

Containers, Cloud Native Computing, DevOps

Oxford University
Software Engineering
Programme
December 2019



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons
Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

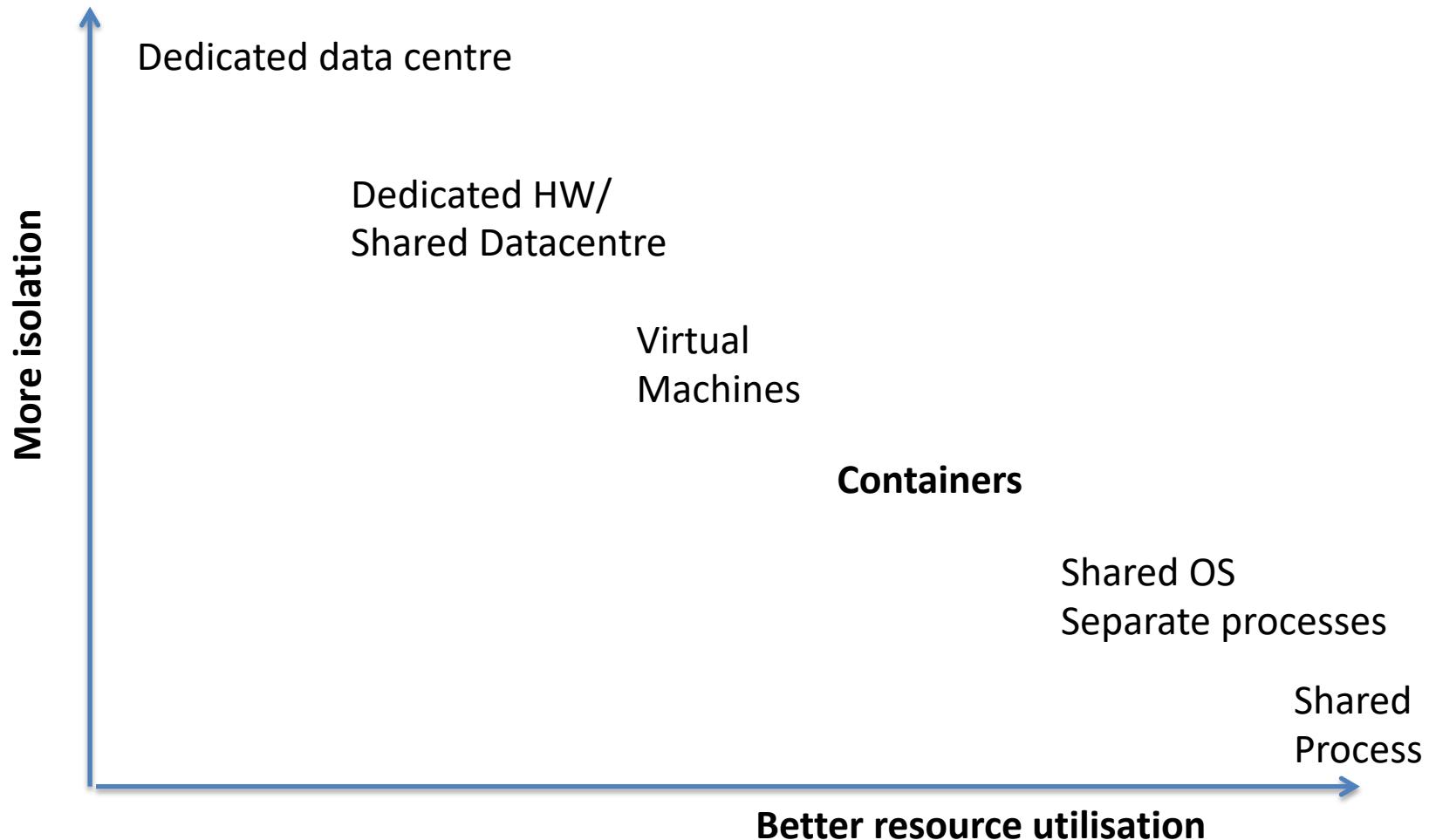
Contents

- Containers
- History and Approach
- Docker
- Docker ecosystem
- PaaS in a container model
- Futures



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Sharing of resources vs Isolation



Lightweight Virtualization history

- **zSystems Virtual Servers from late 1990s**
 - (the mainframe really did do everything first)
- **Solaris Containers**
- **AIX Workload Partitions**
- **FreeBSD Jail**
- ...



What is a Container?

- A **lightweight virtual server**
 - Running within an Operating System
 - Providing various levels of isolation and control
 - E.g. Disk isolation and control
 - Network isolation
 - CPU and memory controls



Containers at Google

- Every GMail session is a container
 - Try doing an export and then searching your email ☺
- “Everything runs in a container”
- 2 billion containers launched a week
- Borg
 - Any Google developer can instantiate their code in 10,000 instances any time they want
 - Takes about 5 minutes to start that many
 - Never exactly 10,000 because of failures



Cloud Native Computing Foundation

- A new definition of “Cloud Native”
 - Container Packaged
 - Dynamically Managed
 - Micro-Service oriented



Docker on top of Containers

- Docker adds several things to LXC and containerization:
 - Copy on write filesystem
 - Layered images and the ability to extend machines easily
 - Simple textual config file
 - Portable deployment across machines
 - Creating an ecosystem of images
 - Application centric
 - Each VM is a process (roughly speaking)
 - Plus others (auto-build, etc)

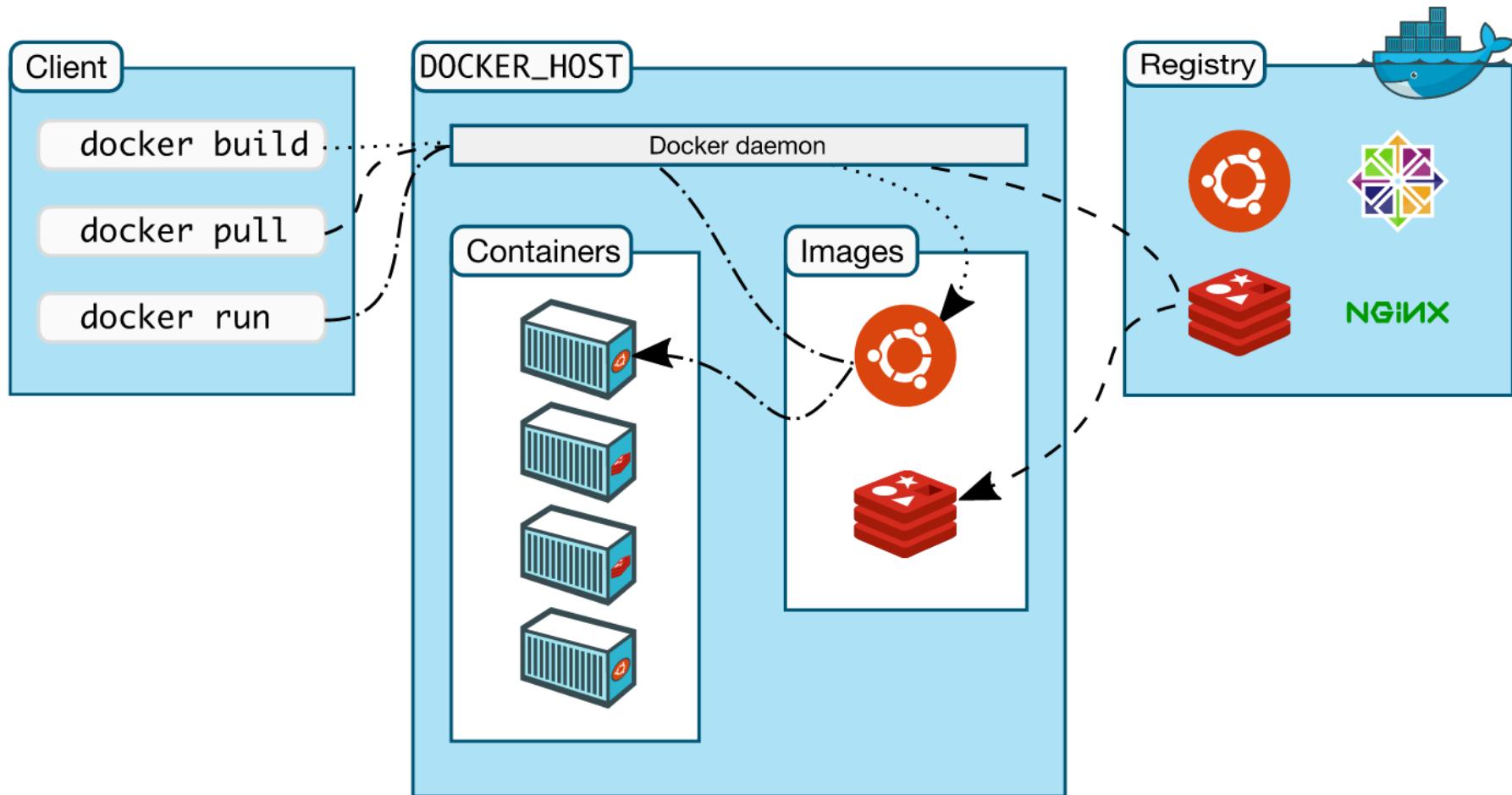


Why Docker?

- The *ecosystem* has created a *network effect*
- Metcalfe's Law states
 - the value of a telecommunications network is proportional to the square of the number of connected users of the system
- There is surely a corollary for ecosystems



How does Docker work?



Dockerfile

```
FROM alpine
RUN apk --update add python py-pip && \
    pip install --upgrade pip && \
    mkdir -p /home/root/python && \
    pip install kafka && \
    pip install httpplib2

COPY tflrepub.py /home/root/python/

WORKDIR /home/root/python/

ENTRYPOINT python tflrepub.py
```



Some simple Docker commands

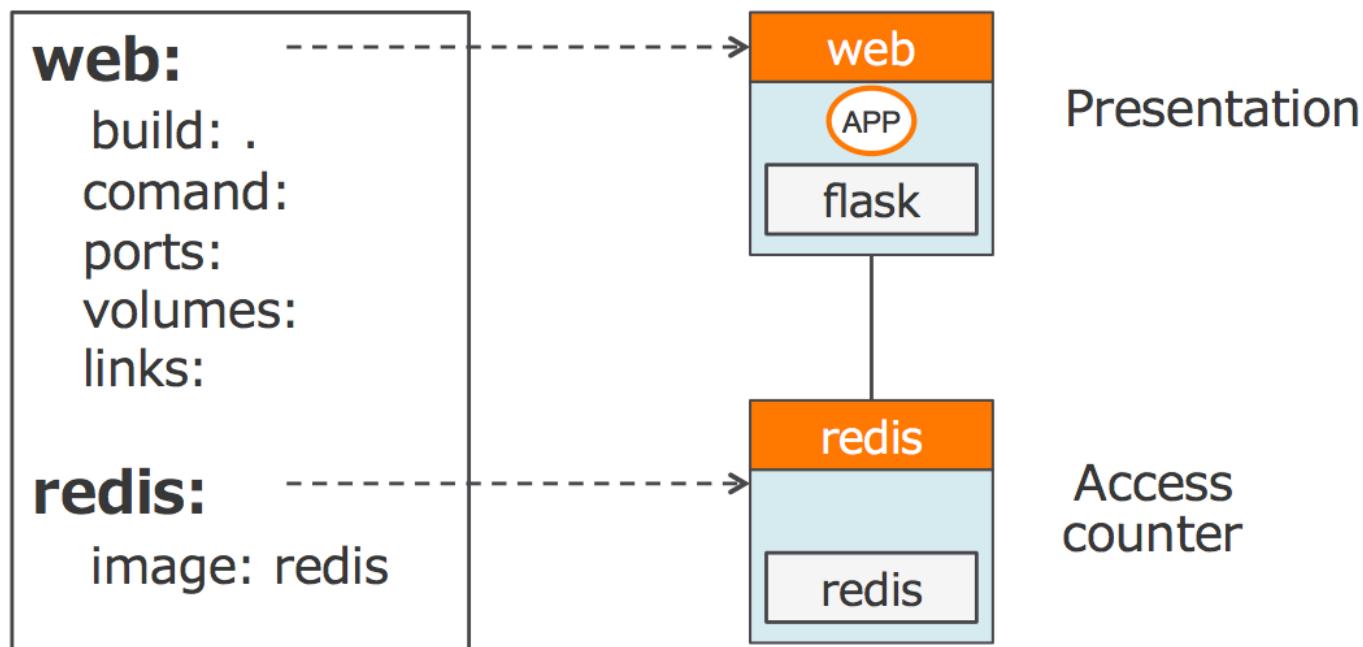
- `apt-get install docker.io`
- `docker pull ubuntu`
- `docker run -t -i ubuntu /bin/bash`
- `docker ps`
- `docker commit funky_freo image`
- `docker push image`



Docker Compose

- How to create a set of containers that work together

docker-compose.yml



docker-compose.yml

```
version: '2'

services:
  zookeeper:
    build:
      context: .
      dockerfile: Dockerfile-zookeeper
    ports:
      - "2181:2181"
  kafka:
    build:
      context: .
      dockerfile: Dockerfile-kafka
    ports:
      - "9092:9092"
  networks:
    default:
      aliases:
        - kafka.freo.me
  depends_on:
    - zookeeper
```



Cloud Orchestration

- What does an Operating System do?
 - Manages processes
 - Co-ordinates the processes access to resources
 - CPUs
 - Memory
 - Disk
 - Devices
 - Fairness and priority between processes



Datacenter Operating System

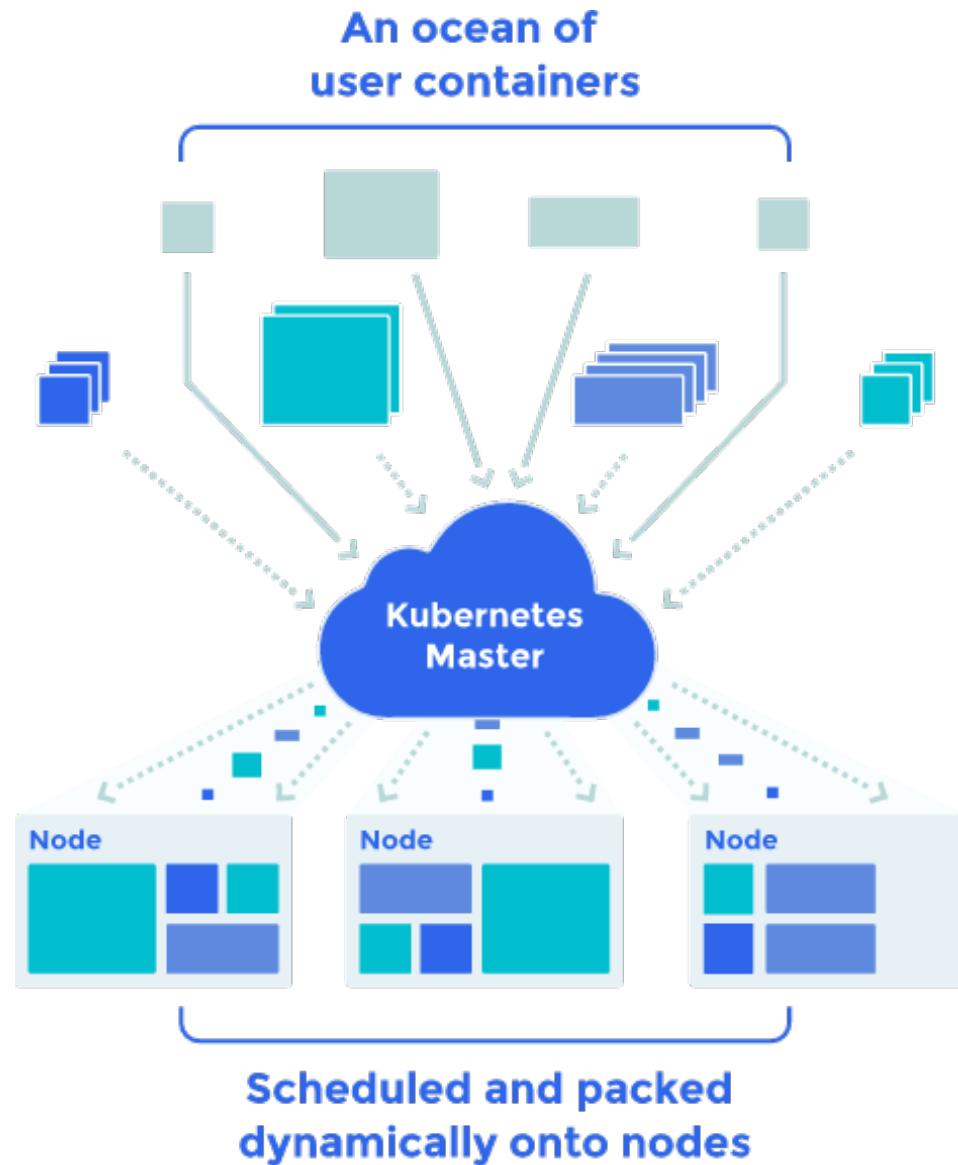
aka Container Orchestration

- Manages the placement of containers
 - Access to resources
 - Configuration and networking
 - Moves containers
 - Load balances across containers
- Effectively creating a single OS across a cloud
 - Containers vs Processes

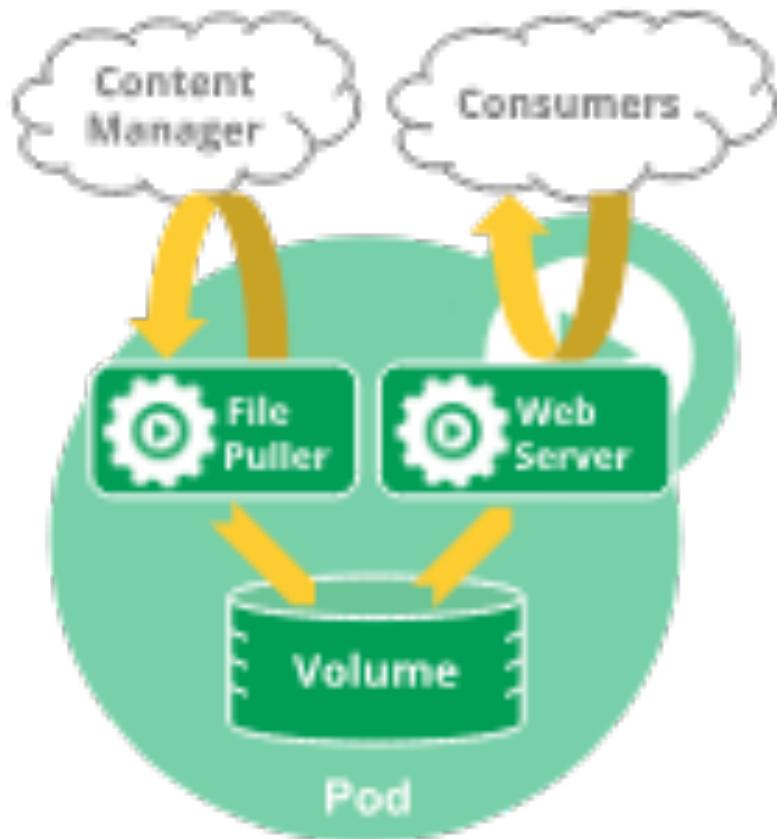


Kubernetes

- Open Source cluster management of containers
- From Google, but separate from the Borg project



Pods



A Pod encapsulates an application container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run.

A Pod represents a unit of deployment: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources.



Services

- An abstract exposure of pods
- Pods die and are recreated, replicated

“A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them”



Volumes

- A persistent virtual disk that belongs to a Pod
- Shares data between containers
- Lives longer than a container, but no longer than the pod



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

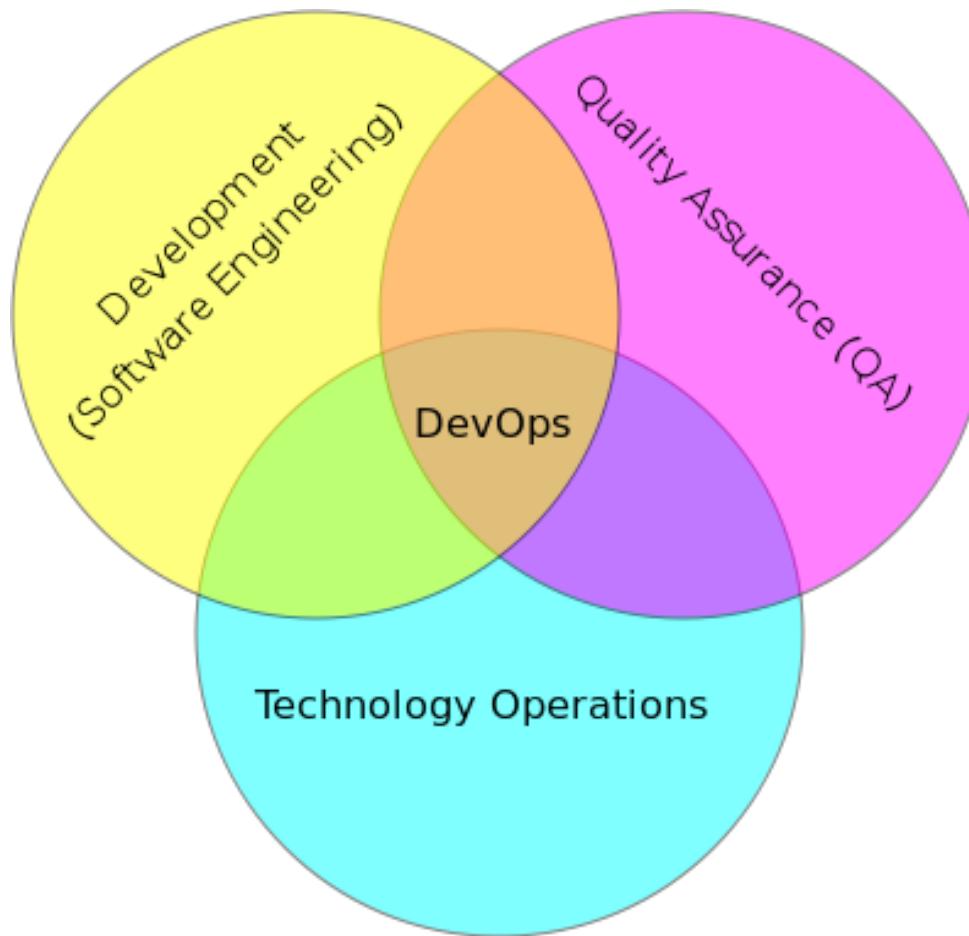
Namespaces

- A virtual cluster
- Names must be unique inside namespaces, can be the same across different namespaces



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

DevOps



DevOps

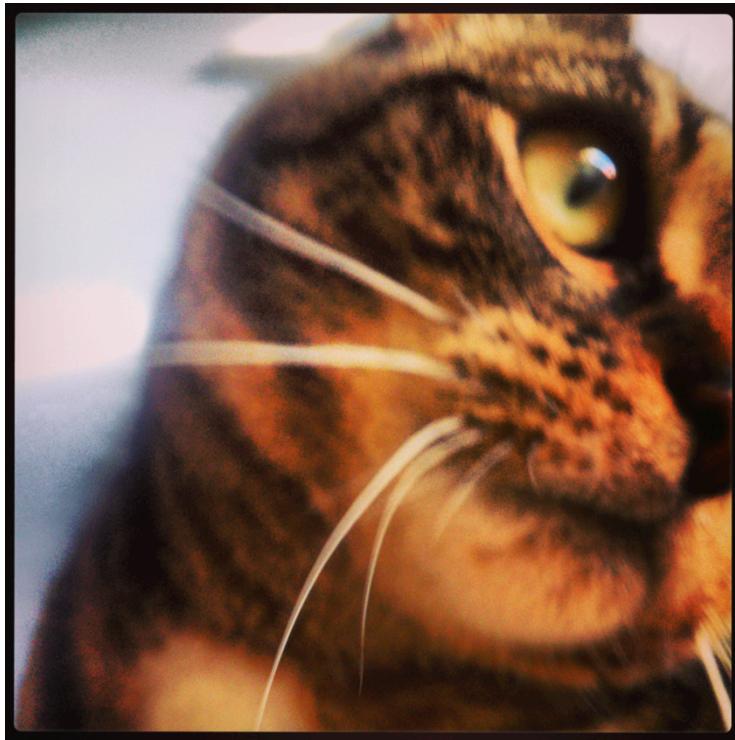
- DevOps is the codification of the interface between Development and Operations
 - Agile
 - Repeatable
 - Collaborative
 - Versioned
 - Automated



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Kittens vs Cattle

(An unpleasant but effective analogy)



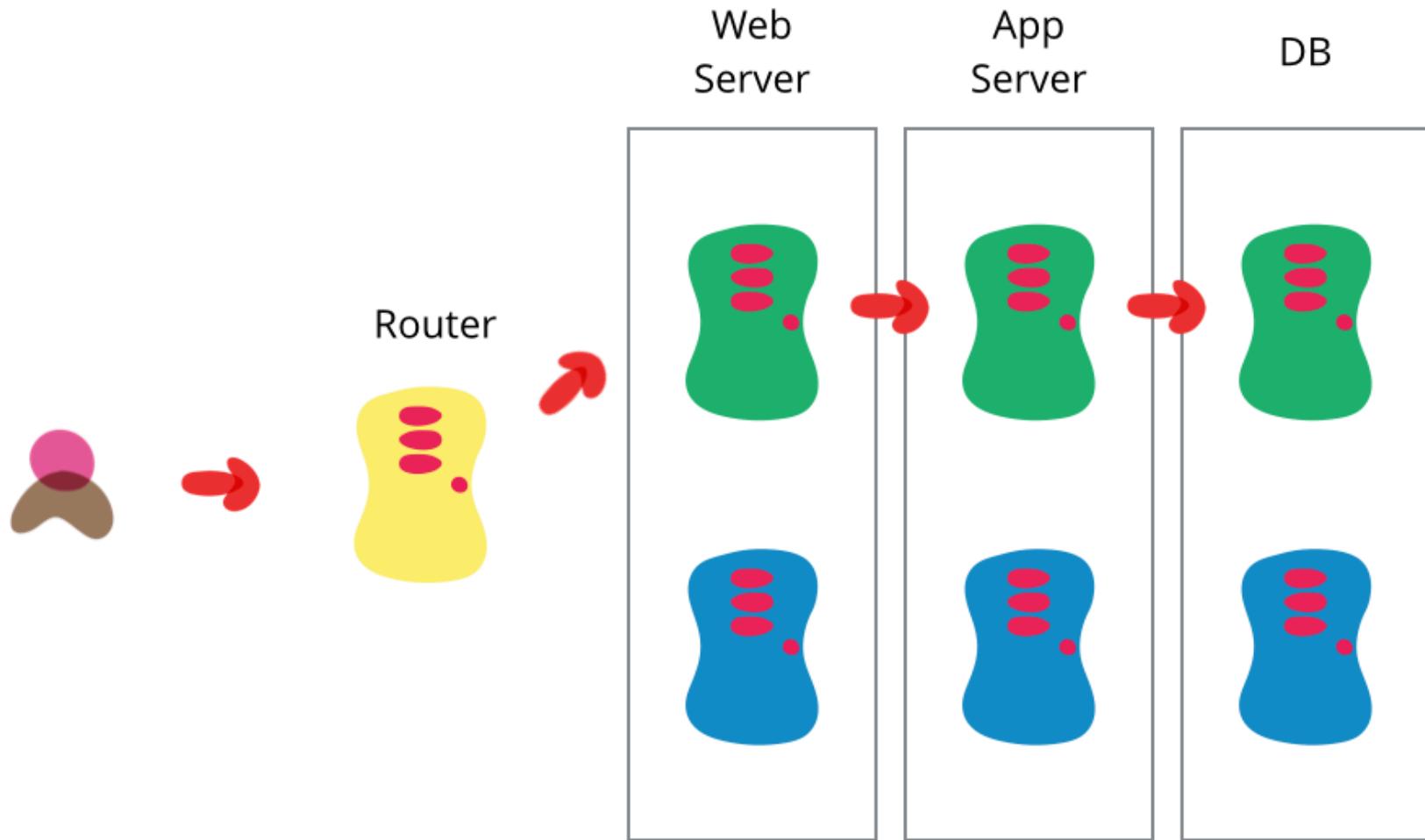
© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Immutable objects

- **String in Java**
- **You can't change a String, only create a new one**
- **Applies to devops:**
 - Never change a running server
 - Only create a new one that is better
 - Track the changes in a version control model



Blue Green Deployment



<http://martinfowler.com/bliki/BlueGreenDeployment.html>

DevOps tools

- Puppet, Chef
 - Automated configuration and deployment tools
 - Allow complex infrastructures to be re-configured automatically
- Vagrant
 - Create VMs instantly
- Plus many many more!



DevOps and Docker

- Docker is a key DevOps tool
- Speeds up the creation of repeatable deployments
- Consistency between development, test and production
- Versioned repository
- Works with Chef, Puppet, etc



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

What can be described and observed, can be automated and operated



<https://www.weave.works/blog/gitops-git-push-all-the-things>



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

GitOps

- Infrastructure as Code
- Terraform + Deployment + Containers + Build
- Everything is in Git
 - Any change to the infrastructure is a Pull Request



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>



THE TWELVE-FACTOR APP

INTRODUCTION

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

<http://12factor.net>



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

12 Factor Apps

- **Codebase**
 - One codebase in revision control, many deploys
- **Dependencies**
 - Explicitly define and declare
- **Config**
 - Store config in the environment
- **Backing Services**
 - Treat as attached resources
- **Build, Release, Run**
 - Strictly separate
- **Processes**
 - Execute the app as stateless processes
- **Port Binding**
 - Export services via port binding
- **Concurrency**
 - Scale out via processes
- **Disposability**
 - Fast startup and graceful shutdown
- **Dev/Prod Parity**
 - Keep dev/staging/prod as similar as possible
- **Logs**
 - Treat logs as event streams
- **Admin Processes**
 - Run admin/mgmt tasks as one-off processes



Summary

- Docker and the Container model
 - Lightweight virtualization and repeatability
 - Blue Green deployment
 - “Warehouse Scale” computing



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>