Introduction to Containers and Kubernetes

Ruben Orduz + Nolan Brubaker
T: @rdodev + @palendae

A bit about Ruben

- Software developer
- Have been using cloud tech in production for about 7 years
- OpenStack developer advocate for a few years
- Presently work as Field Engineer at Heptio

A bit about Nolan

- Software developer
- Background in Python web development
- OpenStack-Ansible contributor for 3 years
- Currently working on Heptio Ark backup software

Administrative Stuff

- Format: two main parts of two subparts each (concepts and practice).
- We'll take couple of human-needs breaks
- If you have already been playing with either of the technologies here, you might not get much value out of this workshop

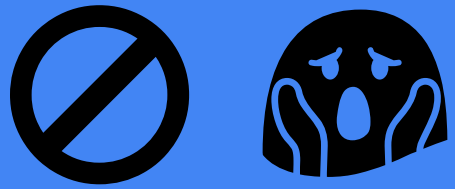
Our Q + A “rules”

- Do you know the answer to this question?
- Is it really a comment but with a question mark tacked on?
- Is the question rhetorical?
- Trying to show off how much you know about the subject?

Our Q + A “rules”

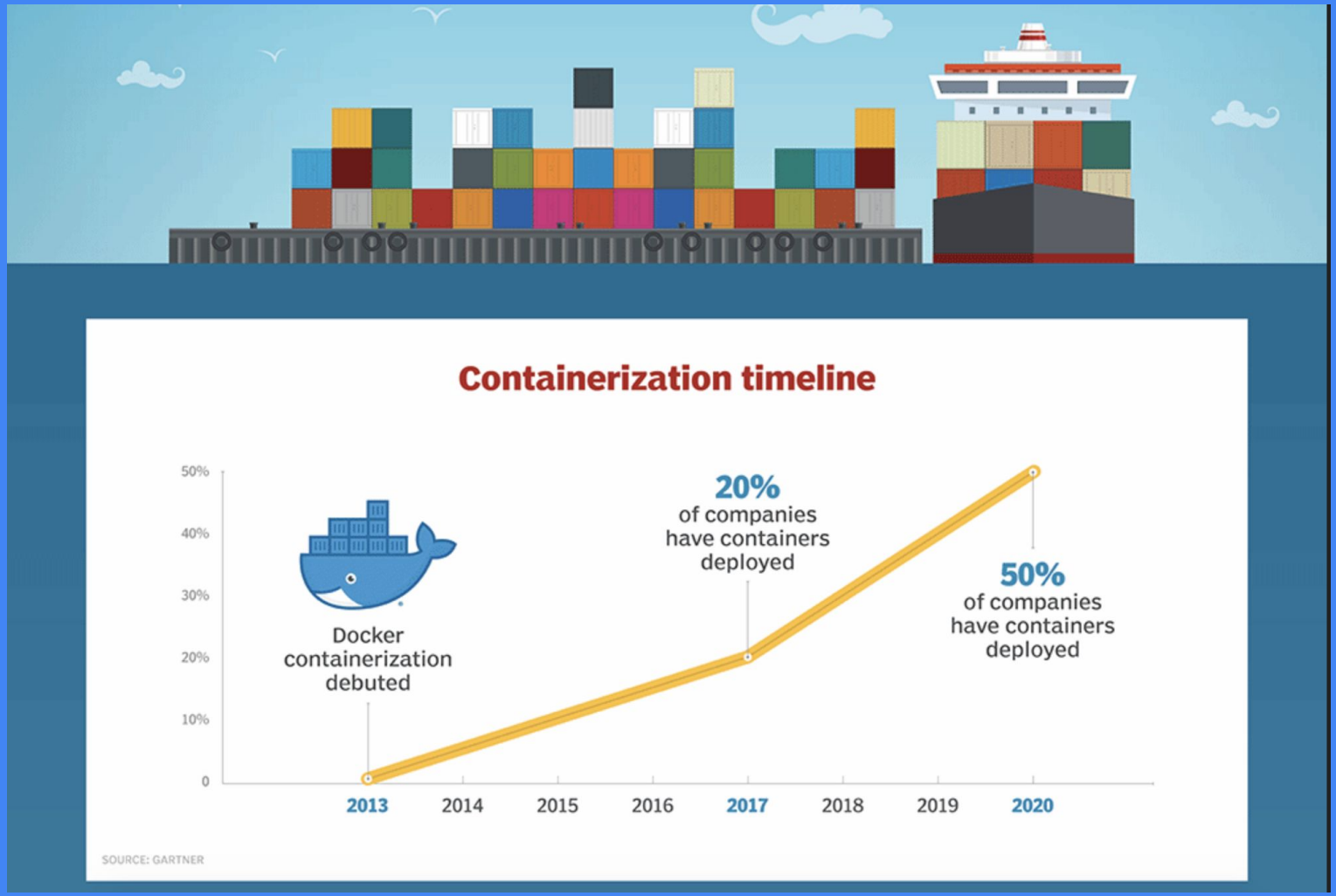
- Speed of delivery
- Not understanding
- Request to repeat explanation
- Anything immediately related to the topic(s) at hand

DON’T PANIC



Why this Workshop?

Container adoption has been insane in the last 4 years.

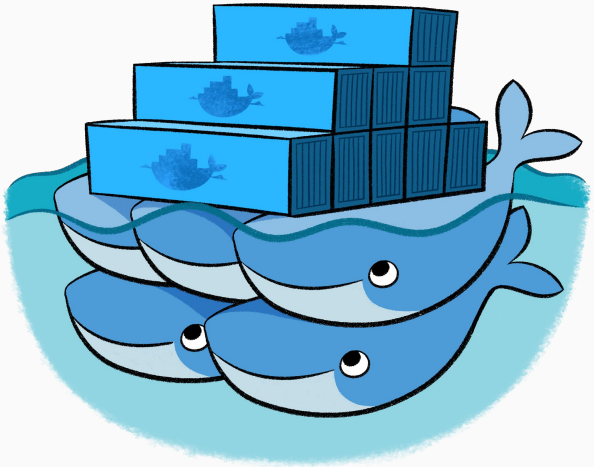

Containerization timeline
50%
40%
30%
20%
10%
0
Docker containerization debuted
20%
of companies have containers deployed
50%
of companies have containers deployed
2013
2014
2015
2016
2017
2018
2019
2020
SOURCE: GARTNER

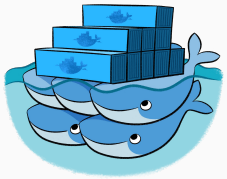
However, containers by themselves are an incomplete story...

Containers

Need a story for ...

- Deployment at scale
- Orchestration
- Monitoring
- Efficient host resources usage

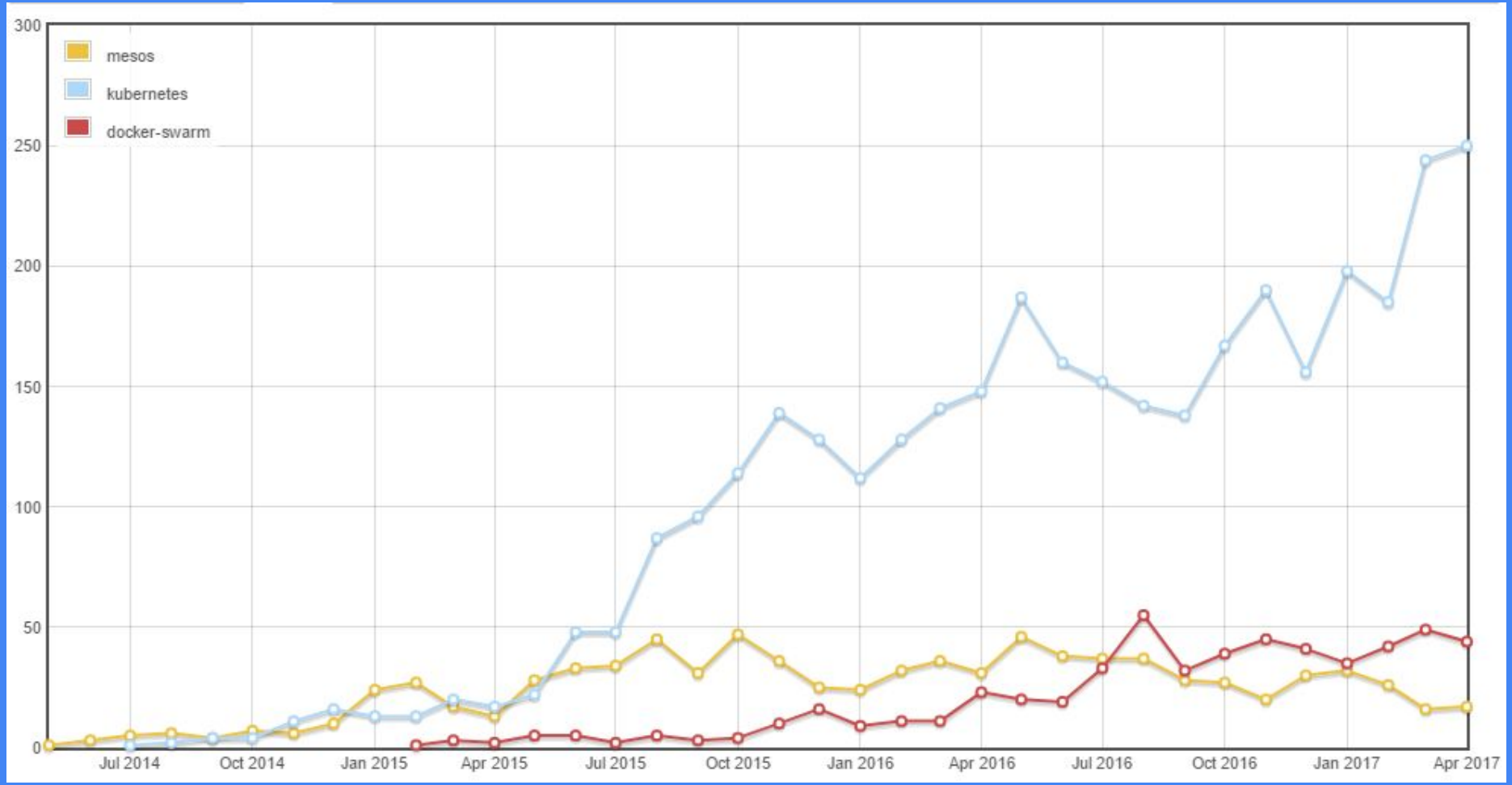




- Tightly coupled to Docker
- Too opinionated
- Difficult to extend
- Internal networking architecture hard to interop
- Project open source but governed by Docker

- Not container specific
- Substantial learning curve
- Scheduler too generic
- Major parts written in different languages
- Overkill for smaller deployments

kubernetes


mesos
kubernetes
docker-swarm
300
250
200
150
100
50
0
Jul 2014
Oct 2014
Jan 2015
Apr 2015
Jul 2015
Oct 2015
Jan 2016
Apr 2016
Jul 2016
Oct 2016
Jan 2017
Apr 2017

- Solved a lot of the issues other orchestrators had
- Gentler learning curve
- Container runtime independent
- … but much more detail about this later

Part I. Containers

Containers are not really a new thing.

Abridged History

2000 “Jails” are included in FreeBSD.

2001-ish, the precursor to Virtual Private Server in introduced to the Linux ecosystem

2004 Sun adds “Zones” to Solaris.

2006-ish early precursors of cgroups raise to the fore

2008 Kernel namespaces and LXC (Linux Containers)

2013 Docker is released

Runtime Alternatives

Main Benefits

- Velocity
- Portability*
- Reliability
- Efficiency
- Isolation
- Infrastructure as code

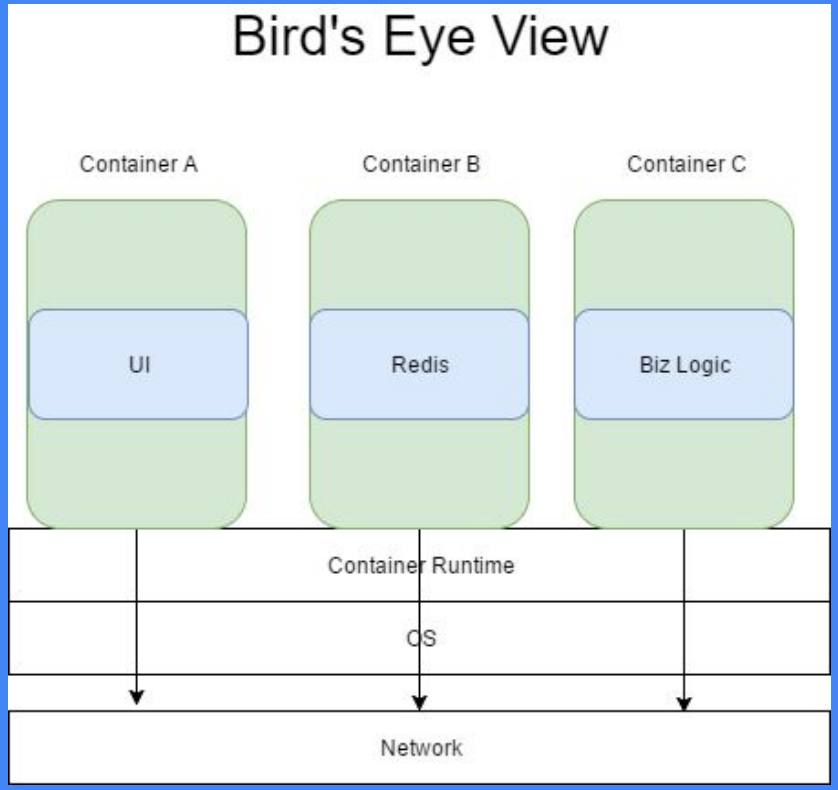
*mostly

Containers aren’t VMs

- Full OS on virtualized hardware
- “Hypervisor Tax”
- Provisioning can take a long time (relatively speaking)
- Images are immutable and monolithic.
- Substantial config work

- Decouples application from underlying OS
- Composable Images
- Lean
- Very fast provisioning times
- Very “Disposable”

“All great stuff,
but now with less
hand waving, pls”


Bird's Eye View
Container A
Container B
Container C
UI
Redis
Biz Logic
Container Runtime
OS
Network

The Container Model

- Shares resources with other processes, but constrained to high degree of isolation
- Can access persistent storage as volumes
- Can share/pass context

Container Images

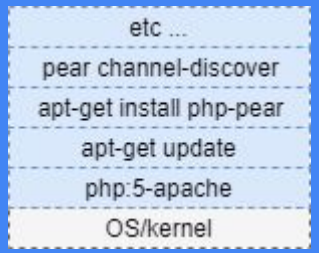
- Vaguely similar in concept with VM images, OS images, etc.
- In Containerland an image is a bundle of layers that together form only logical unit
- 1:N (image:containers)

Container Images

- In the Docker ecosystem they are named and labeled as follows:
  <user>/<imagename>:<tag>
  Ex: ruben/introtock:latest (not an actual thing)
- Generally speaking images default to ‘latest’

Container Images

- Almost every instruction in a Dockerfile creates a layer in the image




```
FROM php:5-apache

RUN apt-get update
RUN apt-get install -y
php-pear
RUN pear channel-discover
```

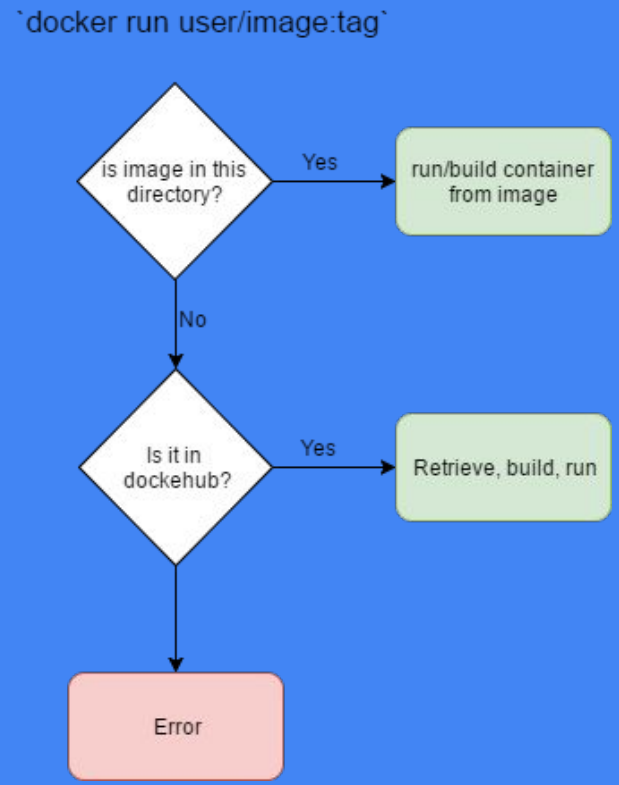
Container Images

- The layered file system is key to images being composable
  - Key difference between container and traditional VM/OS images
  - All layers are presented as one logical unit

Image Registries

- Registries are network storage for images
- You can publish (push) to or download (pull) from them
- Registries can be public, private and local

Image Registries



Let’s hit the shell

Quiz Time!

- Create a new directory
- Create Dockerfile that builds on python:2 and prints to stdout The Zen of Python (hint ‘this’ module)
  - Hint: it only takes 2 lines (one for FROM and one for CMD)
- Build/tag the image
- Push Image to local registry
- Delete image from local cache
- Run container

“I thought you said Kubernetes. Where’s the Kubernetes?”

The Motivation

- Google calculated “human errors” were root cause of over 99% of outages
- Continuous system changes
- Lack of deployment consistency and scalability

- Google creates Borg to address these issues for their own infrastructure
- Insane learning curve
- Engineer and operator ramp-up was (still is!) lengthy

- But 99.99% of users aren’t Google
  - infinitesimal scale in comparison
  - Budget, talent, etc.
  - Line of business

- A group of Googlers (Joe*, Brendan, et al) saw the opportunity
  - General applicability
  - Open Source

Full disclosure: * my employer

Design Principles

- Strong focus on developer and operator experience
- Easily extensible, with few opinions
- A subset of Borg, but for the masses

What can it do?

- Aggregate virtual and bare metal infrastructure into a cluster
- Container orchestration within clusters
- Provide straightforward app lifecycle management

Key Features

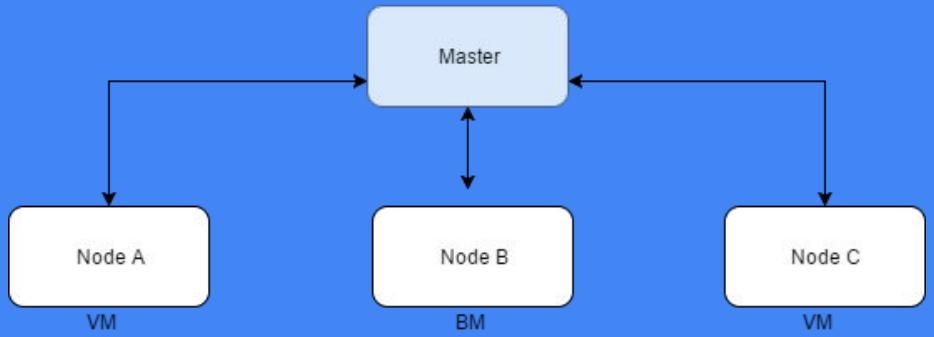
- Automatic binpacking
- Straightforward horizontal scaling
- Automatable rollouts/rollbacks
- Deployment “medic”
- Highly Fault Tolerant (including itself!)

Key Features

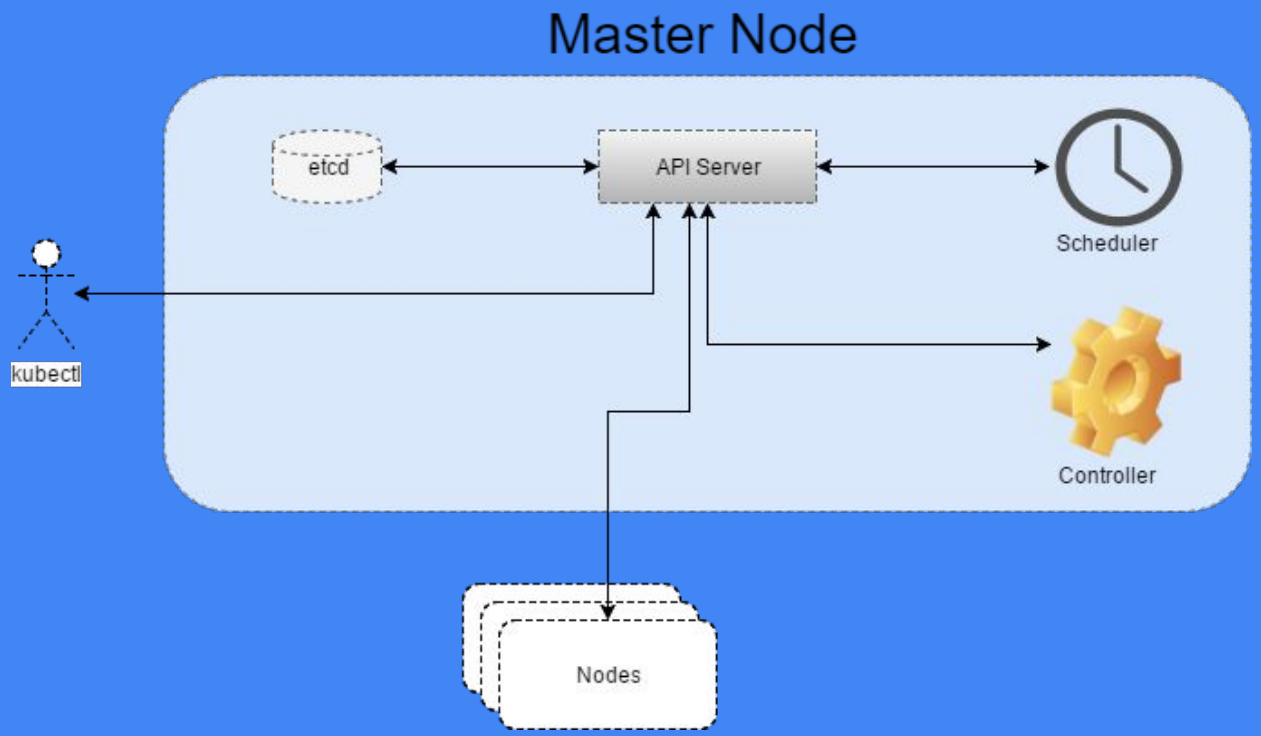
- Service discovery
- Load balancing
- Secret management
- Batch execution

“Nice. But would like more specifics, though”

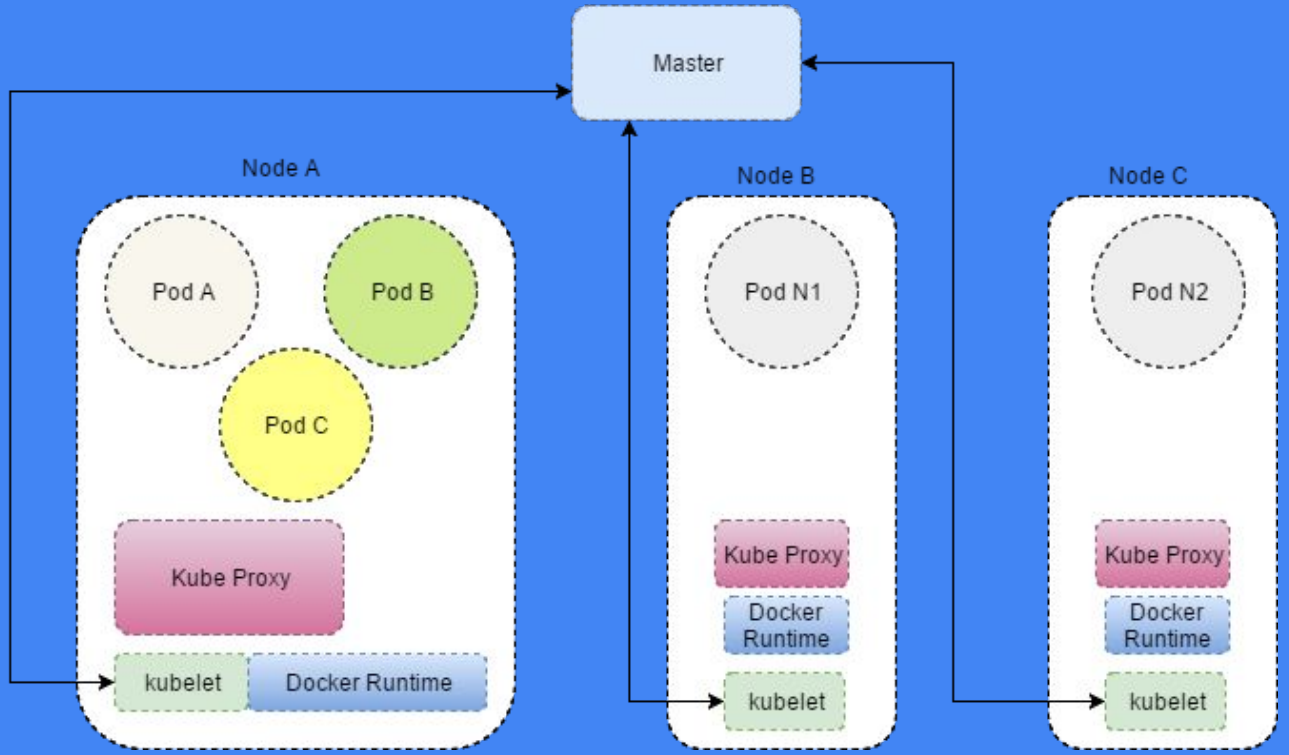
Orbital View



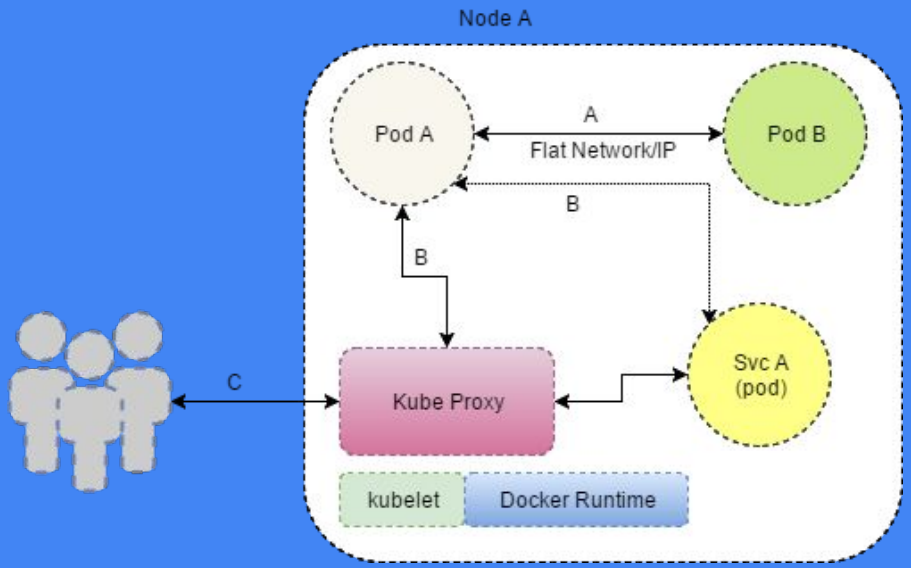
Zoom...enhance....


Master Node
etcd
API Server
Scheduler
kubectl
Controller
Nodes

Worker Nodes "Node" Zoom



Networking


Node A
Pod A
A
Flat Network/IP
Pod B
B
B
C
Kube Proxy
Svc A
(pod)
kubelet
Docker Runtime

Networking

- Scenario A: pod-to-pod (flat network, basic IP)
- Scenario B: Pod-to-Service (virtual-IP, through kubeproxy)
- Scenario C: External-to-service (fronted by a LB -- not shown -- and routed by kubeproxy)

Main Components: Objects

- Objects are persistent entities that manage and represent the state of certain parts of the system
  - Node
  - Service
  - Pod
  - Deployment … etc.

Main Components: Objects

- Objects describe state of:
  - What deployments/services are running
  - Node status
  - Services, storage
  - Deployments
  - Namespaces … etc.

Pods

- Smallest unit of compute that can be managed by Kubernetes
  - Containers within pods run as if they’re in a single host
  - Share namespaces, IP addr and port space
  - Comms through localhost:port or over IPC
  - Mount/use same volumes

Labels

- Key/value pairs that are associated to objects and carry semantic value within Kubernetes
- Organize and select resources
- Commonly used to ID releases, environment, tiers
- Can be used as query selectors

Annotations

- Somewhat similar to Labels, except:
  - Meaningless to Kubernetes
  - Can’t be queried (i.e. not indexed)
  - Usually good for bespoke/extended metadata

Services

- A grouping of Pods running in on the cluster
  - Load balancing
  - Service Discovery
  - Zero downtime deployments/upgrades

ReplicaSets

- Ensures specified number of pod instances are running at any given time.
- Can update many aspects via spec definition
- Can manage one or more Pods based on labels

Deployments

- Manages ReplicaSets
- Best object for deploying and managing applications in Kubernetes
- Manages the update process
- More

Back to the shell ...

Before we go ...

Don’t Forget Security

- Secrets
  - Hold sensitive data (credentials, keys, etc.)
  - RBAC/ABAC configurable
  - Can be leveraged by the system or regular users

Don’t Forget Security

- Authorization:
  - RBAC: system auth managed by Kubernetes
    - Impose restrictions at the Namespace level
  - WARNING: comes “disabled” by default -- not production ready in some bootstrappers.

The End.

Thank you!

Tweet at us @rdodev @palendae