Cloud Computing and Big Data

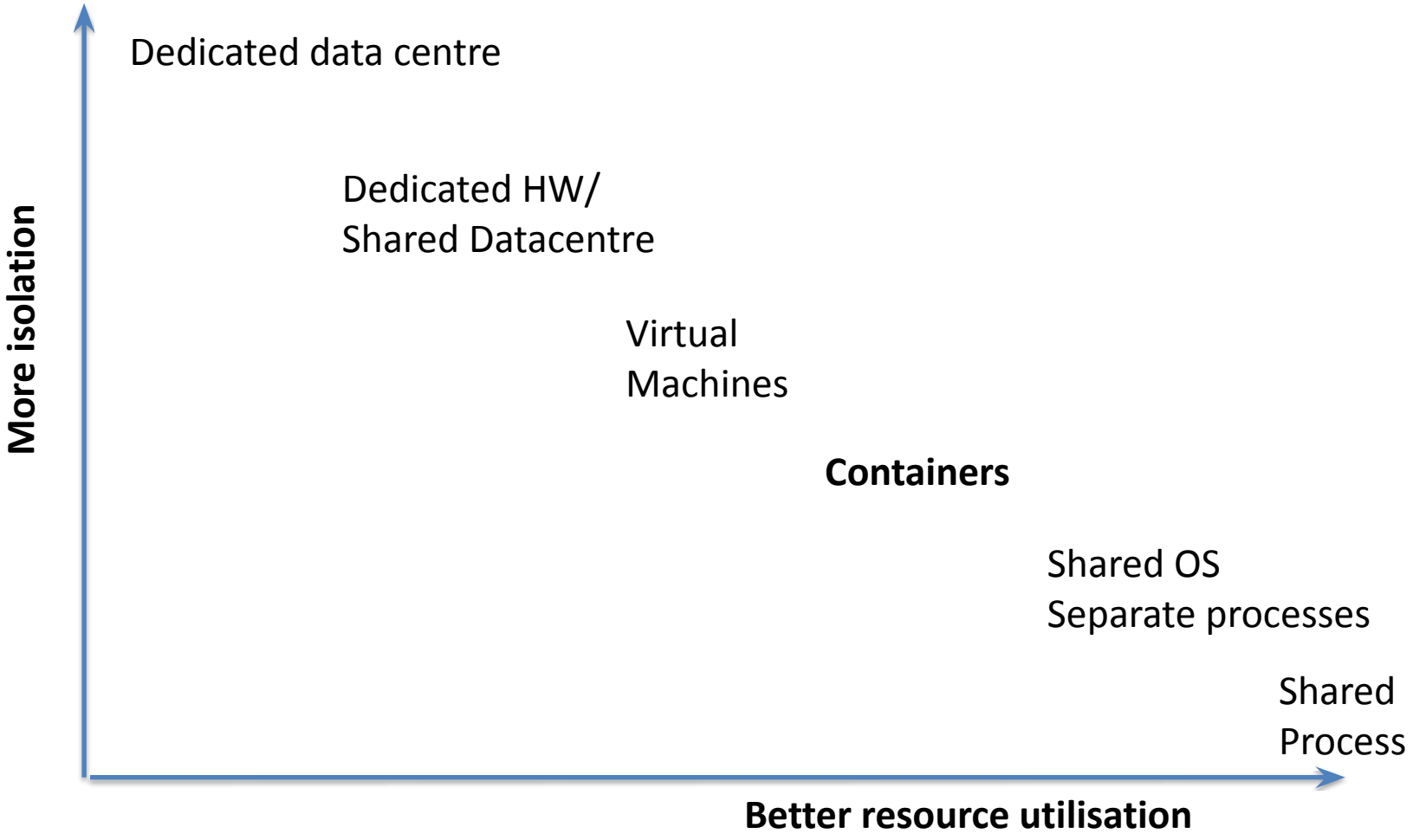
Containers and cloud native DevOps

Oxford University
Software Engineering
Programme
July 2021

Contents

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

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

Linux Containers (LXC)

- Virtualization inside the Linux Operating System
  - Not the only Linux option, but the most popular
- Allows virtualization including CPU, memory, disk
- Simple and effective

cgroups

- Control of resources by process:
  - blkio — this subsystem sets limits block devices such as physical drives
  - cpu - access to the CPU.
  - cpuacct — this reports on CPU usage
  - cpuset — this controls usage by CPUs in a multicore
  - devices — this denies or grants access to devices
  - freezer — suspends and resumes tasks
  - memory — controls and reports on memory usage
  - net_cls — tags network packets with ids for control
  - net_prio — priority of network traffic per interface.
  - ns — the namespace subsystem.

libcontainer and the Open Container Foundation

- A standardised interface into the container layer
  - Part of runC the open runtime from Docker
  - A key basis of the Open Container Foundation

Cloud Native Computing Foundation

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

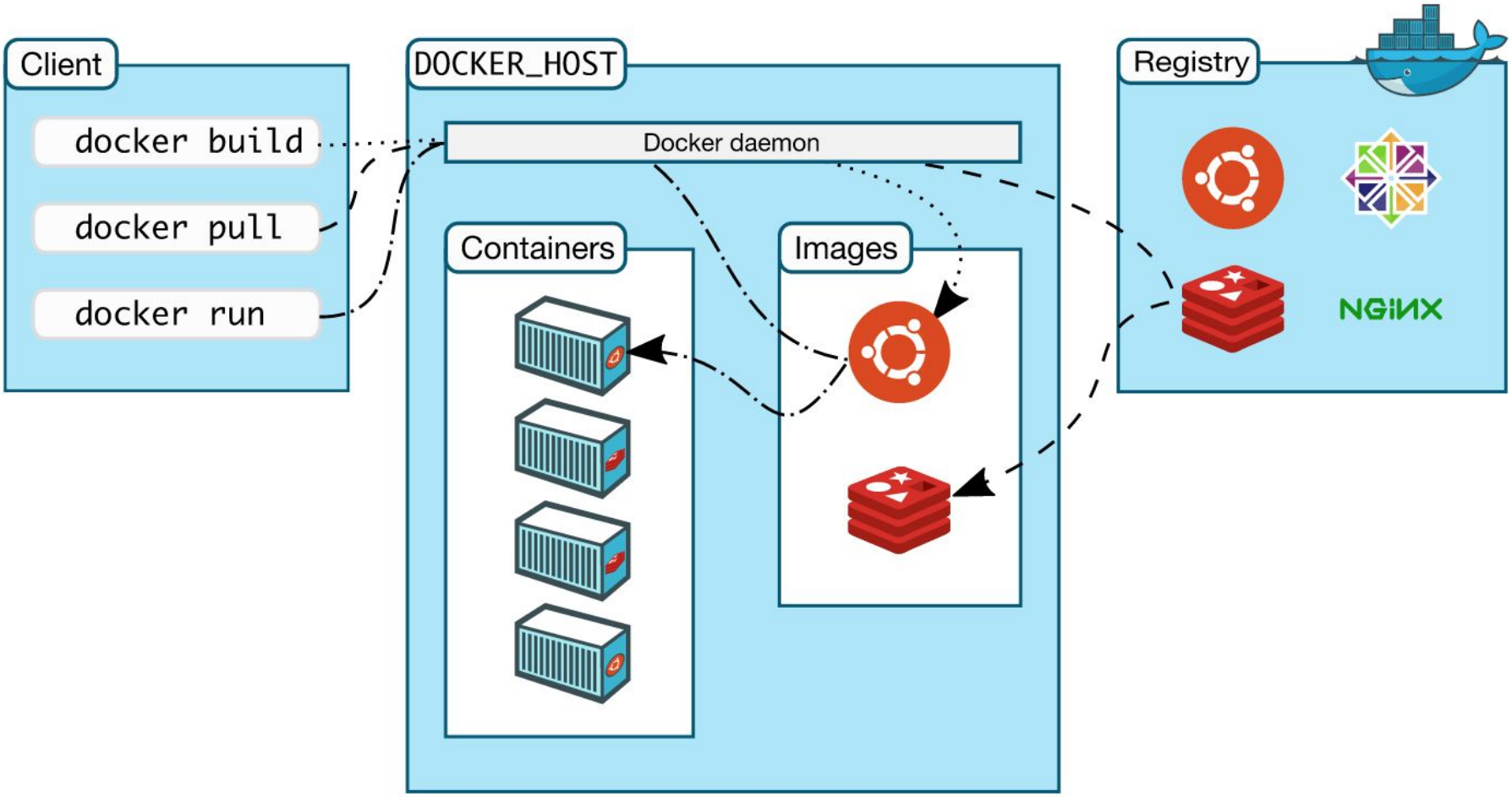
Docker on top of LXC

- 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 httplib2

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

- A way of configuring multiple Docker containers
  - Solves security issues
    - Shouldn't put secrets in Dockerfile or Docker image
  - Manages dependencies between containers

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
```

Docker Machine

- Manages docker servers
  - e.g. VirtualBox, Amazon, DigitalOcean
  - Let's you start/stop and configure Docker to talk to the remote server
  - Unfortunately Docker-Machine is no longer maintained :-(

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

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 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

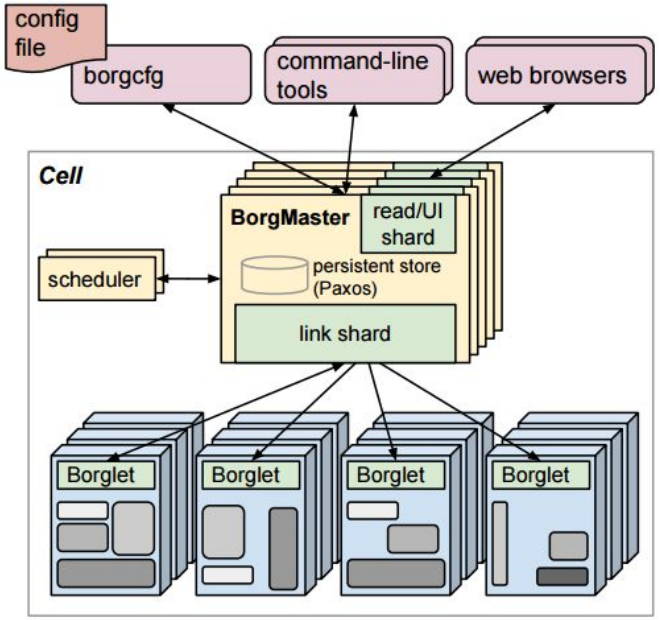


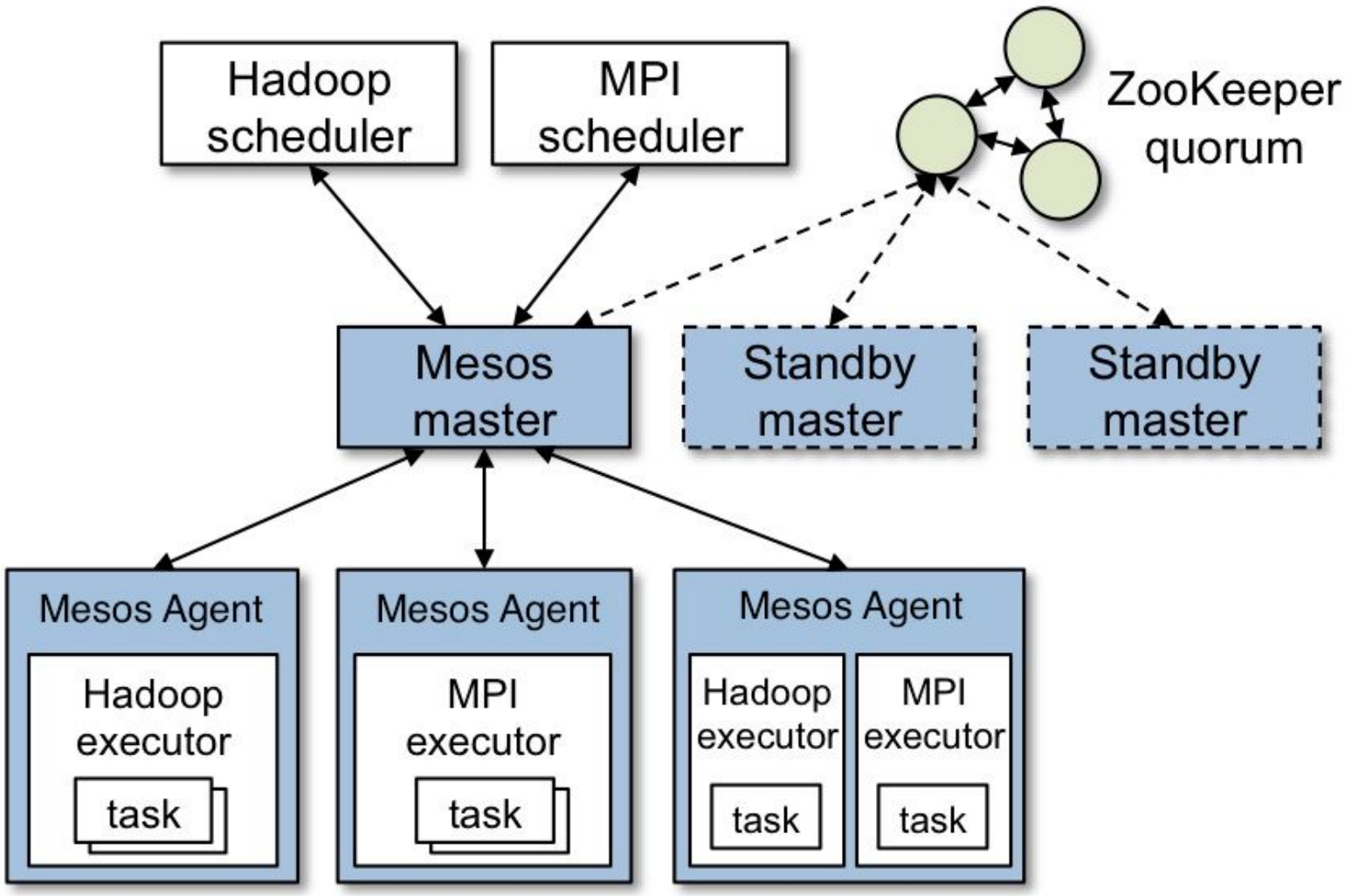
Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

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

Datacenter Operating System
aka Container Orchestration

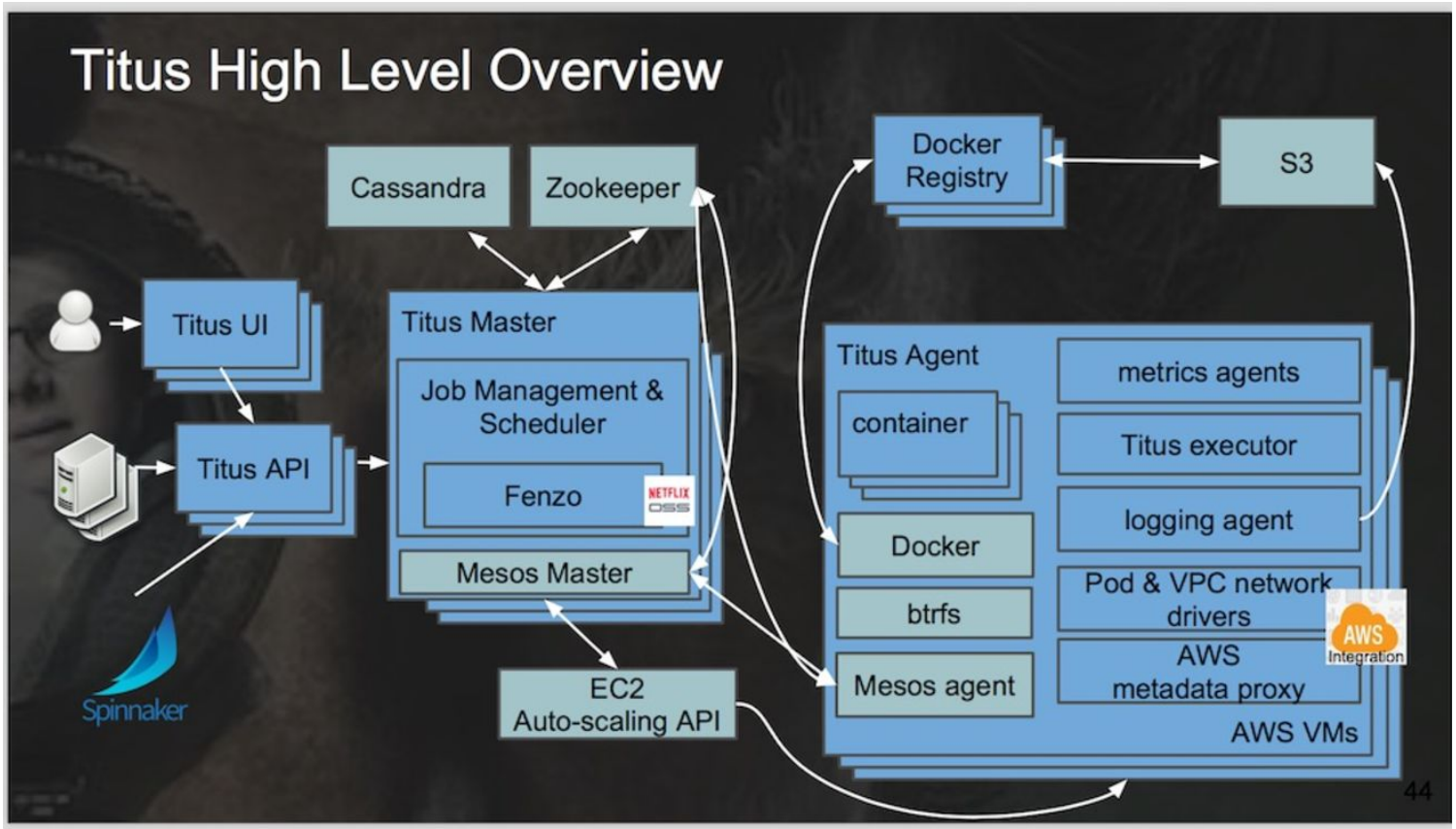
- Manages the placement of containers
  - Access to resources
  - Configuration and networking
  - Moves containers
  - Load balances across containers
- Effectively creating a single OS across a cloud
  - Containers vs Processes

Apache Mesos



Netflix Titus

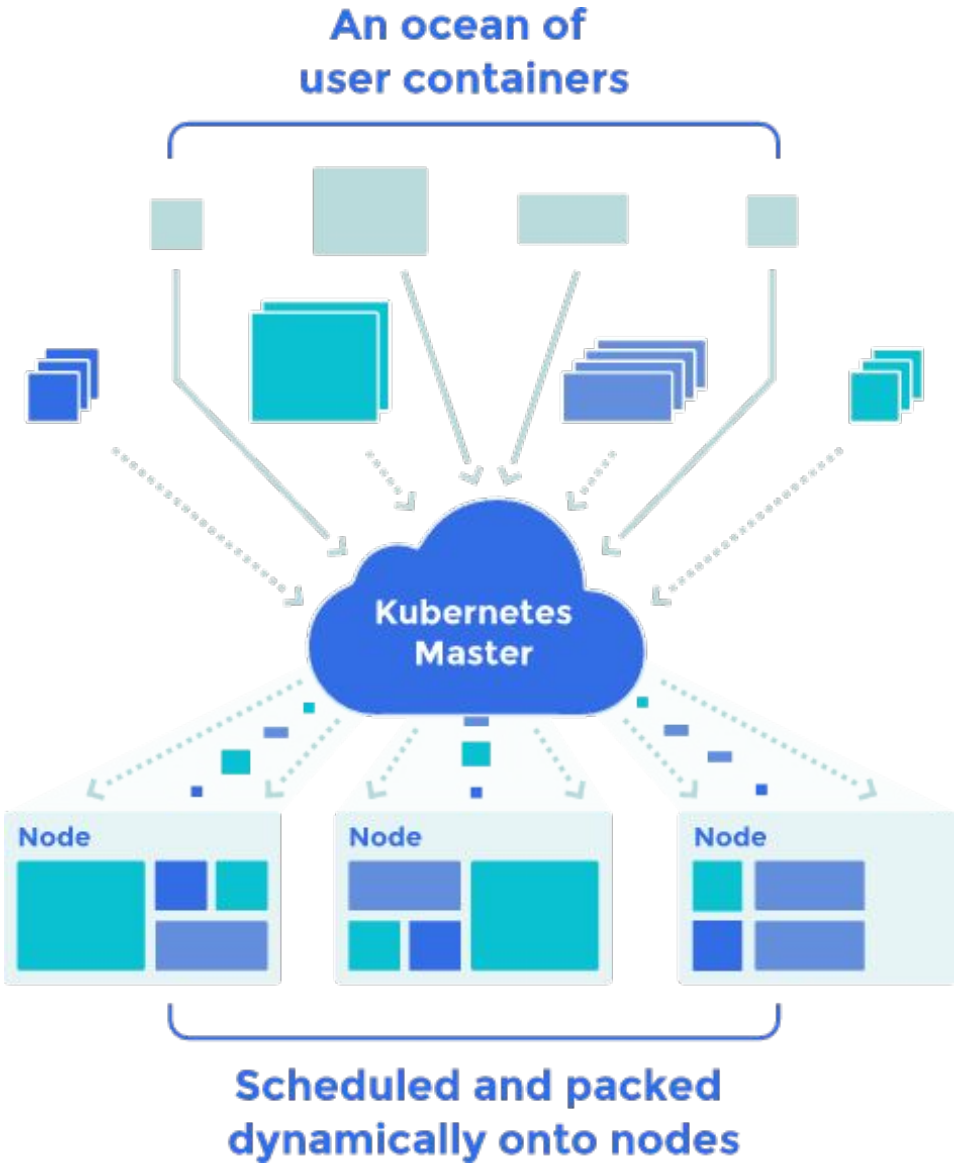
Running on 5000 AWS instances (m4.4xlarge and r3.8xlarge)
Three regions
10,000 containers running at any time
1,000,000 containers launched



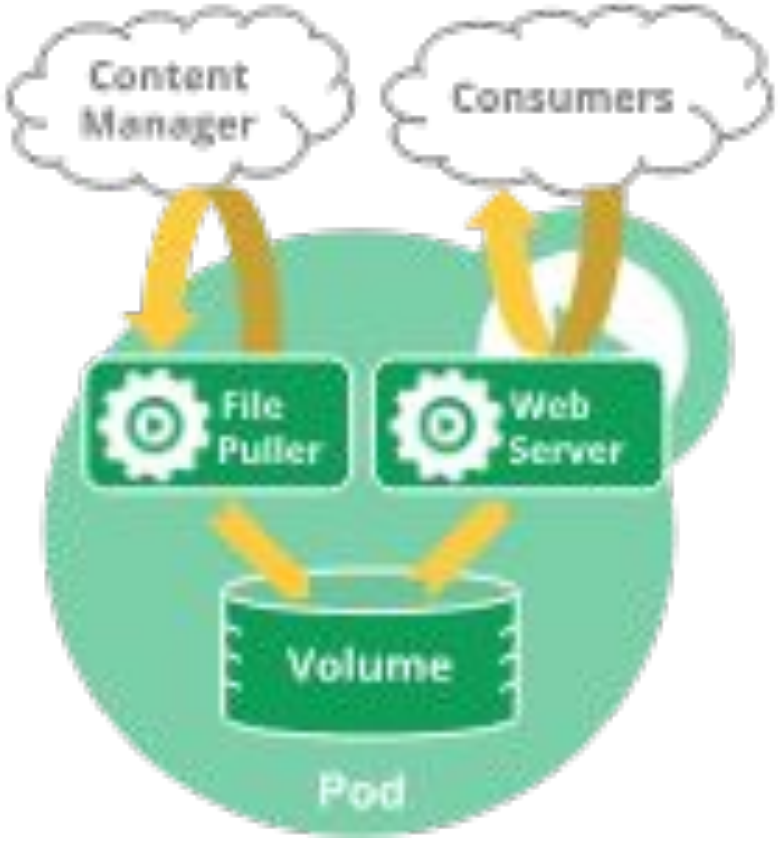
https://www.infoq.com/news/2017/07/netflix-titus

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

Namespaces

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

Kubernetes Operations (kops)

build passing | go report A | godoc reference

The easiest way to get a production grade Kubernetes cluster up and running.

What is kops?

We like to think of it as `kubectl` for clusters.

`kops` helps you create, destroy, upgrade and maintain production-grade, highly available, Kubernetes clusters from the command line. AWS (Amazon Web Services) is currently officially supported, with GCE and VMware vSphere in alpha and other platforms planned.

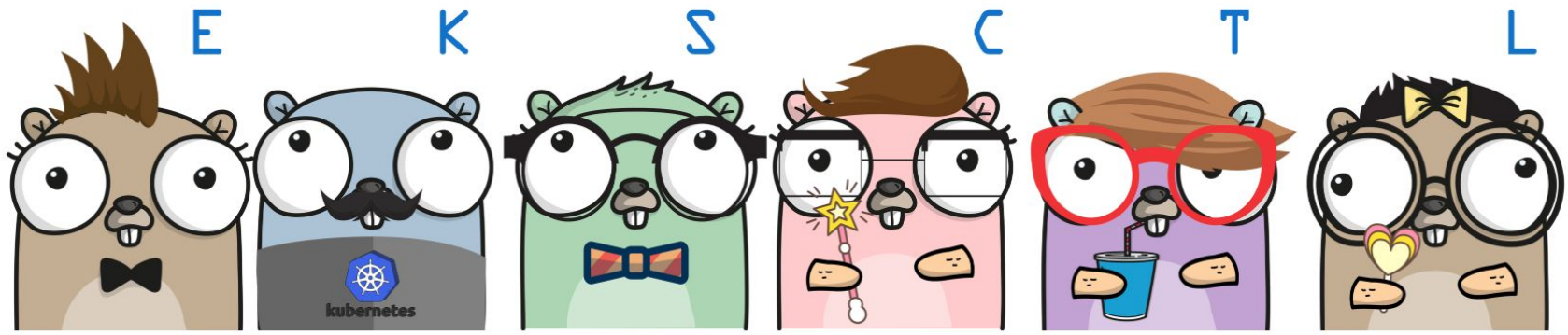
Can I see it in action?

```
  AutoscalingGroup/nodes.example.nivenly.com
        MinSize                 2
        MaxSize                 2
        Subnets                 [name:us-west-2a.example.nivenly.com]
        Tags                    {k8s.io/role/node: 1, Name: nodes.example.nivenly.com, KubernetesCluster: example.nivenly.com}
        LaunchConfiguration     name:nodes.example.nivenly.com


Cluster configuration has been created.

Suggestions:
 * list clusters with: kops get cluster
 * edit this cluster with: kops edit cluster example.nivenly.com
 * edit your node instance group: kops edit ig --name=example.nivenly.com nodes
 * edit your master instance group: kops edit ig --name=example.nivenly.com master-us-west-2a

Finally configure your cluster with: kops update cluster example.nivenly.com --yes

bash-3.2$ kops edit cluster $NAME
```

`eksctl` - a CLI for Amazon EKS

`eksctl` is a simple CLI tool for creating clusters on EKS - Amazon's new managed Kubernetes service for EC2. It is written in Go, and based on Amazon's official CloudFormation templates.

You can create a cluster in minutes with just one command – `eksctl create cluster`!



Usage

To download the latest release, run:

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest_release/eksctl_$(uname -s)_am
sudo mv /tmp/eksctl /usr/local/bin
```

Alternatively, macOS users can use Homebrew:

knative

Knative components

Serving

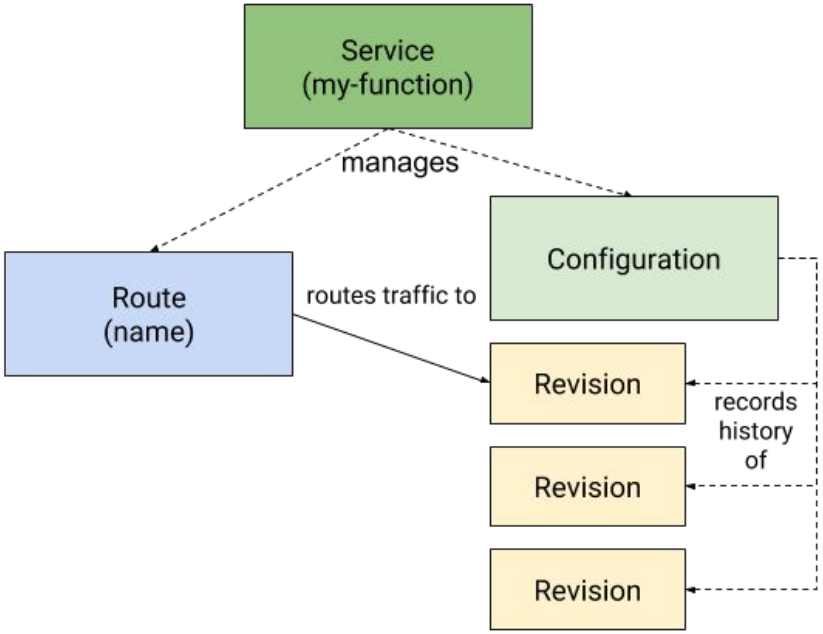
- Higher level abstraction, easy to reason about the object model
- Seamless autoscaling based on HTTP requests
- Gradual rollouts for new revisions
- Integrates networking and service mesh automatically
- Pluggable: connect your own logging and monitoring platform

Eventing

- Universal subscription, delivery, and management of events
- Build loosely coupled, event-driven systems with high-level objects
- Declarative binding between event producers and event consuming services
- Scalable from just a few events to live streams
- Custom event pipelines to connect with your own existing systems

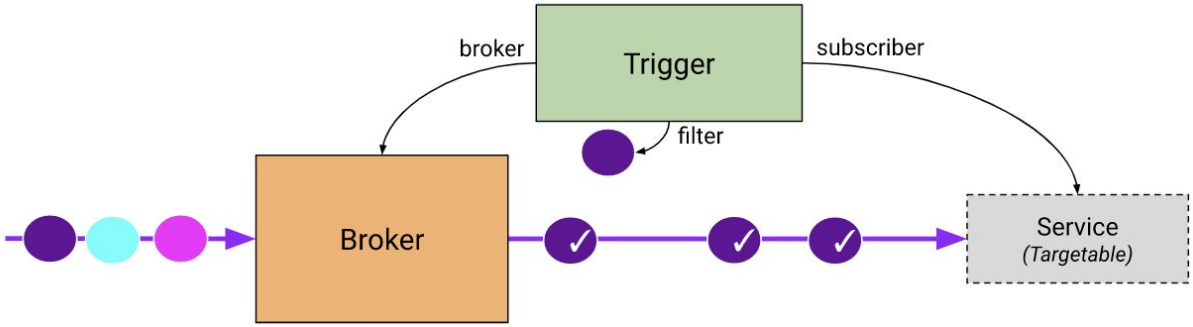
knative

Pronounced “kay-native”



Serverless for Kubernetes

- Serving
  - 0-n scaling for workloads
  - Blue-green deployment and versioning
- Eventing
  - Pub-Sub type messaging for k8s
  - Distribution of events to serving components
  - Pipelines and fanouts



kompose

Kompose (Kubernetes + Compose)

`kompose` is a tool to help users who are familiar with `docker-compose` move to Kubernetes. `kompose` takes a Docker Compose file and translates it into Kubernetes resources.

`kompose` is a convenience tool to go from local Docker development to managing your application with Kubernetes. Transformation of the Docker Compose format to Kubernetes resources manifest may not be exact, but it helps tremendously when first deploying an application on Kubernetes.

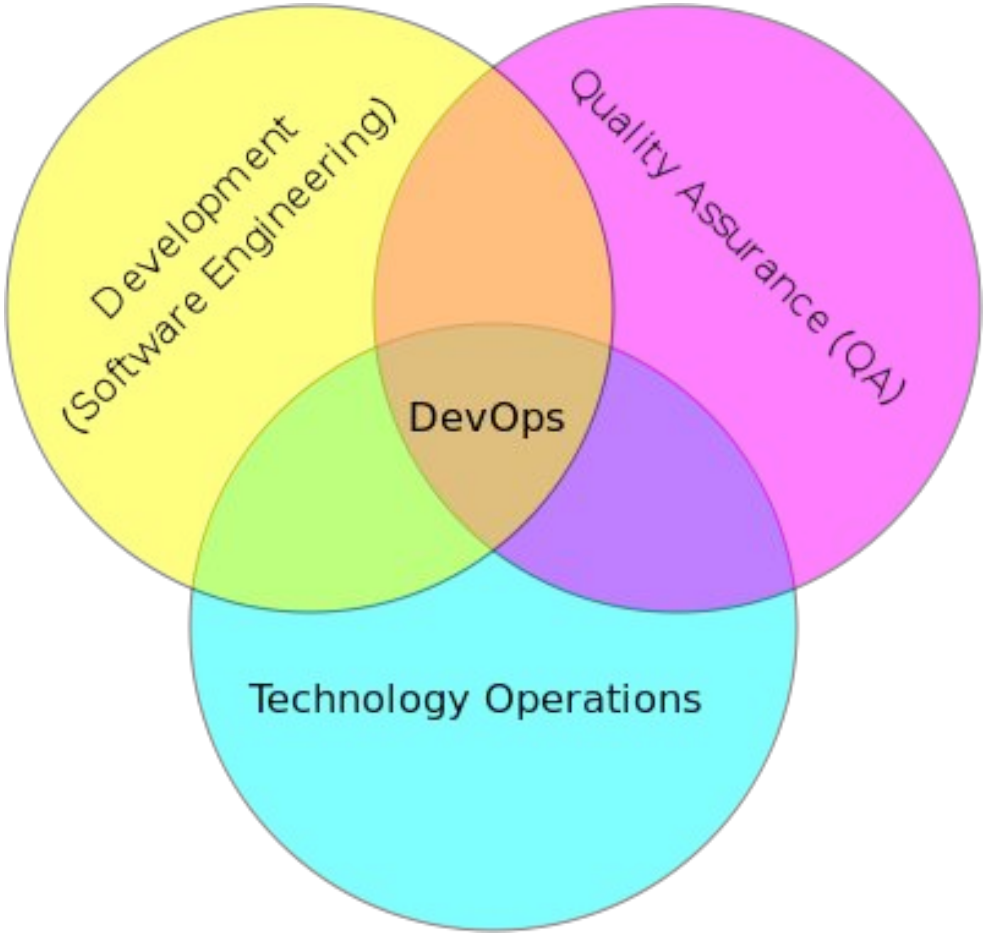
Use Case

Convert `docker-compose.yaml` into Kubernetes deployments and services with one simple command:

```
$ kompose convert -f docker-compose.yaml
INFO Kubernetes file "frontend-service.yaml" created
INFO Kubernetes file "redis-master-service.yaml" created
INFO Kubernetes file "redis-slave-service.yaml" created
INFO Kubernetes file "frontend-deployment.yaml" created
INFO Kubernetes file "redis-master-deployment.yaml" created
INFO Kubernetes file "redis-slave-deployment.yaml" created
```

Other examples are provided in the *examples* directory.

DevOps



DevOps

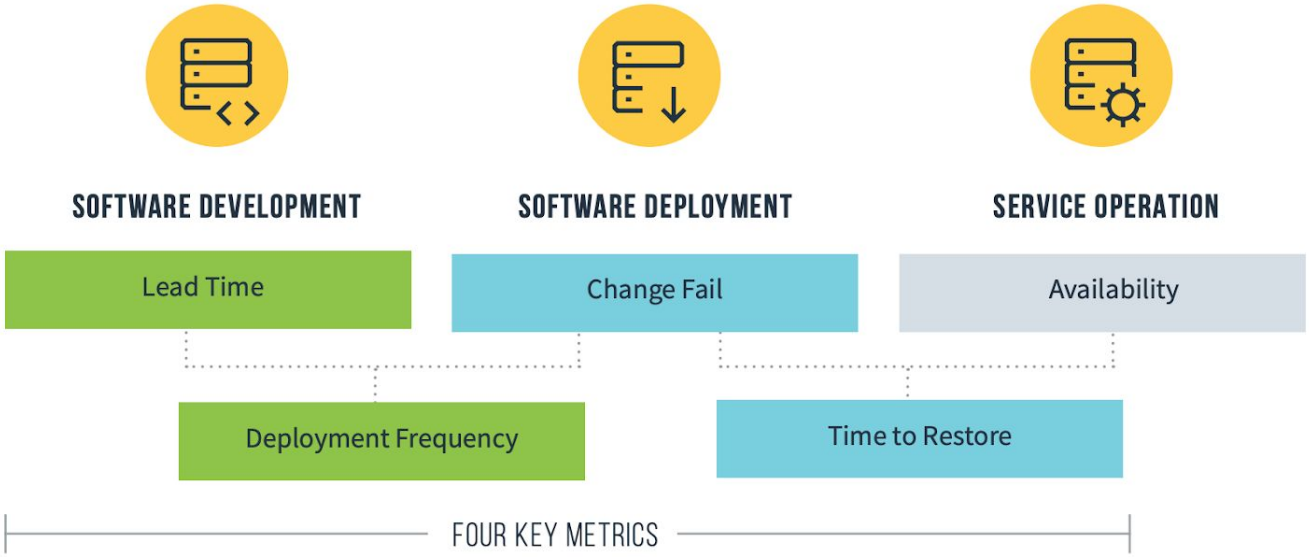
- DevOps is the codification of the interface between Development and Operations
  - Agile
  - Repeatable
  - Collaborative
  - Versioned
  - Automated

Cloud and DevOps

- It could be argued strongly that the rise of DevOps is tied to the rise of Cloud
  - Clear requirement for automated, repeatable configuration and deployment
  - Reducing the hardware provisioning time has highlighted the challenges

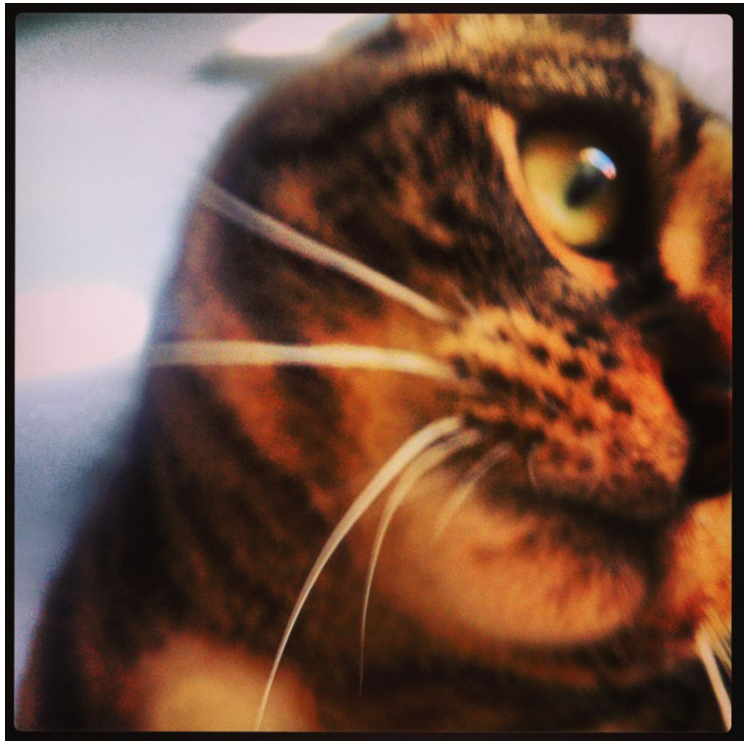
DORA metrics

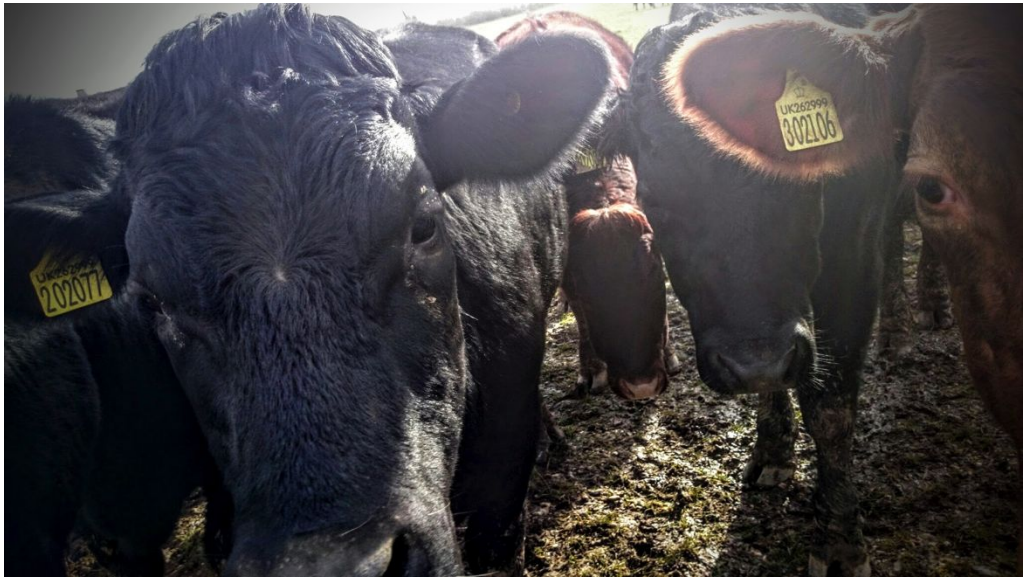
- **Deployment Frequency**—How often an organization successfully releases to production
- **Lead Time for Changes**—The amount of time it takes a commit to get into production
- **Change Failure Rate**—The percentage of deployments causing a failure in production
- **Time to Restore Service**—How long it takes an organization to recover from a failure in production



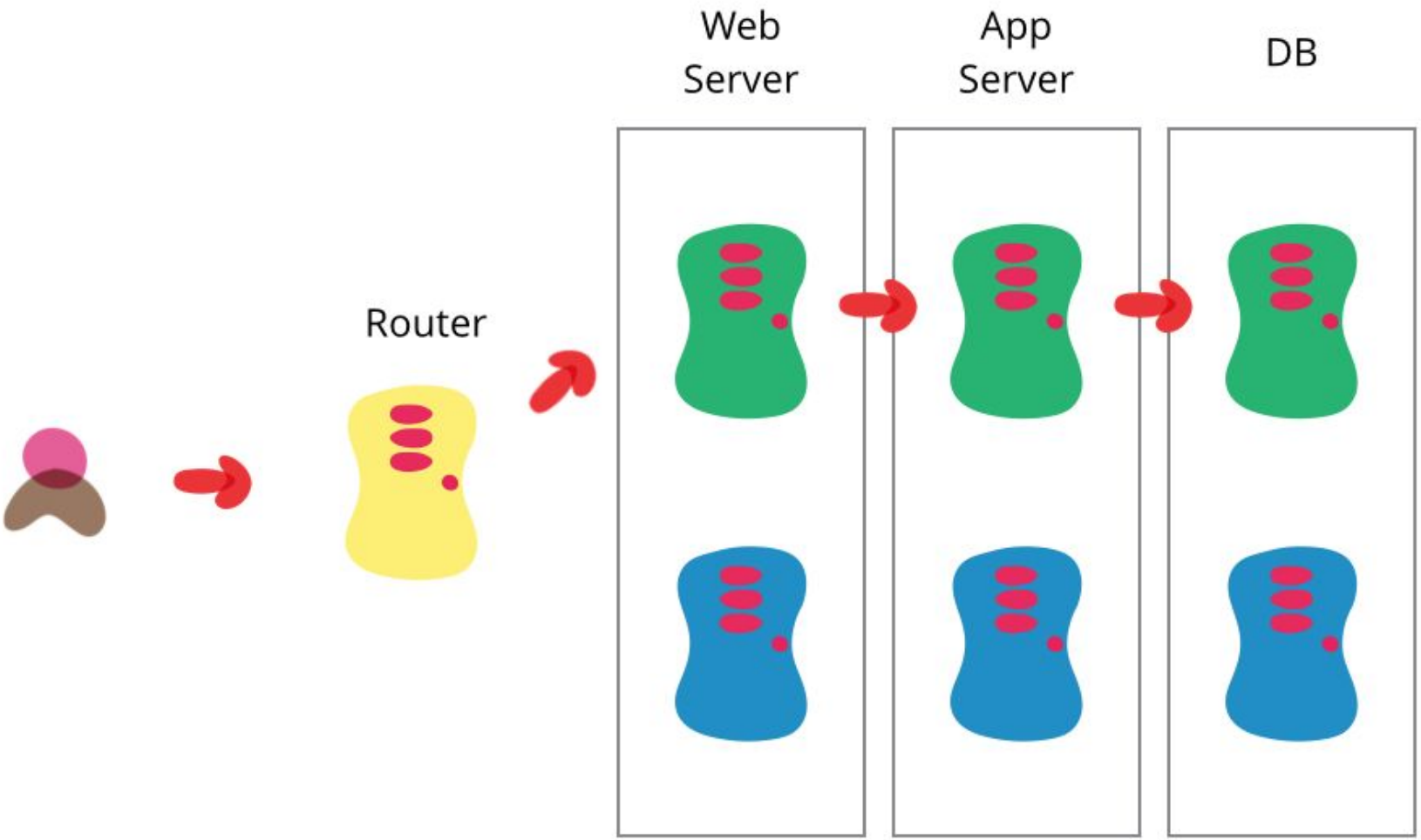
Aspect of Software Delivery Performance*	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per day and once per week	Between once per week and once per month	Between once per month and once every six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one day	Between one day and one week	Between one week and one month	Between one month and six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day[a]	Less than one day[a]	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0-15%[b,c]	0-15%[b,d]	0-15%[c,d]	46-60%

Kittens vs Cattle
(An unpleasant but effective analogy)



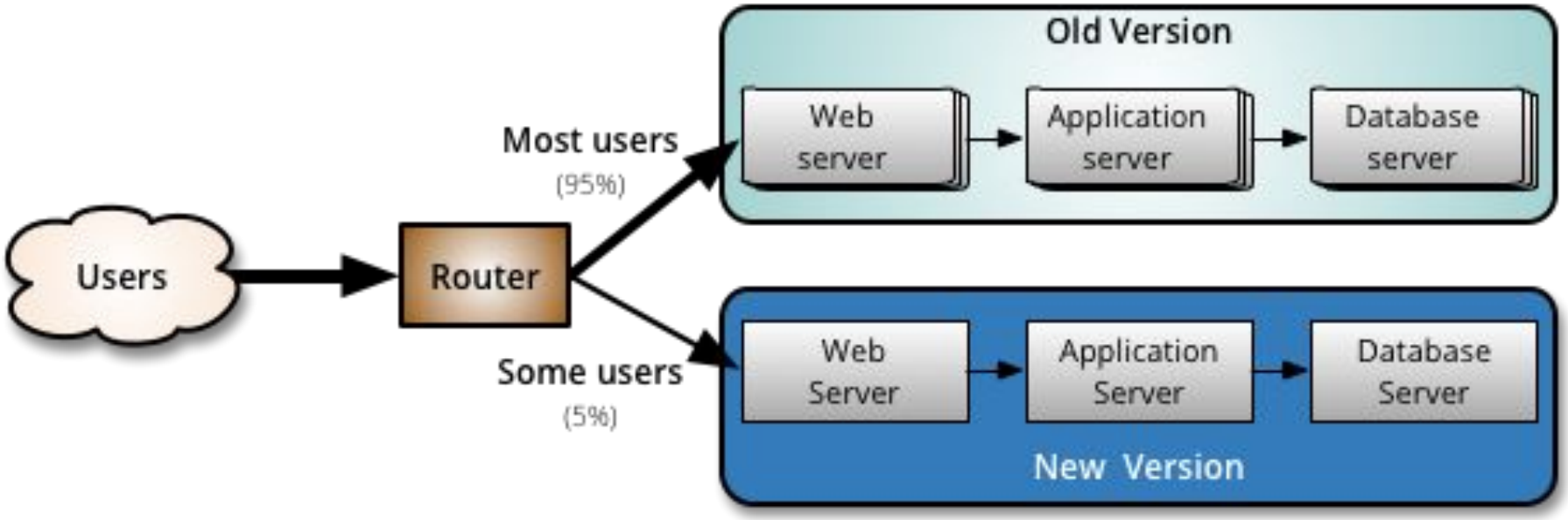


Blue Green Deployment



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

Canary Deployment



DevOps tools

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

DevOps and Docker

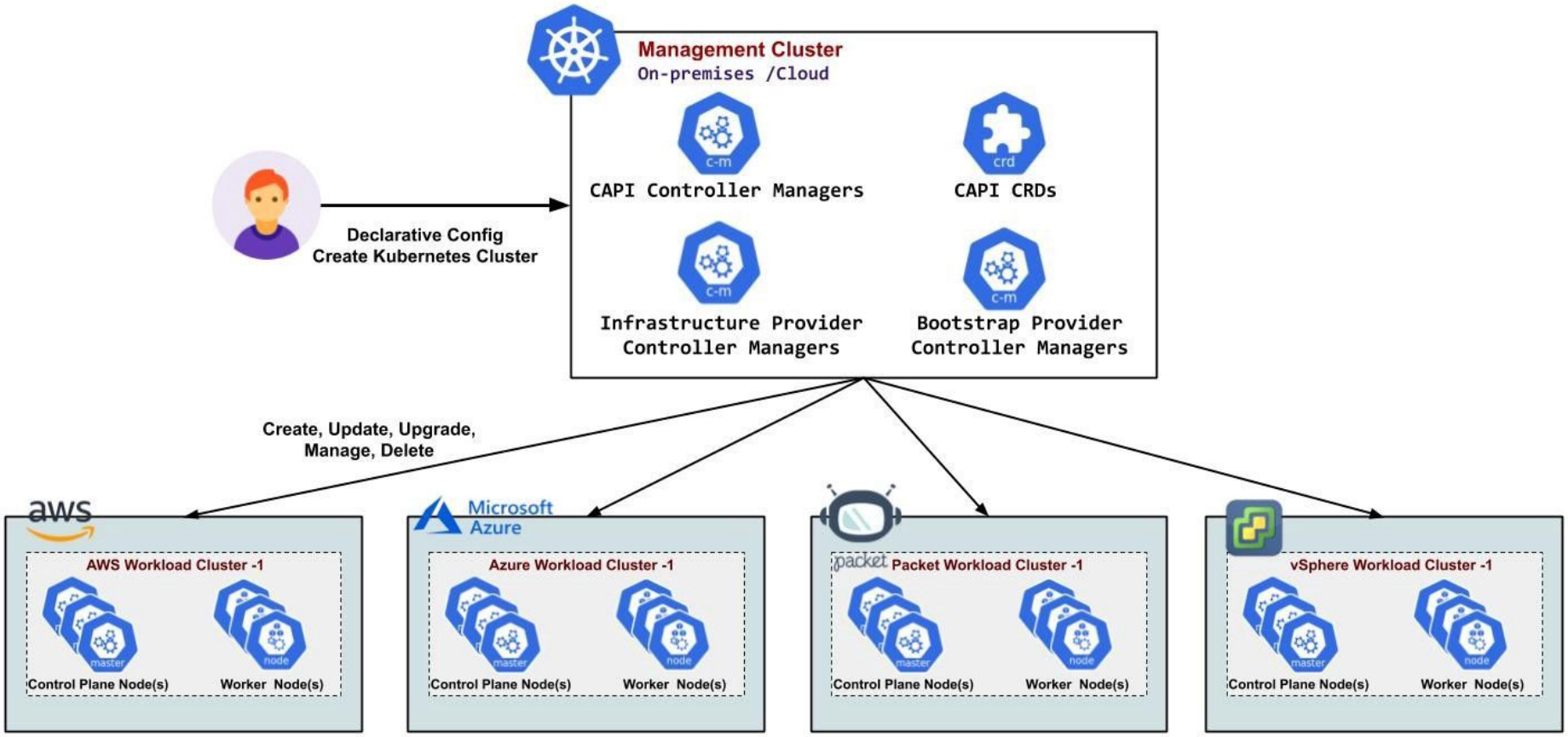
- Docker is a key DevOps tool
- Speeds up the creation of repeatable deployments
- Consistency between development, test and production
- Versioned repository
- Works with Chef, Puppet, etc

Terraform

```
resource "aws_instance" "iac_in_action" {
  ami               = var.ami_id
  instance_type     = var.instance_type
  availability_zone = var.availability_zone

  // dynamically retrieve SSH Key Name
  key_name = aws_key_pair.iac_in_action.key_name

  // dynamically set Security Group ID (firewall)
  vpc_security_group_ids = [aws_security_group.iac_in_action.id]

  tags = {
    Name = "Terraform-managed EC2 Instance for IaC in Action"
  }
}
```

```
$ terraform plan
An execution plan has been generated and is shown below.

Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_ebs_volume.iac_in_action will be created
  + resource "aws_ebs_volume" "iac_in_action" {
      + arn               = (known after apply)
      + availability_zone = "us-east-1a"
      + encrypted         = (known after apply)
      + id                = (known after apply)
      + iops              = 1000
      + kms_key_id        = (known after apply)
      + size              = 100
      + snapshot_id       = (known after apply)
      + tags              = {
          + "Name" = "Terraform-managed EBS Volume for IaC in Act
        }
      + type              = "io1"
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Kubernetes Cluster API (CAPI)

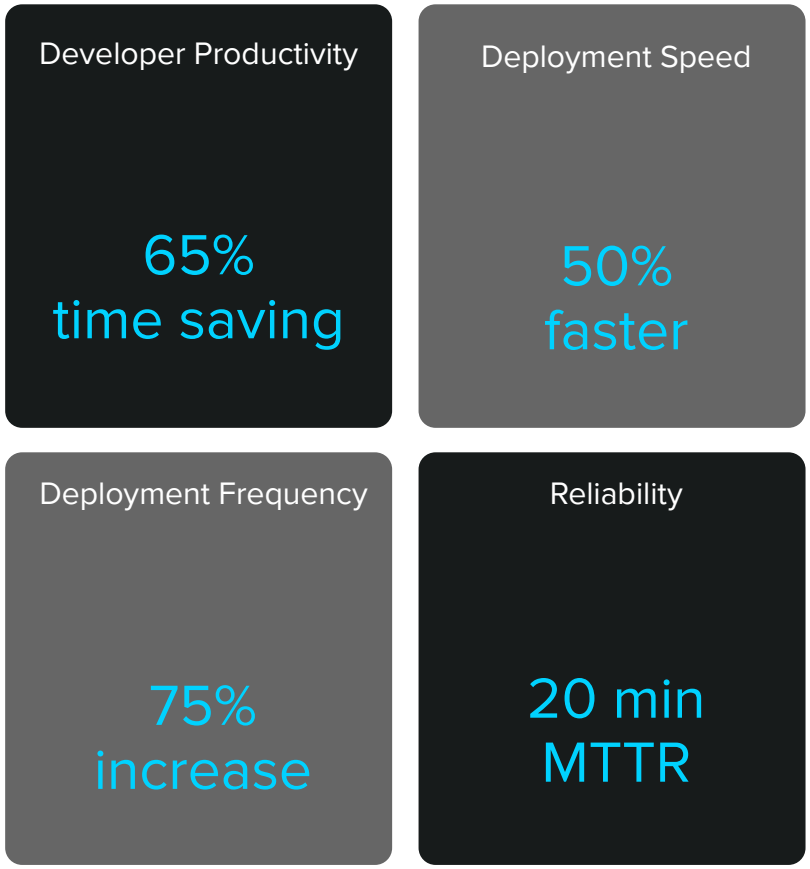


What can be described and observed, can be automated and operated

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

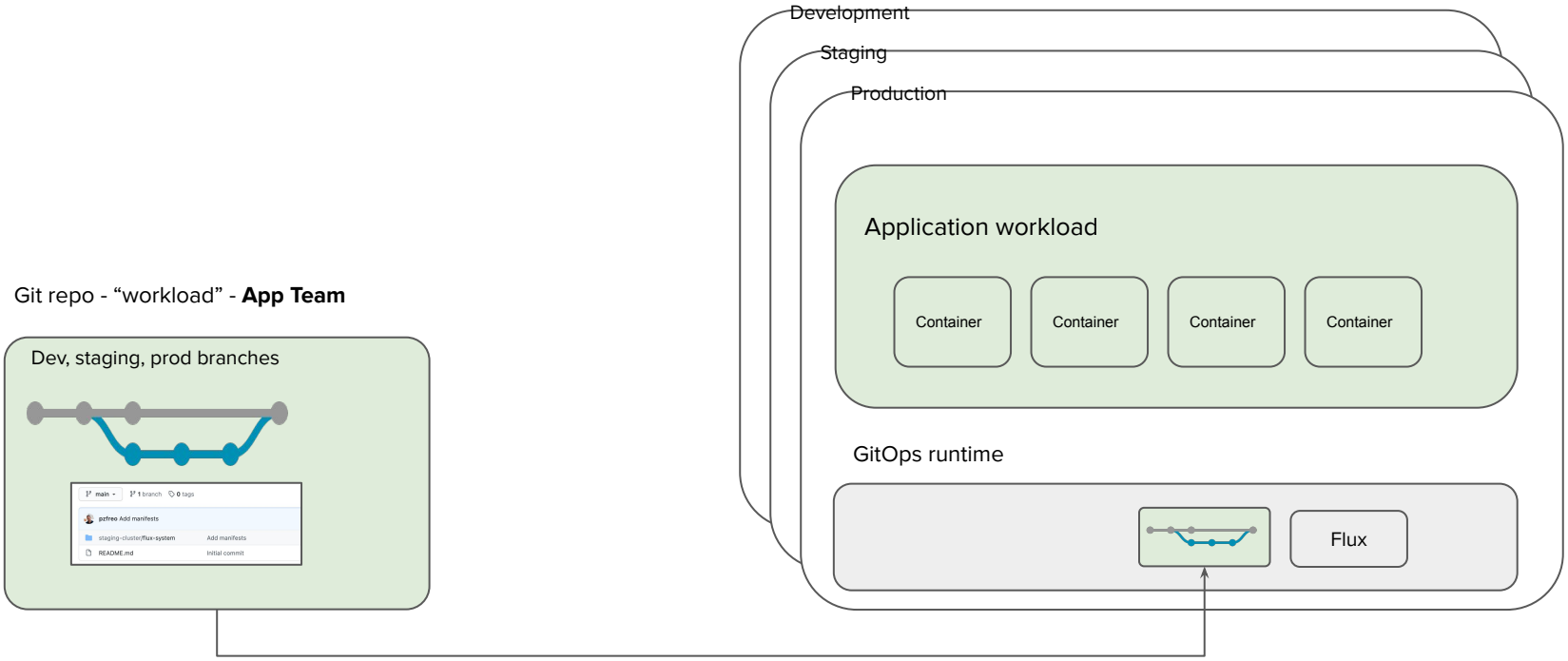
Mettle Case Study

- Mettle is a business banking brand of NatWest bank. Designed to provide an agile banking experience for small business.
- The team is using AWS and Kubernetes to build a completely new technology stack
- Using the GitOps approach the team have been able to revolutionise their agility and reliability
- Overall developers can now spend 75% more of their time working on creating new features and business value
- GitOps also improves reliability dramatically. The team can rebuild their entire system in just 20 minutes. This is a revolution for Mean Time to Recovery (MTTR)



What is GitOps?

Kubernetes clusters
Entirely configured declaratively from Git

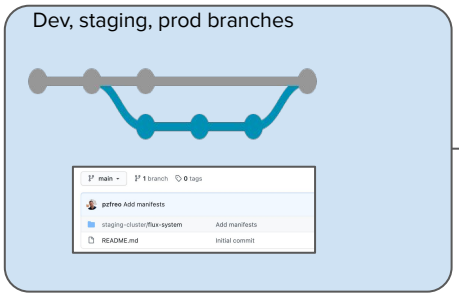


How it works?

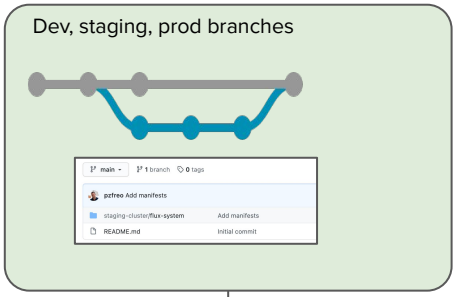
1. A change is created and tested locally
2. A Pull Request (PR) against the dev branch is created
3. The PR is merged
4. The dev environment automatically pulls this
5. If there are problems, cluster can be rolled back instantly
6. Once the dev environment is tested, the change can be merged into staging and then production

What is GitOps?

Git repo - “cluster config” - **Platform team**

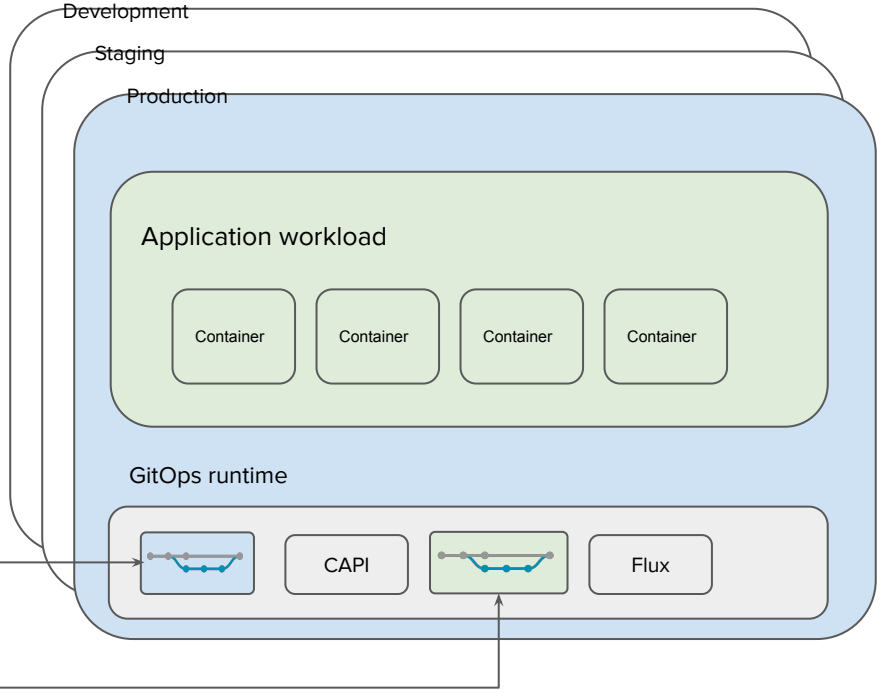


Git repo - “workload” - **App Team**



Kubernetes clusters

Entirely configured declaratively from Git



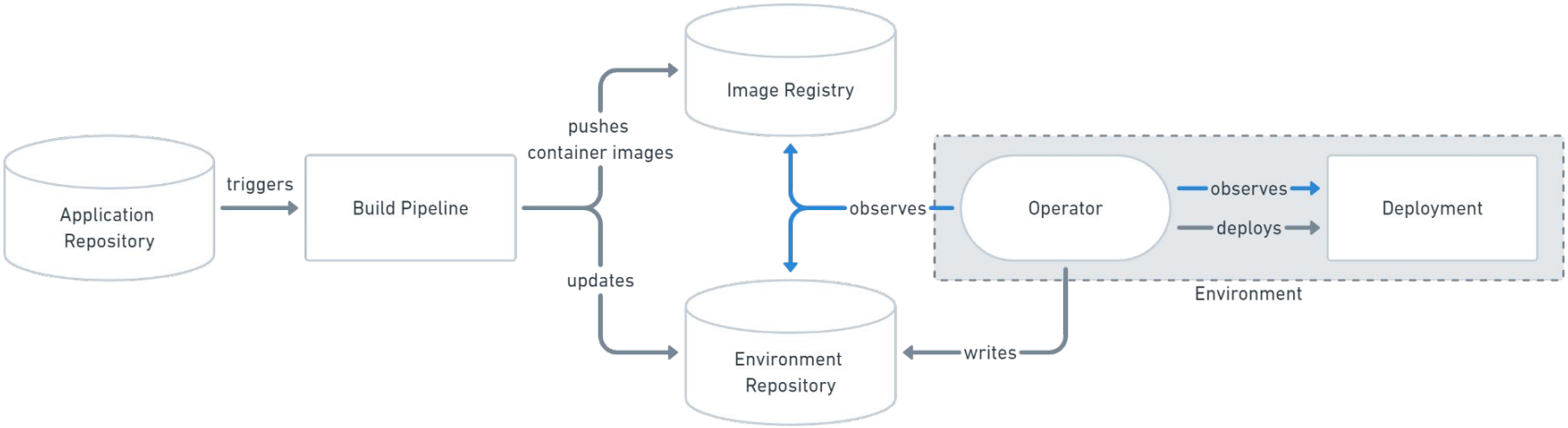
How it works?

1. A change is created and tested locally
2. A Pull Request (PR) against the dev branch is created
3. The PR is merged
4. The dev environment automatically pulls this
5. If there are problems, cluster can be rolled back instantly
6. Once the dev environment is tested, the change can be merged into staging and then production

Why?

- Git is well understood by devs, DevOps and Platform teams
- High quality UIs (Github, Gitlab, etc) and CLI tools
- Standard workflow for approving changes (and rolling back)
  - Branches, Pull Requests (PRs), Merges
  - Every change is approved, authorised and auditable
- With the Cluster API (CAPI), this approach can be applied to both clusters as well as workloads
- Can be rolled out to multiple clusters, fleets and edge networks

GitOps



https://www.gitops.tech/#what-is-gitops

GitOps Principles

The group's initial task – to "clearly define a vendor-neutral, principle-led meaning of GitOps" – is expressed as GitOps *Principles*, which are the foundation of GitOps practices.

This aims to give participants in the cloud native ecosystem a common understanding of GitOps systems based on shared principles, rather than a matter of individual opinion or implementation preference.

🚧 The GitOps Principles are currently a work in progress. For current status, follow PR #48.

🕰 In the meantime, you may refer to the initial Draft Definition, which proposes the principles as follows:

1. **Declarative Configuration**: All resources managed through a GitOps process must be completely expressed declaratively.
2. **Version controlled, immutable storage**: Declarative descriptions are stored in a repository that supports immutability, versioning and version history. For example, git.
3. **Automated delivery**: Delivery of the declarative descriptions, from the repository to runtime environment, is fully automated.
4. **Software Agents**: Reconcilers maintain system state and apply the resources described in the declarative configuration.
5. **Closed loop**: Actions are performed on divergence between the version controlled declarative configuration and the actual state of the target system.

https://github.com/gitops-working-group/gitops-working-group

Questions?