore - v1.AWSElasticBlockStoreVolumeScouce
  - gitRepo - v1.GitRepoVolumeSource
  - Secret - v1.SecretVolumeSource
  - nfs - v1.NFSVolumeSource
  - Iscsi - v1.ISCSIVolumeSource
  - glusterfs - v1.GlusterfsVolumeSource
  - persistentVolumeClaim - v1.PersistentVolumeClaimVolumeSource
  - rbd - v1.RDBVolumeSource
  - flexVolume - v1.FlexVolumeSource
  - cinder - v1.CinderVolumeSource
  - cephfs - v1.CephFSVolumeSource
  - flocker - v1.FlockerVolumeSource
  - downwardAPI - v1.DownwardAPIVolumeSource
  - fc - v1.FCVolumeSource
  - azureFile - v1.AzureFileVolumeSource
  - configMap - v1.ConfigMapVolumeSource

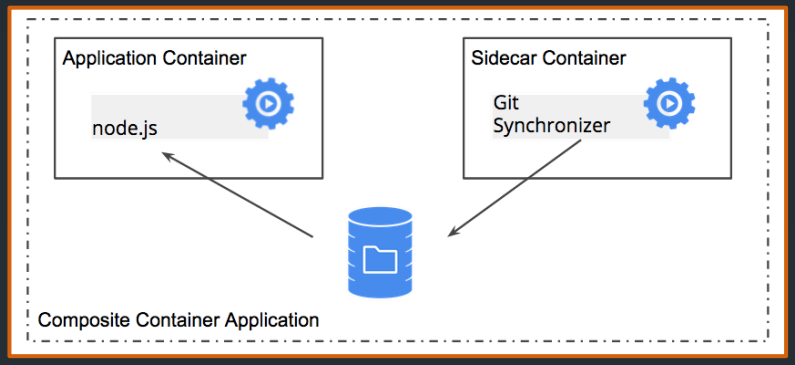
v1.Container

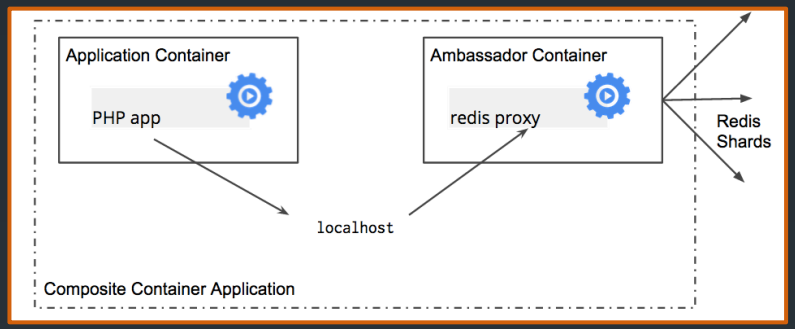
- A single application container that you want to run within a pod
- http://kubernetes.io/docs/api-reference/v1/definitions/#_v1_container

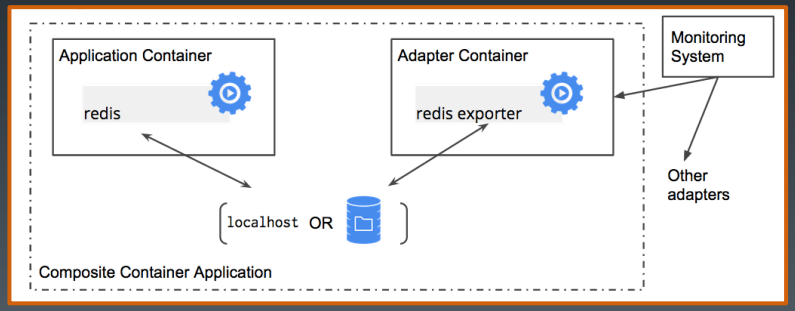
- v1.Container - schema
  - name - string
  - image - string
  - command - string array
  - args - string array
  - workingDir - string
  - ports - v1.ContainerPort array
  - env - v1.EnvVar array
  - resources - v1.ResourceRequirements
  - volumeMounts - v1.VolumeMount array
  - livenessProbe - v1.Probe
  - readinessProbe - v1.Probe
  - lifecycle - v1.Lifecycle
  - terminationMessagePath - string
  - imagePullPolicy - string
  - securityContext - v1.SecurityContext
  - stdin - boolean
  - stdinOnce - boolean
  - tty - boolean

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



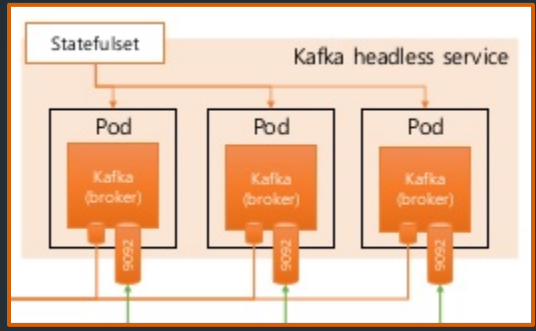


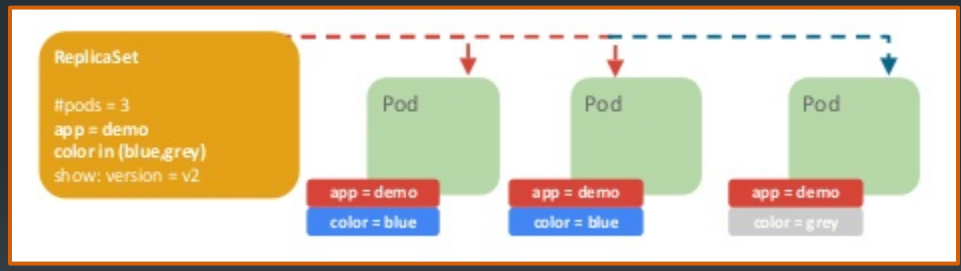


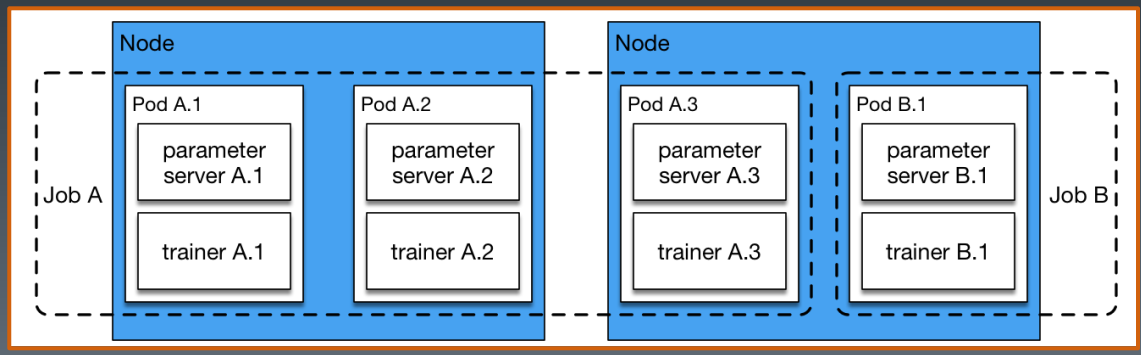
- Brendan Burns, Partner Architect at Microsoft and former Software Engineer at Google

Pod Creation

- While pods can be created directly, in a cluster they are almost always created through a Controller
- Deployments – for pods that back long running services with support for upgrades and scaling
  - Deployment object describes a desired state
    - Deployment controller changes the actual state to the desired state at a controlled rate
- Replication Controller (RC) – ensures that a specified number of Pod replicas are running at any one time
- Replica Set (RS) – the next-generation Replication Controller
  - Deployments recommended instead of directly using ReplicaSets, unless you require custom update orchestration or don't require updates
    - You may never need to manipulate ReplicaSet objects
  - Only difference between a RS and a RC right now is the selector support
- Daemon Set – ensures that all (or some) Nodes run a copy of a Pod
- Job – for one shot pods used to complete a start to finish task
- CronJob – manages time-based Jobs
- StatefulSets – for pods with an identity tied to state storage in volumes or other backends
  - PetSets – renamed StatefulSet starting in Kubernetes version 1.5
- HorizontalPodAutoscaler – automatically scales the number of pods in a RC, Deployment or RS based on observed metrics







Summary

- Kubelet is the the per node agent of a K8s cluster.
- A pod manifest is the primary input for a kubelet.
- A manifest is defined using a Pod Spec.
- A Pod spec can be used by itself (bare) or embedded in other resource types.

Lab 2

- Testing configs with a stand alone kubelet

3. Scheduling

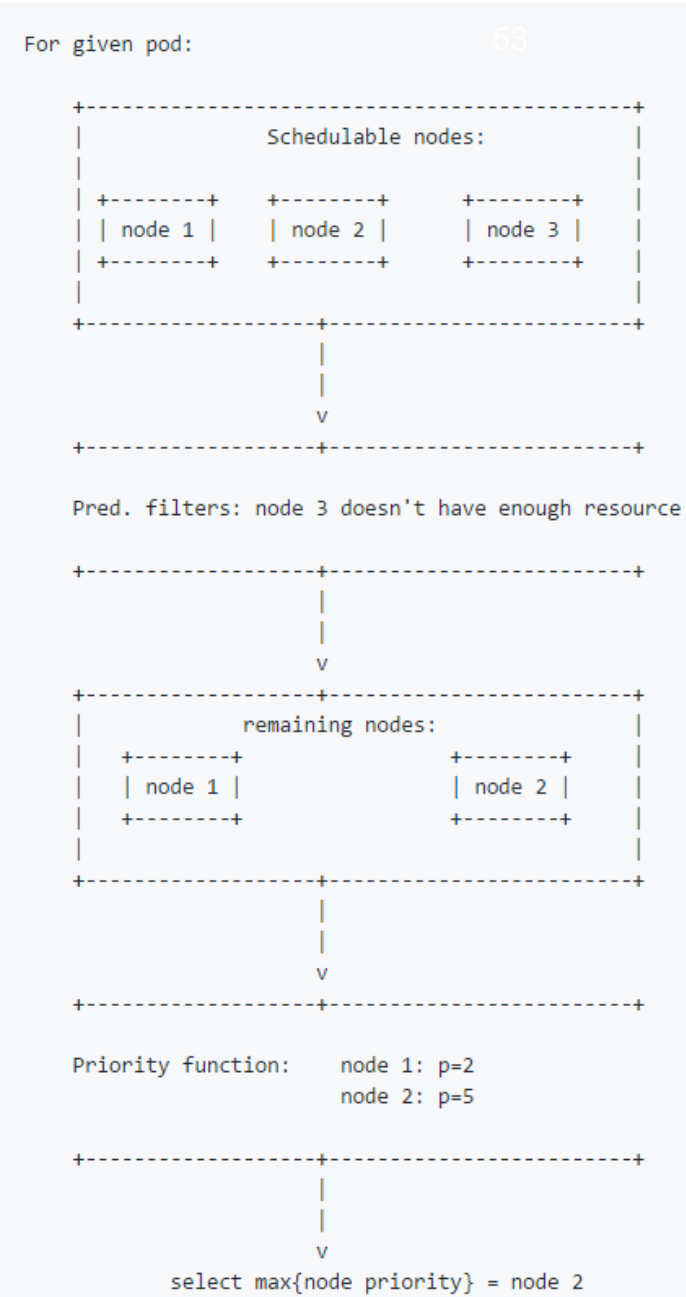
Objectives

- Understand the Kubernetes scheduler
- Explain the default scheduling process
- Examine ways to configure or extend the scheduler
- Understand taints and tolerations
- Learn administrative commands for evicting pods and cordoning off nodes

kube-scheduler

- The Kubernetes scheduler is a policy-rich, topology-aware, workload-specific function that significantly impacts availability, performance, and 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, and so on
- The Kubernetes scheduler runs as a process alongside the other master components such as the API server
- Its interface to the API server is to watch for Pods with an empty PodSpec.NodeName, and for each Pod, it posts a Binding indicating where the Pod should be scheduled

The scheduling algorithm



The scheduling process

- Node: a physical or virtual machine running Kubernetes, onto which pods can be scheduled.
- Pod: collocated group of application containers with shared volumes
  - 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 Controller (RC, RS, Deployment) even if creating a single pod
- The scheduler tries to find a Node for each Pod, one at a time, as it notices these Pods via watch, in three steps:
  - First it applies a set of "predicates" that filter out inappropriate nodes
    - For example, if the PodSpec specifies resource requests, then the scheduler will filter out nodes that don't have at least that much resources available (computed as the capacity of the Node minus the sum of the resource requests of the containers that are already running on the Node)
  - Second, it applies a set of "priority functions" that rank the nodes that weren't filtered out by the predicate check
    - For example, it tries to spread Pods across nodes and zones while at the same time favoring the least-loaded nodes (where "load" here is sum of the resource requests of the containers running on the node, divided by the node's capacity)
  - Finally, the Node with the highest priority is chosen (or, if there are multiple such nodes, then one of them is chosen at random)
    - The code for this main scheduling loop is in the function Schedule() in
      - plugin/pkg/scheduler/generic_scheduler.go

Core Scheduling Loop


```
// Schedule tries to schedule the given pod to one of nodes in the node list
// If it succeeds, it will return the name of the node
// If it fails, it will return a Fiterror error with reasons

func (g *genericScheduler) Schedule(pod *api.Pod, nodeLister algorithm.NodeLister) (string, error) {
        var trace *util.Trace
        if pod != nil {
                trace = util.NewTrace(fmt.Sprintf("Scheduling %s/%s", pod.Namespace, pod.Name))
        } else {
                trace = util.NewTrace("Scheduling <nil> pod") }
        defer trace.LogIfLong(20 * time.Millisecond)

        nodes, err := nodeLister.List()
        if err != nil {
                return "", err }
        if len(nodes.Items) == 0 {
                return "", ErrNoNodesAvailable }

        // Used for all fit and priority funcs.
        nodeNameToInfo, err := g.cache.GetNodeNameToInfoMap()
        if err != nil {
                return "", err }

        trace.Step("Computing predicates")
        filteredNodes, failedPredicateMap, err := findNodesThatFit(pod, nodeNameToInfo, g.predicates, nodes, g.extenders)
        if err != nil {
                return "", err }

        if len(filteredNodes.Items) == 0 {
                return "", &FitError{
                        Pod:              pod,
                        FailedPredicates: failedPredicateMap, } }

        trace.Step("Prioritizing")
        priorityList, err := PrioritizeNodes(pod, nodeNameToInfo, g.prioritizers, algorithm.FakeNodeLister(filteredNodes), g.extenders)
        if err != nil {
                return "", err }

        trace.Step("Selecting host")
        return g.selectHost(priorityList)
}
```

From (generic_scheduler.go)
https://github.com/kubernetes/kubernetes/blob/master/plugin/pkg/scheduler/core/generic_scheduler.go

Built-in Predicates


```
...

// nodeMatchesNodeSelectorTerms checks if a node's labels satisfy a list of node selector terms,
// terms are ORed, and an emtpy a list of terms will match nothing

func nodeMatchesNodeSelectorTerms(node *api.Node, nodeSelectorTerms []api.NodeSelectorTerm) bool {
        for _, req := range nodeSelectorTerms {
                nodeSelector, err := api.NodeSelectorRequirementsAsSelector(req.MatchExpressions)
                if err != nil {
                        glog.V(10).Infof("Failed to parse MatchExpressions: %+v, regarding as not match.", req.MatchExpressions)
                        return false
                }
                if nodeSelector.Matches(labels.Set(node.Labels)) {
                        return true
                }
        }
        return false
}

...
```

From (predicates.go)
https://github.com/kubernetes/kubernetes/blob/master/plugin/pkg/scheduler/algorithm/predicates/predicates.go

```
                        {
                        "kind" : "Policy",
                        "apiVersion" : "v1",
                        "predicates" : [
                                {"name" : "PodFitsPorts"},
                                {"name" : "PodFitsResources"},
                                {"name" : "NoDiskConflict"},
                                {"name" : "NoVolumeZoneConflict"},
                                {"name" : "MatchNodeSelector"},
                                {"name" : "HostName"}
                                ],
                        "priorities" : [
                                {"name" : "LeastRequestedPriority", "weight" : 1},
                                {"name" : "BalancedResourceAllocation", "weight" : 1},
                                {"name" : "ServiceSpreadingPriority", "weight" : 1},
                                {"name" : "EqualPriority", "weight" : 1}
                                ]
                        }
```

From (scheduler-policy-config.json)
https://github.com/kubernetes/kubernetes/blob/master/examples/scheduler-policy-config.json

Filtering Nodes

- The purpose of filtering the nodes is to filter out the nodes that do not meet certain requirements of the Pod
  - NoDiskConflict: Evaluate if a pod can fit due to the volumes it requests and those that are already mounted
    - Currently supported volumes are: AWS EBS, GCE PD, ISCSI and Ceph RBD
      - Only Persistent Volume Claims for those supported types are checked
      - Persistent Volumes added directly to pods are not evaluated and are not constrained by this policy
  - NoVolumeZoneConflict: Evaluate if the volumes a pod requests are available on the node, given the Zone restrictions
  - PodFitsResources: Check if the free resource (CPU and Memory) meets the requirement of the Pod
    - The free resource is measured by the capacity minus the sum of requests of all Pods on the node
  - PodFitsHostPorts: Check if any HostPort required by the Pod is already occupied on the node
  - HostName: Filter out all nodes except the one specified in the PodSpec's NodeName field
  - MatchNodeSelector: Check if the labels of the node match the labels specified in the Pod's nodeSelector field and, as of Kubernetes v1.2, also match the scheduler.alpha.kubernetes.io/affinity pod annotation if present
  - MaxEBSVolumeCount: Ensure that the number of attached ElasticBlockStore volumes does not exceed a maximum value (by default, 39, since Amazon recommends a maximum of 40 with one of those 40 reserved for the root volume -- see Amazon's documentation)
    - The maximum value can be controlled by setting the `KUBE_MAX_PD_VOLS` environment variable
  - MaxGCEPDVolumeCount: Ensure that the number of attached GCE PersistentDisk volumes does not exceed a maximum value (by default, 16, which is the maximum GCE allows -- see GCE's documentation)
    - The maximum value can be controlled by setting the `KUBE_MAX_PD_VOLS` environment variable
  - CheckNodeMemoryPressure: Check if a pod can be scheduled on a node reporting memory pressure condition
    - Currently, no BestEffort should be placed on a node under memory pressure as it gets automatically evicted by kubelet
  - CheckNodeDiskPressure: Check if a pod can be scheduled on a node reporting disk pressure condition
    - Currently, no pods should be placed on a node under disk pressure as it gets automatically evicted by kubelet

Built-in Priorities


```
...

// LeastRequestedPriority is a priority function that favors nodes with fewer requested resources
// It calculates the percentage of memory and CPU requested by pods scheduled on the node, and prioritizes
// based on the minimum of the average of the fraction of requested to capacity
// Details: cpu((capacity - sum(requested)) * 10 / capacity) + memory((capacity - sum(requested)) * 10 / capacity) / 2

func LeastRequestedPriority(pod *api.Pod, nodeNameToInfo map[string]*schedulercache.NodeInfo, nodeLister
algorithm.NodeLister) (schedulerapi.HostPriorityList, error) {
        nodes, err := nodeLister.List()
        if err != nil {
                return schedulerapi.HostPriorityList{}, err
        }

        list := schedulerapi.HostPriorityList{}
        for _, node := range nodes.Items {
                list = append(list, calculateResourceOccupancy(pod, node, nodeNameToInfo[node.Name]))
        }
        return list, nil
}

...
```

From (least_requested.go)
https://github.com/kubernetes/kubernetes/blob/master/plugin/pkg/scheduler/algorithm/priorities/

```
{
"kind" : "Policy",
"apiVersion" : "v1",
"predicates" : [
      {"name" : "PodFitsPorts"},
      {"name" : "PodFitsResources"},
      {"name" : "NoDiskConflict"},
      {"name" : "NoVolumeZoneConflict"},
      {"name" : "MatchNodeSelector"},
      {"name" : "HostName"}
      ],
"priorities" : [
      {"name" : "LeastRequestedPriority", "weight" : 1},
      {"name" : "BalancedResourceAllocation", "weight" : 1},
      {"name" : "ServiceSpreadingPriority", "weight" : 1},
      {"name" : "EqualPriority", "weight" : 1}
      ]
}
```

From (scheduler-policy-config.json)
https://github.com/kubernetes/kubernetes/blob/master/examples/scheduler-policy-config.json

Ranking Nodes

- The filtered nodes are considered suitable to host the Pod, and it is often that there are more than one nodes remaining
- Kubernetes prioritizes the remaining nodes to find the "best" one for the Pod
- A priority function gives a score from 0-10 with 10 representing for "most preferred"
- Each priority function is weighted by a positive number and the final score of each node is calculated by adding up all the weighted scores
  - finalScoreNodeA = (weight1 * priorityFunc1) + (weight2 * priorityFunc2)
- If there is more than one node with an equal highest score, a random one among them is chosen
- Currently, Kubernetes scheduler provides some practical priority functions, including:
  - LeastRequestedPriority: The node is prioritized based on the fraction of the node that would be free if the new Pod were scheduled onto the node
    - (capacity of the Node – (the sum of requests of all Pods already on the node + request of Pod that is being scheduled)) / capacity)
      - CPU and memory are equally weighted
    - The node with the highest free fraction is the most preferred
    - Note that this priority function has the effect of spreading Pods across the nodes with respect to resource consumption
  - BalancedResourceAllocation: This priority function tries to put the Pod on a node such that the CPU and Memory utilization rate is balanced after the Pod is deployed
  - SelectorSpreadPriority: Spread Pods by minimizing the number of Pods belonging to the same service, Replication Controller, or Replica Set on the same node
    - If zone information is present on the nodes, the priority will be adjusted so that pods are spread across zones and nodes
  - CalculateAntiAffinityPriority: Spread Pods by minimizing the number of Pods belonging to the same service on nodes with the same value for a particular label
  - ImageLocalityPriority: Nodes are prioritized based on locality of images requested by a pod
    - Nodes with larger size of already-installed packages required by the pod will be preferred over nodes with small total size of (or no) already-installed packages required by the pod

Select Wining Node (if tie)


```
...
// selectHost takes a prioritized list of nodes and then picks one
// in a round-robin manner from the nodes that had the highest score

func (g *genericScheduler) selectHost(priorityList schedulerapi.HostPriorityList) (string, error) {
        if len(priorityList) == 0 {
                return "", fmt.Errorf("empty priorityList")
        }

        sort.Sort(sort.Reverse(priorityList))
        maxScore := priorityList[0].Score
        firstAfterMaxScore := sort.Search(len(priorityList), func(i int) bool { return priorityList[i].Score < maxScore })

        g.lastNodeIndexLock.Lock()
        ix := int(g.lastNodeIndex % uint64(firstAfterMaxScore))
        g.lastNodeIndex++
        g.lastNodeIndexLock.Unlock()

        return priorityList[ix].Host, nil
}

...
```

From (generic_scheduler.go)
https://github.com/kubernetes/kubernetes/blob/master/plugin/pkg/scheduler/core/generic_scheduler.go

https://github.com/kubernetes/community/blob/master/contributors/devel/scheduler_algorithm.md

Affinity & Anti-Affinity

- NodeAffinity – generalization of the nodeSelector feature but using a more expressive syntax
  - Constrain which nodes your pod is eligible to schedule on, based on labels on the node
- Inter-pod affinity and anti-affinity (beta) – specify rules about how pods should be placed relative to one another
  - Based on labels on pods that are already running on the node
- Two types:
  - requiredDuringSchedulingIgnoredDuringExecution: Specifies rules that *must* be met for a pod to schedule
    - If no node/pod matches the criteria (plus all of the other normal criteria, such as having enough free resources for the pod's resource request), then the pod won't be scheduled
  - preferredDuringSchedulingIgnoredDuringExecution: Specifies *preferences* that the scheduler will try to enforce but will not guarantee
    - If nodes/pods match the rules, they will be chosen first, and only if no preferred nodes are available will non-preferred nodes be chosen

- "IgnoredDuringExecution" part of the names means that, if labels on a node change at runtime such that the affinity rules on a Pod are no longer met, the Pod will *still continue to run on the node*
- Anti-affinity can be achieved by using negative operators (eg: `NotIn`)

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

Taints & Tolerations

- A “Taint” is a condition assigned to a node
- The kubectl command can update taints on one or more nodes
  - `$ kubectl taint nodes nodea nodetype=master:NoSchedule`
    - This places a taint on nodea
    - Taints have an arbitrary key (“nodetype” in this example)
    - Taints have an arbitrary value (“master” in this example)
    - Taints also have a defined effect (“NoSchedule” in this example)
- Pods can specify taints that they can ignore (tolerate)
  - If a pod does not explicitly tolerate a taint it will be affected by the taint’s effect
    - In the example above, a pod can not be scheduled on nodea unless it tolerates the “nodetype=master:NoSchedule” taint
  - To tolerate the above taint you could add the following to the Pod spec:

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

- Taint effects:
  - NoSchedule: pods can not be scheduled to the node unless they can tolerate the taint (pods already on the node are unaffected)
  - PreferNoSchedule: the scheduler “tries” not to run pods on the node unless they can tolerate the taint
  - NoExecute: like NoSchedule but also evicts pods not tolerating the taint even if they are already scheduled and running on the node

Scheduler extensibility

- The scheduler is extensible
- The cluster administrator can choose which of the pre-defined scheduling policies to apply, and can add new ones

- The built-in predicates and priorities are defined in:
  - *plugin/pkg/scheduler/algorithm/predicates/predicates.go*
  - *plugin/pkg/scheduler/algorithm/priorities/*

- The policies that are applied when scheduling can be chosen in one of two ways.
  - Normally, the policies used are selected by the functions `defaultPredicates()` and `defaultPriorities()` in
    - *plugin/pkg/scheduler/algorithmprovider/defaults/defaults.go*
  - However, the choice of policies can be overridden…

Scheduler Policies

- K8s < v1.6:
  - The Kubernetes Scheduler policies overridden by passing the flag `--policy-config-file` to the scheduler pointing to a JSON file specifying which scheduling policies to use
- K8s >= v1.6
  - New approach is to use a ConfigMap to supply the same information to the scheduler
    - `--policy-configmap`
      - Supplies the ConfigMap name that contains scheduler's policy configuration
    - `--policy-configmap-namespace`
      - The namespace where the policy ConfigMap is located
        - The system namespace will be used if this is not provided or is empty (default `kube-system`)
  - `--policy-config-file`
    - Only used if policy ConfigMap is not provided or `--use-legacy-policy-config==true`

Policy Config Files were deprecated in K8s 1.6

kubernetes / kubernetes

Code | Issues 2,391 | Pull requests 430 | Wiki | Pulse | Graphs

Branch: master ▾ kubernetes / examples / scheduler-policy-config.json

justinsb Ubernetes Lite: Volumes can dictate zone scheduling

2 contributors

19 lines (18 sloc) 471 Bytes

```
{
"kind" : "Policy",
"apiVersion" : "v1",
"predicates" : [
        {"name" : "PodFitsPorts"},
        {"name" : "PodFitsResources"},
        {"name" : "NoDiskConflict"},
        {"name" : "NoVolumeZoneConflict"},
        {"name" : "MatchNodeSelector"},
        {"name" : "HostName"}
        ],
"priorities" : [
        {"name" : "LeastRequestedPriority", "weight" : 1},
        {"name" : "BalancedResourceAllocation", "weight" : 1},
        {"name" : "ServiceSpreadingPriority", "weight" : 1},
        {"name" : "EqualPriority", "weight" : 1}
        ]
```

http://kubernetes.io/v1.1/docs/devel/scheduler.html

Custom Schedulers (beta)

- Users can create custom schedulers
  - You can use the normal K8s scheduler with a custom configuration
  - You can write a completely new scheduler of your own delegating responsibility for scheduling arbitrary subsets of pods to your own custom scheduler(s) that run(s) alongside, or instead of, the default Kubernetes scheduler
- Running a scheduler with a unique name string makes it identifiable as a non-default scheduler:
  - `--scheduler-name <string>`
    - Names the scheduler
- Pods can select a non-default scheduler using the "spec.SchedulerName" pod spec setting
  - This value defaults to "default-scheduler", the name of the scheduler registered with no `--scheduler-name`

Administrative Commands

- The K8s API and kubectl offer several scheduler control features useful to sys admins
  - `cordon` Mark node as unschedulable
  - `uncordon` Mark node as schedulable
  - `drain` Drain node in preparation for maintenance (progressively reschedules all pods to other nodes)

```
user@nodea:~/temp$ kubectl cordon nodeb
node "nodeb" cordoned
user@nodea:~/temp$ kubectl uncordon nodeb
node "nodeb" uncordoned
```

Summary

- The responsibility of the scheduler is to place Pods on Nodes
- Scheduling a Pod at time, nodes are selected by the following steps:
  - **Predicates** – ex. Have required resources available
  - **Priority** – ex. Does a particular node have additional favorable conditions
  - **Random selection** – ex. Node tie breaking
- Use policy files to rebalancing existing scheduling rules
- Distributed scheduling is a complex subject, orchestration frameworks consider scheduling a competitive advantage (what is fair after all?)

Lab 3

- Working with the scheduler