

CLOUD-NATIVE PROGRAMMIERUNG

Unit 02:

*DevOps + Deployment
Pipelines*

Stand: 09.10.2020

INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls

Unit 01 Cloud Computing	Unit 02 DevOps	Unit 03 Infrastructure as Code
Unit 04 Standardisierte Deployment Units	Unit 05 Container Orchestrierung	Unit 06 FaaS
Unit 07 Polyglott Programming		

INHALTE

○ DevOps

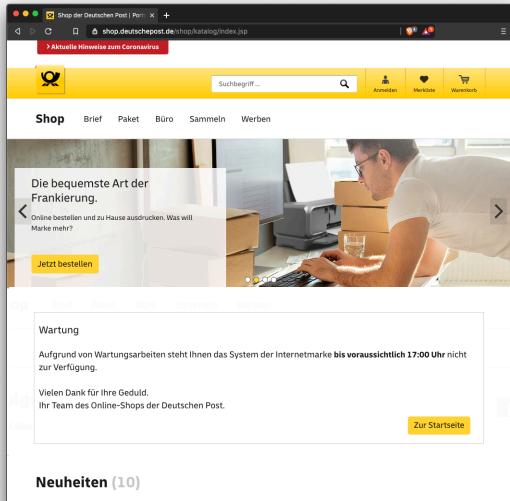
- DevOps Prinzipien
- DevOps-Cycle konforme Architekturen und Umgebungen
- Continuous Begriffe
- Deployment Pipelines

DAS PROBLEM



- Viele Banken datieren ihre IT in Rythmen häufig einmal pro Quartal auf (und nennen das agil).
- Solche Update Zyklen können dann schon mal für alle Bankautomaten der Deutschen Bank mehrere Stunden, ggf. Tage dauern.
- Häufig werden solche Updates in Zeiten gelegt, in denen wenig Publikumsverkehr erwartet wird. Z.B. Sonntag morgens um 04:00 Uhr.
- Bei Bankautomaten akzeptieren wir das meist.
- Häufig gibt es auch einen Automaten einer anderen Bank (z.B. Sparkasse statt Deutsche Bank) in der Nähe, der zu einem anderen Zeitpunkt aktualisiert wird.

DAS PROBLEM

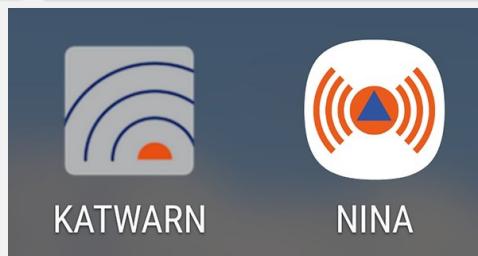


Gesehen am: So., 20.09.2020, 11:45 Uhr

5

DAS PROBLEM

Aber würden Sie das bei diesen Diensten akzeptieren?



verzeihung!

unser Service
steht Ihnen am
kommenen
Mittwoch von
07:00 Uhr bis
16:00 Uhr wegen
eines weltweiten
Updates nicht
zur Verfügung.

6

SCHLAGZEILEN

... die Minuten nach Problemautritt in den Medien landen.

Massive Ausfälle bei Google-Diensten: Gmail, Drive und weitere betroffen

Gmail und andere Google-Dienste kämpfen derzeit mit massiven Problemen. (Foto: Shutterstock)

Mehrere Google-Dienste, darunter **Gmail**, Drive, Docs und Meet, kämpfen derzeit mit Ausfällen. Grund für die Schwierigkeiten, die tausende Nutzer melden, scheinen Serverprobleme zu sein.

Solche Schlagzeilen kosten Geld aber vor allem Renommee (und damit noch mehr Geld).

Garmin down: Knicken Server wegen Ransomware-Attacke ein?

24.07.2020, 13:18 Uhr Seit Mittwoch sind die Server von Smartwatch-Hersteller Garmin nicht erreichbar. Grund dafür könnte eine Ransomware-Attacke sein.

NIX MEHR VERPASSEN: [UNSERE SCHLAGZEILEN](#)

DAS DEVOPS PROBLEM



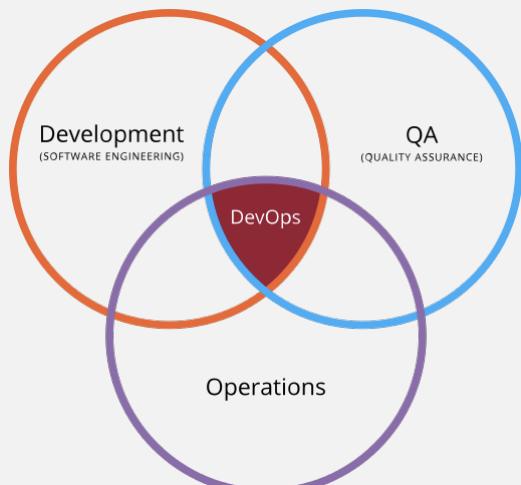
- Was machen Sie also, wenn Sie weltweit tätig sind (in allen Zeitzonen, in allen Kulturregionen)?
- Wenn Sie Ihre Dienstleistung ausschließlich online erbringen?
- Wenn Ihre Dienste überproportional häufig außerhalb der üblichen Geschäftszeiten nachgefragt werden (Video-Streaming, nachts, abends und am Wochenende)?
- Wenn Ihr Konkurrent nur einen Mausklick entfernt ist (Amazon Prime, Google Play, etc.)?
- Wenn Sie daher einfach keine Zeitfenster finden können, um weltweite Updates einzuspielen?

Angeblich spielt Amazon mehrere 100-mal pro Tag Updates in seine Produktivsysteme ein!

Als Vergleich:
Bekannte Banken etwa vier-mal pro Jahr!

DEVOPS

Verbindet *DE*velopment, *O*perations und *Q*uality *A*surance



DevOps ist die verbesserte Integration von **Entwicklung** und **Betrieb** durch mehr Kooperation und **Automation** mit dem Ziel Änderungen **schneller** in Produktion zu bringen.

DevOps ist vor allem eine **Kultur** (der Automatisierung und Verantwortung).

Quelle: „Venn diagram showing DevOps“; Rajiv, Pant, Welve; CC 3.0

DEVOPS

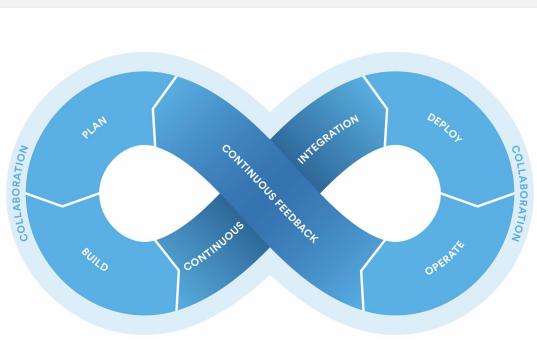
Der Versuch einer Definition

DevOps ist die Kultur, die Entwicklung (Dev) und den Betrieb (Ops) von Anwendungen besser miteinander zu integrieren, um

- kurze Releasezyklen,
- zuverlässige Inbetriebnahme und
- eine hohe Verfügbarkeit

zu erreichen.

Dies wird durch eine verbesserte Kooperation und einen höheren **Automatisierungsgrad** erreicht.



Damit DevOps allerdings auch reibungslos funktioniert, müssen Systeme konsequent darauf darauf ausgelegt werden:

- Kultur
- Arbeitsorganisation
- Architektur
- Deployment Pipeline
- Deployment Environments
- Telemetriedaten

INHALTE

- DevOps
- **DevOps Prinzipien**
- DevOps-Cycle konforme Architekturen und Umgebungen
- Continuous Begriffe
- Deployment Pipelines

DEVOPS PRINZIPIEN

des Flow

Arbeit sichtbar machen



In der IT (anders als in der Produktion) kann Arbeit häufig per „Mausklick“ bewegt werden.

Weil das so einfach ist, werden manche Arbeitspakete zwischen Teams wie Ping Pong hin und her bewegt.

Um zu erkennen, wo Arbeit gut fließt und wo nicht, kann man auf visuelle Arbeitsboards (z.B. Kanban-Boards) zurückgreifen.

Dabei sollte immer die komplette Wertkette abgebildet werden, d.h. bis ein Feature tatsächlich produktiv läuft.

Work in Process beschränken



Unterbrechungen sind bei der Herstellung physischer Güter sehr kostenintensiv.

Studien haben gezeigt, dass sich die Zeit zum Erledigen selbst einfacher Aufgaben (wie dem Sortieren geometrischer Formen) beim Multitasking deutlich verlängert.

Multitasking sollte daher reduziert werden, um zu vermeiden, dass zwischen zu vielen Schritten und Kontexten hin und her gesprungen wird.

Flaschenhälse minimieren

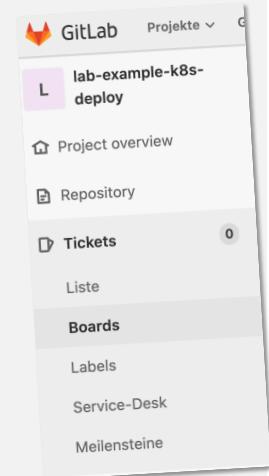
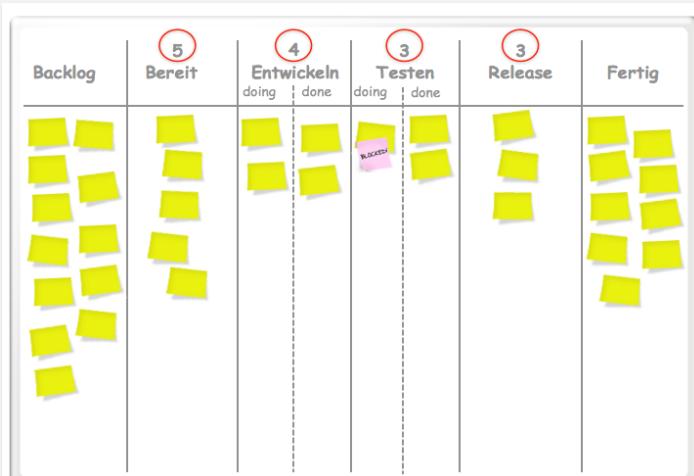


Flaschenhälse folgen häufig diesen Mustern:

- Langwieriges (manuelles) Erstellen von Test- und Produktiv-Umgebungen => Erstellung von Umgebungen im Self-Service auf Anforderung.
- Langwieriges (manuelles) Code-Deployment => Automatisiertes Deployment mittels Deployment Pipelines.
- Langwieriges (manuelles) Einrichten und Durchführen von Tests => Automatisiertes und parallelisiertes Testen, so dass die Testrate mit der Code-Entwicklungsrate mithält.
- Zu stark gekoppelte Architektur (lokale Änderungen müssen mit vielen abgestimmt werden) => Lose gekoppelte Architektur (z.B. Microservices) um Änderungen sicherer und mit mehr Autonomie durchführen zu können.

ARBEIT SICHTBAR MACHEN

Beispiel: Kanban-Board



Das ganze kann man natürlich auch elektronisch unterstützen.

In GitLab können Sie bspw. in jedem Projekt Ihre Tickets als ein solches Board darstellen.

Probieren Sie es aus!

<https://git.mylab.th-luebeck.de>

DEVOPS PRINZIPIEN

des Feedback

Probleme früh erkennen



Komplexe Systeme sind nicht vollständig von einer einzelnen Person überschaubar und verstehbar.

Daher sollten Annahmen, die beim Design getroffen wurden, kontinuierlich geprüft werden. Z.B. mit Chaos Engineering, um Vertrauen in die Fähigkeit des Systems aufzubauen, turbulenten und unerwarteten Bedingungen standzuhalten. Ferner benötigt man für Feedback-Schleifen im Produktivsystem kontinuierlich und automatisiert erhobene Telemetriedaten.

Probleme sofort lösen



Problembehandlungen im Produktivsystem sollten höhere Priorität als die weitere Entwicklung von Features bekommen.

Die Entwicklungspipeline sollte gestoppt werden.

Probleme sollten nie umgangen werden oder deren Lösung verschoben werden, um zu vermeiden, dass sich ein Problem fortsetzt und zu exponentiell steigendem Aufwand für dessen Behebung zu einem späteren Zeitpunkt führt.

Probleme professionell verantworten



Erstaunlicherweise erhöht in komplexen Systemen das Hinzufügen zusätzlicher Kontrollschritte die Wahrscheinlichkeit für zukünftige Fehler. Entscheidungen müssen von Personen getroffen werden, die weit weg vom Problemraum sind.

Besser ist es, wenn Entwickler Verantwortung auch für den operativen Betrieb haben (you build it, you run it).

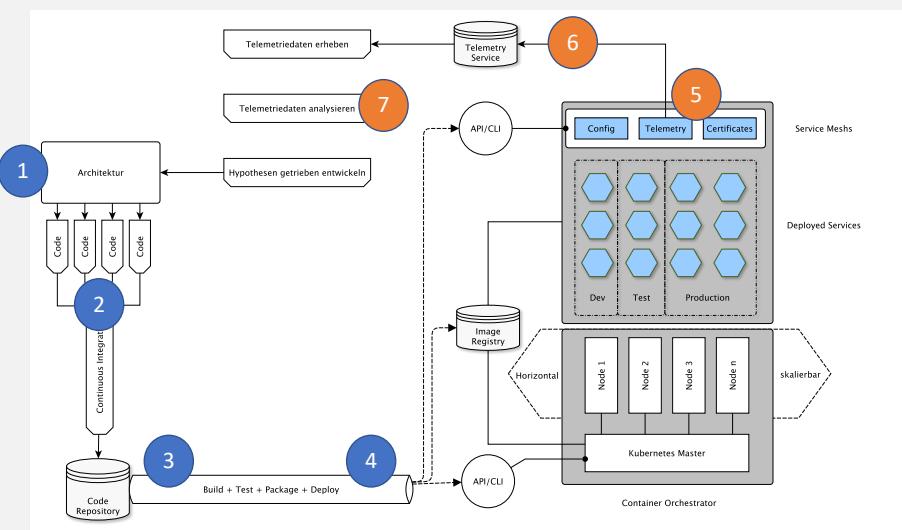
Dadurch liegt die Verantwortung für Qualität, Zuverlässigkeit und Entscheidungsbefugnis dort, wo auch die Arbeit erledigt wird und die meiste technische Expertise vorhanden ist.

INHALTE

- DevOps
- DevOps Prinzipien
- **DevOps-Cycle konforme Architekturen und Umgebungen**
- Continuous Begriffe
- Deployment Pipelines

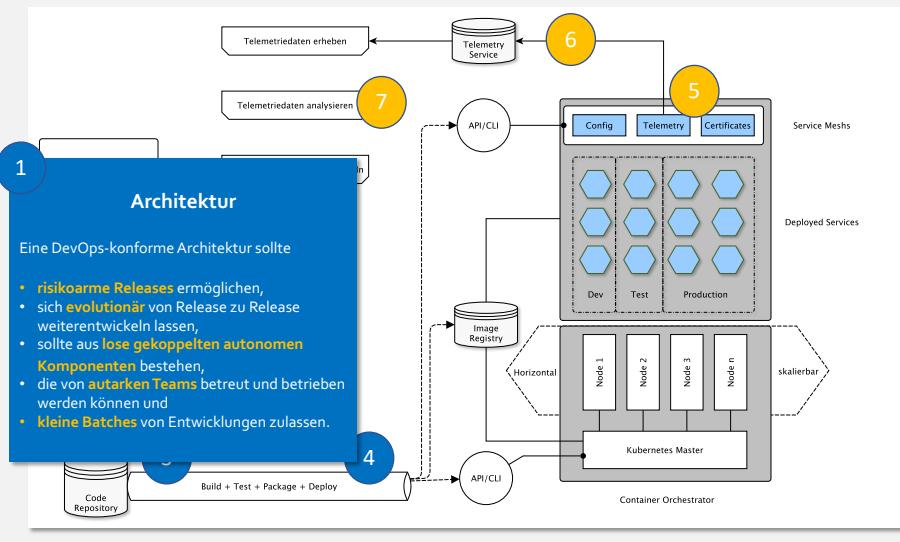
DEVOPS CYCLE

konforme Architekturen und Infrastrukturen



DEVOPS

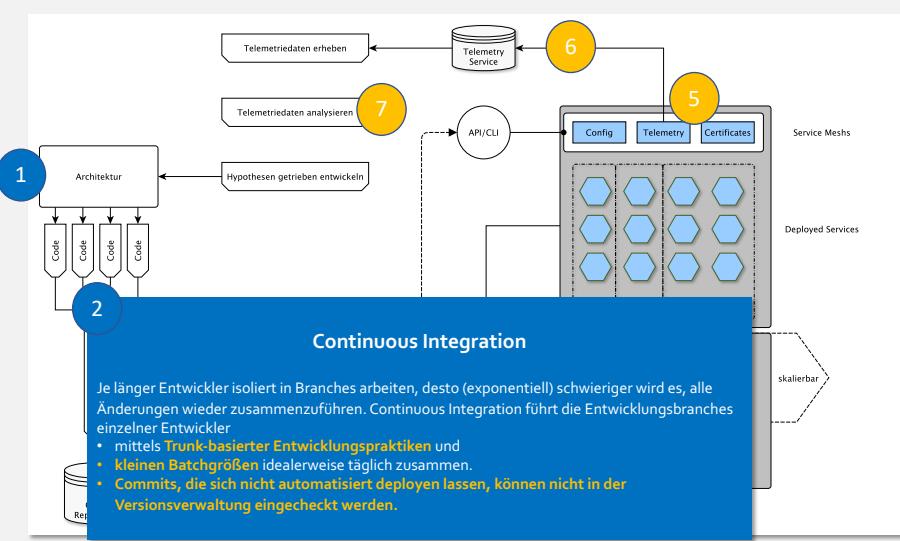
konforme Architekturen und Infrastrukturen



17

DEVOPS

konforme Architekturen und Infrastrukturen

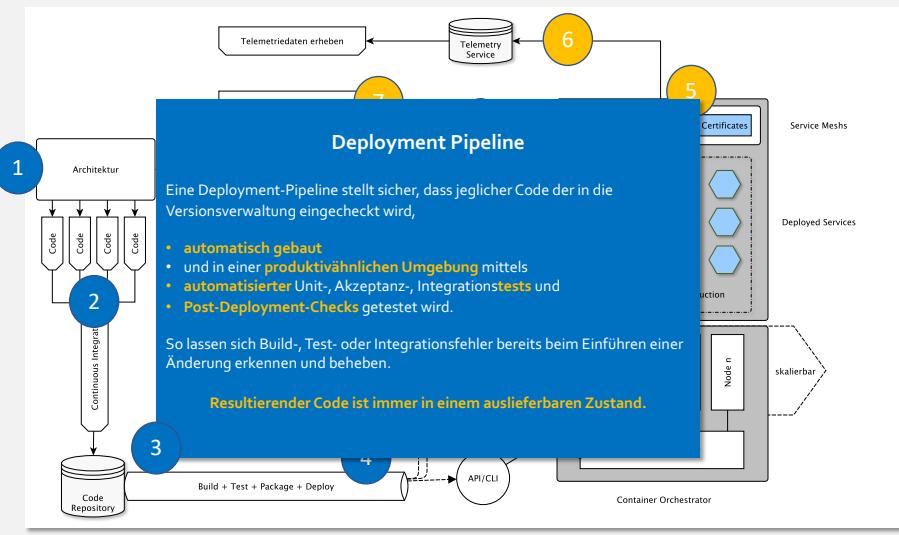


18

DEVOPS

konforme Architekturen und Infrastrukturen

Das Praktikum zu dieser Unit wird sich insbesondere mit Deployment Pipelines befassen.
(Stichwort: `.gitlab-ci.yaml`)



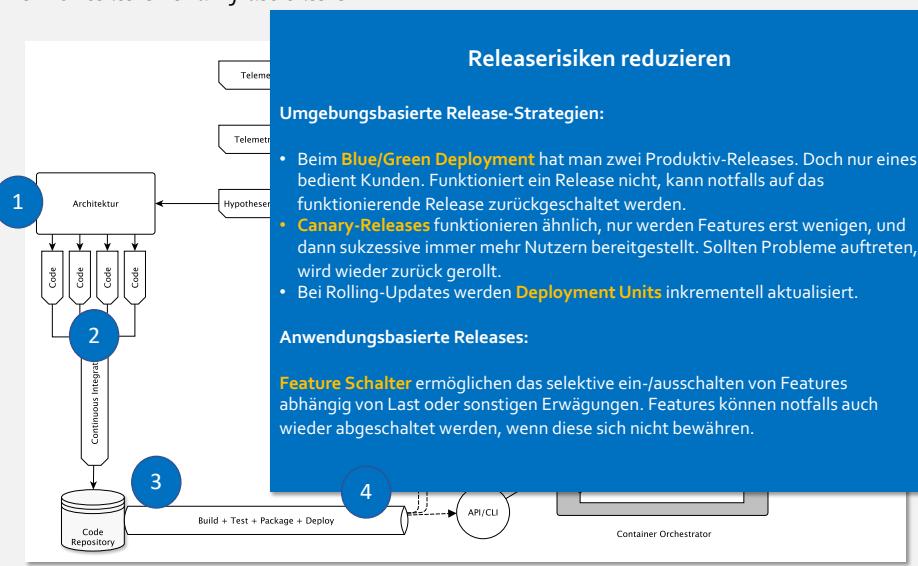
Prinzipien
des Flow

Prinzipien
des Feedbacks

verknüpft über
automatisierbare
technische Plattform

DEVOPS

konforme Architekturen und Infrastrukturen



Prinzipien
des Flow

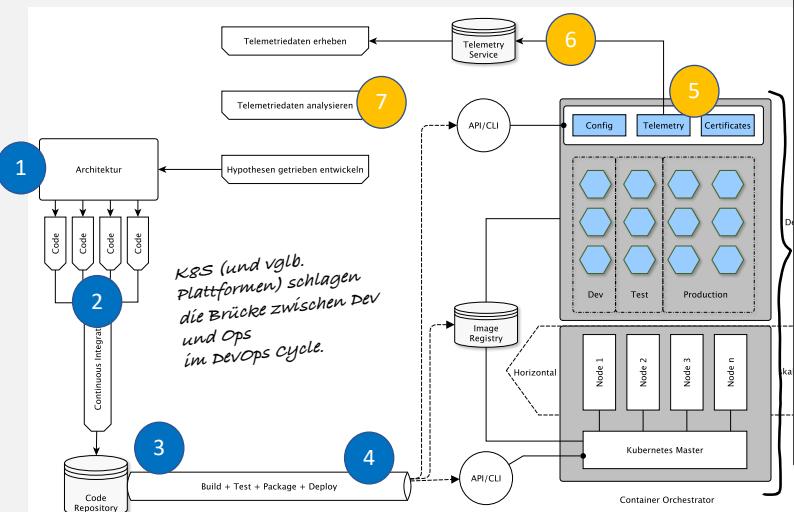
Prinzipien
des Feedbacks

verknüpft über
automatisierbare
technische Plattform

DEVOPS

konforme Architekturen und Infrastrukturen

Mit K8S werden wir uns insbesondere in Unit 05 befassen.



Kubernetes

ist eine Open-Source **Orchestrierungsplattform** und mittlerweile de-facto Standard für die Bereitstellung, Skalierung und Verwaltung von **Container-Anwendungen**.

Es zielt darauf ab, eine „Plattform für das **automatisierte Ausbringen, Skalieren und Warten** von Anwendungscontainern auf verteilten Hosts (**Clustern**)“ zu liefern.

Es kann so Flaschenhälse im DevOps Cycle minimieren und **Infrastructure as Code** für Test- und Produktivumgebungen ermöglichen, sowie die automatisierte Erhebung von **Telemetriedaten** vereinfachen

PROF.DR.
NANE KRATZKE

21

21

DEVOPS

konforme Architekturen und Infrastrukturen

Diesem Thema „*Observable Architectures*“ werden wir uns im zweiten Modul *Cloud-native Architekturen* widmen.

TECHNISCHE
HOCHSCHULE
LÜBECK

Telemetriedaten erheben:

Eine zentrale Telemetrie-Infrastruktur aufbauen

- Geschäftslogik und Infrastruktur erzeugen verteilte Events, Logs und Metriken,
- die in einem zentralen Service konsolidiert werden.

Anwendungen systematisch loggen

- Anwendungen müssen ausreichend Telemetriedaten erzeugen
- und an die Telemetrie-Infrastruktur angebunden werden.
- Jedes entwickelte Feature sollte auch geeignet instrumentiert werden.

Self-Service Zugriff auf Telemetriedaten

- Damit jeder im DevOps Team Probleme auch finden und beheben kann,
- muss auch jeder Metriken erstellen, generieren, anzeigen und analysieren lassen können.

Prinzipien
des Flow

Prinzipien
des Feedbacks

verknüpft über
automatisierbare
technische Plattform

PROF.DR.
NANE KRATZKE

22

22

DEVOPS

konforme Architekturen und Infrastrukturen

Jetzt wissen Sie, warum Sie mit Statistik gequält wurden ;-)

Telemetriedaten analysieren: 6

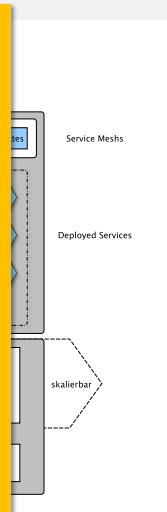
Mit Statistik potentielle Probleme erkennen

- Probleme können bei normalverteilten Ereignissen mit einfachen aber bewährten statistischen Verfahren wie Mittelwert und Standardabweichung identifiziert werden.
- Unterliegen Ereignisse keiner Normalverteilung, kann man auf glättende Verfahren (gleitende Durchschnitte), periodische/saisonale Verfahren (Kolmogorow-Smirnow, etc.) zurückgreifen.

Nur bei signifikanten Ereignissen automatisiert warnen

- Entfernen sich Metriken statistisch signifikant von ihren Normal-/Erwartungswerten,
- so sollten automatisierte Warnungen ausgelöst werden, da dies Vorboten eines Produktivzwischenfalls sein können.
- Hierfür ist solides statistisches Wissen über die Verteilung entsprechender Ereignisse erforderlich.

6



Prinzipien des Flow

Prinzipien des Feedbacks

verknüpft über automatisierbare technische Plattform

DEVOPS

konforme Architekturen und Infrastrukturen

Wenn Sie bei Amazon, Otto oder anderen Online-Shops kaufen, waren Sie mit hoher Wahrscheinlichkeit schon mal Proband (und haben es vermutlich nicht einmal mitbekommen).

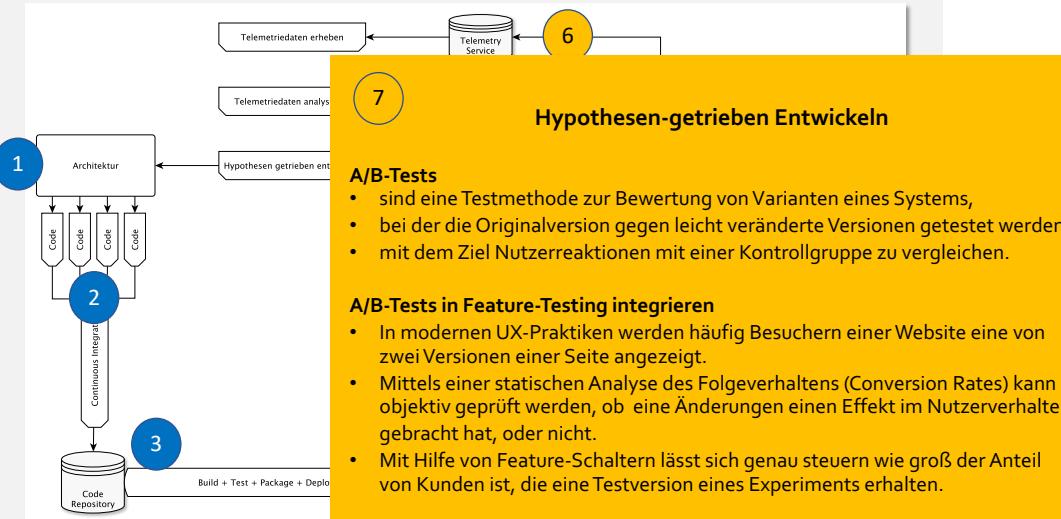
Hypothesen-getrieben Entwickeln 7

A/B-Tests

- sind eine Testmethode zur Bewertung von Varianten eines Systems,
- bei der die Originalversion gegen leicht veränderte Versionen getestet werden
- mit dem Ziel Nutzerreaktionen mit einer Kontrollgruppe zu vergleichen.

A/B-Tests in Feature-Testing integrieren

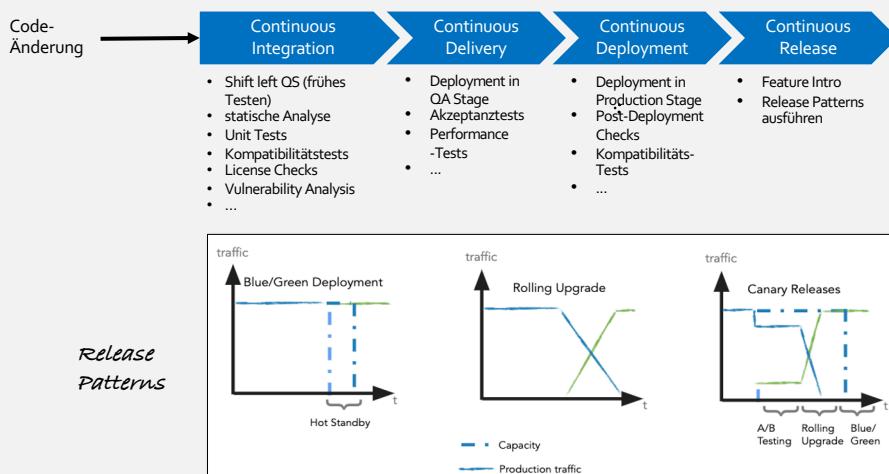
- In modernen UX-Praktiken werden häufig Besuchern einer Website eine von zwei Versionen einer Seite angezeigt.
- Mittels einer statischen Analyse des Folgeverhaltens (Conversion Rates) kann objektiv geprüft werden, ob eine Änderung einen Effekt im Nutzerverhalten gebracht hat, oder nicht.
- Mit Hilfe von Feature-Schaltern lässt sich genau steuern wie groß der Anteil von Kunden ist, die eine Testversion eines Experiments erhalten.



INHALTE

- DevOps
- DevOps Prinzipien
- DevOps-Cycle konforme Architekturen und Umgebungen
- **Continuous Begriffe**
- Deployment Pipelines

CONTINUOUS BEGRIFFE



CONTINUOUS DELIVERY

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.

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.

Merke:

Wikipedia ist oft der Beginn einer Recherche. Eine Recherche sollte aber nie bei Wikipedia enden.

CONTINUOUS X

Vorsicht vorm Begriffswirrwarr

Continuous Integration (CI)

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

Continuous Delivery (CD)

- Der **Code kann jederzeit deployed** werden.
- Der Code muss allerdings nicht immer deployed werden.
- Der Code muss (möglichst) zu jedem Zeitpunkt gebaut, getestet und debuggt werden können.

Continuous Deployment

- Jede **stabile Änderung** wird **in Produktion deployed**.
- Ein Teil der Qualitätstests findet dadurch in Produktion statt.
- Die Möglichkeit mit Fehlern umzugehen muss also vorhanden sein und vom DevOps Team beherrscht werden (z.B. Canary Releases).

Achtung:

Diese Begrifflichkeiten gehen gerne mal durch einander.

Vor allem Continuous Delivery und Deployment (können gleich abgekürzt werden).

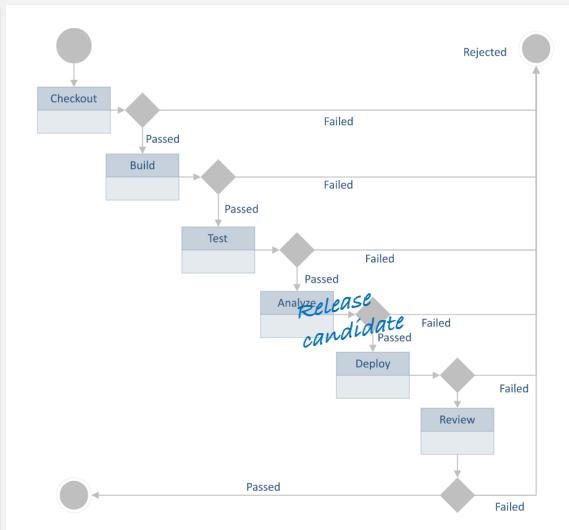
DIE CONTINUOUS DELIVERY PIPELINE

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.“

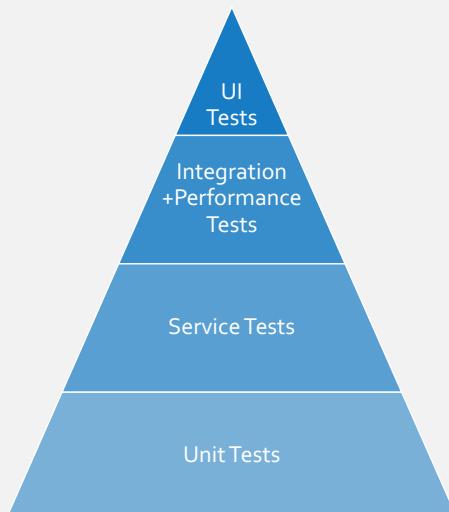
Nach M. Fowler



TEST-PYRAMIDE

Sofortiges Feedback bei Fehlern

- **Unit Tests:** Die „klassischen“ Unit Tests (z.B. Junit, Mockito) testen einzelne Komponenten isoliert.
- **Service Tests:** Tests eines einzelnen Microservices, inkl. der REST-Controller und Client-Calls (z.B. mittels JUnit, Spring MVC Tests). Mocks anderer Microservices sind notwendig (z.B. mittels Wiremock)
- **Integration Tests:** Testet die Integration mehrerer Services und deren Interaktion (z.B. mittels 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!

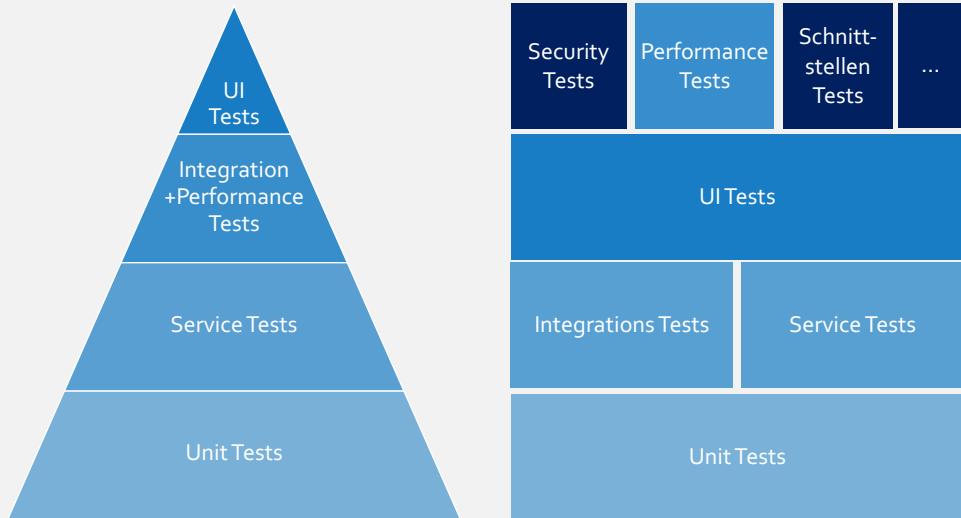
Bei aufwändigen Tests, wird man dies auf ggf. einmal pro Tag, pro Woche, o. ähnlich reduzieren.

TEST-PYRAMIDE

Es gibt nicht die „eine“ Test-Pyramide

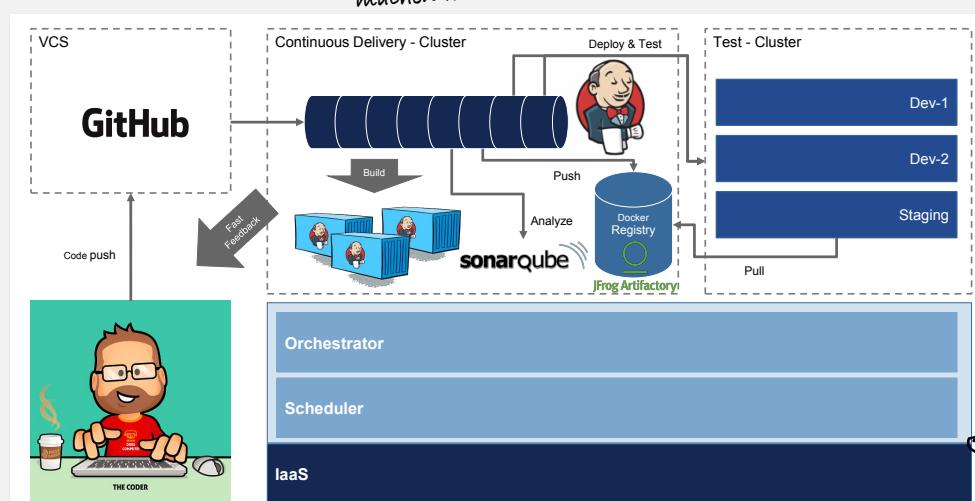
Eine Teststrategie könnte auch so aussehen ...

Der Anspruch sollte aber immer sein, dass alle Testebenen gut automatisierbar konzipiert werden.



BEISPIEL EINER CONTINUOUS DELIVERY PIPELINE

Wie man beliebige Pipelines definiert,
machen wir im Praktikum zu dieser Unit!



Wie man
automatisiert
Testcluster
aufsetzen
kann,
dazu mehr in
Unit 3.

INHALTE

- DevOps
- DevOps Prinzipien
- DevOps-Cycle konforme Architekturen und Umgebungen
- Continuous Begriffe
- **Deployment Pipelines**

EVERYTHING

AS C<>DE

Build-as-Code

Beschreibt wie Anwendungen (oder Komponenten davon) gebaut werden. Z.B.:

- Maven
- Gradle
- ...

Kennen Sie aus dem BA-Studium.

Test-as-Code

Beschreibt wie das Projekt getestet wird. Z.B.:

- Unit-, Component-, Integrationsteste
- API-, UI-Teste
- Performance Teste

Kennen Sie aus dem BA-Studium.

*Alles was die
Continuous
Delivery
umgebung mit
Leben füllt
kommt aus dem
VCS*

Infrastructure-as-Code

Beschreibt wie Laufzeitumgebungen (Deployment Environments) automatisiert aufgebaut werden. Z.B.:

- Docker
- Terraform, Vagrant, Ansible
- ...

*Kommt in
unit 03*

Pipeline-as-Code

Beschreibt alle automatisiert ablaufenden Schritte bis zur lauffähigen Installation. Z.B.:

- Build-Pipeline per Jenkins
- Gitlab CI
- ...

*Machen wir in
dieser Unit und im
Praktikum.*

BEISPIEL


GitLab
GitLab Pipelines

Pipelines

Bestehen aus Stages und Jobs

- Stages (Phasen) definieren wann etwas zu tun ist (erst komplizieren, dann testen)
- Jobs beschreiben, was zu tun ist (komplizieren, testen).

Pipelines werden als Textdatei notiert und stehen versioniert im Repository.

```
.gitlab-ci.yaml
```

stages:
-Build
-Test
-Staging

build:
 stage: build
 script: make build dependencies

test 1:
 stage: Test
 script: make test1

test 2:
 stage: Test
 script: make test2

auto-deploy-machine:
 stage: Staging
 script: terraform apply

pipeline as code

Schon genutzt?

GitLab Pipelines können Sie schon immer im GitLab TTL Service nutzen.

PROF.DR.
NANE KRATZKE

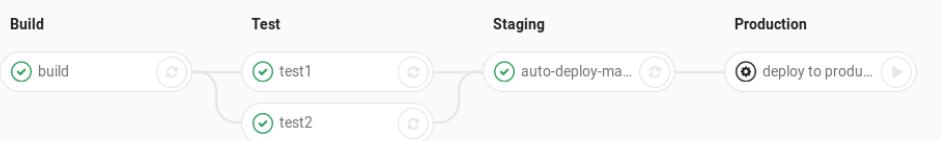
35

35

GITLAB PIPELINES


GitLab
Nur Versuch macht kluch ...

Klonen Sie dieses Repository:
`git clone https://git.mylab.th-luebeck.de/cloud-native/lab-gitlab.git`



Gitlab Pipelines

- > Übung 1: Definieren von Deployment Pipelines
- > Übung 2: Weiterreichen von Job Erzeugnissen (Artifacts)
- > Übung 3: Daten mittels Umgebungsvariablen in Pipelines geben
- > Übung 4: Nutzung von Job-spezifischen Images

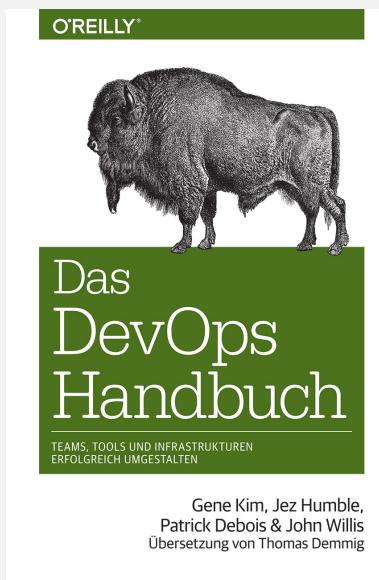
TECHNISCHE
HOCHSCHULE
LÜBECK

PROF.DR.
NANE KRATZKE

36

36

ZUM NACHLESEN



Teil I: Die drei Wege

1. Agile, Continuous und die Drei Wege
2. Der Erste Weg: Die Prinzipien des Flow
3. Der Zweite Weg: Die Prinzipien des Feedbacks
4. Der Dritte Weg: Die Prinzipien des kontinuierlichen Lernens und Experimentierens

Teil III: Der 1. Weg: Die technischen Praktiken des Flow

9. Die Grundlagen für eine Deployment Pipeline legen
10. Schnelles und zuverlässiges automatisiertes Testen ermöglichen
11. Continuous Integration ermöglichen
12. Releases automatisieren und ihr Risiko reduzieren
13. Eine Architektur für risikoarme Releases

Teil IV: Der 2. Weg: Die technischen Praktiken des Feedbacks

19. Telemetriedaten erzeugen, um Probleme zu erkennen und zu beheben
20. Telemetriedaten analysieren, um Probleme besser vorherzusehen und Ziele zu erreichen
21. Feedback ermöglichen, sodass Entwicklung und Operations Code sicher deployen können
22. Hypothesengetriebene Entwicklung und A/B-Tests in die tägliche Arbeit integrieren
23. Review- und Koordinationsprozess schaffen, um die Qualität der aktuellen Arbeit zu verbessern

KONTAKT

Disclaimer

Nane Kratzke ☎ +49 451 300-5549
✉ nane.kratzke@th-luebeck.de
✉ kratzke.mylab.th-luebeck.de

