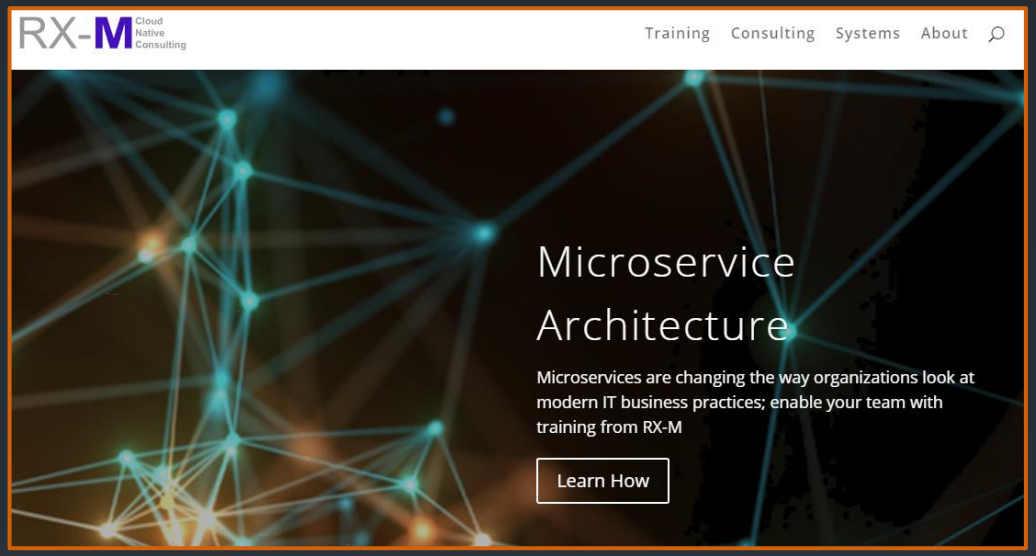
RX-M Cloud Native Consulting

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

Overview

Day One

1. Kubernetes Architecture
2. Advanced Pod configs and kubelet

Day Two

4. Scheduling
5. Services

Day Three

4. Software Defined Networking (SDN)
5. DNS and Service Discovery

Day Four

7. API and Security
8. etcd

Prerequisites:

RX-M Kubernetes Foundation and Docker Foundation or equivalent experience

Equivalent experience:

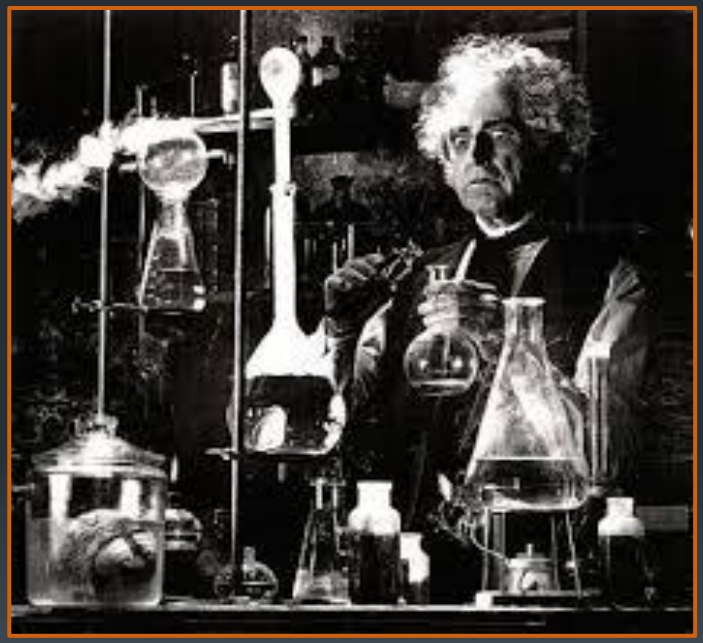
- Basic familiarity with Kubernetes cluster components and kubectl client subcommands
- Basic familiarity with all docker client subcommands
- Clear understanding of containers, isolation, container volumes, and port mapping

Administrative Info

- Length: 4 Days
- Format: Lecture/Labs/Discussion
- Schedule: 8:00AM – 2:00PM
  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. Introduce concepts and ecosystems
     - Covering concepts and where things fit in the world is the primary purpose of the lecture/discussion sessions
     - The instructor will take you on a tour of the museum
       - Like a museum tour, you should interact with the instructor (tour guide), ask questions, discuss
       - Like a museum tour, you will not have time to read the slides during the tour, instead, the instructor will discuss and point out the highlights of the slides (exhibits) which will be waiting for you to read in depth later should you like to dig deeper
  2. Impart practical experience
     - This is the primary purpose of the labs
     - Classes rarely have time for complete real world projects so think of the labs as thought experiments
       - Like hands on exhibits at the museum



Day 1

1. Kubernetes Architecture
2. Advanced Pod configs and kubelet

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
- Examine K8s infrastructure deployment methods

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
  - 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
  - Containers are isolated from each other and from the host:
    - Have their own filesystems
    - Can't see each others' processes
    - Computational resource usage can be bounded
  - 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

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

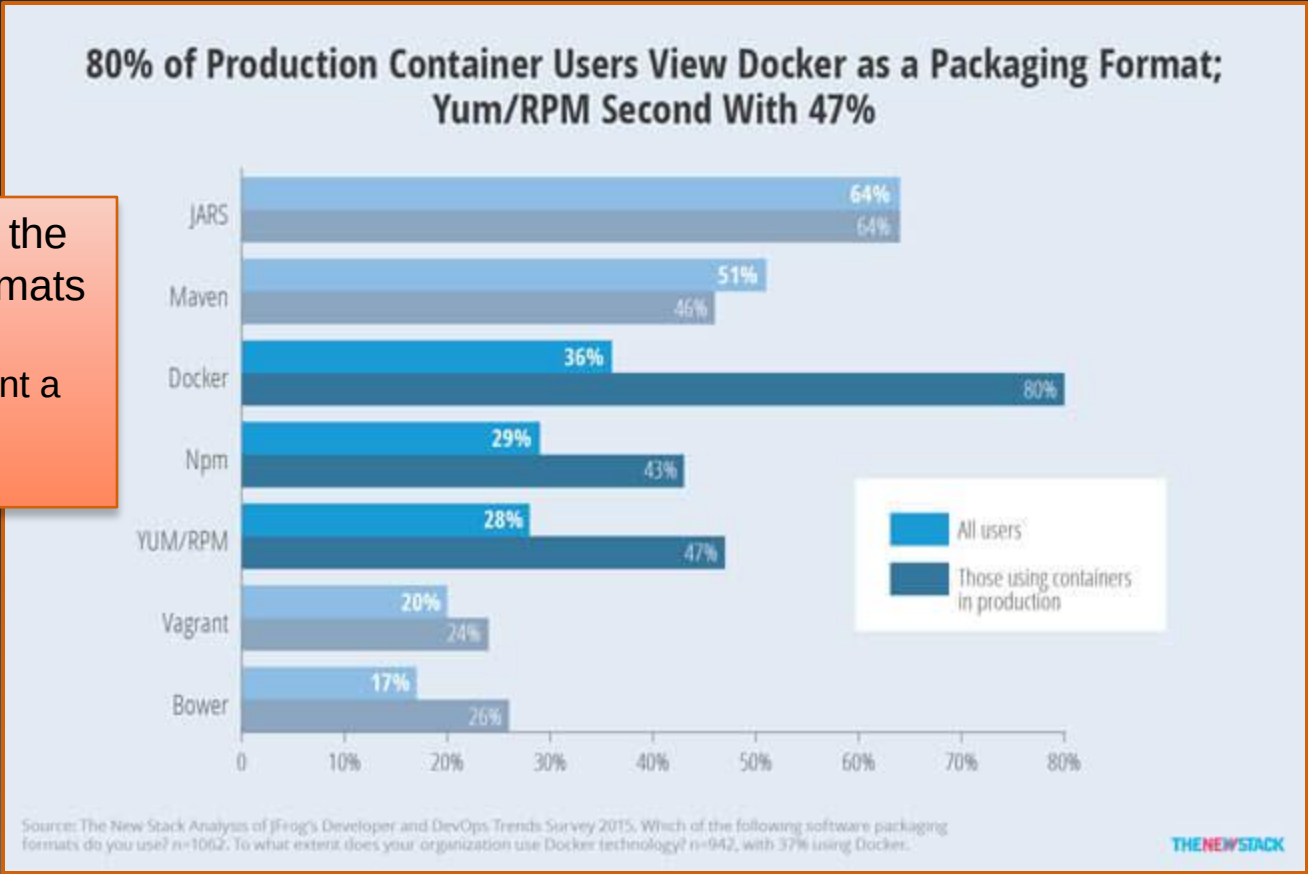
- K8s
  - Benefits include:
    - Co-locating helper processes - Pods
    - Mounting storage systems - Volumes
    - Distributing secrets - Secrets
    - Distributing configuration - ConfigMaps
    - Application health checking - liveness/readiness checks
    - Replicating application instances - Deployment,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

K8s + containers = Isolation and orchestration

Will container images replace RPM/DEB?

- Containers offer much more than packaging for developers and application architects
- Yet to many pure play infrastructure operators they are simply a better, cross platform, package format

JFrog survey: Which of the following packaging formats do you use?
(note: JFrog users represent a distinct JVM ecosystem selection bias)

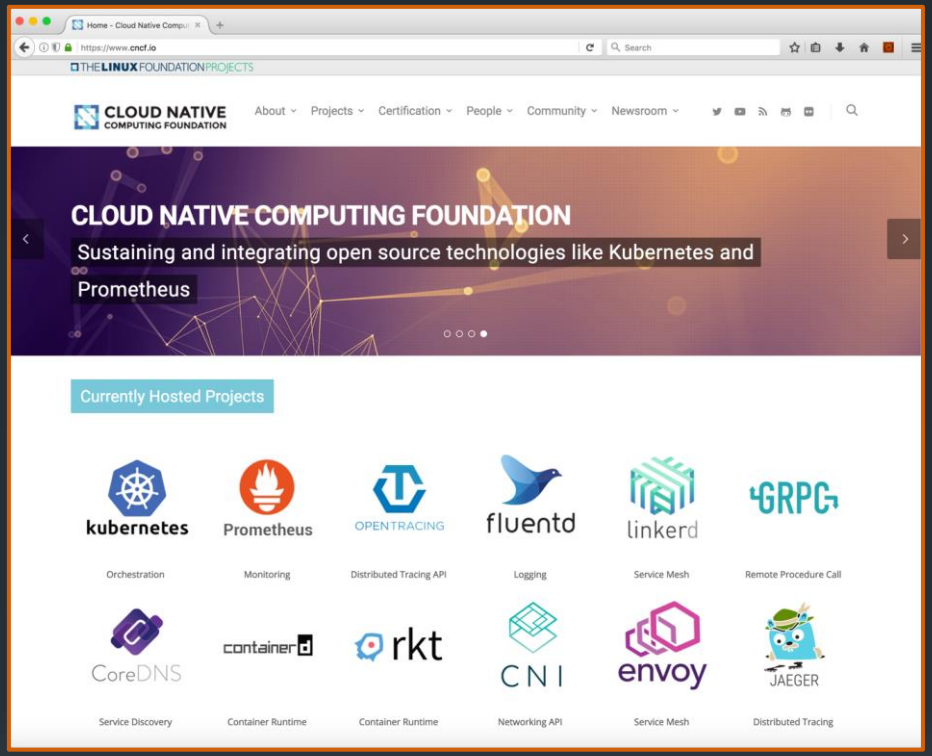


Container & Kubernetes Timeline

- 2004 Google begins work on what becomes Borg
- 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]
- 2011 Container Unification agreement during kernel summit (In tree unified CGroups and Namespaces)
- 2013 Linux supports LXC, Docker, OpenVZ [Linux 3.10 & 3.12 LTS] dotCloud becomes Docker Inc.
- 2013-05 – CoreOS founded – creators of CoreOS Container Linux, the rkt container CLI, etcd distributed key-value store & flannel SDN (1st release October 2013)
- 2014-06 – Docker v1.0
- 2014-07 – Docker v1.1 released (Docker Engine + Docker Hub + APIs) and Docker acquires Orchard (Fig renamed Compose)
- 2014-09 – Google (Beda/Burns/McLuckie) open source Kubernetes
- 2015-01 – etcd v2.0 released; first major stable release
- 2015-06 – 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
- 2015-11 – Docker v1.9 released with support for multi-host networking and named volumes
- 2016-04 – Docker v1.11 released with OCI support (built on containerd and runC)
- 2016-06 – etcd v3.0 released – solves v2 scalability issues, uses new storage engine and gRPC API
- 2016-08 – Docker v1.12 released with integrated cluster management (Swarm Mode)
- 2016-09 – K8s v1.4 kubeadm (alpha)
- 2016-12 – K8s v1.5 StatefulSets
- 2017-03 – K8s v1.6 CRI and federation; etcd3 encouraged w/ protobuf encoding (irreversible change)
- 2017-06 – K8s v1.7 network policy for pod-to-pod communication, automated updates to StatefulSets & DaemonSets
- 2017-09 – K8s 1.8 Containerd alpha support, RBAC to stable, egress network policy (beta), kubelet TLS cert rotation (beta)
- 2017-12 – K8s 1.9 etcd 3.1 support, CSI alpha, Windows server node beta, federation moved out of tree
- 2018-03 – K8s 1.10 CSI moves to beta, ability to switch the DNS service to CoreDNS at install time
- 2018-07 – K8s 1.11 deprecation of Heapster, graduation of IPVS-based load balancing and CoreDNS to general availability
- 2018-12 – K8s 1.13 etcd encryption out of experimental, deprecated etcd2 storage backend removed
- 2019-03 – K8s 1.14 local PVs, kubectl plugins, and Windows support – all graduated to GA

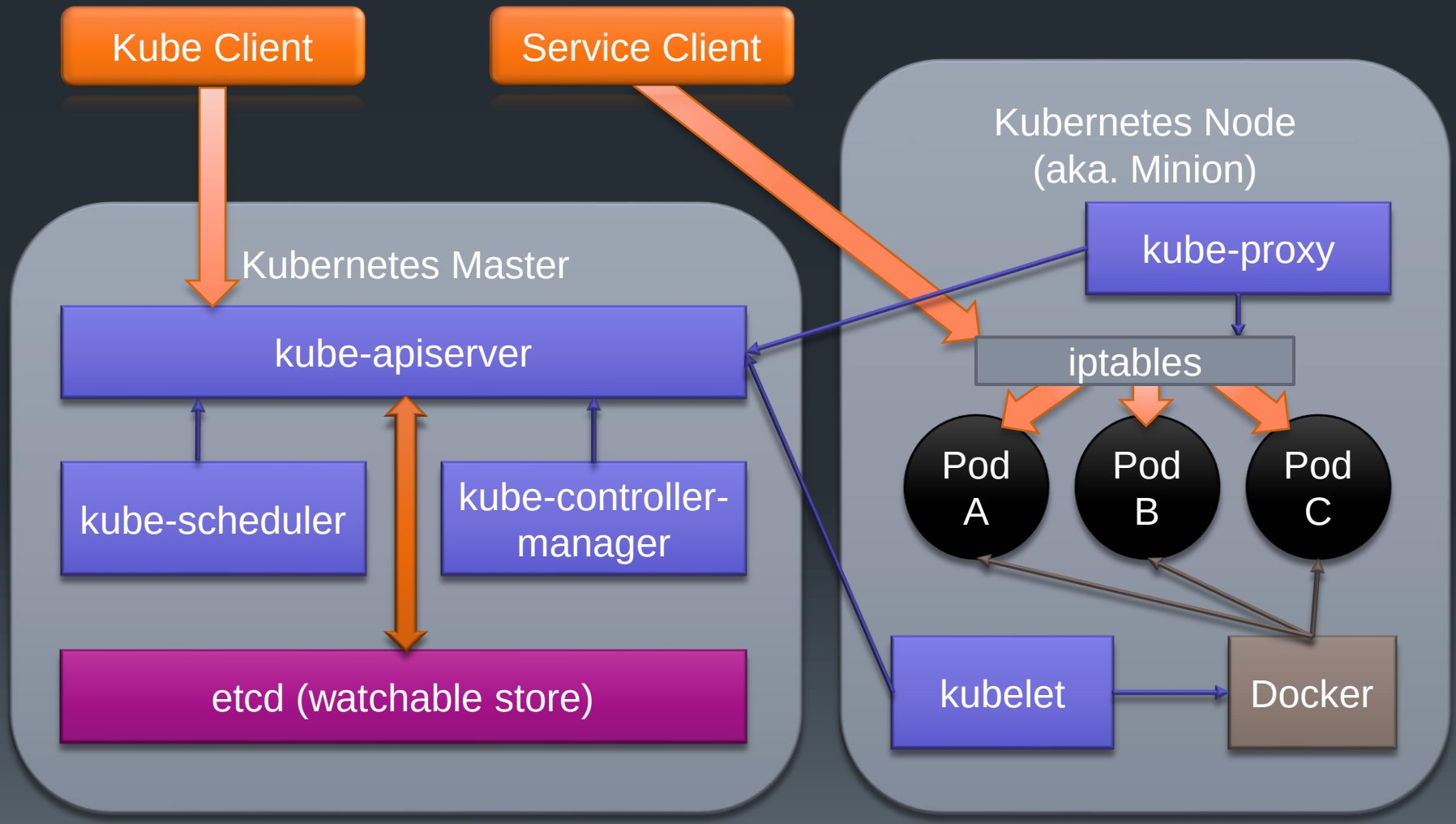
Container Orchestration Initiatives

- 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 handed Kubernetes over to the foundation when it was founded
    - The CNI, Containerd, CoreDNS, Envoy, Fluentd, gRPC, Jaeger, Linkerd, OpenTracing, Prometheus, and rkt projects have also been moved under the foundation

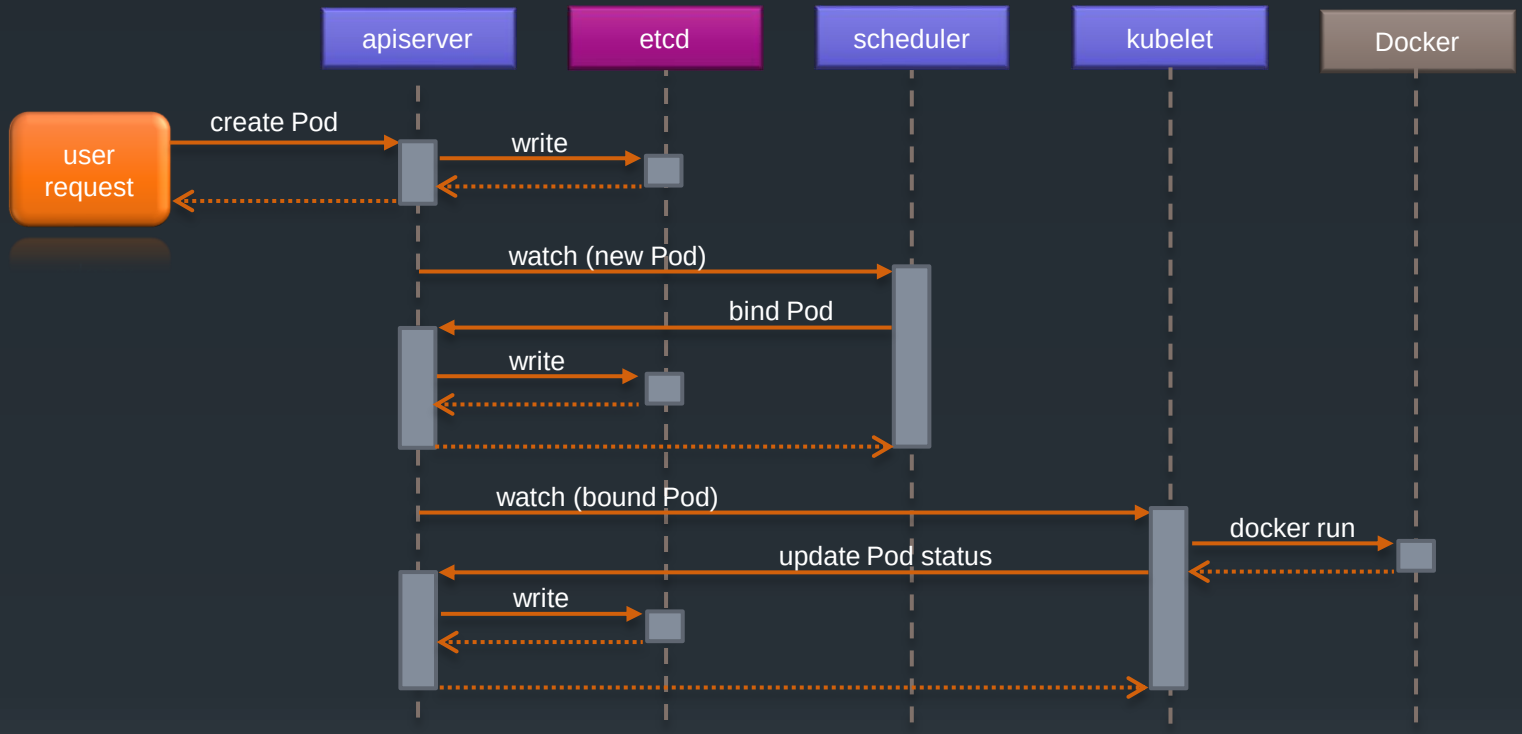


- Members include:
  - Amazon, Cisco, CoreOS, DellEMC, Docker, Fujitsu, Google, Huawei, IBM, Intel, Joyent, Mesosphere, Microsoft, Oracle, Pivotal, Red Hat, Samsung SDS, SAP, Supernap, Vmware, Alibaba Cloud, AT&T, NetApp, SUSE, Tencent Cloud, ZTE, 99 Cloud, Ace Demand, Amihan, Apcera, Apigee Aporeto, Applatix, Apprenda, Aqua Security, AVI Networks, Beijing Shureyun Technology, Bitnami, Bloomberg, Buoyant, Caicloud, Canonical, Capital One, Centrify, Chef, CloudByte, Cloudsoft, CloudVisory, CodeFresh, Container Solutions, Coos, Crunchy, Dao Cloud, Datawire, Deis, Diamanti, Digital Ocean, Dynatrace, Easy Stack, eBay, Eldarion, Exoscale, Galactic Fog, Ghostcloud, Giant Swarm, GitHub, GitLab, Goldman Sachs, Gravitational, Gronau, H3C, Harmonycloud, Heptio, Hewlett Packard Enterprise, Iguaz.io, Infoblox, InfoSiftr, Inwinstack, Juniper, Kinvolk, Kublr, Livewyer, Logdna, Loodse, Minio, Mirantis, Morgan Stanley, NCSoft, NEC, Netsil, Nginx, Nirmata Packet, Platform 9, Plexistor, Portworx, QAware, RackN, Rancher, Resin.io, RX-M, Solinea, Splunk, Stack Point Cloud, Storage OS, Supergiant, Sysdig, Tenxcloud, Tigera, Treasure Data, Twistlock, Twitter, Univa, Virtuozzo, Weaveworks, WSO2, Box, Kuelap, reddit, New York Times, Ticketmaster, Vevo, Zalando, Zendesk, Cloud Foundry, Wikimedia, Zhejiang University, and many more!

Kubernetes Architecture



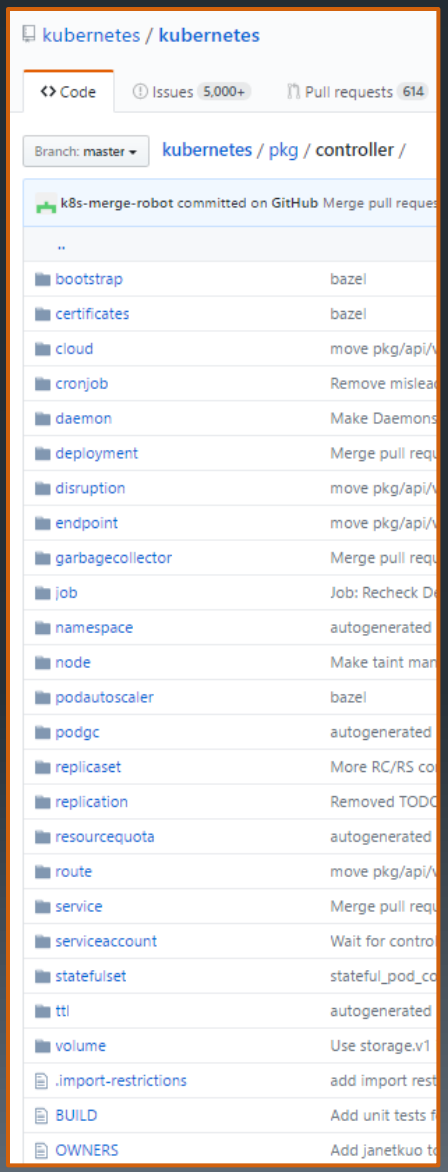
Pod Creation Flow



- 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

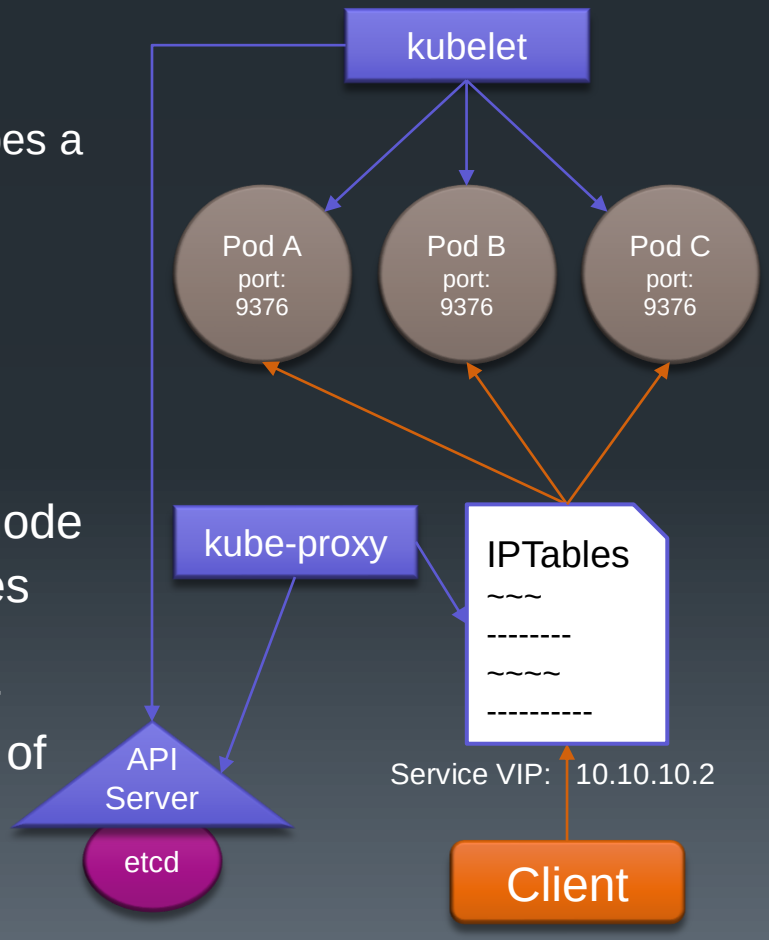
Master Components

- 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



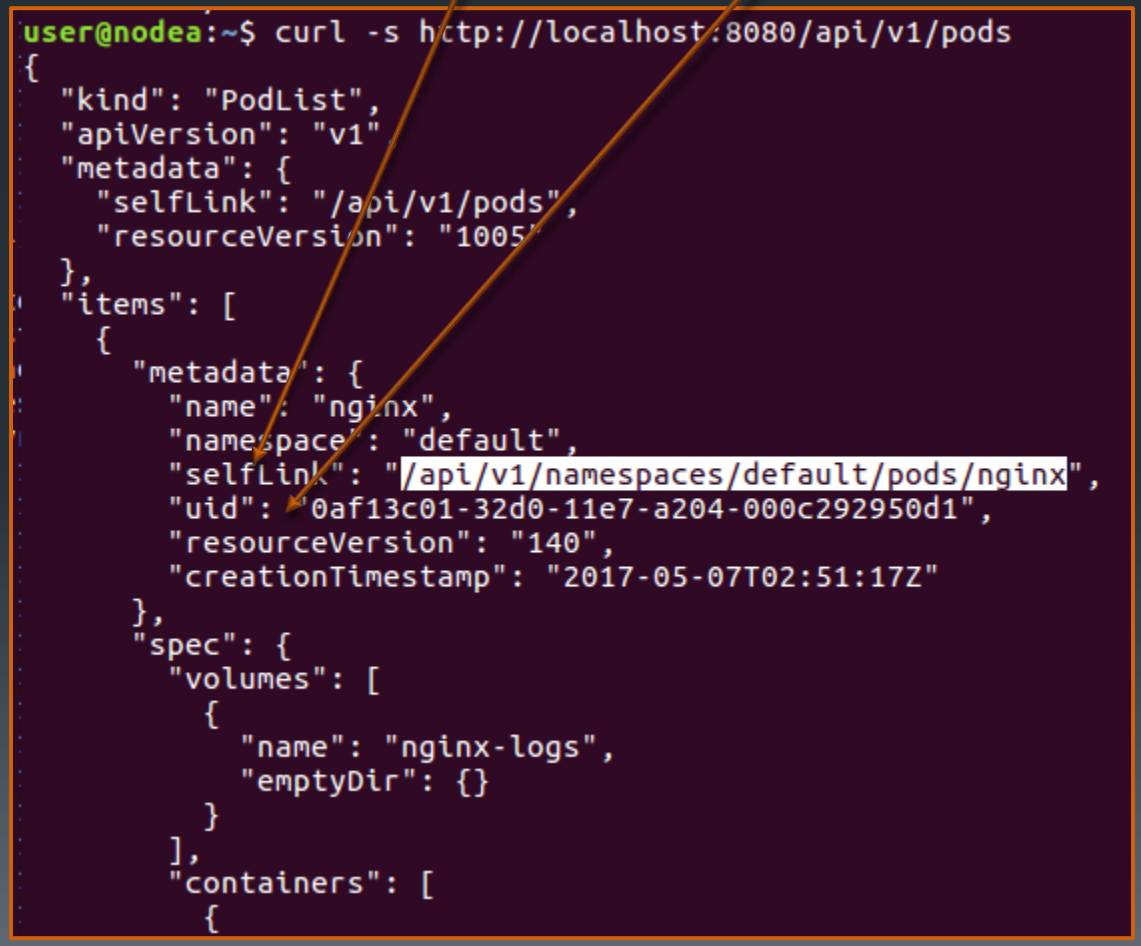
Node Components

- 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



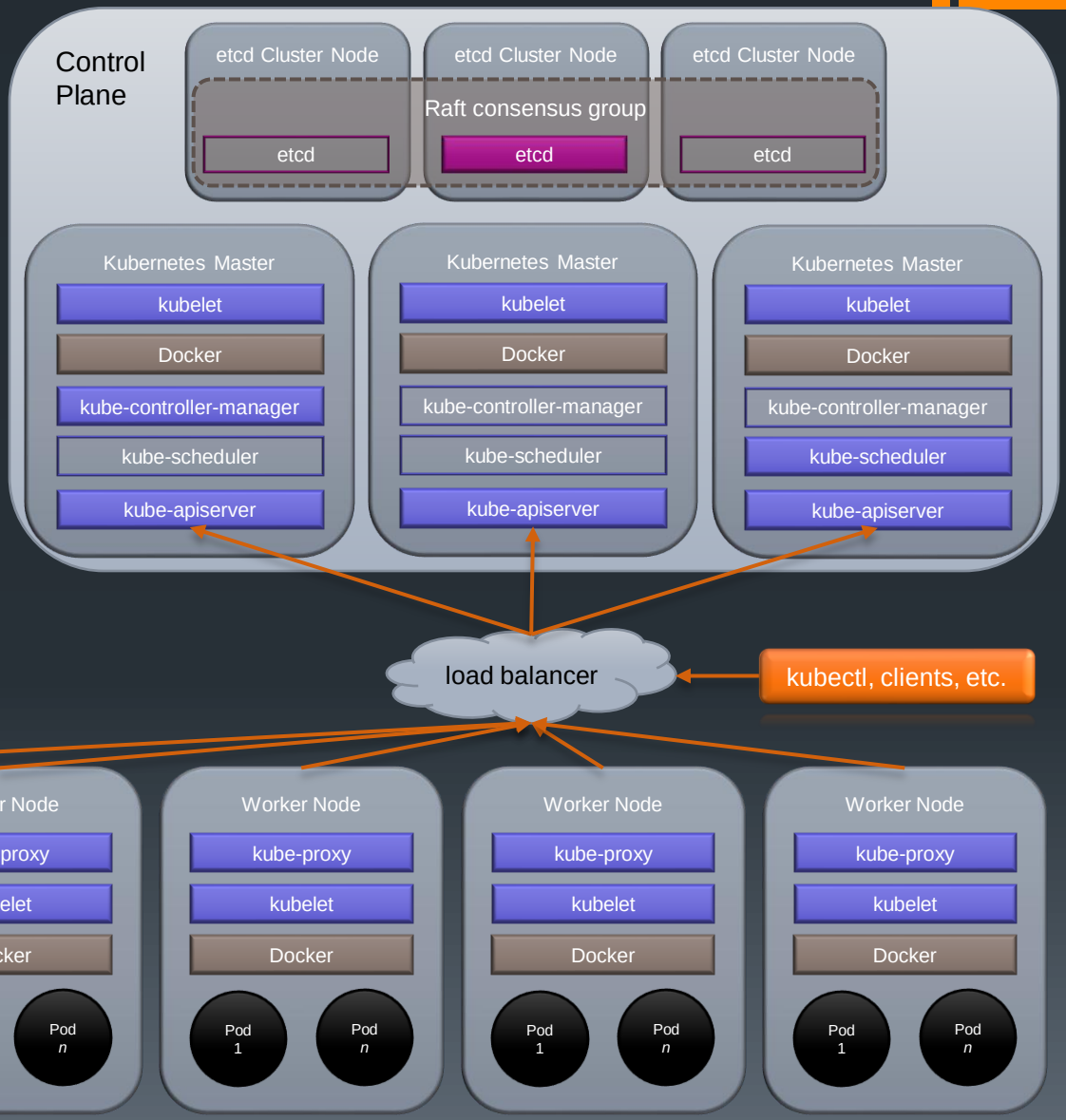
Object Names

- 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



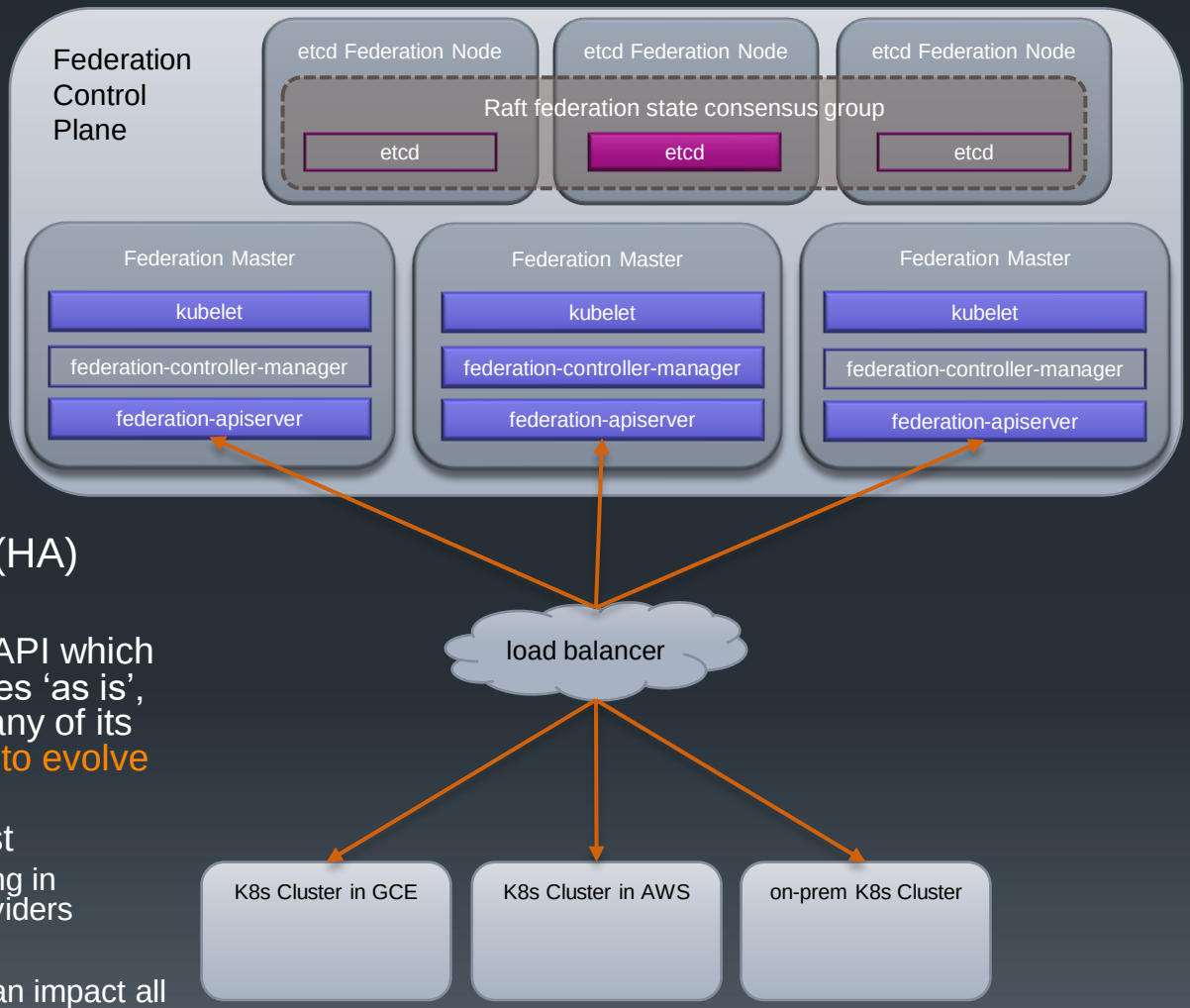
Scaled Architecture & HA Model

- 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



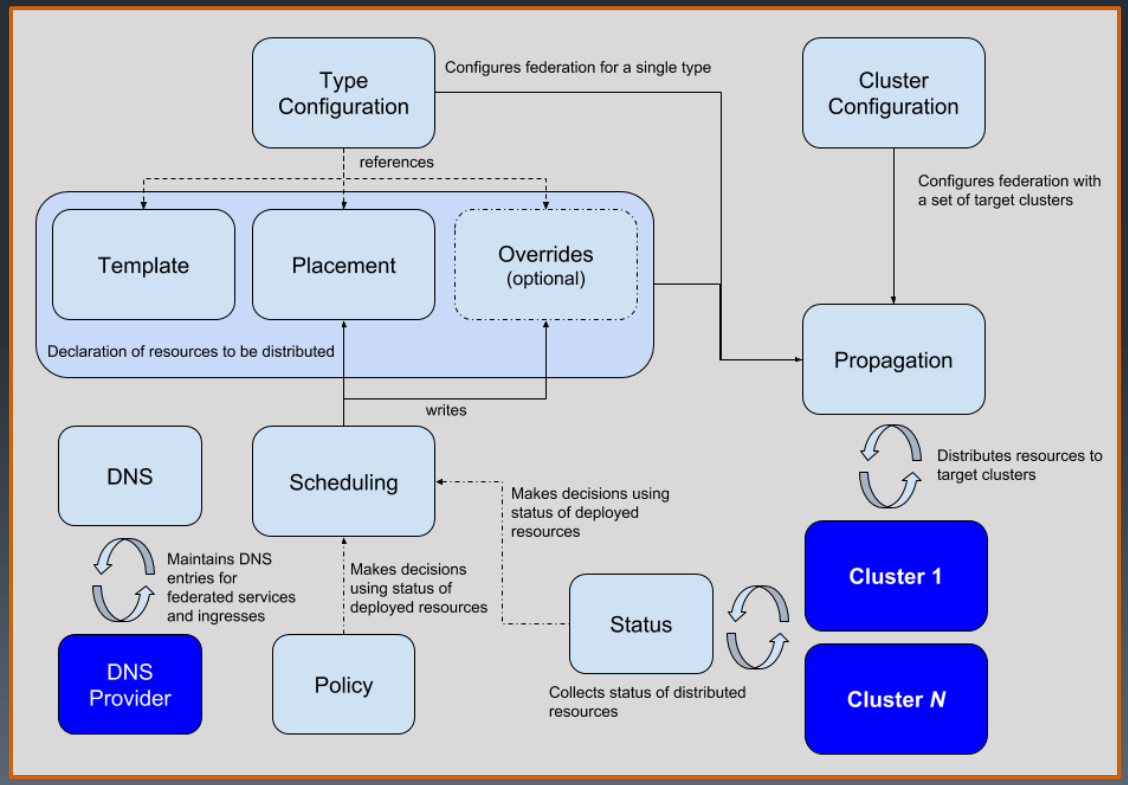
Federated Architecture (abandoned)

- 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
  - The current Kubernetes federation API which reuses the Kubernetes API resources 'as is', is currently considered alpha for many of its features, and there is no clear path to evolve the API to GA
  - 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



Federation v2

- WIP Prototype on GitHub
- Builds on concepts from Federation v1
- Defines specifications of:
  - Resource templates – definition of the resource that will have common settings across clusters
  - Placement – how resources are propagated in each cluster
  - Overrides – how resources are differentiated in each cluster
    - Credential schemes, replica counts, how rolling updates are handled, etc.



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, etcd 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")
```

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`

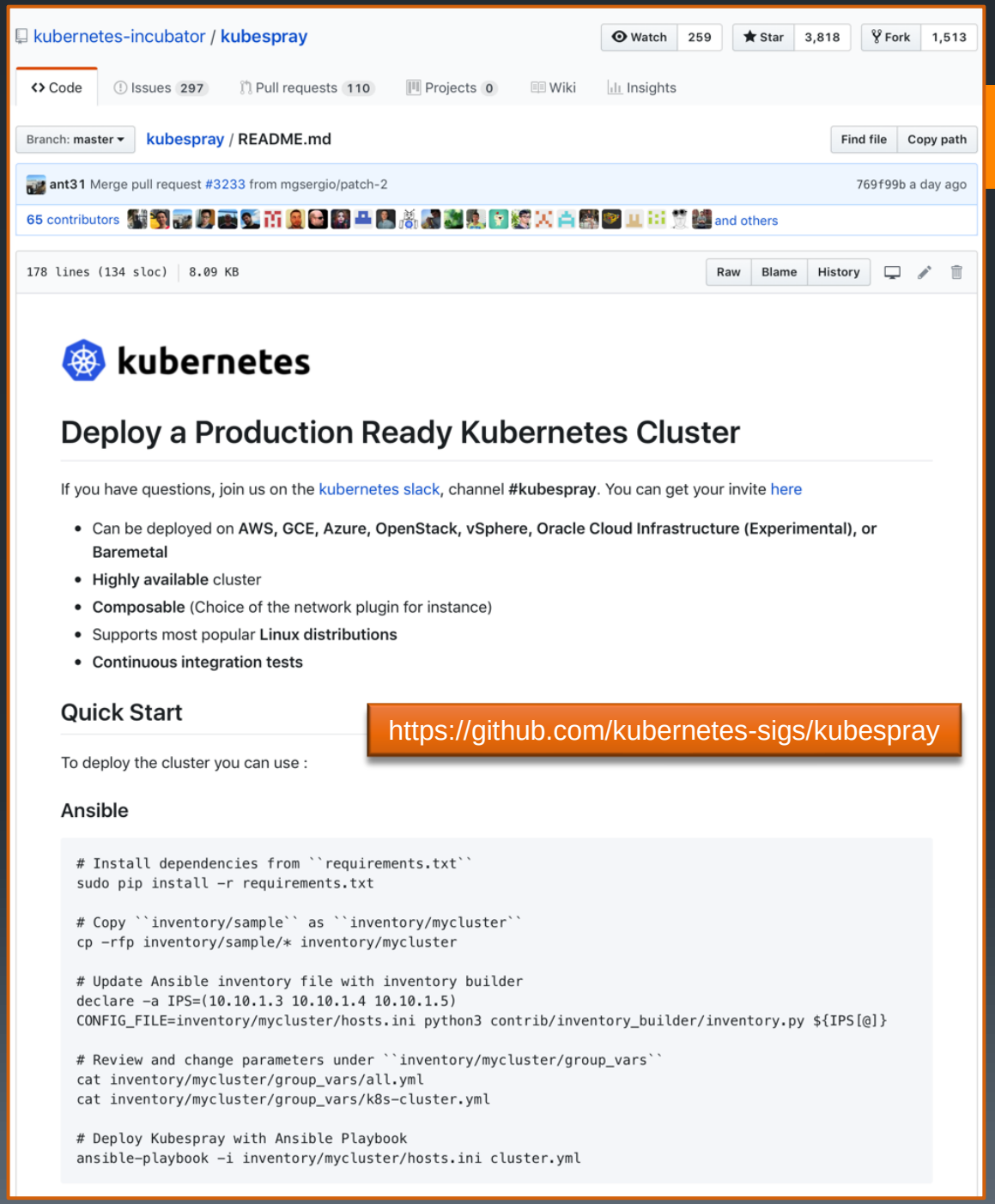
Kubernetes Operations (kops)

- Create & maintain production-grade K8s clusters from the command line
- Currently AWS GA support
  - GCE beta
  - VMware vSphere 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
```

kubespray (formerly kargo)

- Ansible solution for installing K8s
- Supported Linux distributions:
  - Container Linux by CoreOS
  - Debian Jessie, Stretch, Wheezy
  - Ubuntu 16.04, 18.04
  - CentOS/RHEL 7
  - Fedora/CentOS Atomic
  - openSUSE Leap 42.3/Tumbleweed
- Supported components:
  - kubernetes v1.13.5
  - etcd v3.2.24
  - Docker v18.06
  - rkt v1.21.0
  - cri-o v1.11.5
  - Network plugins:
    - calico v3.4.0
    - canal (given calico/flannel versions)
    - cilium v1.3.0
    - contiv v1.2.1
    - flanneld v0.11.0
    - weave v2.5.1
  - Application
    - cephfs-provisioner v2.1.0-k8s1.11
    - cert-manager v0.5.2
    - coredns v1.2.6
    - ingress-nginx v0.21.0



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
- CNCF & K8s SIGs/WGs
  - Best place for project velocity!

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 the beginnings of a K8s 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
  - Examine how Kubelet protects Nodes from pressure conditions based on Pod resource requests and limits (or QoS levels)
- Pod Configuration (aka PodSpec)
  - Review key components of a PodSpec
  - Understand interaction between single-container and multi-container
  - Examine lifecycle configurations and operations

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
  - `syncFrequency:` / `--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)
  - `maxPods:` / `--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`

```
$ sudo netstat -natp | grep kubelet
tcp        0      0 127.0.0.1:43857         0.0.0.0:*               LISTEN      13903/kubelet
tcp        0      0 127.0.0.1:10248         0.0.0.0:*               LISTEN      13903/kubelet
tcp        0      0 192.168.225.145:53694   192.168.225.145:6443    ESTABLISHED 13903/kubelet
tcp        0      0 192.168.225.145:53454   192.168.225.145:6443    ESTABLISHED 13903/kubelet
tcp6       0      0 :::10250                :::*                    LISTEN      13903/kubelet
```

Default kubelet Ports
10248 healthz
10250 kubelet API
10255 kubelet API (ro)

Docker Engine Events

- The `docker events` subcommand allows you to monitor Docker engine events
  - Greatly expanded in Docker Engine v1.10 (new events in orange) (Docker v1.12 additions in green)
- Container events reported:
  - attach, commit, copy, create, destroy (rm), detach, die (container processes exited), exec_create, exec_start, export, health_status, kill, oom (container out of memory), pause, rename, resize, restart, start, stop, top, unpause, update
- Image events reported:
  - delete, import, load, pull, push, tag, untag
- Volume events reported:
  - create, mount, unmount, destroy
- Network events reported:
  - create, connect, disconnect, destroy

```
user@ubuntu:~$ docker container run -it cirros /bin/sh
/ # exit
user@ubuntu:~$
```

```
user@ubuntu:~$ docker events
2017-03-21T21:08:28.210925030-07:00 container create 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (image=cirros, name=distracted_franklin)
2017-03-21T21:08:28.211757033-07:00 container attach 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (image=cirros, name=distracted_franklin)
2017-03-21T21:08:28.227509338-07:00 network connect 9036ad93c6fa19d8a2832ebea3e03cdcb5967bd5bedc59e4ead0ee4a78f3754f (container=0c478f036b3c9c6fb36a4a2a43a4a654
d929f30aee81372fc378bde4b3b29e7d, name=bridge, type=bridge)
2017-03-21T21:08:28.430192357-07:00 container start 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (image=cirros, name=distracted_franklin)
2017-03-21T21:08:28.431313651-07:00 container resize 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (height=26, image=cirros, name=distracted_
franklin, width=110)
2017-03-21T21:08:32.348356755-07:00 container die 0c478f036b3c9c6fb36a4a2a43a4a654d929f30aee81372fc378bde4b3b29e7d (exitCode=0, image=cirros, name=distracted_fr
anklin)
2017-03-21T21:08:32.434930103-07:00 network disconnect 9036ad93c6fa19d8a2832ebea3e03cdcb5967bd5bedc59e4ead0ee4a78f3754f (container=0c478f036b3c9c6fb36a4a2a43a4a
654d929f30aee81372fc378bde4b3b29e7d, name=bridge, type=bridge)
2017-03-21T21:08:34.064378105-07:00 container die 61a4420ffa718e13ea2fd528e43b8268fa86c77d6c28bb26bee932b3e7143004 (exitCode=0, image=ubuntu, name=friendly_goli
ck)
2017-03-21T21:08:34.150178809-07:00 network disconnect 9036ad93c6fa19d8a2832ebea3e03cdcb5967bd5bedc59e4ead0ee4a78f3754f (container=61a4420ffa718e13ea2fd528e43b8
268fa86c77d6c28bb26bee932b3e7143004, name=bridge, type=bridge)
2017-03-21T21:08:34.204698610-07:00 network connect 9036ad93c6fa19d8a2832ebea3e03cdcb5967bd5bedc59e4ead0ee4a78f3754f (container=61a4420ffa718e13ea2fd528e43b8268
fa86c77d6c28bb26bee932b3e7143004, name=bridge, type=bridge)
2017-03-21T21:08:34.407191116-07:00 container start 61a4420ffa718e13ea2fd528e43b8268fa86c77d6c28bb26bee932b3e7143004 (image=ubuntu, name=friendly_golick)
^C
user@ubuntu:~$
```

A Few Kubelet Options

- Over 80+ options being migrated to a config file
- `kubelet --help |& more`

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

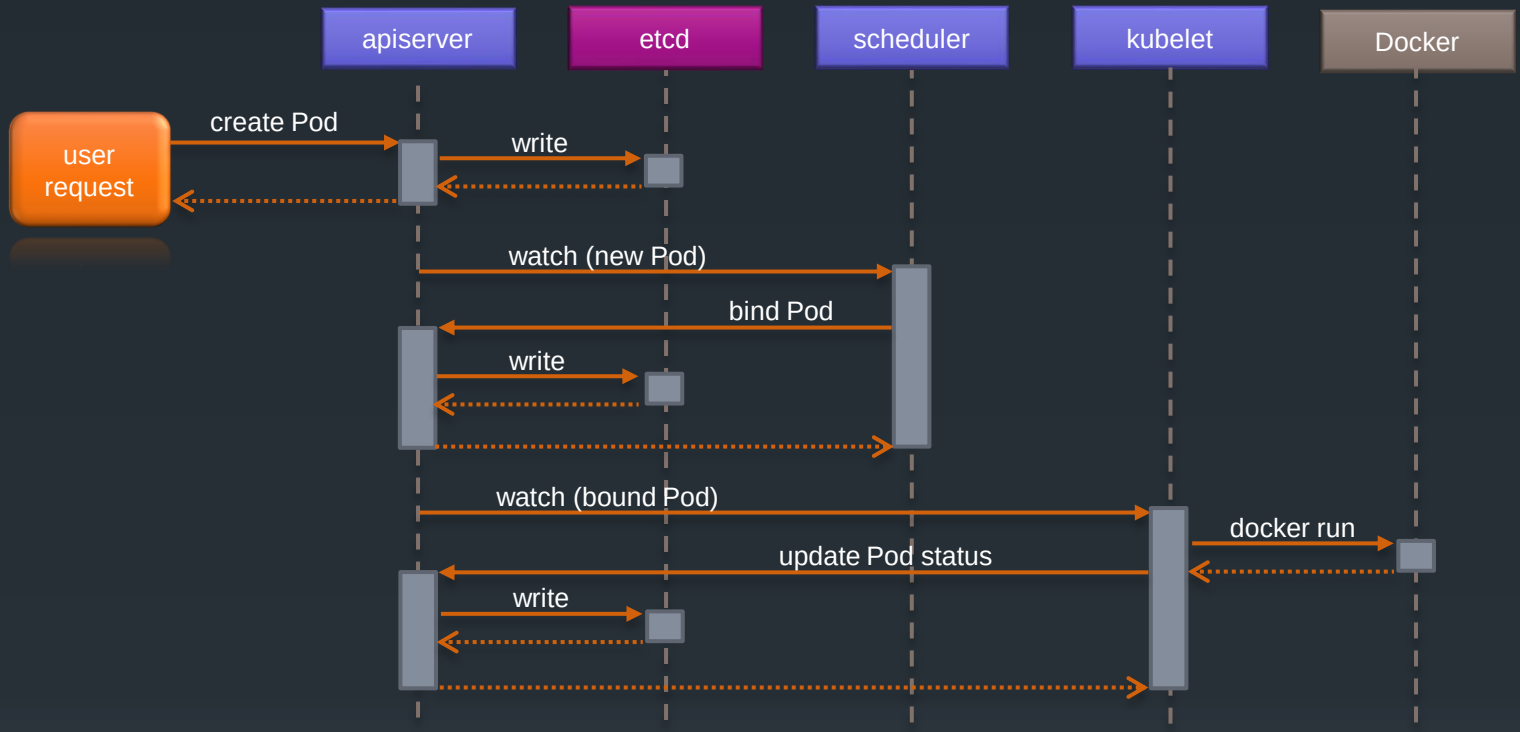
- `address:` / `--address=0.0.0.0`
  - The IP address for the Kubelet to serve on (set to `0.0.0.0` for all interfaces)
- `--allow-privileged[=false]`
  - If true, allow containers to request privileged mode [`default=false`]
- `--cert-dir="/var/run/kubernetes"`
  - The directory where the TLS certs are located (by default `/var/run/kubernetes`)
    - If `--tls-cert-file` and `--tls-private-key-file` are provided, this flag will be ignored
- `clusterDNS:` / `--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
- `clusterDomain:` / `--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
- `--network-plugin`
  - Defines the network plugin to use
- `--cni-bin-dir`
  - Path of the directory in which to search for CNI plugin binaries (default `/opt/cni/bin`)
- `--cni-conf-dir`
  - Path of the directory in which to search for CNI config files (default `/etc/cni/net.d`)
- `--kubeconfig=/etc/kubernetes/kubelet.conf`
  - Path to a kubeconfig file, specifying how to authenticate to API server (the master location is set by the api-servers flag)
- `--config=/var/lib/kubelet/config.yaml`
  - Path to a config file containing configuration options for kubelet

Provide PodSpec to Kubelet

- Other than a PodSpec from the apiserver, there are three ways that a pod manifest can be provided to the Kubelet:
  - File: files found on the `staticPodPath` / `--pod-manifest-path` passed on the command line
    - Rechecked every 20 seconds (configurable with `fileCheckFrequency` / `--file-check-frequency`)
    - Example: $ `sudo kubelet --pod-manifest-path=./kubelet/pod.yaml`
  - HTTP endpoint: HTTP endpoint passed as a parameter on the command line: `staticPodURL` / `--manifest-url`
    - Rechecked every 20 seconds (configurable with `httpCheckFrequency` / `--http-check-frequency`)
    - Example: `$ sudo 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 $HOME/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}]
...
```

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

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, ClusterFirstWithHostNet, Default, or none*
  - 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
    - `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
      - `volumeMounts[]`: volumes which can be mounted

```
$ 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
NAME          READY     STATUS      RESTARTS   AGE
hello-world   0/1       Completed   0          15s
```

Pod Spec Reference:
https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.14/#_v1_podspec
Container Spec Reference:
https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.14/#_v1_container

Init Containers

- Init Containers run before App Containers
  - Can contain setup scripts, registration hooks, etc.
- Init Containers are started in order
  - After network and volumes are initialized
  - Each Init Container must exit successfully before the next is started
  - If an Init Container fails to start or exits with failure, it is retried according to the Pod restartPolicy (if the Pod restartPolicy is Always, Init Containers use OnFailure)
- A Pod is not Ready until all Init Containers have succeeded
- The ports on an Init Container are not aggregated under a service

```
spec:
  initContainers:
  - name: "init-chown-data"
    image: "busybox:latest"
    # 65534 is the nobody user that prometheus uses.
    command: ["chown", "-R", "65534:65534", "/data"]
    volumeMounts:
    - name: storage-volume
      mountPath: /data
      subPath: ""
  containers:
  - name: prometheus-server
    image: "prom/prometheus:latest"
    args:
      - --config.file=/etc/config/prometheus.yml
      - --storage.tsdb.path=/data
      - --web.console.libraries=/etc/prometheus/console_libraries
      - --web.console.templates=/etc/prometheus/consoles
      - --web.enable-lifecycle
    ports:
      - containerPort: 9090
    volumeMounts:
      - name: storage-volume
        mountPath: /data
  volumes:
    - name: storage-volume
      persistentVolumeClaim:
...
```

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

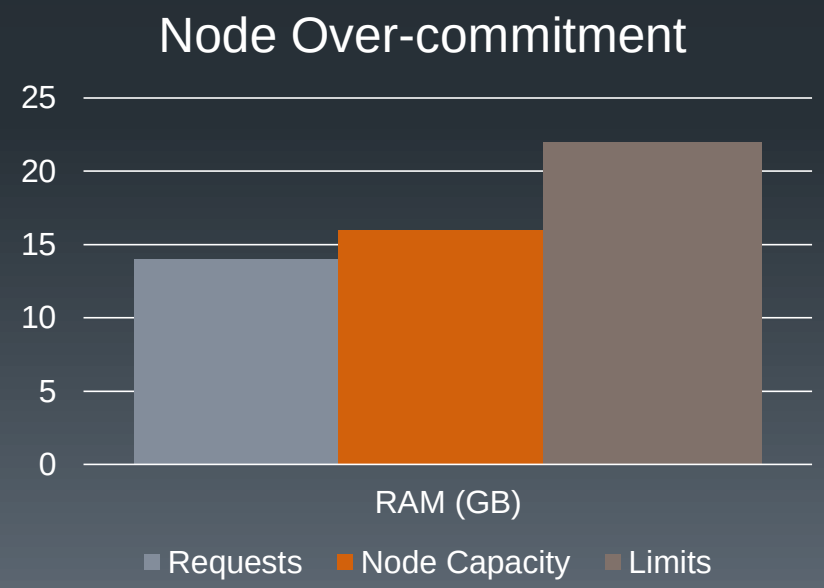
```
apiVersion: v1
kind: Pod
metadata:
  name: blogsite
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "0.25"
      limits:
        memory: "128Mi"
        cpu: "0.5"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "0.25"
      limits:
        memory: "128Mi"
        cpu: "0.5"
```

Ephemeral Storage

- Kubelet's root directory (`/var/lib/kubelet` by default) is also shared and consumed by Pods via `emptyDir` volumes
  - This partition is ephemeral and applications cannot expect any performance SLAs (Disk IOPS)
- Requests and limits for local ephemeral storage
  - Each Container of a Pod can specify:
    - `spec.containers[].resources.limits.ephemeral-storage`
    - `spec.containers[].resources.requests.ephemeral-storage`
- The scheduler ensures that the sum of the resource requests of the scheduled Containers is less than the capacity of the node
- A pod will be evicted if:
  - A container's writable layer and logs usage exceeds its storage limit
  - The sum of the local ephemeral storage usage from all containers plus the pod's `emptyDir` volumes exceeds the limit

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



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:
  - `evictionSoft` / `--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
  - `evictionSoftGracePeriod` / `--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
  - `evictionHard` / `--eviction-hard` A set of eviction thresholds (e.g. memory.available<1Gi) that if met would trigger a pod eviction. (default "memory.available<100Mi")
  - `evictionMaxPodGracePeriod` / `--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
  - `evictionMinimumReclaim` / `--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
  - `evictionPressureTransitionPeriod` / `--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

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
evictionHard:
  imagefs.available: 15%
  memory.available: 100Mi
  nodefs.available: 10%
  nodefs.inodesFree: 5%
```

Pod Phases

- Phases
  - Describe the macro states in the lifecycle of a Kubernetes resource
  - Simple, high-level summary of where the Pod is in its lifecycle, 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
  - Completed
    - The pod has run to completion; there's nothing to keep it running
  - CrashLoopBackOff
    - One of the containers in the pod has exited unexpectedly, likely with a non-zero error code even after restarting due to restart policy

Status	Meaning
`Init:x/y`	The Pod has x Init Containers, and y have completed so far
`Init:Error`	An Init Container has failed to execute
`Init:CrashLoopBackOff`	An Init Container has failed repeatedly
`Pending`	The Pod has not yet begun executing Init Containers
`PodInitializing` or `Running`	The Pod has already finished executing Init Containers

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!)
  - It is up to the hook implementation to handle this correctly
- `postStart` – called immediately after a container is created
  - If the handler fails, the container is terminated and restarted according to its restart policy
  - Other management of the container blocks until the action completes
- `preStop` – called immediately before a container is terminated
  - Blocking, so it must complete before the call to delete the container can be sent
    - If the hook hangs during execution, the Pod phase stays in a Terminating state and is killed after `terminationGracePeriodSeconds` of pod ends
    - Otherwise, the container is terminated 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)
- Two types of hook handlers:
  - Exec - Executes a command inside the cgroups and namespaces of the container
  - HTTP - request against a specific endpoint on the container

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-hook
spec:
  replicas: 3
  selector:
    matchLabels:
      app: apache-hook
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"]
```

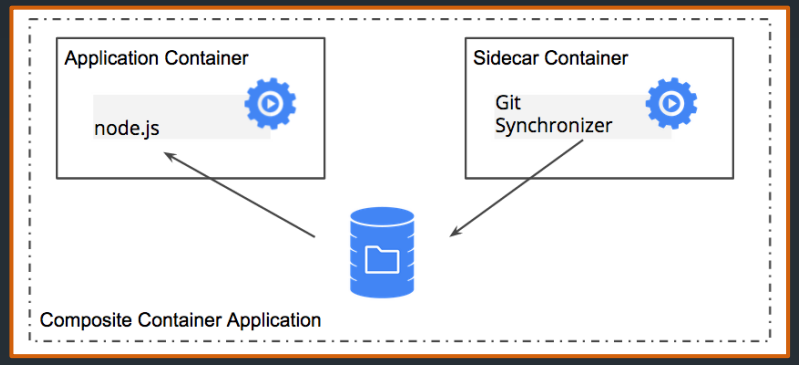
Pause

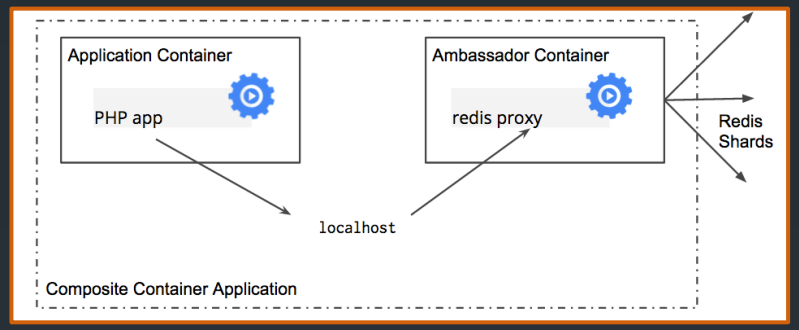
- The Pause container is often referred to as the pod infrastructure container and is used to set up and hold the networking namespace and resource limits for each pod

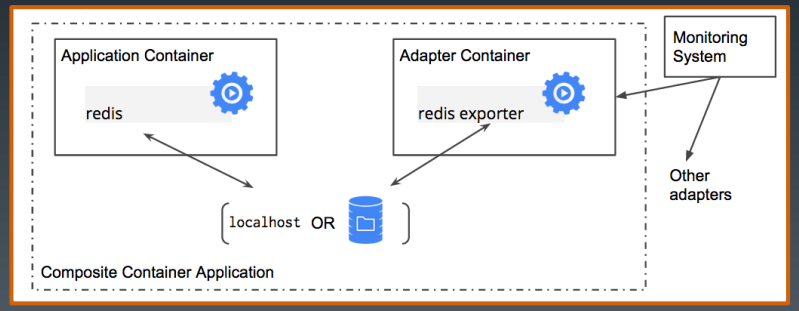
```
$ cat long.yaml

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...
```

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
- Benefits
  - Containers are units of:
    - Resource accounting and allocation – "main" container given priority over resources, "helper" container can be given strict restraints
    - Reuse – single version of "helper" used among many apps
    - Deployment – roll out/back independently
    - Failure boundaries – "main" container can continue even if "helper" fails
  - Divide responsibility for development between two separate programming teams



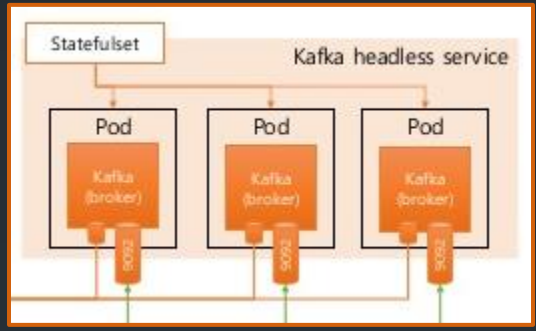


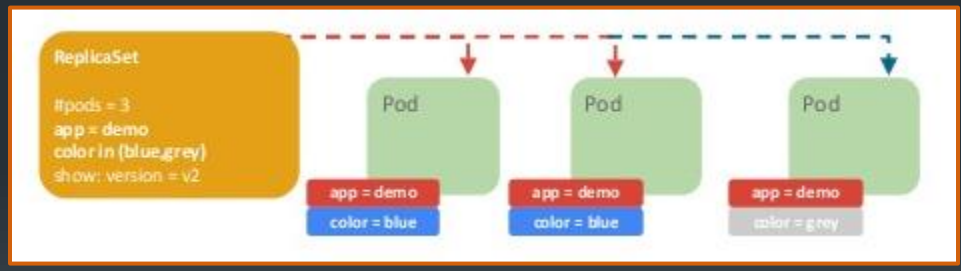


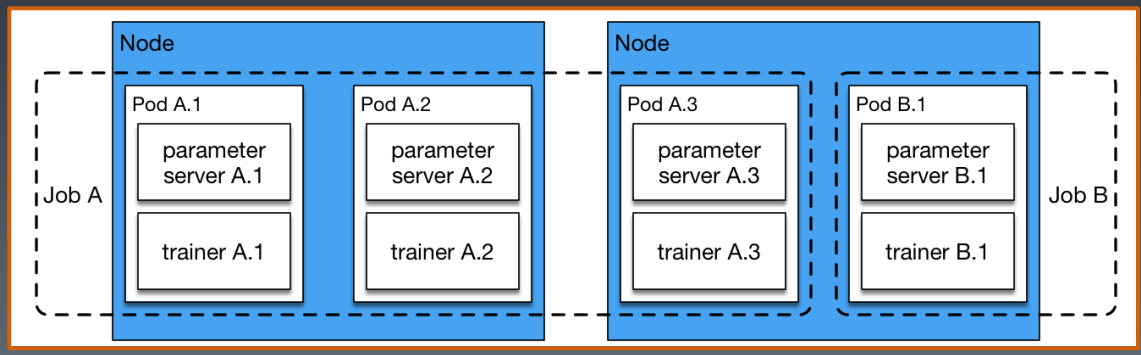
- Brendan Burns, Distinguished Engineer at Microsoft and former Software Engineer at Google
https://research.google.com/pubs/pub45406.html

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
- Details from Pod Specs are used to determine QoS levels and order of Eviction when pressure condidtions occur
- A Pod spec can be used by itself (bare) or embedded in other resource types like Controllers

Lab 2

- Testing configs with a stand alone Kubelet

Day 2

3. Scheduling
4. Services & kube-proxy

3. Scheduling

Objectives

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

The scheduling process

- Node: a physical or virtual machine running Kubernetes, onto which pods can be scheduled.

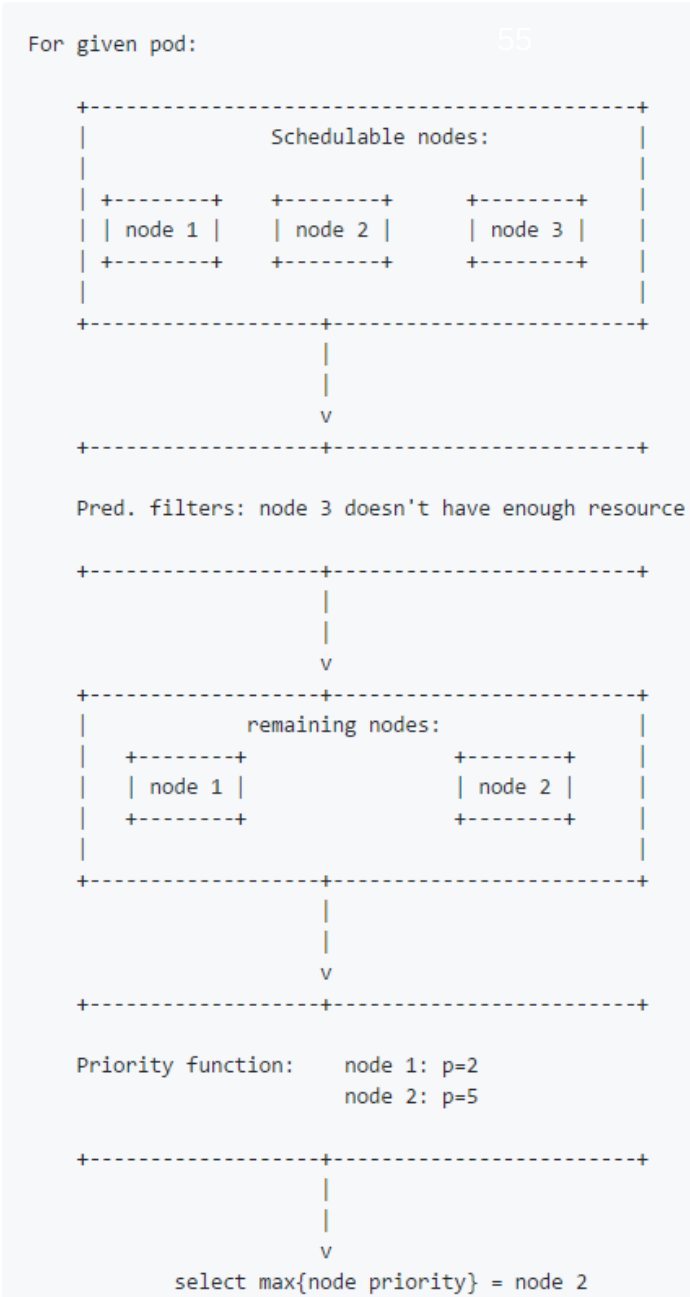
- Pod: collocated group of application containers with shared volumes
  - Smallest deployable units that can be created, scheduled, and managed with Kubernetes
  - Pods can be created individually, but it's recommended that you use a Controller (RC, RS, Deployment) even if creating a single pod

- 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
    - Ex, 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
    - EX, 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

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



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 *v1.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 }

              ...

        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")
        startPriorityEvalTime := time.Now()
        // When only one node after predicate, just use it.
        if len(filteredNodes) == 1 {
              metrics.SchedulingAlgorithmPriorityEvaluationDuration.Observe(metrics.SinceInMicroseconds(startPriorityEvalTime))
              return filteredNodes[0].Name, nil
        }

        metaPrioritiesInterface := g.priorityMetaProducer(pod, g.cachedNodeInfoMap)
        priorityList, err := PrioritizeNodes(pod, g.cachedNodeInfoMap, metaPrioritiesInterface, g.prioritizers, filteredNodes, g.extenders)
        if err != nil {
              return "", err
        }
        metrics.SchedulingAlgorithmPriorityEvaluationDuration.Observe(metrics.SinceInMicroseconds(startPriorityEvalTime))

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

From (generic_scheduler.go)
https://github.com/kubernetes/kubernetes/blob/release-1.10/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/pkg/scheduler/algorithm/predicates/predicates.go

```
            {
            "kind" : "Policy",
            "apiVersion" : "v1",
            "predicates" : [
                    {"name" : "PodFitsPorts", "order": 1},
                    {"name" : "PodFitsResources", "order": 2},
                    {"name" : "NoDiskConflict", "order": 5},
                    {"name" : "NoVolumeZoneConflict", "order": 4},
                    {"name" : "MatchNodeSelector", "order": 6},
                    {"name" : "HostName", "order": 3}
                    ],
            "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/release-1.10/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: Evaluates 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
- NewMaxPDVolumeCountPredicate: Ensures that the number of attached disk volumes does not exceed a maximum value
  - AzureDiskVolumeFilter is a VolumeFilter for filtering Azure Disk Volumes
  - GCEPDVolumeFilter is a VolumeFilter for filtering GCE PersistentDisk Volumes
    - Default, 16, which is the maximum GCE allows
  - EBSVolumeFilter is a VolumeFilter for filtering AWS ElasticBlockStore Volumes
    - Default, 39, since Amazon recommends a maximum of 40 with one of those 40 reserved for the root volume
  - The maximum value can be controlled by setting the `KUBE_MAX_PD_VOLS` environment variable
- NewVolumeZonePredicate: Evaluates if the volumes a pod requests are available on the node, given the Zone restrictions
  - Currently only supported with PersistentVolumeClaims, and looks to the labels only on the bound PersistentVolume
- 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
- PodFitsHost: Filter out all nodes except the one specified in the PodSpec's `NodeName` field
- NewNodeLabelPredicate: evaluates whether a pod can fit based on the Node labels which match a filter that it requests
- NewServiceAffinityPredicate: matches Nodes in such a way to force a ServiceAffinity
  - If the first pod of a service was scheduled to a node with label "`region=foo`", all the other subsequent pods belong to the same service will be schedule on Nodes with the same "`region=foo`" label

Filtering Nodes 2

- PodFitsHostPorts: Check if any HostPort required by the Pod is already occupied on the node
- NewPodAffinityPredicate: Checks if a pod can be scheduled on the specified node with pod affinity/anti-affinity configuration
- PodMatchNodeSelector: Check if the labels of the node match the labels specified in the Pod's nodeSelector field
- CheckNodeUnschedulablePredicate: Checks if a pod can be scheduled on a node with Unschedulable spec
- 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
- CheckNodePIDPressurePredicate: Checks if a pod can be scheduled on a node reporting pid pressure condition
- CheckNodeConditionPredicate: checks if a pod can be scheduled on a node reporting out of disk, network unavailable and not ready condition
  - Only node conditions are accounted in this predicate

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/tree/master/pkg/scheduler/algorithm/priorities

```
  {
  "kind" : "Policy",
  "apiVersion" : "v1",
  "predicates" : [
          {"name" : "PodFitsPorts", "order": 1},
          {"name" : "PodFitsResources", "order": 2},
          {"name" : "NoDiskConflict", "order": 5},
          {"name" : "NoVolumeZoneConflict", "order": 4},
          {"name" : "MatchNodeSelector", "order": 6},
          {"name" : "HostName", "order": 3}
          ],
  "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/release-1.10/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/MostRequestedPriority: The node is prioritized based on the fraction of the node's resources that are free (used)
    - Least: cpu((capacity-sum(requested))*10/capacity) + memory((capacity-sum(requested))*10/capacity)/2
    - Most: (cpu(10 * sum(requested) / capacity) + memory(10 * sum(requested) / capacity)) / 2
      - CPU and memory are equally weighted
    - The node with the highest free (used) fraction is the most preferred
    - Note that this priority function has the effect of spreading Pods across the nodes (or packing them) 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
    - MUST be used together with LeastRequestedPriority
  - SelectorSpreadPriority: Spread Pods by minimizing the number of Pods belonging to the same Service, Replication Controller, StatefulSets,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
  - ImageLocalityPriority: Nodes are prioritized based on locality of images requested by a pod
    - If none of the images are present, the node will be given the lowest priority
    - If some of the images are present on a node, the larger their sizes' sum, the higher the node's priority
  - NewNodeLabelPriority: Prioritizes nodes that have the specified label
  - InterpodAffinity & NodeAffinity: compute a sum if the corresponding PodAffinity is satisfied for a node

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/release-1.10/pkg/scheduler/core/generic_scheduler.go

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

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 but still work

```
{
"kind" : "Policy",
"apiVersion" : "v1",
"predicates" : [
        {"name" : "PodFitsPorts", "order": 1},
        {"name" : "PodFitsResources", "order": 2},
        {"name" : "NoDiskConflict", "order": 5},
        {"name" : "NoVolumeZoneConflict", "order": 4},
        {"name" : "MatchNodeSelector", "order": 6},
        {"name" : "HostName", "order": 3}
        ],
"priorities" : [
        {"name" : "LeastRequestedPriority", "weight" : 1},
        {"name" : "BalancedResourceAllocation", "weight" : 1},
        {"name" : "ServiceSpreadingPriority", "weight" : 1},
        {"name" : "EqualPriority", "weight" : 1}
        ]
}
```

Custom Schedulers

- 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
    - Can be written in any language and can be as simple or complex as you need
      - One example in the K8s docs is a bash script that assigns nodes randomly!
  - A custom scheduler or the K8s scheduler with a custom config can run 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=my-scheduler`
    - 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`

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  schedulerName: my-scheduler
  containers:
  - name: nginx
    image: nginx:1.10
```

Assigning Pods to Nodes

- `nodeName` – simplest form of node selection constraint using the node's name
- `nodeSelector` – specifies a map of key-value pairs
  - For a pod to be eligible to run on a node, the node *must have* each of the indicated key-value pairs as labels
    - The most common usage is one key-value pair
- Built-in node labels – in addition to user-created labels, nodes come pre-populated with a standard set of labels:
  - `kubernetes.io/hostname`
  - `failure-domain.beta.kubernetes.io/zone`
  - `failure-domain.beta.kubernetes.io/region`
  - `beta.kubernetes.io/instance-type`
  - `kubernetes.io/os`
  - `kubernetes.io/arch`

```
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: database
  labels:
    app: database
spec:
  serviceName: database-svc
  replicas: 3
  podManagementPolicy: OrderedReady
  revisionHistoryLimit: 5
  updateStrategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: database
  template:
    metadata:
      labels:
        app: database
    spec:
      nodeSelector:
        disk-type: local-storage
...
```

Node 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
- Two types:
  - `requiredDuringSchedulingIgnoredDuringExecution`: Specifies rules that *must* be met for a pod to schedule
    - If no node 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 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*
- 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
...
```

Pod Affinity & Anti-Affinity

- Inter-pod affinity and anti-affinity – specify rules about how pods should be placed relative to one another
  - Based on labels on pods that are already running on nodes
- Spread or pack pods within a service or relative to pods in other services
- Requires substantial processing which can slow down scheduling in large clusters significantly
  - Not recommended in clusters larger than several hundred nodes
- For `requiredDuringSchedulingIgnoredDuringExecution` pod affinity and anti-affinity, `topologyKey` *must not* be empty
- For `preferredDuringSchedulingIgnoredDuringExecution` pod anti-affinity, an empty `topologyKey` is interpreted as "all topologies" (hostname, zone, and region)

```
...
affinity:
  podAntiAffinity:
    ## Pods will not be scheduled on the same node as
    ##   another pod (by hostname)
    ##
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: app
          operator: In
          values:
          - cassandra
        topologyKey: kubernetes.io/hostname
    ## Pods will not be scheduled in the same AZ as
    ##   another pod unless all AZs already have pods
    ##
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - cassandra
        topologyKey: failure-domain.beta.kubernetes.io/zone
...
```

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 optional 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

Administrative Commands

- The K8s API and kubectl offer several control features that affect scheduling
  - Also 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)

  - The unschedulable field of a node is not respected by the DaemonSet controller
    - `--ignore-daemonsets` is required or the `drain` command will fail

```
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
- Taints and Tolerations as well as Pod and Node affinities and anti-affinities can influence scheduling
- Use policy files to rebalance 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

4. Services & kube-proxy

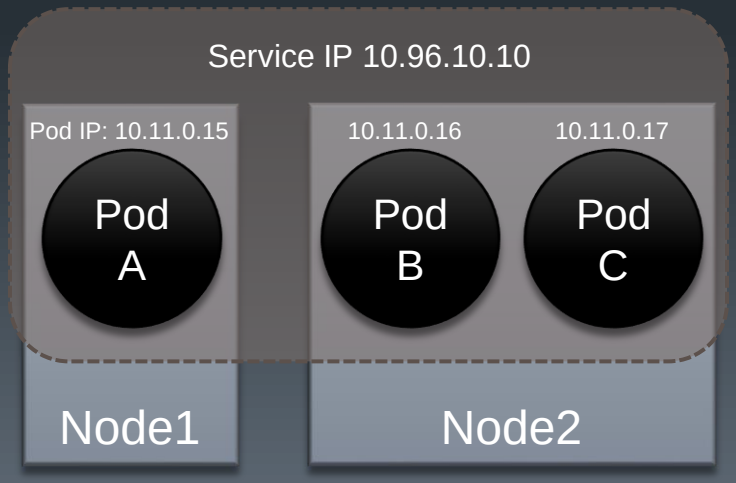
Objectives

- Explore the nature of Kubernetes services
  - ClusterIP
  - NodePort
  - LoadBalancer
- Understand the roll of kube-proxy and load balancers
- Examine IPTables concepts: tables, chains, rules and targets
- Describe the IPTables manipulations made by the proxy
- Discuss alternatives to Kubernetes' service mesh

Services

- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them
  - The set of Pods targeted are identified by a Selector
  - Although each Pod gets its own IP address, Pod IP addresses cannot be relied upon to be stable over time
    - Controllers create and destroy Pods dynamically
      - e.g. when scaling up or down or when doing rolling updates
- Services can be exposed in different ways by specifying a type in the ServiceSpec:
  - ClusterIP – [Default] exposes the Service on an internal IP in the cluster
  - NodePort – exposes the Service on the same port of each selected Node in the cluster using NAT
  - LoadBalancer – Creates an external load balancer with a cloud provider and assigns an external IP to the Service
  - ExternalName – xternal reference that the DNS service will return as a CNAME record for this service

    No proxying will be involved



Service Config spec

- `type` - type of exposed service
  - `ClusterIP` – [Default] uses a cluster based virtual IP (available only inside the cluster)
  - `NodePort` – exposes the service on the same port on each Node (for external access)
  - `LoadBalancer` – uses an external load balancer, with a public IP and port (AWS ELB, etc.)
  - `ExternalName` – forwards traffic to the specified external DNS name
- `ports` - The list of ports that are exposed by this service
  - Need not be the same as the port of the actual service
- `targetPort` – The port exposed by the backend Pods
  - By default the targetPort will be set to the same value as the port field
  - Can be a string, referring to the name of a port in the backend Pods
- `selector` - This service will route traffic to pods having labels matching this selector
  - If empty, all pods are selected (!)
- `clusterIP` - IP is usually assigned by the master and is the IP address of the service
  - If specified, it will be allocated to the service if it is unused or else creation of the service will fail
  - Valid values are `None` (headless service), empty string (""), or a valid IP address
  - Cannot be updated
- `externalIPs` - externalIPs is a list of IP addresses for which nodes in the cluster will also accept traffic for this service
  - These IPs are not managed by Kubernetes
  - The user is responsible for ensuring that traffic arrives at a node with this IP
  - A common example is external load-balancers that are not part of the Kubernetes system
- `sessionAffinity` - Supports `ClientIP` and `None` (default)
  - Used to maintain session affinity
  - In IPTables mode, relies on the "recent" module
- `loadBalancerIP` - Only applies to Service type: LoadBalancer
  - LoadBalancer will get created with the IP specified in this field
  - This feature depends on whether the underlying cloud-provider supports specifying the loadBalancerIP when a load balancer is created
  - This field will be ignored if the cloud-provider does not support the feature

```
apiVersion: v1
kind: Service
metadata:
  name: testweb
  labels:
    name: testweb
spec:
  type: ClusterIP
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 600
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 80
  selector:
    name: testweb
```

```
$ kubectl get endpoints
NAME         ENDPOINTS                AGE
kubernetes   192.168.131.134:6443     5d
testweb      172.17.0.9:80            1h
```

https://kubernetes.io/docs/api-reference/v1.14/#_v1_service

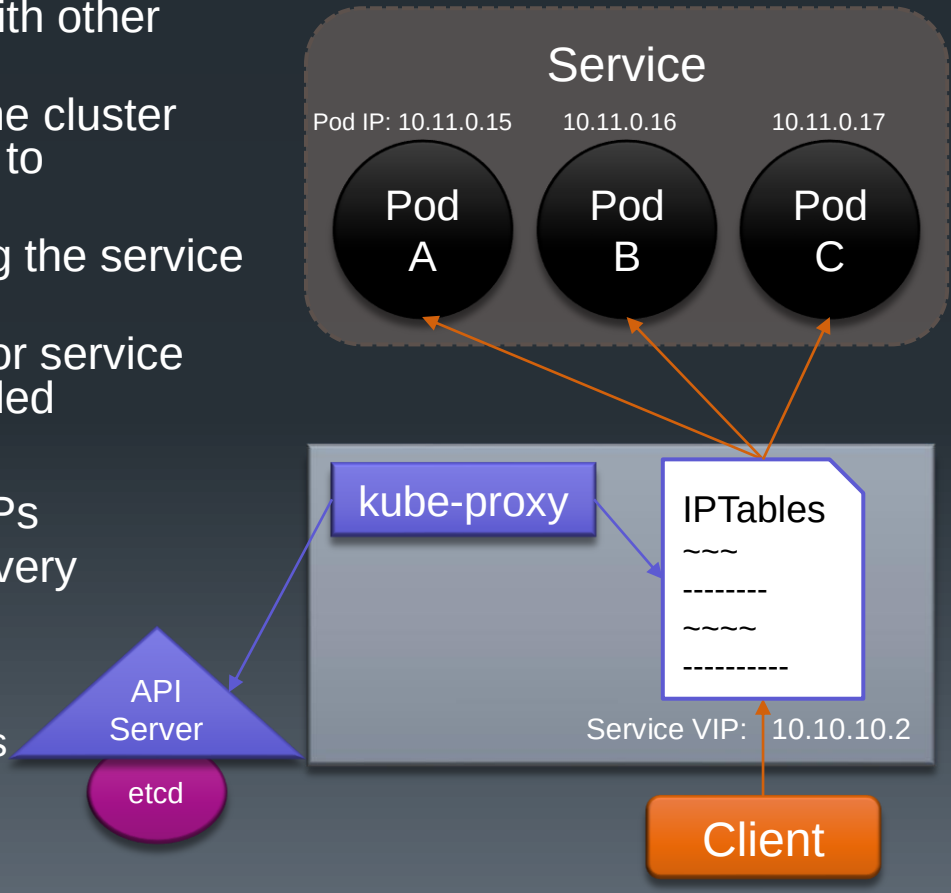
Node Port Details

- The Kubernetes master allocates a port from a flag-configured range
  - Default: 30000-32767
- Each Node will proxy that port (the same port number on every Node) into your Service
  - That port will be reported in your Service's spec.ports[*].nodePort field
- If you want a specific port number, you can specify a value in the nodePort field
  - The system will allocate you that port or else the API transaction will fail
  - The value you specify must be in the configured range for node ports
  - The port may not conflict with existing ports
- This gives developers the freedom to set up their own load balancers, to configure cloud environments that are not fully supported by Kubernetes
- Node Port Services also have all of the features of Cluster IP
  - Visible as both:
    - NodeIP: spec.ports[*].nodePort
    - ClusterIP: spec.clusterIp:spec.ports[*].port

```
user@ubuntu:~/svc$ cat nodeport.yaml

apiVersion: v1
kind: Service
metadata:
   name: testweb-np
spec:
   type: NodePort
   ports:
     - port: 80
       protocol: TCP
  selector:
     run: testweb

user@ubuntu:~/svc$ kubectl create –f nodeport.yaml

user@ubuntu:~/svc$ kubectl get service
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   10.96.0.1        <none>        443/TCP        1d
testweb-np   10.111.101.166   <nodes>       80:30143/TCP   32s

user@ubuntu:~/svc$ kubectl describe service testweb-np
Name:                     testweb-np
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 run=testweb
Type:                     NodePort
IP:                       10.111.101.166
Port:                     <unset>      80/TCP
NodePort:                 <unset>      30143/TCP
Endpoints:                10.32.0.4:80,10.32.0.5:80,10.32.0.6:80
Session Affinity:         None
Events:                   <none>
```

Kube-Proxy

- Microservices
  - Most sophisticated K8s applications are microservice oriented
    - Networks of small services
  - Each logical service is implemented by a set of pod replicas
- Routing Mesh
  - Pods frequently need to communicate with other services
  - The Kube-Proxy runs on each node in the cluster proiding a routing mesh to map services to implementation pods
  - Kube-Proxy identifies pods implementing the service via a selector
  - All Kube-Proxies watch the API Server for service changes, updating their iptables as needed
- Service Discovery
  - Clients need a way to discover service IPs
  - Kubernetes DNS supports service discovery
    - Kube-DNS acts like a normal DNS service but knows the addresses of all services
    - DNS A records (address records) provide resolution of domain names to IP addresses
    - DNS SRV records (service records) can be used to provide IP addresses and ports



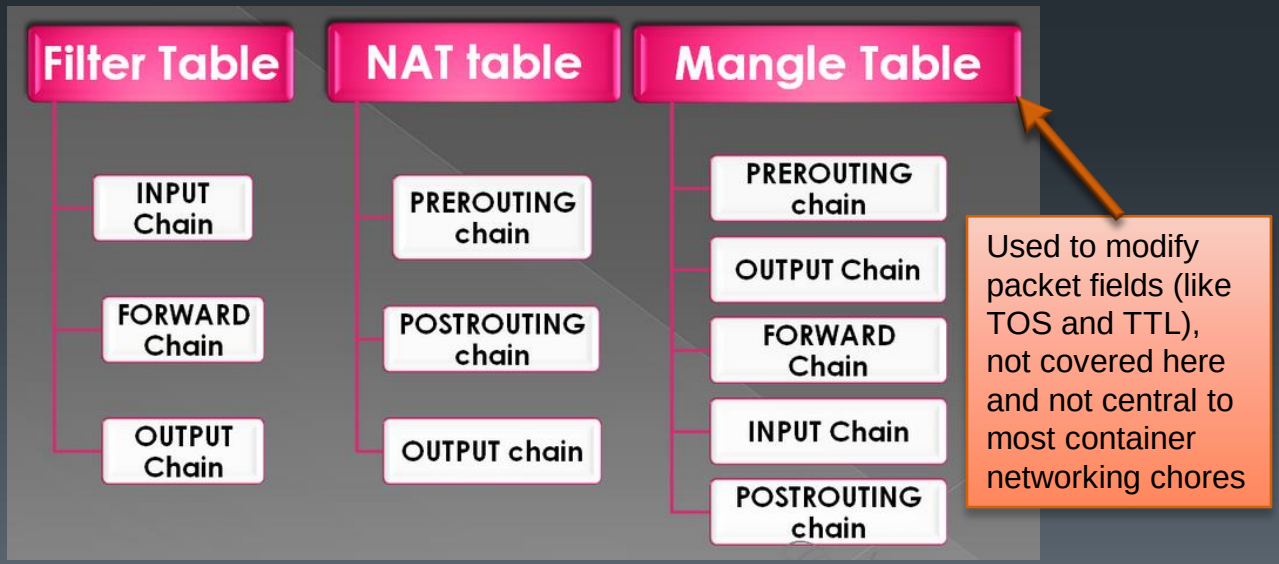
Proxy Modes

- The kube-proxy can be configured to operate in one of three proxy modes:
  - Userspace (older)
    - All local traffic destined for clusterIPs or nodePorts is delivered to the proxy which then forwards the traffic on to the destination
    - Much like the docker-proxy loopback, this places a user mode process in the data path and doubles the connection count
    - Originally this was the only solution and it was the default until k8s 1.5
  - IPTables (default)
    - In IPTables mode Kube-Proxy creates iptables rules for every service
      - Produces a virtual endpoint that routes traffic to one of the service's target pods
        - Typically a ClusterIP, aka Virtual IP (VIP)
      - IPTables loadbalancing randomly selects the target pod to forward to
    - Every sync kube-proxy flushes and reloads Kubernetes chains (every 30 seconds, Service & Pod events)
  - IPVS (experimental)
- If unset the kube proxy uses the best-available proxy which is currently iptables
  - If the system's kernel or iptables versions are insufficient, the userspace proxy is the fall back
- The `--proxy-mode` switch can be used to set the proxy mode

iptables

http://www.iptables.info/en/

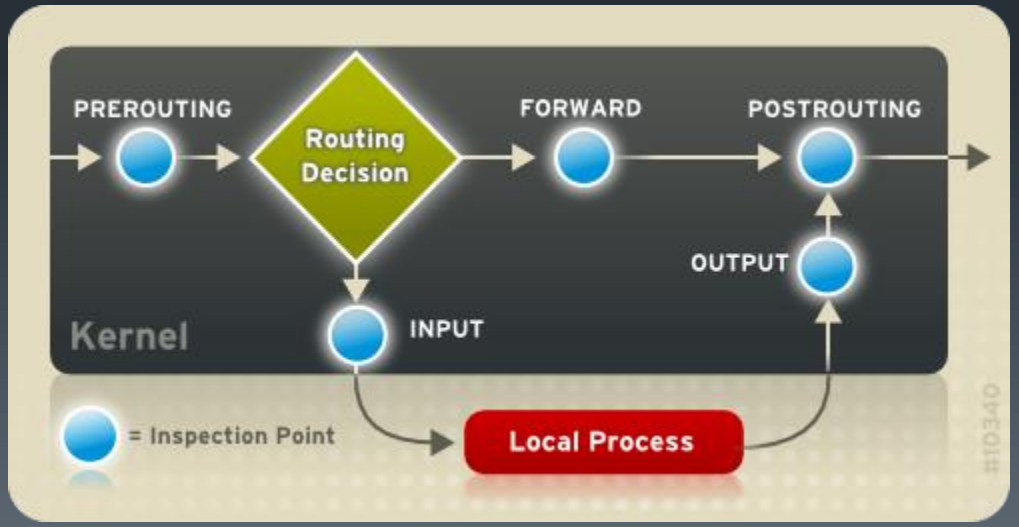
- iptables is a program that configures rules housed in tables used by the Linux kernel when processing network packets
- The term iptables is also commonly used to refer to the kernel packet processing implementation, Netfilter
  - Different Netfilter kernel modules are currently used for different protocols
    - iptables applies to IPv4
    - ip6tables to IPv6
    - arptables to ARP
    - ebtables to Ethernet frames
  - x_tables is the kernel module providing the shared code used by all four modules
  - The successor of iptables is nftables
    - Backward compatible with iptables
    - Merged into Linux kernel version 3.13 (19 January 2014)
- Running the iptables command requires elevated privileges (root)



- Definitions
  - Tables house collections of related packet processing rules
    - **Filter** table
    - **Nat** table
    - **Mangle** table
  - Rules have a matching part and an action part (target)
    - Rules are organized into chains
  - Targets describe what to do with a packet when the rule matches
  - Chains are sequences of rules applied at different phases of packet processing
    - **Prerouting**
    - **Input**
    - **Forward**
    - **Output**
    - **Postrouting**

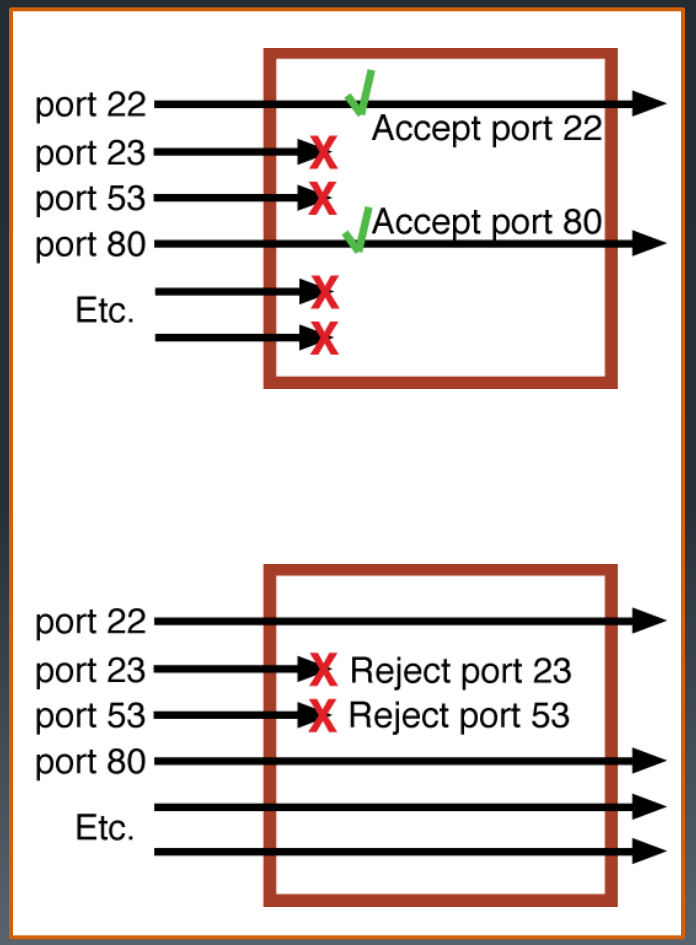
Chains

- *xtables* allows the definition of tables containing chains of rules for the treatment of packets
  - Each table is associated with a different kind of packet processing
  - Packets are processed by sequentially traversing the rules in chains
  - A rule in a chain can cause a goto or jump to another chain, and this can be repeated to whatever level of nesting is desired
    - `-j` Jump – like a "call", i.e. the point that was jumped from is remembered and returned to
    - `-g` Goto – like, well a "goto", i.e. the point of departure is forgotten, returns will resume at the last jump point
  - Every network packet arriving at or leaving from the computer traverses at least one chain
- Packet flow paths
  - Packets start at a given box and will flow along a certain path, depending on the circumstances
  - The origin of the packet determines which chain it traverses initially
  - There are five predefined chains (mapping to the five available Netfilter hooks), though a table need not have all chains
  - Predefined chains have a policy, for example DROP, which is applied to the packet if it reaches the end of the chain
  - The system administrator can create as many chains as desired
    - These chains have no policy; if a packet reaches the end of a user defined chain it is returned to the prior table
  - A chain may be empty
- Chains
  - `PREROUTING`: Packets enter this chain before a routing decision is made
  - `INPUT`: Packet is going to be locally delivered
  - `FORWARD`: All packets that have been routed and were not for local delivery will traverse this chain
    - Inbound traffic bound for containers takes this path
  - `OUTPUT`: Packets sent from the machine itself visit this chain
  - `POSTROUTING`: Routing decision has been made, packets enter this chain just before handing them off to the hardware



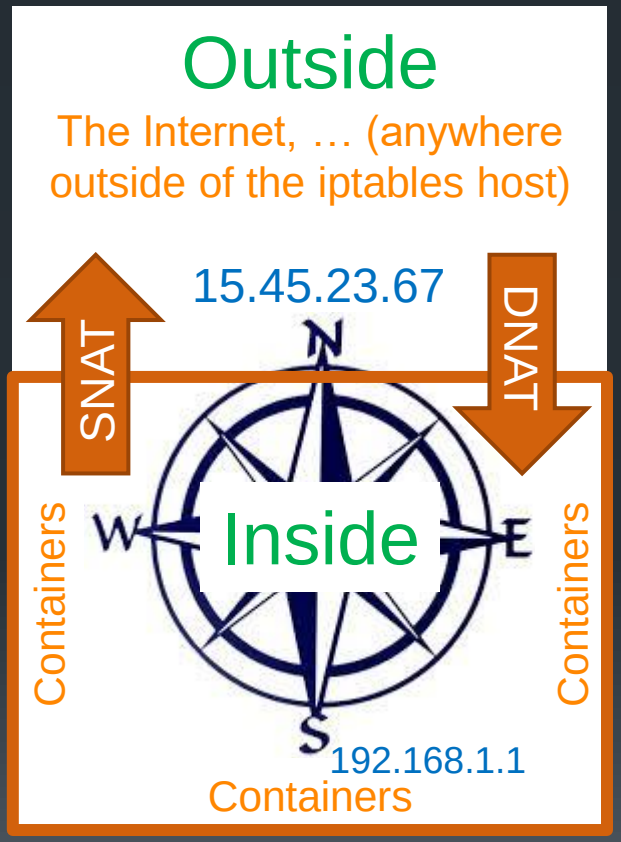
Filter table

- Used to allow or disallow packet transmission
  - Packets can be matched and filtered in many ways
  - Provides basic firewalling
  - The default table
    - No need for the `-t` switch, but if you must: `-t filter`
- Most targets are legal in this table:
  - ACCEPT
    - `iptables -A INPUT -p tcp --dport 22 -j ACCEPT`
    - Packets matching this rule are transmitted on immediately and will not continue traversing the current chain or any other chains in the same table
    - A packet accepted in one chain might still travel through chains within other tables which may drop it
  - DROP
    - `iptables -A INPUT -p tcp --dport 22 -j DROP`
      - Change source of outbound traffic to 15.45.23.67
    - Discards packets with no further processing
    - This action may leave sockets pending response on hosts
    - If a packet is DROPed in a sub chain it will not be processed further in any of the main chains or in any other table
    - The target will not send any kind of information in either direction
  - REJECT
    - `iptables -A FORWARD -p tcp --dport 22 -j REJECT --reject-with tcp-reset`
    - The same as the DROP target but sends back an error message to the source host
    - Only valid in the INPUT, FORWARD and OUTPUT chains or their sub chains
    - All chains that use the REJECT target may only be called by the INPUT, FORWARD, and OUTPUT chains
    - Contrary to popular belief, DROP does not give better security than REJECT, rather it inconveniences legitimate users without additional protection from malicious ones
      - http://www.chiark.greenend.org.uk/~peterb/network/drop-vs-reject



NAT Table

- Used to translate a packet's source IP field or destination IP field
- Compass Model
  - Traffic from inside to outside: S -> N SNAT/MASQ
  - Traffic from outside to inside: N -> S DNAT
  - Traffic between containers: E -> W (or if you prefer W -> E)
- There are 4 NAT Targets:
  - DNAT – Destination NAT : North to South
    - `iptables -t nat -A PREROUTING -p tcp -d 15.45.23.67 --dport 80 -j DNAT --to-destination 192.168.1.1:8089`
      - Change destination of inbound traffic hitting 15.45.23.67:80 to 192.168.1.1:8089
    - Changes the destination address (and optionally port) of the packet and reroutes it
    - Allows an interface with a public IP to redirect traffic to an inside host
      - To a machine on a physical (e.g. DMZ) network behind the firewall
      - To a container on a virtual (e.g. Docker) network behind the firewall
  - SNAT – Source NAT : South to North
    - `iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to-source 15.45.23.67`
      - Change source of outbound traffic to 15.45.23.67
    - Changes the source address of packets from inside destined for outside using a defined outside IP address
    - Makes it possible to connect to the Internet from inside
  - MASQUERADE – South to North
    - `iptables -t nat -A POSTROUTING -p tcp -o eth0 -j MASQUERADE`
    - Like SNAT but translates based on an interface not a preconfigured IP
    - The IP of the outside interface is used allowing this to work when the interface receives its address dynamically (DHCP, etc.)
  - REDIRECT
    - `iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080`
    - Used to redirect packets to the machine itself
    - Only valid within the PREROUTING and OUTPUT chains of the nat table or within user-defined chains that are only called from those chains
- Only the first packet in a stream will hit this table
  - After initial inspection all other packets sent over the connection will automatically have the same action taken on them



```
user@ubuntu:~/svc$ sudo iptables -nL -t nat
Chain PREROUTING (policy ACCEPT)
target                          prot opt source              destination
KUBE-SERVICES                   all  --  0.0.0.0/0           0.0.0.0/0       /* kubernetes service portals */
DOCKER                          all  --  0.0.0.0/0           0.0.0.0/0       ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target                          prot opt source              destination

Chain OUTPUT (policy ACCEPT)
target                          prot opt source              destination
KUBE-SERVICES                   all  --  0.0.0.0/0           0.0.0.0/0       /* kubernetes service portals */
DOCKER                          all  --  0.0.0.0/0           !127.0.0.0/8    ADDRTYPE match dst-type LOCAL

Chain KUBE-MARK-DROP (0 references)
target                          prot opt source              destination
MARK                            all  --  0.0.0.0/0           0.0.0.0/0             MARK or 0x8000

Chain KUBE-MARK-MASQ (4 references)
target                          prot opt source              destination
MARK                            all  --  0.0.0.0/0           0.0.0.0/0             MARK or 0x4000

Chain KUBE-NODEPORTS (1 references)
target                          prot opt source              destination

Chain POSTROUTING (policy ACCEPT)
target                          prot opt source              destination
KUBE-POSTROUTING                all  --  0.0.0.0/0           0.0.0.0/0       /* kubernetes postrouting rules */
MASQUERADE                      all  --  172.17.0.0/16       0.0.0.0/0

Chain KUBE-POSTROUTING (1 references)
target                          prot opt source              destination
MASQUERADE                      all  --  0.0.0.0/0           0.0.0.0/0       /* kubernetes service traffic requiring SNAT */ mark match 0x4000/0x4000

Chain KUBE-SERVICES (2 references)
target                          prot opt source          destination
KUBE-SVC-XGLOHA7QRQ3V22RZ       tcp  --  0.0.0.0/0       10.106.125.172  /* kube-system/kubernetes-dashboard: cluster IP */ tcp dpt:80
KUBE-SVC-NPX46M4PTMTKRN6Y       tcp  --  0.0.0.0/0       10.96.0.1       /* default/kubernetes:https cluster IP */ tcp dpt:443
KUBE-SVC-TCOU7JCQXEZGVUNU       udp  --  0.0.0.0/0       10.96.0.10      /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
KUBE-SVC-ERIFXISQEP7F7OF4       tcp  --  0.0.0.0/0       10.96.0.10      /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
KUBE-NODEPORTS                  all  --  0.0.0.0/0       0.0.0.0/0       /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */
ADDRTYPE match dst-type LOCAL

Chain KUBE-SVC-XGLOHA7QRQ3V22RZ (1 references)
target                          prot opt source              destination
KUBE-SEP-WIZ23EZGADKT3IPN       all  --  0.0.0.0/0           0.0.0.0/0             /* kube-system/kubernetes-dashboard: */

...

Chain KUBE-SEP-WIZ23EZGADKT3IPN (1 references)
target                          prot opt source              destination
KUBE-MARK-MASQ                  all  --  10.32.0.3           0.0.0.0/0             /* kube-system/kubernetes-dashboard: */
DNAT                            tcp  --  0.0.0.0/0           0.0.0.0/0             /* kube-system/kubernetes-dashboard: */ tcp to:10.32.0.3:9090
```

KubeProxy and iptables

```
$ kubectl run testweb --image=nginx
deployment.apps/testweb created

$ kubectl create -f svc.yaml
service/testweb created

$ kubectl scale deployment --replicas=3 testweb
deployment.extensions/testweb scaled

$ kubectl get deploy,po,rs
NAME              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/testweb    1         1         1            1           5m

NAME                           READY     STATUS    RESTARTS   AGE
po/testweb-3248343053-d74h5    1/1       Running   0          4m
po/testweb-3248343053-ftfz8    1/1       Running   0          4m
po/testweb-3248343053-m8htj    1/1       Running   0          5m

NAME                     DESIRED   CURRENT   READY     AGE
rs/testweb-3248343053    1         1         1         5m

$ kubectl get service
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.0.0.1      <none>        443/TCP        2d
testweb      NodePort    10.96.0.149   <none>        80:30143/TCP   3m

$ kubectl get endpoints testweb
NAME      ENDPOINTS                                      AGE
testweb   10.32.0.4:80,10.32.0.5:80,10.32.0.6:80         3m

$ curl -s http://10.96.0.149 | head -4
...
<title>Welcome to nginx!</title>

$ curl -s http://10.32.0.4 | head -4
...
<title>Welcome to nginx!</title>
```

```
$ vim svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: testweb
  labels:
    svc: testweb
spec:
  type: NodePort
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    ClientIP:
      timeoutSeconds: 600
  ports:
  - port: 80
  selector:
    run: testweb
```

NodePort & ClusterIP Service

NodePort & ClusterIP Service Nat

```
user@ubuntu:~/svc$ sudo iptables -nL -t nat

Chain PREROUTING (policy ACCEPT)
target          prot opt source              destination
KUBE-SERVICES   all  --  0.0.0.0/0           0.0.0.0/0       /* kubernetes service portals */
DOCKER          all  --  0.0.0.0/0           0.0.0.0/0       ADDRTYPE match dst-type LOCAL


Chain KUBE-NODEPORTS (1 references)
target                    prot opt source              destination
KUBE-MARK-MASQ            tcp  --  0.0.0.0/0           0.0.0.0/0     /* default/testweb: */ tcp dpt:30143
KUBE-SVC-EG66NKXDU7X2QZTA tcp  --  0.0.0.0/0           0.0.0.0/0     /* default/testweb: */ tcp dpt:30143


Chain KUBE-SERVICES (2 references)
target                         prot opt source         destination
KUBE-SVC-XGLOHA7QRQ3V22RZ      tcp  --  0.0.0.0/0      10.106.125.172  /* kube-system/kubernetes-dashboard: cluster IP */ tcp dpt:80
KUBE-SVC-NPX46M4PTMTKRN6Y      tcp  --  0.0.0.0/0      10.96.0.1       /* default/kubernetes:https cluster IP */ tcp dpt:443
KUBE-SVC-TCOU7JCQXEZGVUNU      udp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
KUBE-SVC-ERIFXISQEP7F7OF4      tcp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
KUBE-SVC-EG66NKXDU7X2QZTA      tcp  --  0.0.0.0/0      10.96.0.149     /* default/testweb: cluster IP */ tcp dpt:80
KUBE-NODEPORTS                 all  --  0.0.0.0/0      0.0.0.0/0       /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */
ADDRTYPE match dst-type LOCAL


Chain KUBE-SVC-EG66NKXDU7X2QZTA (1 references)
target                    prot opt source      destination
KUBE-SEP-7KX2LIG5LIB5UVVO  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-7KX2LIG5LIB5UVVO side: source mask: 255.255.255.255
KUBE-SEP-TSJSFJG2GUXROSTI  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-TSJSFJG2GUXROSTI side: source mask: 255.255.255.255
KUBE-SEP-RY3VYSR3AYX2BBO2  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-RY3VYSR3AYX2BBO2 side: source mask: 255.255.255.255
KUBE-SEP-7KX2LIG5LIB5UVVO  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */ statistic mode random probability 0.33332999982
KUBE-SEP-TSJSFJG2GUXROSTI  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */ statistic mode random probability 0.50000000000
KUBE-SEP-RY3VYSR3AYX2BBO2  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */


Chain KUBE-SEP-7KX2LIG5LIB5UVVO (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.4   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-7KX2LIG5LIB5UVVO side: source mask: 255.255.255.255 tcp to:10.32.0.4:80


Chain KUBE-SEP-TSJSFJG2GUXROSTI (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.5   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-TSJSFJG2GUXROSTI side: source mask: 255.255.255.255 tcp to:10.32.0.5:80


Chain KUBE-SEP-RY3VYSR3AYX2BBO2 (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.6   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-RY3VYSR3AYX2BBO2 side: source mask: 255.255.255.255 tcp to:10.32.0.6:80
```

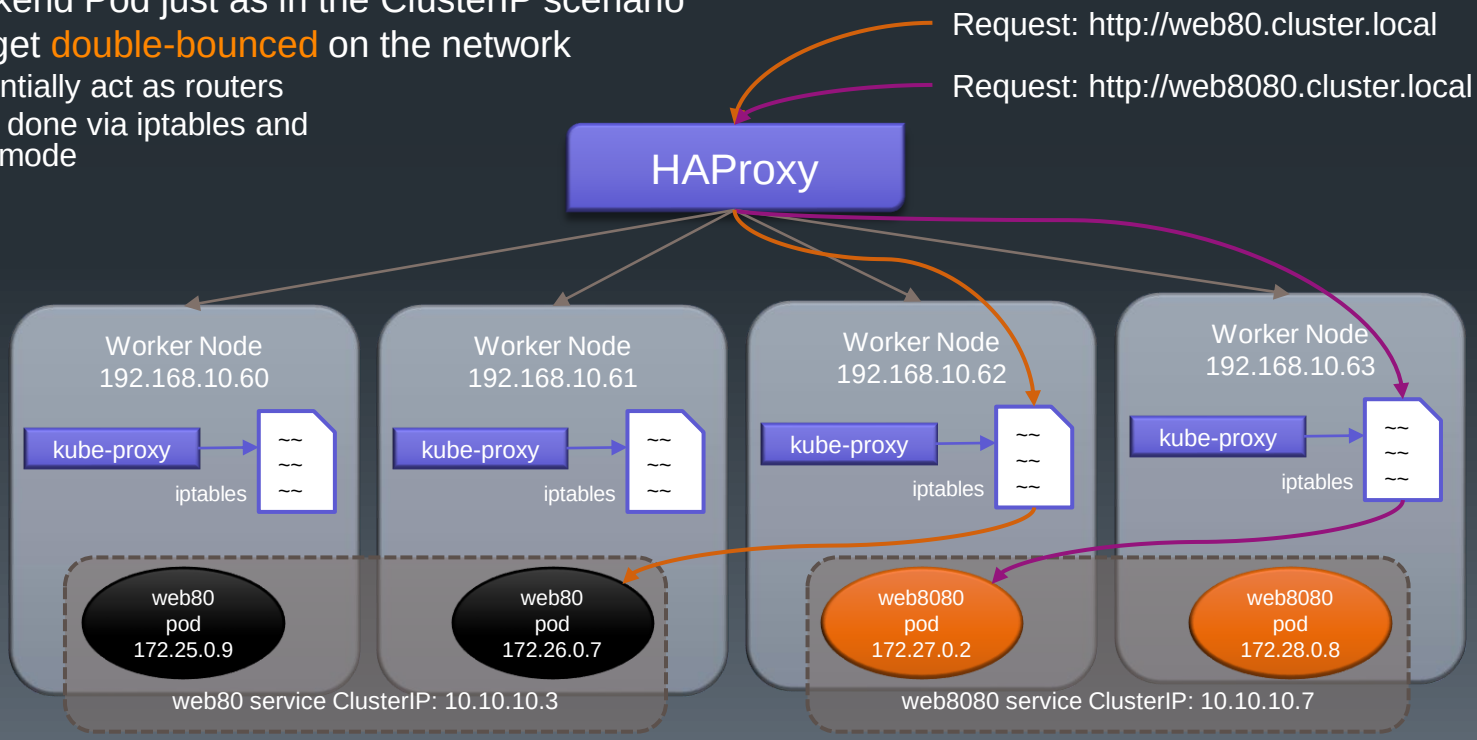
Statistic IPTables module: apply a rule with a probability

IPTables module "recent" enables session affinity

Reverse NAT for hairpin traffic

External Load Balancers

- Access to a pod or service from within the cluster is clean and simple, using the Pod IP or the kube-proxy established service ClusterIP (VIP)
- Accessing a pod from outside the cluster requires an alternate approach
- LoadBalancer services wire up cloud load balancer
  - GCE's ForwardingRules
  - AWS's ELB
  - Azure Load Balancer
- NodePort services expose the service on a single port on every node's external interface
  - When traffic arrives at a node on the node port it is routed to an appropriate backend Pod just as in the ClusterIP scenario
  - Most traffic will get double-bounced on the network
    - The nodes essentially act as routers
    - All forwarding is done via iptables and purely in kernel mode



IPVS

- IPVS (IP Virtual Server)
  - ipvs 1.2.1 is the latest stable version
- Transport-layer load balancing inside the Linux kernel (aka Layer-4 switching/load balancing)
- Function:
  - IPVS running on a host can act as a load balancer at the front of a cluster of real servers, pods or containers
  - Directs requests for TCP/UDP based services to the real servers, and makes services of the real servers to appear as a virtual service on a single IP address
- Added to the official kernel 2.6.10 released on December 24, 2004
  - IPv6 support for IPVS was included in the Linux kernel 2.6.28-rc3 on November 2, 2008
- IPVS offers several connection scheduling algorithms:
  - RR (Round-Robin) – assigns connections sequentially
  - WRR (Weighted Round-Robin) – servers receive an integer weight which defines how many times they appear in a single RR sequence (A-2,B-3 == BABAB)
  - LC (Least-Connection) – directs connections to the server with the least connections
  - WLC (Weighted Least-Connection) – servers receive an integer weight, connections go to servers with the lowest ratio of connections/weight
    - Default K8s algorithm with weight of 1 per server unless set
  - LBLC (Locality-Based Least-Connection) – target IP traffic is sent to the same server (locality) until it has more connections than its weight, after which traffic is sent to the server with least connections (which must be at <=50% of its weight)
  - LBLCR (LBLC with Replication) – allows multiple servers to act as the locality target, adding new servers to the target set as the existing servers become over weighted
  - DH (Destination Hashing) – destination selected by destination hash key
  - SH (Source Hashing) – destination selected by source hash key (ClientIP affinity)
  - SED (Shortest Expected Delay) – like WLC except (con+1)/weight
  - Never Queue – destination with no connections selected, if none exist then SED
- ipvsadm is used to set up, maintain or inspect the IP virtual server table in the Linux kernel

Summary

- Kubernetes implements a simple Linux native networking model using standards driven technologies
  - Virtual IPs
  - IPTables / IPVS
  - DNS
  - Etc.
- Services enable users to write IPTables / IPVS rules via the Kubernetes API

Lab 4

- Services and kube-proxy