

Cloud Computing

Kapitel 12: Continuous Delivery & DevOps

Kapitel 13: Serverless Computing

Simon Bäumler

Simon.baeumler@qaware.de

TH Bingen, 23.01.2018

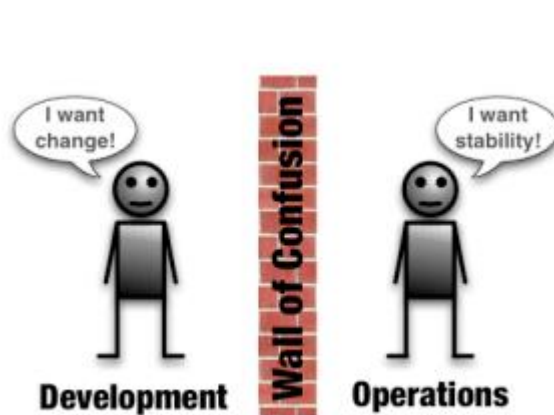
The background is a solid dark blue color with a subtle, light blue geometric pattern. This pattern consists of numerous small dots connected by thin lines, forming a complex, interconnected network that resembles a molecular structure or a digital data mesh. The lines and dots are more concentrated in some areas, creating a sense of depth and complexity.

Kapitel 12

Continuous Delivery

Was ist DevOps?

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.

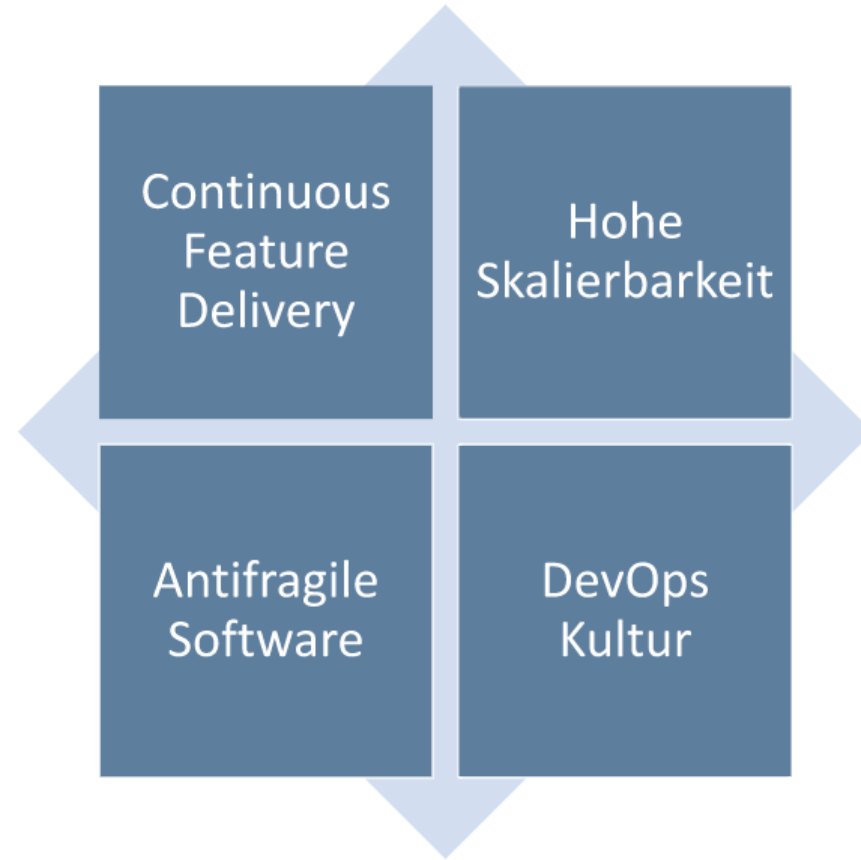


MVP + Feature-Strom

Pro Feature:

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

Treiber für Cloud-native Anwendungen



Continuous Delivery - Definition

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

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

Continuous Deployment

- Jede Änderung wird in Produktion deployed
- Ein Teil der Qualitätstests finden dadurch in Produktion statt
 - → Möglichkeit damit Umzugehen muss vorhanden sein (z.B. Canary Release, siehe später)

Continuous Delivery

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

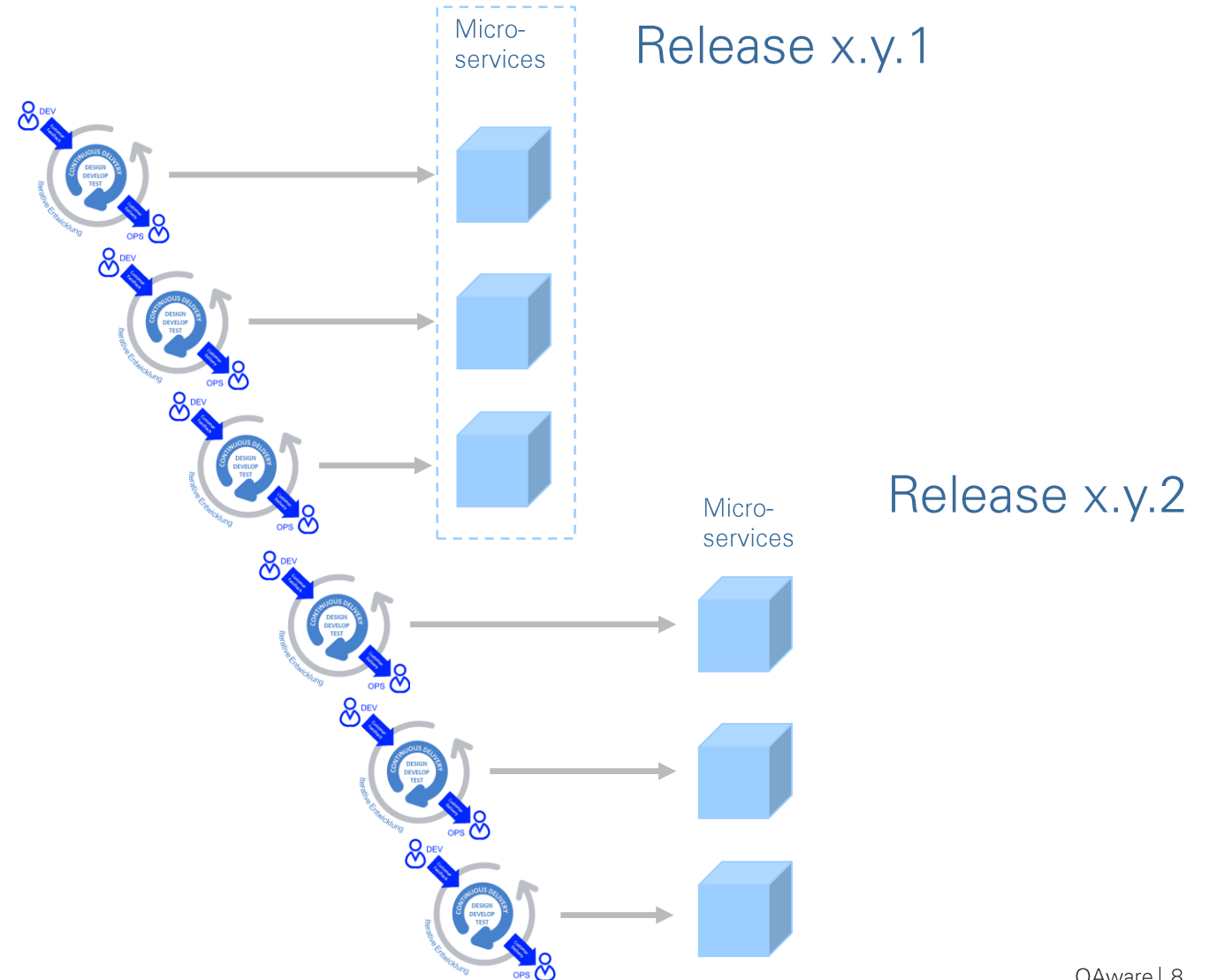
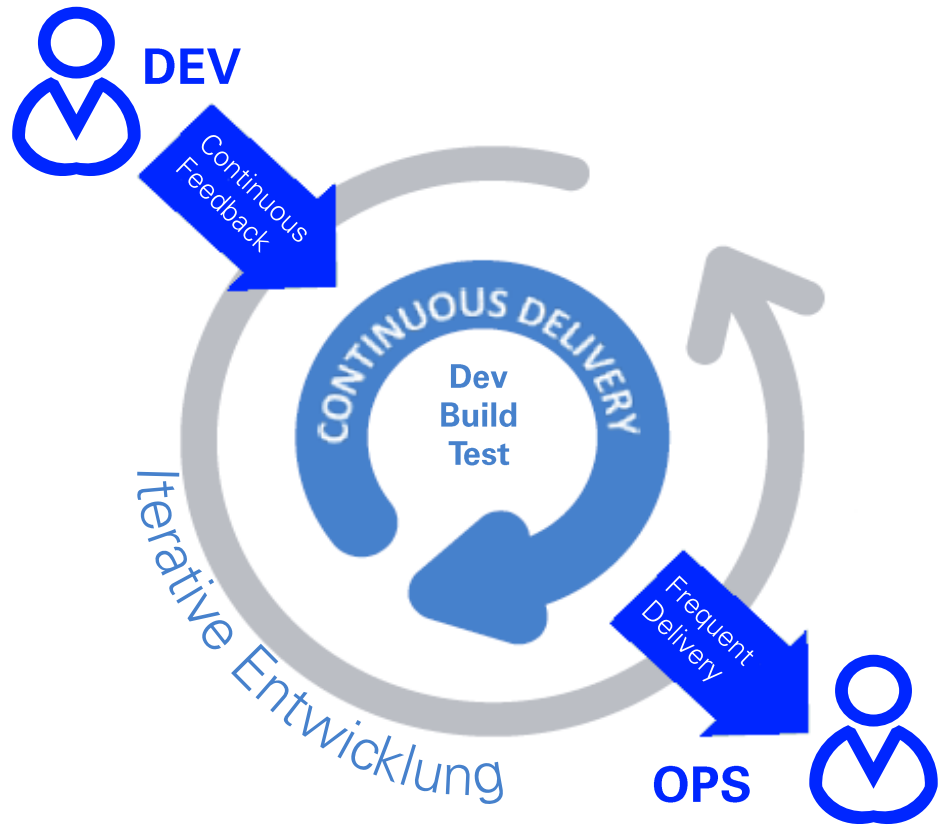
Kriterien für Continuous Deployment

“You’re doing continuous delivery when:

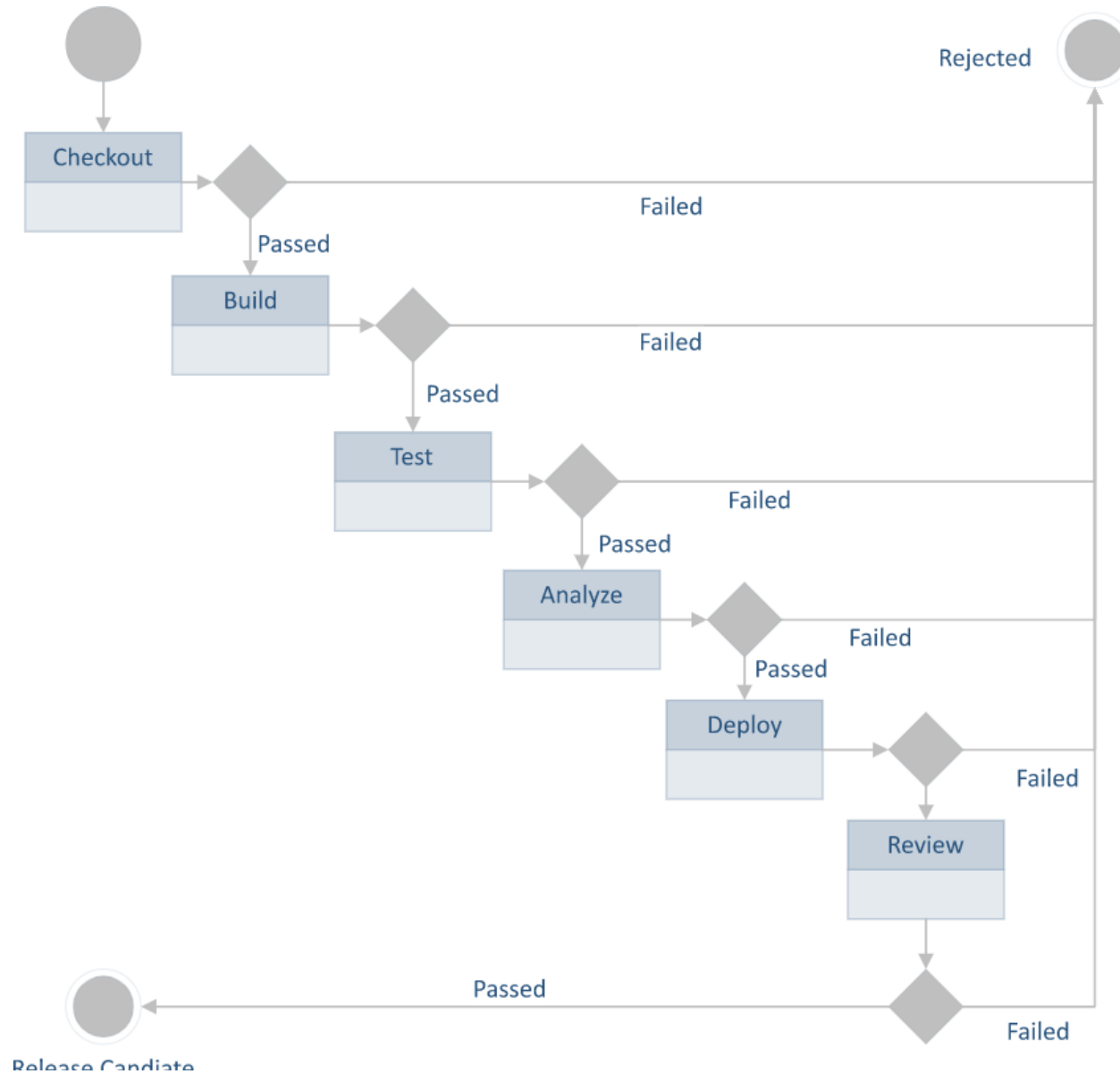
- Your software is deployable throughout its lifecycle
- Your team prioritizes keeping the software deployable over working on new features
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them
- You can perform push-button deployments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

nach M. Fowler / Continuous Delivery working group at ThoughtWorks

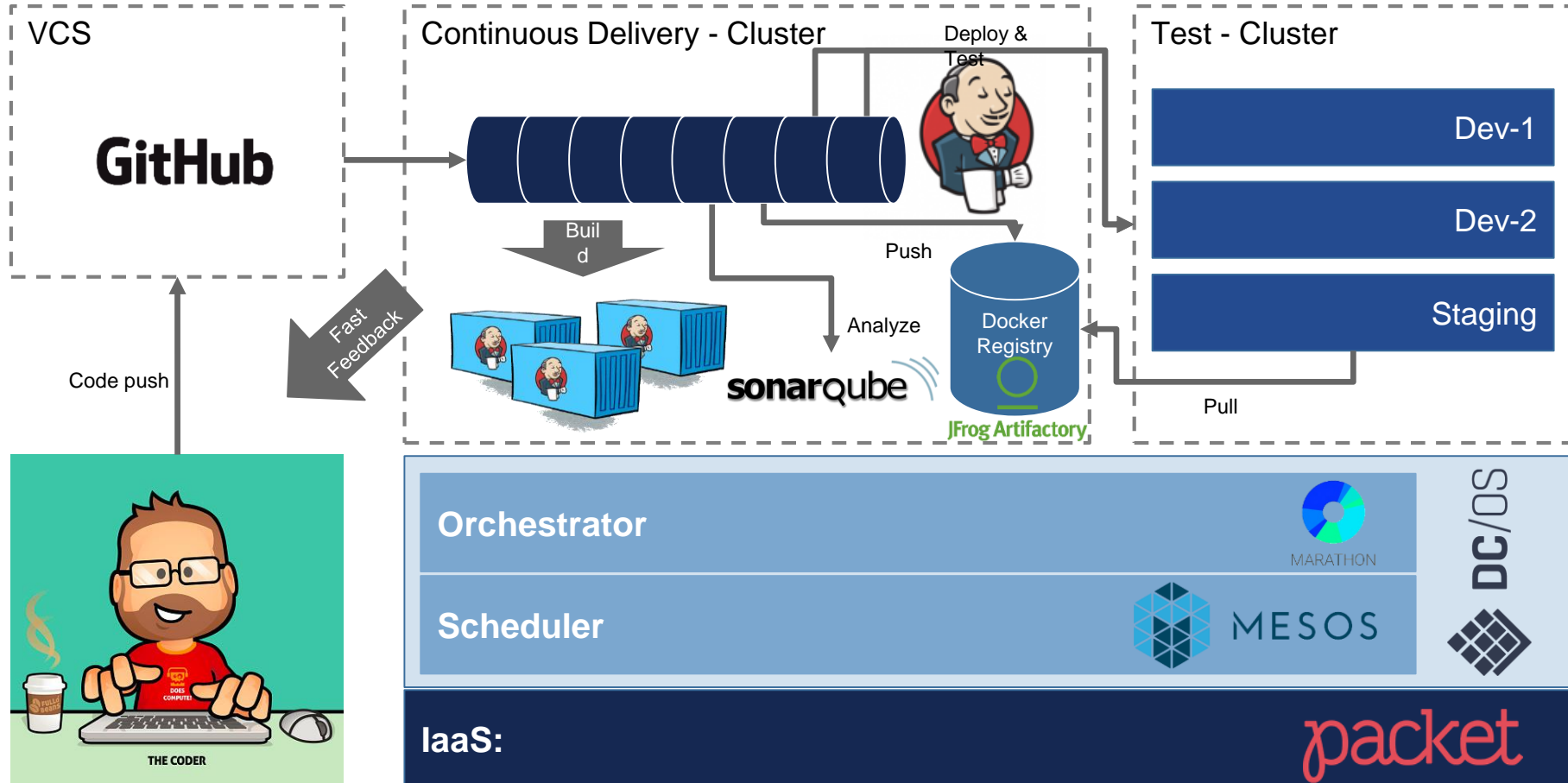
Continuous Delivery: Build und Release Modell in Microservicearchitekturen



Die Continuous Delivery Pipeline



Beispiel einer Continuous Delivery Pipeline





Continuous Delivery Bausteine



Everything as C<>de

Beispiel: Bootstrapping der CD-Plattform



Marathon-Deployments

Services:

- Marathon Event Subscriber
- OpsBot



Marathon-Deployments

Platform:

- Jenkins
- SonarQube (mit Datenbank)
- NGINX Reverse Proxy
- Artifactory



Terraform

Infrastructure:

- CoreOS VMs
- DC/OS Nodes (Master, Private, Public)
- NFS

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

- Build-as-Code
 - Maven, Gradle, ...
 - Beschreibt wie der Code compiliert 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 Laufzeitumgebung aufgebaut wird
- Pipeline-as-Code
 - Build-Pipeline per Jenkinsfile
 - Buildklammer: Beschreibt alle Schritte bis zur Lauffähigen Installation



Self service & Blueprints

Damit Entwickler schnell arbeitsfähig sind, sind Generatoren und Blueprints wichtig (1/2).

ChatBots sind eine Lösung zur intuitiven Steuerung von Generatoren.

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



The screenshot displays a chat interface with a user and a chatbot. The user, @TobiasPlacht, sends a message to create a GitHub project and a Jenkins job. The chatbot, svc.qabuild, responds with success messages and provides the necessary URLs and instructions. The chat log is color-coded: green for successful actions, yellow for build status, and red for failures.

Chat Log:

- 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/>)

Damit Entwickler schnell arbeitsfähig sind, sind Generatoren und Blueprints wichtig (2/2).

Blueprints & Templates:

- Die Microservice Build-Pipelines in einem Projekt sind sich sehr ähnlich.
- Blueprints und Templates geben einen Rahmen vor und schaffen implizit Konventionen.
- Beispiele:
 - Durch die Verwendung des 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.BUILD_NUMBER}]")  
        }  
    }  
  
    stage('Build project') {  
        steps {  
            sh './gradlew clean build --info --no-daemon'  
        }  
    }  
  
    stage('Unit Test Reporting') {  
        steps {  
            junit allowEmptyResults: true, testResults: '**/build/test-results/*.xml'  
        }  
    }  
}
```

Continuous Delivery Pipeline für Cloud-native Anwendungen



Advanced & Enterprise Topics:

Diagnosability, CD für CD & Deployments

Nicht nur unsere Cloud Native Anwendungen müssen diagnostizierbar sein, sondern auch die CD-Umgebung. (1/2)



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 unsere Cloud Native Anwendungen müssen diagnostizierbar sein, sondern auch die CD-Umgebung. (2/2)

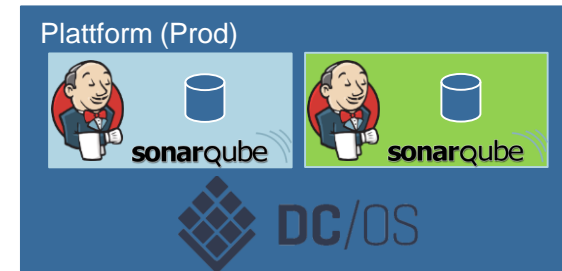
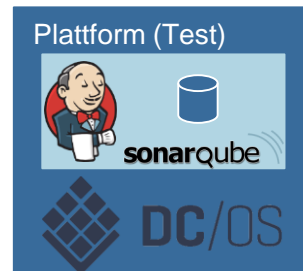
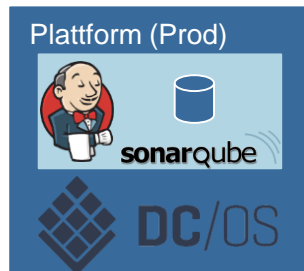


Monitoring mit Prometheus:

- Prometheus kann für das Monitoring der CD-Plattform sowie für das Monitoring der Applikationsumgebungen benutzt werden.
- Prometheus kann Marathon als Service Discovery nutzen.
- 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, HipChat, Slack ... melden.

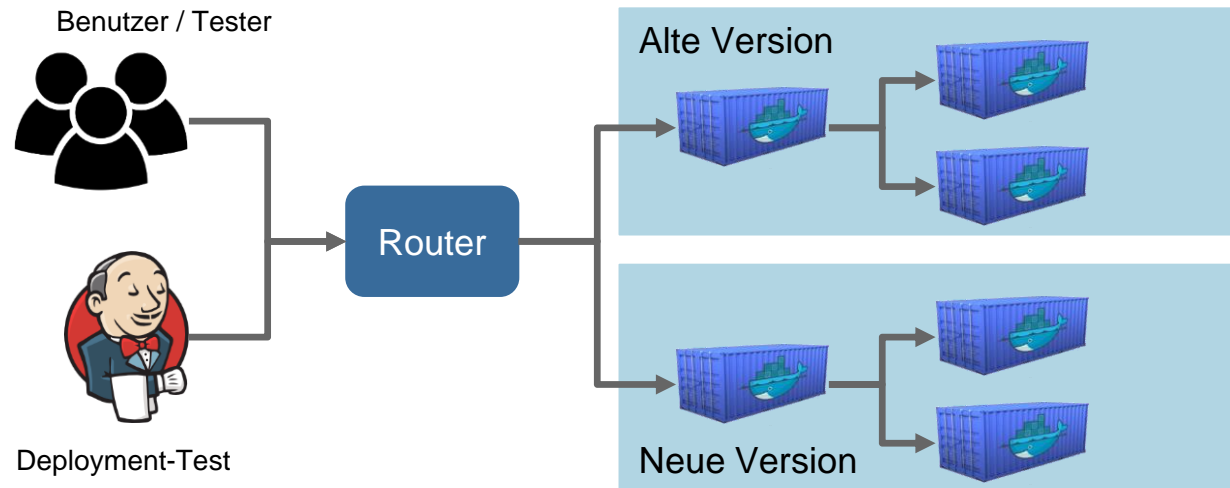
Continuous Delivery für Continuous Delivery

- 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 unter einer Stunde instanziiert werden.
 - Build-Plattform kann für Tests (z.B. bei Jenkins-Update) als weitere Instanz in DC/OS angelegt werden.




Wie kommen die Microservices in die Testumgebung? (2/2)

■ Canary-Release mit Vamp (Very Awesome Microservices Platform)



1. Nur der Post-Deployment Test aus der Pipeline heraus wird auf die neue Version geleitet.
2. Erst danach wird die erste Teilgruppe der Benutzer / Tester auf umgeleitet.
3. Die alte Version wird offline genommen

The background is a solid dark blue color with a faint, white, abstract network pattern. This pattern consists of numerous small dots (nodes) connected by thin, white lines (edges), forming a complex, interconnected web that resembles a molecular structure or a data network. The pattern is more dense in some areas and more sparse in others, creating a sense of depth and connectivity.

Kapitel 13

Serverless

Computing

Serverless Computing - Definition

Serverless computing is a [cloud computing execution model](#) in which the cloud provider dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity.^[1] It is a form of [utility computing](#).

Serverless computing still requires servers, hence it's a [misnomer](#).^[1] The name "serverless computing" is used because the server management and capacity planning decisions are completely hidden from the developer or operator. Serverless code can be used in conjunction with code deployed in traditional styles, such as [microservices](#). Alternatively, applications can be written to be purely serverless and use no provisioned services at all.^[2]

wikipedia.org

Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for [nearly any type of application](#) or backend service, and everything required to run and scale your application with high availability is handled for you.

amazon.com

Serverless Computing – Überblick & Vergleich mit PaaS

- Serverless Computing wird häufig auch als Function as a Service (**FaaS**) bezeichnet
- Deployment und Betrieb wird vom Cloud Betreiber gemanaged. Insofern ähnelt eine Serverless Computing Plattform einer PaaS
 - Unterschied zu ‚traditionellen‘ PaaS Plattformen: Der Betreiber garantiert nicht, dass eine einzelne Function ständig deployed ist. Eventuell wird diese bei Bedarf erst geladen/deployed.
- Üblicherweise wird Serverless Computing hauptsächlich für einzelne Funktionen benutzt. PaaS kann dagegen auch für komplexe Applikationen benutzt werden.
- Der primäre Architekturstil von FaaS ist Ereignisgetriebene Architektur (Event-driven Architecture / EDA)
 - Im gegensatz dazu sind die meisten klassischen Anwendungen in zustandsgetrieben
- Die größten Anbieter sind Google mit Google App Engine, Amazon mit AWS Lambda und Microsoft mit Azure Functions:



Serverless Computing – Vor- und Nachteile

Vorteile:

- Kosten: Da einzelne Funktionen nur bei Benutzung Deployed werden ist dies oft kosteneffektiver, als Server ständig zu betreiben
- Produktivität: Einzelne Funktionen können sehr schnell geschrieben, deployed und aktualisiert werden.
- Performance: Einzelne Funktionen können sehr feingranular skaliert werden.

Nachteile:

- Performance: Da einzelne Funktionen evtl. erst bei Bedarf geladen werden, können starke Schwankungen bei der Ausführung auftreten.
- Debugging: Außer Fehlermeldungen und Log-Output, hat man üblicherweise keine Möglichkeit zur Diagnose. Dies erschwert das Debugging / Profiling der Anwendung.

Serverless Computing – Open Source Frameworks

Eine Auswahl einiger Open Source Frameworks:

- Fission: FaaS Framework auf Basis von Kubernetes. Unterstützte Sprachen: NodeJs, Python, Ruby, Go, .Net, Perl
- Apache OpenWhisk: Momentan ein Incubating Projekt. Sprachen: NodeJs, Swift, Python, Java,...
- OpenFaaS: Framework für Docker und Kubernetes.
- Oracle Fn: Docker Natives Framework. Unterstützte Sprachen: Java, Go, Ruby, Python, PHP, and Node.js,...

Beispiel AWS Lambda



- Lambda ist ein AWS Service. Er wurde 2014 eingeführt.
- Zielgruppe: Vereinfachtes bauen von on-demand Applikationen.
- Ursprüngliches Ziel war die einfache Umsetzung von Use-Cases, wie z.B. Image-Upload in die AWS Cloud
- Unterstützte Sprachen:
 - Node.js, Python, Java, C# und Go.
- Im Gegensatz zur EC2 werden AWS-Lambda Funktionen in Inkrementen von 100ms abgerechnet (EC2: wird in Stunden abgerechnet).