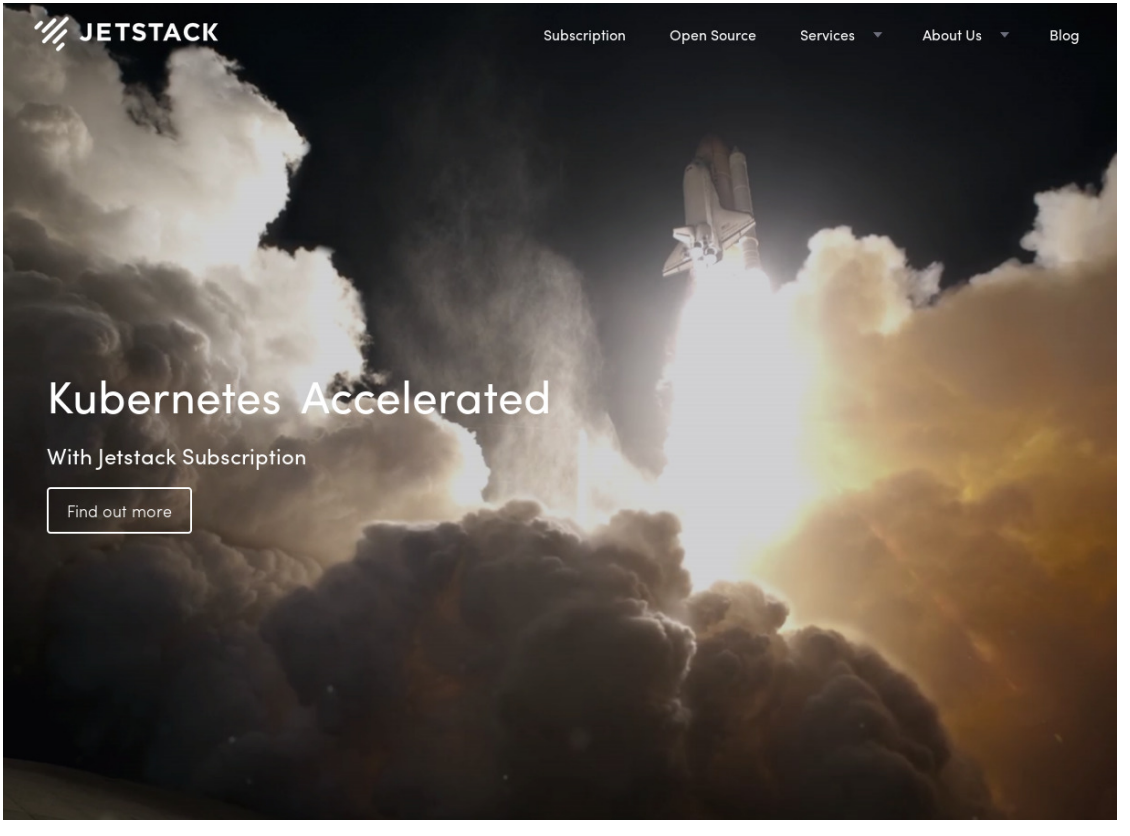
Kubernetes in Practice

Beginner

Schedule

- Morning
  - Setup
  - What is a container?
  - Introduction to Kubernetes resources
- Afternoon
  - Deployments
  - Persistent volumes
  - Configmaps and Secrets
  - Deploying with Helm

Welcome


JETSTACK
Subscription
Open Source
Services
About Us
Blog
Kubernetes Accelerated
With Jetstack Subscription
Find out more

Welcome

Each person say:

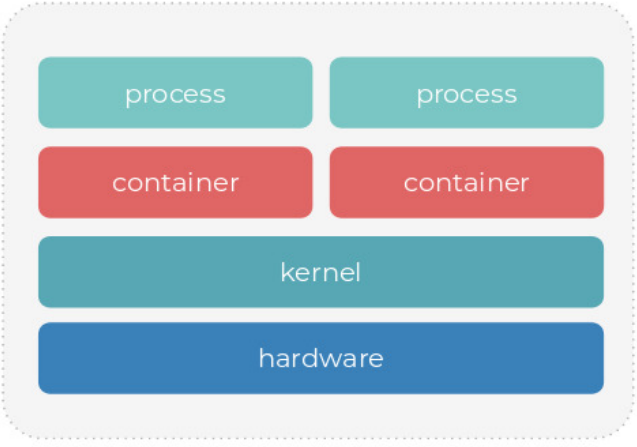
- Hello & where they are from
- How much k8s experience they have
- The main thing they want to learn
- Favourite something computer related
- Favourite something not computer related

Background

What are containers and where did they come from?

What is a container?

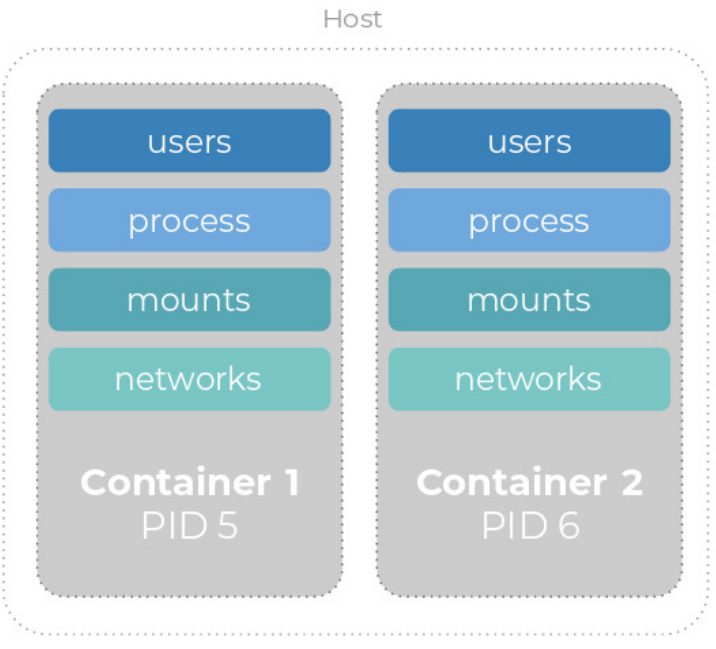
- A Linux process
- Shares a host kernel with other container processes
- Kernel-level virtualisation



Kernel Namespaces

- Feature of the Linux kernel to isolate processes

  Isolate from each other and the host



Kernel Namespaces

- pid
  - See only the processes running inside the namespace
  - The pids are local to the container
  - *ps -A* behaves like a fresh VM
  - You can run several processes in the same namespace and they can see each other

```
root@a5bcac57bf92:/# ps -A
  PID TTY          TIME CMD
    1 ?        00:00:00 bash
   13 ?        00:00:00 ps
root@a5bcac57bf92:/#
```

Kernel Namespaces

- network
  - *ifconfig* shows a single interface
  - Packets are routed internally by the kernel
  - No more port collisions

```
root@a5bcac57bf92:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
          inet addr:172.17.0.3  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:17525 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8530 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:25921796 (25.9 MB)  TX bytes:466151 (466.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@a5bcac57bf92:/#
```

Kernel Namespaces

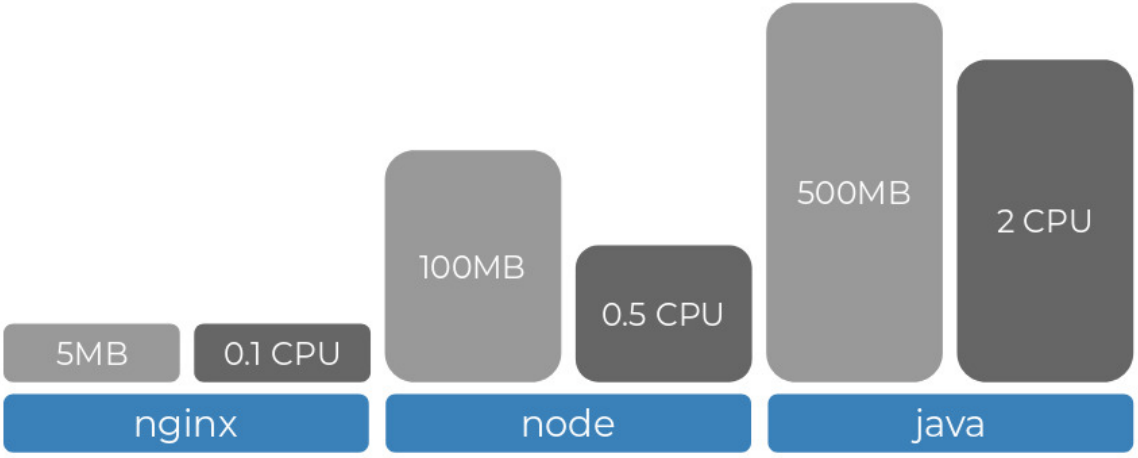
- mount
  - Root volume is copy-on-write
  - Mount external volumes at chosen paths in container

Kernel Namespaces

- *User*, one container can see a different group of users to another
- *IPC*, isolate processes from SysV style inter-process communication
- *UTS*, control hostname and domains

Kernel cgroups

- Feature of the Linux kernel to allocate resource limits
- Control memory, CPU and I/O usage on a per-process level
- Prevents over-consumption and exhaustion of resource by 'noisy neighbours'

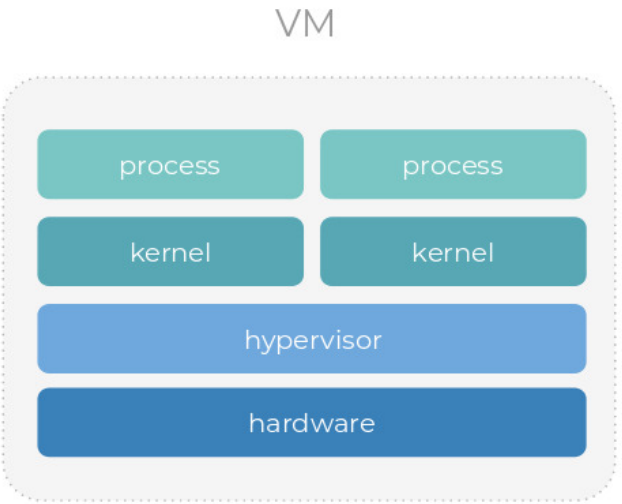


cgroups + namespaces = containers

- Containers are actually these lower level kernel features combined
  In some form - have been around for a while
- Chroot, FreeBSD Jails, LXC, OpenVZ, Solaris Zones

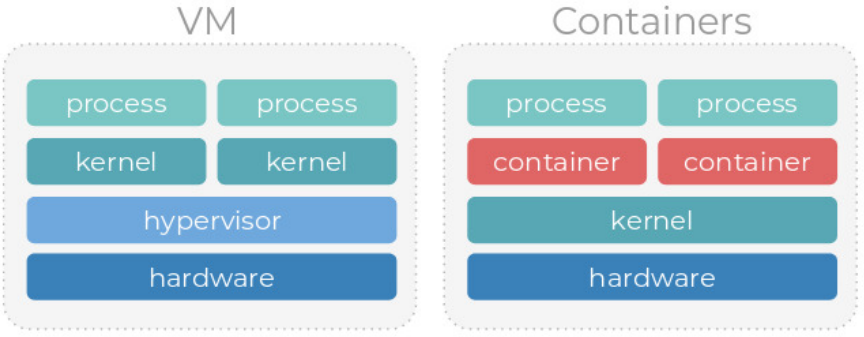
VM vs Container

- VMs run their own kernel
- You can run Windows and Linux on the same hypervisor
- Different hardware drivers for each VM
- BIOS is virtualized so virtual hardware can be attached



VM vs Container

- Containers share the same kernel
- Linux only because the kernel is shared
- The isolation happens **inside** the kernel not beneath it
- Start-up time reduced because kernel is already running



Containers at Scale

- Running containers across thousands of machines
- The need for containers in companies like Google comes from massive scale
- The efficiency of containers can be seen in:
    - Start-up time
    - Server density
    - Ease of immutable deployments

Enter: Docker

- Containers with a friendly API
- Docker announced at PyCon
    - For many the first experience with containers first-hand
- Massive adoption because of
    - Ease of the API
    - Container image format
    - Dockerhub container registry to push/pull images

Google and containers

- Tried and tested at massive scale
- (Almost) everything at Google runs in a container – even a VM
- 10 years+ (pre-VMs)
- Enables huge resource efficiencies

Borg

- Internal Google tool. Allows Google to treat its data centres as a single pool of resources and schedule containers on them
- Was a closely-guarded secret until a paper was published in 2015 (freely available online from Google Research)
- The predecessor of Kubernetes

Large-scale cluster management at Google with Borg

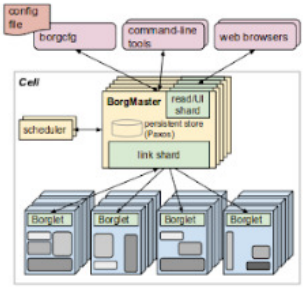
Abhishek Verma† Luis Pedrosa‡ Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

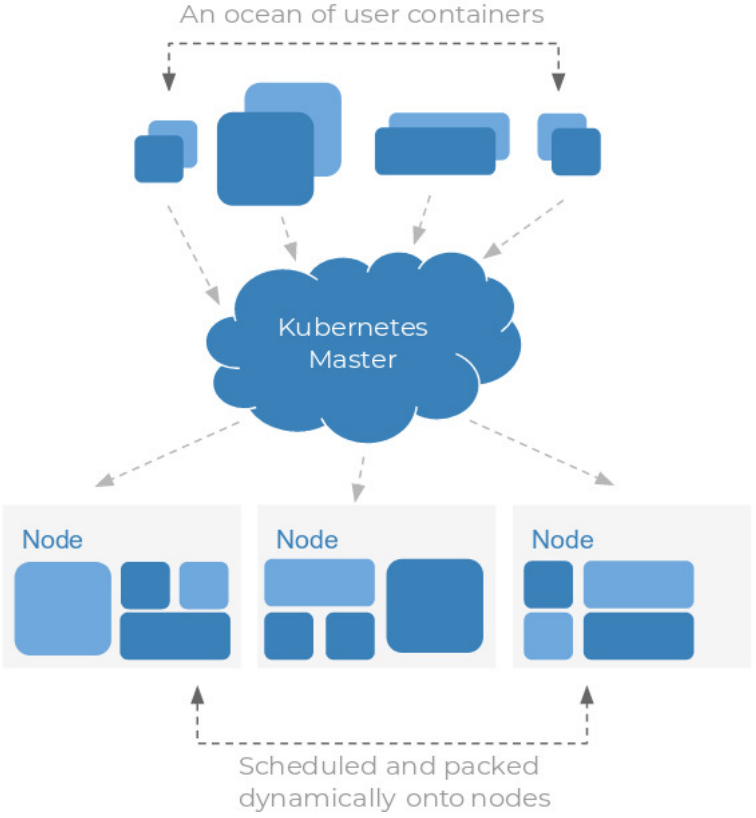
We present a summary of the Borg system architecture



Kubernetes Born

- Started by
  - Joe Beda, Brendan Burns, Craig McLuckie, later Brian Grant and Tim Hockin
  - These same folks who worked on and wrote the paper for Borg (inspiration for Kubernetes)
- Open source alternative to Borg and Omega (its successor)
  - 1.0 Announced at OSCON in 2015 (first public release, June 2014)

Kubernetes Overview



Kubernetes Overview

- Visibility
  - Oversight and control of what is currently running in the cluster
- Reproducibility:
  - Describe resources in declarative files - infrastructure can be committed to version control
- Portability:
  - Can be installed on many different types of infrastructure
- Resilience and elasticity:
  - Pro-actively monitors, scales, auto-heals and updates

Kubernetes Overview

- Declarative system description using
  - Application abstractions
  - Pods
  - Replica Sets
  - Deployments
  - Services
  - Persistent Volumes
  - Ingress
  - Secrets
  - and many more!

Kubernetes Overview

- Kubernetes is often abbreviated to k8s
- 8 letters between ‘k’ and ‘s’
- You can say this as *Kates*

Kubernetes Overview

GitHub - extracted December 17th, 2018

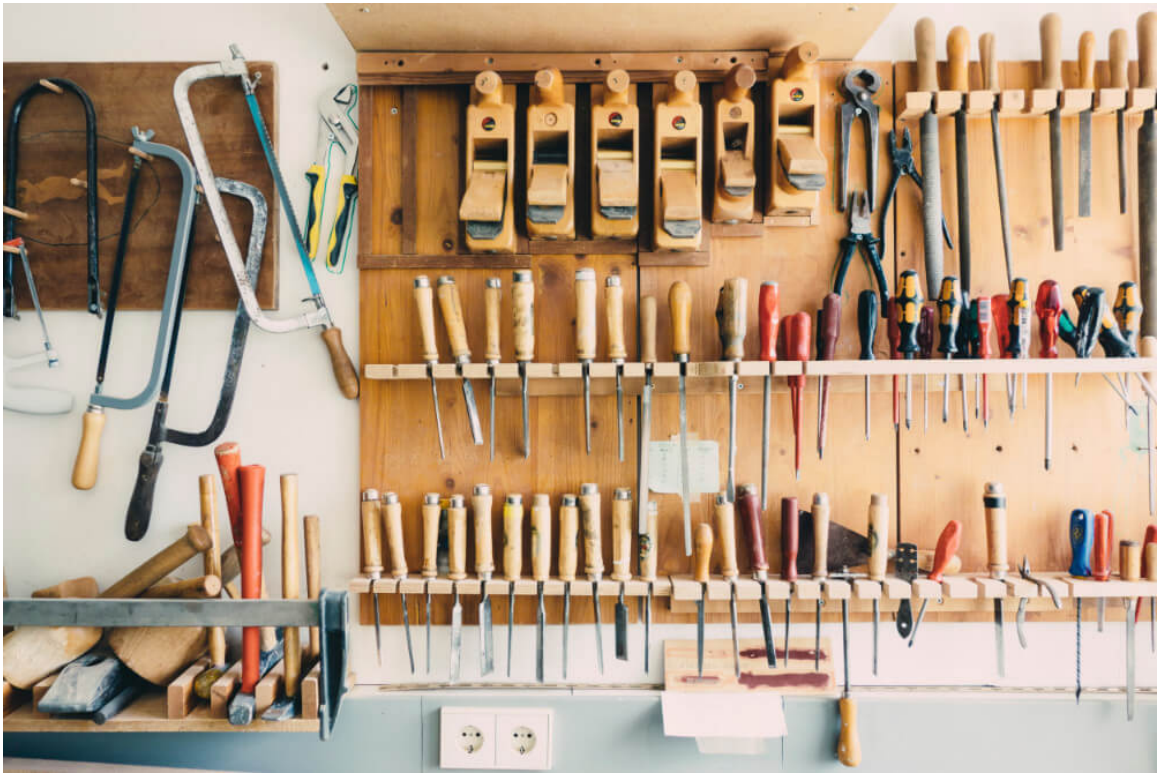
Workshop

Getting Access

- Browse to the workshop site: **beginner.k8s.school**
- Username: **beginner**
- Password: **k8sworkshop**
- Download workshop materials (top right)

Workshop

1. Setup & Install



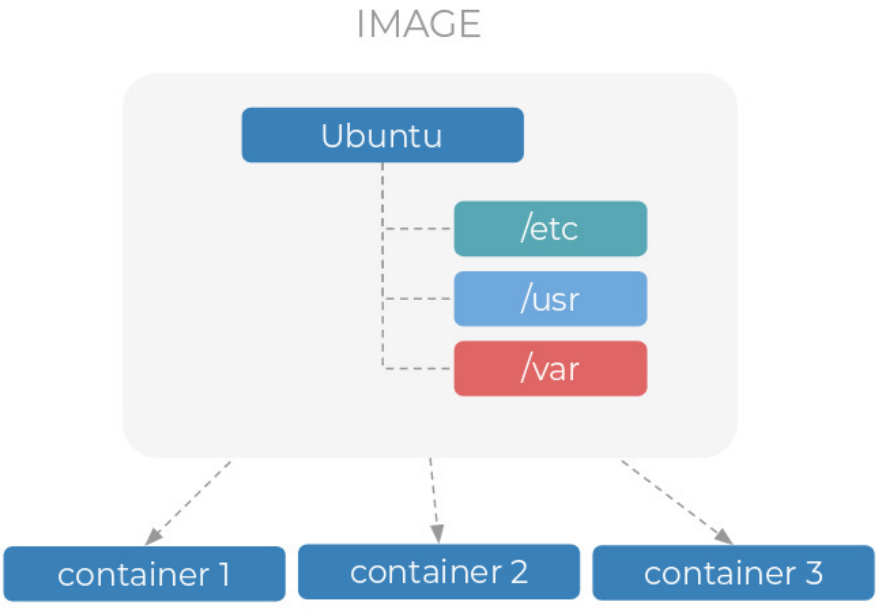
Docker Feature Overview

Docker usage refresher

Docker Feature Overview

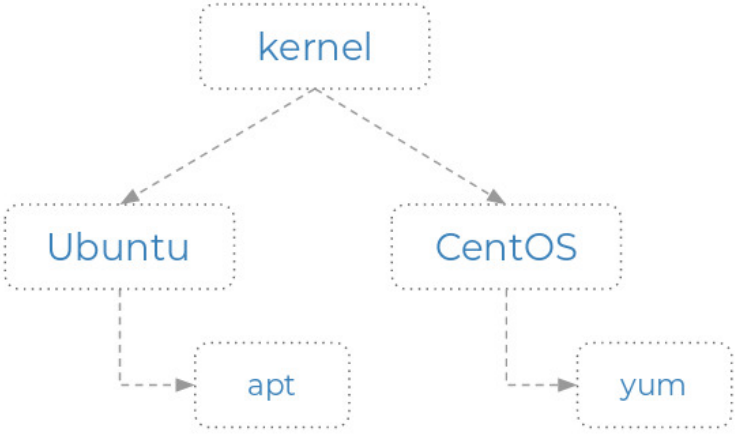
- An **image** is essentially a root filesystem
  - It can be distributed as a .tar.gz
  - Or 'pulled' from a 'registry'
- Containers are *run* from images
  - The image controls what the container will see as its root file system

Docker Feature Overview



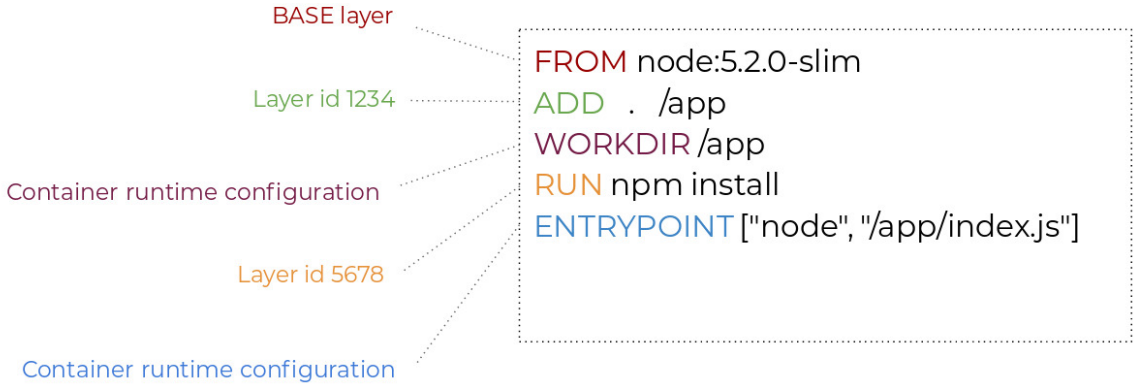
Docker Feature Overview

- A Linux distro is the kernel plus an opinionated file layout
- Because containers share the kernel – all you need is the file layout
- This is what the image provides



Docker Feature Overview

- Images are made up of a series of layers



Docker Feature Overview

- Use *docker build* to create images

```
$ docker build -t hello src
```

TAG - the name of the image that docker will build out of our Dockerfile

PATH - a path containing a Dockerfile and other code that can be copied into the image

Docker Feature Overview

- See the pulled and built images on your local Docker host
- *$ docker images*

Docker Feature Overview

- Naming and labelling containers
  - This allows us to control version numbers when we update images. It creates a pointer to the original image name - you can have many tags for the same image.

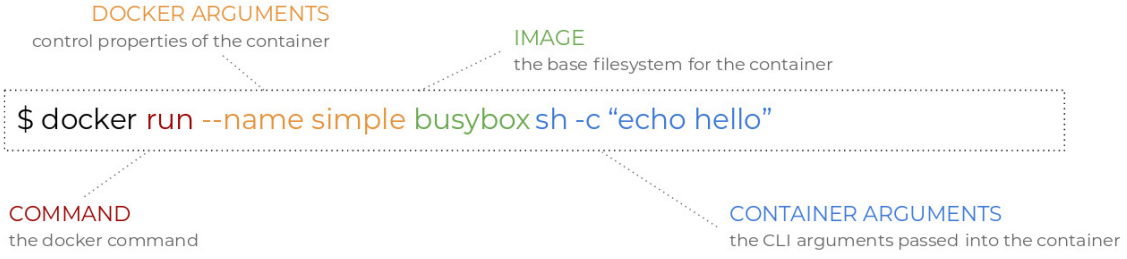
```
$ docker tag helloworld -t hello:0.0.2
```

IMAGE NAME - the name of the image to add a tag to

TAG - the name of the tag to add to that image

Docker Feature Overview

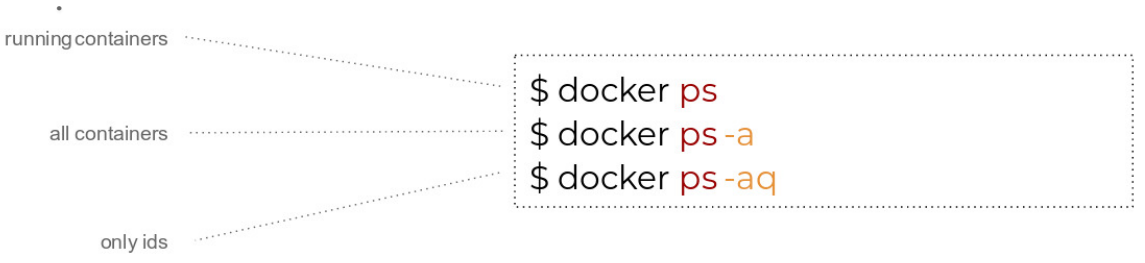
- Running containers!
- The docker run command is split into 3 parts



Docker Feature Overview

- Listing our containers
  - We can see our containers using the ps command.
  - Use the -a flag to show all containers (including stopped ones).

    Use the -q flag to show only container ids



Docker Feature Overview

- Viewing detailed container information

```
$ docker inspect helloworld
```

CONTAINER ID or NAME
which container to inspect

Docker Feature Overview

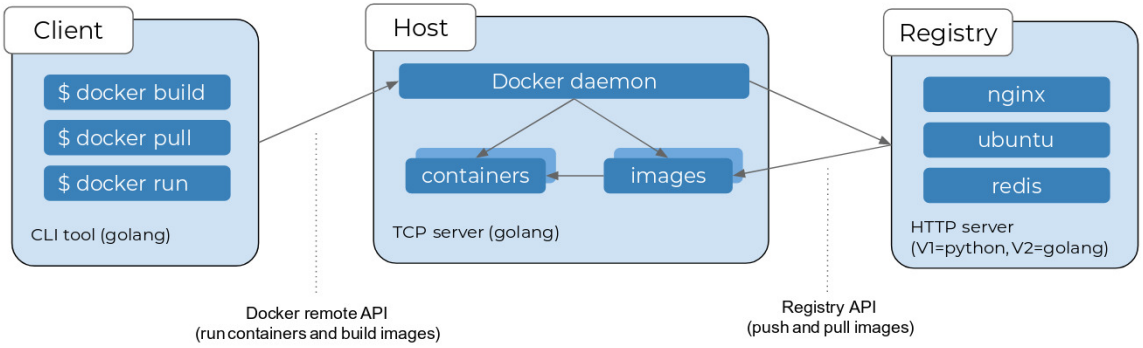
- Removing containers
- Remove containers using the rm command. You can use the auto-generated container ID or the name given by the –name flag passed to docker run.

```
$ docker rm helloworld
```

CONTAINER ID or NAME
which container to remove

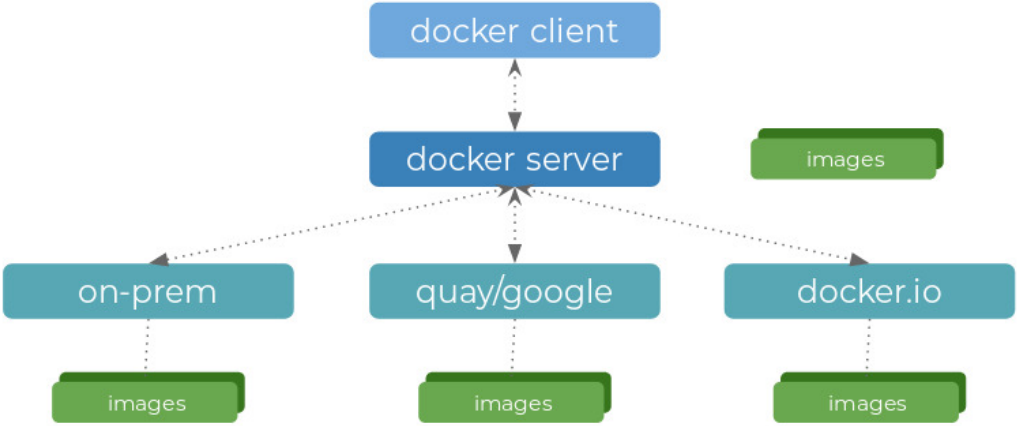
Docker Feature Overview

- Publishing Docker images



Docker Feature Overview

- Docker Hub (docker.io)
- Other public/private options
    - Google Container Registry (GCR)
    - quay.io (CoreOS)
- Can also be self-hosted almost anywhere

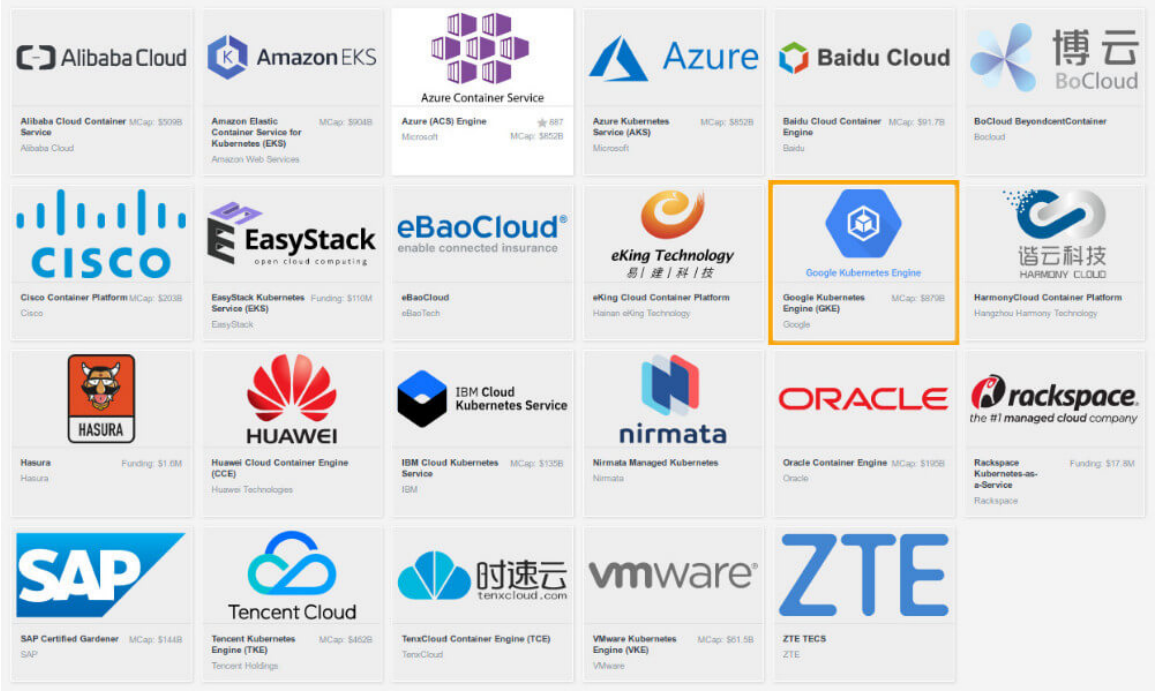


Infrastructure

I need a cluster

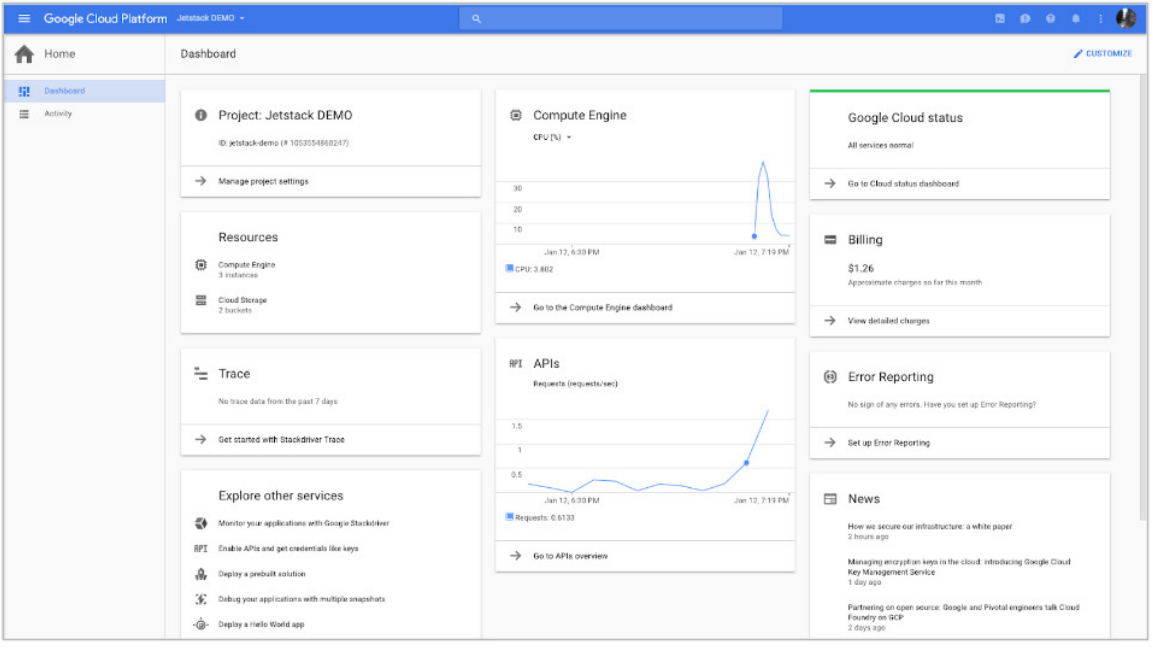
Infrastructure

Kubernetes is designed to be infrastructure agnostic



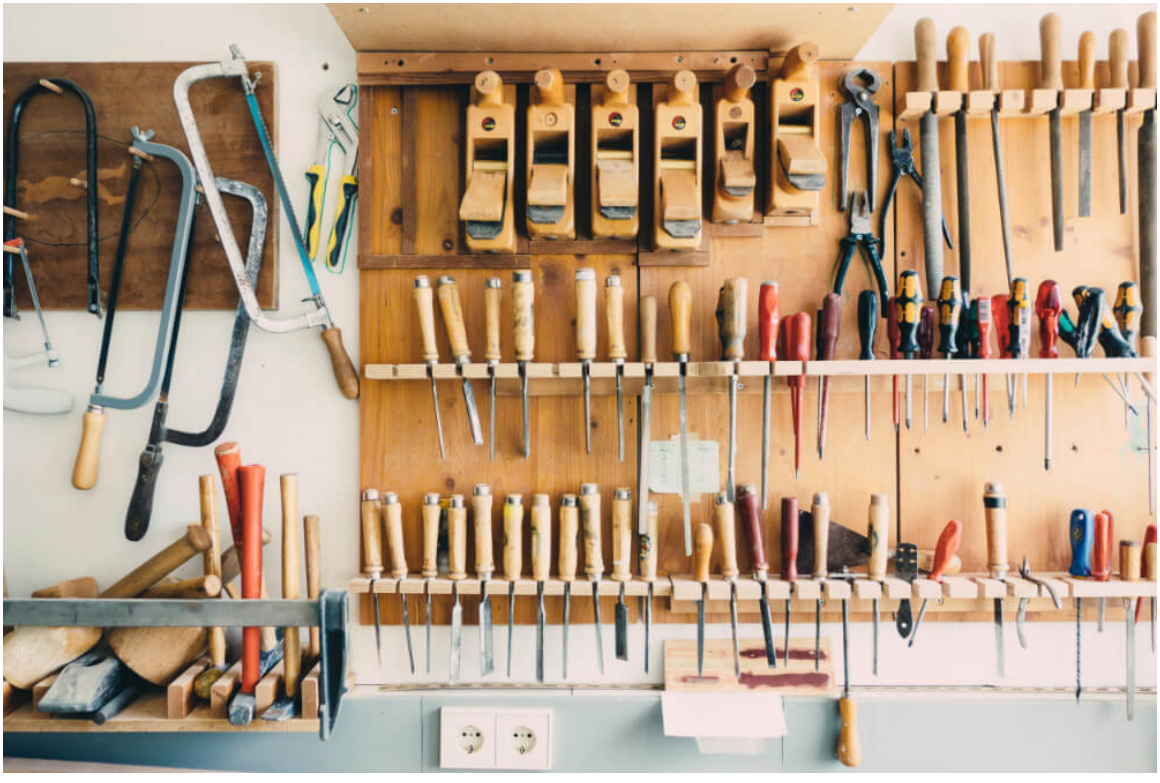
Infrastructure

We're going to be using GKE



Workshop

2. Muster a Cluster



Kubernetes Pods

The smallest deployable units of computing on Kubernetes.

Kubernetes Pods

- One or more containers
- Shared:
  - namespaces – network and IPC
  - hostname (but not UTS namespace)
  - volumes
  - fate
- Extra containers in pods:
  - data pullers, pushers, proxies etc
- IP-per-pod networking model
- Pod health probes – readiness, liveness

Kubernetes Pods

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-world-pod
  labels:
    app: hello-world
spec:
  containers:
  - name: hello-world-container
    image: jetstack/hello-world:0.1
    ports:
    - containerPort: 80
      protocol: TCP
```

Workshop

3a. Run a container on the cluster

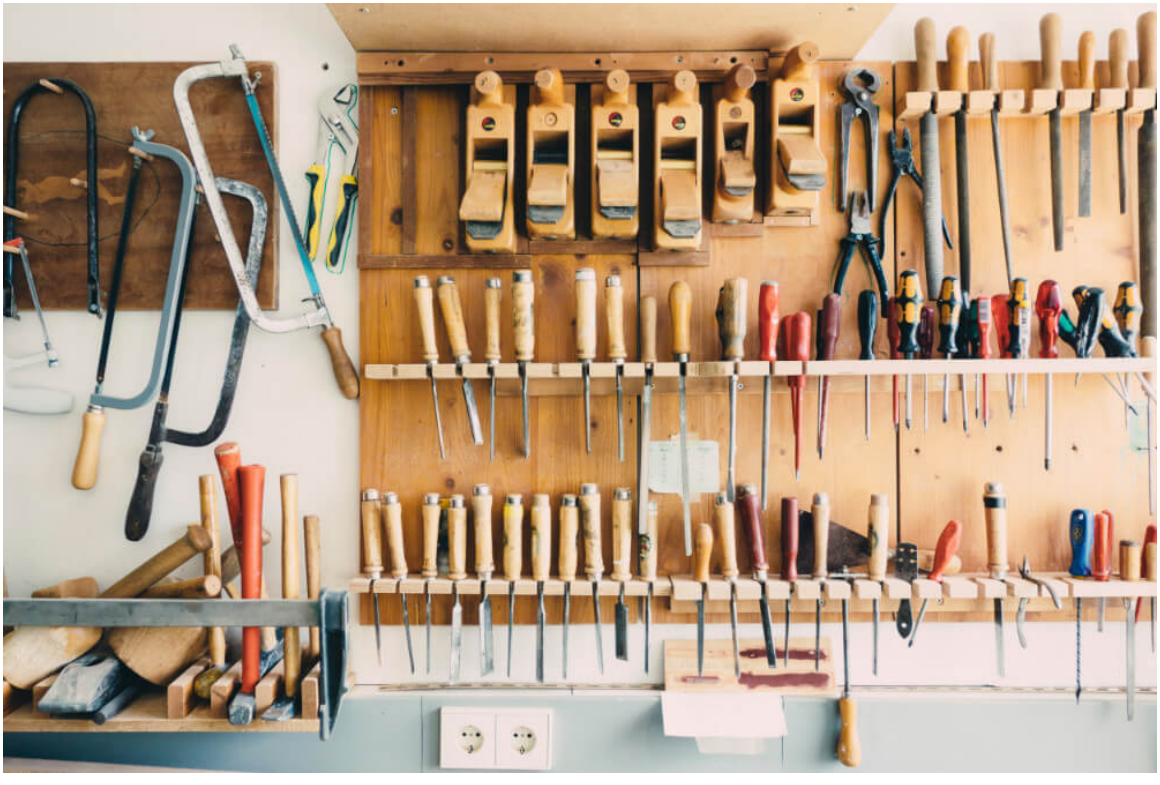


Photo by Barn Images on Unsplash

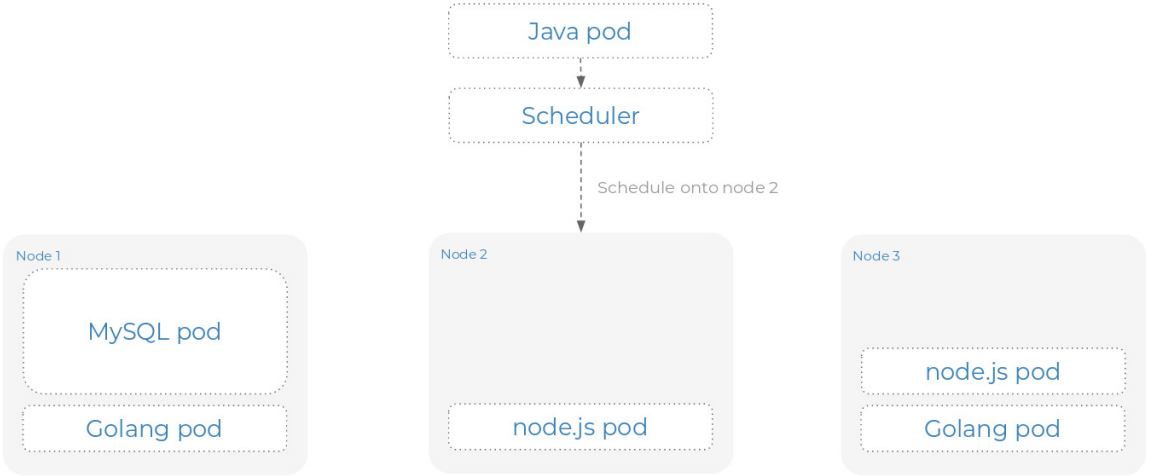
kube-dashboard & Access Control

Aside, where's my dashboard?

Pod Limits

Rein in your pods

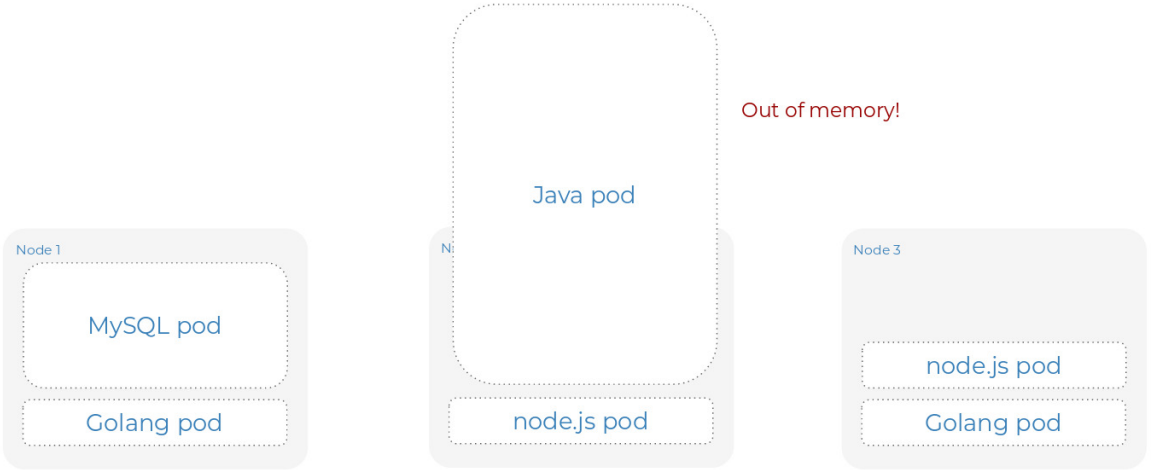
Pod Limits


Java pod
Scheduler
Schedule onto node 2
Node 1
MySQL pod
Golang pod
Node 2
node.js pod
Node 3
node.js pod
Golang pod

Pod Limits


Node 1
MySQL pod
Golang pod
Node 2
Java pod
node.js pod
Node 3
node.js pod
Golang pod

Pod Limits


Java pod
Out of memory!
Node 1
MySQL pod
Golang pod
node.js pod
Node 3
node.js pod
Golang pod

Pod Limits

```
spec:
  containers:
  - image: gcr.io/google_containers/serve_hostname
    imagePullPolicy: Always
    name: kubernetes-serve-hostname
    resources:
      limits:
        cpu: "0.9"
        memory: 512Mi
      requests:
        cpu: "500m"
        memory: 256Mi
```

Pod Limits

Best-Effort

- If requests and limits are not set for all of the resources, across all containers

Burstable

- If requests and optionally limits are set (not equal to 0) for one or more resources across one or more containers, and they are not equal

Guaranteed

- If limits and optionally requests (not equal to 0) are set for all resources across all containers **and they are equal**

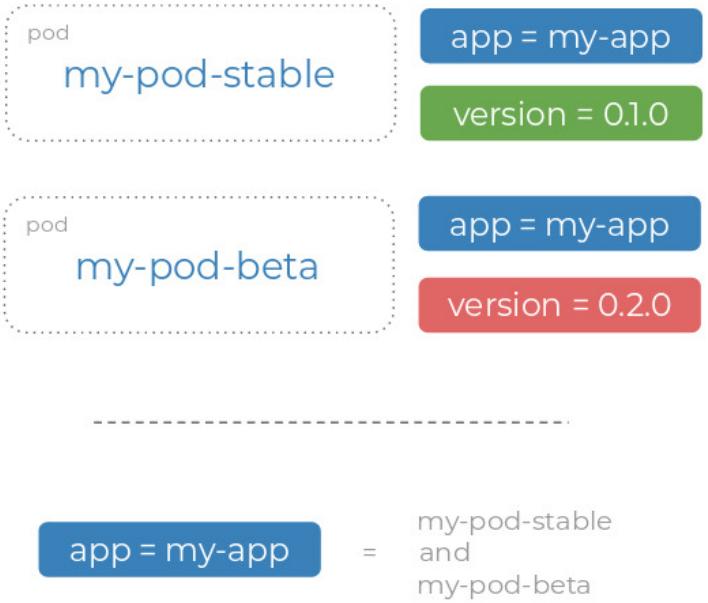
Labels

Help Kubernetes help you. Tag & find your resources

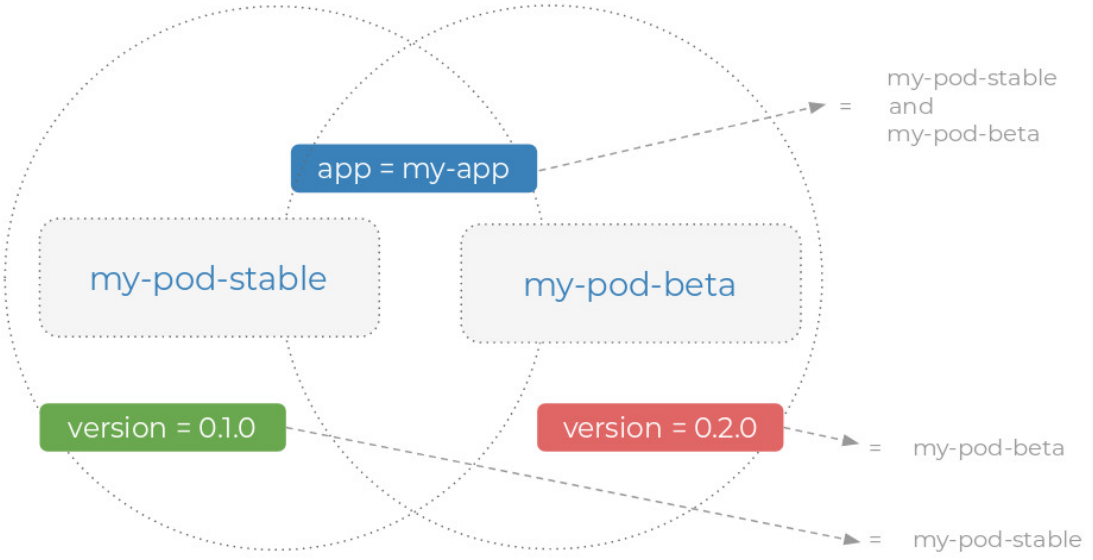
Labels

- Key/values attached to any API objects (pods, RCs, services – even nodes)
- Identifying attributes
- Mutable
- Used throughout Kubernetes
- Queried using selectors (set, equality)

Labels


pod
my-pod-stable
app = my-app
version = 0.1.0
pod
my-pod-beta
app = my-app
version = 0.2.0
app = my-app
=
my-pod-stable
and
my-pod-beta

Labels


app = my-app
my-pod-stable
my-pod-beta
version = 0.1.0
version = 0.2.0
=
my-pod-stable
and
my-pod-beta
=
my-pod-beta
=
my-pod-stable

Selecting Resources

- Labels allow to you “select” resources based on label predicates
- They “connect” one type of resource to another
- Things that use the selector pattern:
    - Services selecting which pods they route to
    - Deployments selecting which pods they control
    - Scheduler selecting which nodes to assign a pod onto

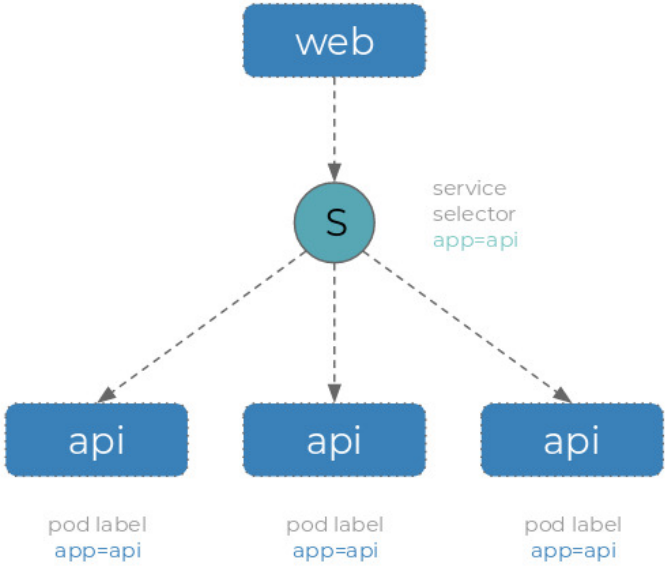
```
apiVersion: v1
kind: Service
spec:
  selector:
    app: my-app
...
```

Services

Address your pods

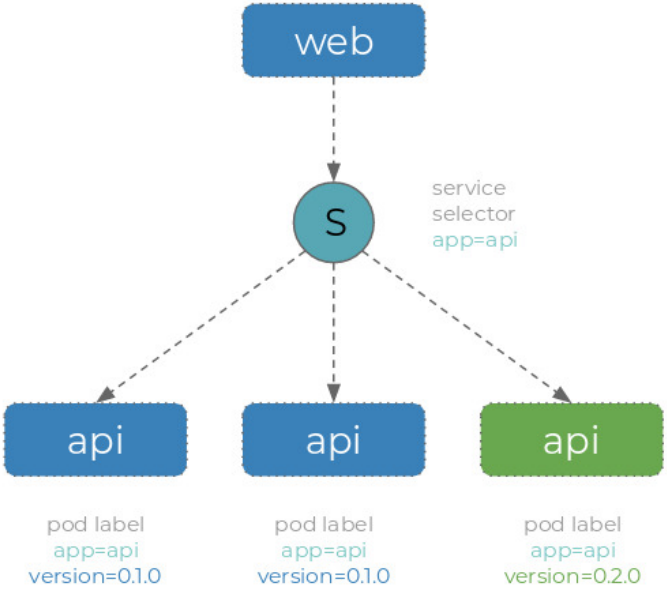
Services

- Provides a single, stable network identity to a set of potentially dynamic pods
- Pods selected using labels
- Round-robin load balanced



Services

- Because services match pods using selectors, you can be creative.
- For example, run 2 Deployments matched by 1 Service for a canary deployment.
- We'll be running a workshop on canary deployments later

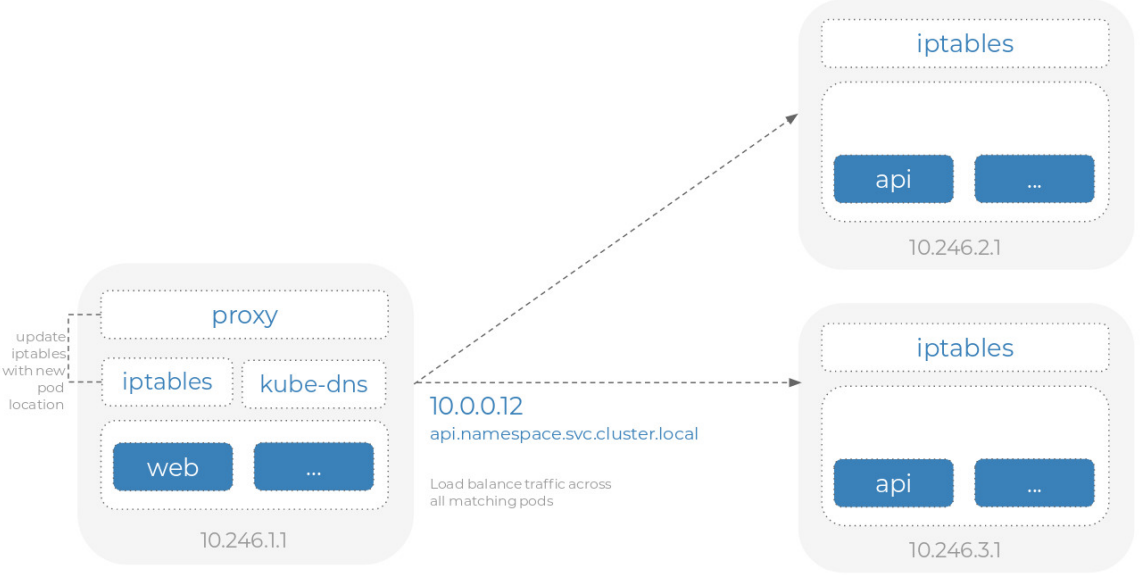


Services

- ClusterIP
  - Internal to the cluster only; chosen or specified manually (default)
- NodePort
  - ClusterIP + same port opened on each node
  - Allocated from configurable range (default: 30000-32767)
- LoadBalancer
  - ClusterIP + NodePort, also ask a cloud provider for a LB - e.g:
    - GCE Network/HTTP(s) Load Balancer
    - AWS Elastic Load Balancer

Services

- Intra-cluster, Service VIPs & DNS



Sock Shop

Start working with a ready-made set of microservices

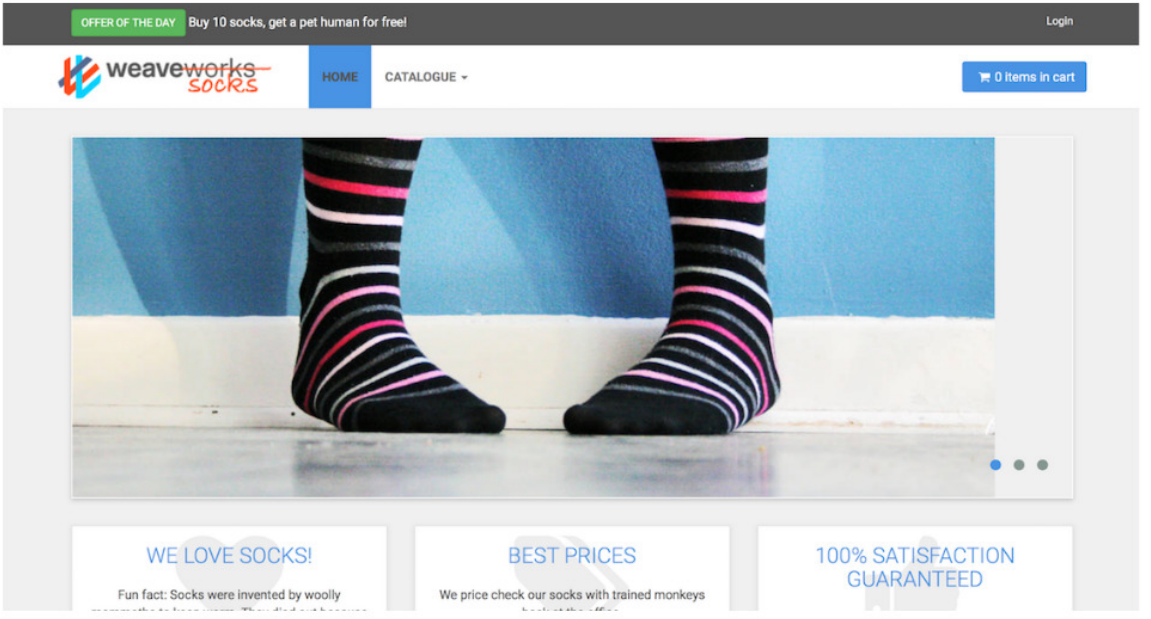
Sock Shop

Our CEO has decided to acquire a startup that sells socks and expand the corporate strategy towards designer cotton footwear.

The board considers a move from insurance into socks both an unorthodox and risky move. The boss's neck is on the line and so she has gone all out with a massive PR campaign that will launch ASAP.

We need to focus on how to deal with the massive spike in traffic we are expecting after the PR campaign.

Sock Shop


OFFER OF THE DAY
Buy 10 socks, get a pet human for free!
Login
weaveworks
socks
HOME
CATALOGUE
0 items in cart
WE LOVE SOCKS!
Fun fact: Socks were invented by woolly
BEST PRICES
We price check our socks with trained monkeys
100% SATISFACTION GUARANTEED

Sock Shop

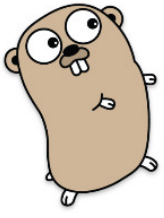
The CEO has hired some of the very best marketing agencies and they are talking up the numbers of visitors the site will get on launch. Our choice of infrastructure will be critical to the success of the project.

Fortunately - we have some friends at Google and they have recommended using **Google Kubernetes Engine (GKE)** as a production-grade, low-friction solution.

We can always move the cluster later - Kubernetes can be installed on many different types of infrastructure.

Sock Shop

Our ludicrous developers have gone all-in on polyglot micro-services and have a stack consisting of...



Java

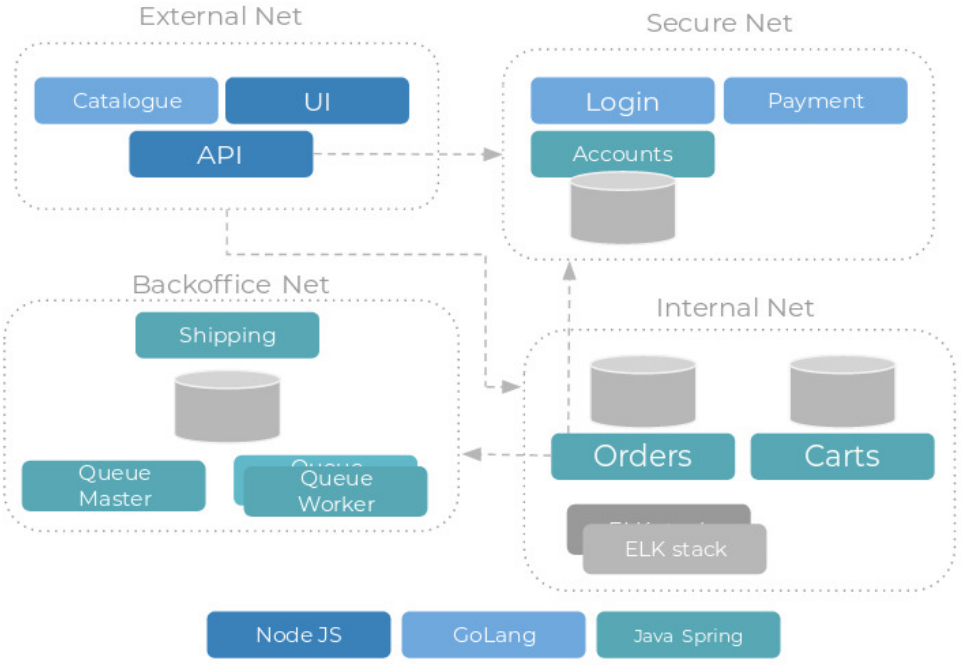
MySQL

ZIPKIN

mongoDB

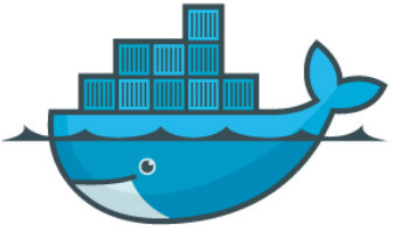
Sock Shop

- Microservices built using Spring Boot, Go kit and Node.js
- Polyglot data persistence - MongoDB, MySQL, etc
- All ready packaged in Docker containers



Sock Shop

- Thankfully, they have used Docker for development and have created a Docker image for each of the services.
- We're going to make use of their images and Kubernetes manifests.



Workshop

3b. Run the sock-shop pods on the cluster

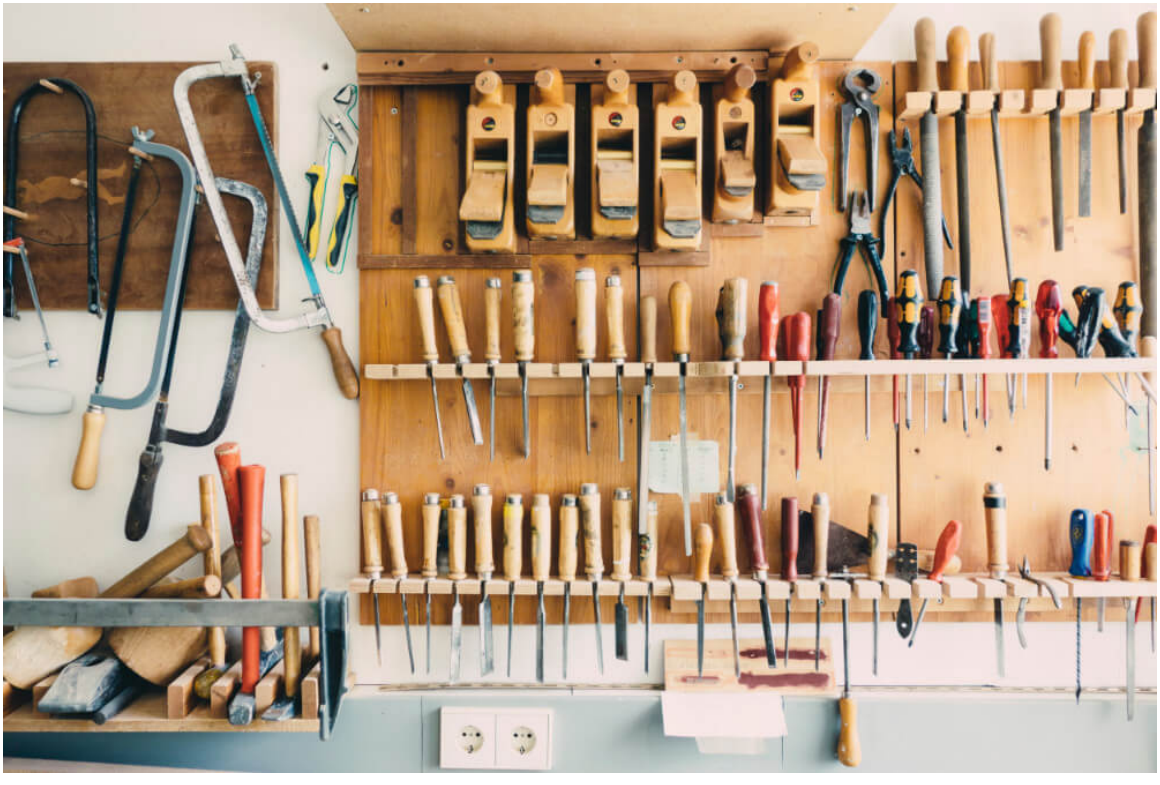


Photo by Barn Images on Unsplash

Namespaces

Define a stricter grouping on cluster resources

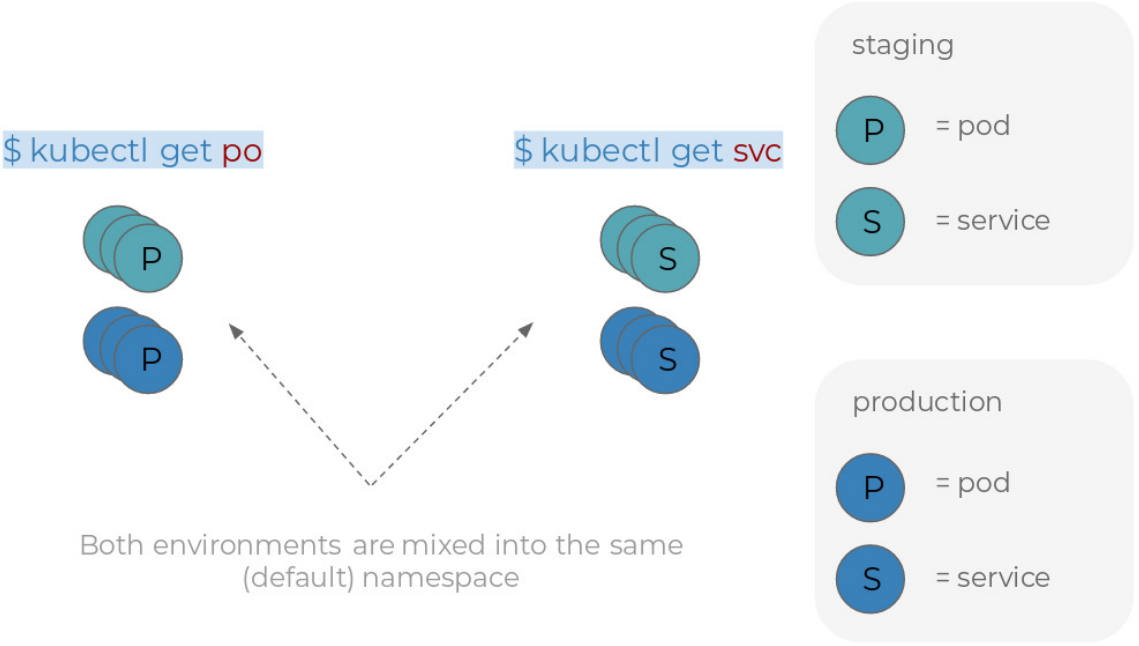
Namespaces

```
kubectl get pod -n kube-system
```

```
kubectl get nodes -n non-existent-namespace
```

Namespaces

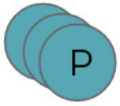
Without namespaces



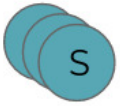
Namespaces

With namespaces

```
$ kubectl get po \
 --namespace=staging
```




```
$ kubectl get svc \
 --namespace=staging
```




```
$ kubectl get po \
 --namespace=production
```


```
$ kubectl get svc \
 --namespace=production
```

Workshop

3c. Experiment with Kubernetes namespaces

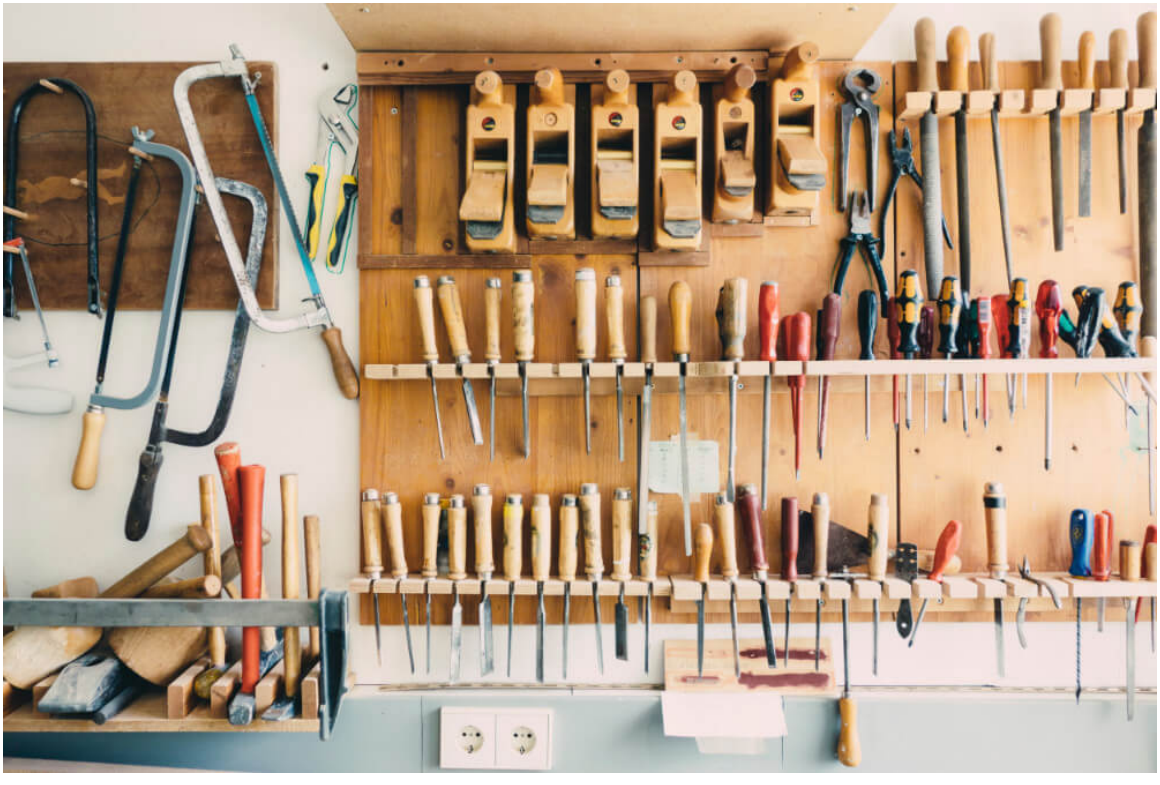


Photo by Barn Images on Unsplash

Replica Sets & Deployments

Control how changes to your workloads are applied

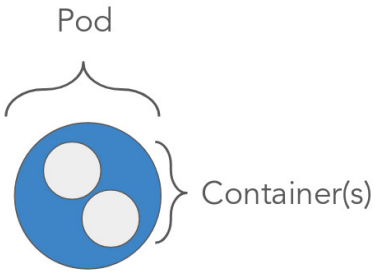
Replica Sets & Deployments

- Pods can come and go; they're fungible (replaceable by another identical item)
- Control loop (controller) dynamically ensures desired number of pods is running
- Converges desired & actual state
- Cookie-cuts new 'replicas'

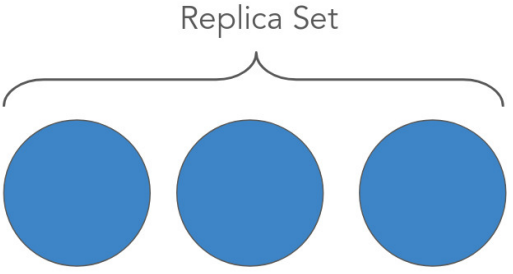
Deployments

- Declarative, server-side updates to pods and replication controllers
- Deployment controller converges desired and actual state
- Can be used for configurable rolling updates

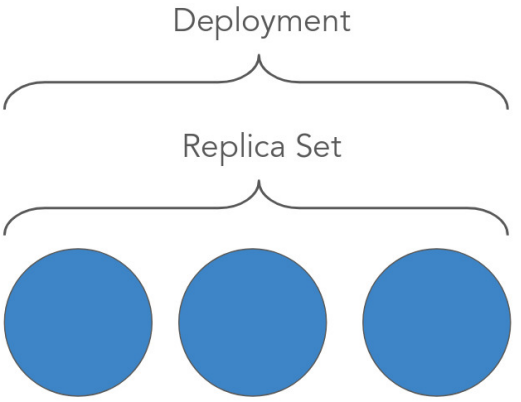
Workload Example



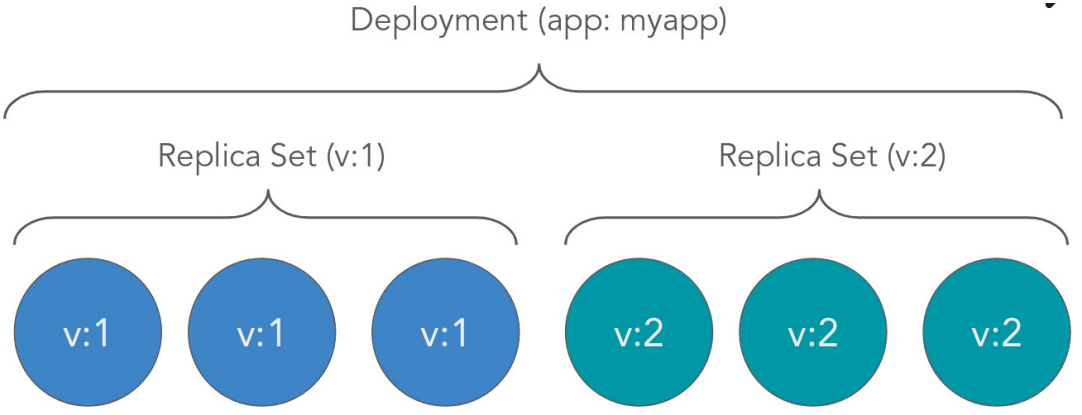
Workload Example



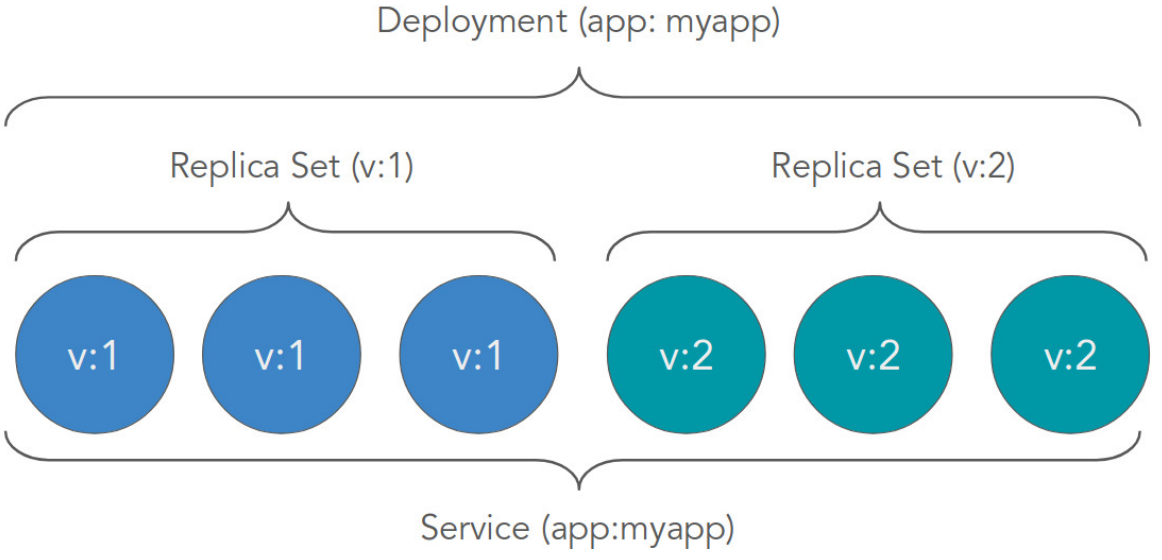
Workload Example



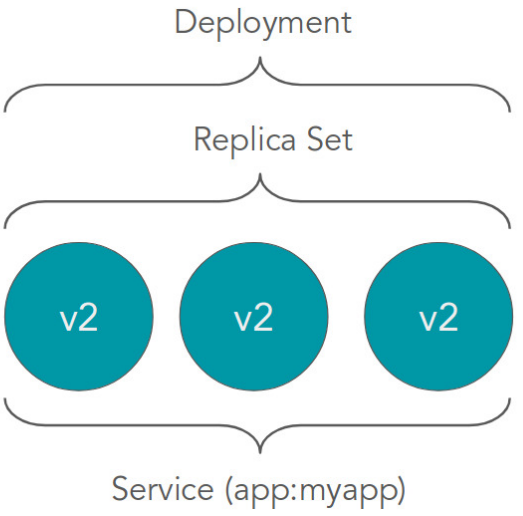
Workload Example



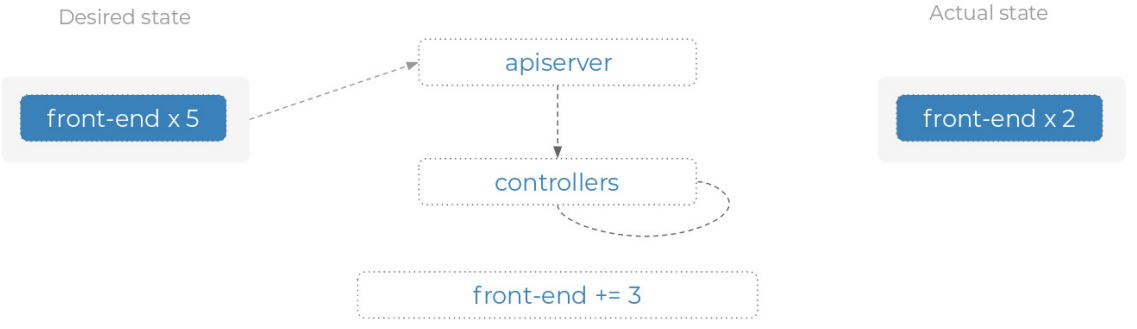
Workload Example



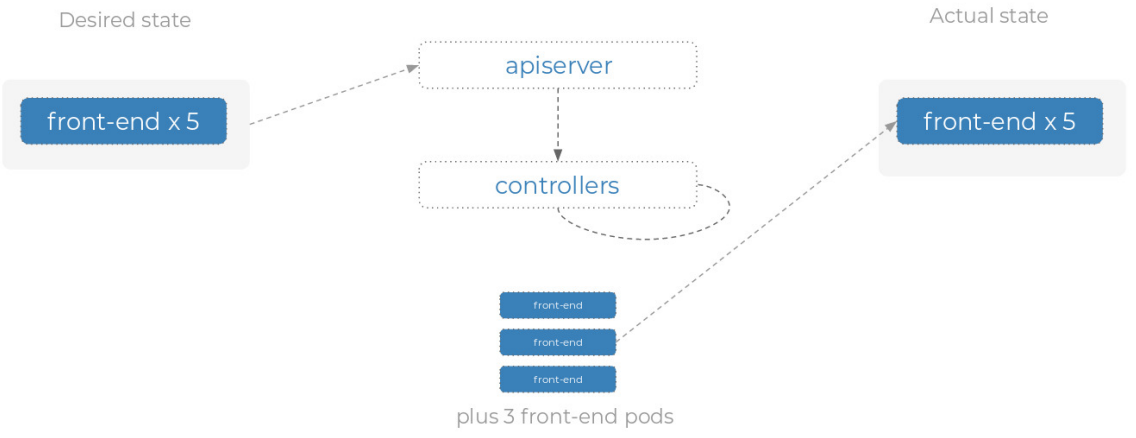
Workload Example



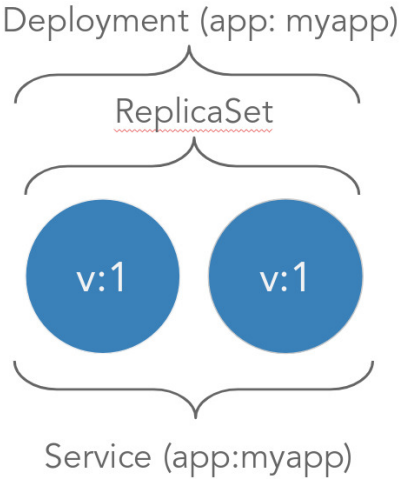
Scaling Deployments



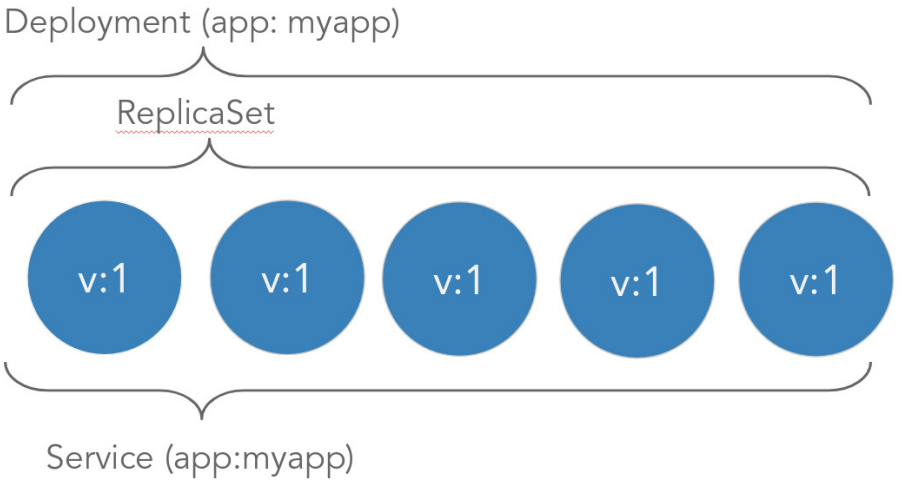
Scaling Deployments



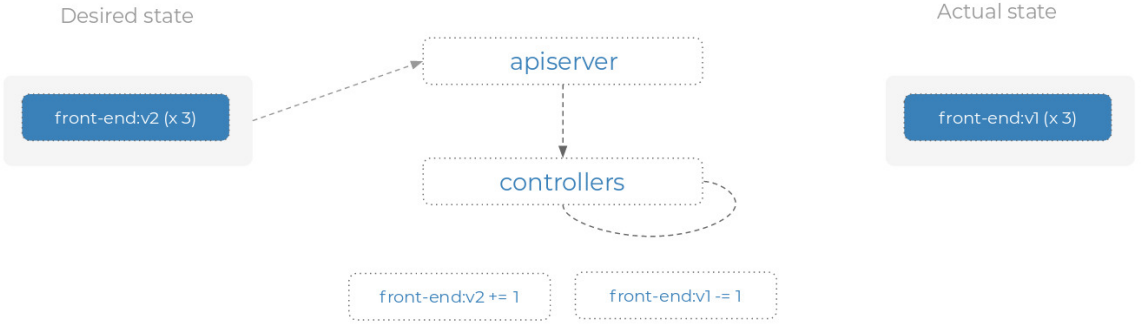
Scaling Deployments



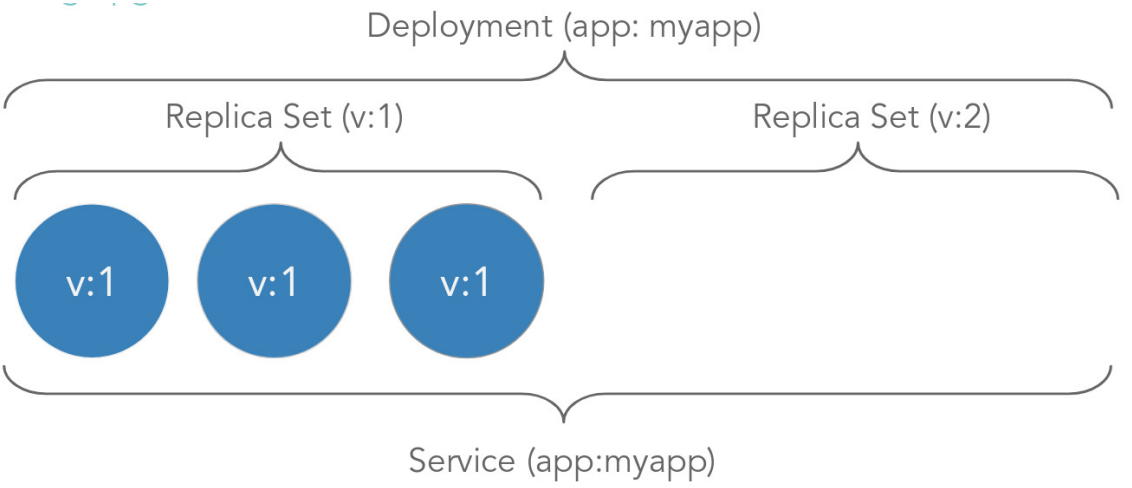
Scaling Deployments



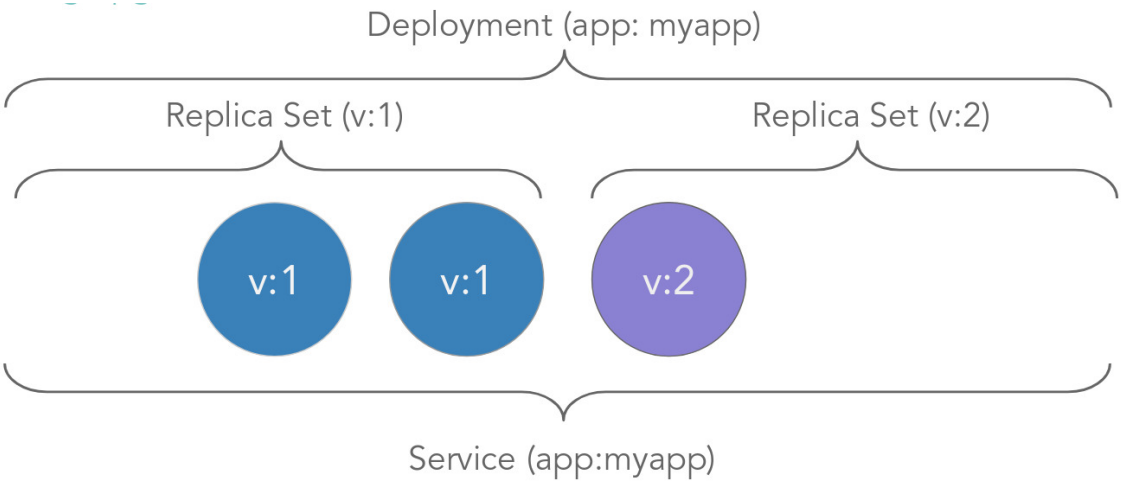
Scaling Deployments



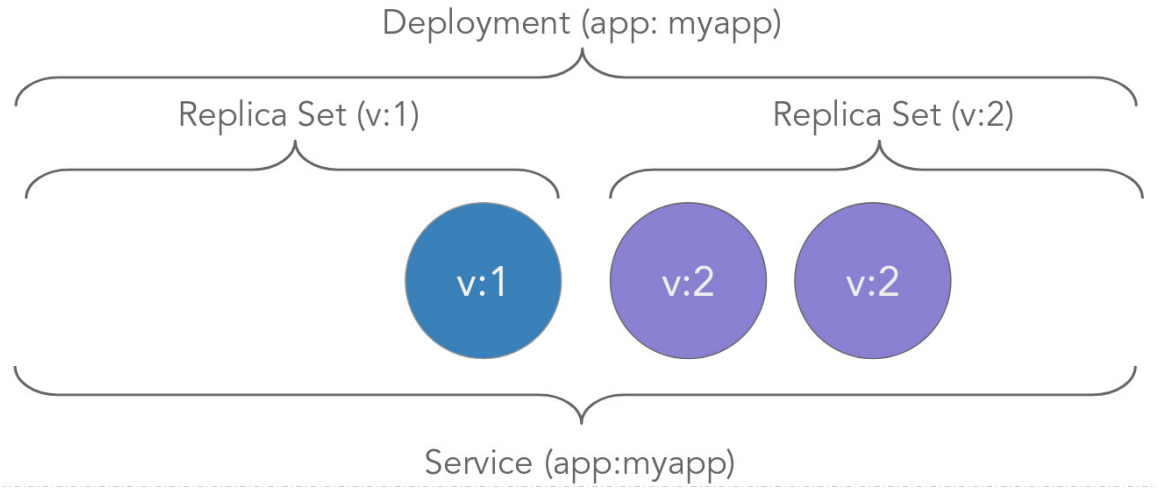
Scaling Deployments



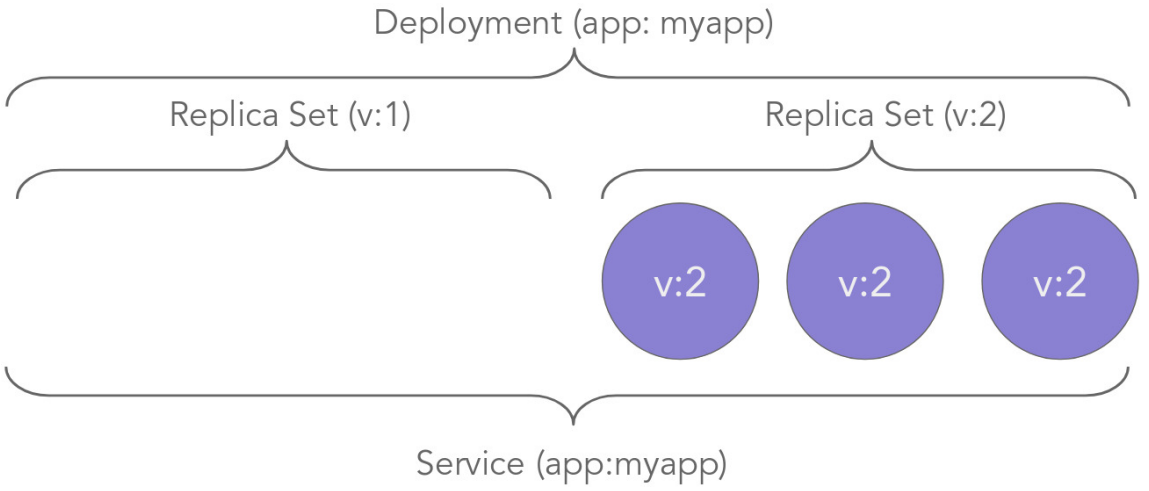
Scaling Deployments



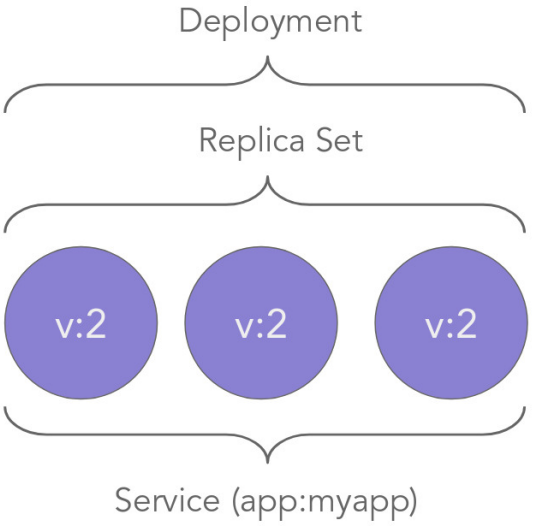
Scaling Deployments



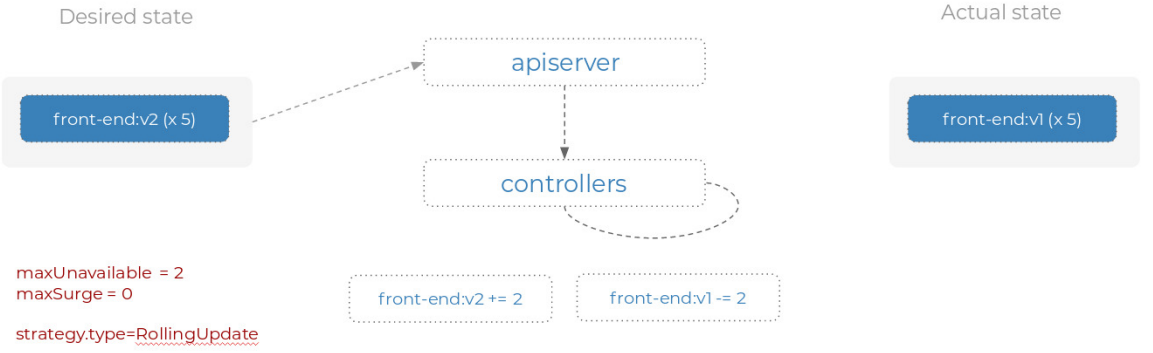
Scaling Deployments



Scaling Deployments

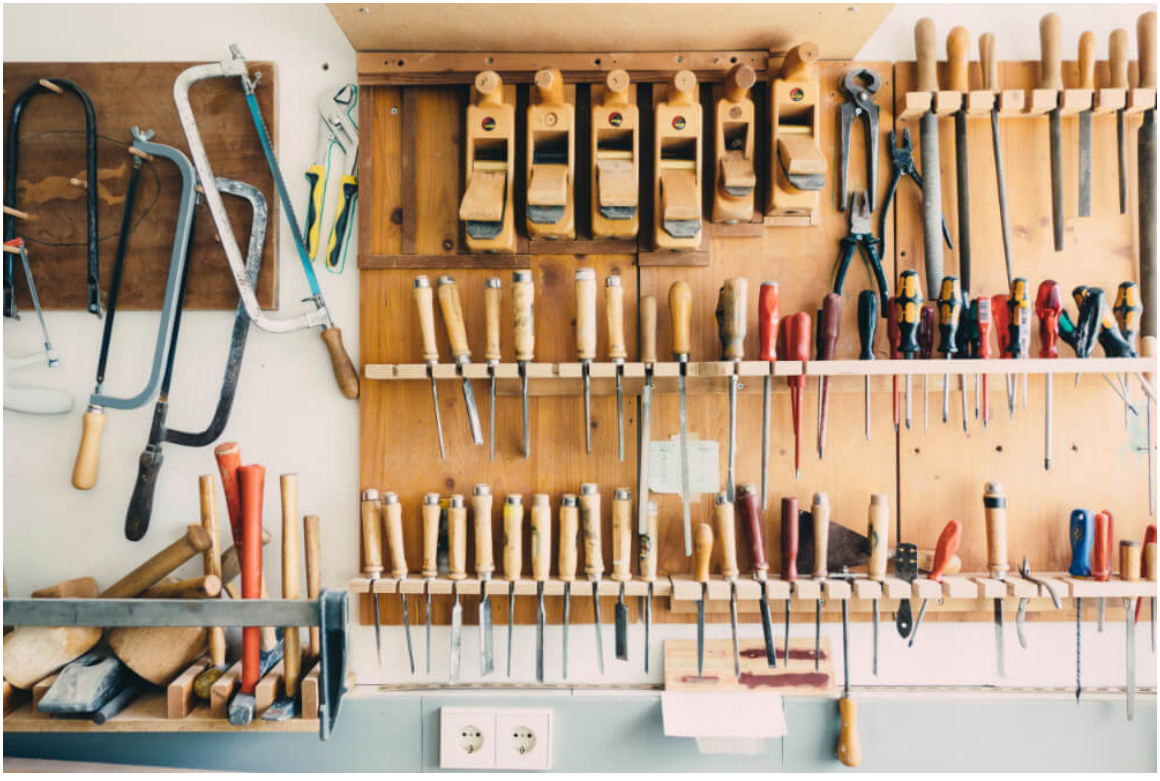


Scaling Deployments



Workshop

4. Deploy the sock-shop in a declarative way



Canary Deployments

- A common deployment pattern on Kubernetes
- Also referred to as an 'incremental rollout'
- New release of a pod/microservice is deployed side-by-side with the existing version to allow the new release to receive live production traffic

  Opportunity to test its functionality before
  - rolling out the deployment fully – or
  - rolling back to the existing deployment

This deployment pattern takes advantage of *labels*, which we discussed earlier.

Canary Deployments

```
kubectl explain deployment.spec.template.metadata.labels
```

In our sock-shop configuration we've defined a label on our front-end deployment:-

```
...
template:
  metadata:
    labels:
      name: front-end
...
```

We can bump this value to keep track of the fact that we're deploying a new version

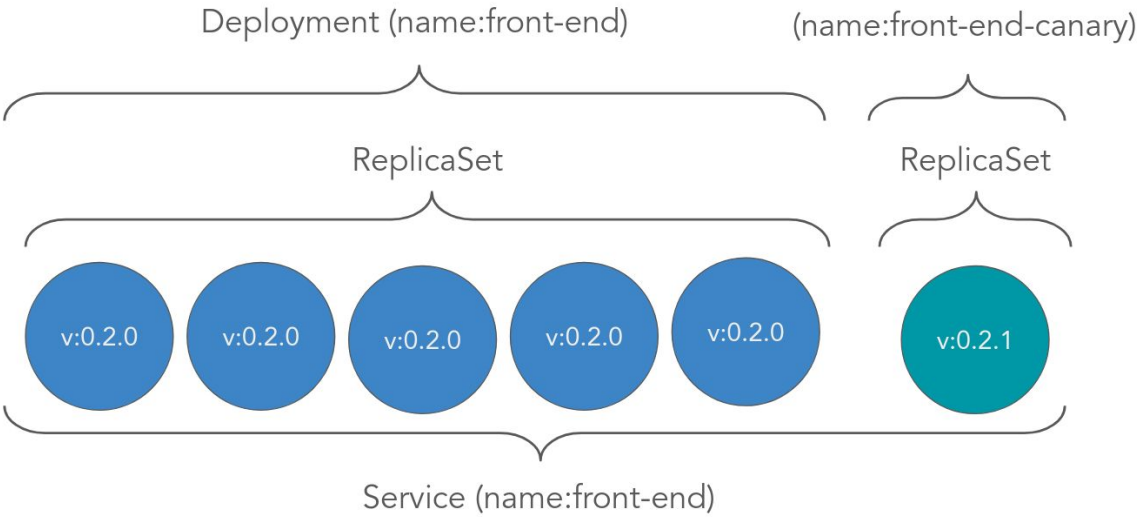
Canary Deployments

```
kubectl explain deployment.spec.template.spec.containers.image
```

The container image which is currently running in this deployment is specified in this field

```
...
  spec:
    containers:
    - name: front-end
      image: weaveworksdemos/front-end:0.3.12
...
```

Canary Deployments



Workshop

5a. Update a canary deployment to a bad image then rollback

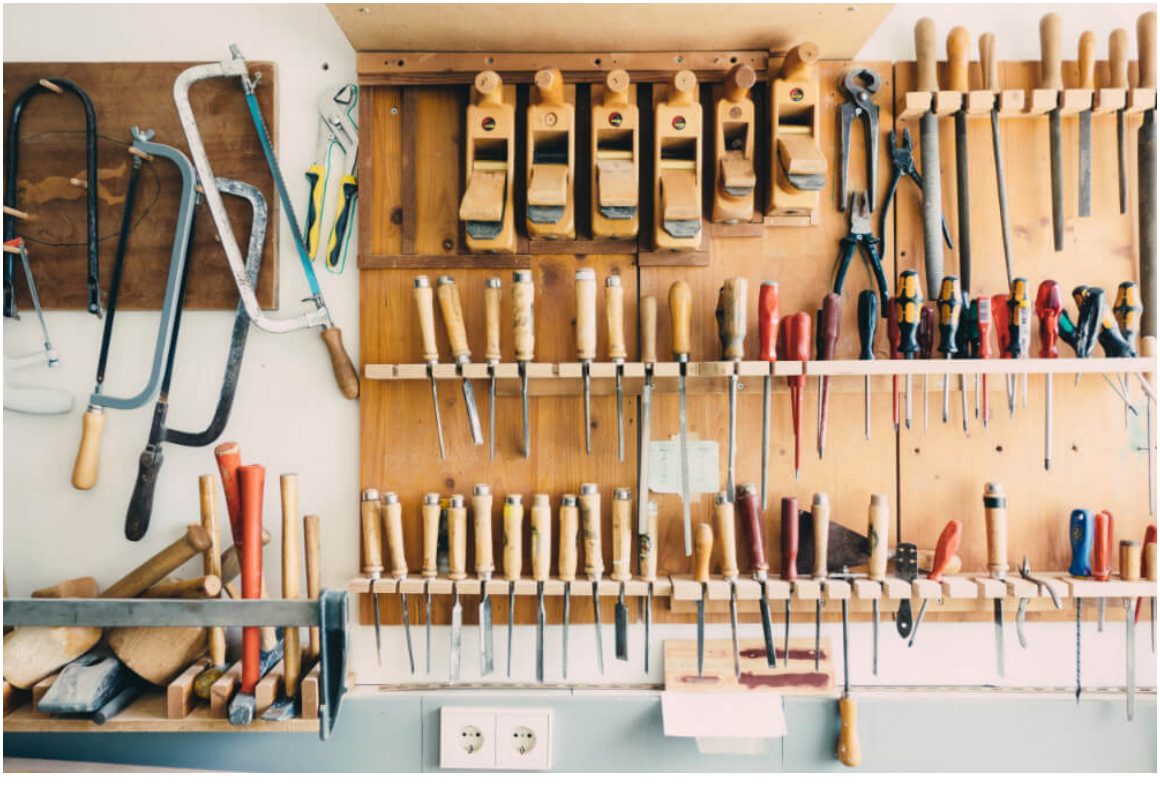


Photo by Barn Images on Unsplash

Readiness & Liveness

Tell Kubernetes how to monitor your pods

Liveness Probes

- Define rules about when to restart containers
- kubelet tests that a container is running correctly
- If not, the container will be restarted
- Useful in cases where the application is stuck
- One of the following types:
    - exec (command to be run inside the container – must complete with 0 exit status to succeed)
    - HTTP (request to a endpoint – success must have HTTP status code between >= 200 & < 400)
    - TCP (socket connection – success if the connection can be established)

Readiness Probes

- Define rules about when containers are ready to serve traffic
- kubelet tests that an application is ready to receive traffic
- If not, no traffic will be routed to it via k8s services
- Same options as liveness probes

Workshop

5b. Explore how liveness and readiness probe behave

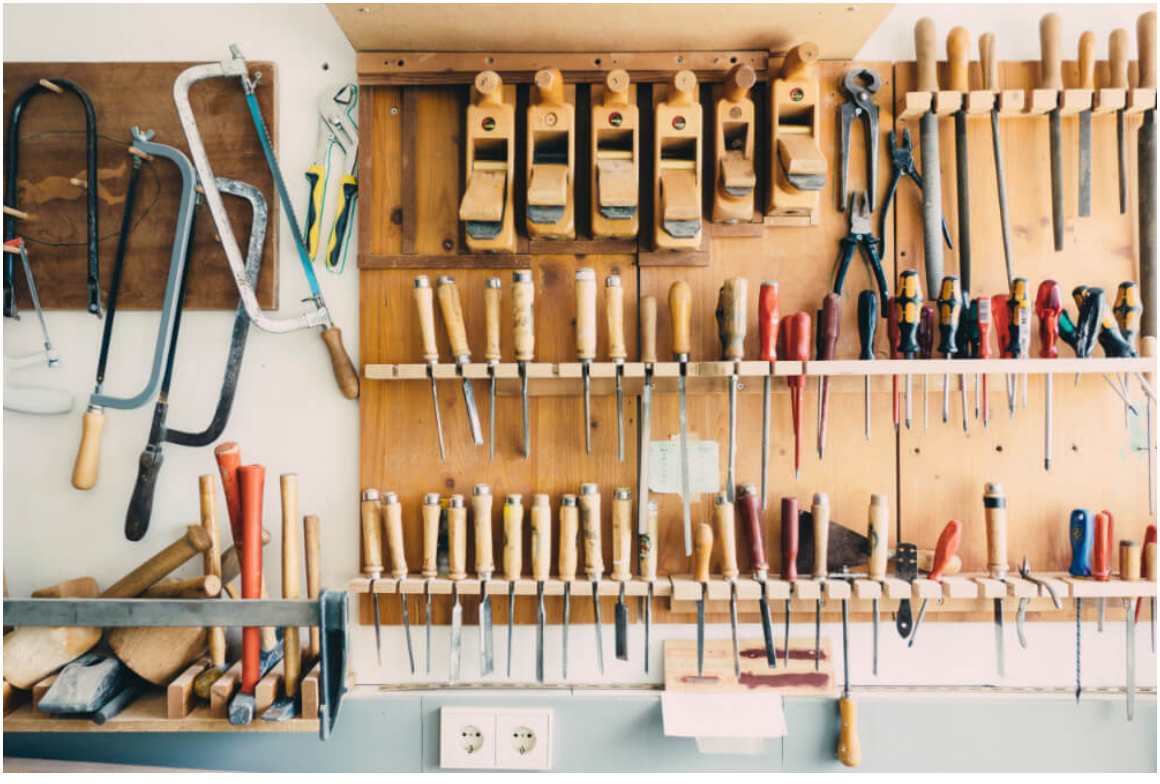


Photo by Barn Images on Unsplash

Discussion & Recap

Deployment Strategies Discussion

- Deployment strategies and how they fit with developers workflow
- Using namespaces
- Using labels
- Use-cases for deployment scenarios (dev, test, staging)
- Canary & blue/green deployments
- Controlling rollouts and down-time

Recap

- **Pods** – limits, requests, ports, containers
- **Labels** – and selectors to link pods with services
- **Services** – ClusterIP, LoadBalancer, NodePort
- **ReplicaSets** – ensure N pods exist
- **Deployments** – managed ReplicaSets
- **Scaling** – increasing and decreasing replicas
- **Probes** – checking if your application is healthy

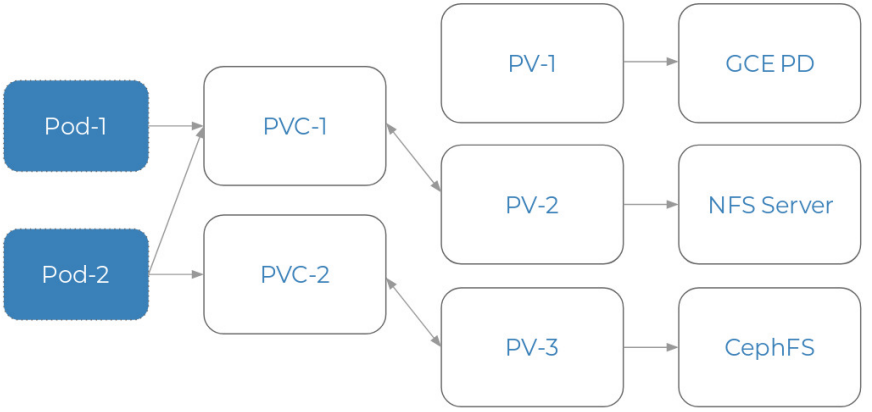
Storage & State

Configure applications that store state

Storage & State

- Disaster has struck! We were asleep having dreams of socks made from dollar bills and one of our 3 nodes blew up
- Even worse – it was the node that had the orders and payment database. Luckily we have a database backup from not so long ago but we do have to get out of bed at 4am to fix it

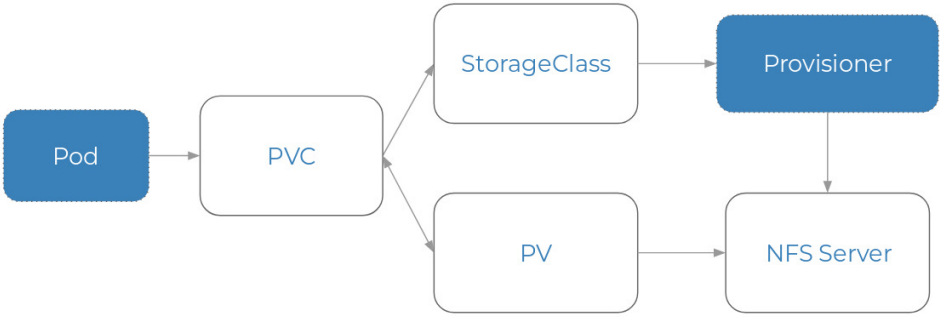
Storage & State



Storage & State

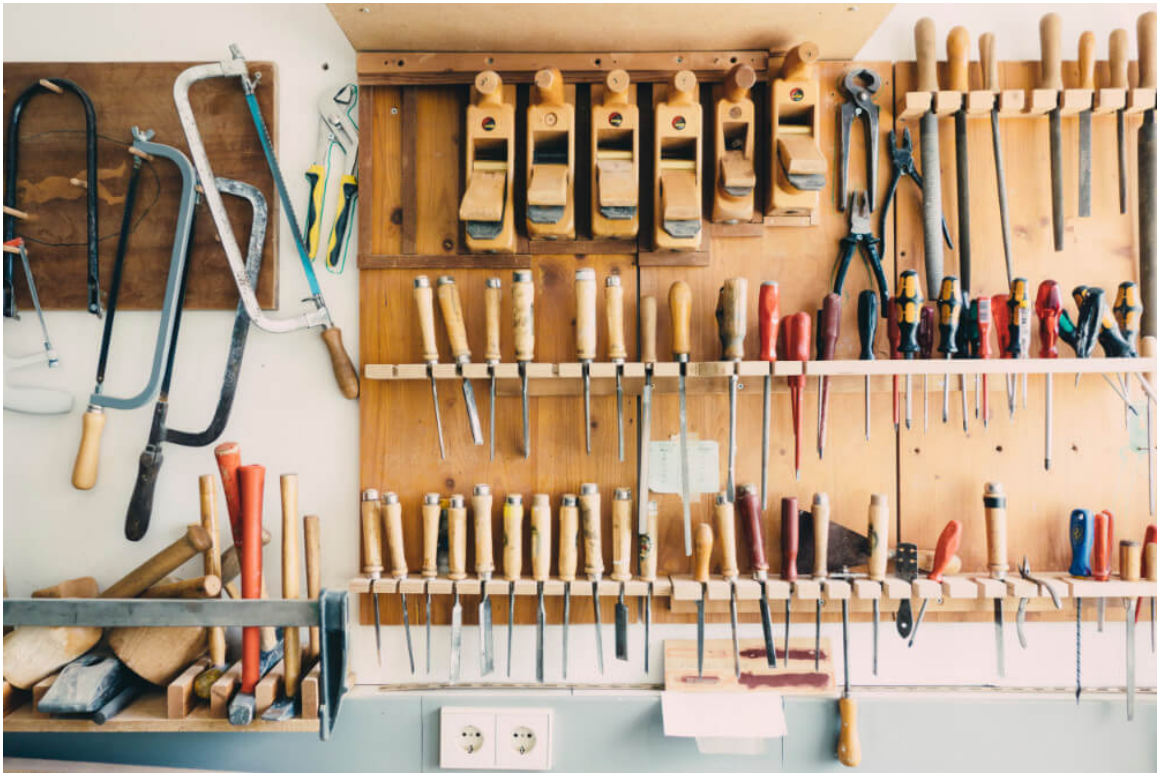
```
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
      protocol: TCP
    volumeMounts:
    - mountPath: /nfs
      name: nfs
    - mountPath: /gce-pd
      name: gce-pd
  volumes:
  - name: nfs
    nfs:
      path: /nginx
      server: my.nfs.server
  - name: gce-pd
    persistentVolumeClaim:
      claimName: my-gce-pvc
```

Storage & State



Workshop

6. Destroy the database pod



Persistent Volumes

- Portable / mountable block devices
- Volume drivers
- Kubelet handling of volumes
- Types of storage
- IOPS/performance settings
- Storage classes

Stateful Containers Discussion

- Types of storage

  Drivers

- Master/slave dbs (postgres)
- Peer-to-peer dbs (elastic)
- Auto provisioning

Secrets & ConfigMaps

Configuration as code

Secrets & ConfigMaps

Hackers from the ‘socks and sandals society’ defaced the site

Secrets & ConfigMaps

A hacker broke in overnight and defaced the sock-shop and renamed it to ‘socks and sandals shop’ - a **criminal** choice of name.

Secrets & ConfigMaps

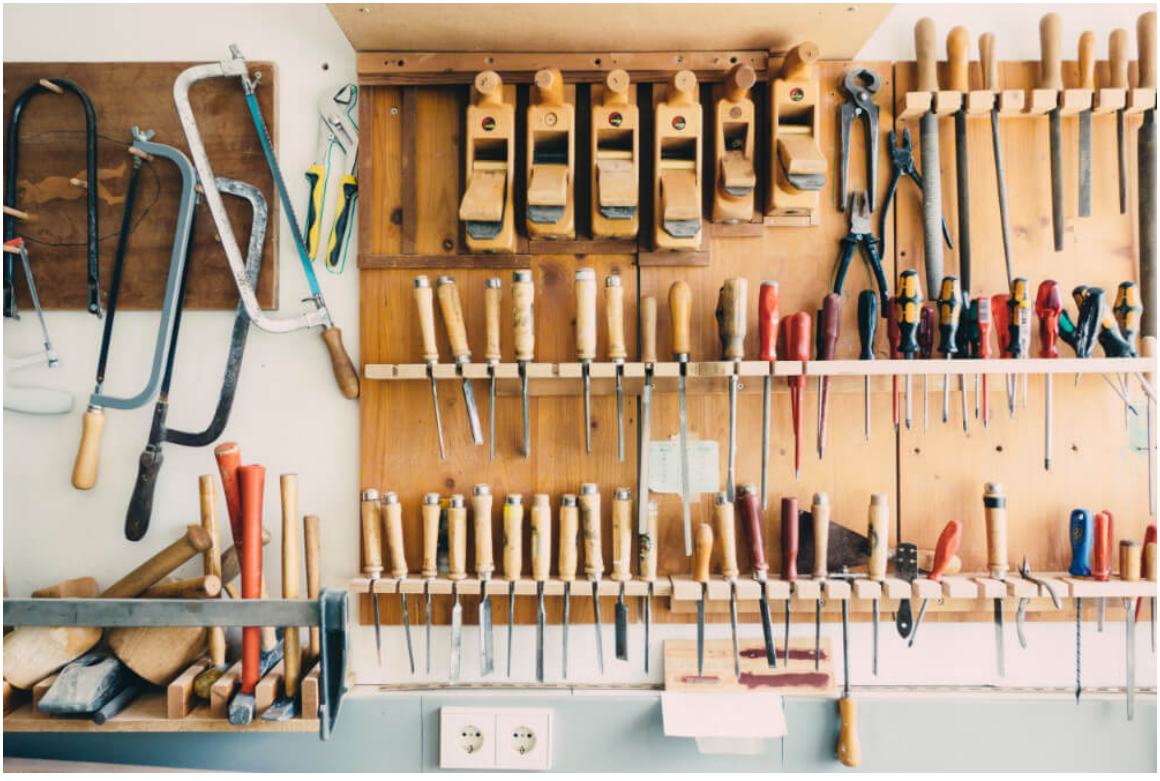
It turns out that our database password was “password” and before anyone finds out it’s our fault, **we need to fix it** .

Secrets & ConfigMaps

- Easy enough to embed secrets (e.g. passwords) and config into a container image - but please don't
- Best to determine this based on environment (12-Factor style)
- Use k8s Secret and ConfigMap API resources
- Consumable in a variety of ways in a pod manifest
  - Environment variables
  - Volumes
  - Command line arguments

Workshop

7. Secure the database password using ConfigMap



Logging & Monitoring

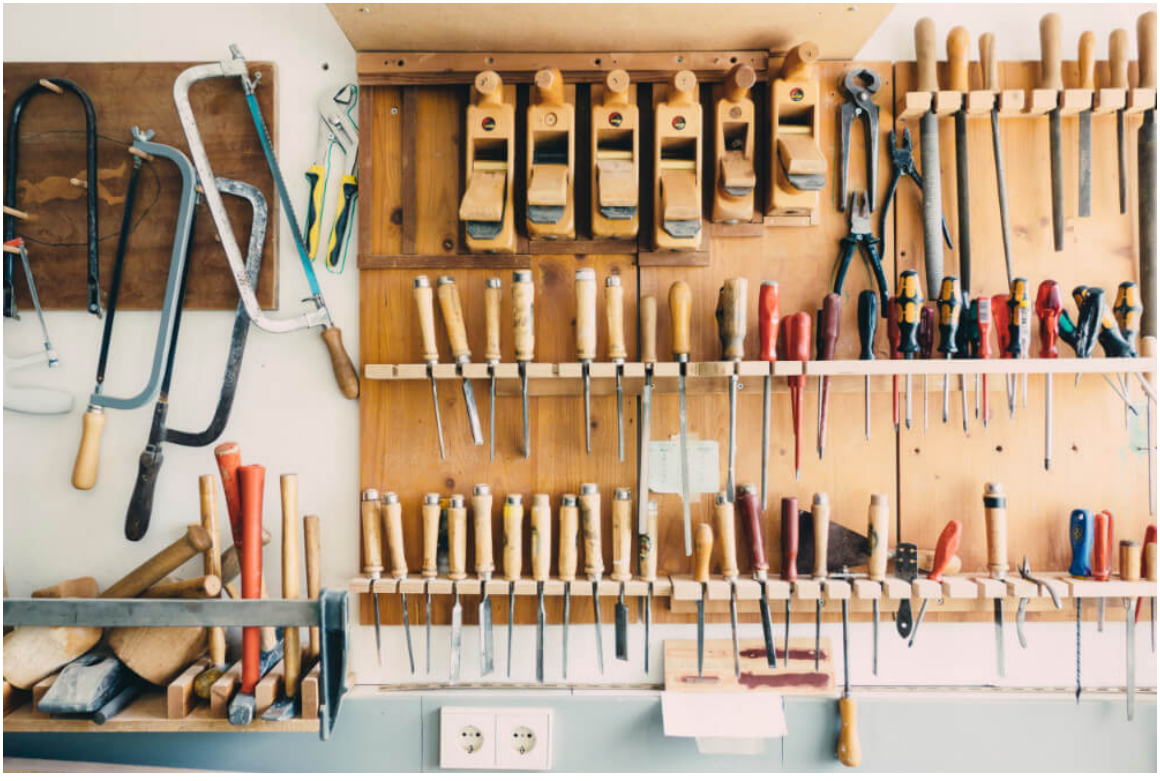
Keep tabs on your cluster and services

Logging & Monitoring

- Know what is happening in your cluster
  - kubectl logs
  - Cloud Logging
  - Dashboard
  - Stackdriver

Workshop

8. What's happening in your cluster?



Helm

A ‘package manager’ for Kubernetes

Helm

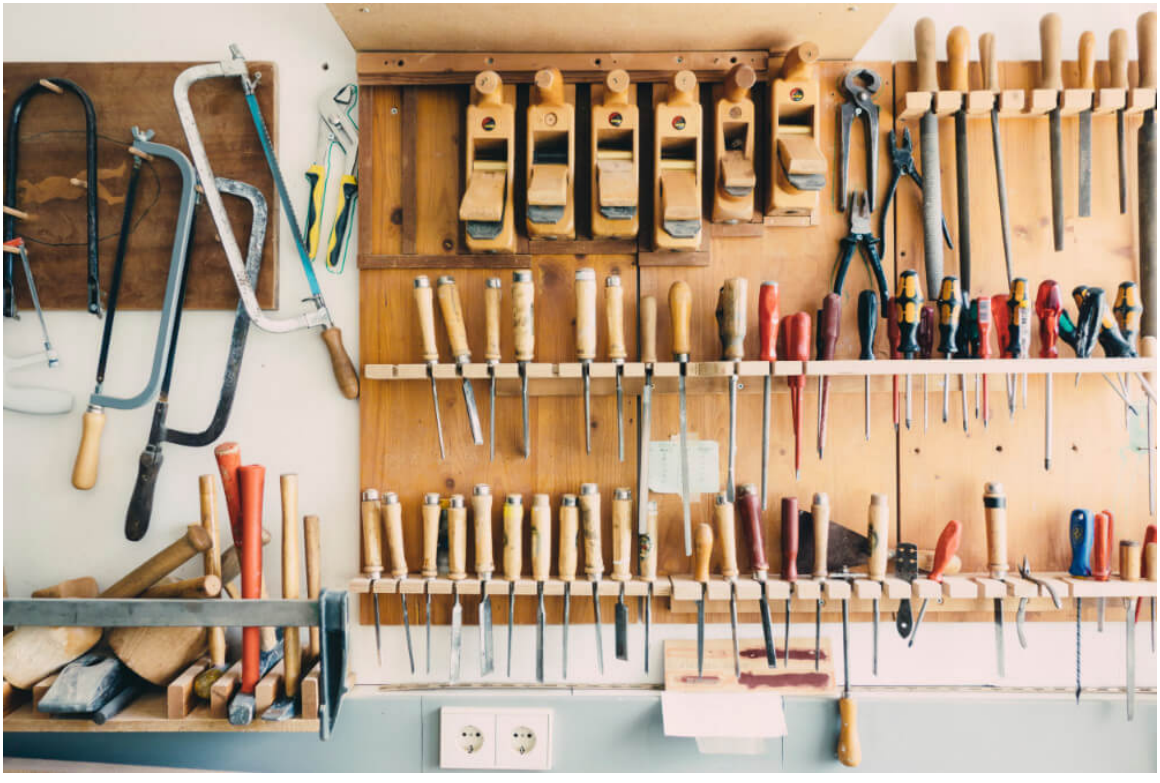
- Easily package your application into a single deployable unit
- A single application consists of many resources
- Difficult to manage, becomes messy
- Parameterise your application
- Makes it easy to deploy to different environments

Helm

- It's also possible to use Helm to deploy our own apps
- Useful for templating out different versions (perhaps for different environments)

Workshop

9. Write a simple helm chart for the Sock Shop and deploy it



The End

Photo by Zoltan Tasi on Unsplash