
Kubernetes Beginner
A workshop by Jetstack
Scanrail1/shutterstock

Schedule

Action-packed and fast-moving

Morning

- What is a container (and Docker intro)
- Getting a K8S cluster on GKE
- Introduction to Kubernetes resources

Afternoon

- Labels, Namespaces
- Stateless deployments, workloads
- Storage, statefulness
- Configuration: ConfigMaps and Secrets
- Templating with Helm

Welcome

Each person please say:

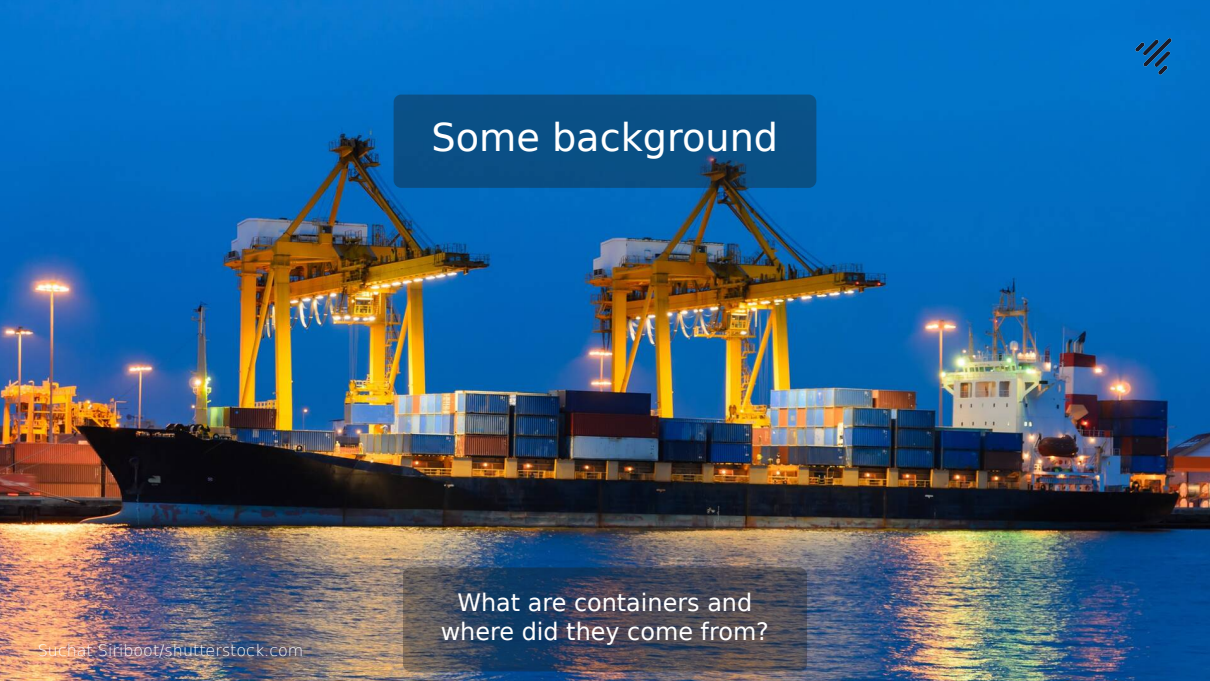
- Their name
- What is their background
- How much k8s experience they have
- The main thing they want to learn

Download the materials

Files and slides for you to keep

http://errge.k8s.school/

- Username: **errge**
- Password: **kubectl2019**
- Download and open the slides in a tab
- Keep this page open, we will need it for the labs
- Download the YAML files and unzip them on your computer
- Send slide and lab feedback to: gergely.risko@jetstack.io
- Join the course chat: https://tlk.io/k8s



Some background

What are containers and where did they come from?

Suchat Siriboot/shutterstock.com

Introduction

Technologies behind Kubernetes: Linux kernel and Docker

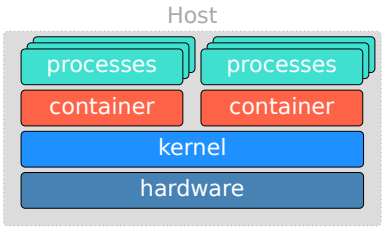
Recent developments that make Kubernetes possible:

- Linux namespaces/cgroups = containers
- Docker: the command line tools that make containers useful
- we discuss these, so you know what is behind the scenes
- this knowledge helps during debugging and docs reading

Then we discuss Kubernetes' goals

What is a container?

- An isolated group of Linux processes
- Shares the kernel with other containers
- Kernel-level virtualization features
  - **Namespaces**: isolation (PIDs, mounts, ...)
  - **Cgroups**: resource control (CPU, mem)

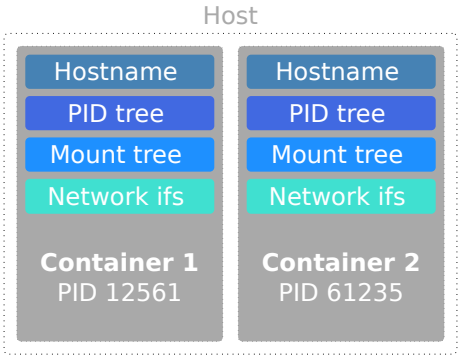


Namespaces

Feature of the Linux kernel to isolate processes

Isolate from each other and the host

- Hostname (called UTS)
- Process tree (PIDs)
- Mount points (including /)
- Network interfaces



Namespaces

PID tree

- the PIDs are local to the container
- `ps -A` behaves like in a fresh VM
- processes in the container see each other
- can’t kill processes outside

```
root@2e7169b7ef49:/# ps -A
  PID TTY          TIME CMD
    1 pts/0    00:00:00 bash
  260 pts/0    00:00:00 ps
root@2e7169b7ef49:/#
```

Namespaces

Network

- `ifconfig` shows a single interface
- Docker sets up the interface and the network bridge
- kernel does routing and filtering
- advantage: no more port collisions

```
root@2e7169b7ef49:/# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 888  bytes 8759414 (8.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 498  bytes 36945 (36.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
 root@2e7169b7ef49:/#
```

Namespaces

Mount

- `mount` system calls are isolated
- Docker mounts the container image to / (copy-on-write)
- Other volumes can be mounted anywhere in the container
- Inside one machine containers can share data through these other volumes
- Somewhat similar to chroot
- Non-container use case: solving conflicting shared libs
- Non-container use case: self-contained software deployment

```
$ docker run -it --rm \
    -v /home/errge/tmp:/home-tmp debian
root@2e7169b7ef49:/# mount
/dev/sda2 on /                  ← Docker's root image.
/dev/sda2 on /home-tmp          ← User's volume from host.
/dev/sda2 on /etc/resolv.conf ← Docker specific hacks for
/dev/sda2 on /etc/hostname      smooth intercontainer
/dev/sda2 on /etc/hosts         network access.
root@2e7169b7ef49:/#
```

Namespaces

Other, less important namespaces

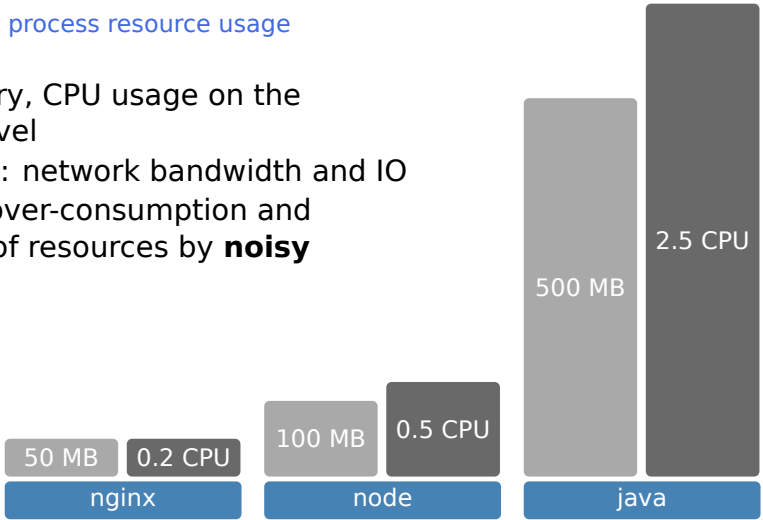
- **User**: the same process can have different UID/GID on the host and inside a container
- **IPC**: isolate various IPC mechanism that are not related to filesystem objects
- **Cgroup**: isolate the cgroup resource control mechanism

None of this is used by current Docker/Kubernetes.

cgroups

Kernel feature to limit process resource usage

- Limit memory, CPU usage on the container level
- In the works: network bandwidth and IO
- “Prevents” over-consumption and exhaustion of resources by **noisy neighbors**

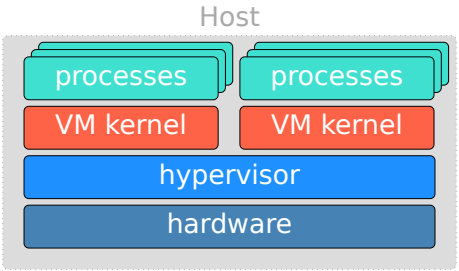


VMs vs Containers

Separated vs Shared Kernel

VMs run their own kernel

- You can run Windows and Linux on the same hypervisor (flexibility)
- Even possible to attach hardware directly to a VM (e.g. video card)
- The kernel has to be loaded in each VM (memory waste)
- The host kernel doesn't see the processes inside the VM (difficulty)

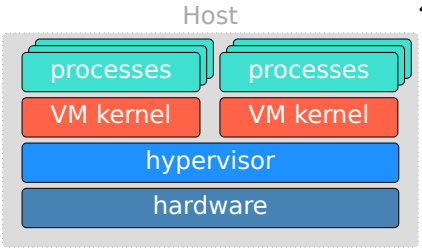


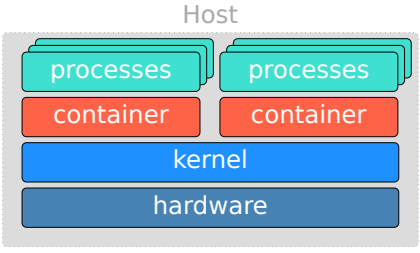
VMs vs Containers

Separated vs Shared Kernel

Containers

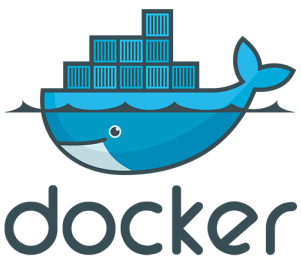
- Linux only (shared kernel)
- Isolation happens in the kernel, not in the hardware
- Start-up time reduced, density increased
- Easy to have a bunch of **stateless** containers
- Possible to use them in VMs without VM nesting





Docker born

Containers with a friendly API

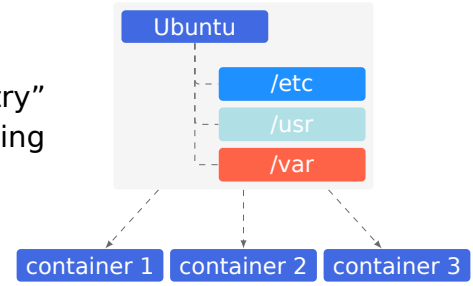


- Docker announced at PyCon 2013
  - For many the first experience with containers
- Massive adoption, thanks to:
  - Easy to use API
  - Container image format
    (Mount namespace++: images, copy-on-write)
  - Docker Hub container registry to push/pull images
    - The “appstore of containers”
    - Free to publish, free to consume
    - Monetized if you are not working on free software

Docker

Overview

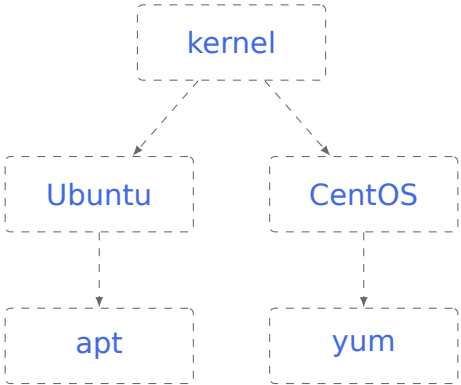
- An **image** is essentially a root fs:
  - It can be distributed as a `.tar.gz`
  - Or it can be “pulled” from a “registry”
- Containers are run with the image being the root filesystem
- Images are shared (copy-on-write) between running containers



Docker

Images and Linux distributions

- A GNU/Linux distribution is the kernel plus an opinionated file layout
- Since the containers share the kernel, all we need is the file layout
- This is what an image provides



Docker

Image description: Dockerfile

Images are made up of layers, specified in a Dockerfile (a build file).

```
FROM node:5.2.0-slim ← Base layer to start from
ADD . /app           ← Layer id 1234, adding a directory
WORKDIR /app         ← Container config: workdir
RUN npm install      ← Layer id 5678, storing the result
ENTRYPOINT ["node","/app/index.js"]
    ↖ Container config: what to start when this image is ran
```

We can use `$ docker build -t nodeapp:0.1 src` to build this and name the result nodeapp version 0.1 (with a so called *tag*).

Docker

Running containers

```
$ docker run --name simple busybox sh -c "echo hello"
```

This command has a lot of parts:

- `docker`: calling the Docker executable
- `run`: the docker subcommand, similar to git
- `--name simple`: docker subcommand options
- `busybox`: docker run parameter (image to run)
- `sh -c "echo hello"`: docker run parameter (container cmdline)

Docker

Dealing with Docker, useful commands

- Listing running containers: `$ docker ps`
- Listing detailed info: `$ docker inspect mycontainer`
- Listing all containers: `$ docker ps -a`
- Removing a container: `$ docker rm mycontainer`
- Removing all container: `$ docker rm $(docker ps -aq)`
- Listing images: `$ docker images`
- Removing an image from local cache: `$ docker rmi myimage`

Docker registries:

- Used when you need an image that you don’t have in the local cache
- This download is called “pulling” and needs a registry (server)
- The default is Docker Hub, it is free for public, paid for private
- Google has GCR (Google Container Registry), paid
- There is quay.io (CoreOS), paid for private
- The treescale.com site gives you some private registry for free
- You can also self-host your own registry

Google and containers

Tried-and-tested at massive scale

- (Almost) everything at Google runs in a container (even VMs)
- 10+ years and major kernel contributions from Google (LXC/namespaces, cgroups)

4

BILLION

containers

per

week

Google and containers

Borg

- Internal Google tool to treat a data center as a big shared resource pool and schedule containers there
- Published in 2015, secret before
- “Borglet” run on each node
- “BorgMaster” coordinates the cluster
- Paxos for distributed consensus amongst BorgMasters
- Doesn’t use Docker (predates it)

Large-scale cluster management at Google with Borg

Abhishek Verma[1] Luis Pedrosa[2] 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 and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

1. Introduction

The cluster management system we internally call Borg admits, schedules, starts, restarts, and monitors the full range of applications that Google runs. This paper explains how.

Borg provides three main benefits: it (1) hides the details of resource management and failure handling so its users can focus on application development instead; (2) operates with very high reliability and availability, and supports applications that do the same; and (3) lets us run workloads across tens of thousands of machines effectively. Borg is not the first system to address these issues, but it’s one of the few op-

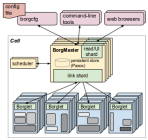


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

cluding with a set of qualitative observations we have made from operating Borg in production for more than a decade.

2. The user perspective

Borg’s users are Google developers and system administrators (site reliability engineers or SREs) that run Google’s applications and services. Users submit their work to Borg in the form of *jobs*, each of which consists of one or more *tasks* that all run the same program (binary). Each job runs in one Borg *cell*, a set of machines that are managed as a unit. The remainder of this section describes the main features exposed in the user view of Borg.

Kubernetes born

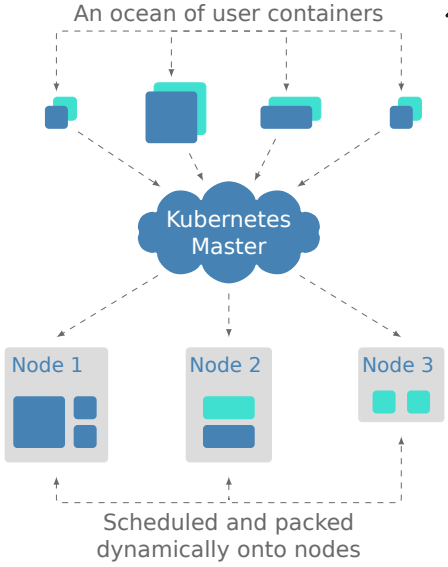
Based on 15 years of experience in Google with containers

- Announced mid-2014 at OSCON
- The open-source “reimplementation” of Borg
- Built on top of Docker
- Started by the same folks who worked on and wrote the paper for Borg (tried and tested)
- Docker is for one machine, Kubernetes is for a scalable cluster!

Kubernetes

Overview

- Handles container “Cattle”
- **Visibility:** Oversight of what is currently running on the cluster
- **Reproducability:** Describe resources in declarative files, infrastructure can be in git
- **Portability:** Works on many different types of infrastructure
- **Resilience and elasticity:** pro-actively monitors, scales, auto-heals and updates

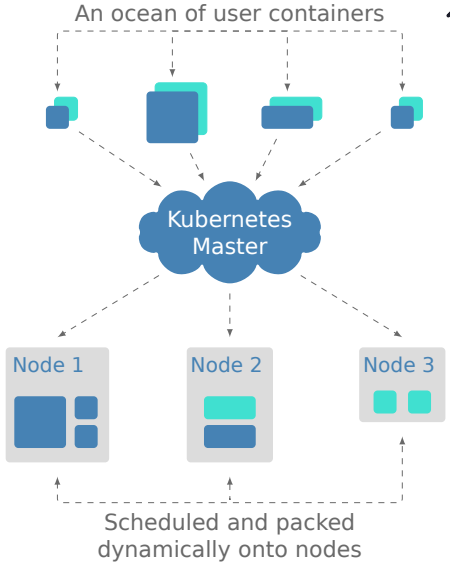


Kubernetes

Declarative system description

Application abstraction

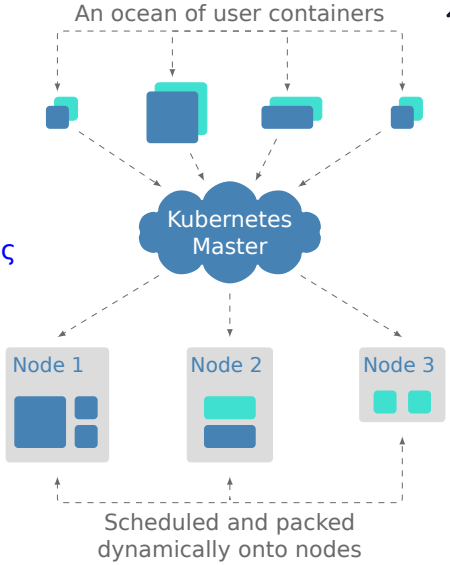
- Pods
- Replica Sets
- Deployments
- Services
- Persistent Volumes
- Persistent Volume Claims
- ConfigMaps, Secrets
- .. and many more!



Kubernetes

Project details

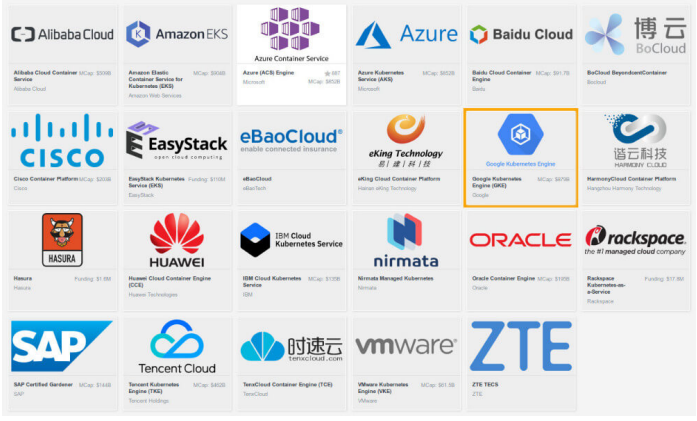
- Often abbreviated as k8s
- We also call it “Kates”
- Etymology
  - https://en.wiktionary.org/wiki/κυβερνήτης
  - Captain, pilot, navigator
- Actively maintained
- Huge community on GitHub
  `https://github.com/kubernetes`



Kubernetes

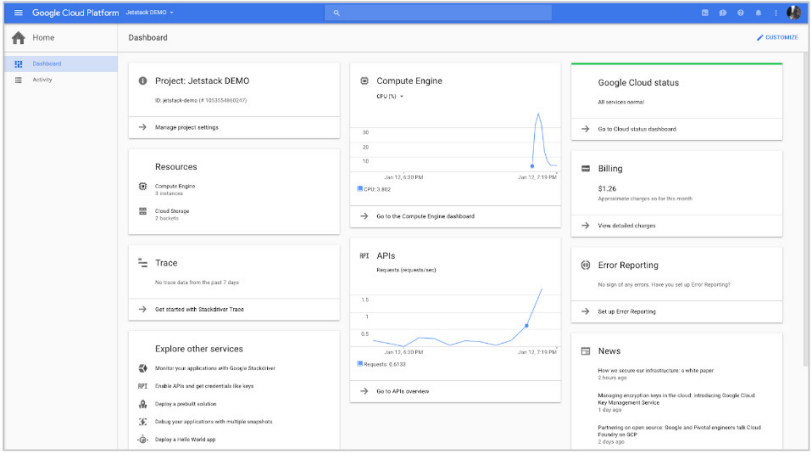
Infrastructure agnostic

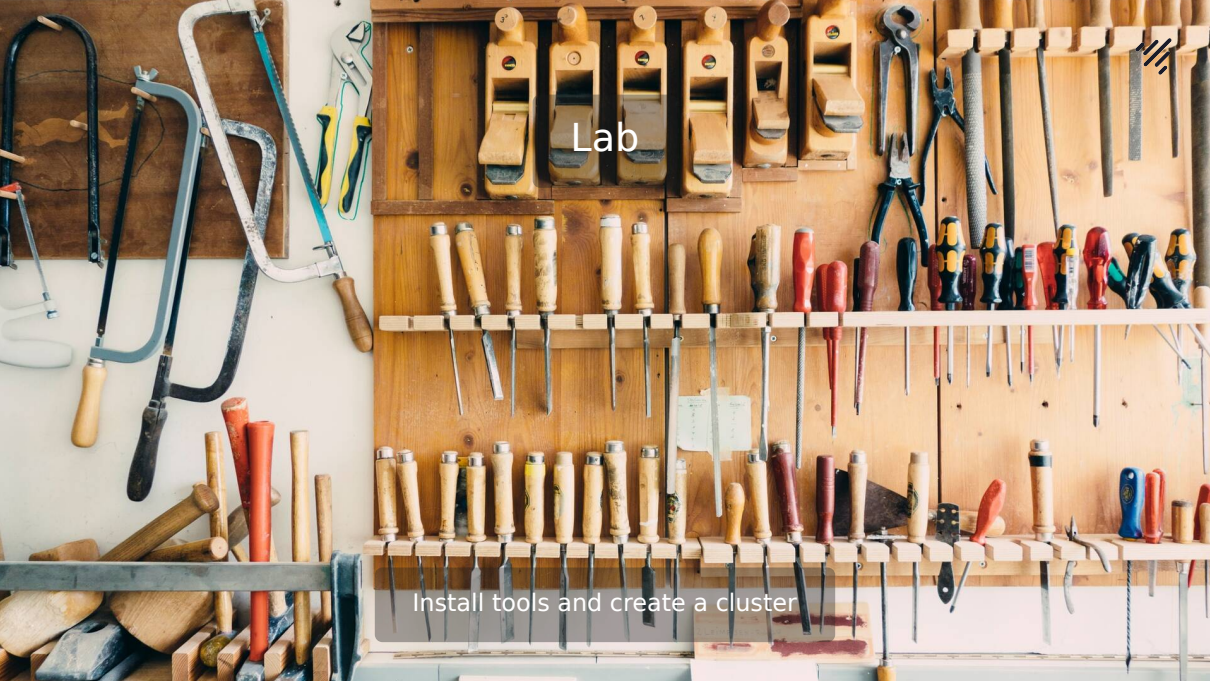
https://kubernetes.io/docs/setup/



Kubernetes

We are going to use GKE (Google's hosted solution)



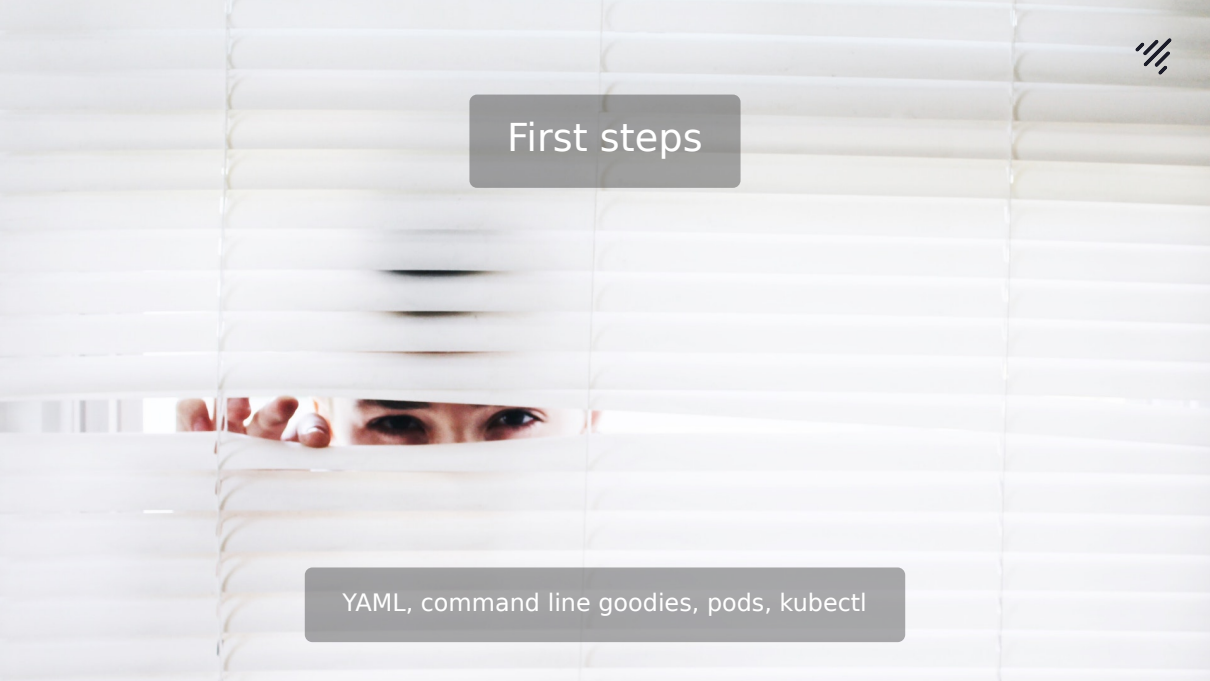

Lab
Install tools and create a cluster

Lab: Install tools and create a cluster

Get a 3 node Kubernetes cluster and access it

- GCP (Google Cloud Platform) setup
  - Login to `@gmail.com`, project creation, billing setup, free credits!
  - Install and initialize command line tools on your laptop
  - Enable GKE APIs on GCP
- Create a cluster with the `gcloud` command
  - 3 nodes, that are paid from the $300 we got
  - Master paid and managed by Google for free forever
  - Some sane customizations, feel free to ask for details!
- Connect with `kubectl`
- Add an event to your calendar NOW to delete this GCE project
  - Otherwise the $300 of credits will be spent in 2-3 months
  - You won't be charged even if you forget about this

First steps



YAML, command line goodies, pods, kubectl

First steps

Section overview

- YAML: the language of Kubernetes manifests (config files)
- Pod: the smallest unit of computation that you can run on K8S
- `kubectl`: the command line utility that you will use all day
- we also set up command line autocompletion and help

YAML

Basics

- Practice URL: `https://www.json2yaml.com/`
- YAML is similar to JSON, but more powerful and **very tricky**
- Literals
  - strings like webserver
  - numbers like 12
  - true false null
- Sequences
  - `[1, "whatever", true]`
  - or indented lists with hyphens
- Dictionaries (objects)
  - `{JSON: like, oneliner: pairs}`
  - or indented key:value pairs

YAML

Mapping of sequences and sequences of maps

```yaml
fibonacci:
  - 1
  - 1
  - 2
  - 3
  - 5
factorial: [1, 2, 6, 24, 120]
```

```yaml
- v: 1
  fib: true
  fac: true
- v: 2
  fib: true
  fac: true
- { v: 3, fib: true, fac: false }
```

YAML

Exercises: how to get these?

```
[
  [
    1,
    "fibonacci",
    "factorial"
  ],
  [
    2,
    "fibonacci",
    "factorial"
  ],
  [
    3,
    "fibonacci"
  ]
]
```

```
{
  "1": {
    "fibonacci": true,
    "factorial": true
  },
  "2": {
    "fibonacci": true,
    "factorial": true
  },
  "3": {
    "fibonacci": true
  }
}
```

YAML

Bug Squashing

```yaml
"1":
  - fibonacci: true
  - factorial: true
"2":
  - fibonacci: true
  - factorial: true
3:
  - fibonacci: true
```

```yaml
swiss_2016_basic_income:
  yes: 23.1
  nah: 76.9
js_address:
  street: Hopton Street
  no: 65
why_do_we_hate_java:
  null: 40%
  memory_usage: 40%
  verbosity: 20%
```

With great power comes great responsibility!

YAML

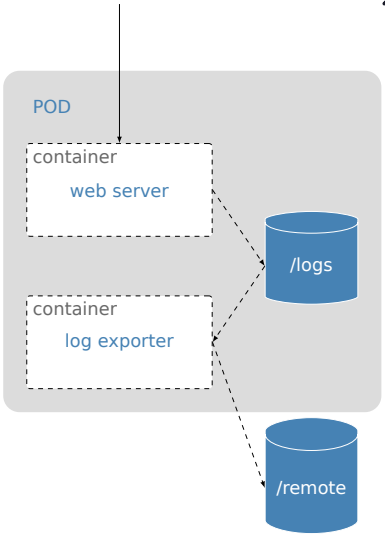
Yet more warnings

- Editors sometimes put spaces and tabs into files
- Try to use all spaces when editing YAML
- The YAML files in Kubernetes are case sensitive, so pay attention to `apiVersion` (vs `apiversion`)

Pod

The smallest deployable unit of computing on Kubernetes

- One or more containers
- The containers:
  - Share network NS and IPC NS (shared memory)
  - Can share volumes, doesn't share root volume
  - Share 1 IP
  - Share fate (eviction, deletion)
  - Share pod readiness
  - Do not share UTS NS (but same hostname)
  - Do not share mount NS and pid NS
- Usual extra containers in pods:
  - data pullers, pushers, proxies, etc.



Pod

Basic example YAML

```yaml
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
```

- Name has to be unique within a Kind
- Every Kubernetes object has a Name (even autogenerated ones)
- Almost all of them have a spec section
- (We will talk about labels.)

Getting ready

Goodies

Here are some goodies!

- https://kubernetes.io/docs/reference/kubectl/cheatsheet/
- Get exact documentation in the command line: `kubectl explain` (demo with pods)
- For bash users: `source <(kubectl completion bash)` (in any terminal window or in your `.bashrc` or `.profile`)

Getting ready

kubectl versions

- Check: `$ kubectl version` (current client and cluster version)
- Quite usual versioning: major.minor.patch
- Server/client compatibility rule: minor version +-1 difference
- Best to have the same version, if you can
- Symptoms: random commands fail with cryptic messages

Getting ready

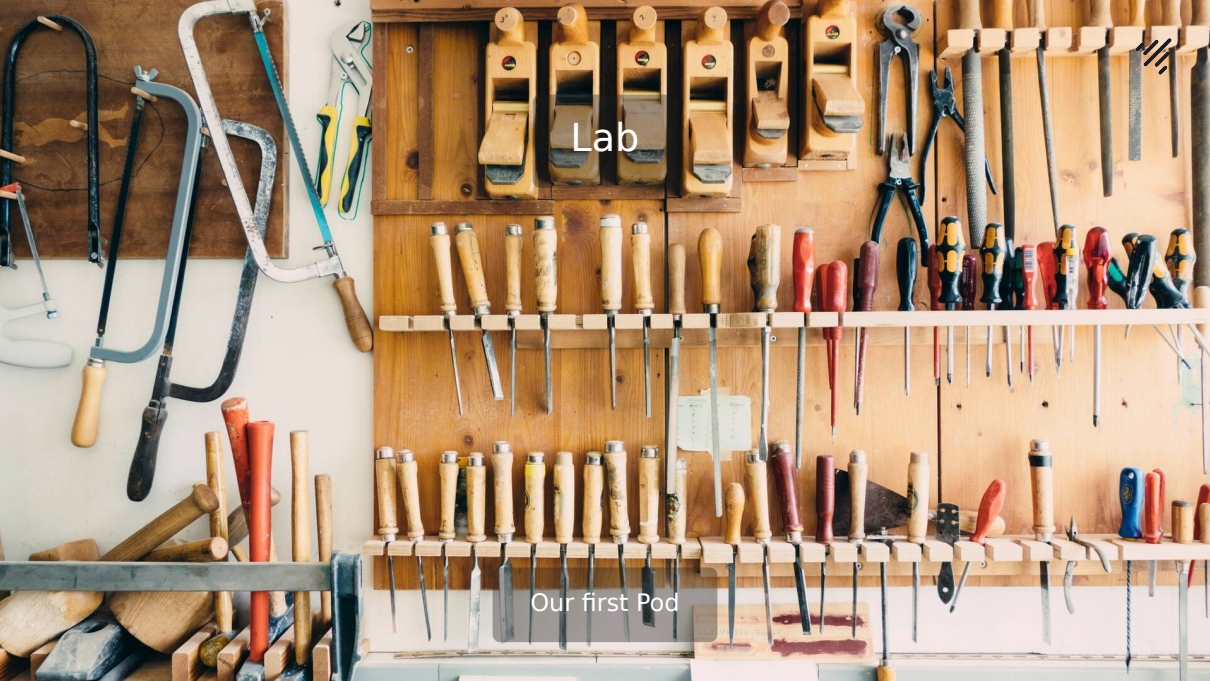
Obtaining specific kubectl versions

How to have the same version?

- OSX: https://storage.googleapis.com/kubernetes-release/release/v1.11.10/bin/darwin/amd64/kubectl
- Win: https://storage.googleapis.com/kubernetes-release/release/v1.11.10/bin/windows/amd64/kubectl.exe
- Lnx: https://storage.googleapis.com/kubernetes-release/release/v1.11.10/bin/linux/amd64/kubectl

Trick on linux:

```
gcloud compute scp <node>:/home/kubernetes/bin/kubectl ~/bin
```


Lab
Our first Pod

Lab: Our first Pod

Run your first pod on our new cluster

- Create a pod manifest with a text editor (e.g. `vi`, `mcedit`, `joe`, `emacs`)
- Use `kubectl apply` to deploy it
- Use `kubectl` commands to inspect the status of the cluster
- Use `kubectl port-forward` to see the webpage
- Nice to know: `get xxx -o wide` and `get xxx -o yaml`
- Cleanup: we delete this test pod

Scheduling

How many pods are fit together?

Scheduling

Section overview

We discuss:

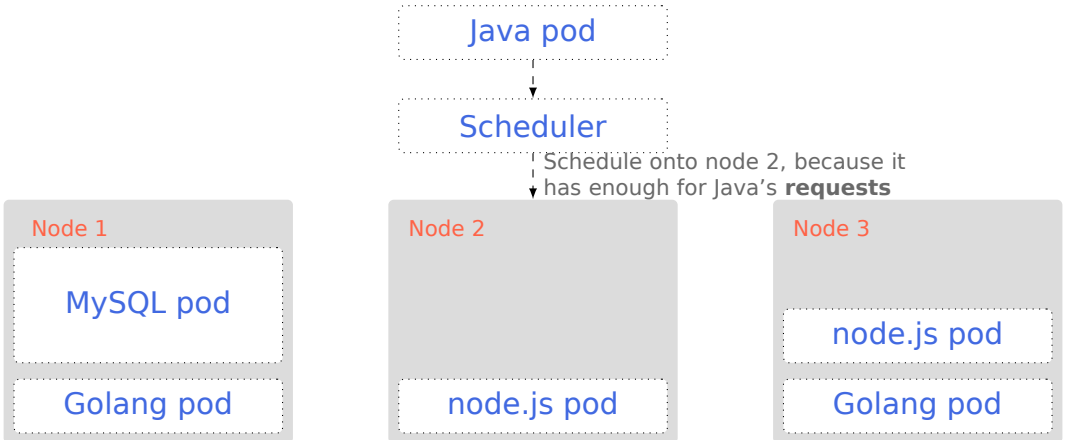
- K8S scheduling: how do we fit many pods together
- Keywords: limits and requests
- Example scenarios

Reference info:

- https://kubernetes.io/docs/tasks/administer-cluster/out-of-resource
- https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/
- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/resource-qos.md

Pod scheduling

Java is being modest

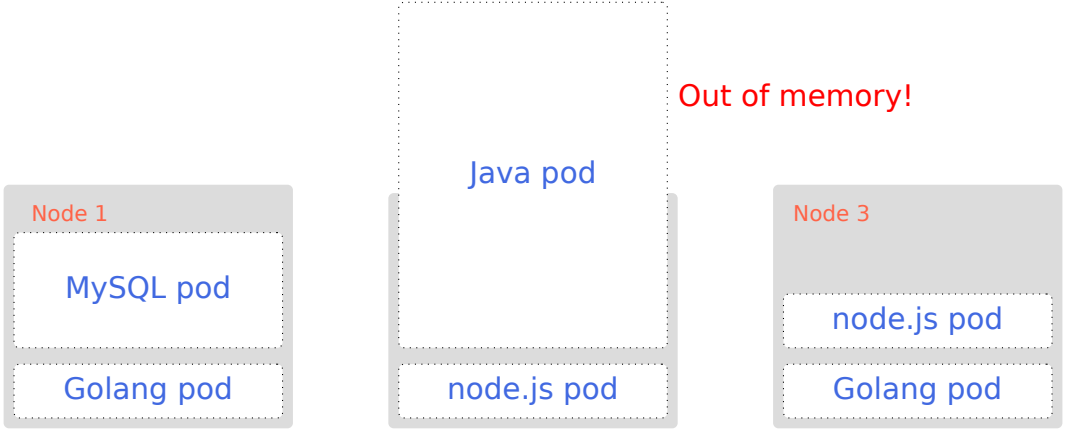


Pod scheduling

Java is starting up, looking good

Pod scheduling

This is why we also need **limits**



Pod limits

An example

```yaml
spec:
  containers:
  - image: gcr.io/google_containers/serve_hostname
    name: kubernetes-serve-hostname
    resources:
      limits:
        cpu: "0.9"       # Throttling if tries to use more
        memory: 512Mi    # OOM kill if tries to use more
      requests:
        cpu: "500m"      # Info for scheduling and Docker.  Chances for
        memory: 256Mi    # eviction increase if we use more than requested.
```

Pod requests and limits

K8S rules for QoS categorization

BestEffort

If **no** requests **and no** limits are set **neither** for memory **nor** for CPU, in **none** of the containers.

Guaranteed

If **limits == requests** are set for **both** memory and CPU on **all** containers. (Requests defaults to limits, so it's enough to set limits.)

Burstable

In all other cases. E.g.: limits are different than requests for some container or limits are simply unset for an other.

The QoS prioritization of a pod can be debugged with

```
kubectl describe pod xxx | grep QoS
```

Pod requests and evicton

Order of OOM eviction when usage > request

OOM scores are adjusted by Kubernetes, so that the system evicts unruly pods in the following order:

- BestEffort
- Burstable
- Guaranteed
- Kubelet, Docker

There are also explicit pod priorities (stable in Kubernetes v1.14):

kubernetes.io/docs/concepts/configuration/pod-priority-preemption/

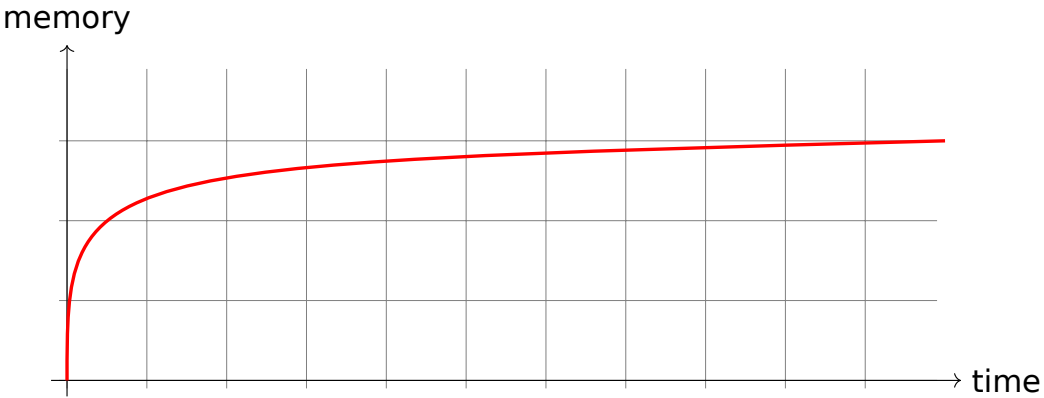
Pod requests and limits

Notes and Best practices

- Never run production stuff without **requests**!
- If you don't give info about how big are the pieces,
  the scheduling algorithms inside Kubernetes will be useless
- `kubectl top pod --containers` can be used to get an initial idea
- Gotcha: an i3 core is not the same as an i7 core
- The scheduler doesn't overcommit in the requests,
  but does overcommit in the limits

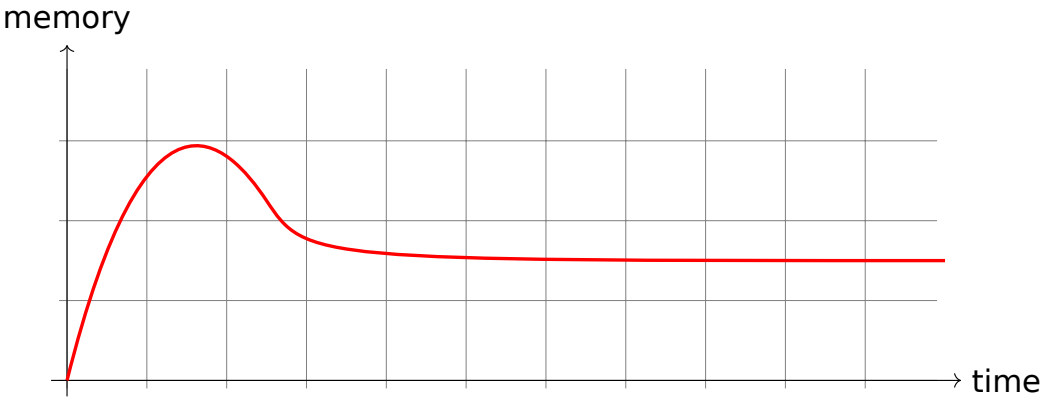
Pod requests and limits

How to set request and limit?



Pod requests and limits

How to set request and limit?




Networking
Labels and services

Networking with labels and services

Section overview

We discuss:

- **Labels**: mechanism to tie objects together in K8S
- **Services**: load balancing and stable identities on the network

Difficult and complicated material, ask questions!

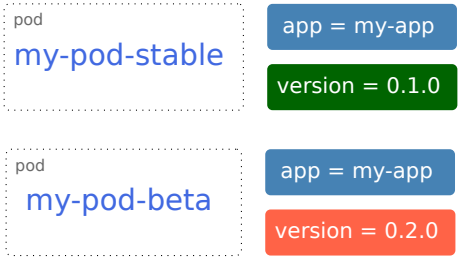
Reference info:

https://kubernetes.io/docs/concepts/services-networking/service/

Labels

General selection/tagging mechanism throughout K8S

- Key-values attached to any object (Pods, Deployments, Nodes, etc.)
- Used as identifying attributes throughout Kubernetes
- Mutable (without delete and recreate)
- Queried using selectors (set operations, equality)

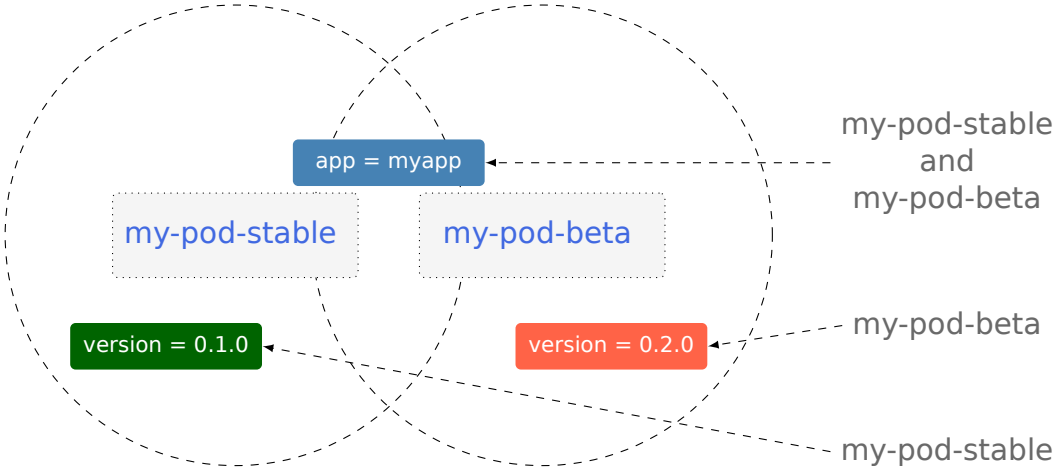


Then some selectors:

- `app=myapp`
  matches both
- `version=0.1.0`
  matches only `my-pod-stable`

Labels

Querying with selectors



Labels

Making use of labels with selectors

- We can select resources with selectors
- They connect one type of resource to another
- Things that use the selector pattern:
  - Services selecting which pods to route to
  - Deployments selecting the pods they control
  - Scheduler selecting nodes for a pod

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

Networking

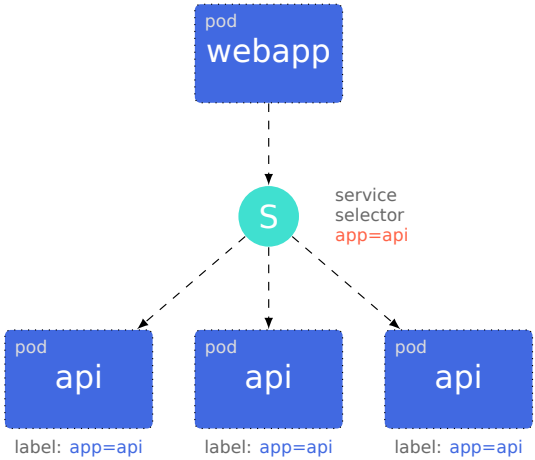
IPs for Pods

- In Kubernetes every pod has an IP
- This is called a **Pod IP** (or an **Overlay IP**)
- These IPs are as short-lived as our Pods are
- Most of the time this is provided with some overlay network technology (Flannel, Calico, there are a lot of plugins)
- Can be provided by the real switches and routers
- We will not discuss the details of setting this up, it just works on GKE
- This abstraction idea was originally missing from Borg
- Makes maintenance and administration easier (compared to Borg)

Networking

Services: how to have stable identity

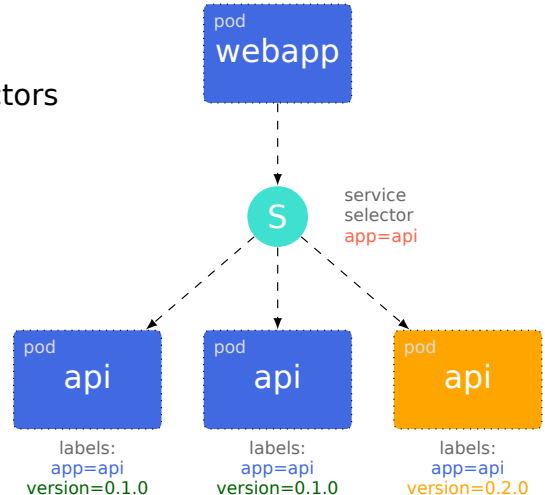
- Pod IPs are ephemeral
- We need service discovery
- This is **Service** in Kubernetes
- Implemented with labels: the matching pods provide the service
- IP allocated: single, stable address (**Cluster IP**)
- Round-robin load balanced
- There is a cluster wide DNS, so you don't need to hard code these Cluster IPs



Networking

Services example: canary deployments

- Services match pods using selectors
- This makes them flexible
- E.g. we can run 2 versions matched by 1 service for a canary deployment



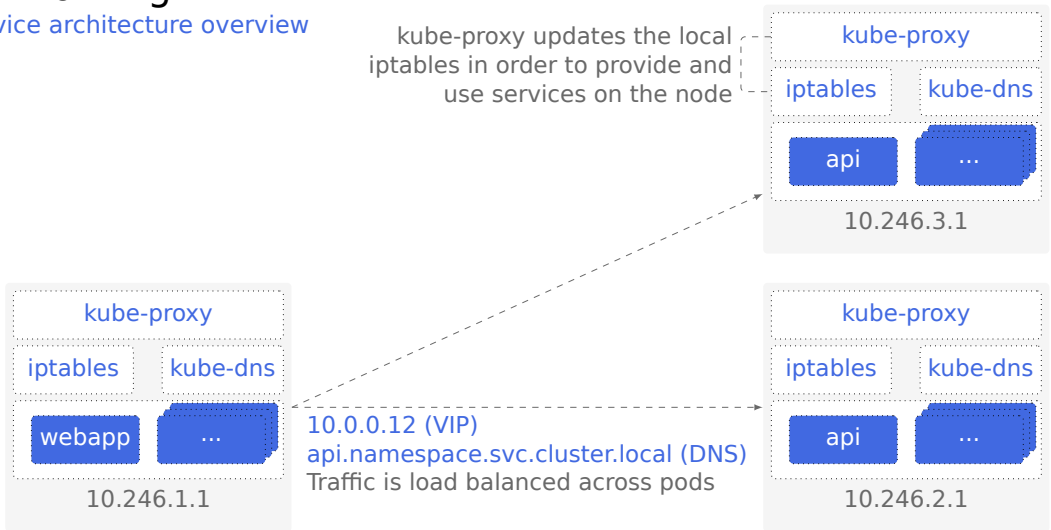
Networking

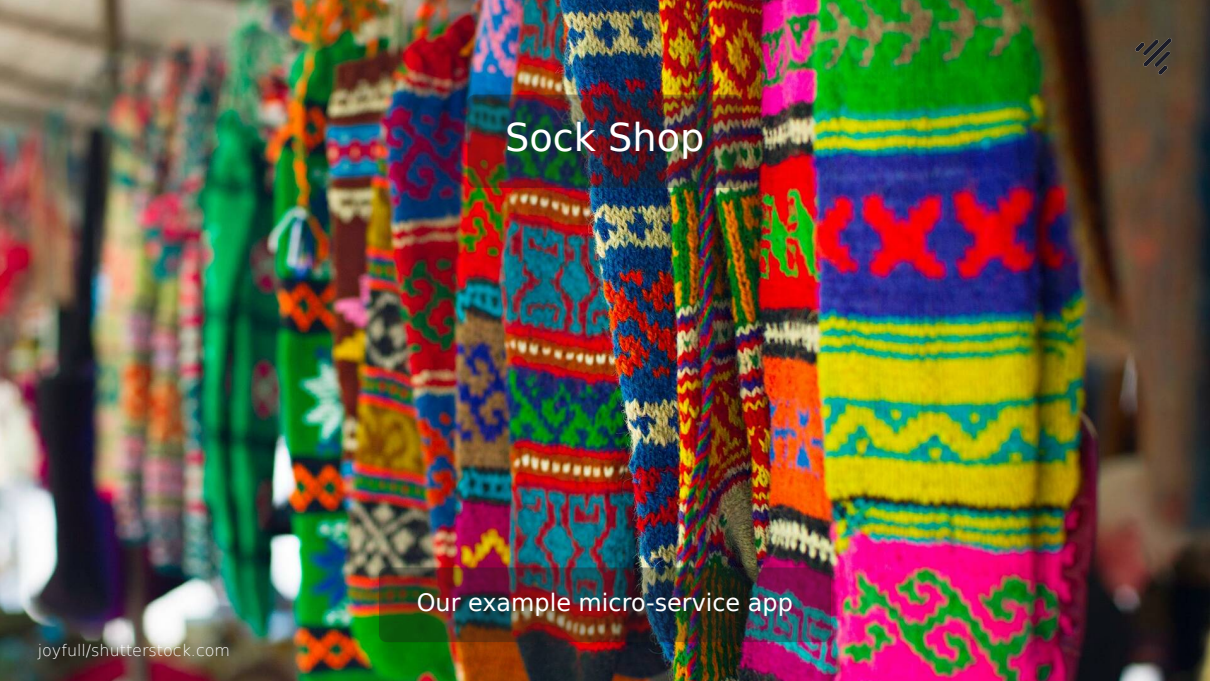
Service types

- **ClusterIP**: internal to the cluster only; chosen manually or automatically (default)
- **NodePort**: ClusterIP + a port that is open on *each* node (even on nodes without a relevant pod), allocated from the range of 30000-32767
- **LoadBalancer**: ClusterIP + NodePort + asks the “cloud provider” for a load balancer
  - GCE: Network/HTTPs LB (Linux hacks too)
  - AWS: ELB (Redirects to the NodePort)

Networking

Service architecture overview




Sock Shop
Our example micro-service app
joyfull/shutterstock.com

Sock Shop

Our CEO wants to expand and sell what?

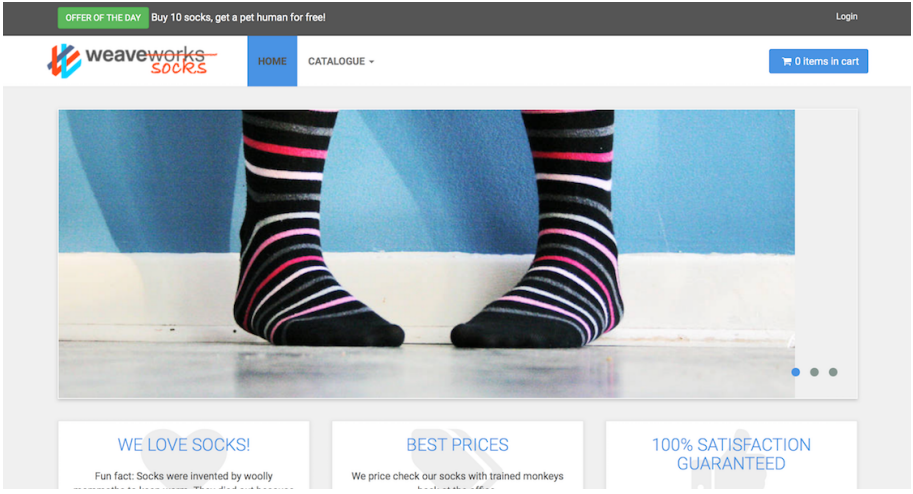
Our CEO has decided to acquire a startup that sells socks and expand the corporate strategy towards designer cotton footwear.

The board considers a move from insurance into socks both a risky and an unorthodox 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

The screenshot



Sock Shop

Prepare for the flood

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 recommended Google Kubernetes Engine as a production-grade, low-friction solution.

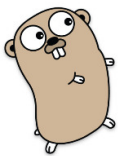
We can always move the cluster later, because Kubernetes can be installed on lots of infrastructure so we decide to get going with GKE.

Sock Shop

The developers have been busy

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

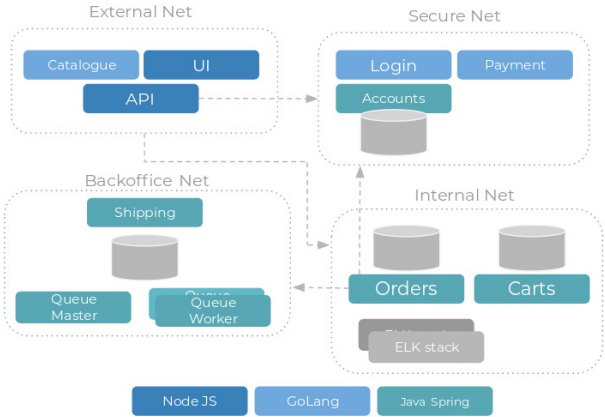
- openresty
- go x 3
- mysql
- java x 3
- mongo x 2
- rabbitmq
- zipkin
- node.js



Sock Shop

Architecture

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

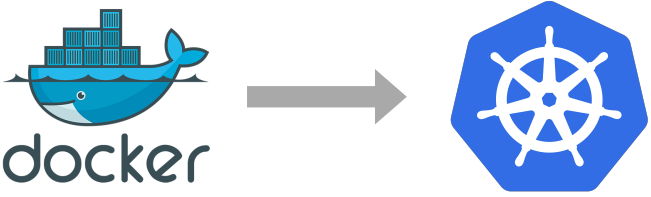


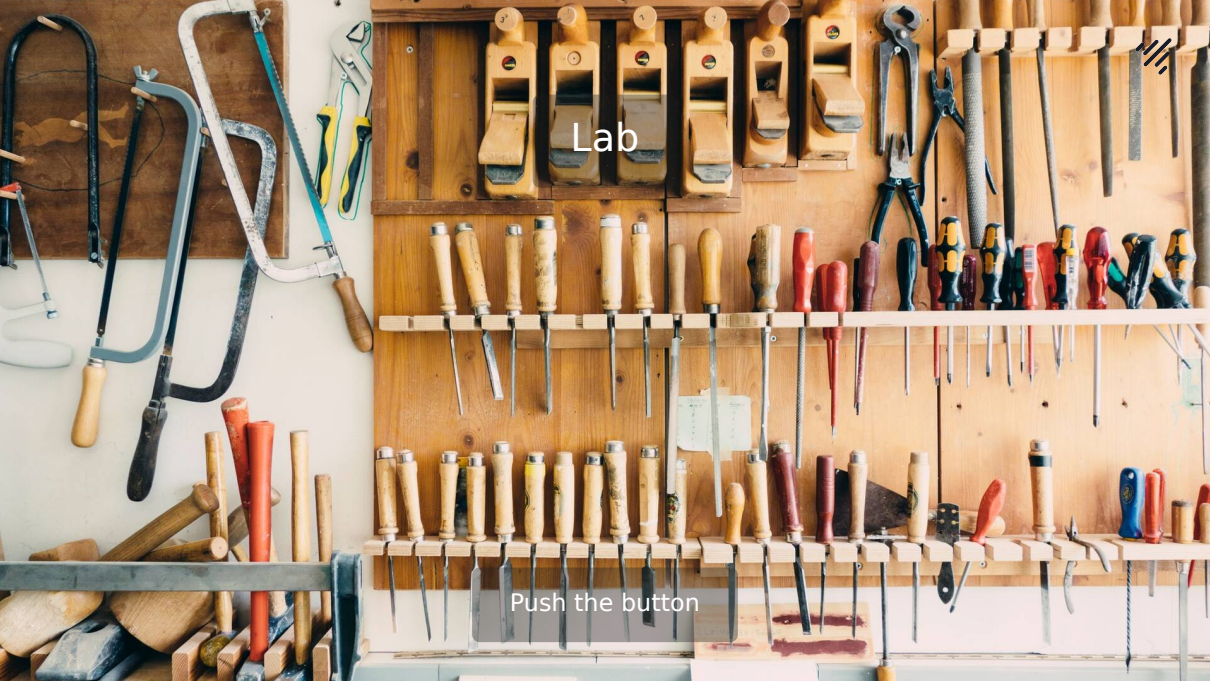
Sock Shop

Using Docker and K8S manifests

Thankfully they have used Docker for development and have created a Docker image for each service.

We’re going to make use of their images and Kubernetes manifests.




Lab
Push the button

Lab: Push the button

Deploy the whole sock shop stack to our cluster

- explore `beginner-labs/sock-shop`
- use `$ kubectl apply` to create all pods and services
- use `$ kubectl get {pods,services}` to confirm
- use `$ kubectl describe {pod,service}` to explore
- browse to the external-ip to see the website
- stop the front-end pod to prove that it was serving the traffic
- restart it, so we are in a healthy state, ready to continue
- try `$ kubectl get all`
- cleanup: delete everything, we will redo it in a better way

Namespaces

Namespaces, isolation of projects

Namespaces

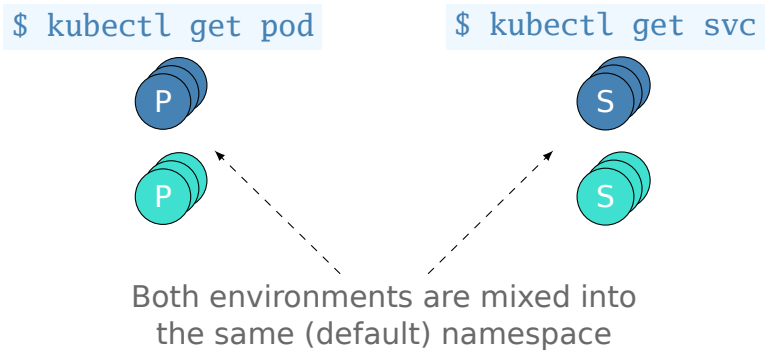
How to run virtual clusters on one actual cluster

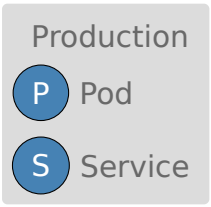
- Up until now, we have been using the `default` namespace
- What about `staging` and `prod` environments?
- Namespaces are the solution to having **everything** in one big list

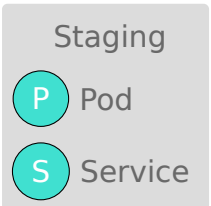
This is **not** to say, that namespaces magically provide **perfect isolation, security or safe multi-tenancy**. They are there for your convenience, as an **organizational tool**.

Namespaces

Without namespaces







Namespaces

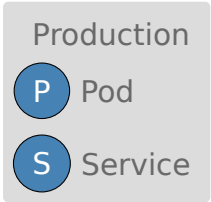
With namespaces

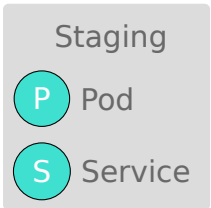
```
$ kubectl get pod
  -n production
```


```
$ kubectl get svc
  -n production
```


```
$ kubectl get pod
    -n staging
```


```
$ kubectl get svc
    -n staging
```



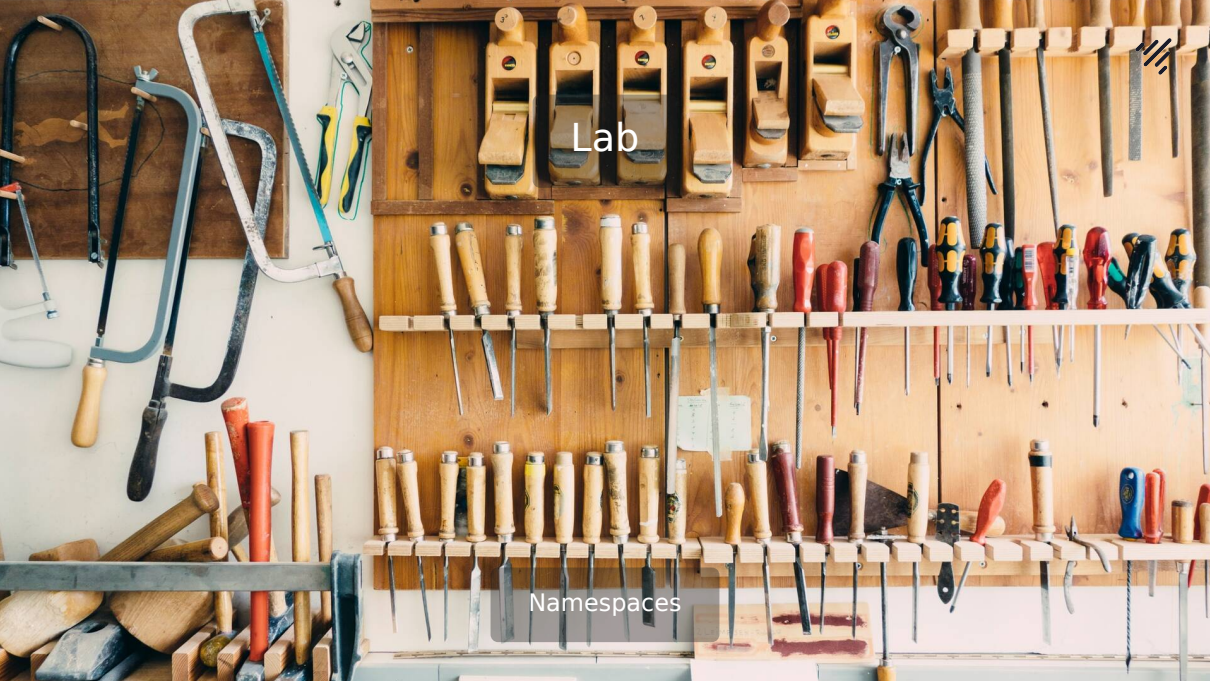


Namespaces

Notes

- Not every resource is namespaced
- E.g. namespaces are not namespaced :)
- Nodes are also not namespaced
- Try: `$ kubectl get node -n non-existent-namespace`
- The system is using the `kube-system` namespace, an empty K8S is not really empty
- Try: `$ kubectl get pod -n kube-system`

Lab



Namespaces

Lab: Namespaces

Demonstrate namespaces with a single pod

- list existing namespaces with `$ kubectl get namespaces`
- create a demo namespace with `$ kubectl create namespace`
- list existing namespaces with `$ kubectl get ns`
- run a demo pod in the demo namespace
- list pods in various namespaces
- Try `$ kubectl get pods --all-namespaces`

Best practice: put the `-n mynamespace` part of the command in the front, so you can easily edit the rest in your shell!

Workloads on Kubernetes

ReplicaSets, Deployments, canaries

Mopic/shutterstock.com

Workloads on Kubernetes

Section overview

- Running workloads (programs) safely and at scale
- Discussing desired vs actual state
- Handling upgrades
- Label trick: canaries

Using Services for loose-coupling.

ReplicaSets

Desired vs actual state

Pods can come and go; they’re fungible.

ReplicaSet: a controller, that dynamically ensures that the desired number of pods are running.

- Creates new pods if needed (based on a pod template)
- Stops existing pods if needed
- So it converges desired and actual state

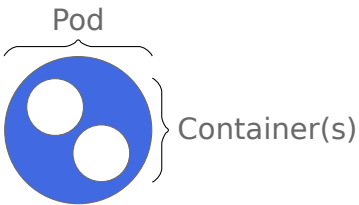
Deployments

Desired vs actual state

- ReplicaSets: can’t update the pod template in a controlled fashion
- Deployments provide declarative, server-side updates
- They oversee ReplicaSets (similar to how ReplicaSets oversee Pods)
- Deployment controller converges desired and actual state
- Supports rolling and recreate update
- Rolling update parameters are tunable

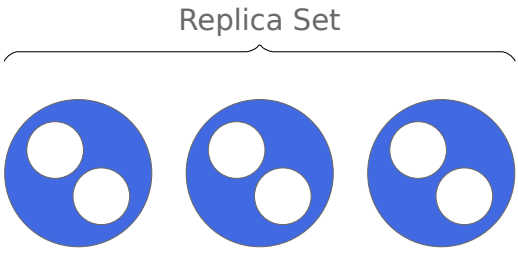
Workloads on Kubernetes

Pods and Containers



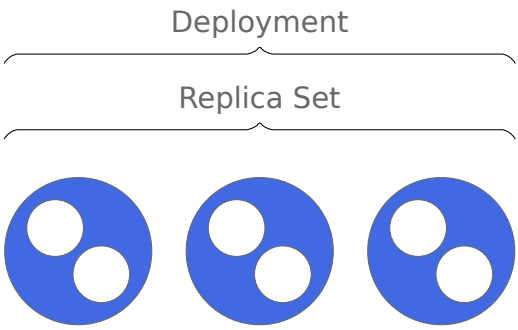
Workloads on Kubernetes

ReplicaSet



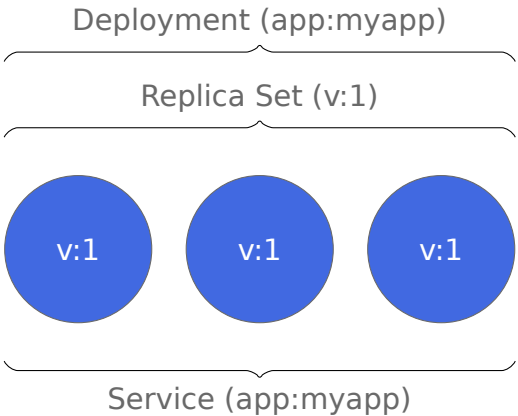
Workloads on Kubernetes

Deployment



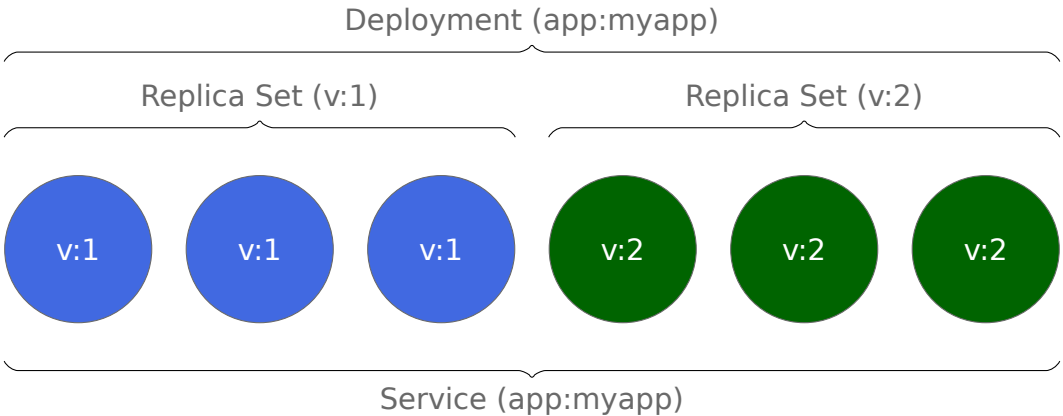
Workloads on Kubernetes

Recreate Update 1



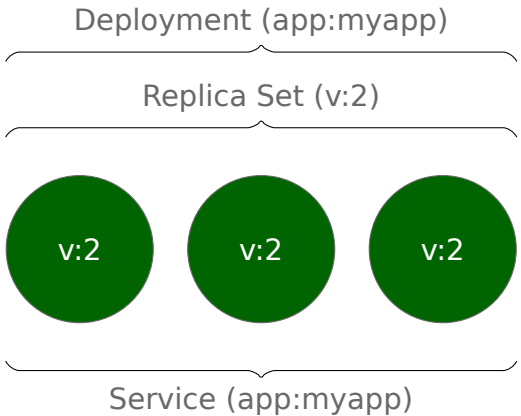
Workloads on Kubernetes

Recreate Update 2



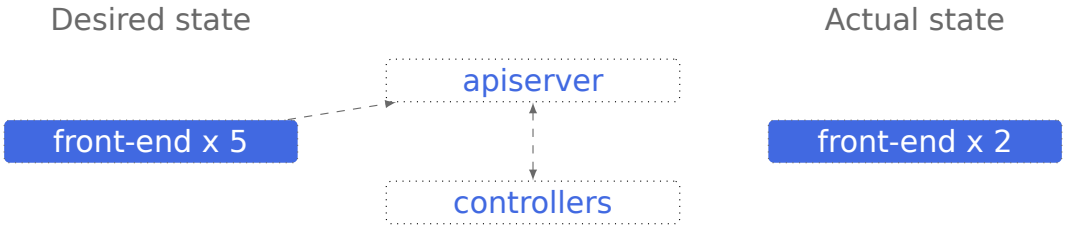
Workloads on Kubernetes

Recreate Update 3



Workloads on Kubernetes

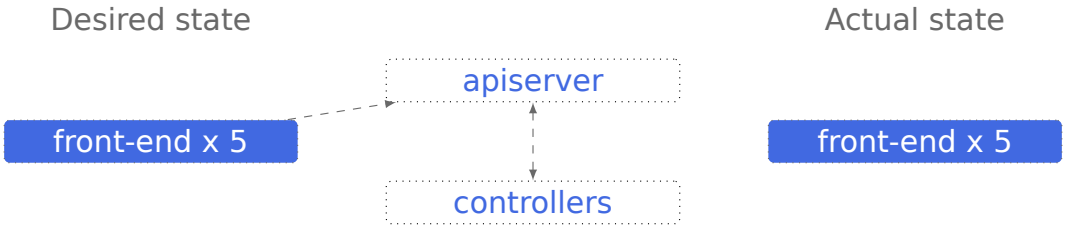
Scaling Deployments



- The Deployment Controller watches the Deployment resource (for desired data) and the ReplicaSet resource (for state data)
- Notices the 2 vs 5 difference, changes the ReplicaSet spec (increases the replica count)
- The ReplicaSet Controller adds the three new pods

Workloads on Kubernetes

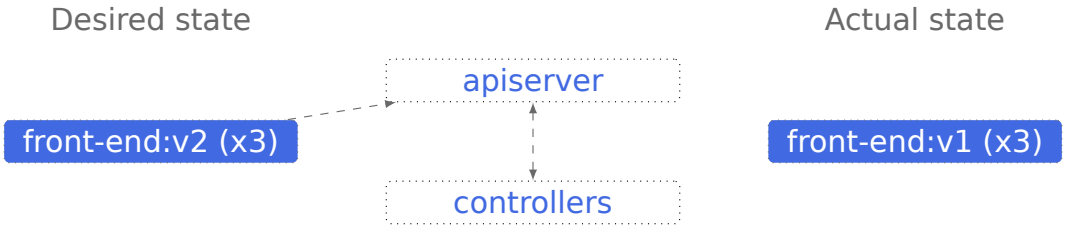
Scaling Deployments



- The Deployment Controller monitors the ReplicaSet status and updates the Deployment status
- Note: any service routing to the front-end pods will start using the new pods too

Workloads on Kubernetes

Scaling Deployments

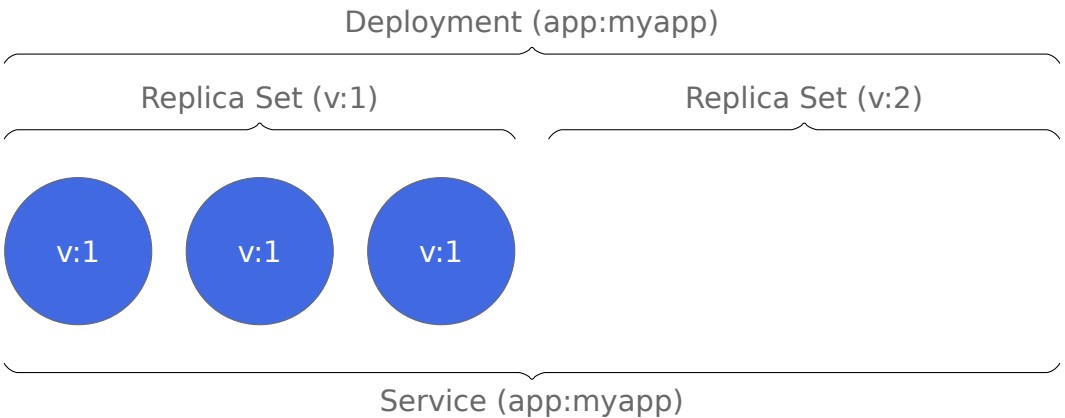


- We have 3 pods with v1 image, but the new spec wants v2
- Possible with Deployments (RollingUpdate strategy is the default)
- Discuss what would happen with Pods! (no effect)
- What would happen with ReplicaSets? (effect on new pods only)

Workloads on Kubernetes

Rolling Update 1

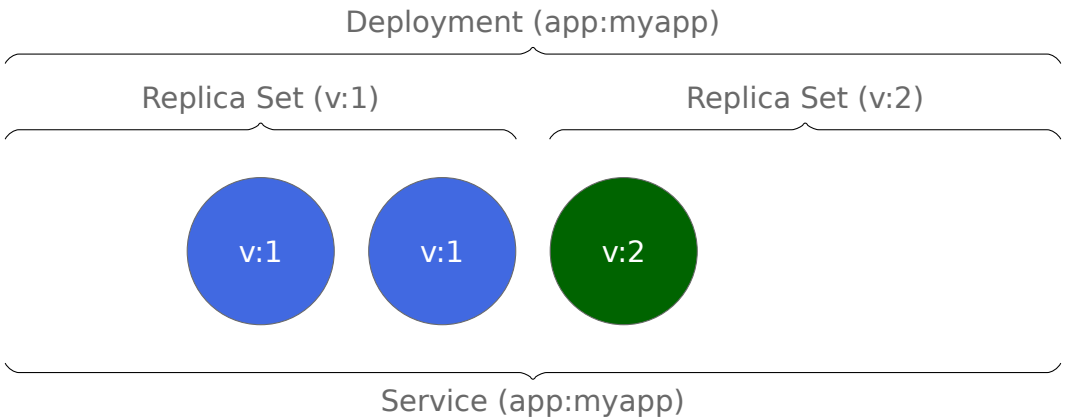
1st step: the Deployment Controller creates a new ReplicaSet with the v2 image in the pod template



Workloads on Kubernetes

Rolling Update 2

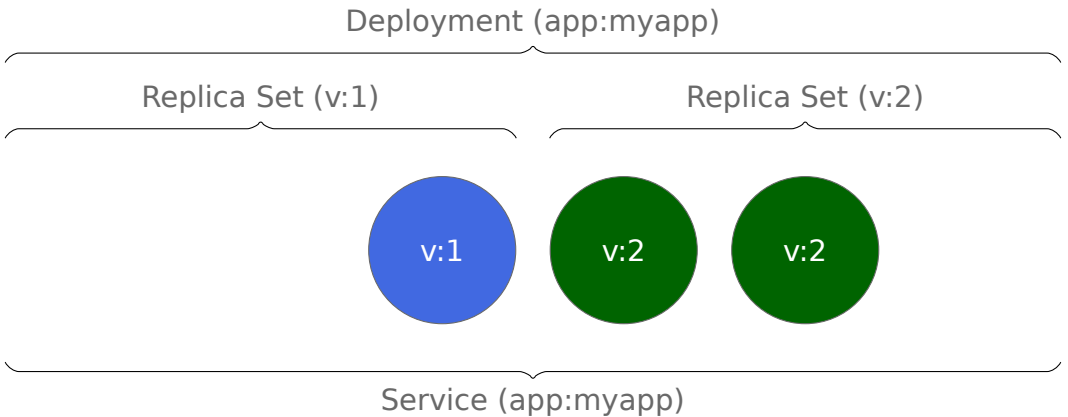
Then scale v1 down by 1, v2 up by 1. And repeat!

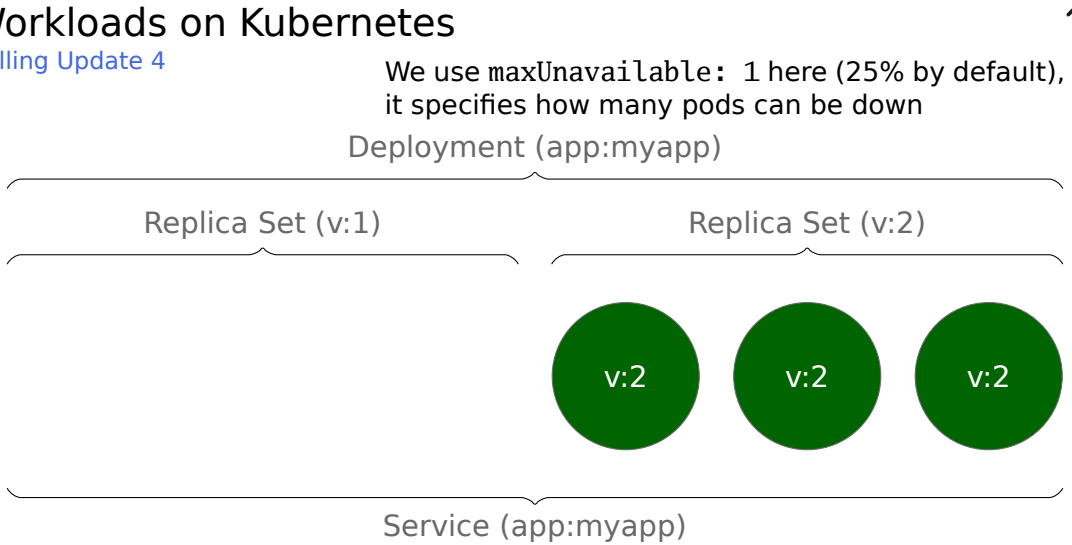


Workloads on Kubernetes

Rolling Update 3

We use `maxSurge: 0` here (25% by default),
it specifies the budget for extra pods

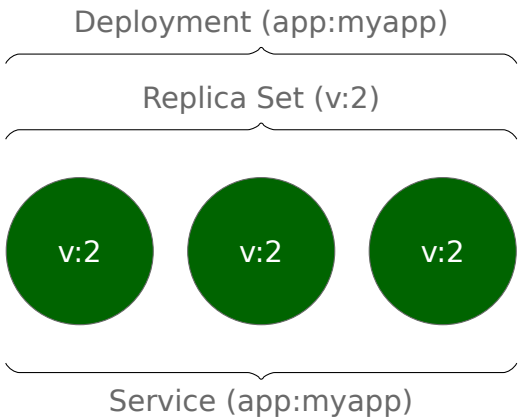



We use maxUnavailable: 1 here (25% by default),
it specifies how many pods can be down
Deployment (app:myapp)
Replica Set (v:1)
Replica Set (v:2)
v:2
v:2
v:2
Service (app:myapp)

Workloads on Kubernetes

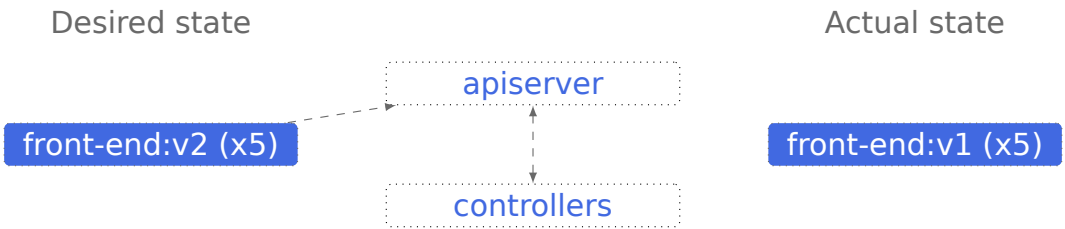
Rolling Update 5

- Finally the ReplicaSet v1 can be deleted. (This depends on `revisionHistoryLimit`, defaults to 10.)
- Note: the end-user didn't notice anything, the Service was there the whole time for whole of `app:myapp`.



Workloads on Kubernetes

Exercise



- `maxUnavailable=2`, `maxSurge=1`, `strategy.type=RollingUpdate`
- Discuss what will happen in this case!

Deployments, ReplicaSets and state

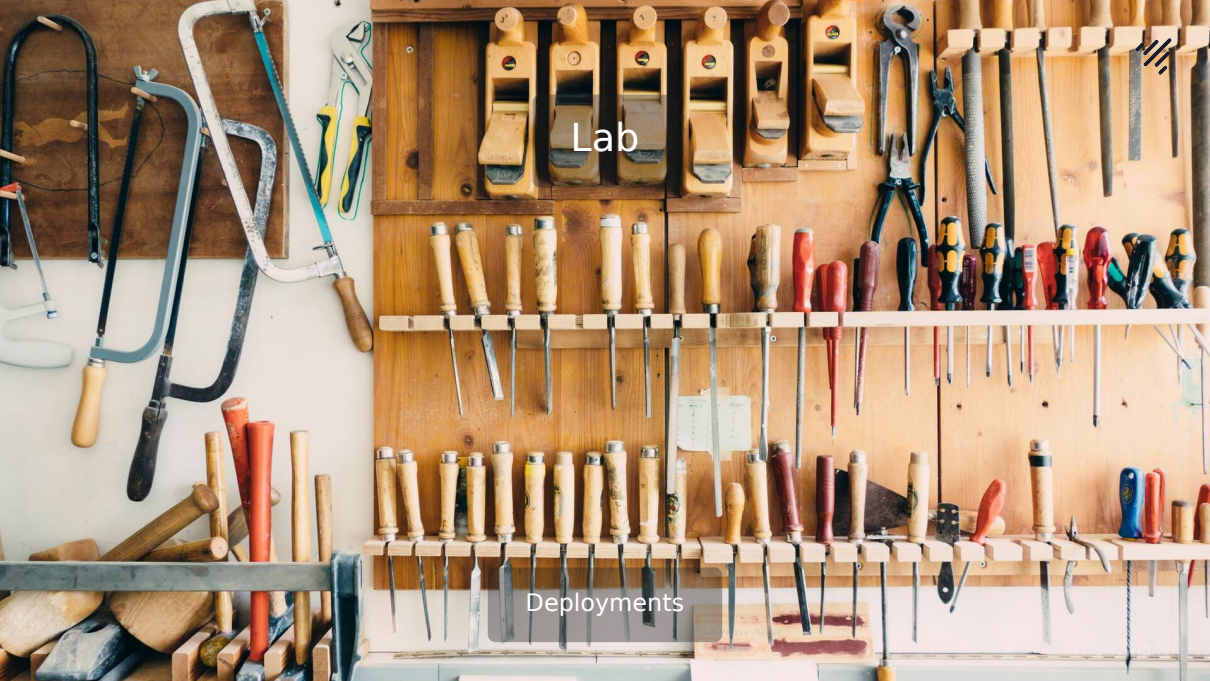
Assumptions

All of this:

- Assumes no state on the pods
- So no databases, master-slave or anything like that
- But you can use CloudSQL, Spanner, BigTable, etc.
- Or great for pods that are stateless (e.g. scalable image resizing)

Alternative: StatefulSet

- We will learn about it later
- Handles stateful volumes on pods
- Assigns same volume to same pod always, even after scaling
- Stable names for pods (xxx-0, xxx-1, xxx-2, ...)


Lab
Deployments

Lab: Deployments

Deploy the sock shop into namespaces

- create the namespace with `$ kubectl apply -f`
- create the services with `$ kubectl apply -f`
- create the deployments with `$ kubectl apply -f`
- view the site
- use `$ kubectl apply` to update the image
- check the site
- use `$ kubectl apply` to revert the change
- do not cleanup, we will use this for the rest of the day!

Best practice: store stuff in Git and use `kubectl apply` always, never `kubectl expose` or similar.

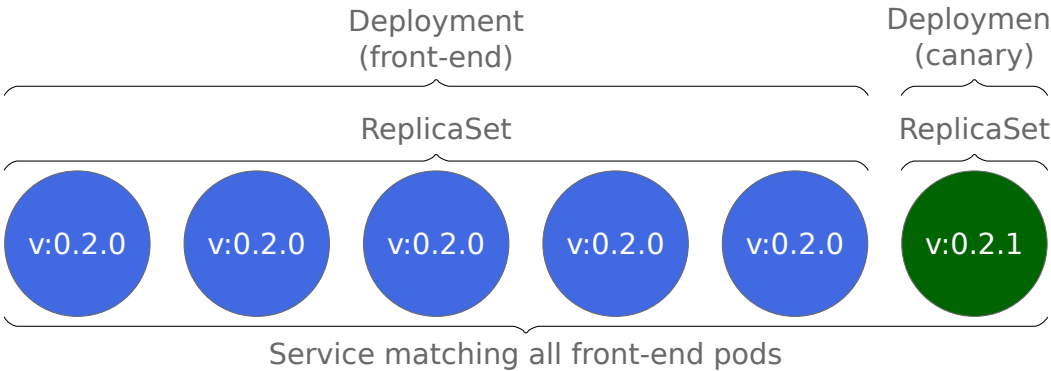
Canary deployments

Testing out new backend features

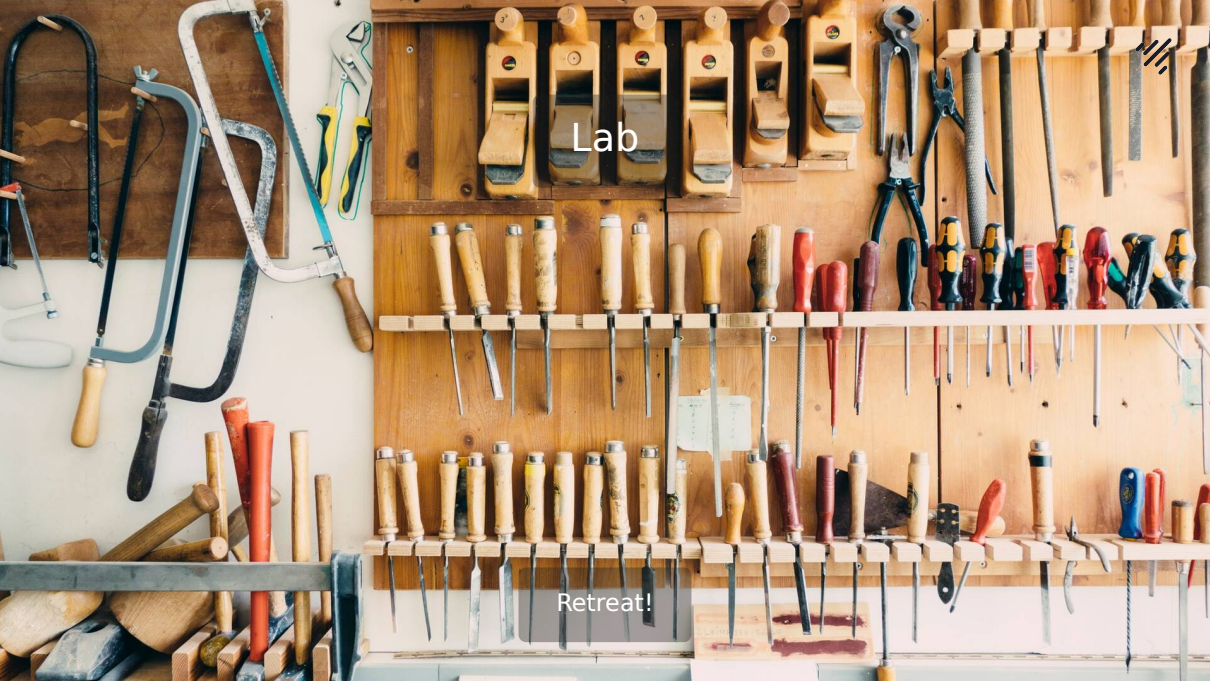
- A common deployment pattern on Kubernetes
- Also referred to as an “incremental rollout”
- New release of a microservice is deployed beside the existing version
- The new release can receive some live production traffic
- Opportunity to test functionality before full roll-out
- Easy to roll back and cancel the roll-out
- Fairly easy to implement with *labels*
  - Create new deployment with changed image
  - Use a new version for the pods in the template
  - But continue to use the same name in the template
  - If your service routes by the name label, you have your canary

Canary deployments

Implementation with labels: an overview



This setup would result in a 16% (1/6) canary, in the lab, we will set up a 50%, so it's easier to see its effects.


Lab
Retreat!

Lab: Retreat!

Create a canary deployment and practice rollback

- create an alternative deployment manifest as a canary
- push the deployment
- check the site
- remove the canary

Readiness & Liveness Probes

Tell Kubernetes how to monitor the pods

Readiness & Liveness Probes

Section overview

We discuss:

- Failure handing in Kubernetes
- Type of tests to detect failures: exec, HTTP, TCP
- Automated container restarts
- Service routing in face of failures and slow starts
- Deployment upgrades in face of slow start

Liveness probes

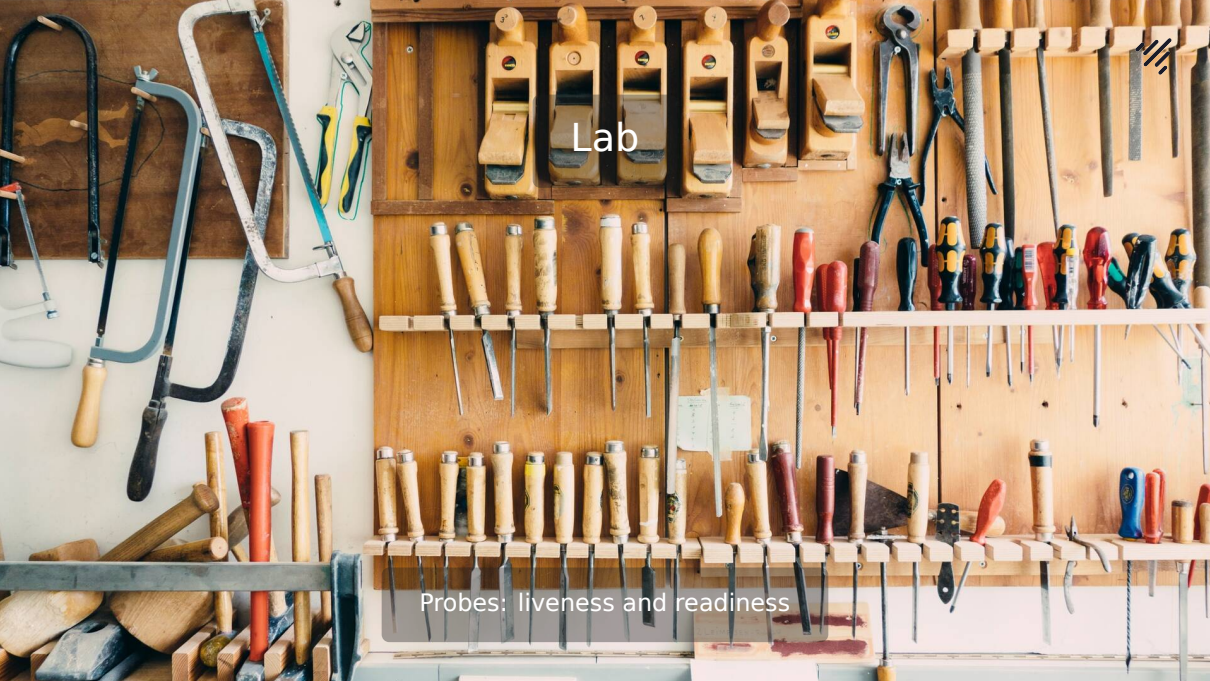
Define rules about when to restart containers

- kubelet tests that a container is running correctly
- If not, then the container will be restarted
- Useful in cases where the application get stuck
- One of the following types:
  - exec: command run inside the container, fail if return != 0
  - HTTP: request sent, fail if HTTP code outside [200;400)
  - TCP: socket connection, fail if cannot be established
- Important! We restart the failing container, not the whole pod: https://github.com/kubernetes/kubernetes/issues/40908

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 types as with liveness probes
- This is just info for service endpoints
- No auto-repair, no restart or anything like that will happen
- Also affects rollouts


Lab
Probes: liveness and readiness

Lab: probes, liveness and readiness

- set up readiness and liveness probes
- simulate readiness failure
- simulate liveness failure
- use the `wget` and `kubectl get` commands to see what happens

Recap

Configuration objects and settings we discussed

- **Pods**: containers, limits, requests
- **Deployments**: managing pods with ReplicaSets (rolling updates)
- **Scaling**: increasing and decreasing replicas
- **Namespaces**: organization (e.g. dev, test, staging)
- **Labels and Selectors**: link Kubernetes objects together
- **Services**: stable names and IPs with labels
- **Service types**: ClusterIP, NodePort, LoadBalancer
- **Label tricks**: canaries
- **Probes**: check if your application is healthy and ready to serve

Storage

Handling state in Kubernetes

Mike Mols/shutterstock.com

Storage

Handling state in Kubernetes

- To handle stateful pods (e.g. databases) we need storage
- This is orthogonal to Kubernetes
  - In cloud providers this is usually provided as a “disk”
  - Called GCE-PD on Google; EBS on Amazon
  - On premise: Ceph or similar
- Kubernetes is **only** responsible for the wiring
- Easiest: `pod.spec.containers.volumeMounts`, `pod.spec.volumes`
- Abstract: `PersistentVolume`, `PersistentVolumeClaim`
- Automated: `StorageClass`

Storage

A simple example for the direct definition

```
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - { mountPath: /website, name: nfs }
    - { mountPath: /mongo, name: mongo }
  volumes:
  - name: nfs
    nfs: { path: /homepage, server: my.nfs.server }
  - name: mongo
    gcePersistentDisk:
      pdName: mongo-disk
```

Storage: PVs, PVCs, `StorageClasses`

An abstraction for storage

- In the previous example we had to create the `mongo-disk` in GCE
- Instead we can automate this with a `PersistentVolumeClaim` (PVC)
- If the PVC specifies a SC (or there is a default one), then that SC does the auto-provisoning of the PV
- On GKE we have a default SC, that uses GCE Persistent Disks
- All this automation is the basis for `StatefulSets`

Note:

- PVs can also be defined by the cluster owner by hand (obsolete)
- Then you can use PVCs even without `StorageClasses`
- A Kubernetes controller in the master will look for a matching PersistentVolume (PV) and bind the PVC to it forever

Storage

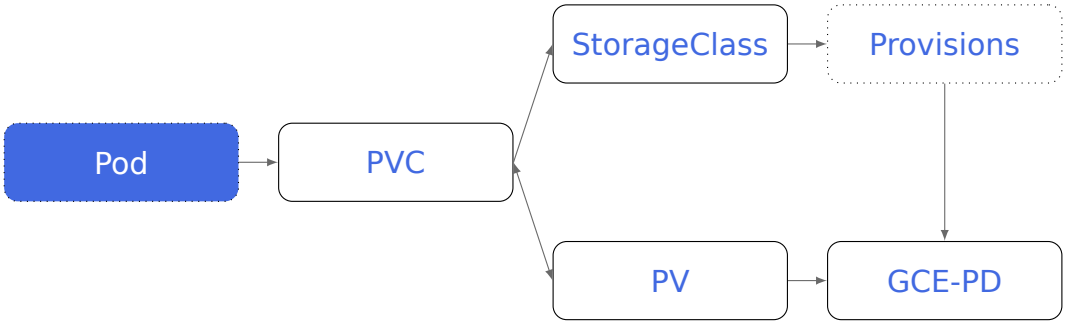
A simple example for a PVC

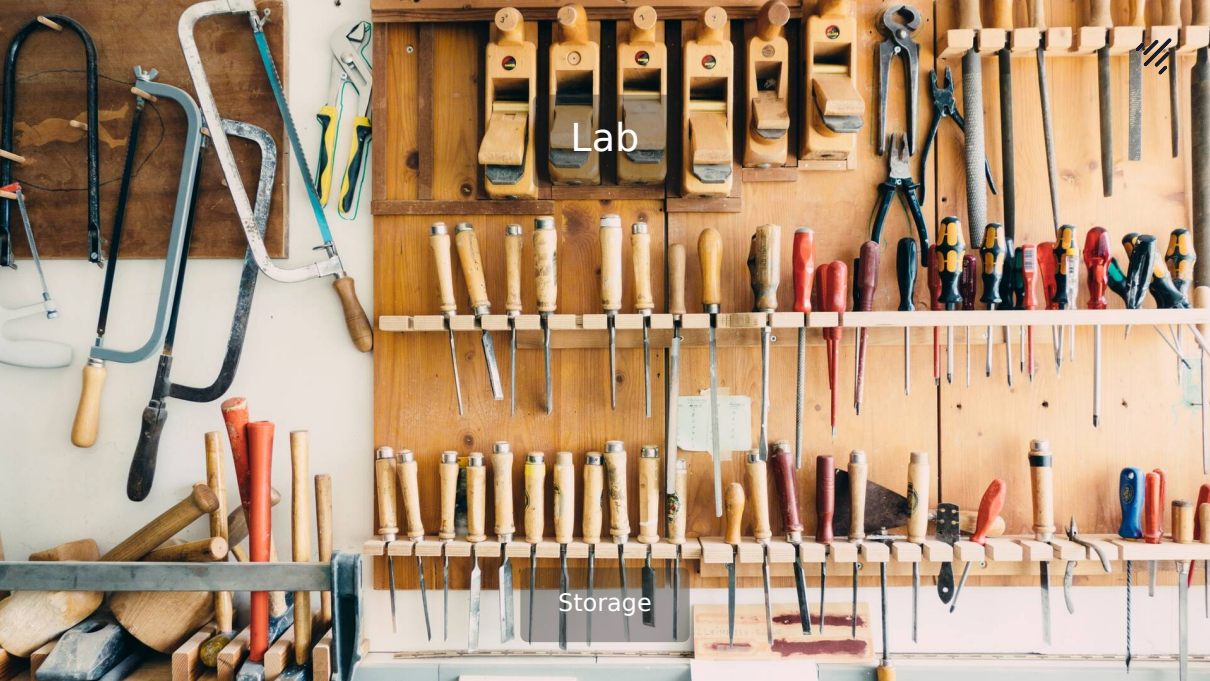
```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata: { name: mongo-pvc }
spec:
  accessModes: [ ReadWriteOnce ]
  resources: { requests: { storage: 12Gi } }
```

```yaml
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - { mountPath: /website, name: nfs }
    - { mountPath: /mongo, name: mongo }
  volumes:
  - name: nfs
    nfs: { path: /homepage, server: my.nfs.server }
  - name: mongo
    persistentVolumeClaim: # instead of gcePersistentDisk
      claimName: mongo-pvc # instead of pdName
```

Storage: PVs, PVCs, StorageClasses

Architecture overview




Lab
Storage

Lab: Storage

Delete the carts-db pod to see that the data is really lost

- create some state
- delete the carts-db pod
- watch the pod re-deploy
- refresh the browser and notice how the data is gone
- now use a GCE Persistent Disk to create a setup,
  where the data is not lost
- replace the GCE PD with a PVC, to demonstrate dynamic provisioning

Storage

Alternatives

- Cloud SQL and similar hosted solutions
- Spanner for multi-region relational
- Bigtable and friends for NoSQL
- Cloud Storage and S3 buckets for big files
- Yes, they are expensive, but you save on maintenance

ConfigMaps and Secrets

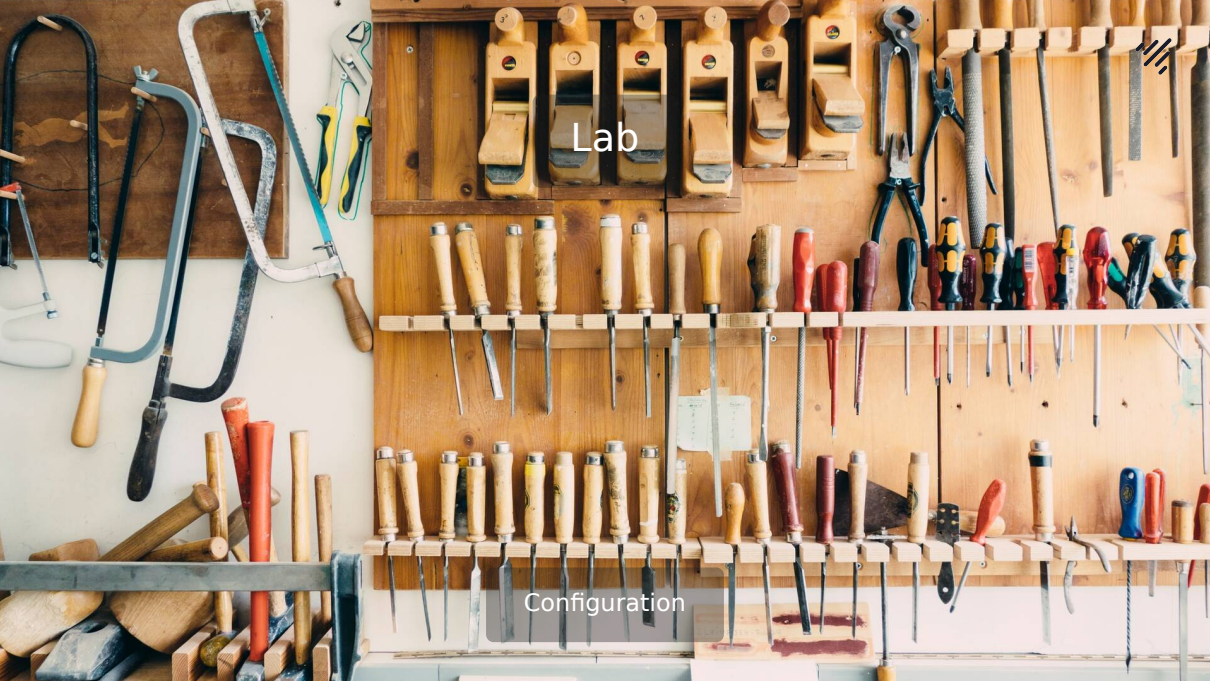
Distributing configuration and credentials to the pods

Spainter vfx/shutterstock.com

ConfigMaps and Secrets

Overview

- Easy enough to embed secrets (e.g. passwords) and other configurations into a container image – but please don’t
- Use K8S ConfigMaps and Secrets
- Consumable in a variety of ways from PodSpec:
  - Environment variables
  - Command line arguments
  - Volumes
- We will talk about them more on the intermediate course!


Lab
Configuration

Lab: Configuration

Configure our MySQL pods with ConfigMaps and Secrets

- Create the MySQL config as a ConfigMap
- Record the MySQL password in a Secret
- Update the catalogue-db deployment to use these

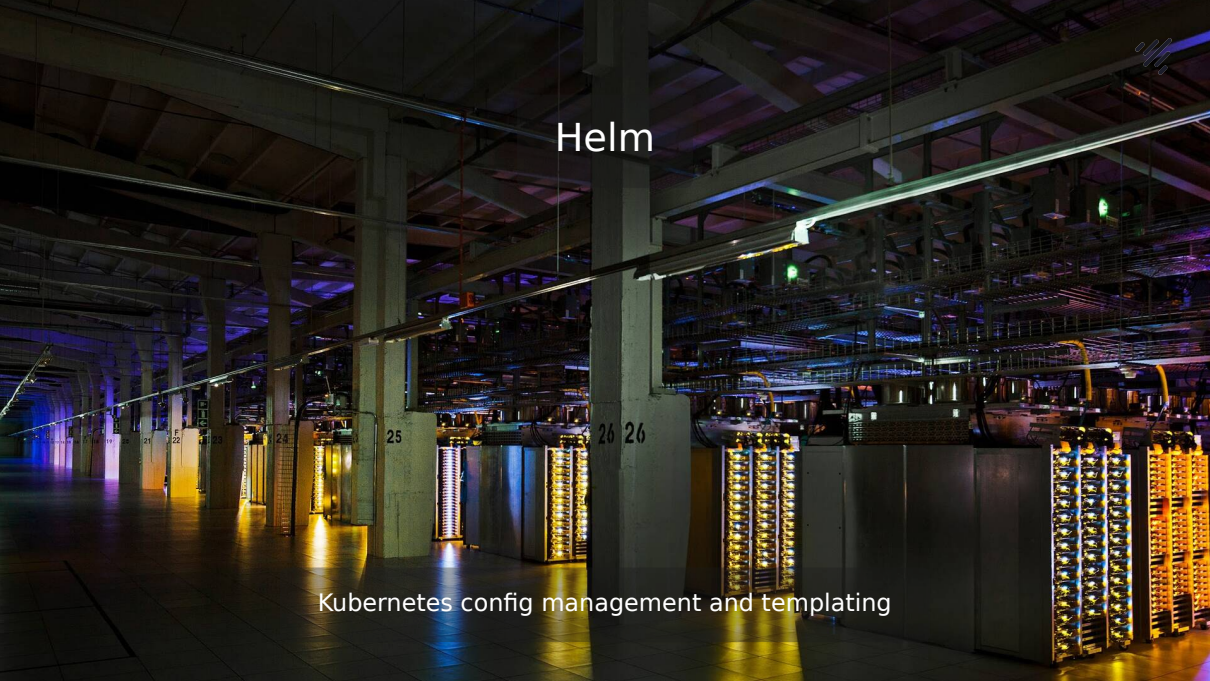
Logging and Monitoring

Keep tabs on your cluster and services

Logging and Monitoring

- Check out the GKE Dashboard
- `kubectl logs`
- `kubectl top`

These are demonstrated in the “Logging” lab!



Helm

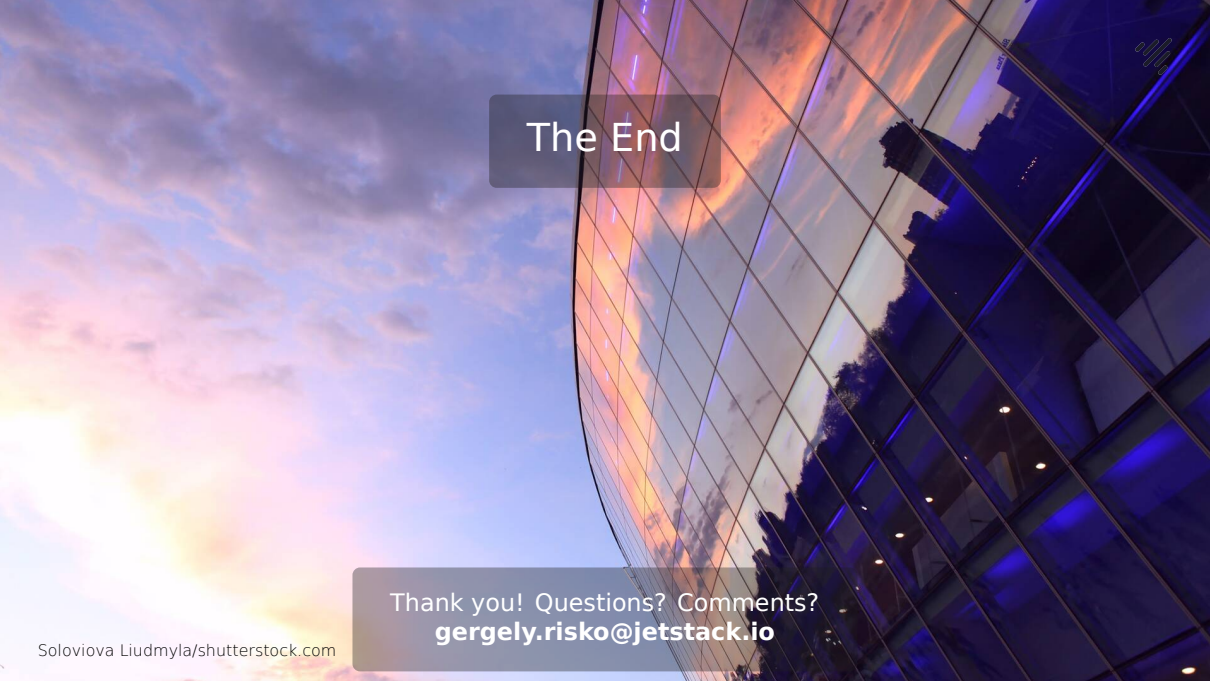
Kubernetes config management and templating

Helm: templating in the YAML

Make your deployments customizable to work in different environments

- A single application consists of many resources (YAML files)
- Changes (e.g. namespace name) are difficult to manage
- Git branches (for different environments) become messy
- Helm's `values.yaml`: specify parameter key-value pairs
- And use these parameters in YAML templates
- Helm just compiles this to one output that you can `kubectl apply`
- Helm also provides pre-made packages for popular stuff
- You can use these to get an initial config
- We will do this on the intermediate course (with MongoDB)

For the templating part, please see the "Helm templating" lab



The End

Thank you! Questions? Comments?
gergely.risko@jetstack.io

Soloviova Liudmyla/shutterstock.com