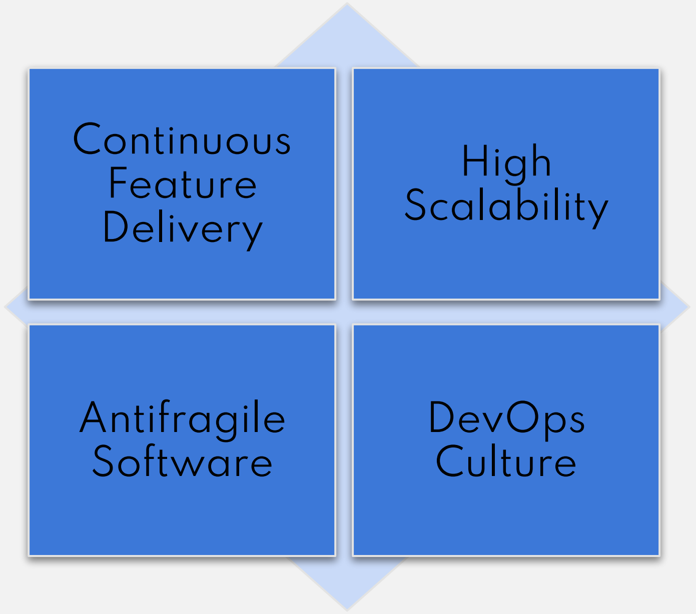
qaware.de

QA|WARE
SOFTWARE ENGINEERING

DevOps & Continuous Delivery

Lukas Buchner
lukas.buchner@qaware.de

Drivers of Cloud-native applications



What is DevOps?

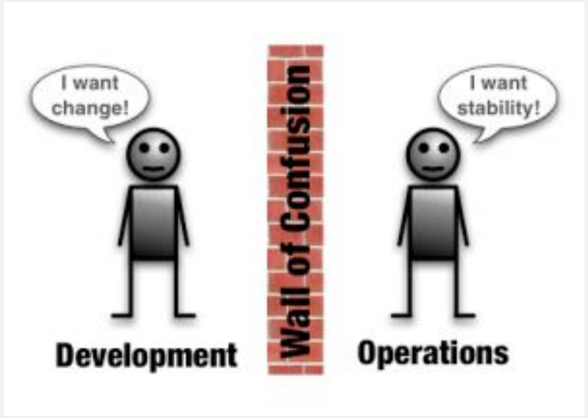
DevOps is a methodology in the software development and IT industry. Used as a set of practices and tools, DevOps integrates and automates the work of software development (*Dev*) and IT operations (*Ops*) as a means for improving and shortening the systems development life cycle.[1] DevOps is complementary to agile software development; several DevOps aspects came from the *agile*approach.

… the term is used in multiple contexts. At its most successful, DevOps is a combination of specific practices, culture change, and tools.[8]

https://en.wikipedia.org/wiki/DevOps

What is DevOps?

DevOps is the improved integration of **development** and **operations** through greater **collaboration** and **automation**, aiming to **deploy** changes to production more quickly and keep the **MTTR** (Mean Time to Recovery) low. **DevOps** is therefore a **culture**.

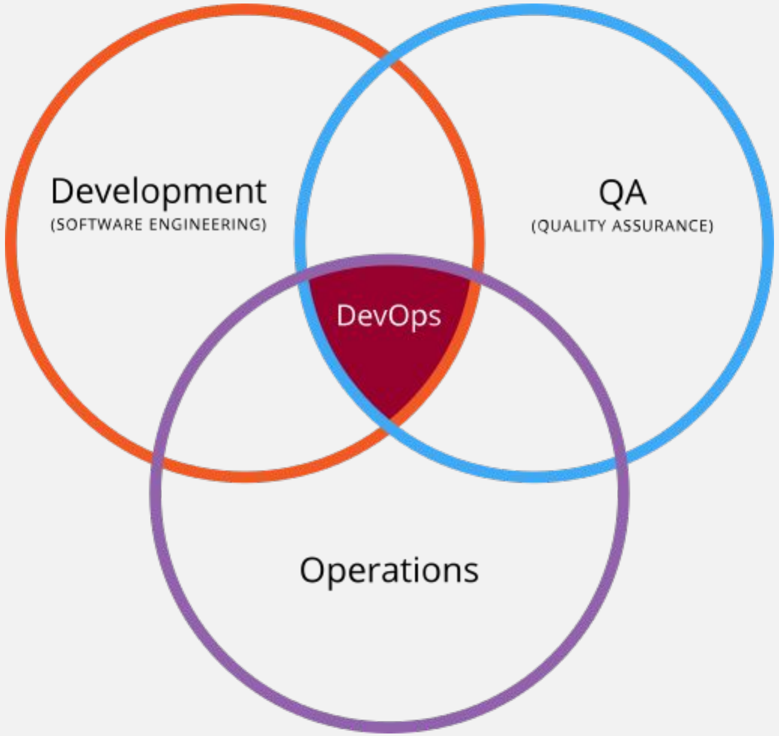


MVP + Feature Stream
Per Feature:

- Minimal manual post-commit effort until PROD
- Diagnosability of a feature's success
- Ability to disable/roll back the feature

DevOps connects DEVelopment, OPerations and Quality Assurance

QA|WARE

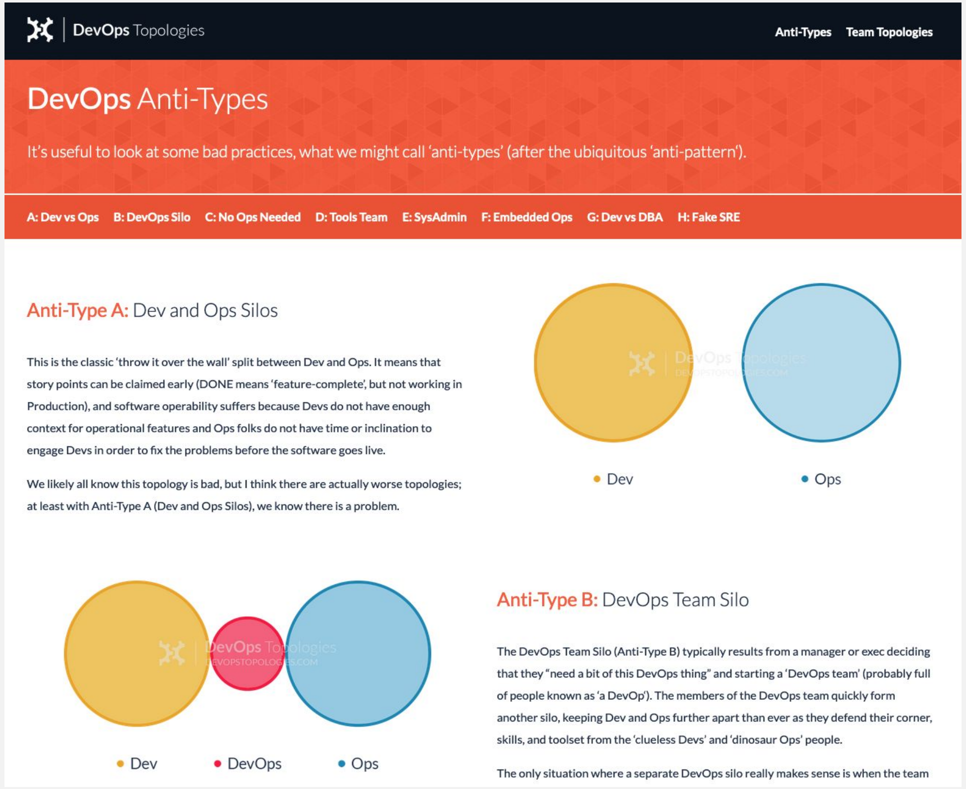


„Venn diagram showing DevOps“ by Rajiv.Pant/Wylve; Creative Commons 3.0

DevOps Topologies

Not everything labeled as DevOps truly embodies DevOps.

Numerous anti-patterns and types can be found at: https://web.devopstopologies.com/



DevOps Topologies

Anti-Types Team Topologies

DevOps Anti-Types

It's useful to look at some bad practices, what we might call 'anti-types' (after the ubiquitous 'anti-pattern').

A: Dev vs Ops B: DevOps Silo C: No Ops Needed D: Tools Team E: SysAdmin F: Embedded Ops G: Dev vs DBA H: Fake SRE

Anti-Type A: Dev and Ops Silos

This is the classic 'throw it over the wall' split between Dev and Ops. It means that story points can be claimed early (DONE means 'feature-complete', but not working in Production), and software operability suffers because Devs do not have enough context for operational features and Ops folks do not have time or inclination to engage Devs in order to fix the problems before the software goes live.

We likely all know this topology is bad, but I think there are actually worse topologies; at least with Anti-Type A (Dev and Ops Silos), we know there is a problem.

Anti-Type B: DevOps Team Silo

The DevOps Team Silo (Anti-Type B) typically results from a manager or exec deciding that they "need a bit of this DevOps thing" and starting a 'DevOps team' (probably full of people known as 'a DevOp'). The members of the DevOps team quickly form another silo, keeping Dev and Ops further apart than ever as they defend their corner, skills, and toolset from the 'clueless Devs' and 'dinosaur Ops' people.

The only situation where a separate DevOps silo really makes sense is when the team

Image based on work at devopstopologies.com - licensed under CC BY-SA.

Pipelines

Continuous X

Continuous Integration (CI)

- All changes are immediately integrated into the current development branch and tested.
- This ensures continuous testing of compatibility between changes.

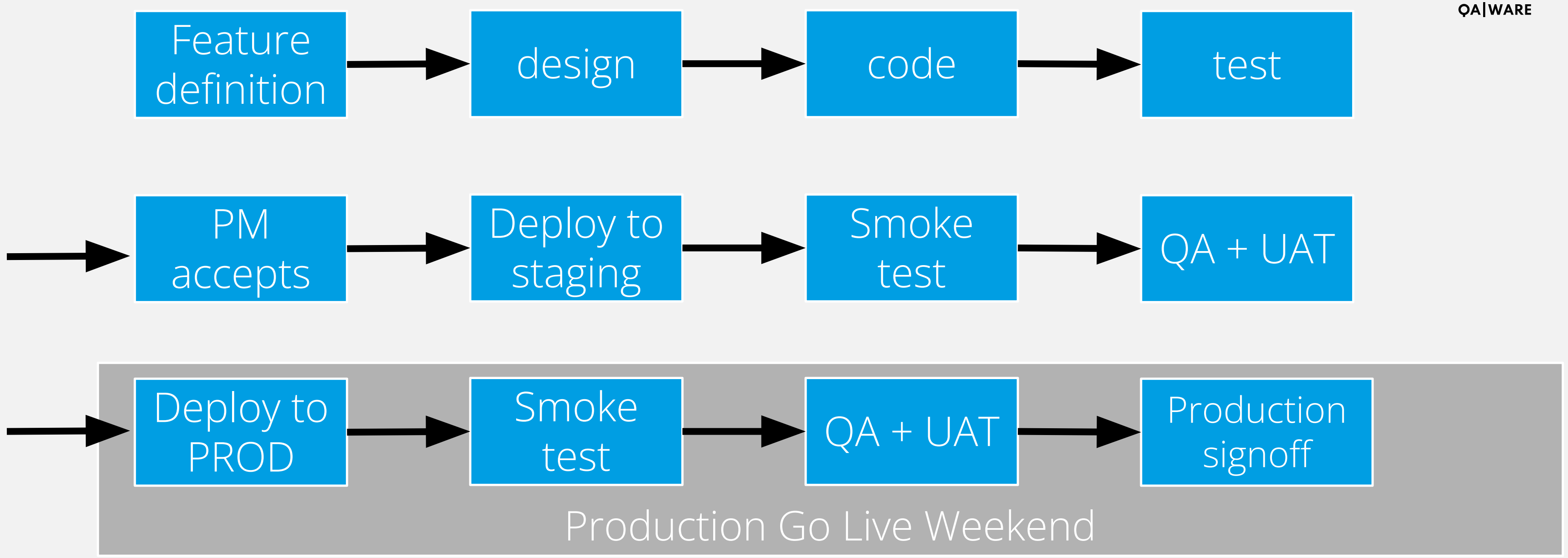
Continuous Delivery (CD)

- The code *can* be deployed at any time.
- However, it does not have to be deployed immediately.
- This means the code must (ideally) always build, be tested, and debugged when necessary.

Continuous Deployment

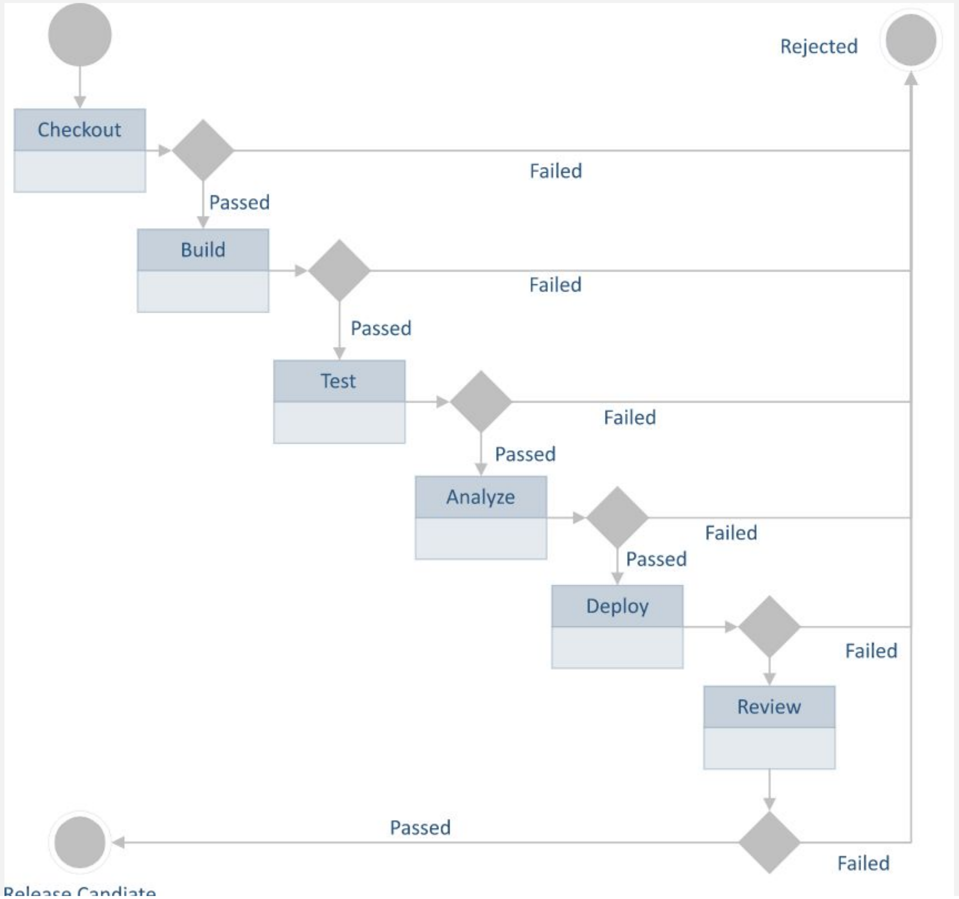
- Every stable change is deployed to production.
- Part of the quality testing takes place directly in production.
- The ability to handle errors must be in place (e.g., Canary Release, see later).

Example: Pipeline without Continuous Delivery



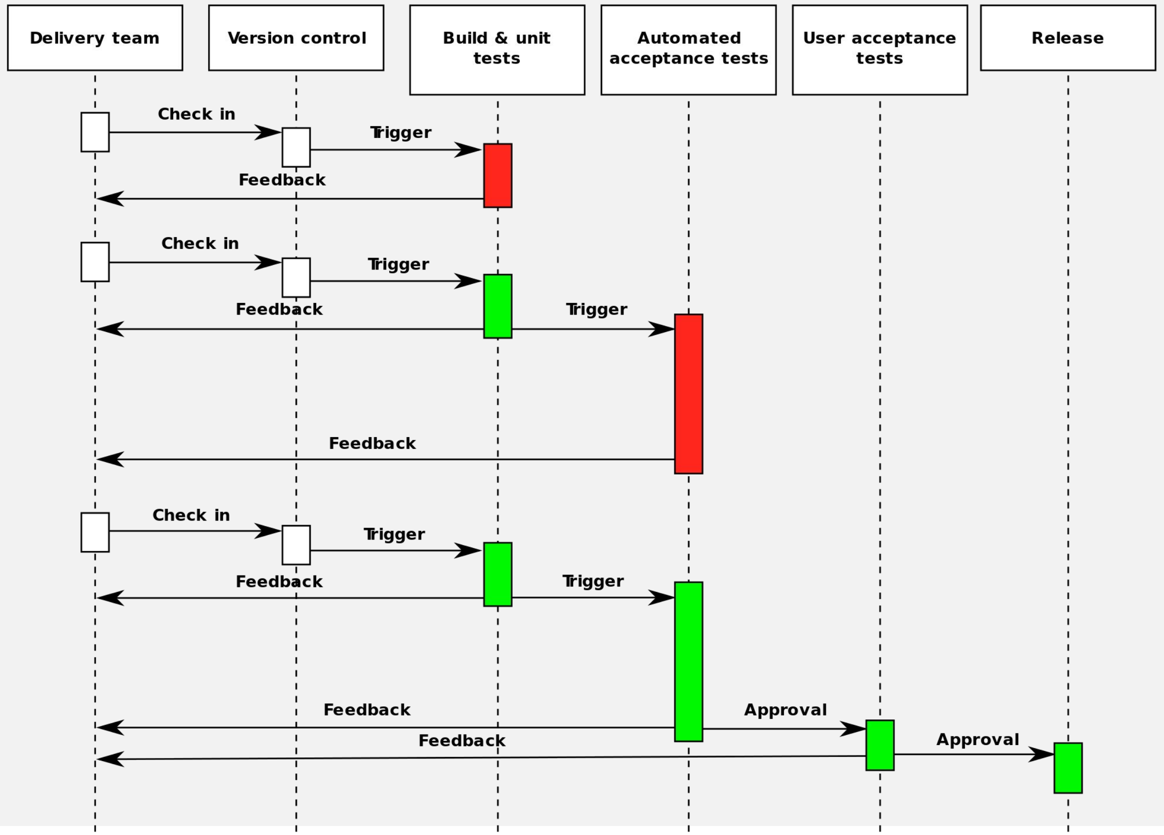
https://www.scaledagileframework.com/continuous-delivery-pipeline/

Example: Continuous-Delivery-Pipeline



Shift left philosophy- give feedback to developers as early as possible

QA|WARE



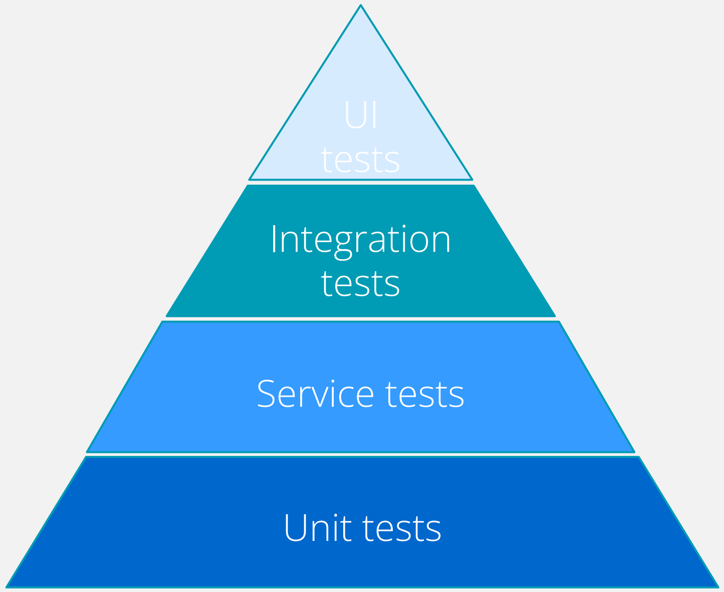
„Continuous delivery process diagramm“
by Grégoire Détrez
Creative Commons 4.0

Example of a test pyramid that aids early feedback on errors

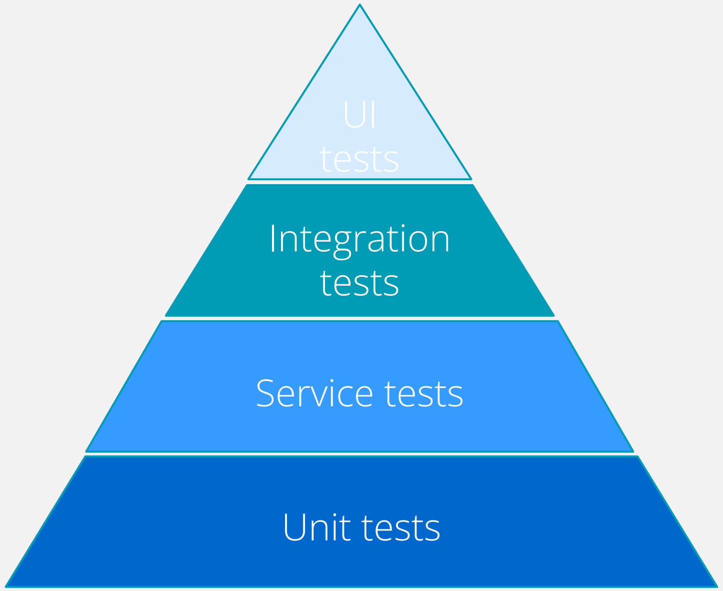
QA|WARE

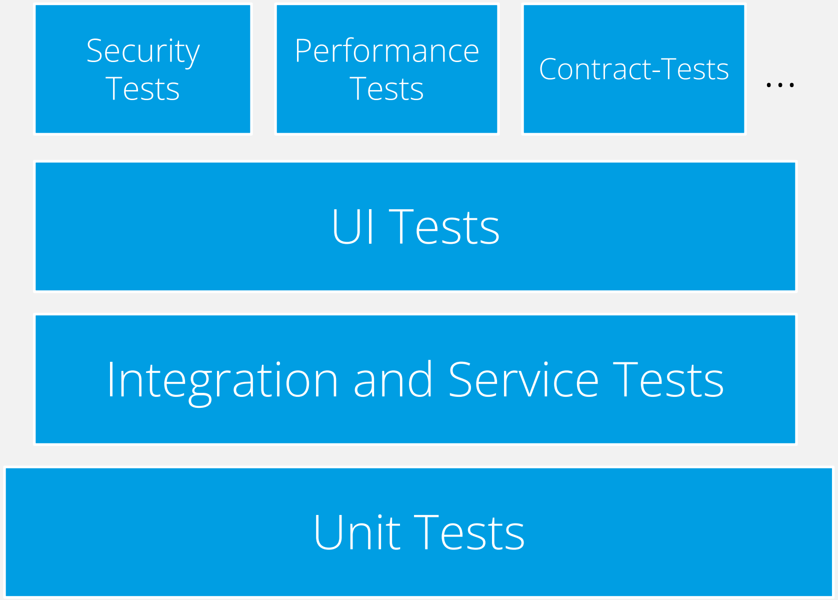
Unit Tests:
Classic unit tests (e.g., JUnit, Mockito).
Service Tests:
Tests for a single microservice, including REST controllers and client calls (e.g., JUnit, Spring MVC Tests, Wiremock).
Mocks for other microservices are necessary (e.g., using Wiremock).
Integration Tests:
Tests the integration of multiple services and their interactions (e.g., JUnit, Spring MVC Tests).
Performance Tests:
Checks for significant performance changes (e.g., Gatling).
UI Tests:
Tests UI functionality and its interaction with the backend (e.g., Selenium, Protractor).

All tests should be executed as often as possible—ideally with every commit!

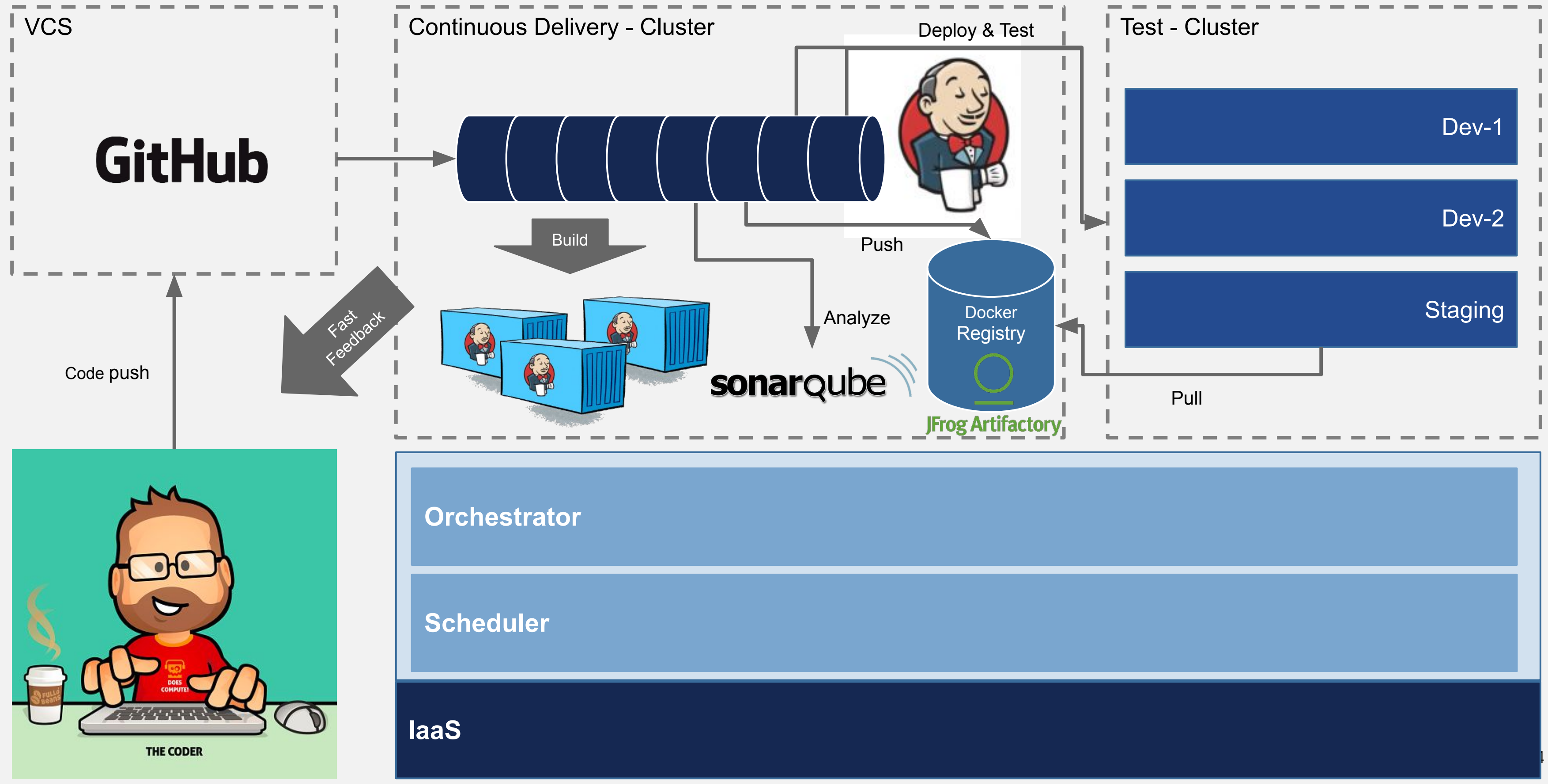


All layers should run in an automated fashion. The pyramid may also come in different shapes:



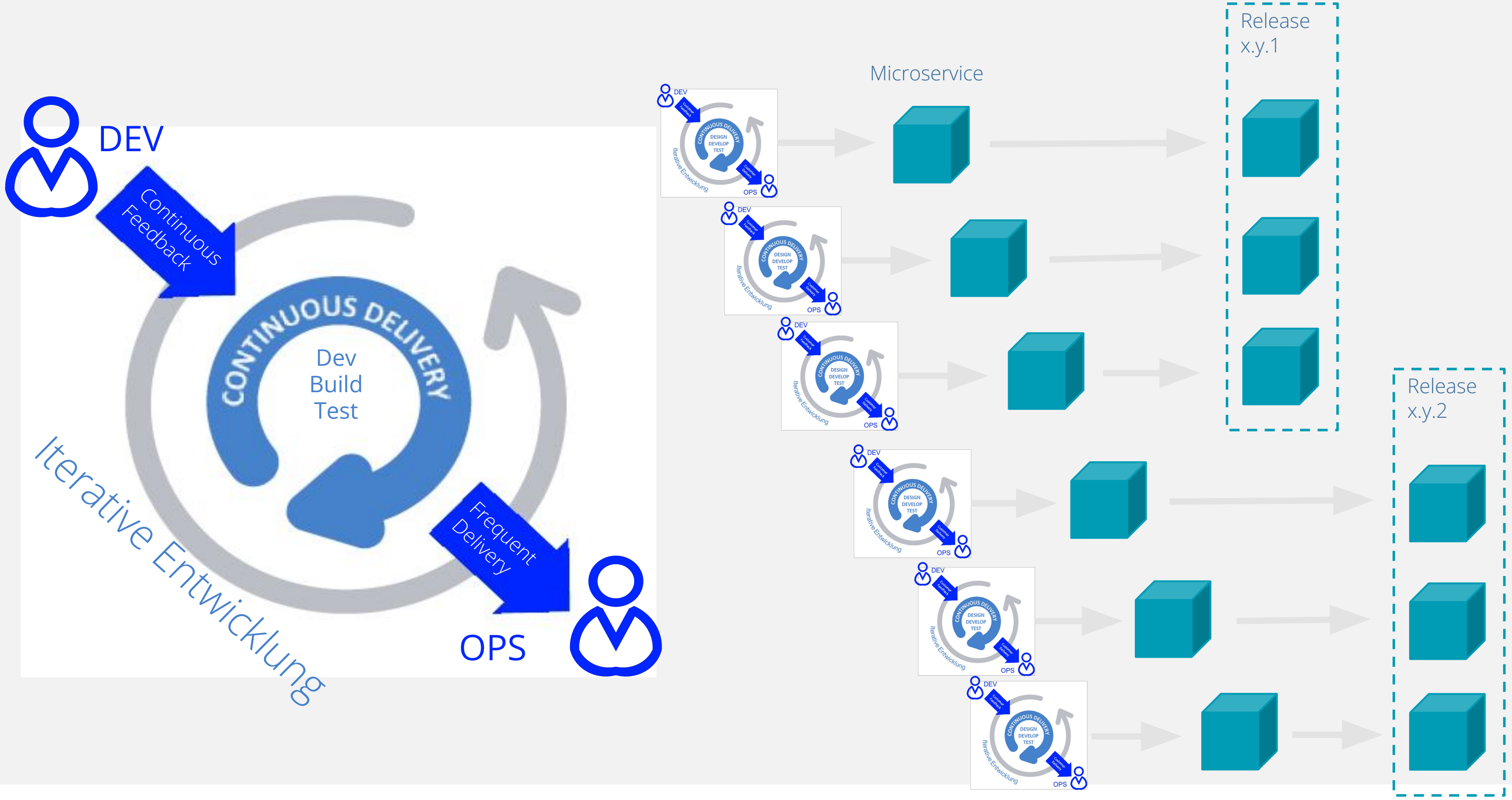


Example architecture of continuous delivery pipeline



Continuous Delivery in Microservice-Architectures:

QA|WARE



Continuous Delivery: Building Blocks

Everything as C<>de

Everything that brings the CD environment to life comes from the VCS

QA|WARE

Build-as-Code

- Maven, Gradle, etc.
- Describes how the application is built.

Test-as-Code

- Unit, component, integration, API, UI, and performance tests.
- Describes how the project is tested.

Infrastructure-as-Code

- Docker, Terraform, Vagrant, Ansible, Kubernetes deployments.
- Describes how runtime environments are set up.

Pipeline-as-Code

- Build pipeline via Jenkinsfile, Gitlab CI etc..
- Build wrapper: Describes all steps up to a runnable installation.

Self-Service und Blueprints

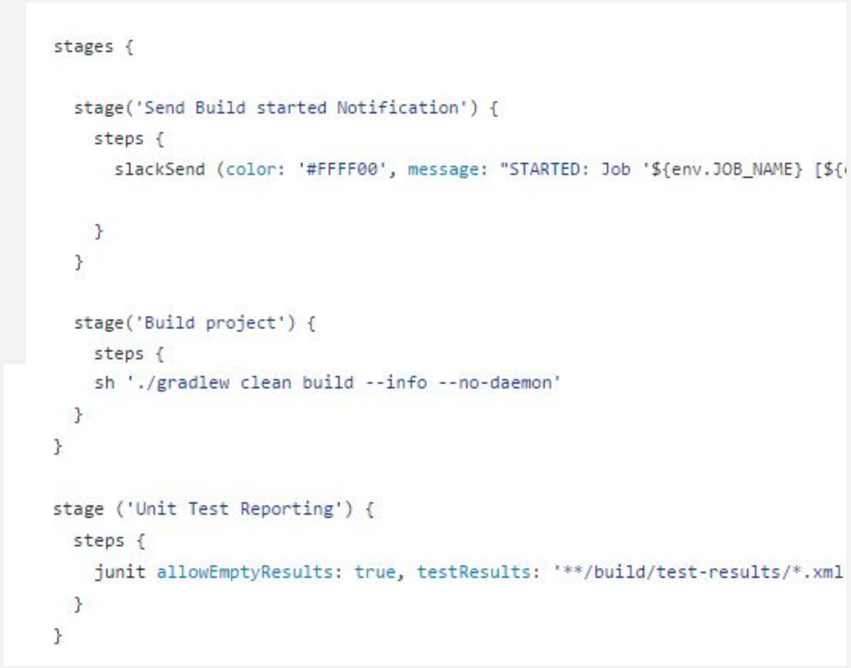
Blueprints and Templates aid the Setup

Blueprints & Templates:
The build pipelines of (micro)services within a project are often very similar.

Blueprints and templates provide a framework and implicitly establish conventions.

Examples:

- Using the Jenkins Pipeline Multibranch Plugin automatically creates a dedicated pipeline for each branch.
- Consistent integration with platform components (e.g., SonarQube, Artifactory).



```
stages {

  stage('Send Build started Notification') {
    steps {
      slackSend (color: '#FFFF00', message: "STARTED: Job '${env.JOB_NAME} [${


    }
  }

  stage('Build project') {
    steps {
    sh './gradlew clean build --info --no-daemon'
  }
}

stage ('Unit Test Reporting') {
  steps {
    junit allowEmptyResults: true, testResults: '**/build/test-results/*.xml
  }
}
```

GitOps

The Idea

QA|WARE

1 Declarative

A system managed by GitOps must have its desired state expressed declaratively.

2 Versioned and Immutable

Desired state is stored in a way that enforces immutability, versioning and retains a complete version history.

3 Pulled automatically

Software agents automatically pull the desired state declarations from the source.

4 Continuously reconciled

Software agents continuously observe actual system state and attempt to apply the desired state.

Advantages gained out of the box

QA|WARE

- Ideally allows restoring any system state in history, e.g., for easy rollbacks.
- Enforces pipelines.
- Provides transparency for changes; in the case of Git, it also shows who made changes.
- Ensures that the system state does not deviate from the desired state.

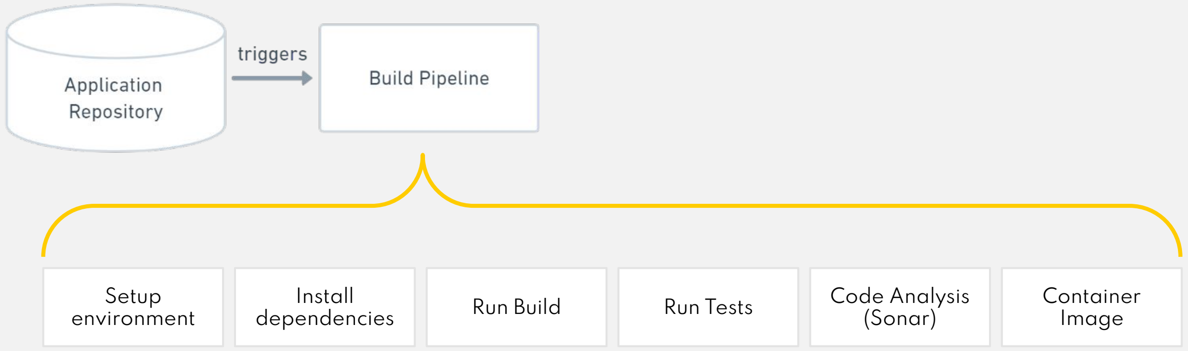
GitOps in the K8s ecosystem

QA|WARE

ArgoCD

Flux

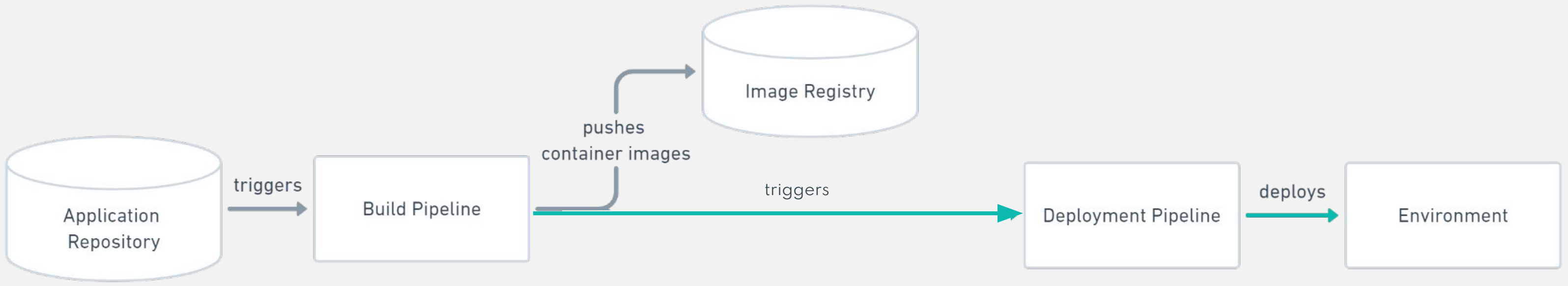
Continuous Integration



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

Continuous Integration with automated deployments

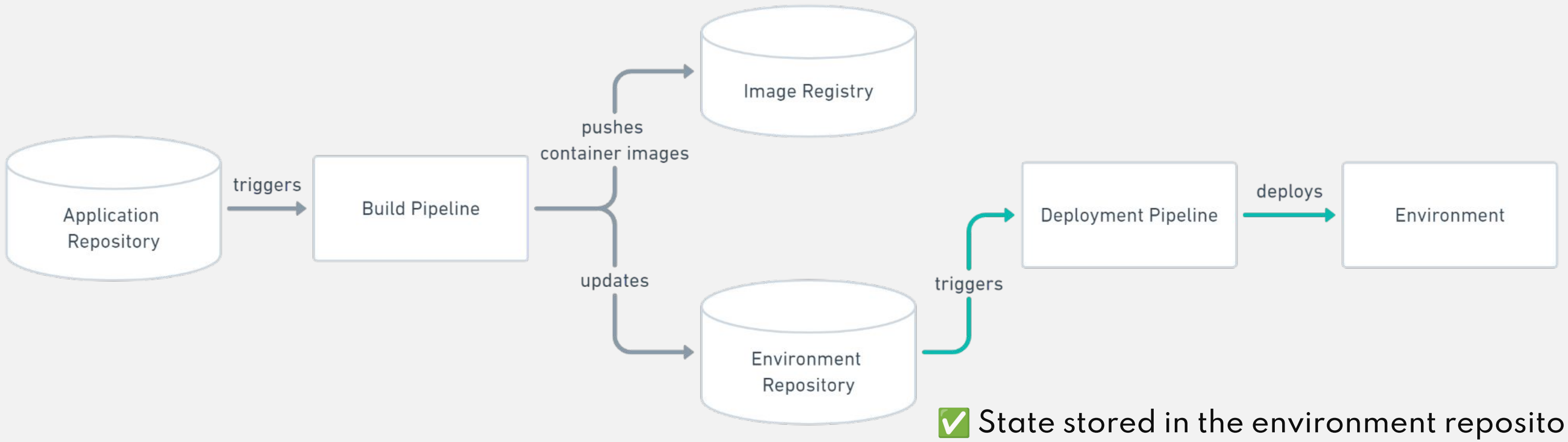
QA|WARE



✅ Automated deployments
❌ Deployment pipeline requires extensive permissions
❌ Execution is often not idempotent
❌ Cluster state is not fixed

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

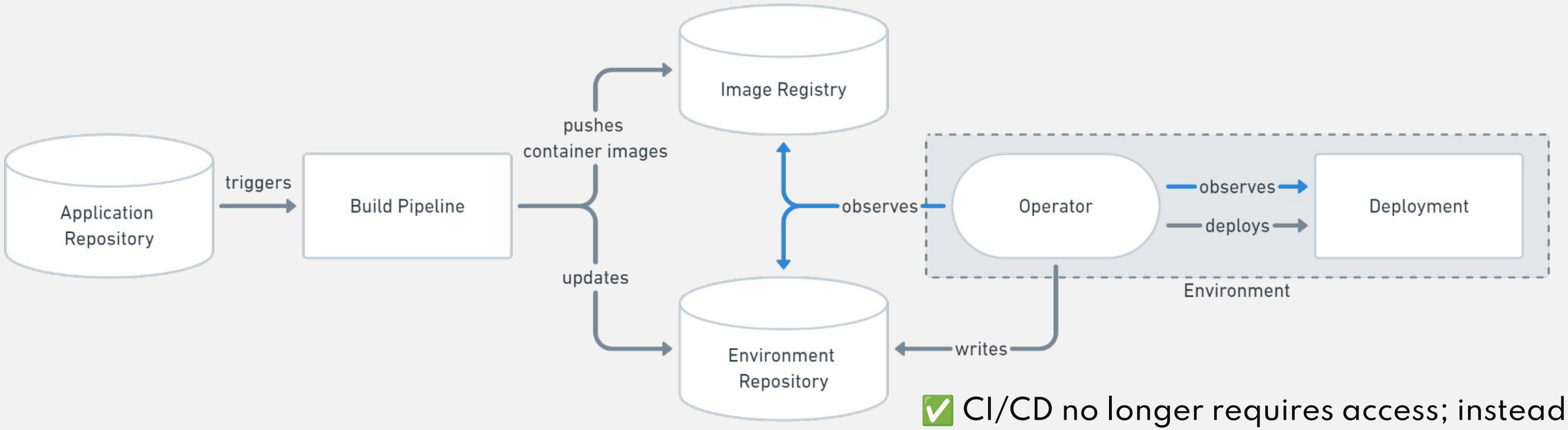
Push-based deployments with GitOps



✅ State stored in the environment repository and can be restored
❌ Deployment pipeline requires extensive permissions
❌ Deviations in the cluster outside of a deployment are not detected or corrected
❌ Deleting resources is difficult

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

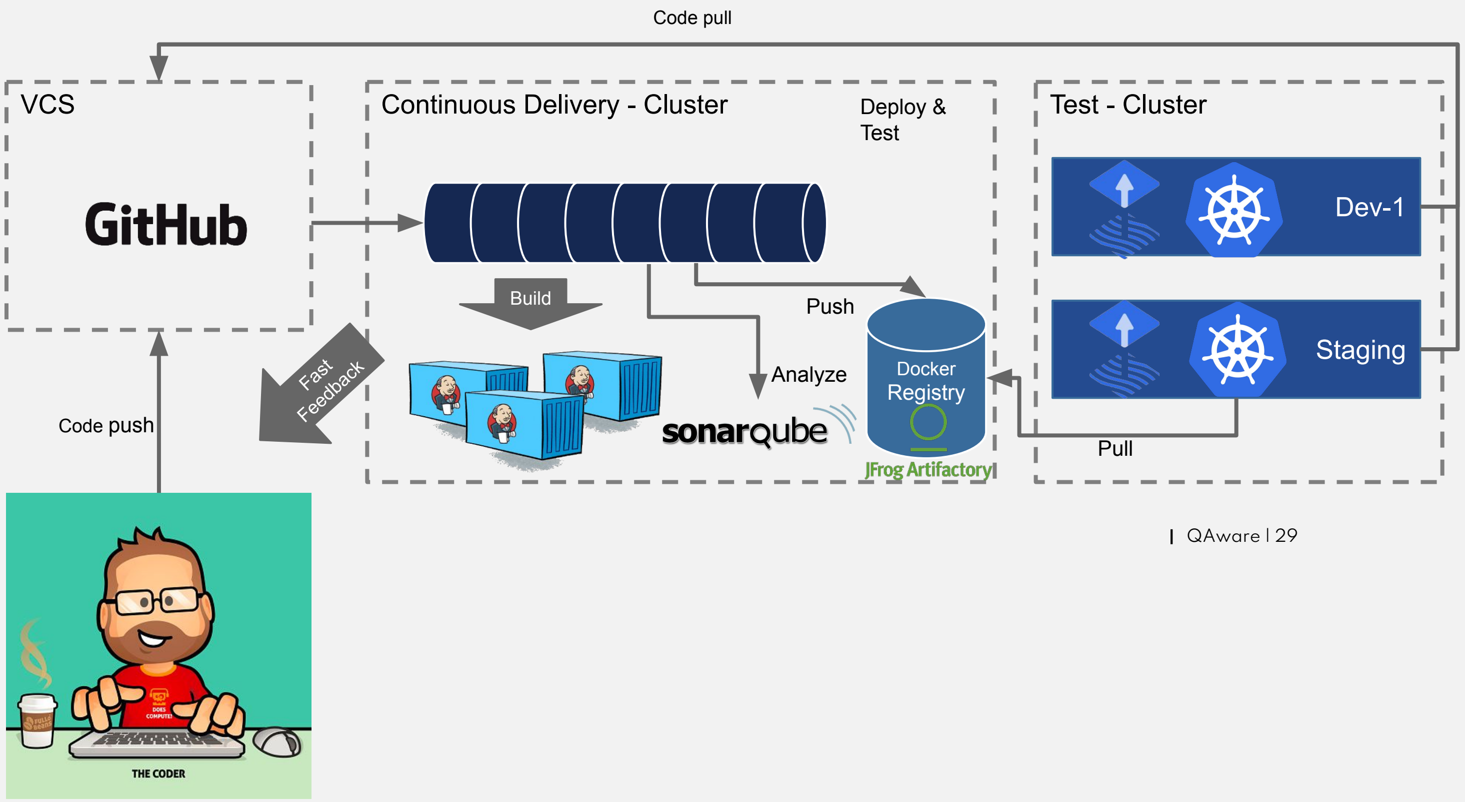
Pull-based deployments mit GitOps



✅ CI/CD no longer requires access; instead, the operator accesses the environment repo and registry.
✅ The operator attempts to establish the desired cluster state, including resource deletion.
❌ More complex

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

GitOps with Flux



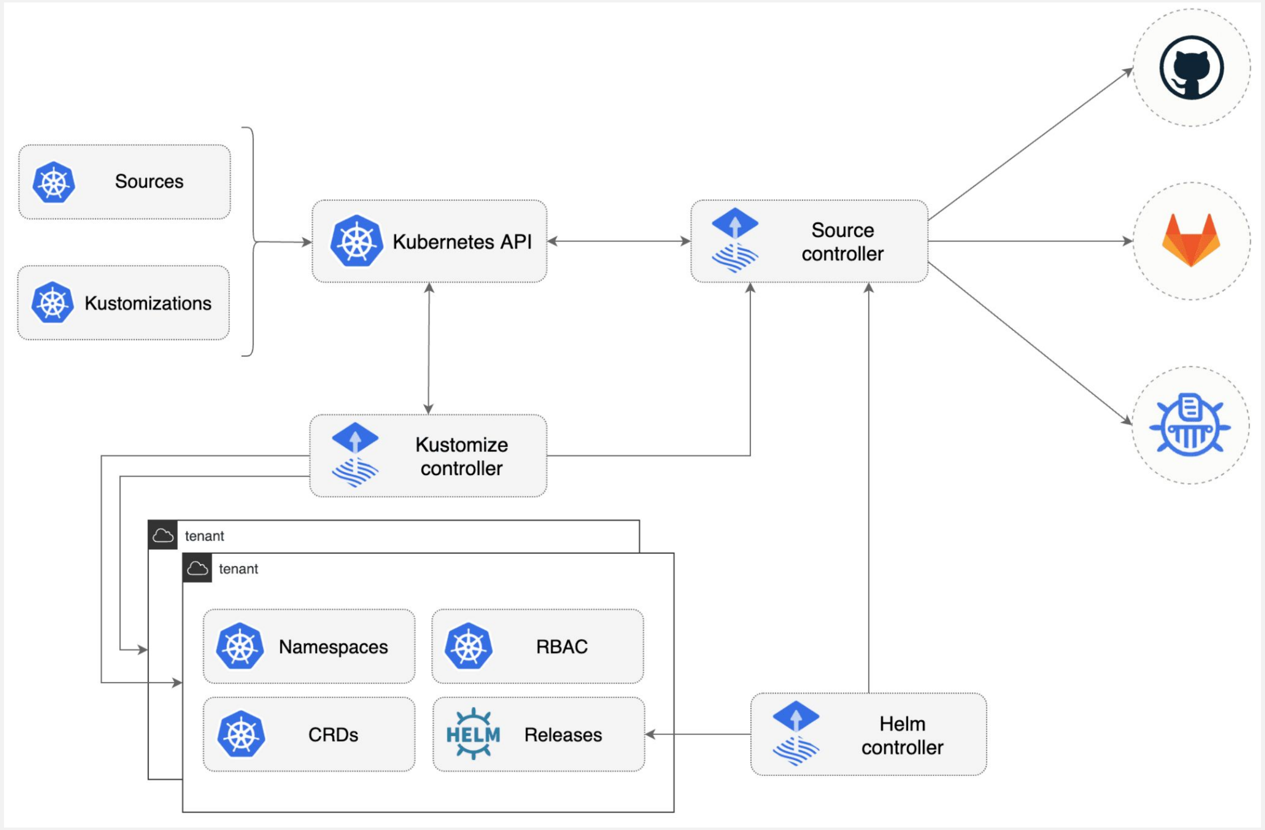
Differences in the architecture

CI/CD pipeline no longer requires cluster access
=> Improved security, as no high-privilege credentials are needed for the pipeline.

Clusters now actively pull source repositories that contain the desired state.

Clusters reconcile their state, meaning the current state is compared with the desired state, and the cluster attempts to reach the desired state.
=> Clusters automatically converge toward the desired state.

Flux in detail



https://fluxcd.io/flux/components/

Flux in detail

Source Controller

- Connects to various sources, e.g., Git repositories, OCI repositories, etc.
- Provides the downloaded packages to other controllers.

Helm Controller

- Offers Helm integration for Flux.
- In combination with the Source Controller, Helm charts can be automatically downloaded, installed, and updated.

Kustomize Controller

- Uses the Git repositories managed by the Source Controller to execute Kustomize and apply the configured resources to the cluster.
- Tracks all resources, enabling automated garbage collection of unused resources.

Notification Controller

- Provides insights into Flux events, such as successes or errors.
- Includes integrations with chat systems.

Configure a source

```yaml
apiVersion: source.toolkit.fluxcd.io/v1

kind: GitRepository

metadata:

  name: podinfo

  namespace: default

spec:

  interval: 5m0s

  url: https://github.com/stefanprodan/podinfo

  ref:

    branch: master
```

- Configures a GitRepository as a source in the Source Controller.
- Clones the master branch from the URL and makes it available to other controllers as a tar.gz file.
- Polls the URL every 5 minutes to check for changes.

Configure a kustomization

```yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: podinfo
  namespace: default
spec:
  interval: 10m
  targetNamespace: default
  sourceRef:
    kind: GitRepository
    name: podinfo
  path: "./kustomize"
  prune: true
  timeout: 1m
```

Creates a Kustomization in the Kustomize Controller that references the GitRepository from the Source Controller via **sourceRef**.

The Kustomize Controller retrieves the source code from the Source Controller and uses Kustomize (a Kubernetes tool) to apply everything under the specified **path**.

Flux in action!

Deployment Strategies

Rolling Upgrades

Strategy:
Update one instance at a time, before proceeding with the next one.

Advantages:
- Easily understandable
- Easily achievable with Kubernetes
- Does not require fancy monitoring infrastructure

Disadvantages:
- Requires special care regarding database migrations as the old and new version running at the same time. If not done right, the application is flaky for users during deployment.

Blue/Green Deployments

Strategy:
For the time of the deployment the old and new software versions are running in side by side. Once ensured, that the new version meets all requirements, the load balancer switches over to the new version.

Advantages:
- Easily understandable
- Rather easy to achieve with Kubernetes
- Does not require fancy monitoring infrastructure

Disadvantages:
- Requires special care regarding database migrations as the old and new version running at the same time. If not done right, the application is flaky for users during deployment.

Canary Deployments

Strategy:
For the time of the deployment the old and new software versions are running in side by side. Gradually a part of the traffic (e.g. 5%) is routed to the new version. If there are no errors in monitoring, more traffic is shifted. The deployment is done, when 100% of traffic hits the new software version.

Advantages:

- Theoretically ideal, as bad software versions are never rolled out fully. Bad software impacts a small amount of users only

Disadvantages:

- Requires special tooling (e.g. monitoring infrastructure)
- Hard to really get right. What happens if there's no continuous traffic incoming during deployment?
- Requires special care regarding database migrations as the old and new version running at the same time. If not done right, the application is flaky for users during deployment.