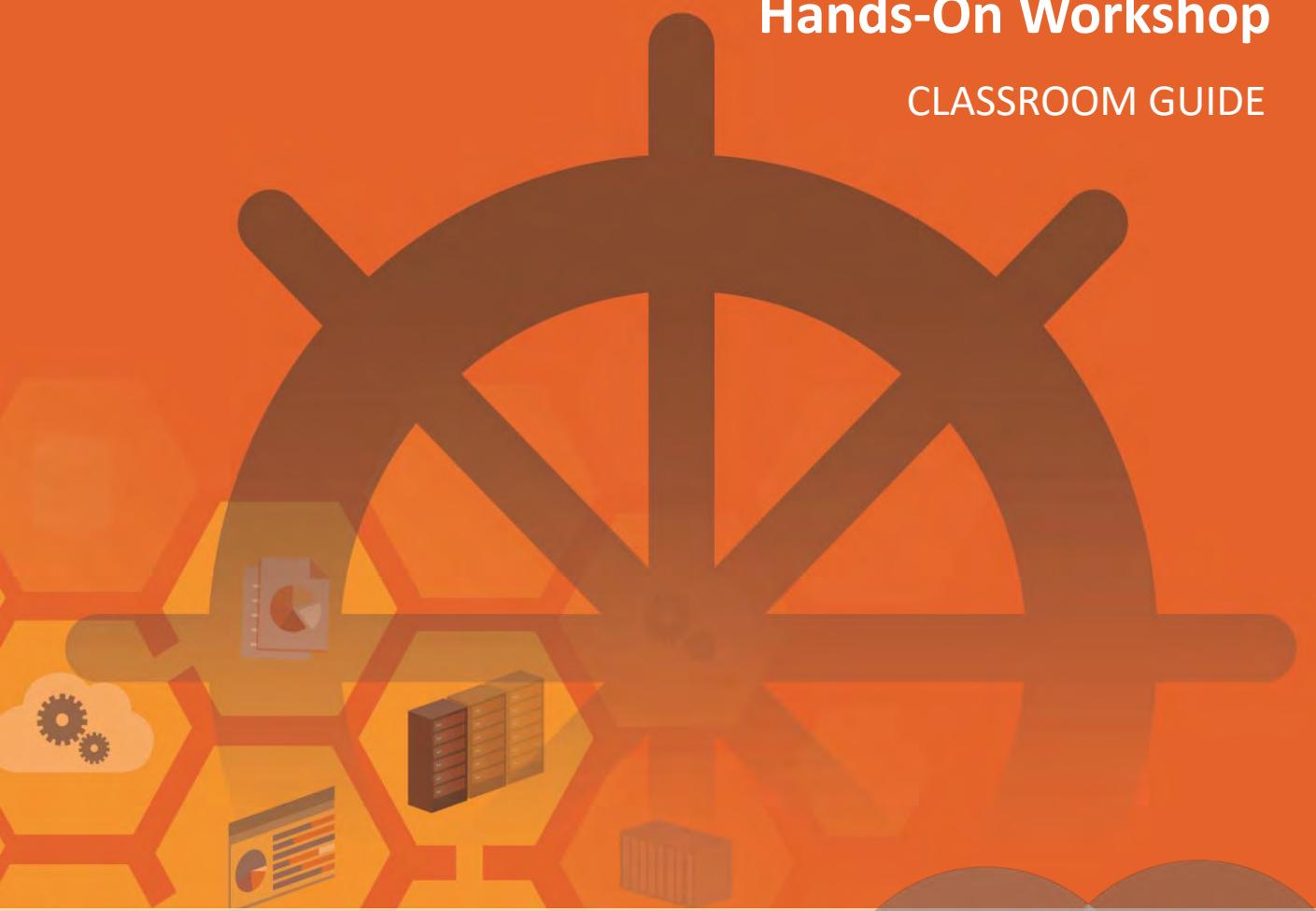


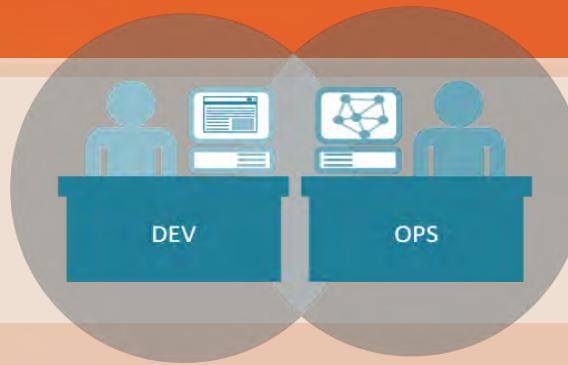
Introduction to Kubernetes

Hands-On Workshop

CLASSROOM GUIDE



Getting up and running with DevOps-style containers, microservices, legacy refactoring, and integrated security in production



ASPE
TRAINING

a division of Fortis College

64300PG_2.0_2018



Introduction to Kubernetes

64300M_2.0_2018



NOTES:

Welcome!

Logistics (breaks,
facilities, lunch, etc.)

Rules of Engagement

Introductions

Let's Get Started!



NOTES:

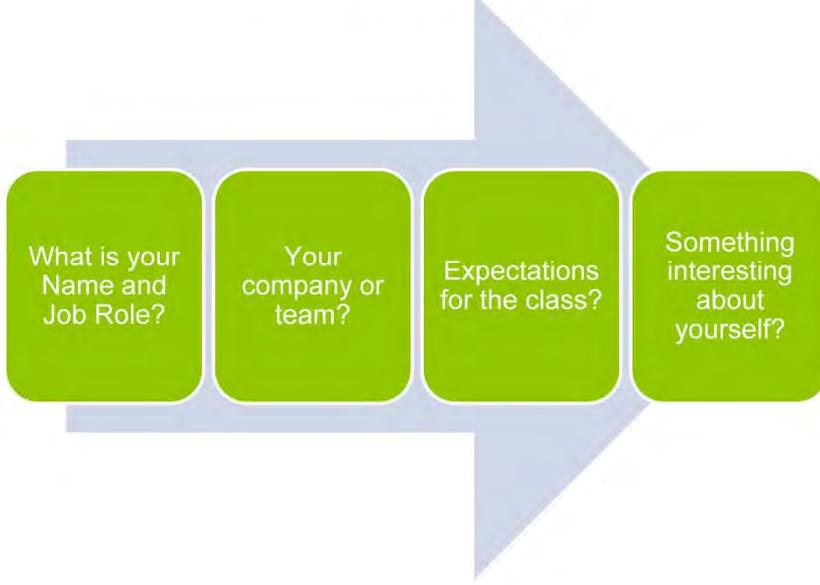


Who is your instructor? *A little about me...*



NOTES:

Introductions

- 
- What is your Name and Job Role?
 - Your company or team?
 - Expectations for the class?
 - Something interesting about yourself?



NOTES:

What to expect from this class

- Flexibility
- Conversations
- Literacy and awareness on many of the principles, tools, and practices **surrounding Microservices culture, architecture and implementation.**
- An effort to focus on your own situations and challenges so you can act on what you learn.



NOTES:

Introduction to Kubernetes

Part 1:



NOTES:

Journey is a “Team of Teams”

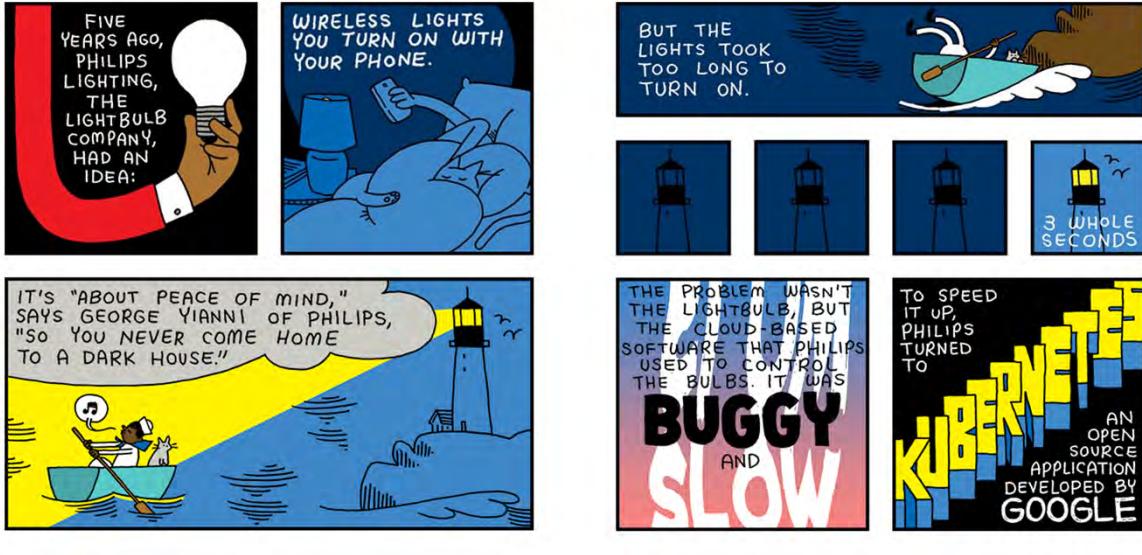
“First I needed to shift my focus from moving pieces on the board to shaping the ecosystem.”

— General S. McChrystal, *Team of Teams: New Rules of Engagement for a Complex World*



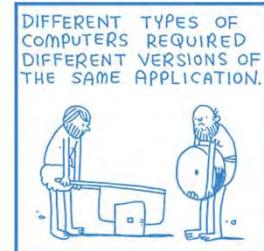
NOTES:

Kubernetes Intro (Phillips Case)



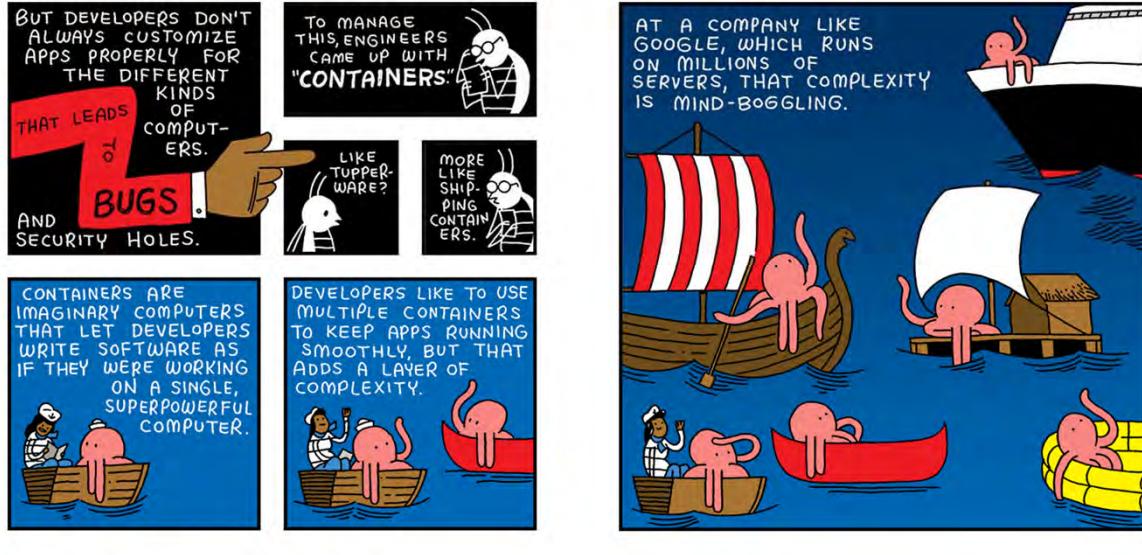
<https://www.bloomberg.com/features/2017-kubernetes/>

Kubernetes Intro (Phillips Case)



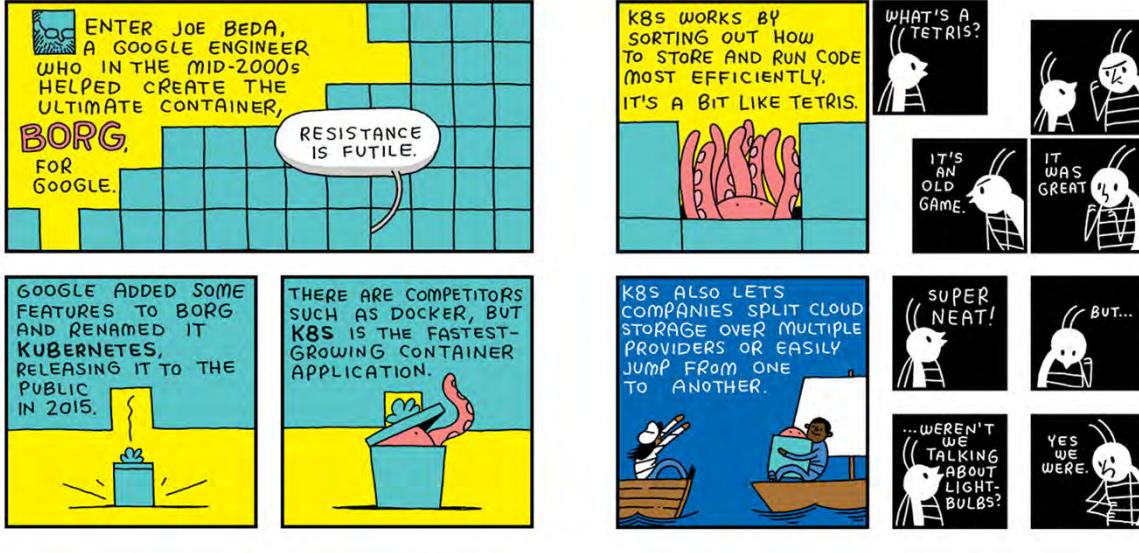
<https://www.bloomberg.com/features/2017-kubernetes/>

Kubernetes Intro (Phillips Case)



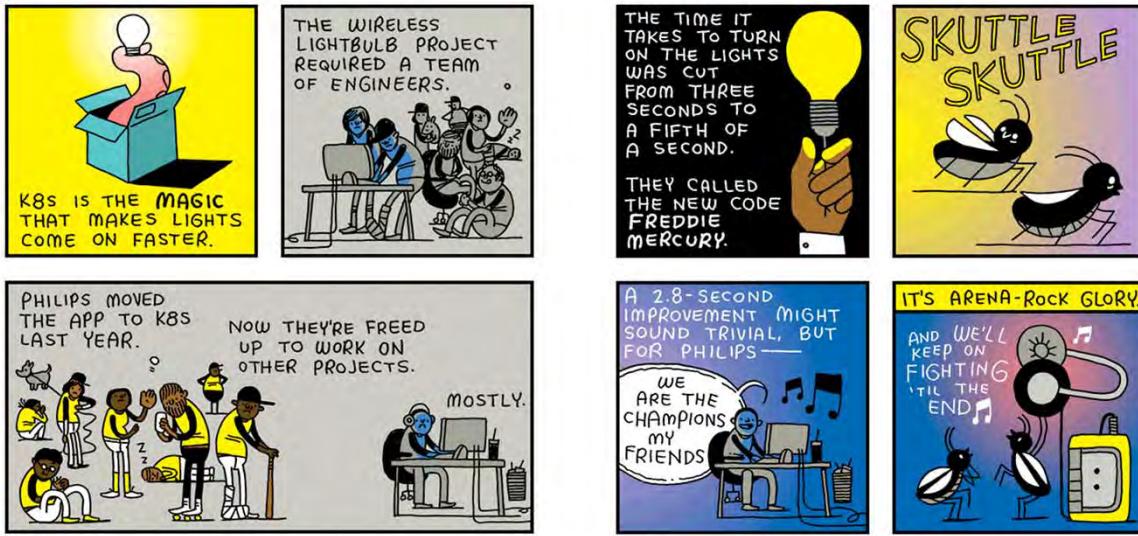
<https://www.bloomberg.com/features/2017-kubernetes/>

Kubernetes Intro (Phillips Case)



NOTES:

Kubernetes Intro (Phillips Case)



ASPE
TRAINING
Exclusive to Frits Olsthoorn

<https://www.bloomberg.com/features/2017-kubernetes/>

Kubernetes

The name Kubernetes is Greek

- Translates to “helmsman” or “pilot”
- Root of “Governor” and “Cybernetic”

“K8s” is a shorthand abbreviation derived by replacing the 8 letters “ubernete” with 8.



Trivia:

The founders initially wanted to call it “seven of nine” in reference to the nice, friendlier borg from Star Trek. This name did not survive, but for this reason the Kubernetes logo has seven sides

Reference:

<https://cloudplatform.googleblog.com/2016/07/from-Google-to-the-world-the-Kubernetes-origin-story.html>

Kubernetes

Project was originally started by Google, and shortly after joined by Redhat

- Based on 10+ years of experience of using containers
 - The most famous of which named “Borg”, Google’s cluster manager “Borg”
- Declarative
 - State desire (i.e. 5 pods)
 - System confirms to State (if currently 3 pods, system changes to 5. If unexpectedly goes to 4 pods, system restores to 5)



<https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44843.pdf>

Cloud Native Computing Foundation (CNCF)

Mission of the Cloud Native Computing Foundation.

- “The Foundation’s mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self healing multi-tenant nodes.”

Formed in Nov 6, 2015 with Kubernetes as its first project

- The CNCF is building an ecosystem of complimentary projects including Prometheus, Fluentd, Linkerd, Rkt, and others



<https://www.cncf.io/about/charter/>

1. Mission of the Cloud Native Computing Foundation.

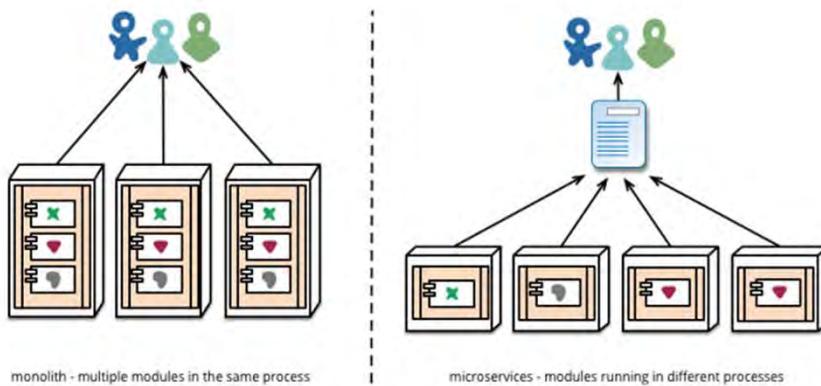
The Foundation’s mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self healing multi-tenant nodes.

Cloud native systems will have the following properties:

- (a) Container packaged. Running applications and processes in software containers as an isolated unit of application deployment, and as a mechanism to achieve high levels of resource isolation. Improves overall developer experience, fosters code and component reuse and simplify operations for cloud native applications.
- (b) Dynamically managed. Actively scheduled and actively managed by a central orchestrating process. Radically improve machine efficiency and resource utilization while reducing the cost associated with maintenance and operations.
- (c) Micro-services oriented. Loosely coupled with dependencies explicitly described (e.g. through service endpoints). Significantly increase the overall agility and maintainability of applications. The foundation will shape the evolution of the technology to advance the state of the art for application management, and to make the technology ubiquitous and easily available through reliable interfaces.

What is a Microservice?

- A service that can be independently deployed



<https://martinfowler.com/articles/microservices.html>

How big is a Microservice?

- A **single programmer** can design, implement, deploy and maintain a microservice
– Fred George
- Software that **fits in your head** – Dan North

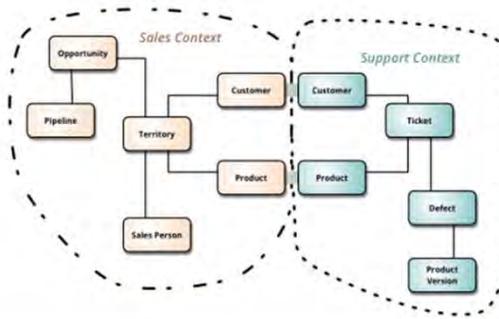
Monolithic



NOTES:

Where is the data for a Microservice?

- A single logical **database per service** – Chris Richardson
- A microservice implements a single **Bounded Context** – Martin Fowler, Sam Newman



<https://martinfowler.com/bliki/BoundedContext.html>

How to maintain Microservices?

- Addressable through a **service discovery** system – Chris Richardson
- Microservices **built & deployed independently. Stateless**, with state as backing service s – 12Factor.net



NOTES:

How to architect Microservices?

Typically

- CRUD – Create, Read, Update and Delete api implementations
- Range between RPC, Message and Stream processing
- Evolve from a Monolith 1st iteration



NOTES:

Microservice: advantages

- **Simple** services that are focused on doing one thing well
- Each service can be built **using** the **best** tool for the job
- Systems built this way are inherently **loosely coupled**
- Teams can deliver and **deploy independently** from each other
- Enable **continuous delivery** but allowing frequent releases while the rest of the system continues to be stable



Can be (but not always)
< 6000 lines

<https://martinfowler.com/articles/microservices.html>

Microservices

Solve organizational problems

- Teams blocked by or waiting on other teams
- Communication overhead
- Slow velocity

Cause Technical Problems

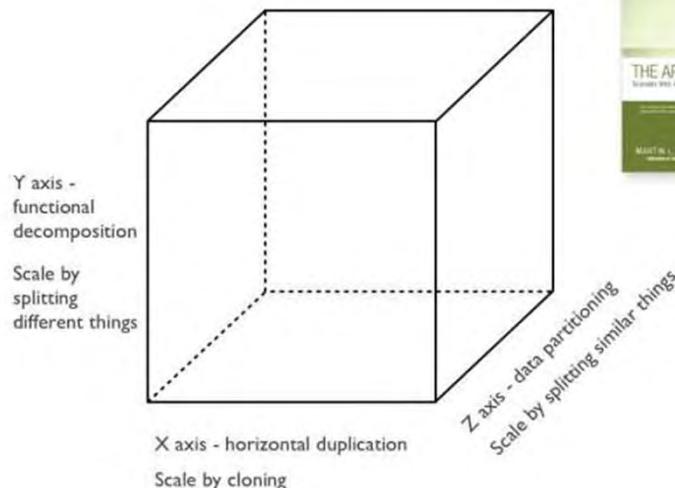
- Requires Devops, devs deploying and operating their services
- Need well defined business domains
- More distributed systems
- Need an orchestration layer



NOTES:

The Art of Scaling

3 dimensions to scaling



<http://microservices.io/articles/scalecube.html>

X-axis: Run multiple copies behind a load balancer

Pros

- Simple

Cons

- Doesn't decrease complexity
- Often limited by state not being effectively shared between instances

Y-axis: Splits the application into multiple different services

Pros

- Decreases complexity
- Most resource efficient

Cons

- Requires intentional effort

Z-axis: Split application into different segments of data

Pros

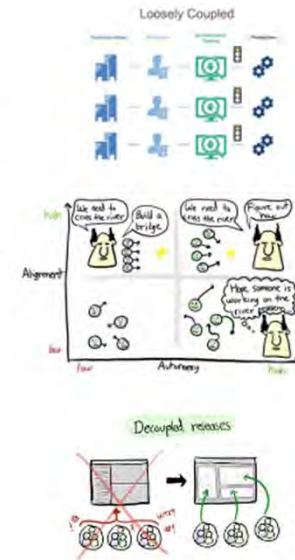
- Most memory resource efficient
- Enables scalability

Cons

- Most complex
- Difficult to refactor

Small Team Size

- Build platform with small teams of 6-8 engineer teams
- **Self Sufficient:** Each team completely owns one or more services and has everybody they need to develop functionality
- **Automate Interfaces:** Teams connect to each other through API's. Teams don't hire an army of program managers to manage dependencies.
- **Decentralized:** Teams plan, develop and deploy autonomously



Starting and Scaling DevOps in the Enterprise by Gary Gruver
https://www.amazon.com/dp/B01M332BN2/ref=dp-kindle-redirect?_encoding=UTF8&btkr=1



<https://clusterhq.com/2016/03/22/microservices-spotify-petter-mahlen>

"There's a yearly survey about what sucks about Spotify's infrastructure. In 2015 service DB, the predecessor showed up as, I think, number 3 in what sucks. Whereas now, in 2016, System-Z is one of the things that gets the most frequent, "It's great!" Even though we're asking what sucks."

"Another interesting thing we noticed is that it leads to teams building related features, infrastructure teams building related features, like an example is the team that owns hardware and provisioning, the team that owns DNS infrastructure, and the team that owns System-Z together have built like an improved version of how you do provisioning and deployment."

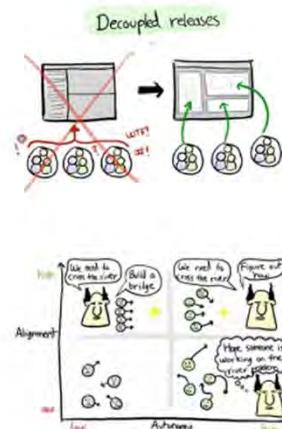
Why Small Team Size? Scaling the Organization

Autonomy

- Decentralizes power and creates autonomy which is critical to scaling the organization
- Each team can focus on the business metric they are responsible for and act autonomously to maximize the metric.

Amazon CTO Werner Vogels explained the advantages of this structure to Larry Dignan of *Baseline* in 2005. Dignan writes:

"Small teams are fast...and don't get bogged down in so-called administrivia....Each group assigned to a particular business is completely responsible for it....The team scopes the fix, designs it, builds it, implements it and monitors its ongoing use. This way, technology programmers and architects get direct feedback from the business people who use their code or applications—in regular meetings and informal conversations."



Science behind why two-pizza team works <http://blog.idonethis.com/two-pizza-team/>
ITRevolution, Conway's Law by Gene Kim <http://itrevolution.com/conways-law/>

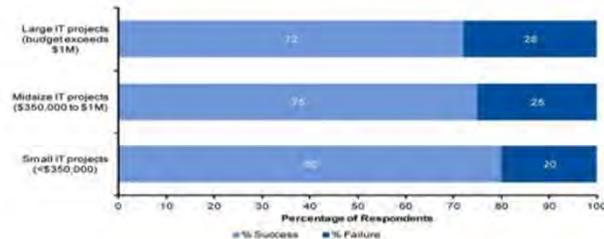


<http://blog.idonethis.com/two-pizza-team/>
<http://itrevolution.com/conways-law/>

Why do Microservices?

Reduce complexity,
increase success

- According to a Gartner survey of 150 participants in 2011, the failure of projects exceeding \$1 million was found to be almost 50% higher than for projects with budgets below \$350,000.
- This result was reinforced by prior Gartner IT projects which found small IT projects to experience a one-third lower failure rate than large projects

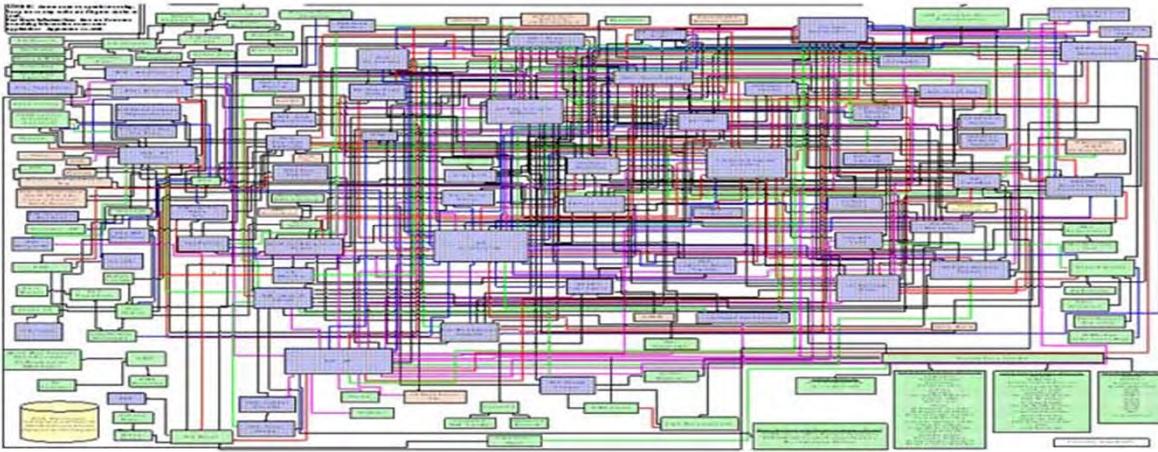


<https://thisiswhatgoodlookslike.com/2012/06/10/gartner-survey-shows-why-projects-fail/>



NOTES:

The Hairball(*monolith*)



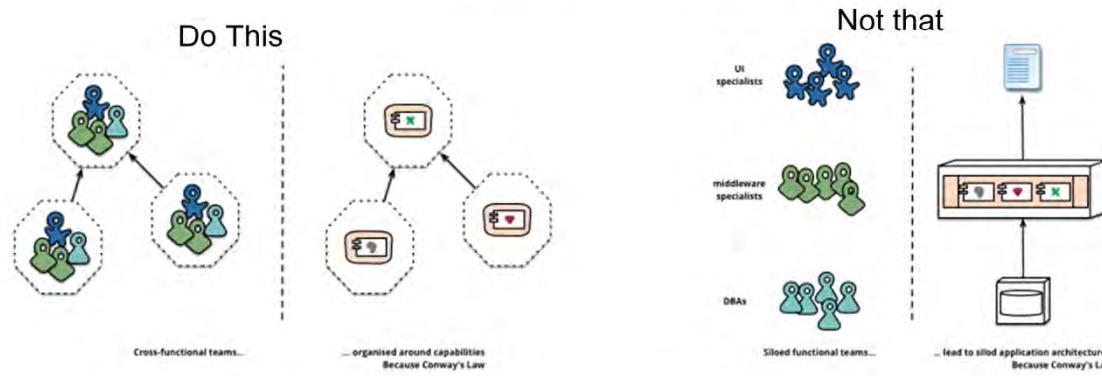
ASPE
TRAINING
a division of Fortis College

NOTES:

Because Conway's Law

"Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."

—Melvin Conway



ASPE
TRAINING
A Division of Forest College

<https://martinfowler.com/articles/microservices.html>

Microservice building blocks



APIs provide the flexibility for businesses to grow by adopting new business models and enable accelerated development of new applications.

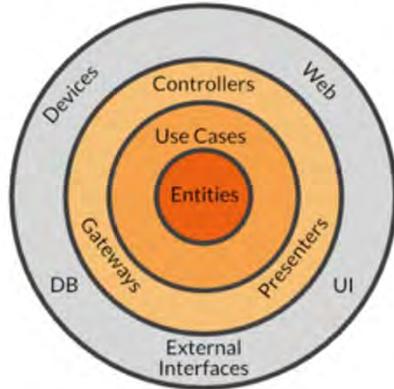
It is like picking different LEGO blocks to build a toy house.



Team Skills | Process & Measurements | Architecture & Engineering
3 bullets under each

(Insert Benchmark Here)

Clean Architecture



The central rule of The Clean Architecture is the Dependency Rule, which says:

**Source code
dependencies can only
point inwards
(Dependency Injection)**



Dependency Injection
<https://appliedgo.net/di/>

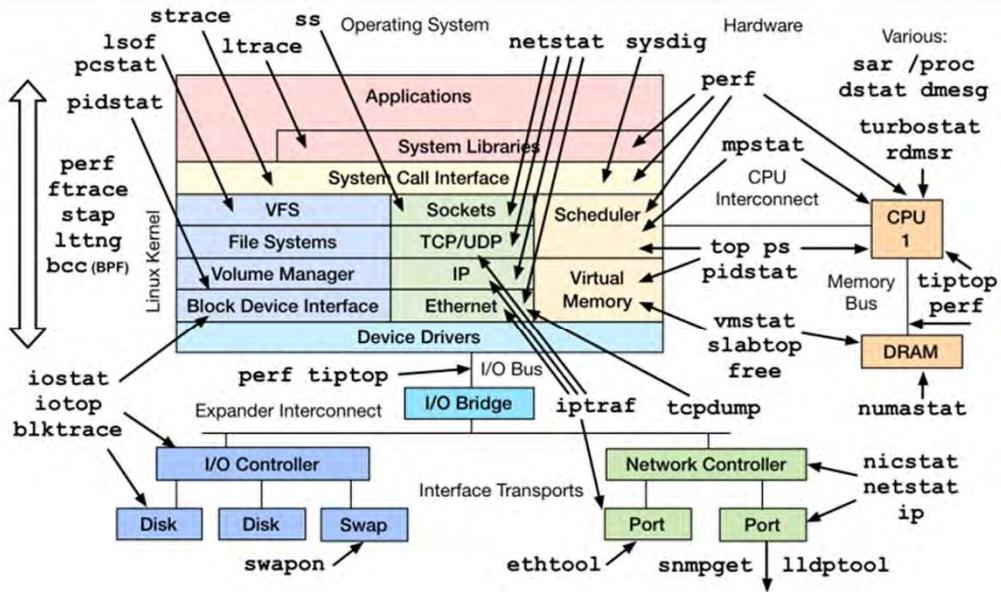
Observability Metrics

- **USE** method (Brendan Gregg)
- For every resource, check
 - **Utilization**: average time that the resource was busy servicing work
 - Example: One disk is running at 90%
 - **Saturation**: the degree to which the resource has extra work which it can't service, often queued
 - Example: The CPUs have an average run queue length of four
 - **Error count (rate)**: the count of error events
 - Example: This network interface has had fifty late collisions
- Useful for resources such as queues, CPU's, memory, interconnects, etc



<http://www.brendangregg.com/usemethod.html>

Linux Performance Observability Tools



<http://www.brendangregg.com/linuxperf.html> 2017



<http://www.brendangregg.com/linuxperf.html>

Observability Metrics

RED method (Tom Wilkie)

For every service, check that

- **Request count** (rate)
- **Error count** (rate)
- **Duration**

Are within SLO/A (*service level objectives*)

- Useful for API endpoints

Note:

Observability = Logging + Monitoring + Tracing + Visualization

All good instrumentation libraries monitoring and trending systems (Prometheus, Graphite, etc) have at least three main primitives

- Counter: records events that happen such as incoming requests, bad requests, errors, etc
- Gauge: records things that fluctuate over time such as the size of a threadpool
- Histogram: records observations of scalar quantities of events such as request durations



<http://www.brendangregg.com/usemethod.html>

Optimize for Time to Value

“Today, when organizations measure and optimize their activities, time to value is becoming a dominant metric”

—Adrian Cockcroft, Devops thought leader reknown for architecture work at Netflix and VP of AWS



NOTES:

Kubernetes Setup

Kubernetes can be setup on multiple clouds

- Google(GCP) & Microsoft offer Kubernetes as a service, with reliable experiences
- AWS has two great open source methods, 'Kops' and Heptio's Cloud Formation Script
- Last year AWS added Kubernetes as a Service
- Kubernetes can readily be setup locally, with 'Minikube' or 'kubeadm'



<http://gist-reveal.it/bit.ly/ato-k8s#/presented-by-ryanj>

Local Kubernetes



CLASSROOM WORK

Minikube is a option for developing locally, but some features are not available and require a cloud provider.

<https://github.com/kubernetes/minikube>



Get prerequisites and install from <https://github.com/kubernetes/minikube>



Google Cloud Setup

CLASSROOM WORK

Install GCE Client Tools

https://github.com/smhillin/intro_to_kubernetes/blob/master/install-gce-client-tools.md

1. Signup with Google Cloud(\$300 free credit)
2. Install gcloud (CLI)
3. Create a VM
4. Test VM
5. Delete VM



<https://cloud.google.com/container-engine/docs/quickstart>

Once you have gcloud installed, first set a default zone:

```
$ gcloud config set compute/zone us-west1-a
```

Then you can create a cluster:

```
$ gcloud container clusters create kuar-cluster
```

This will take a few minutes. When the cluster is ready you can get credentials for the cluster using:

```
$ gcloud auth application-default login
```

At this point, you should have a cluster configured and ready to go. Unless you would prefer to install Kubernetes elsewhere, you can skip to “The Kubernetes Client”.

Google Compute Engine Kubernetes Connect



CLASSROOM WORK

GCE Kubernetes Test Cluster

1. Install kubectl (gcloud components install kubectl)
2. Create test cluster
3. Delete test cluster



<https://cloud.google.com/container-engine/docs/quickstart>

Once you have gcloud installed, first set a default zone:

```
$ gcloud config set compute/zone us-west1-a
```

Then you can create a cluster:

```
$ gcloud container clusters create kuar-cluster
```

This will take a few minutes. When the cluster is ready you can get credentials for the cluster using:

```
$ gcloud auth application-default login
```

At this point, you should have a cluster configured and ready to go. Unless you would prefer to install Kubernetes elsewhere, you can skip to “The Kubernetes Client”.

Pokemon Go



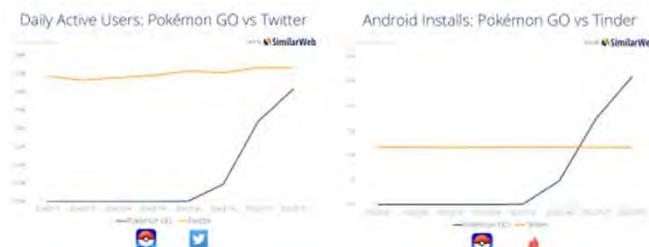
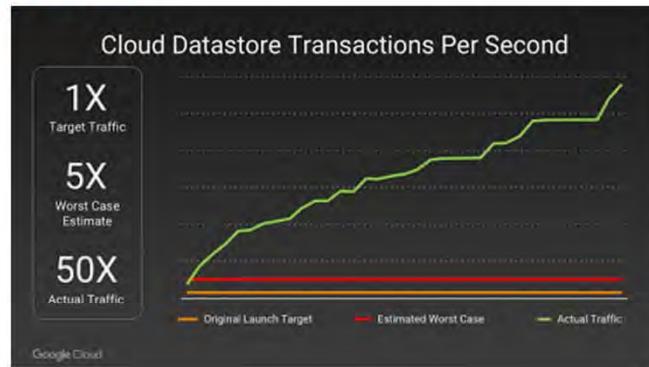
NOTES:

Pokemon Go: Situation

Pokemon Go launched a hit app! The application was a viral, overnight roaring success.

The challenge? It rapidly had 50x the worst case traffic estimate and grew rapidly

- Achieved the same active users in a month that took Twitter years to reach
- Was the most downloaded app in history within a few days



Y Axis – Traffic to Cloud Datastore

X Axis - Date

Pokemon Go: Action

Application for the logic ran on Google Container Engine powered by Kubernetes.

- Enabled automation to scale the application to millions of users
- Also enabled upgrading the version of the cluster and load balancer without downtime while millions of players were online



NOTES:

Pokemon Go: Action

Even Google doesn't have unlimited resources and engineers

- Google CRE (Cloud Reliability Engineer team) worked closely with Niantic to review and optimize the architecture.
- Pokemon Go used Cloud Datastore, a cloud native NoSQL database.
- This enabled the application to scale without the sharding requirements that would have been needed with MySQL or Postgres for example



NOTES:

Pokemon Go: Result

Was able to launch in Japan a mere two weeks after the US launch, where they received 3x the US users without incident.

- Was the largest Kubernetes deployment ever, with multitude of bugs identified, fixed and merged into the open source repo
- The cluster ended up utilizing tens of thousands of cores
- Arguably the first successful overnight/viral planet-wide launch
- Was eventually downloaded more than 500 million times and 25 million players at peak
- Inspired users to walk over 5.4 billion miles over the course of a year due to gameplay



<https://news.ycombinator.com/item?id=12606995>

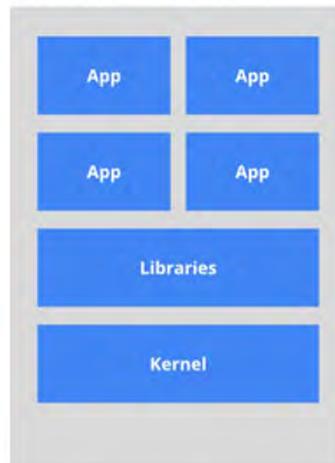
<https://cloudplatform.googleblog.com/2016/09/bringing-Pokemon-GO-to-life-on-Google-Cloud.html>

<http://www.theserverside.com/feature/How-Pokemon-Go-created-a-Kubernetes-powered-Java-cloud>

<https://uploadvr.com/google-pokemon-go-server-issues-unavoidable/>

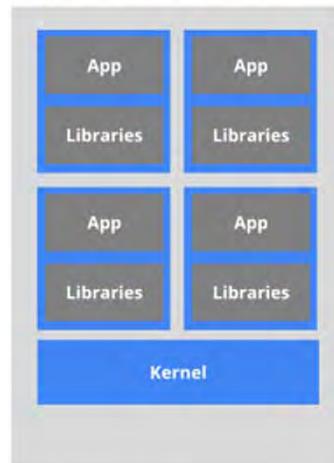
Containers

The old way: Applications on host



*Heavyweight, non-portable
Relies on OS package manager*

The new way: Deploy containers



*Small and fast, portable
Uses OS-level virtualization*



Traditional Methods of deploying applications

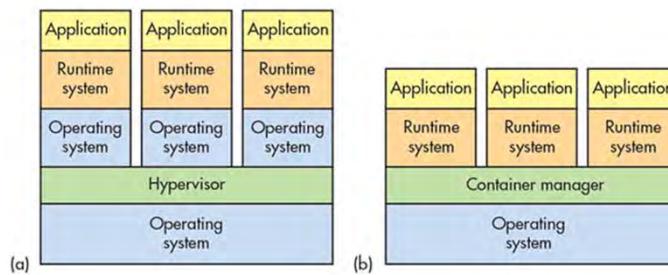
- Typically virtualize the hardware layer (i.e. virtual server)
- Entangled executable, libraries, configuration and lifecycle with host OS
- VMs tend to be heavier, approx 600MB to several GB
- Hypervisor layer can take up to 33% of CPU overhead

Container based approaches

- Operating system virtualization (i.e. virtual Linux machine)
- Encapsulated executables, libraries, configuration, and lifecycle
- Containers tend to be lighter, approx. 20MB to several hundred MB.
- Container layer can typically take between 2-5% CPU overhead

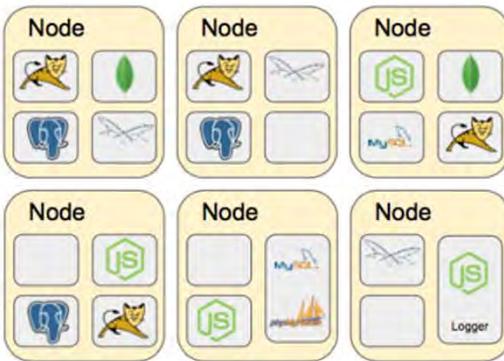
Container Advantages

- Portable
- Isolated
- Lighter footprint & overhead (vs VMs)
- Simplify Devops practices
- Speed up Continuous Integration
- Empower Microservice architectures and adoption



NOTES:

Challenges with multiple containers



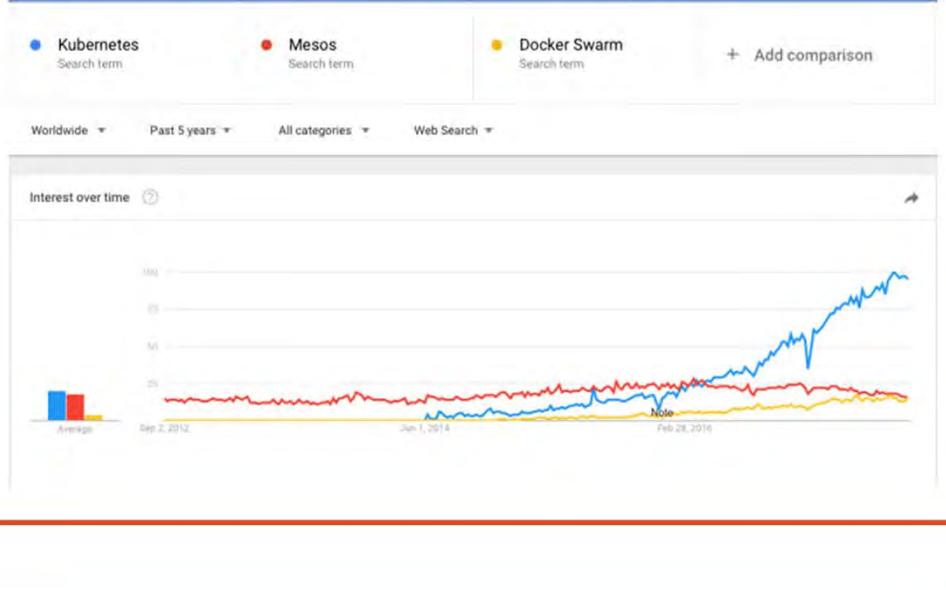
- How to scale?
- Once I scale, where are they?
- How do my containers find each other?
- How should I manage port conflicts?
- What if a host fails?
- How to update them? Health checks?
- How will I track their logs?



NOTES:

Container Orchestration Tools

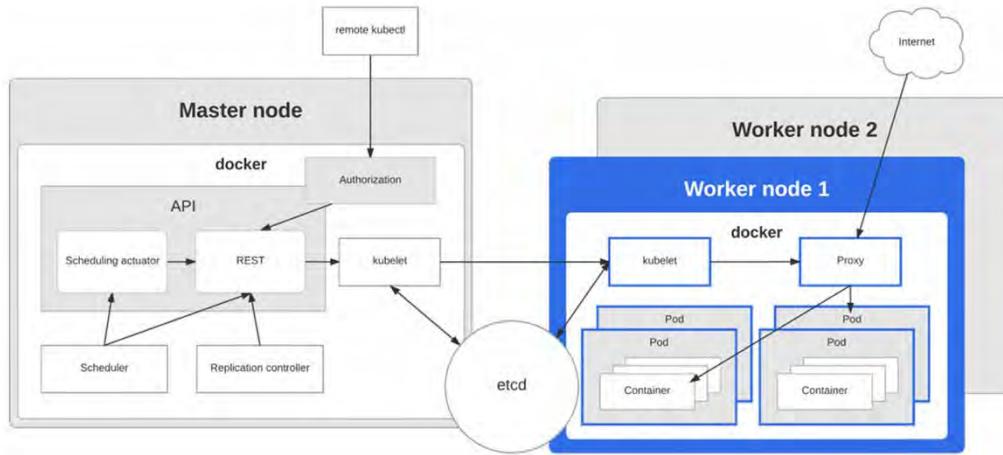
- The three most popular are: Kubernetes, Docker Swarm & Mesos
- Kubernetes has become the unofficial standard



Google started creation of its own open source orchestration tool ‘Kubernetes’. It has slowly gained dominance as the standard container orchestration tool except for niche use cases.

- Mesos
 - Still used in early container adoption shops and big data applications
- Docker Swarm
 - Lightweight and may be useful for rapid prototyping. With the ease of deploying Kubernetes to public cloud and greater feature set of Kubernetes this use case is increasingly shrinking.

Kubernetes Architecture



NOTES:

K8s Architecture – Networking view

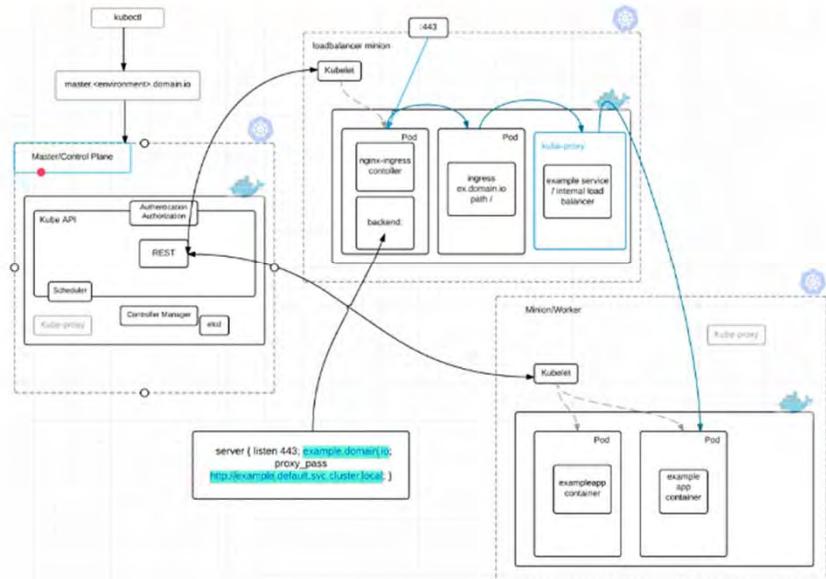


Image: <http://www.devoperandi.com/load-balancing-in-kubernetes/>



K8s Architecture – Simplified

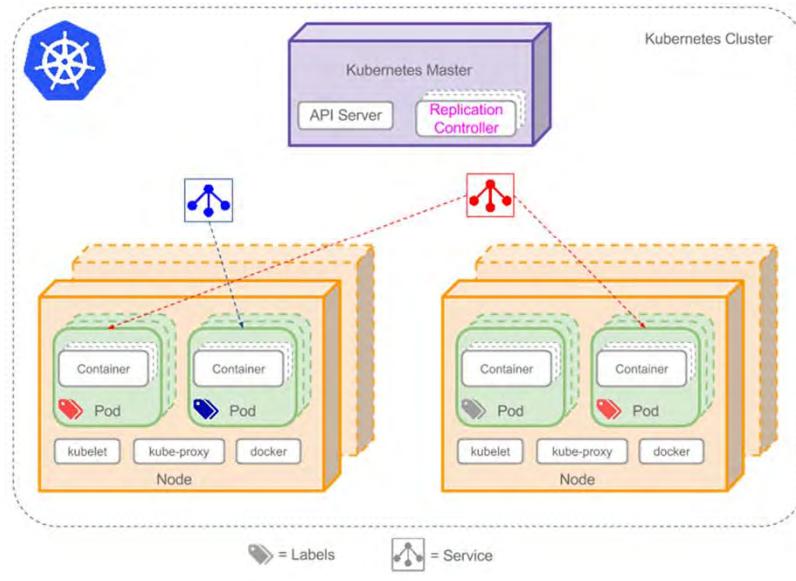
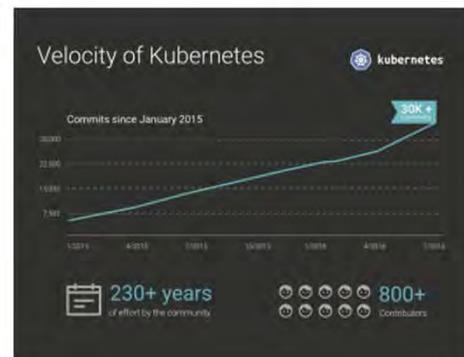


Image: <http://www.devoperandi.com/load-balancing-in-kubernetes/>

Kubernetes Stats

- Top 100 project by stars
- 22k LinkedIn professionals
- Largest container management ecosystem

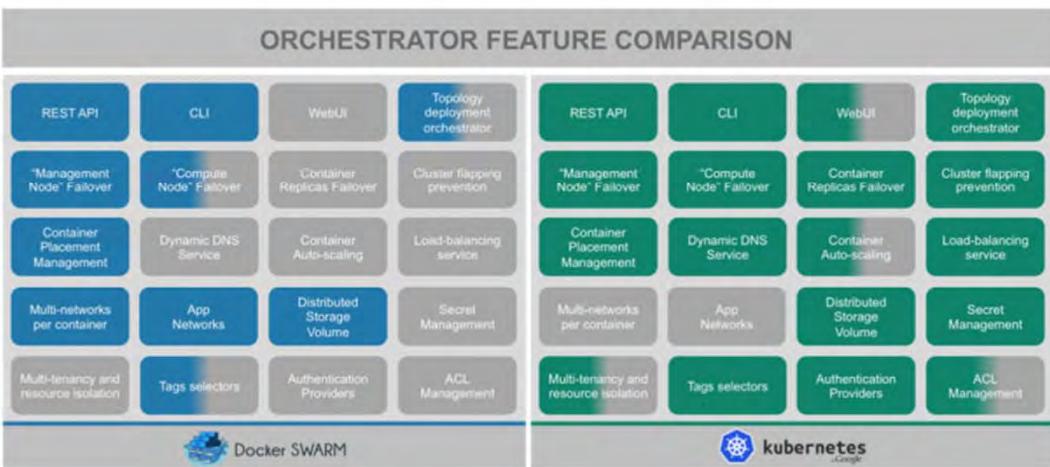
Github		
54k+ Commits	280+ Releases	1365+ Contributor s
Top 100 Forked Github Project	Top 2 Starred Go Project	Top 0.01% Starred Github Project



ASPE
TRAINING
a division of Fortis College

NOTES:

Feature Comparison (March 2016)



ASPE
TRAINING
a division of Fortis College

This is a rapidly changing patch work. This snapshot is of 03/2016.

(<https://blog.octo.com/docker-en-production-la-bataille-sanglante-des-orchestrateurs-de-conteneurs/>)

Since then Kubernetes has an improved WebUI, Autoscaling and Multi-tenancy features among others. It's a fast moving space!

Github

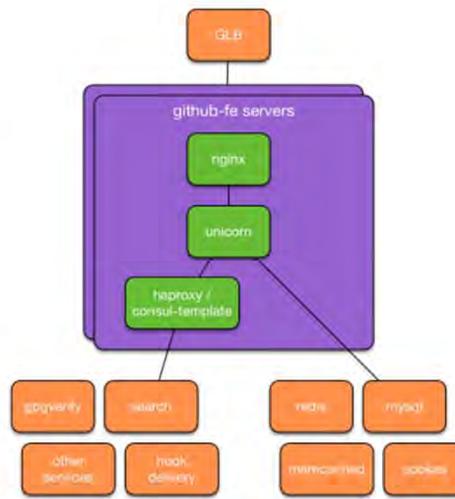


NOTES:

Github: Situation

Written in Ruby with Puppet managed servers, the Github legacy architecture had challenges

- As traffic grew, latency started to increase
- As more GitHubbers came on board, overhead increased
- New services began to take days, weeks or months to deploy
- Increasingly the SRE team began to spend a large percentage of their time on maintenance, not features



<https://githubengineering.com/kubernetes-at-github/>

"Ruby on Rails application (we call it `github/github`) was configured a lot like it was eight years ago: [Unicorn](#) processes managed by a Ruby process manager called [God](#) running on Puppet-managed servers. Similarly, our [chatops deployment](#) worked a lot like it did when it was first introduced: Capistrano established SSH connections to each frontend server, then [updated the code in place](#) and restarted application processes. When peak request load exceeded available frontend CPU capacity, GitHub Site Reliability Engineers would [provision additional capacity](#) and add it to the pool of active frontend servers."

Github: Situation

Over time, it became clear that this approach was not scaling. A joint team was formed to evaluate platforms against the following:

- Self Service platform for engineers to enable capacity
- Automate responding to changes in demand so changes can happen in seconds, not hours, days or more
- Insulate applications from differences between development, staging, production, enterprise and other environments

Several properties helped Kubernetes stand out from other platforms

- Vibrant open source community
- First run experience meant a small application was up within hours
- Wealth of information about the experience that motivated the design



<https://githubengineering.com/kubernetes-at-github/>

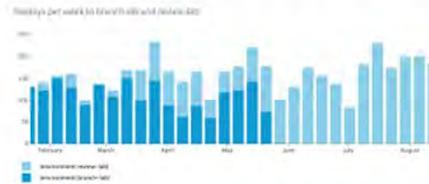
Github: Action

Before:

Deploy environments were booked solid during business hours, challenging engineers ability to test their pull requests

After:

Implemented capability for a chat-based interface for creating isolated deploys of Github for pull requests. This greatly enabled developer throughput as it empowered engineers to self service new environments with their Pull Requests.



jnewland
.deploy https://github.com/github/github/pull/4815162342 to review-lab



Hubot
@jnewland's review-lab deployment of github/add-pre-stop-hook (00cafefe) is done!
(12 ConfigMaps, 17 Deployments, 1 Ingress, 1 Namespace, 6 Secrets, and 23 Services)
(77.62s) your lab is available at https://jnewland.review-lab.github.com



Sub projects shipped:

A Kubernetes cluster running in an AWS VPC managed using a combination of [Terraform](#) and [kops](#).

A set of Bash integration tests that exercise ephemeral Kubernetes clusters, used heavily in the beginning of the project to gain confidence in Kubernetes.

A Dockerfile for github/github.

Enhancements to our internal CI platform to support building and publishing containers to a container registry.

YAML representations of 50+ Kubernetes resources, checked into github/github.

Enhancements to our internal deployment application to support deploying Kubernetes resources from a repository into a Kubernetes namespace, as well as the creation of Kubernetes secrets from our internal secret store.

A service that combines haproxy and consul-template to route traffic from Unicorn pods to the existing services that publish service information there.

A service that reads Kubernetes events and sends abnormal ones to our internal error tracking system.

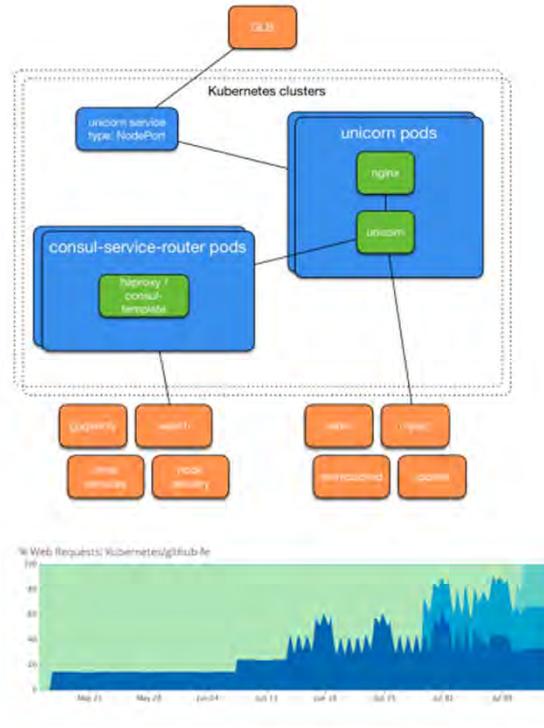
A [chatops-rpc](#)-compatible service called kube-me that exposes a limited set of kubectl commands to users via chat.

"

Github: Result

Building on their success with the previous Kubernetes deployment, Github began to migrate their front end to Kubernetes

- Migrated from Legacy system to a K8s cluster in AWS to finally a K8s cluster in their data center on bare metal
- Traffic was migrated to the Kubernetes system without incident
- The original goals of self service, responsiveness and scalable environments was achieved
- Engineers have rolled out dozens of projects in a self service manner, and Github's SRE's are focusing more on delivering infrastructure products.
- Github plans to start investigating and moving stateful services to K8s in the future.



<https://githubengineering.com/kubernetes-at-github/>

<https://news.ycombinator.com/item?id=15029086>

Subprojects in Github FE Deployment

"A timely and thorough post about [container networking](#) helped us select the [Calico](#) network provider, which provided the out-of-the box functionality we needed to ship a cluster quickly in ipip mode while giving us the flexibility to explore peering with our network infrastructure later.

Following no less than a dozen reads of [@kelseyhightower's indispensable Kubernetes the hard way](#), we assembled a handful of manually provisioned servers into a temporary Kubernetes cluster that passed the same set of integration tests we used to exercise our AWS clusters.

We built a small tool to generate the CA and configuration necessary for each cluster in a format that could be consumed by our internal Puppet and secret systems.

We Puppetized the configuration of two instance roles - Kubernetes nodes and Kubernetes apiservers - in a fashion that allows a user to provide the name of an already-configured cluster to join at provision time.

We built a small Go service to consume container logs, append metadata in key/value format to each line, and send them to the hosts' local syslog endpoint.

We enhanced [GLB](#), our internal load balancing service, to support Kubernetes NodePort Services."

Why Kubernetes?

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

- [Co-locating helper processes](#), facilitating composite applications and preserving the one-application-per-container model
- [Mounting storage systems](#)
- [Distributing secrets](#)
- [Checking application health](#)
- [Replicating application instances](#)
- [Using Horizontal Pod Autoscaling](#)
- [Naming and discovering](#)
- [Balancing loads](#)
- [Rolling updates](#)
- [Monitoring resources](#)
- [Accessing and ingesting logs](#)
- [Debugging applications](#)
- [Providing authentication and authorization](#)
- This provides the simplicity of Platform as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS), and facilitates portability across infrastructure providers.



What does *Kubernetes* mean? K8s? Pronounced "Kates"

The name **Kubernetes** originates from Greek, meaning *helmsman* or *pilot*, and is the root of *governor* and [cybernetic](#). K8s is an abbreviation derived by replacing the 8 letters "ubernete" with "8"

Master Components

- **kube-apiserver** - Component on the master that exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.
- **etcd** - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data
- **kube-scheduler** - Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on.
- **kube-controller-manager**
 - Node Controller: Responsible for noticing and responding when nodes go down.
 - Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
 - Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).
 - Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces.
- **cloud-controller-manager** - runs controllers that interact with the underlying cloud providers.



NOTES:

Node Components

- **kubelet** - An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
- **kube-proxy** - [kube-proxy](#) enables the Kubernetes service abstraction by maintaining network rules on the host and performing connection forwarding.
- **Container Runtime** - The container runtime is the software that is responsible for running containers. Kubernetes supports several runtimes: [Docker](#), [rkt](#), [runc](#) and any OCI [runtime-spec](#) implementation



NOTES:

Addons

- **DNS** - While the other addons are not strictly required, all Kubernetes clusters should have [cluster DNS](#), as many examples rely on it.
- **Web UI (Dashboard)** - [Dashboard](#) is a general purpose, web-based UI for Kubernetes clusters.
- **Container Resource Monitoring** - [Container Resource Monitoring](#) records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.
- **Cluster-level Logging** - A [Cluster-level logging](#) mechanism is responsible for saving container logs to a central log store with search/browsing interface.



NOTES:

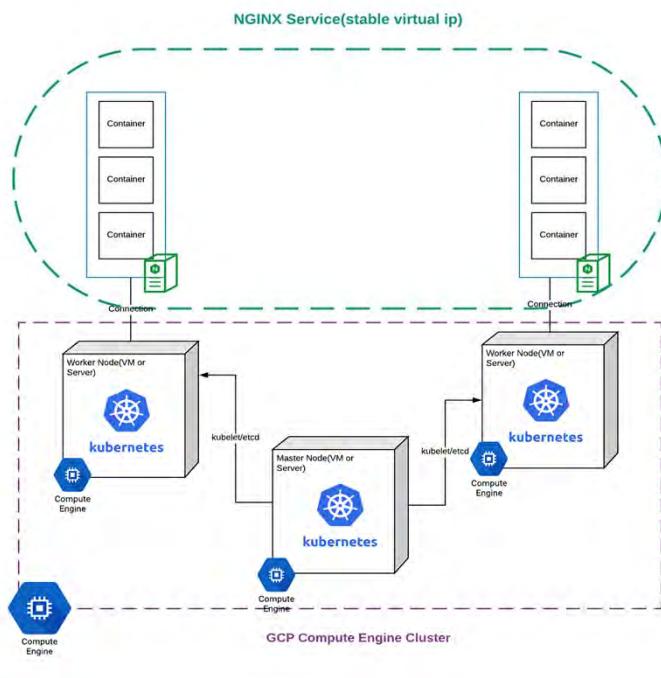
Kubernetes Terms

- Containers**
- Nodes**
- Clusters**
- Deployment**
- Pods**
- ReplicaSets**
- Services**
- Labels**
- Volumes**



NOTES:

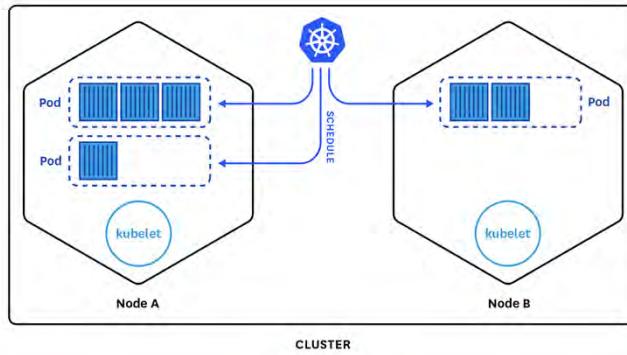
Kubernetes: Abstraction



NOTES:

Kubernetes: Nodes

A node is a worker machine in Kubernetes. A node may be a VM or physical machine, depending on the cluster. Each node has the services necessary to run pods and is managed by the master components. The services on a node include Docker, kubelet and kube-proxy. See The Kubernetes Node section in the architecture design doc for more details.

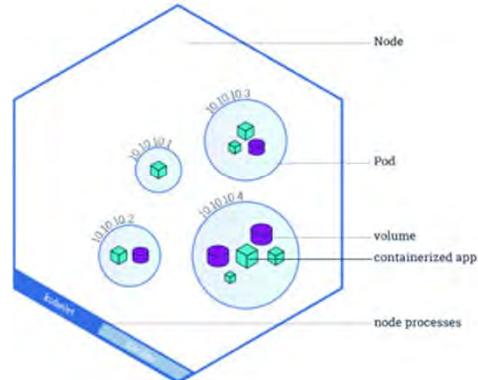


NOTES:

Kubernetes: Nodes

Allows Pods to be scheduled. A basic worker physical or virtual machine of Kubernetes.

- Must be managed by a master
- May host multiple pods
- Internal IP Address endpoint
- Can be tagged and filtered using labels
- Must be running two processes
 - Kubelet – Kubernetes client process. Communicates with master and schedules pods
 - Container runtime – Docker, Rocket, etc. Pulls and runs containers from registry



NOTES:

Kubernetes: Nodes

Node Status

A node status contains:

- ◆ Addresses
 - ◆ Hostname
 - ◆ ExternalIP
 - ◆ InternalIP
- ◆ Condition - describe condition of nodes.
- ◆ Capacity - describe the resources available on the node
- ◆ Info - general information about the node



NOTES:

\$ kubectl get nodes

```
root@Azure:~$ kubectl get nodes
NAME           STATUS    AGE     VERSION
k8s-agent-743d5653-0   Ready    21h    v1.6.6
k8s-agent-743d5653-1   Ready    21h    v1.6.6
k8s-agent-743d5653-2   Ready    21h    v1.6.6
k8s-master-743d5653-0 Ready,SchedulingDisabled 21h    v1.6.6
```

Note: Affinity & Anti-affinity (Beta) allow further constraints on where a pod may run (coupled, de-coupled, only nodes with certain labels, etc)



1. Kubectl get nodes

As long as you have a working cluster and connection to the cluster this command should work. This is usually one of the first commands you call to ensure the cluster is alive.

<http://kubernetesbyexample.com/nodes/>

Kubernetes: Clusters

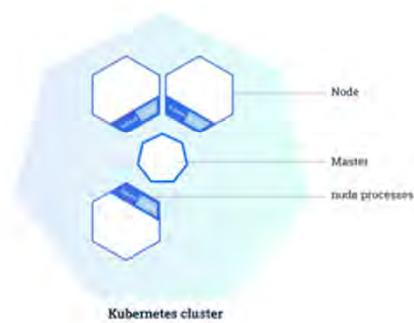
Allows you to deploy containerized applications to a cluster without tying them specifically to individual machines.

Masters

- Coordinates the cluster
 - Schedules applications
 - Maintains application state
 - Scales applications
 - Rolls out updates
- Can be duplicated when HA (High Availability) is desired.
- Often run as a service by public cloud providers (Azure/Google Kubernetes as a Service)

Nodes

- House workers that run the applications
- Three node minimum for production clusters

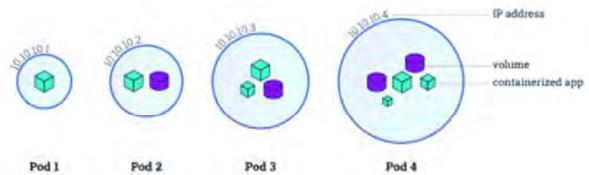


NOTES:

Kubernetes: Pods

Allows you to create a group of one or more containers and some shared resources for those containers

- Pod Resource Types
 - Networking, as a unique cluster IP address
 - Container information:
 - container image version
 - specific ports
 - Etc
- Atomic unit of Kubernetes
- Tied to node which it is scheduled
- Always co-located and co-scheduled
- Containers in a pod share
 - IP address and port space
 - Filesystem
 - Storage (Volumes)
 - Labels
 - Secrets

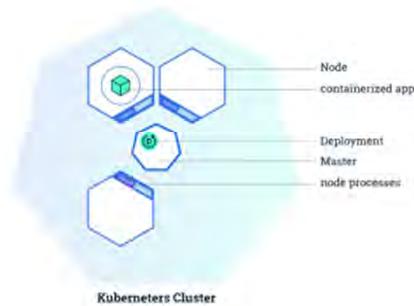


Additionally, pods can be tagged with a label for recommended scheduling advice

Kubernetes: Deployments

Allows you to deploy a (self-healing) instance of an application.

- **Self Healing:** Continuously monitors and replaces instances if necessary
- **Provides declarative updates for Pods and ReplicaSets**
 - Updates actual state to desired state at a controlled rate
 - For example: Current state, 3 instances of Tomcat. Desired state, 5 instances of Tomcat. → Create 2 more instances of Tomcat
- **Relay on declarative manifest to determine intent (i.e. yaml/json file most commonly)**



Deployments (and controllers) are loops that maintain state.

\$ kubectl create

```
peter@Azure:~$ cat pod.yaml
# 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
peter@Azure:~$ kubectl create -f pod.yaml
pod "nginx" created
peter@Azure:~$ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
nginx    1/1      Running   0          17s
```

Note: This is just creation of the pod. In order to expose it, a service must be created. In order for it to be respawned, a deployment or replication control must be created.



1. Create a pod.yaml with the following contents

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

2. Kubectl create -f pod.yaml

3. Kubectl get pods

This highlights an example of Kubernetes deployment creation. As with many Kubernetes objects, deployments are usually created from a file such as a yaml or json. While this could be done via command line it is often impractical to do so.

Further reading:

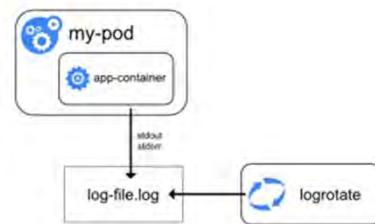
<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#creating-a-deployment>

Note: This is a simple pod. This is interesting.

\$ kubectl get pods

```
user@k8s: ~ $ kubectl get pods      READY   STATUS    RESTARTS   AGE
meme-vote-back-3048739398-dsklm   1/1     Running   0          9s
meme-vote-front-3379644005-j9jh5  1/1     Running   0          9s
user@k8s: ~ $ kubectl logs meme-vote-back-3048739398-dsklm
1:0 06 Sep 12:24:18.141 # Redis version=4.0.1, bits=64, commit=00000000
1:0 06 Sep 12:24:18.141 # Redis is running on port 6379.
1:0 06 Sep 12:24:18.143 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
1:0 06 Sep 12:24:18.143 # Server initialized
1:0 06 Sep 12:24:18.143 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
1:0 06 Sep 12:24:18.143 # HZ=250 you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' or 'echo never > /proc/sys/vm/transp
1:0 06 Sep 12:24:18.143 * Ready to accept connections
```

Note: Logs from pods are readily available with a kubectl logs <podname> command.



This highlights an example of listing Kubernetes pods and also getting the logs from said pod.

Logging: more detail

<https://kubernetes.io/docs/concepts/cluster-administration/logging/>

Kubernetes: ReplicaSets

A **ReplicaSet** ensures that a specified number of pod replicas are running at any other time. If there are too many pods, the ReplicationController terminates the extra pods. If there are too few, the ReplicationController starts more pods. Unlike manually created pods, the pods maintained by a ReplicationController are automatically replaced if they fail, are deleted, or are terminated.

For example, your pods are re-created on a node after disruptive maintenance such as a kernel upgrade. For this reason, you should use a ReplicationController even if your application requires only a single pod. A ReplicationController is similar to a process supervisor, but instead of supervising individual processes on a single node, the ReplicationController supervises multiple pods across multiple nodes.



NOTES:

Kubernetes: Service

Kubernetes **Pods** are mortal. They are born and when they die, they are not resurrected. **ReplicaSets** in particular create and destroy **Pods** dynamically (e.g. when scaling up or down). While each **Pod** gets its own IP address, even those IP addresses cannot be relied upon to be stable over time. This leads to a problem: if some set of **Pods** (let's call them backends) provides functionality to other **Pods** (let's call them frontends) inside the Kubernetes cluster, how do those frontends find out and keep track of which backends are in that set?

A Kubernetes **Service** is an abstraction which defines a logical set of **Pods** and a policy by which to access them - sometimes called a micro-service. The set of **Pods** targeted by a **Service** is (usually) determined by a **Label Selector** (see below for why you might want a **Service** without a selector).

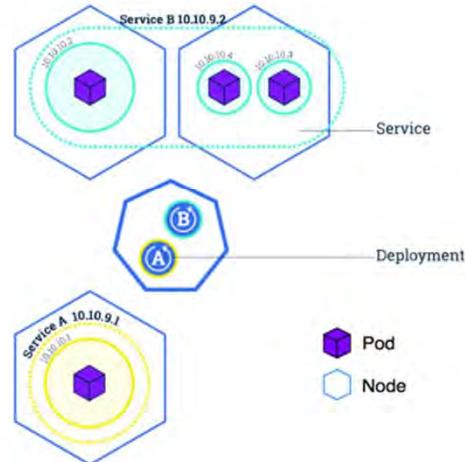


NOTES:

Kubernetes: Service

Abstraction regarding a set of pods which enables load balancing, traffic exposure, load balancing and service discovery.

- Abstraction that allows pods to die and replicate in Kubernetes without affecting your application.
- Enable loose coupling between different Pods.
- Provides a **stable** virtual IP and port
- Services allow Pods to receive traffic.
 - Each Pod has a unique IP but those IP addresses are not exposed outside the Pod without a service.



NOTES:

\$ kubectl get services

```
root@k8s-node1:~$ kubectl get services --all-namespaces
```

NAMESPACE	NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	kubernetes	10.0.0.1	<none>	443/TCP	22h
kube-system	heapster	10.0.74.107	<none>	80/TCP	22h
kube-system	kube-dns	10.0.0.10	<none>	53/UDP, 53/TCP	22h
kube-system	kubernetes-dashboard	10.0.69.57	<nodes>	80:31476/TCP	22h
kube-system	tiller-deploy	10.0.103.65	<none>	44134/TCP	22h

Note: A component called ‘kube-proxy’ runs on each node and implements a form of virtual IP for services of type other than ‘ExternalName’



1. kubectl get services --all-namespaces

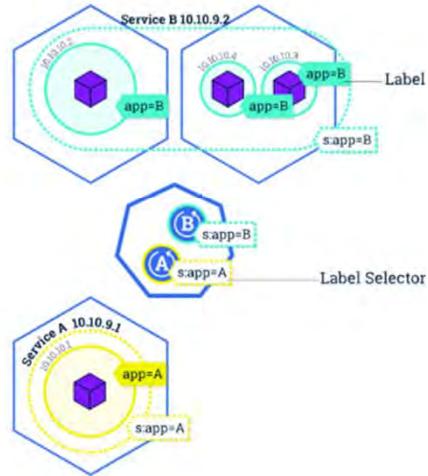
This reveals the Kubernetes services that are usually running as part of the Kubernetes system namespace.

<http://kubernetesbyexample.com/services/>

Kubernetes: Label

Key/Value pairs that can be attached to objects to enable a variety of use cases.

- Designate objects for specific environments such as development, test and production
- Set versions to objects
- Set roles or other arbitrary information to classify objects
- Can be added or modified at any time



NOTES:

Kubernetes Command Line Tool: kubectl

Most common access point for launching workloads on Kubernetes clusters.

- **kubectl [command] [TYPE] [NAME] [flags]**
- Very popular commands
 - **kubectl get** - list resources about a target.
 - kubectl get services
 - kubectl get pods
 - **kubectl describe** - show basic information about a resource
 - Kubectl describe pods my-pod
 - **kubectl logs** - print the logs from a container in a pod.
Extremely useful
 - Kubectl logs my-pod
- Also useful:
 - **kubectl exec** - execute a command on a container in a pod
 - **Kubectl top** – Show metrics for a node



Trivia: Some internal Kubernetes engineers pronounce kubectl as 'kubecuttle'

\$ kubectl get pod -l name=<label>

```
peter@Azure:~$ kubectl run nginx --image nginx --port 80
deployment "nginx" created
peter@Azure:~$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
nginx-158599303-04h8d  1/1     Running   0          4s
peter@Azure:~$ kubectl label pods nginx-158599303-04h8d new-label=awesome
pod "nginx-158599303-04h8d" labeled
peter@Azure:~$ kubectl get pods -l new-label=else
No resources found.
peter@Azure:~$ kubectl get pods -l new-label=awesome
NAME          READY   STATUS    RESTARTS   AGE
nginx-158599303-04h8d  1/1     Running   0          3m
peter@Azure:~$ kubectl get pods --show-labels
NAME          READY   STATUS    RESTARTS   AGE   LABELS
nginx-158599303-04h8d  1/1     Running   0          3m   new-label=awesome,pod-template-hash=158599303,run=nginx
```

Note: Best mechanism to organize Kubernetes objects



1. kubectl run nginx --image nginx --port 80
2. kubectl get pods
3. kubectl label pods nginx-158599303-04h8d new-label=awesome
4. kubectl get pods -l new-label=else
5. kubectl get pods -l new-label=awesome
6. kubectl get pods --show-labels

<http://kubernetesbyexample.com/labels/>



Kubernetes Connect

CLASSROOM WORK

Kubernetes Simple App Deployment

https://github.com/smhillin/intro_to_kubernetes/blob/master/Kubernetes-Simple-Example.md

1. Create 3 Node Kubernetes cluster
2. Create and upload docker image
3. Stand Up app on cluster
4. View live app in browser



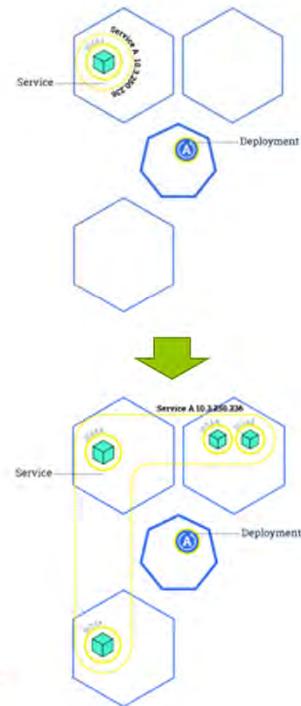
1. Create a cluster (or reuse a cluster from earlier exercises) and connect to it
2. Execute the nginx example
 1. <https://github.com/kubernetes/kubernetes/blob/master/examples/simple-nginx.md>

Some repetition here is intentional. You should be comfortable creating clusters, establishing a connection, verifying its healthy, and then cleaning up resources. This foundation is essential before moving on to advanced concepts

Kubernetes: Scaling

Changing the number of resources to meet a desired state

- Accomplished by adjusting the number of replicas in a deployment
- Accommodates both scaling up and scaling down to a minimum of 0
- Traffic is automatically sent to newly created instances through the service load balancer
- Can be used to enable rolling updates without downtime



NOTES:



Scaling Up Applications

CLASSROOM WORK

Kubernetes Scale Up Simple App

https://github.com/smhillin/intro_to_kubernetes/blob/master/Scale-Up-Simple-Example.md

1. Scale up simple app
2. Run a new deployment with ReplicaSet
3. Check for new replicas



<https://github.com/gravitational/workshop/blob/master/k8s101.md>

Git clone <https://github.com/gravitational/workshop.git>

Sections:

- Running nginx
- Pod IPs
- Pod Containers

Note: Make sure to substitute the specific names and IPs for the ones visible to you. I.e. my-nginx-3800858182-uuusv is the name in the example, but for you the name from 'kubectl get pods' should be used

\$ kubectl scale deployments/<deployment> --replicas=<new num>

```
peter@Azure:~$ kubectl run nginx --image nginx --port 80
deployment "nginx" created
peter@Azure:~$ 
peter@Azure:~$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     1          1          1           1          39s
peter@Azure:~$ kubectl scale deployments/nginx --replicas=2
deployment "nginx" scaled
peter@Azure:~$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     2          2          2           1          1m
```

Note: Make sure to delete the deployments or replicaset when trying to clear out pods. Many a new user has repeated deleted pods and gotten frustrated when they respawn (as the deployments/replicaset are programmed to do).



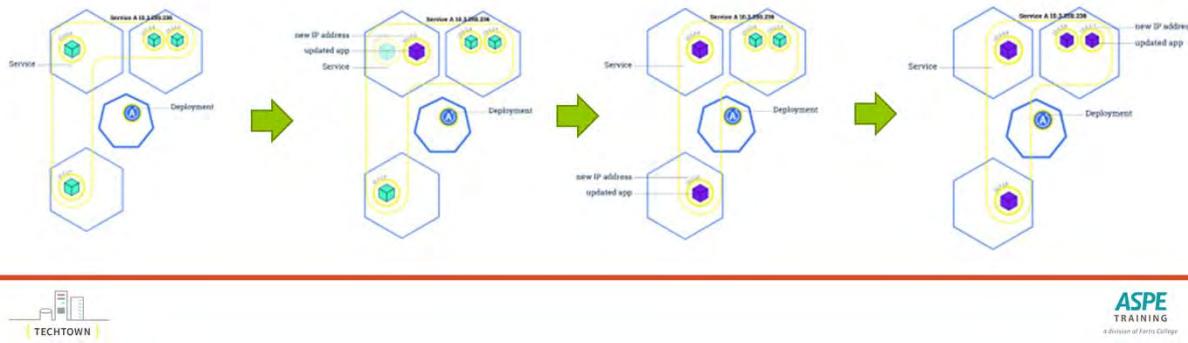
1. kubectl run nginx –image nginx –port 80
1. kubectl get deployments
1. kubectl scale deployments/nginx –replicas=2
1. kubectl get deployments

<http://kubernetesbyexample.com/labels/>

Kubernetes: Rolling Update

Updates the pods in a serial manner will load balancing traffic to available instances

- If deployment is exposed publicly, the service will load-balance traffic to only active and available pods
- Used to enable zero downtime updates
- Allows greater application update frequency
- Can also be leveraged to rollback to previous versions



NOTES:

\$ kubectl set image <deployment> <new-image>

```
peter@Asure:~$ kubectl run nginx --image nginx:1.12.1 --port 80
deployment "nginx" created
peter@Asure:~$ kubectl set image deployments/nginx nginx=nginx:latest
deployment "nginx" image updated
peter@Asure:~$ kubectl describe deployment nginx
Name:           nginx
Namespace:      default
CreationTimestamp: Fri, 15 Sep 2017 13:08:50 +0000
Labels:          run=nginx
Annotations:    deployment.kubernetes.io/revision=2
Selector:        run=nginx
Replicas:       1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 1 max surge
Pod Template:
  Labels:       run=nginx
  Containers:
    nginx:
      Image:      nginx:latest
      Port:       80/TCP
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
  Conditions:
    Type     Status  Reason
    ----   -----  -----
    Available  True    MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-2688028062 (1/1 replicas created)
Events:
  FirstSeen  LastSeen  Count  From            SubObjectPath  Type      Reason                Message
  ----       ----       ---   ---            ---          ---      ---
  0s         32s       1    deployment-controller          Normal    ScalingReplicaSet  Scaled up replicas set nginx-1295584306 to 1
  10s        10s       1    deployment-controller          Normal    ScalingReplicaSet  Scaled up replicas set nginx-2688028062 to 1
  10s        10s       1    deployment-controller          Normal    ScalingReplicaSet  Scaled down replicas set nginx-1295584306 to 0
```

Note: Rolling updates can also be undone (see next slide)



Stand up nginx but with an older version, 1.12.1

1. kubectl run nginx --image nginx:1.12.1 --port 80

Set image to the newer version (latest). This triggers kubernetes to begin a rolling update

2. kubectl set image deployments/nginx nginx=nginx:latest

Observe the newer image has been rolled out

3. kubectl describe deployment nginx

\$ kubectl rollout undo <deployment>

```
peter@Azure:~$ kubectl rollout undo deployments/nginx
deployment "nginx" rolled back
peter@Azure:~$ kubectl describe deployments/nginx
Name:           nginx
Namespace:      default
CreationTimestamp: Fri, 15 Sep 2017 13:36:49 +0000
Labels:          run=nginx
Annotations:    deployment.kubernetes.io/revision=3
Selector:        run=nginx
Replicas:       1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 1 max surge
Pod Template:
  Labels:  run=nginx
  Containers:
    nginx:
      Image:      nginx:1.12.1
      Port:       80/TCP
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
  Conditions:
    Type  Status  Reason
    ----  ----  -----
    Available  True  MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-1295584306 (1/1 replicas created)
Events:
FirstSeen  LastSeen  Count  From            SubObjectPath  Type    Reason          Message
-----  -----  ----  ----  -----  ----  -----
1m         1m        1  deployment-controller  Normal  ScalingReplicaSet  Scaled up replica set nginx-2688028062 to 1
1m         1m        1  deployment-controller  Normal  ScalingReplicaSet  Scaled down replica set nginx-1295584306 to 0
1m         10s       2  deployment-controller  Normal  ScalingReplicaSet  Scaled up replica set nginx-1295584306 to 1
10s        10s       1  deployment-controller  Normal  DeploymentRollback  Rolled back deployment "nginx" to revision 1
10s        10s       1  deployment-controller  Normal  ScalingReplicaSet  Scaled down replica set nginx-2688028062 to 0
```

Note: Rollbacks can also be also enacted with zero-downtime



Roll back to previous version nginx 1.12.1

4. kubectl rollout undo deployments/nginx

Observe changes

5. kubectl describe deployments/nginx



New Version Updates

CLASSROOM WORK

Kubernetes Rolling Updates

https://github.com/smhillin/intro_to_kubernetes/blob/master/Deploy-New-Version-Simple-Example.md

1. Create version 2 of simple app
2. Create new docker Image
3. Apply rolling update
4. Delete the Service



<https://github.com/gravitational/workshop/blob/master/k8s101.md>

Git clone <https://github.com/gravitational/workshop.git>

Sections:

- Deployments and Replicasets
- Services
- Back to Deployments
- Configuration management basics

Kubernetes at Box



NOTES:

Box: Situation



"In the summer of 2014, Box was feeling the pain of a decade's worth of hardware and software infrastructure that wasn't keeping up with the company's needs"

- 50 Million users, including several large enterprises such as General Electric
- Originally a PHP monolith of millions of lines of code
- Primarily hosted on bare metal within their own data centers
- Early adopter of Puppet representing one of its largest implementations
- Customer pressure to host workloads and data on every major cloud
- Was taking more than six months to deploy a new microservice



NOTES:

Box: Action

Box was an early adopter in Kubernetes. Working with the open source and contributing to the very early versions.

- Was initially attracted to the [design documentation](#) and influence from Google's experience with Borg, a bare metal container orchestration platform
- Also enjoyed the shared API primitives such as pods, services, and deployments that other PaaS layers such as OpenShift and Deis were building on top of.
- "There was clearly a pent-up demand for a better way of building software through microservices, and the increase in agility helped our developers be more productive and make better architectural choices."



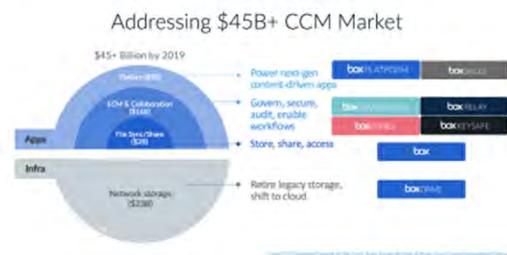
Full quote:

While observing teams who began to use Kubernetes for their microservices, "we immediately saw an uptick in the number of microservices being released," Ghods notes. "There was clearly a pent-up demand for a better way of building software through microservices, and the increase in agility helped our developers be more productive and make better architectural choices."

Box: Result

Kubernetes at Box represented a shift towards cloud native development techniques

- "Kubernetes, and cloud native in general, represents a pretty big paradigm shift, and it's not very incremental"
 - "We're essentially making this pitch that Kubernetes is going to solve everything because it does things the right way and everything is just suddenly better"
 - Microservices have decreased to taking less than five days to deploy (from six months), and have the goal to getting that number under an hour



"Ghods predicts that Kubernetes has the opportunity to be the new cloud platform. '...because it's a never-before-seen level of automation and intelligence surrounding infrastructure that is portable and agnostic to every way you can run your infrastructure.'"

<https://kubernetes.io/case-studies/box/>

<https://news.ycombinator.com/item?id=12145423>

<https://cnkc16.sched.com/event/8K8u/how-box-runs-containers-in-production-with-kubernetes-sam-ghods-box>

<https://architect.io/box-co-founder-on-moving-to-microservices-and-the-promise-of-kubernetes-a49f01b1c0c0>

Kubernetes cheat sheet

Creating Objects

Kubernetes manifests can be defined in json or yaml. The file extension .yaml, .yml, and .json can be used.

```
# kubectl create -f ./my-manifest.yaml      # create resource(s)
# kubectl create -f ./my1.yaml -f ./my2.yaml  # create from multiple files
# kubectl create -f ./dir                   # create resource(s) in all manifest files in dir
# kubectl create -f https://git.io/vPee     # create resource(s) from url
# kubectl run nginx --image=nginx          # start a single instance of nginx
# kubectl explain pods,svc                # get the documentation for pod and svc manifests
```

Viewing, Finding Resources

```
# Get commands with basic output
# kubectl get services                      # List all services in the namespace
# kubectl get pods --all-namespaces          # List all pods in all namespaces
# kubectl get pods -o wide                  # List all pods in the namespace, with more details
# kubectl get deployment my-dep             # List a particular deployment

# Describe commands with verbose output
# kubectl describe nodes my-node            # Describe a node named "my-node"
# kubectl describe pods my-pod              # Describe a pod named "my-pod"

# kubectl get services --sort-by=.metadata.name # List Services Sorted by Name
```

Deleting Resources

```
# kubectl delete -f ./pod.json               # Delete a pod using the type and name specified in pod.json
# kubectl delete pod,service baz foo          # Delete pods and services with same names "baz" and "foo"
# kubectl delete pods,services -l name=myLabel # Delete pods and services with label name=myLabel
# kubectl -n my-ns delete po,svc --all         # Delete all pods and services in namespace my-ns
```



Kubectl cheat sheet (collection of useful commands):

<https://kubernetes.io/docs/user-guide/kubectl-cheatsheet/>

Kubernetes: Volumes

On-disk files in a container are the simplest place for an application to write data, but this approach has drawbacks. The files are lost when the container crashes or stops for any other reason. Furthermore, files within a container are inaccessible to other containers running in the same [Pod](#). The [Kubernetes Volume](#) abstraction addresses both of these issues.

- A directory accessible to all containers in a pod
- Volumes out live containers that run within a pod
- May be passed between pods
- Data is preserved across container restarts



NOTES:

Kubernetes: Volumes

```
peter@Azure:~$ kubectl exec sharevol -c c1 -i -t -- bash
[root@sharevol /]# mount | grep xchange
/dev/sda1 on /tmp/xchange type ext4 (rw,relatime,discard,data=ordered)
[root@sharevol /]# echo 'some data' > /tmp/xchange/data
[root@sharevol /]# exit
exit
peter@Azure:~$ kubectl exec sharevol -c c2 -i -t -- bash
[root@sharevol /]# mount | grep /tmp/data
/dev/sda1 on /tmp/data type ext4 (rw,relatime,discard,data=ordered)
[root@sharevol /]# cat /tmp/data/data
some data
[root@sharevol /]# █
```

Note: Volumes are based on a wide assortment of underlying storage providers. Every major public cloud (including Azure) has several options for volume storage



1. `kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/volumes/pod.yaml`

Shell into the first pod

2. `kubectl exec sharevol -c c1 -i -t -- bash`

Write data to an 'xchange' file

3. `mount | grep xchange`

4. `echo 'some data' > /tmp/xchange/data`

Exit out of container

5. `exit`

Now shell into the 2nd container

6. `kubectl exec sharevol -c c2 -i -t -- bash`

7. `mount | grep /tmp/data`

8. `cat /tmp/data/data`

View the whole pod.yaml file here: <https://github.com/openshift-evangelists/kbe/blob/master/specs/volumes/pod.yaml>

Kubernetes by example Volumes: <http://kubernetesbyexample.com/volumes/>

Kubernetes: Persistent Volumes

Persistent storage in a cluster

Managing storage is a distinct problem from managing compute. The PersistentVolume subsystem provides an API for users and administrators that abstracts details of how storage is provided from how it is consumed.

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.



NOTES:

Kubernetes: Persistent Volumes

Persistent storage in a cluster

- Separate to **PersistentVolumeClaim**, which is a request for storage.
- May be created ahead of time in a static manner for use by a cluster
- May be created dynamically from available, unclaimed resources
- May be freed and passed from user to user so care should be taken to delete contents when done with use
- Critical for Stateful containers



NOTES:



Kubernetes Volumes

CLASSROOM WORK

Kubernetes Volumes

https://github.com/smhillin/intro_to_kubernetes/blob/master/creating-volume.md

1. Create pods and shared volume
2. Save data to volume
3. Mount volume
4. Read from shared pod



<https://github.com/gravitational/workshop/blob/master/k8s101.md>

Git clone <https://github.com/gravitational/workshop.git>

Sections:

- Deployments and Replicasets
- Services
- Back to Deployments
- Configuration management basics

Introduction to Kubernetes

Part 2:



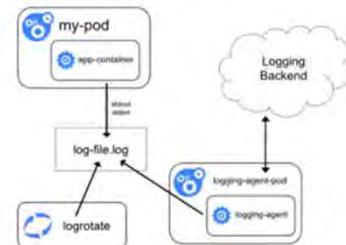
NOTES:

Kubernetes: Logging

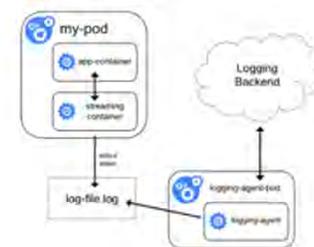
Aggregate, access and print logs from containers

- Logs can be accessed from a currently run container or any number of previously run containers, accessed by container name
- Containerized apps write logs to stdout and stderr
- Cluster level logging can be performed using several different techniques including node level agents, container sidecars, or even having containers push directly to an application
- Node level logging is the most common approach, typically using Elasticsearch.

Using a node logging agent



Streaming sidecar container



Print Logs

Kubectl logs logme

Stream logs

Kubectl logs -f logme



Kubernetes Logging

CLASSROOM WORK

Kubernetes Logging

https://github.com/smhillin/intro_to_kubernetes/blob/master/kubernetes-logging.md

1. Create a logging pd
2. View the logs
3. Create oneshot pod



<https://github.com/gravitational/workshop/blob/master/k8s101.md>

Git clone <https://github.com/gravitational/workshop.git>

Sections:

- Deployments and Replicases
- Services
- Back to Deployments
- Configuration management basics

Kubernetes: Logging

```
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/pod.yaml
W1009 12:03:08.9702      121 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling back to swagger
pod "logme" created
peter@Azure:~$ kubectl logs --tail=5 logme -c gen
Mon Oct 9 12:28:18 UTC 2017
Mon Oct 9 12:28:19 UTC 2017
Mon Oct 9 12:28:19 UTC 2017
Mon Oct 9 12:28:20 UTC 2017
Mon Oct 9 12:28:20 UTC 2017
peter@Azure:~$ kubectl logs -f --since=10s logme -c gen

peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/oneshotpod.yaml
W1009 12:46:45.895922      139 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling back to swagger
pod "oneshot" created
peter@Azure:~$ kubectl logs -p oneshot -c gen
9
8
7
6
5
4
3
2
1
```

Note: An external system can perform the log rotation by calling logrotate, which would result in kubectl logs returning an empty result



Create a logging pod

1. kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/pod.yaml

Grab the last few lines of logs

2. kubectl logs --tail=5 logme -c gen

Grab a stream of the logs

3. kubectl logs -f --since=10s logme -c gen

Clean up

4. kubectl delete -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/pod.yaml

Now launch a container that performs an action and then exits. Give it a minute or so to close out

5. kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/oneshotpod.yaml

Now output the logs of this container

6. kubectl logs -p oneshot -c gen

Clean up

7. kubectl delete -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/oneshotpod.yaml

<http://kubernetesbyexample.com/logging/>

Elasticsearch on K8s



CLASSROOM WORK

Kubernetes Elasticsearch Deployment

https://github.com/smhillin/intro_to_kubernetes/blob/master/kubernetes-elasticsearch-repo.md

1. Clone the examples repo
2. View the logs
3. Create oneshot pod



<https://github.com/gravitational/workshop/blob/master/k8s101.md>

Git clone <https://github.com/gravitational/workshop.git>

Sections:

- Deployments and Replicasets
- Services
- Back to Deployments
- Configuration management basics

Kubernetes: Stateful Sets

Stateful Sets manage the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.

Like a Deployment, a **Stateful Set** manages Pods that are based on an identical container spec.

Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

A StatefulSet operates under the same pattern as any other Controller. You define your desired state in a StatefulSet object, and the StatefulSet controller makes any necessary updates to get there from the current state.



<https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>

Kubernetes: Stateful Sets

StatefulSets are valuable for applications that require one or more of the following:

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, graceful deletion and termination.
- Ordered, automated rolling updates.

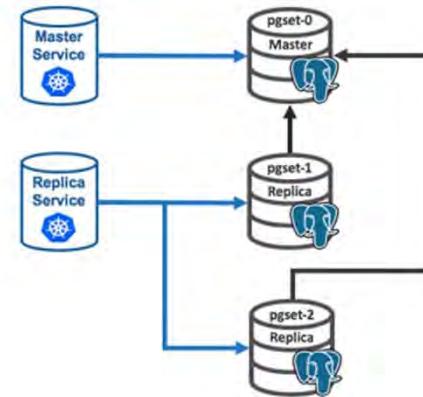


<https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>

Kubernetes: Stateful Sets

Functionality of deployments with guarantees about ordering and unique identities per Pod.

- Pods in a stateful set are not interchangeable
- Each Pod has a persistent identifier
- Define desired state object, controller performs updates to desired state
- Useful for applications that require persistent storage, unique network identifiers, ordered deployment, deletes or updates.

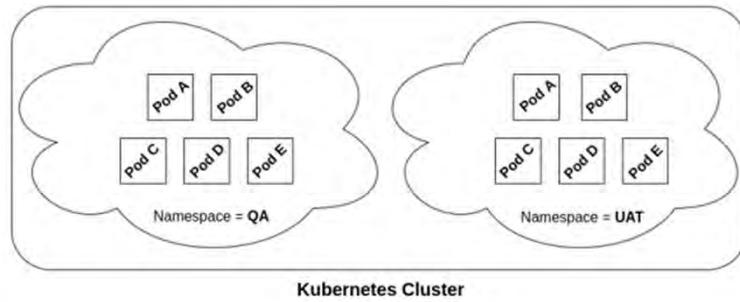


<https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>

Kubernetes: Namespace

Partitions the names of API objects

- Resources are segmented within namespaces
 - Sandbox per developer for example
- Pods will route dns to default (own) namespace
 - api.default.svc.cluster.local
 - Api.foo...
- The same app could run independently in different namespaces



NOTES:

Kubernetes: Namespace

```
peter@Azure:~$ kubectl get ns
NAME      STATUS   AGE
default   Active   23d
kube-public   Active   23d
kube-system   Active   23d
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/ns/ns.yaml
W1006 13:59:41.630630    134 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling back to swagger
namespace "test" created
peter@Azure:~$ kubectl get ns
NAME      STATUS   AGE
default   Active   23d
kube-public   Active   23d
kube-system   Active   23d
test      Active   19s
peter@Azure:~$ kubectl create --namespace=test -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/ns/pod.yaml
W1006 14:00:18.027941    145 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling back to swagger
pod "podintest" created
peter@Azure:~$ kubectl get pods --namespace=test
NAME        READY   STATUS    RESTARTS   AGE
podintest   0/1     ContainerCreating   0          11s
```

Note: Namespaces create an excellent mechanism to subdivide resources and provide isolation. This enables using a cluster for multiple projects or multiple teams



Get default namespaces

1. kubectl get ns

Create a 'test' namespace

2. kubectl create -f

<https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/ns/ns.yaml>

Recheck namespaces and observe

3. kubectl get ns

Now create a pod in this new namespace

4. kubectl create --namespace=test -f

<https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/ns/pod.yaml>

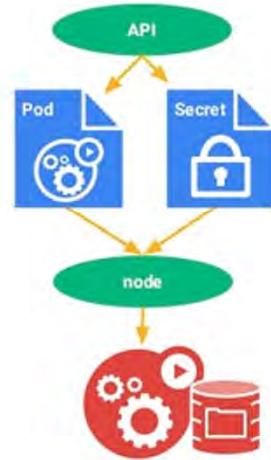
Check pods in this new namespace

5. kubectl get pods --namespace=test

Kubernetes: Secrets

An object that contains a small amount of sensitive data such as a password, token or key.

- According to [12-factor](#) configuration comes from the environment
 - Config is everything that is likely to vary between deployments
- Can be used by pods and the underlying Kubelet when pulling images
- Secrets belong to a specific Kubernetes Namespace
- Secret size cannot exceed 1MB



Additional Info:

<https://developer.ibm.com/recipes/tutorials/use-kubernetes-secrets-to-make-your-app-portable-across-clouds/>

Kubernetes: Secrets

```
peter@Azure:~/secret$ echo -n "A19fh68B001j" > ./apikey.txt
peter@Azure:~/secret$ ls
apikey.txt config.json vault
peter@Azure:~/secret$ kubectl create secret generic apikey --from-file=./apikey.txt
secret "apikey" created
peter@Azure:~/secret$ kubectl describe secrets/apikey
Name:          apikey
Namespace:    default
Labels:        <none>
Annotations:  <none>

Type:  Opaque

Data
=====
apikey.txt:  12 bytes
peter@Azure:~/secret$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/secrets/pod.yaml
W1003 11:54:36.624294     136 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested
resource), falling back to swagger
pod "consumesec" created
peter@Azure:~/secret$ kubectl exec consumesec -c shell -i -t -- bash
[root@consumesec /]# mount | grep apikey
tmpfs on /tmp/apikey type tmpfs (ro,relatime)
[root@consumesec /]# cat /tmp/apikey/apikey.txt
```

Note: Can be mounted as data volumes or exposed as environment variables



<http://kubernetesbyexample.com/secrets/>

1. **echo -n "A19fh68B001j" > ./apikey.txt**
2. **kubectl create secret generic apikey --from-file=./apikey.txt**
3. **kubectl describe secrets/apikey**
4. **kubectl create -f
<https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/secrets/pod.yaml>**
5. **kubectl exec consumesec -c shell -i -t – bash**
6. **mount | grep apikey**
7. **cat /tmp/apikey/apikey.txt**

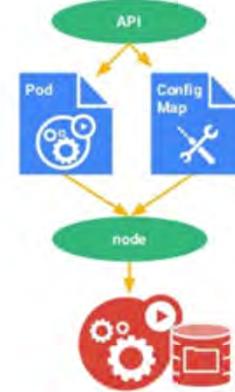
Kubernetes: Configmap

A set of key-value pairs that serve as configuration data for pods. They solve the problem of how to pass config data such as environment variables to pods.

Config maps are commonly composed of:

- Command line arguments
- Environment variables
- Files in a volume

Config maps function similar to secrets, but values are stored as strings and are more readily readable.



```
$ kubectl create -f etcd-config.yml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: etcd-env-config
data:
  discovery_v1: http://etcd-discovery:2379
  etcd1: http://etcd:2379
```



ASPE
TRAINING
Inspire. Educate. Create.

<http://pwittrock.github.io/docs/user-guide/configmap/>

Extra resources:

https://docs.openshift.org/latest/dev_guide/configmaps.html

Kubernetes Tools : Prometheus

Prometheus is an open source time series monitoring and alerting toolkit (originally written by SoundCloud). It is sponsored by the Cloud Native Computing Foundation and was the second project to be adopted after Kubernetes.

- ◆ Dimensional Data
- ◆ Powerful Queries
- ◆ Visualization Tools
- ◆ Efficient Storage
- ◆ Alerting System



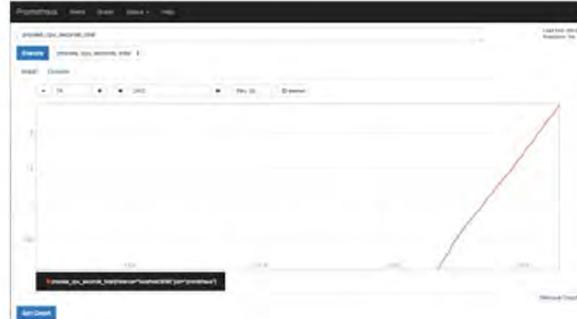
NOTES:

Kubernetes & Prometheus (Metrics)



CLASSROOM WORK

- 1. Stand up Prometheus**
- 2. Gather and monitor metrics**



Prometheus is an open source time series monitoring and alerting toolkit (originally written by SoundCloud). It is sponsored by the Cloud Native Computing Foundation and was the second project to be adopted after Kubernetes.

<https://prometheus.io/docs/introduction/overview/>

Stand up simple Prometheus cluster

1. <https://github.com/kelseyhightower/oscon-2017-kubernetes-tutorial/blob/master/labs/06-tutorial-metrics.md>

Note: Accessing the prometheus cluster locally with minikube: localhost:9090

Additional info:

<https://www.weave.works/blog/monitoring-kubernetes-infrastructure/>

<https://coreos.com/blog/prometheus-and-kubernetes-up-and-running.html>

<https://koudingspawn.de/kubernetes-monitoring-prometheus/>

Extending Kubernetes

- **Helm - package manager**
- **Third party extensions/plugins**
- **Operators**



NOTES:

Secure Service to Service Communication

- How to authenticate the user and then pass the login context between Microservices (pods)?
- How will each Microservice authorize the user?



<https://medium.facilelogin.com/securing-microservices-with-oauth-2-0-jwt-and-xacml-d03770a9a838>

Secure Service to Service Communication: OpenID & JSON Web Token (JWT)

OpenID is a flavor of OAuth2

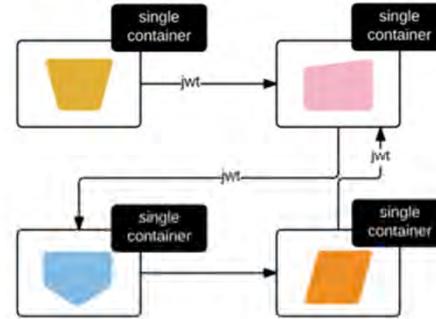
OAuth2 returns an access token called an ID Token or JSON Web Token

Pros

- Distributes identity and entitlements between services
- Transfers securely over an unsecured channel
- Asserts identity as long as sender is trusted

Cons

- Must provision the trusted certificates by service to each Microservice
- Must trust every certificate



<https://kubernetes.io/docs/admin/authentication/#openid-connect-tokens>

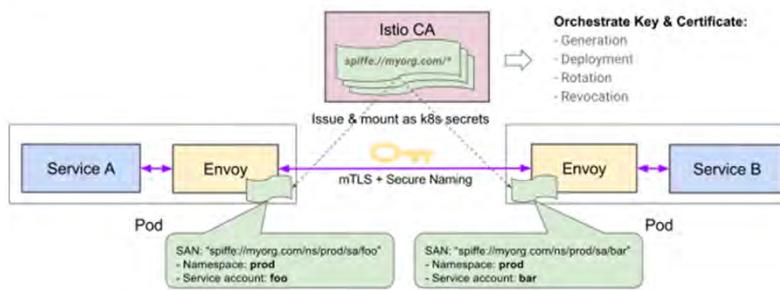
Secure Service to Service Communication: Private Certificate Authority

Pros

- Enables Services to only trust either root certificate authority or intermediary
- Reduced overhead in certificate provisioning

Cons

- Newer technique and technologies, must be appropriate for the given project



'SPIFFE' and 'Istio' projects are developing a solution for this issue

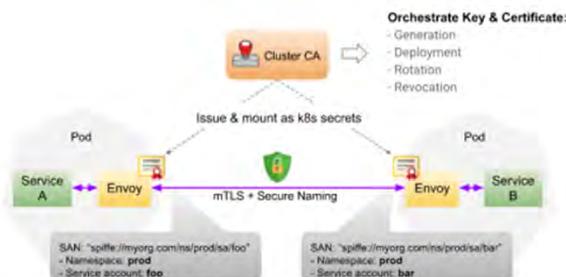
<https://github.com/istio/auth>

<https://github.com/spiffe/spiffe>



Secure Traffic

- Secure traffic between Microservices
 - HTTPS
 - Re-configure ELBs (on AWS)
 - Add certificate (letsencrypt.org)
- Secure traffic for users
 - HTTPS
 - Secure Identity with [SPIFFE](#) & [Istio](#)



SPIFFE is an early effort to provide automation around certificates, identity and inter microservice communication
<https://github.com/spiffe/spire>

Istio is a platform desired to connect, manage, and secure microservices
<https://istio.io/>

Security Actions

- Ensure attacker who gains access to web container can't get into the database
- Create separate DB users for each microservice
- Strong passwords
 - K8's secrets and/or Vault
 - Only accessed by containers who need them



NOTES:

Kubernetes Secrets



CLASSROOM WORK

- 1. Create Secret config file and pods**
- 2. Install and Launch Vault**
- 3. (Optional) Challenge: Replicate Vault Secrets into Kubernetes**



Tutorial

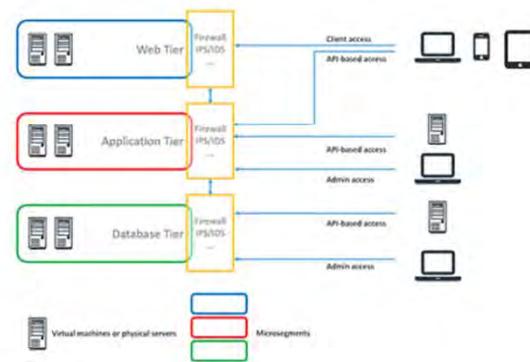
<https://github.com/kelseyhightower/oscon-2017-kubernetes-tutorial/blob/master/labs/04-tutorial-secrets.md>

Optional Challenge:

<https://github.com/kelseyhightower/oscon-2017-kubernetes-tutorial/blob/master/labs/05-challenge-secrets.md>

Security - Microsegmentation

- Segment the network
 - Only allow white-listed traffic between pods
- Common tools
 - Istio
 - Calico
 - Weave
 - OpenContrail



NOTES:

Security – Least Privilege

“Each subject in a system be granted the most restrictive set of privileges”



- Disable public IP access for any machine that doesn't need it
- VPN only for admin access



NOTES:

Security Actions

- Limit admin access
- ABAC mode by default
 - Commonly in srv/kubernetes
- Access control by token
 - Can allow/deny API operations
 - Can be used to only allow certain actions
- Webhook (optional)
 - Add url for k8s to verify authorization



NOTES:

Secure Images

- Follow best practices for packaging and distributing applications
 - In general, try not to mix secrets and images
 - This can include containers with passwords baked in



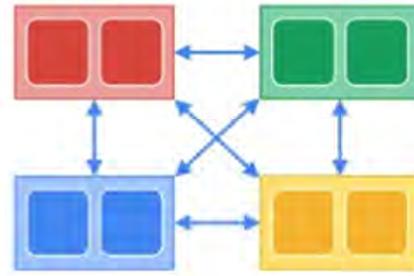
NOTES:

Kubernetes Networking

Kubernetes assumes that pods can communicate with other pods, regardless of host

- Kubernetes requires
 - All containers to communicate with other containers without NAT
 - All nodes can communicate with all containers without NAT
 - IP that container sees itself as is the same as others see it

In practice, Kubernetes cannot be installed simply with Docker. It usually requires a networking layer such as Weave, Flannel, etc



Additional Info:

<https://jvns.ca/blog/2016/12/22/container-networking/>

Kubernetes Networking

Every pod has its own IP

- No mapping ports
- Clean, backwards compatible model where Pods can be treated like VMs

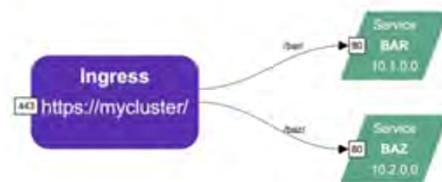


NOTES:

Kubernetes Ingress

Simplified Layer 7 access to Kubernetes services i.e. allows “internet” to reach services

- Works with load balancers, including nginx & cloud
- Single root URL mapping
- Publicly expose private networks
- Terminates TLS/SSL



NOTES:
