



Kapitel: Continuous Delivery

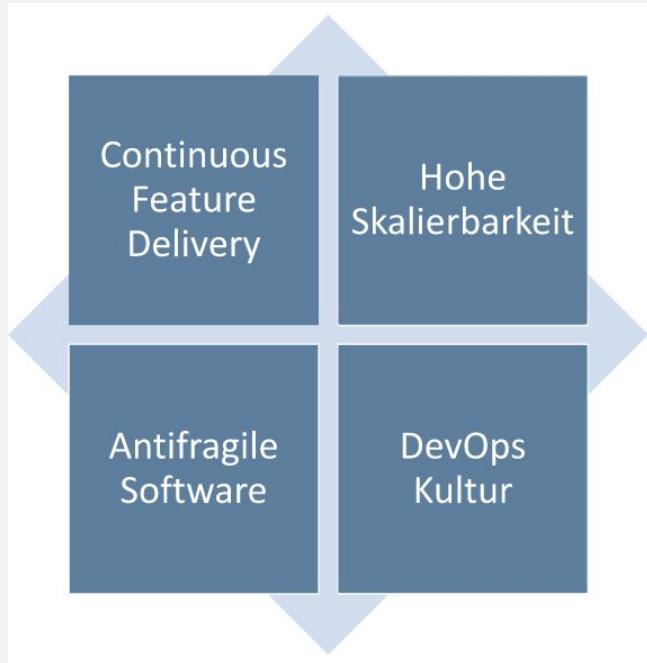
Franz Wimmer

franz.wimmer@qaware.de

Treiber für Cloud-native Anwendungen



QA|WARE



Continuous Delivery - Definition



QA|WARE

ContinuousDelivery



Martin Fowler

30 May 2013

Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.

martinfowler.com

Continuous delivery

From Wikipedia, the free encyclopedia

Continuous delivery (CD) is a [software engineering](#) approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.^[1] It aims at building, testing, and releasing software faster and more frequently. The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production. A straightforward and repeatable deployment process is important for continuous delivery.

Abgrenzung zu Continuous X



Continuous Integration (CI)

- Alle Änderungen werden sofort in den aktuellen Entwicklungsstand integriert und getestet.
- Dadurch wird kontinuierlich getestet, ob eine Änderung kompatibel mit anderen Änderungen ist.

Continuous Delivery (CD)

- Der Code *kann* zu jeder Zeit deployed werden.
- Er muss aber nicht immer deployed werden.
- D.h. der Code muss (möglichst) zu jedem Zeitpunkt bauen, getestet und ge-debugged sein.

Continuous Deployment

- Jede stabile Änderung wird in Produktion deployed.
- Ein Teil der Qualitätstests finden dadurch in Produktion statt.
 - Die Möglichkeit, mit Fehlern umzugehen, muss vorhanden sein (z.B. Canary Release, siehe später)

Kriterien für Continuous Delivery



“You’re doing continuous delivery when:

- Your software is deployable throughout its lifecycle
- Your team prioritizes keeping the software deployable over working on new features
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them
- You can perform push-button deployments of any version of the software to any environment on demand”

nach M. Fowler / Continuous Delivery working group at ThoughtWorks

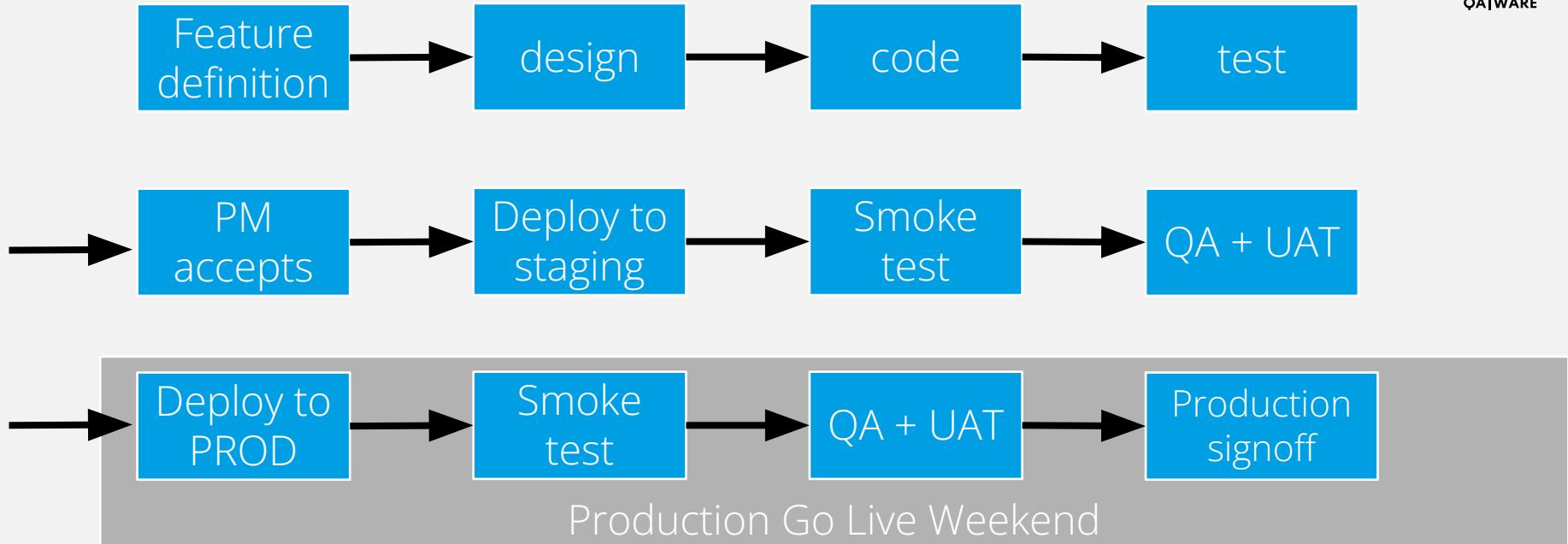
A photograph of a large industrial complex, likely a steel mill or refinery, under a clear blue sky. The foreground features massive, rusted, corrugated metal pipes supported by a network of steel beams and walkways. In the background, there are several large, cylindrical structures, possibly blast furnaces or storage tanks, and a tall, dark chimney. The lighting suggests it's either early morning or late afternoon, casting long shadows and highlighting the textures of the industrial equipment.

Pipelines

Beispiel: Pipeline ohne Continuous Delivery



QA|WARE

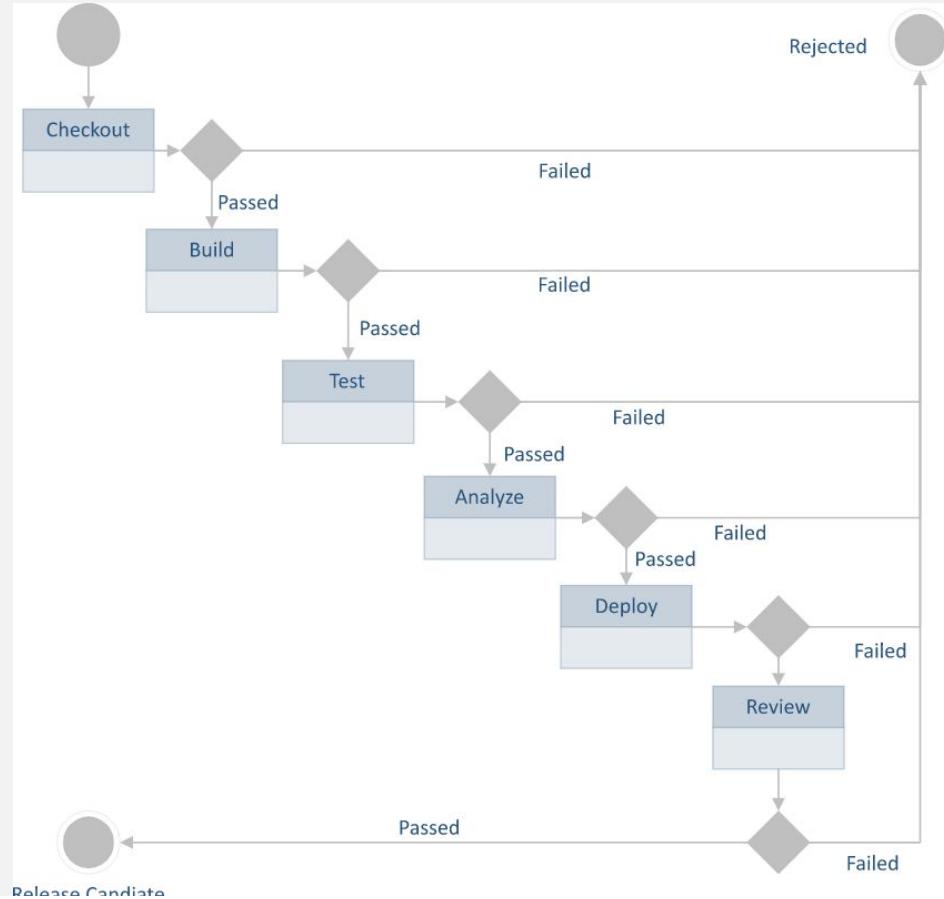


<https://www.scaledagileframework.com/continuous-delivery-pipeline/>

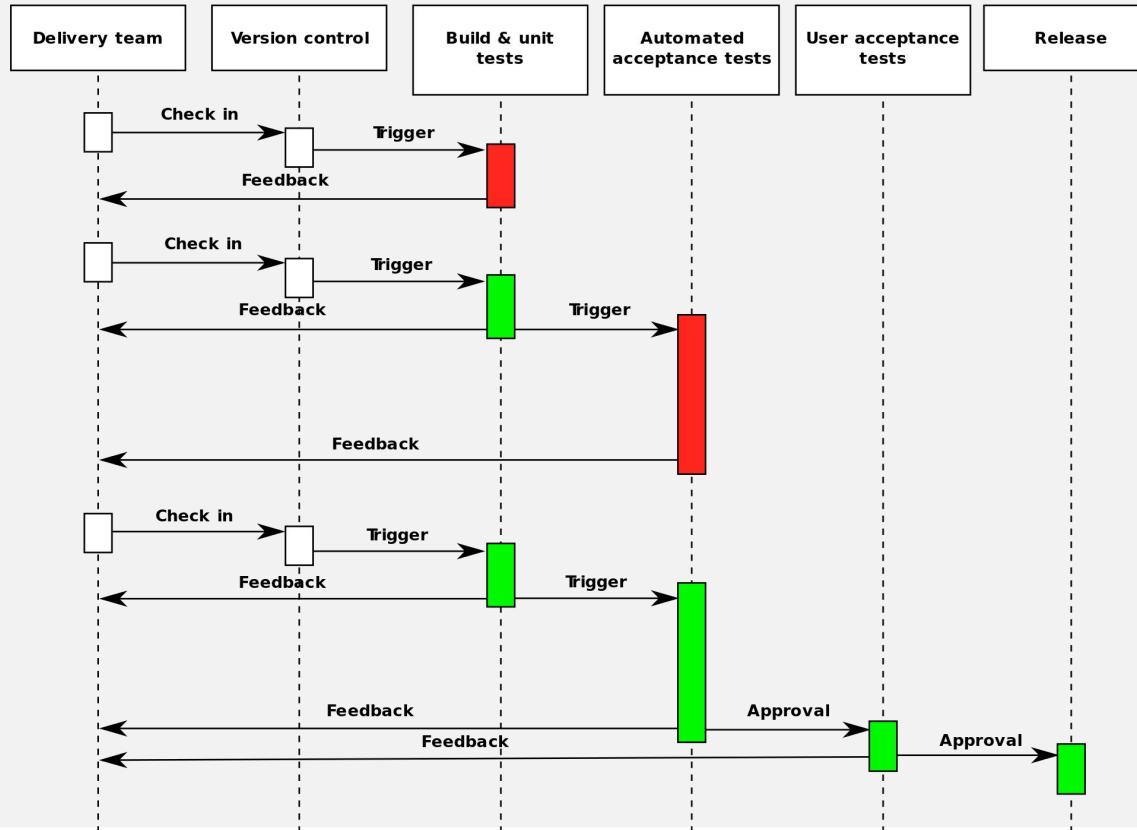
Beispiel: Continuous Delivery Pipeline



QA|WARE



Wichtig ist ein schnelles Feedback an das Entwicklerteam, damit Fehler zügig behoben werden

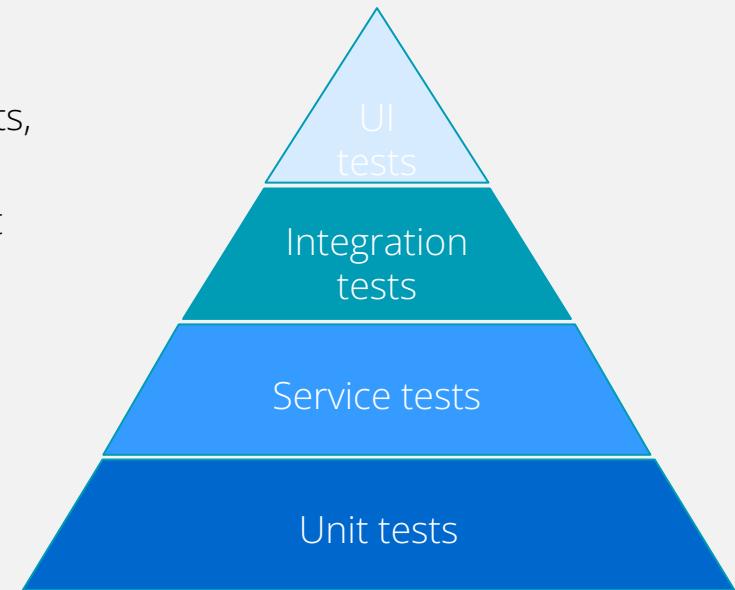


„Continuous delivery process diagramm“
 by Grégoire Détrez
 Creative Commons 4.0

Beispiel-Aufbau einer Test-Pyramide für sofortiges Feedback bei Fehlern

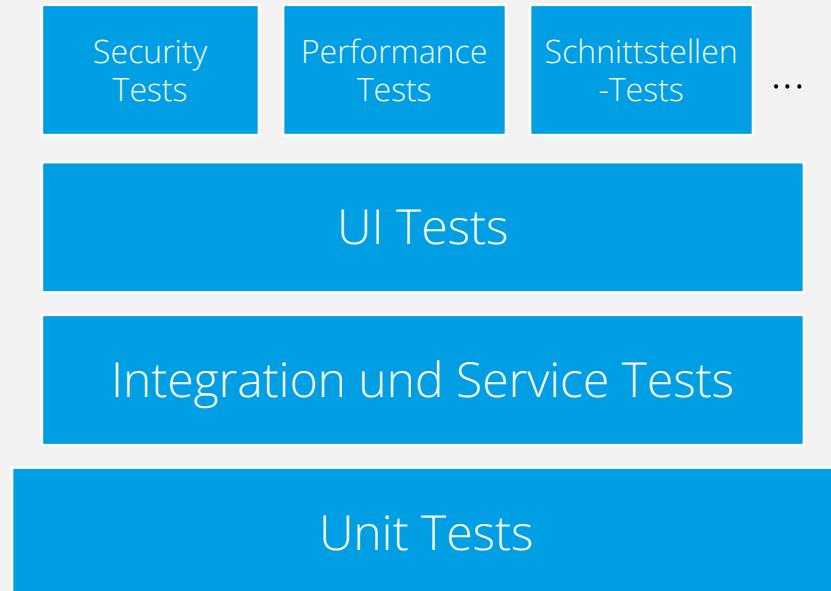
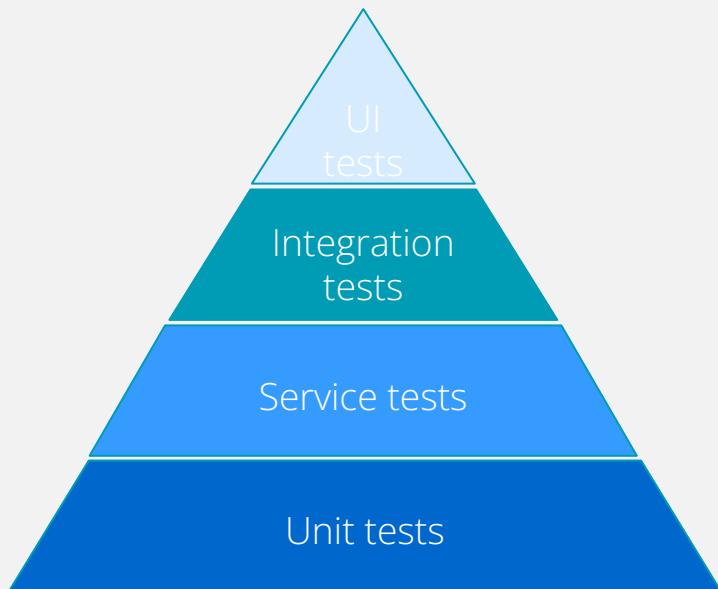


- Unit Tests: Die klassischen Unit Tests (z.B. JUnit, Mockito)
- Service Tests: Tests eines einzelnen Microservices, inkl. der REST-Controller und Client-Calls (z.B. JUnit, Spring MVC Tests, Wiremock)
 - Mocks der anderen Microservices notwendig (z.B. mit Wiremock)
- Integration Tests:
 - Testet die Integration mehrerer Services und deren Interaktion (z.B. JUnit, Spring MVC Tests)
 - Performance Tests: Testet, ob es signifikante Performance-Änderungen gibt (z.B. Gatling)
- UI-Tests: Testet die UI-Funktionalität und deren Zusammenspiel mit dem Backend (z.B. Selenium, Protractor)

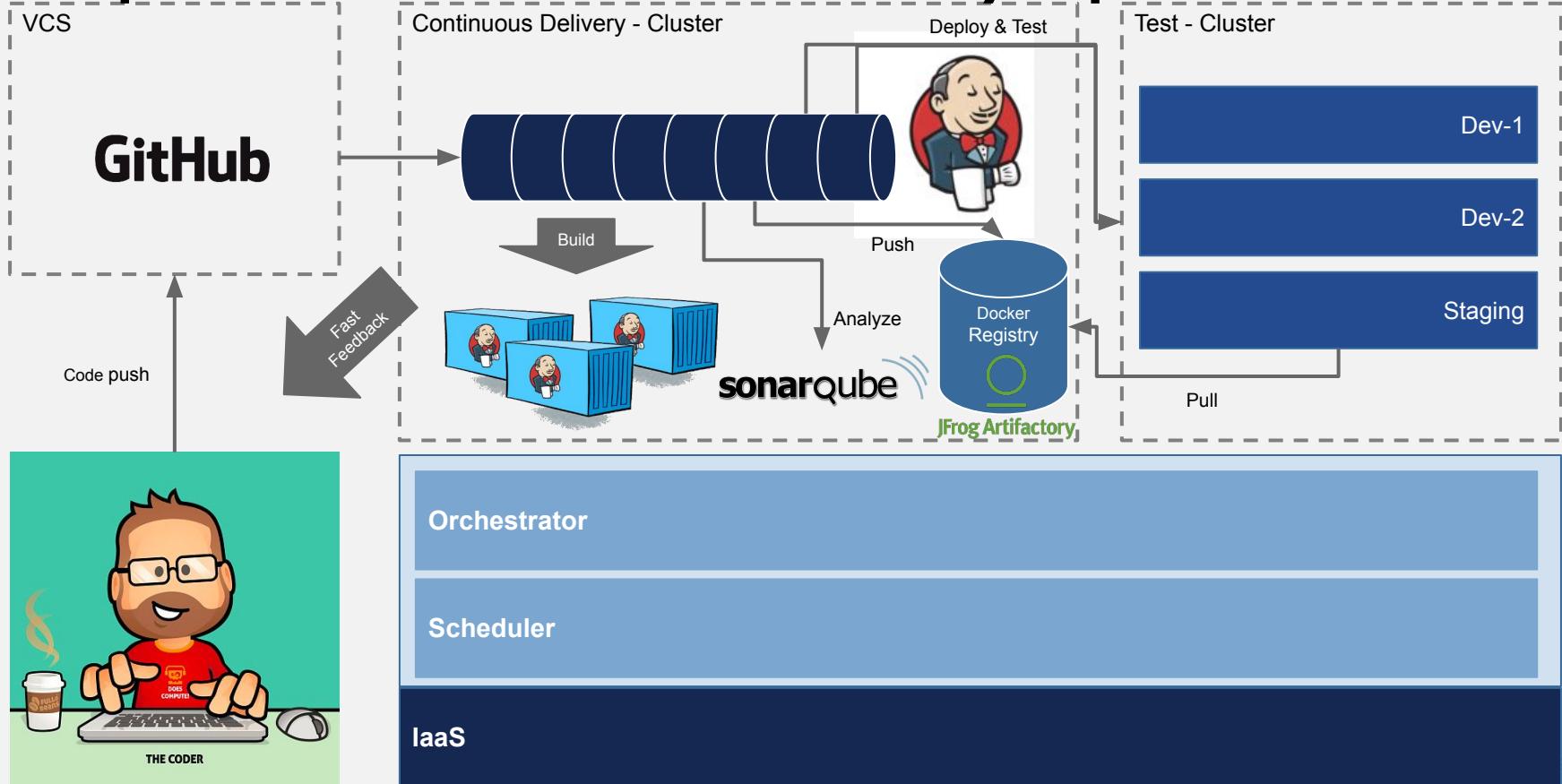


Alle Tests sollten so oft wie möglich ausgeführt werden.
Idealerweise bei jedem Commit!

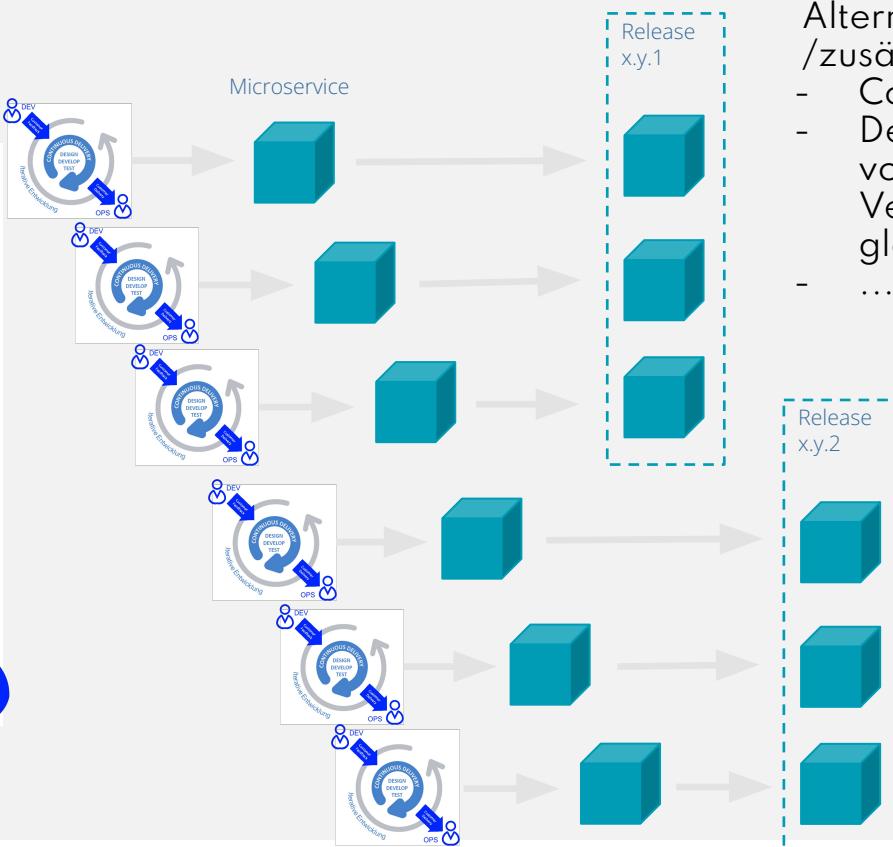
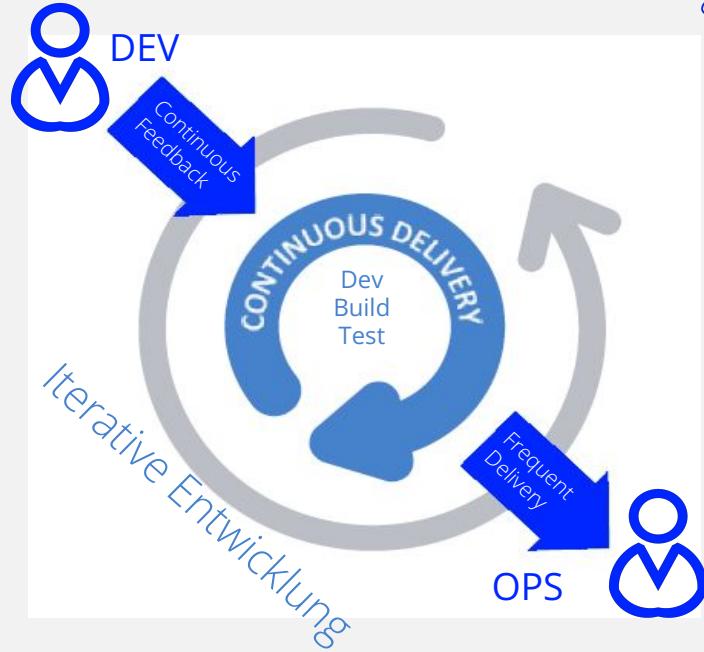
**Alle Testebenen sind heute gut automatisierbar.
Anstelle der klassischen Testpyramide kann die
Verteilung auch so aussehen:**



Beispiel einer Continuous Delivery Pipeline



Continuous Delivery in Microservice-Architekturen: Build und Release Modell für Releaseeinheiten



Alternativ /zusätzlich:

- Contract Tests
- Deployment von mehreren Versionen gleichzeitig
- ...

Übung 1: Sonarqube und Jenkins



DevOps

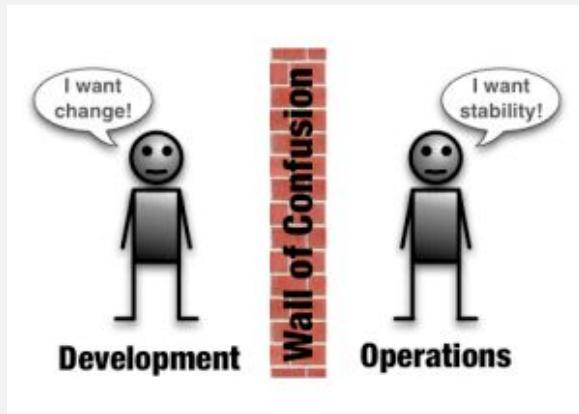


Was ist eigentlich DevOps?



QA|WARE

DevOps ist die **verbesserte Integration** von **Entwicklung und Betrieb** durch mehr **Kooperation und Automation** mit dem Ziel, Änderungen schneller in Produktion zu bringen und die MTTR dort gering zu halten. DevOps ist somit eine Kultur.



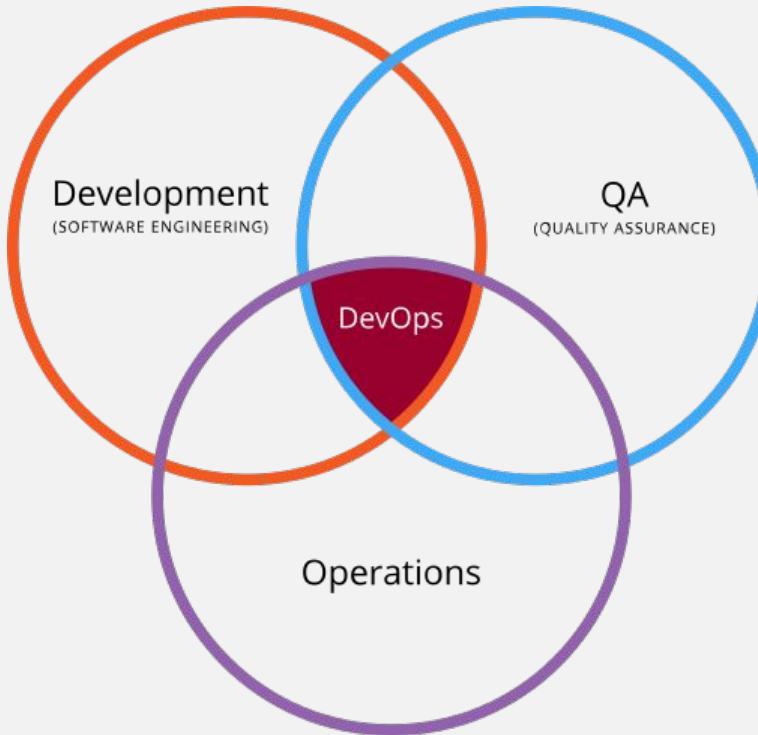
MVP + Feature-Strom
Pro Feature:

- Minimaler manueller Post-Commit-Anteil bis PROD
- Diagnostizierbarkeit des Erfolgs eines Features
- Möglichkeit Feature zu deaktivieren / zurückzurollen

DevOps verbindet DEVelopment, OPerations und Quality Assurance



QA|WARE

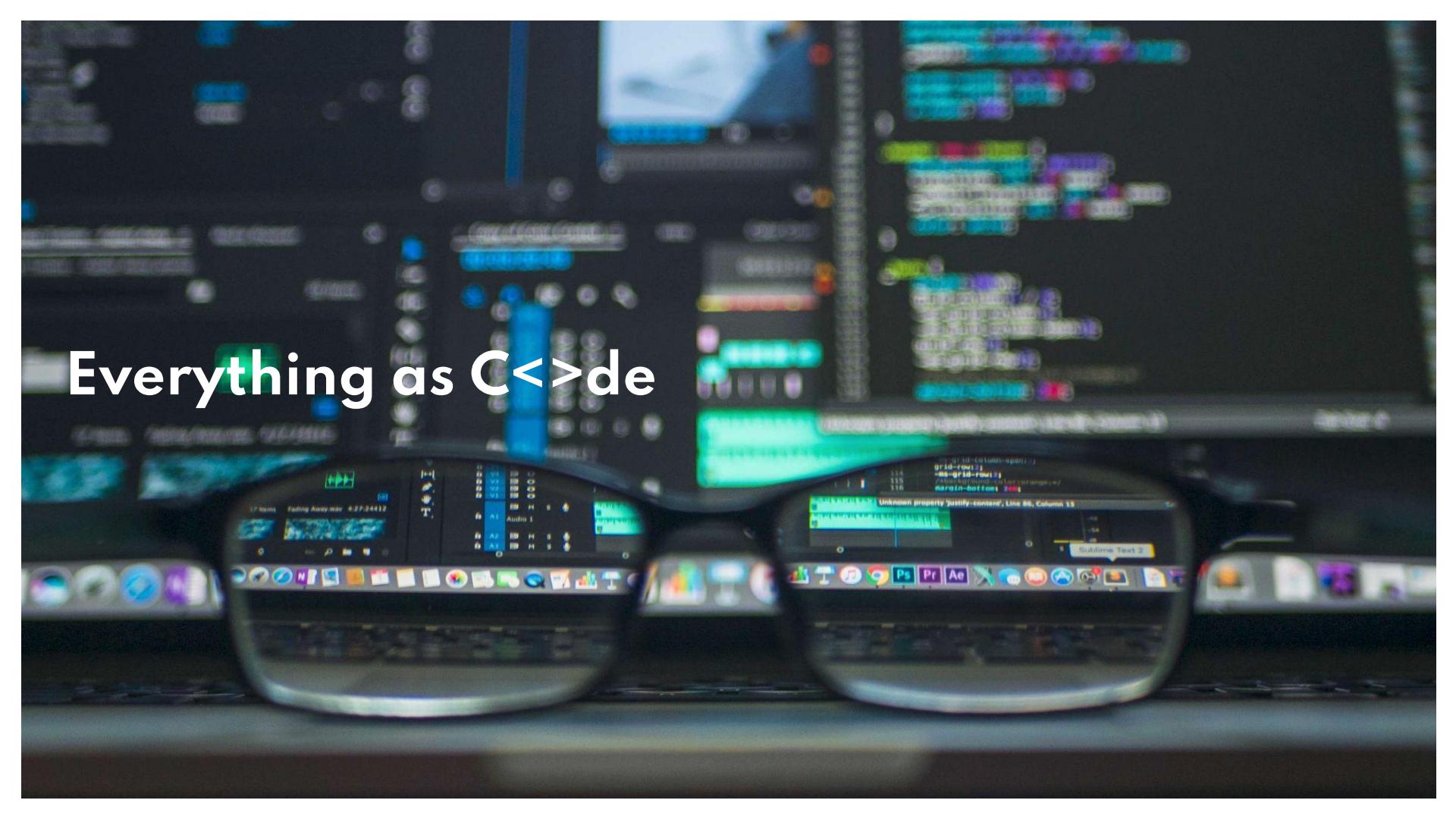


„Venn diagram showing DevOps“ by Rajiv.Pant/Wylve; Creative Commons 3.0
QAware | 17

Übung 2: DevOps Topologies



Continuous Delivery: Bausteine



Everything as Code

Alles, was die CD-Umgebung mit Leben befüllt, kommt aus dem VCS:



QA|WARE

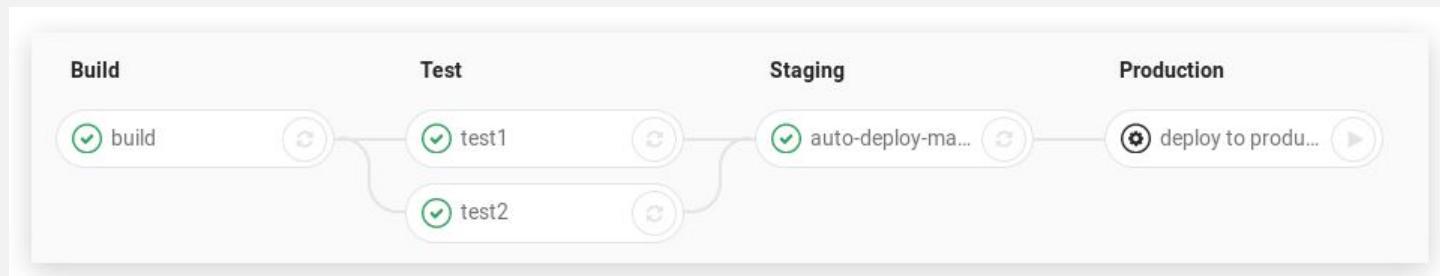
- Build-as-Code
 - Maven, Gradle, ...
 - Beschreibt wie die Anwendung gebaut wird
- Test-as-Code
 - Unit-, Component-, Integration-, API-, UI-, Performance-Tests
 - Beschreibt wie das Projekt getestet wird
- Infrastructure-as-Code
 - Docker, Terraform, Vagrant, Ansible, Marathon-Deployments
 - Beschreibt, wie die Laufzeitumgebungen aufgebaut werden
- Pipeline-as-Code
 - Build-Pipeline per Jenkinsfile
 - Buildklammer: Beschreibt alle Schritte bis zur lauffähigen Installation

Beispiel: GitLab Pipelines

Beispiel: GitLab Pipelines

- Bestehen aus Jobs und Stages
 - Jobs: Beschreiben, was zu tun ist (kompilieren, Tests ausführen)
 - Stages: Beschreiben, wann etwas zu tun ist (erst kompilieren, dann testen)

- `build`, with a job called `compile`.
- `test`, with two jobs called `test` and `test2`.
- `staging`, with a job called `deploy-to-stage`.
- `production`, with a job called `deploy-to-prod`.



Pipeline as Code: .gitlab-ci.yml

```
stages:
- build
- test
- deploy

job 1:
  stage: build
  script: make build dependencies
job 2:
  stage: build
  script: make build artifacts
job 3:
  stage: test
  script: make test
job 4:
  stage: deploy
  script: make deploy
```

Die Pipeline wird bei jedem Push ausgeführt

All 2	Pending 0	Running 0	Finished 2	Branches	Tags	Run Pipeline	CI Lint
Status	Pipeline	Commit	Stages				
passed	#7927123 by 🎨 latest	git pipeline-graph -o d4de4a5c Update .gitlab-ci.yml	✓ ✓ ✓ ➡	⌚ 00:00:55	📅 1 hour ago		
passed	#7562143 by 🎨 latest	git master -o 4a2f619e Update .gitlab-ci.yml	✓ ✓ ✓ ➡	⌚ 00:00:57	📅 2 weeks ago		

Pro Pipeline und Job kann man sich die Logs anschauen:

The screenshot shows a GitLab interface with a sidebar on the left and a main content area. The sidebar includes sections for Flight (Project, Repository, Issues, Merge Requests), CI / CD (Pipelines, Jobs, Schedules, Charts), Operations, Wiki, Snippets, and Settings. The 'Jobs' section is currently selected. The main content area displays the log output for a pipeline run. The log starts with a message about Knapsack global time execution, followed by a list of pending failures. It then details a merge request and its execution. The log concludes with a summary of the run, a timestamp, and a series of commands and their outputs related to artifact uploads. On the right side of the main content area, there is a sidebar titled 'slack' which lists a commit, a pipeline, and a notification configuration.

```
Knapsack global time execution for tests: 09m 14s

Pending: (Failures listed here are expected and do not affect your suite's status)

1) Filter issues filter issues by assignee only assignee filters issues by invalid assignee
   # to be tested, issue #26546
   # ./spec/features/issues/filtered_search/filter_issues_spec.rb:116

2) Merge request < User sees mini pipeline graph as json avoids repeated database queries
   # Temporarily skipped with xit
   # ./spec/features/merge_request/user_sees_mini_pipeline_graph_spec.rb:36

Finished in 9 minutes 43 seconds (files took 16.72 seconds to load)
200 examples, 0 failures, 2 pending

$ date
Mon Jun 3 14:16:44 UTC 2019
▼ Running after script...
$ date
Mon Jun 3 14:16:46 UTC 2019
► Not uploading cache debian-stretch-ruby-2.6.3-node-10.x-3 due to policy
► Uploading artifacts...
coverage/: found 5 matching files
knapsack/: found 4 matching files
rspec_flaky/: found 4 matching files
rspec_profiling/: found 1 matching files
WARNING: tmp/capybara/: no matching files
Uploading artifacts to coordinator... ok          id=224162288 responseStatus=201 Created token=bBmyXJNW
Uploading artifacts...
junit_rspec.xml: found 1 matching files
Uploading artifacts to coordinator... ok          id=224162288 responseStatus=201 Created token=bBmyXJNW
Job succeeded
```

slack

Commit 554bc7e2 in !3
Publish UMD bundle to flight-umd

Pipeline #66 for make-release-umd-bundle

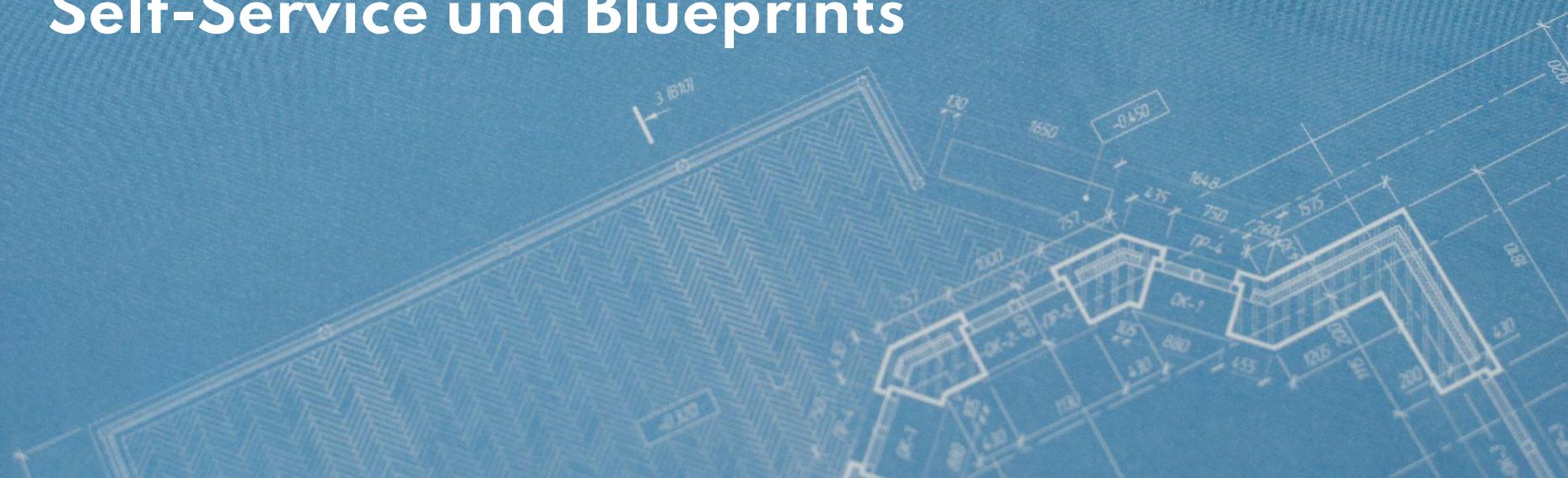
notify

slack

slack

ARCHITECTURAL DRAWINGS

Self-Service und Blueprints



Blueprints und Templates helfen beim Setup



QA|WARE

Blueprints & Templates:

- Die Build-Pipelines der (Micro-)Services in einem Projekt sind sich oft sehr ähnlich.
- Blueprints und Templates geben einen Rahmen vor und schaffen implizit Konventionen.
- Beispiele:
 - Durch die Verwendung des Jenkins Pipeline Multibranch Plugins wird für jeden Branch automatisch eine eigene Pipeline angelegt.
 - Identische Anbindung / Integration von Plattform-Komponenten (z.B. SonarQube, Artifactory)

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

Generatoren können intuitiv z.B. über ChatBots gesteuert werden



QA|WARE

ChatBots sind z.B. eine Lösung zur intuitiven Steuerung von Generatoren.

- Direkte Integration in Slack / Mattermost / ...
- Aufträge an den OpsBot werden einfach per Message gestellt
- Feedback von CI/CD Ereignissen und Aufträgen kommen als Antwort zurück

 svc.qabuild · Apr-6 15:51 @TobiasPlacht Success! Your Project is available on GitHub: https://github.com/foo-org/hello-world
Creating Github Webhook
Creating Jenkins Job
@TobiasPlacht Success! Your Jenkins Job is available at: 147.75.100.119/job/hello-world
@TobiasPlacht To start the Jenkins Job use buildNow hello-world
 qabuild-jenkins · Apr-6 15:51 Deployment with ID 043527e6-9c2e-4135-b9df-c658437c22d3 succeeded at 13:50:54 Application: /zwitscher-eureka
qabuild-jenkins · Apr-6 15:51 Deployment with ID 043527e6-9c2e-4135-b9df-c658437c22d3 completed the following step at 13:50:54 Completed Step: ScaleApplication
qabuild-jenkins · Apr-6 15:52 Started Build for: Job 'hello-world/master [1]' (http://147.75.100.133:8001/job/hello-world/job/master/1/)
qabuild-jenkins · Apr-6 15:52 FAILED: Job 'hello-world/master [1]' (http://147.75.100.133:8001/job/hello-world/job/master/1/)
qabuild-jenkins · Apr-6 15:52 Started Build for: Job 'hello-world/new-feature-branch [1]' (http://147.75.100.133:8001/job/hello-world/job/new-feature-branch/1/)
qabuild-jenkins · Apr-6 15:53 SUCCESS: Job 'hello-world/new-feature-branch [1]' (http://147.75.100.133:8001/job/hello-world/job/new-feature-branch/1/)

Continuous Delivery Pipeline für Cloud-native Anwendungen



Diagnosability



Nicht nur Cloud Native Anwendungen müssen diagnostizierbar sein, sondern auch die CD-Umgebung.



Beispiel Log-File Auswertung mit ELK/EFK:

- ELK:
 - Elasticsearch als DB
 - Kibana für Dashboards und Auswertungen
 - Logstash zum einsammeln der verteilten Logdaten
- Auch hier sollte die Plattform (Cloud, Jenkins, ...) und die Applikationsumgebungen integriert werden.
- Für Docker Container, die auf stdout loggen, können Log-Driver (z.B. Fluentd) konfiguriert werden.
 - = EFK (Elasticsearch, Fluentd, Kibana)
- Log-Files in den Containern können per Logstash & Filebeat integriert werden.

Nicht nur Cloud Native Anwendungen müssen diagnostizierbar sein, sondern auch die CD-Umgebung



QA|WARE



Beispiel Monitoring mit Prometheus:

- Prometheus kann für das Monitoring der CD-Plattform sowie für das Monitoring der Applikationsumgebungen benutzt werden.
- Client Libraries zur Instrumentierung sind für alle wichtigen Programmiersprachen vorhanden.
- Dashboards können einfach mit Grafana angelegt werden.
- Alert-Manager kann Störungen per E-Mail, Pager-Duty, Slack ... melden.

CD für CD

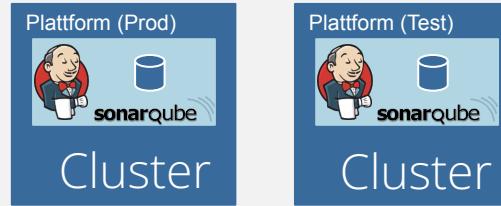


Continuous Delivery für Continuous Delivery



QA|WARE

- Auch Änderungen und Erweiterungen der CD Plattform müssen getestet werden.
- Durch den „Everything-as-Code“ Ansatz ist das aber sehr einfach:
 - Komplette Klone der Testumgebung (z.B. für Infrastruktur-Tests) können in unter einer Stunde instanziert werden.
 - Build-Plattform kann für Tests (z.B. bei Jenkins-Update) können als weitere Instanz angelegt werden.



Deployment

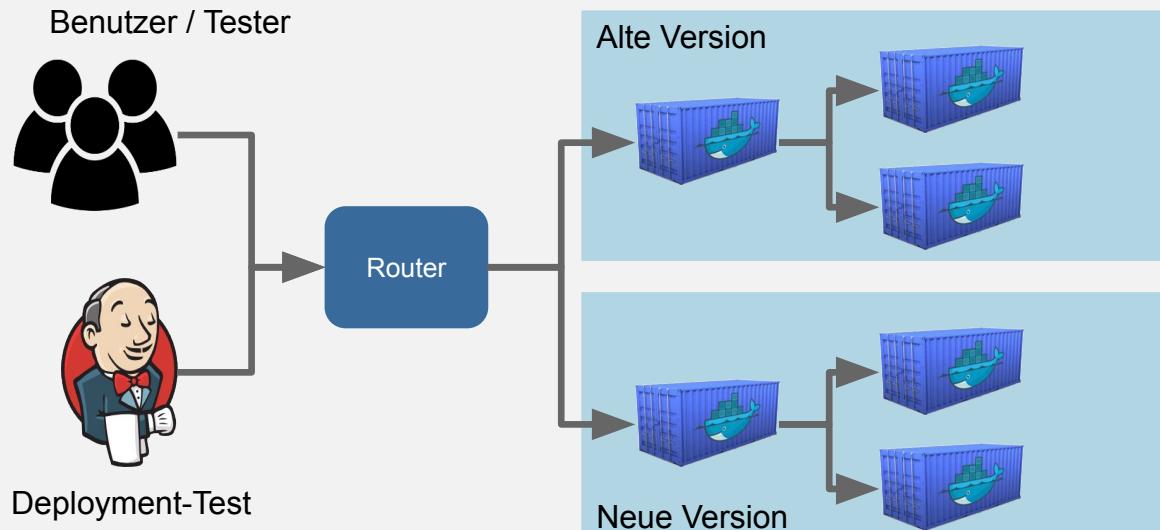


Mögliche Strategie: Canary Releases



QA|WARE

- Grundidee:
 - Neue Deployments werden nur von Testern / einem kleinen Teil der Nutzer benutzt.
 - Der Großteil der Benutzer wird erst auf die neue Version geleitet, falls sich diese als stabil erwiesen hat.



1. Die neue Version wird neben der alten Version deployed. Ein Router steuert, wer welche Version benutzt
2. Nur der Post-Deployment Test aus der Pipeline heraus wird auf die neue Version geleitet.
3. Erst danach wird die erste Teilgruppe der Benutzer / Tester auf umgeleitet.
4. Wenn keine Fehler auftreten, werden alle Benutzer umgeleitet
5. Die alte Version wird offline genommen

Wiederholungsfragen

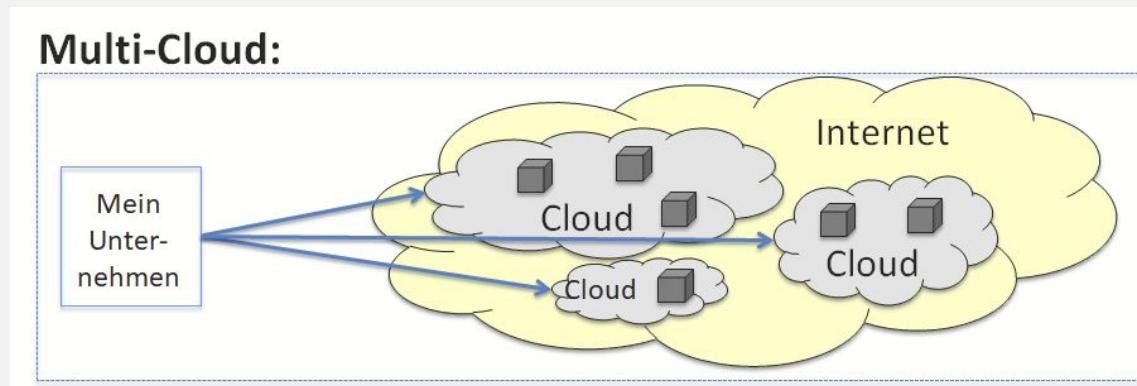
Was sind die Fünf Gebote der Cloud?

- Everything fails all the time
- Focus on MTTR, not on MTTF
- Respect the eight fallacies of distributed computing
- Scale out, not up
- Treat resources as cattles, not as pets



Quelle: https://de.wikipedia.org/wiki/Zehn_Gebote

Was sind mögliche Gründe für Unternehmen, einen Multi-Cloud-Ansatz zu nutzen? Nennen Sie mindestens zwei Gründe.



- Ausfallsicherheit durch Redundanz (unterschiedliche Standorte)
- Lokalität (nicht jeder Anbieter bietet weltweit Rechenzentren an)
- Geringere Kosten (je nach Anwendungsfall und Tageszeit)
- Unterschiedliches Angebot (spezielle Angebote für ML, ...)

Was sind mögliche Vorteile eines Binärprotokolls im Vergleich zu REST mit JSON-Nachrichten? Benennen Sie mindestens zwei Vorteile.

- Encoding der Payload als komprimiertes Binärformat. Dadurch kleinere Datenmenge und Übertragungszeit.
- Separate Schnittstellenbeschreibungen (IDLs, *Interface Definition Languages*) aus denen dann Client- und Server-Code in mehreren Programmiersprachen generiert werden können. Polyglott.

Gehen Sie davon aus, dass sich in Ihrer lokalen Docker Registry ein Image mit dem Namen *book-service* und dem Tag *latest* befindet.

Erzeugen Sie mit Docker einen laufenden Container für das Image *book-service*. Leiten Sie dabei den Container-Port 8080 auf den Host-Port 18080 weiter. Setzen Sie zusätzlich die Umgebungsvariable **JAVA_OPTS=-Xmx128m. Wie lautet die vollständige Befehlszeile?**

```
docker run -p 18080:8080 -e JAVA_OPTS=-Xmx128m book-service
```

Alternativ:

```
docker run -p 18080:8080 -e JAVA_OPTS=-Xmx128m book-service:latest
```

Was versteht man unter dem Begriff *Provisionierung*?

Provisionierung ist die Bezeichnung für die automatisierte Bereitstellung von IT-Ressourcen.

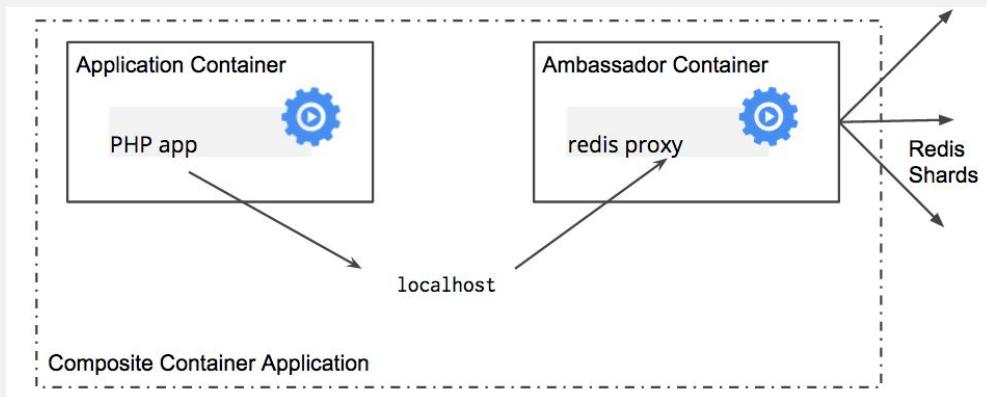
Alternativ:

Provisionierung ist die Überführung von einem System in seinem aktuellen Zustand auf einen Ziel-Zustand.

Welche Ressourcen-Typen werden typischerweise über Infrastructure-as-a-Service bereitgestellt?

- Rechenleistung
- Speicher
- Netzwerk

Wählen Sie eines der Orchestrierungsmuster „Ambassador Containers“, „Adapter Containers“ oder „Sidecar Containers“ aus und stellen Sie dieses graphisch dar. Geben Sie einen Use-Case an, für den sich dieses Orchestrierungsmuster gut eignet.



Use Case: Proxy von Verbindungen via localhost zu echten Endpoints. Für die lokale Entwicklung kann eine Datenbank auf localhost angesprochen werden, in der Produktion dann die echte DB. Aus Sicht der Anwendung läuft in beiden Fällen die Kommunikation über localhost.

Was sind die Unterschiede zwischen *Parallelität im Kleinen* und *Parallelität im Großen*?

Bei Parallelität im Kleinen gibt es einen Knoten mit einem Prozess, in dem N Threads laufen (1 Knoten, 1 Prozess, N Threads).

Bei Parallelität im Großen gibt es zusätzlich mehrere Knoten und Prozesse (L Knoten, M Prozesse, N Threads).