



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

DevOps and GitOps

Dr. Philipp Leitner



philipp.leitner@chalmers.se



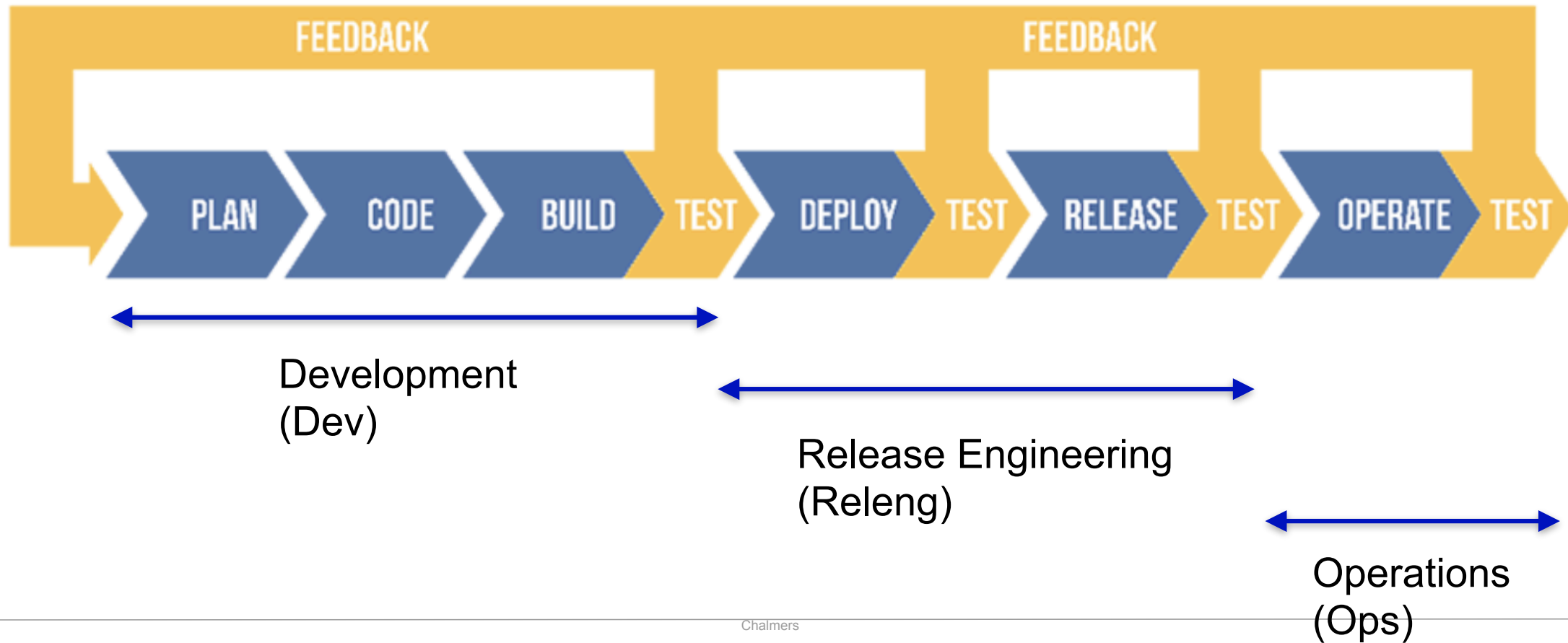
@xLeitix

LECTURE 5

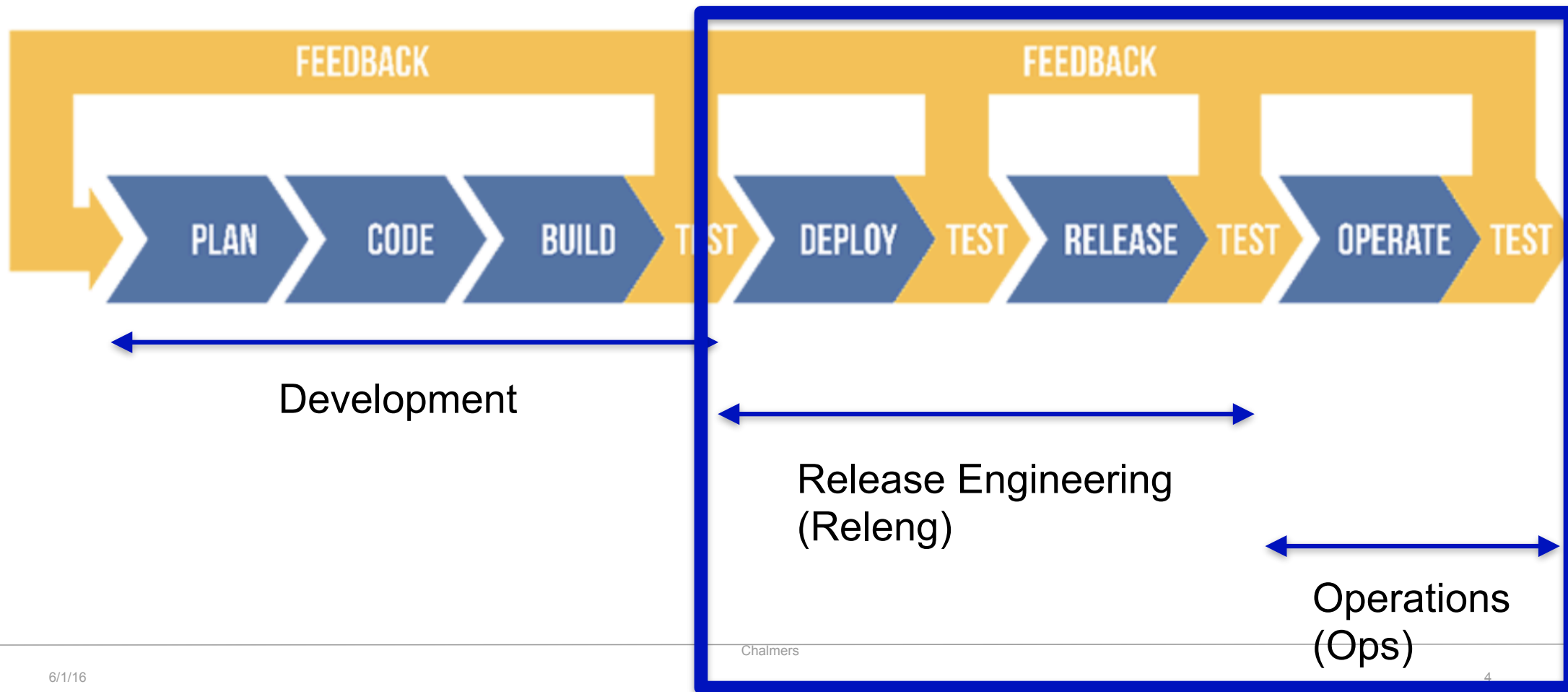
Covers

CD Pipelines
DevOps and GitOps

SaaS Application Lifecycle



SaaS Application Lifecycle



Release Engineering

Traditional Tasks:

- Maintaining the build pipeline
- Quality Assurance (beyond unit testing)

Release Engineering

Traditional Tasks:

- Maintaining the build pipeline
- Quality Assurance (beyond unit testing)

Operations (ops)

Traditional Tasks:

- Capacity planning and capacity management
- Systems administration
- Incident management

Build Pipelines

Build Server

To enable CI/CD you need:

... a code repository (e.g., Git)

... a **build server** (e.g., Jenkins, Travis, integrated in GitLab)

The build server runs the **continuous integration (deployment) pipeline**

✓ passed

#861



🔗 master 🔗 [ffce3fba](#)



Improve local deployment docs



🕒 00:02:43

📅 3 months ago

✓ passed

#860



🔗 master 🔗 [44902938](#)



Extend local deployment docs



🕒 00:02:38

📅 3 months ago

✗ failed

#859



🔗 master 🔗 [0ee12db7](#)



Attempt to fix Heroku deploy...



🕒 00:01:51

📅 3 months ago

✗ failed

#858



🔗 master 🔗 [eb2fea33](#)



Add local deployment instruc...



🕒 00:01:46

📅 3 months ago

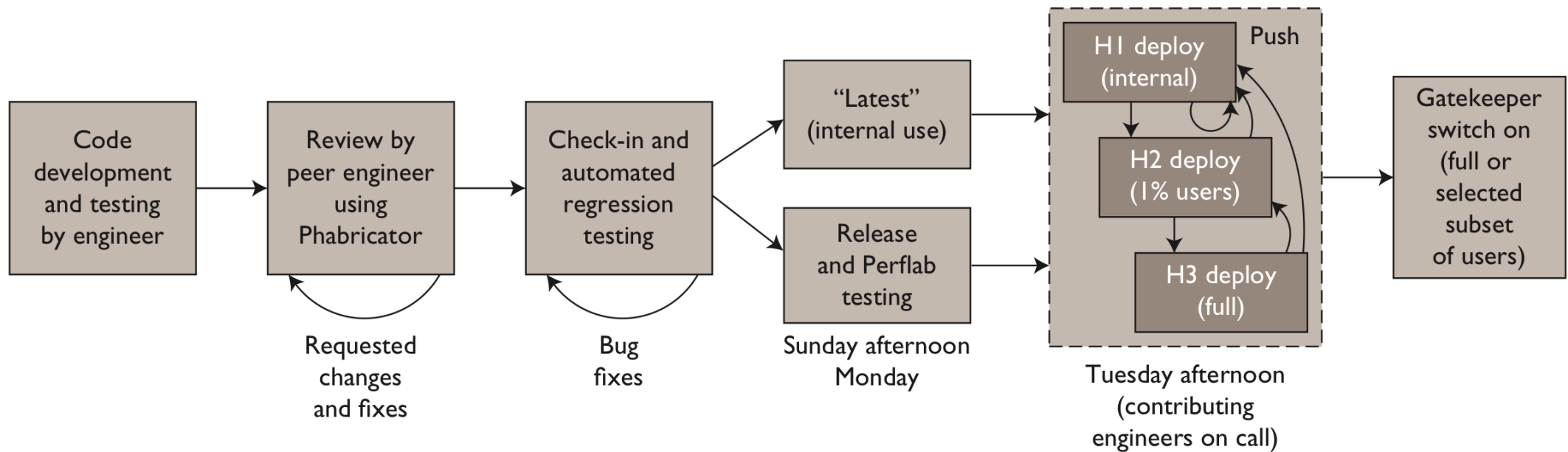
Build Status

The outcome of each CI run is a **build status**

Green - all quality gates passed, everything ok

Red - at least one gate (compile, test, deployment, ...) failed

A Real-Life Example (from Facebook, 2013)



Source:
Feitelson, Frachtenberg, Kent. Development and Deployment at Facebook. IEEE Software, 2013.

Auto DevOps in GitLab

In Assignment 3 you will get to know and use GitLab's **Auto DevOps** feature

Fancy name for a pre-configured CI/CD pipeline

Intro:

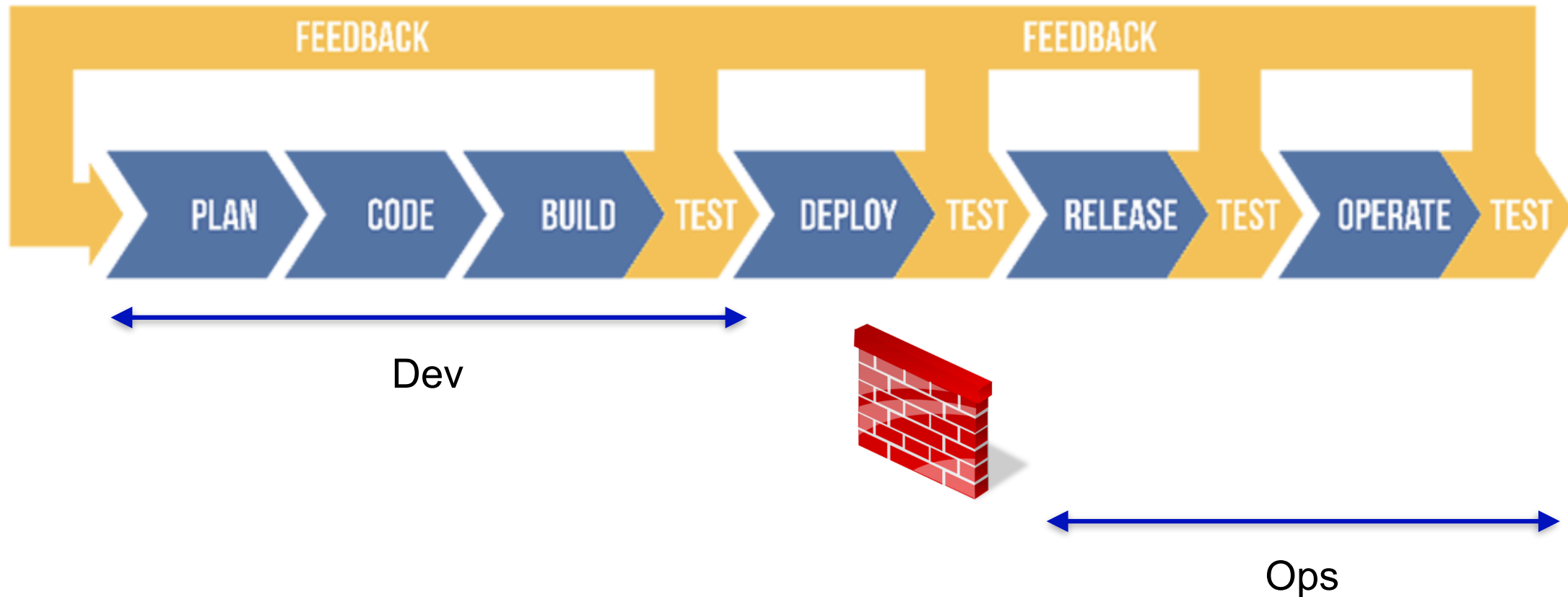
<https://docs.gitlab.com/ee/topics/autodevops/>

Customize the generated pipeline through a file `.gitlab-ci.yml` file in project root



DevOps

Dev-Ops Handover

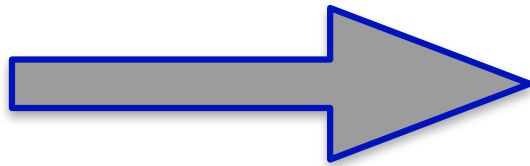


Traditional Dev-Ops Handover

- (1) Development team specs, builds, and tests new version*
- (2) Development makes a new release*
(“lobs it over the wall” to the ops team)
- (3) Ops takes new release, provisions servers, deploys and runs it*

Traditional Dev-Ops Handover

- (1) Development team specs, builds, and tests new version*
- (2) Development makes a new release*
(“lobs it over the wall” to the ops team)
- (3) Ops takes new release, provisions servers, deploys and runs it*



Challenges:

- Friction between dev and ops
- Automation changes the nature of ops

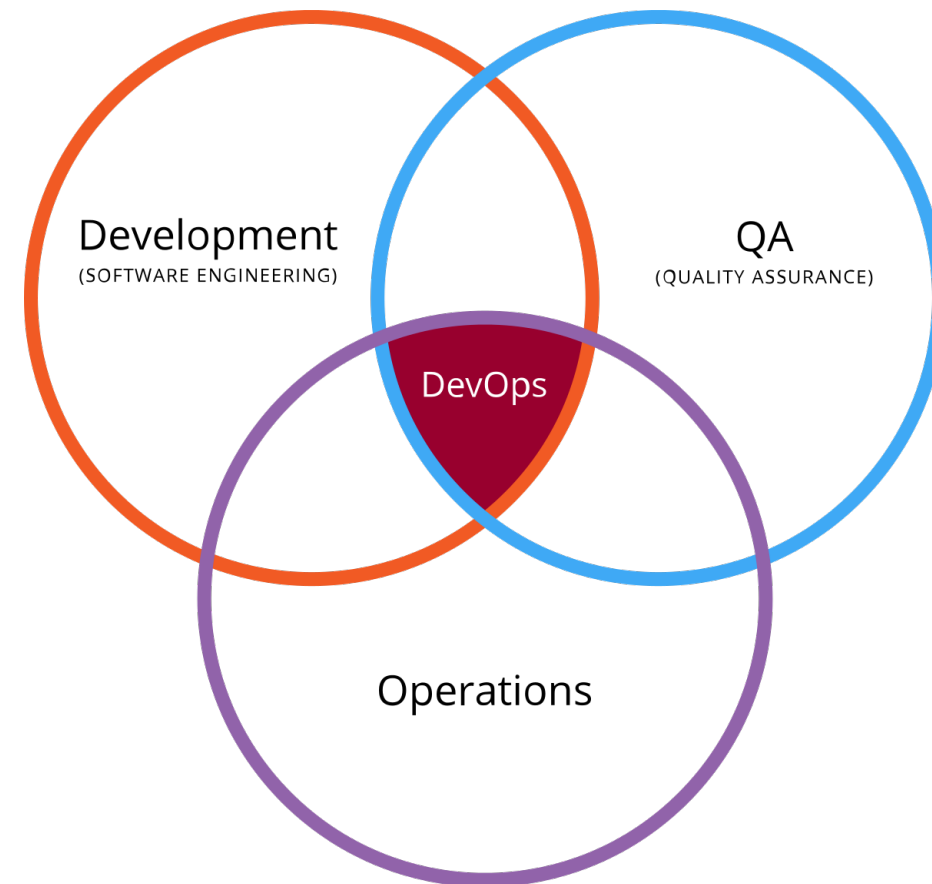
	Dev	Ops
Wants:	<ul style="list-style-type: none"> • Release quickly • Have few technology restrictions • Be able to change anything at any time 	<ul style="list-style-type: none"> • Stable, well-documented releases • Clear release windows that are not too frequent • Processes to be followed
Gets evaluated on:	<ul style="list-style-type: none"> • Release of new features and bugfixes • “Velocity” 	<ul style="list-style-type: none"> • Number of incidents • Incident response time • “Stability”
Ideal world:	Software can be released whenever something is ready, on short notice, and with little vetting.	Nobody is ever releasing anything, except maybe to fix bugs :)

(..) 80% of operations issues originate in design and development. (..) [However], when systems fail, there is a natural tendency to look first to operations since that is where the problem actually took place.

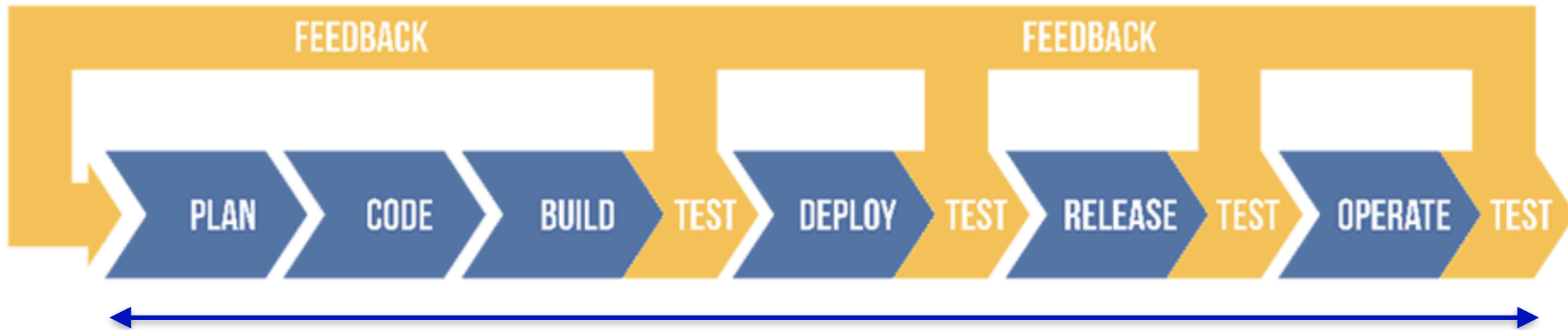
James Hamilton: On designing and deploying internet-scale services (LISA '07)

DevOps Definition

“DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.” [Wikipedia]

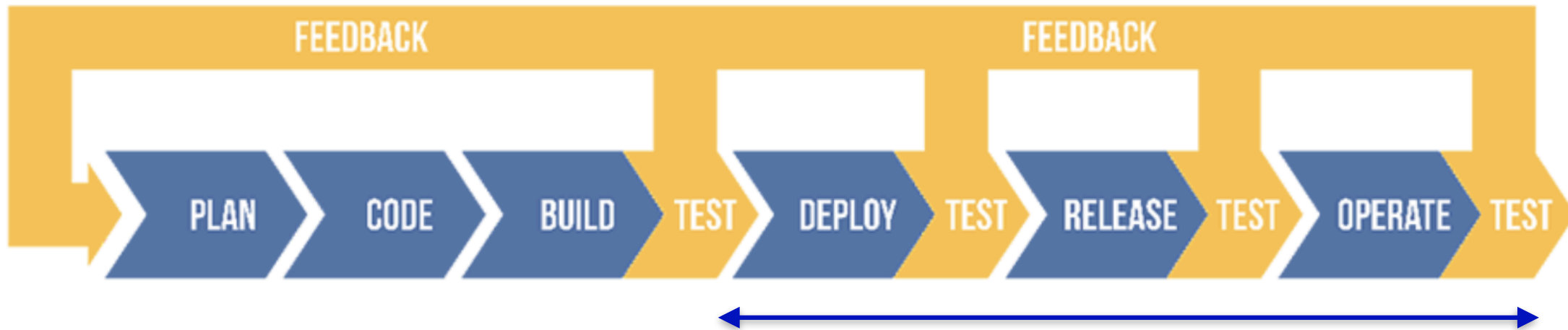


DevOps



“DevOps” (in theory)

DevOps



“DevOps”
(how it’s often understood in practice)

Common Tenets of DevOps Organisations

System admins get displaced by “DevOps engineers”

Sometimes embedded in software teams

High degree of automation

Testing, provisioning, configuration, deployment, alerting

Small, frequent releases

Continuous delivery / deployment

Developers have “on call” duty

Either rotating or after one of “their” features is deployed

Automation

Basic idea:

Treat operations tasks as software problems waiting to be automated

Provisioning servers -> **Infrastructure-as-Code** (scriptable infra)

Incident management -> **Monitoring + Root Cause Analysis**

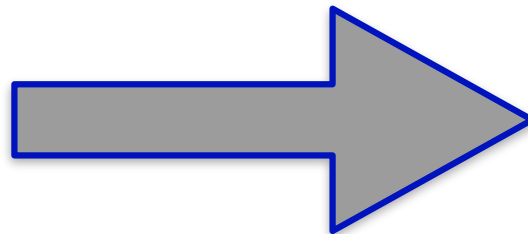
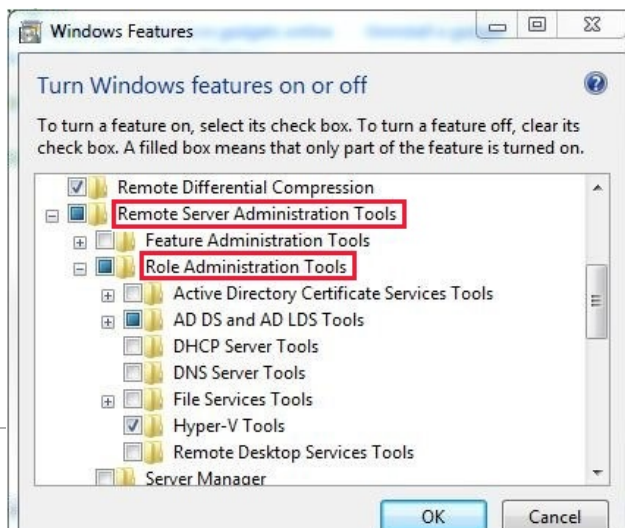
Automation

Basic idea:

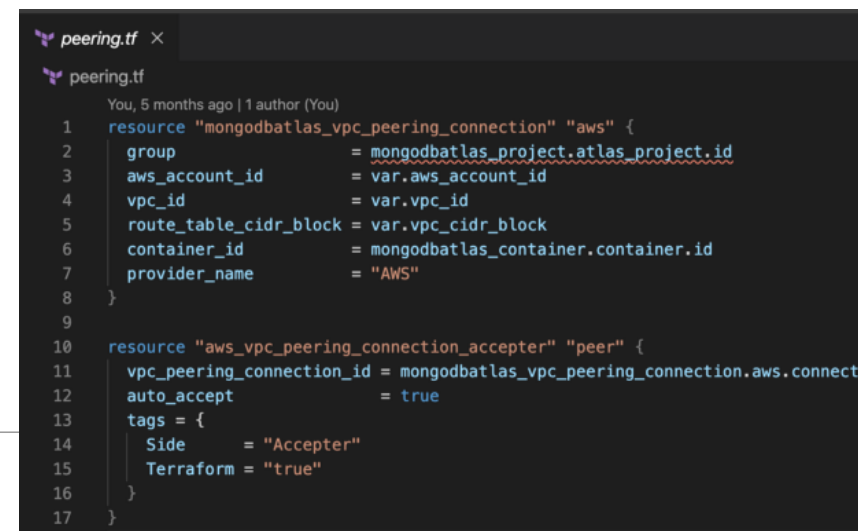
Treat operations tasks as software problems waiting to be automated

Provisioning servers -> **Infrastructure-as-Code** (scriptable infra)

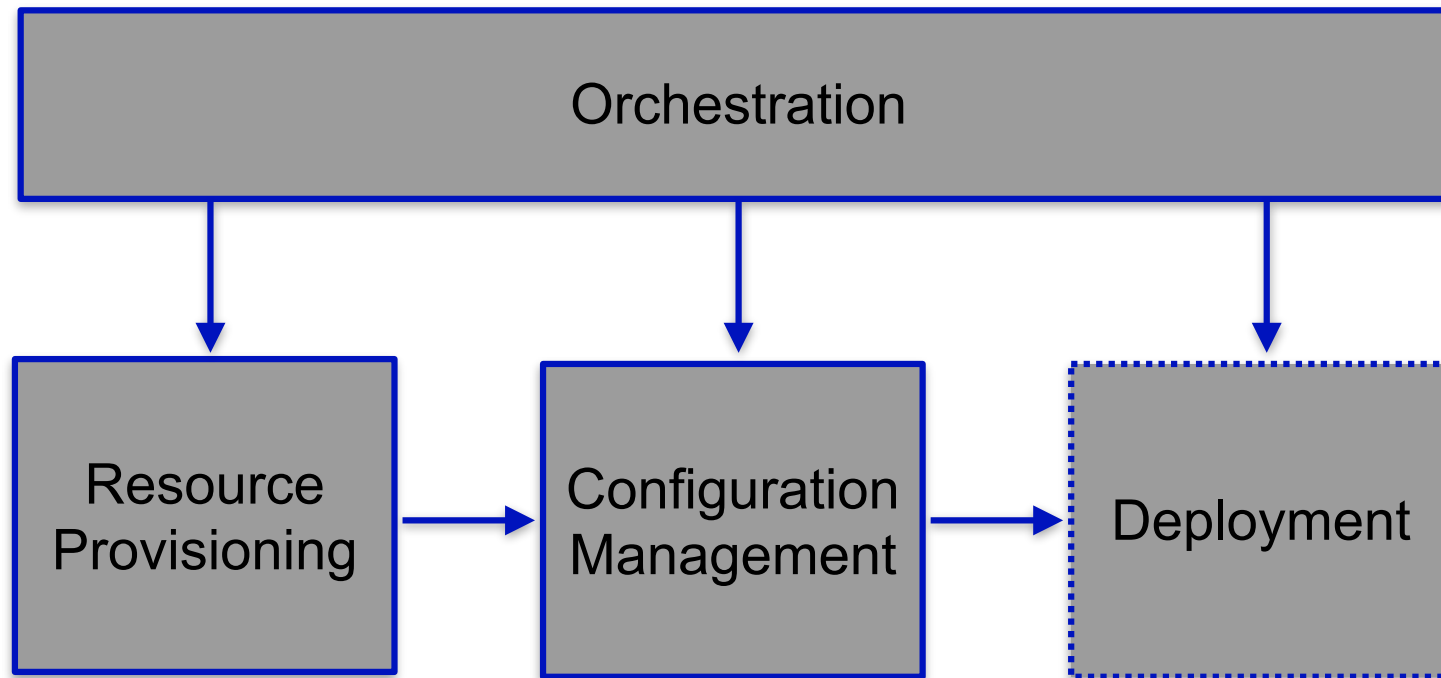
Incident management -> **Monitoring + Root Cause Analysis**



Chalmers

A screenshot of a code editor showing Terraform configuration files. The top file is 'peering.tf' and the bottom file is 'peer.tf'. The 'peering.tf' file contains a resource 'mongodbatlas_vpc_peering_connection' of type 'aws' with various attributes like group, aws_account_id, vpc_id, route_table_cidr_block, container_id, and provider_name. The 'peer.tf' file contains a resource 'aws_vpc_peering_connection_accepter' of type 'peer' with attributes like vpc_peering_connection_id, auto_accept, and tags. The code is written in HCL (HashiCorp Configuration Language) syntax.

IaC General Process



IaC General Process

Resource provisioning:

Dynamically acquire resources as required

E.g., create new virtual machines, or reserve machines in a pool

Configuration management:

Set up necessary software stack on new resources

Configure software stack as required (e.g., create users, assign IP address, etc.)

E.g., install correct OS, install MongoDB server software, create service account

Deployment:

Deploy new version of software via integration with build system (continuous deployment)

Orchestration:

If the application consists of >1 service, an orchestration plane is required to handle dependencies between services

E.g., provision and start database before backend, configure correct dynamic IP address in backend

Types of IaC Tools

“Imperative” IaC (sometimes: procedural IaC)

IaC script contains all commands necessary to provision the service

IaC tool runs the commands in order (and hopes for the best)

“Declarative” IaC

IaC scripts define the **state that a service should be in** after deployment

(Rather than how to actually get to that state)

Provisioning is run in a loop until correct state is reached (or until the tool gives up)

Examples

“Imperative” IaC

“Start 10 new AWS EC2 virtual machines” (using Ansible)

```
- ec2:
  count: 10
  image: ami-v1
  instance_type: t2.micro
```

“Declarative” IaC

“Make sure that we have 10 AWS EC2 virtual machines” (using Terraform)

```
resource "aws_instance" "example" {
  count      = 10
  ami       = "ami-v1"
  instance_type = "t2.micro"
}
```

<https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c>

Docker

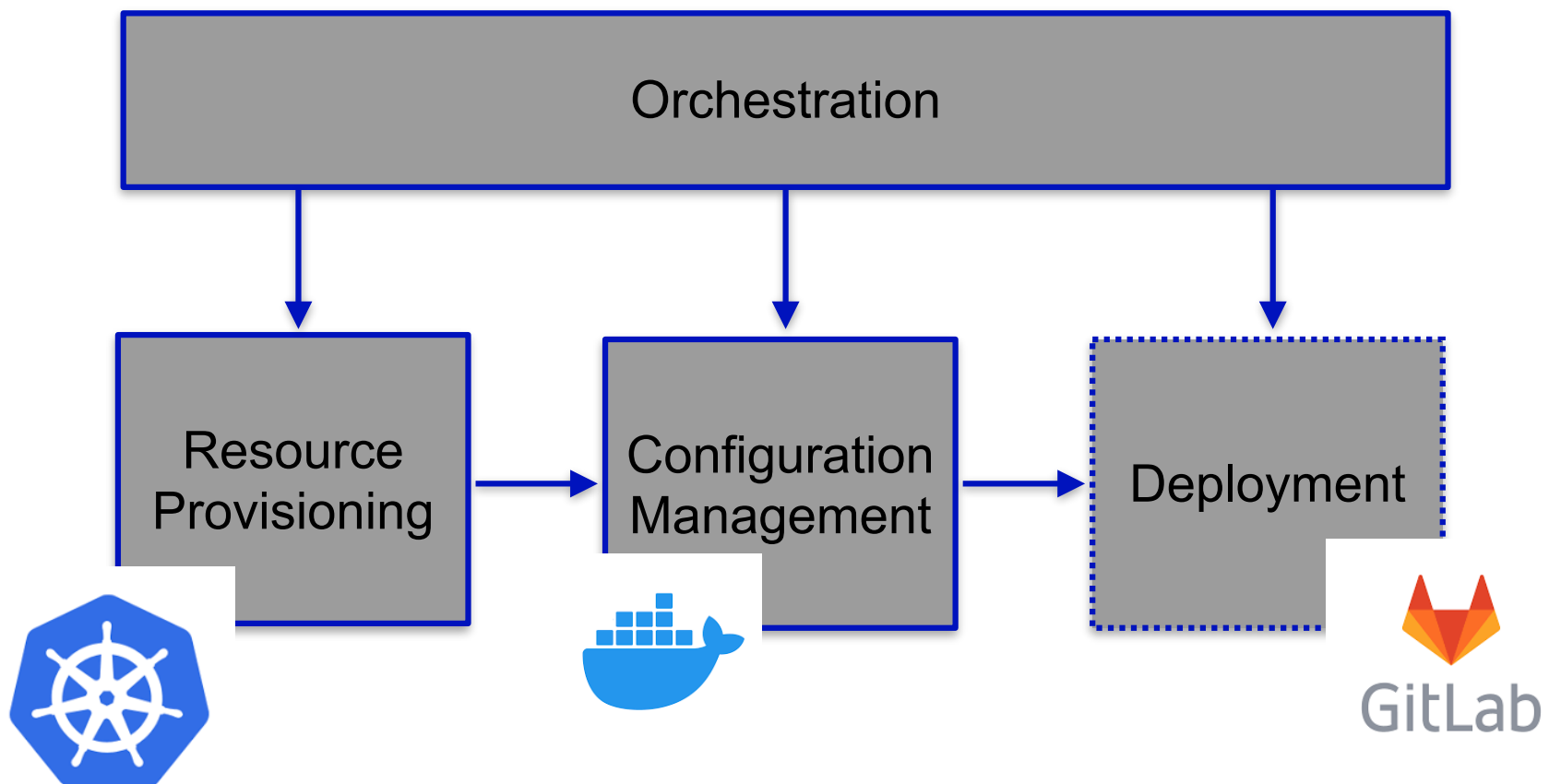
```
1 FROM alpine
2 RUN apk add nodejs npm
3
4 WORKDIR /node_app
5 COPY ./ /node_app
6
7 RUN npm install express
8
9 EXPOSE 5000
10
11 CMD [ "node", "express.js" ]
```

Imperative
Configuration Management

Kubernetes

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  rules:
    - host: nginx.kubernetesfinland.com
      http:
        paths:
          - path: /
            backend:
              Service Reference
              serviceName: nginx
              servicePort: 80
```

Declarative
Resource Provisioning + Orchestration



GitOps

Definition

*“GitOps is an operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD, and applies them to infrastructure automation.”
[GitLab, <https://about.gitlab.com/topics/gitops/>]*

Basically: apply good SE principles to IaC code

In Practice

Use

- Version control
- Issue tracking
- Code review
- Testing (as far as possible - often difficult)
- CI / CD

For your infrastructure code (scripts and configuration files)

Git becomes single source of truth about the state of the operational environment.

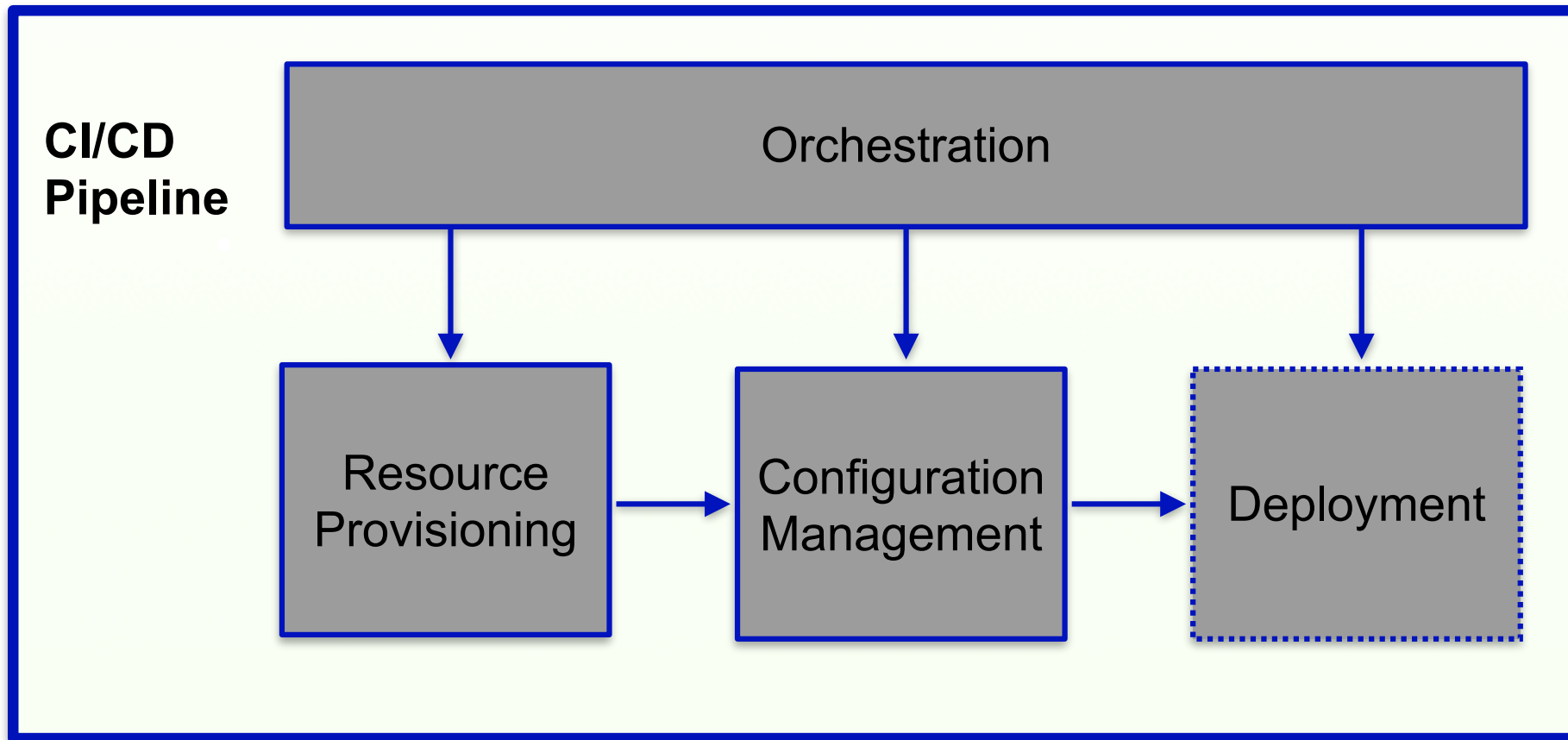
CD and GitOps

Question - what does “deploying” configuration code even mean?

Normally:

Applying configuration changes automatically as soon as they are merged to the main branch

- > Consequence: config in main branch == state of production
- > CI/CD pipeline becomes the “orchestrator” of infra automation




GitLab Example - Configuring your Kubernetes Cluster With GitLab

- (1) Connect cluster to GitLab
- (2) Create a **cluster management project** (special type of project that holds Helm `values.yaml` files)
- (3) Register the cluster management project as responsible to manage your Kubernetes cluster

Now every commit to the cluster management project applies every change to the cluster


 **passed** Pipeline #41763 triggered 1 week ago by  philipp.leitner

Updated certManager

 1 job for **master** in 21 seconds (queued for 7 seconds)

 **latest**

 **1a4ddd90** 

 No related merge requests found.

Pipeline Needs Jobs **1** Tests **0**

Deploy

 **apply**

