

the  native web.

DevOps

# About me

- ◇ Michael Scherer
- ◇ Softwareentwickler
- ◇ Lead Cloud Development bei **the native web GmbH**
- ◇ Mit dem Thema DevOps seit ca. 5 Jahren beschäftigt
- ◇ Momentan hauptsächlich Einführung von DevOps- und Cloudstrategien bei einem Kunden
- ◇ Immer neugierig darauf, was sich jenseits des Tellerrandes befindet

# Agenda

- ◊ Geschichte und Motivation
- ◊ Prinzipien
- ◊ DevOps ist nicht...
- ◊ Culture
- ◊ Automation
- ◊ Lean
- ◊ Measurement
- ◊ Sharing
- ◊ DevOps in traditionellen Firmen

# Geschichte & Motivation

# Geschichte

◇ 1980er und 1990er:

Schlanke und ganzheitliche Organisation (Lean Management)

◇ um das Jahr 2000:

Agile Entwicklungsmethoden (Extreme Programming, Scrum)

◇ 2006:

Amazon startet Amazon Web Services (**AWS**)

◇ 2009:

Erste **Devopsdays** Konferenz (Patrick Debois)

# Motivation

## Wahrgenommene Probleme

- ◇ Silos
- ◇ Manueller Overhead
- ◇ Kommunikation
- ◇ Unterschiedliche Ziele
- ◇ Vertrauen

# Motivation

## Entwicklung einer *DevOps Kultur*

- ◊ Verbessern der Zusammenarbeit zwischen den Teams zur...
  - ◊ Entwicklung (*Dev*) und
  - ◊ Betrieb (*Ops*)...eines Softwareprodukts
- ◊ Nicht ausschließlich auf diese beiden Teams beschränkt
- ◊ Agile Prinzipien im ganzen Unternehmen
- ◊ Ergebnis:
  - ◊ Schnellere Auslieferung der Software
  - ◊ Höhere Stabilität und weniger Fehler
  - ◊ Weniger administrativer Aufwand

# Optimierungen

Hauptsächlich 3 Bereiche:

- ◇ Interaktionen
- ◇ Prozesse
- ◇ Werkzeuge

Iterativer Prozess

Basiert auf *Vertrauen*. Wird mit jeder Iteration verstärkt.



# Optimierungen

## Interaktionen

Hauptsächlich Verbesserung der Kommunikation

- ◊ Bessere Verbreitung von Wissen
- ◊ Unterschiede als Chance
- ◊ Gemeinschaftsgefühl
  - ◊ Innerhalb einer Gruppe
  - ◊ Zwischen verschiedenen Gruppen
- ◊ Transparenz
- ◊ Vorbild: Agile Entwicklungsmethoden

# Optimierungen

## Prozesse

- ◇ Sinn verständlich machen
- ◇ Möglichst schlank
- ◇ Automatisierung
  - ◇ Beschleunigung
  - ◇ Reduzierung von Fehlern

# Optimierungen

## Werkzeuge

- ◇ Unterstützen bei Interaktionen und Prozessen
- ◇ Nutzen und Probleme abwägen (für *alle* Beteiligten)
- ◇ Immer wieder neu bewerten
- ◇ Möglichst gleiche Werkzeuge für alle
  - ◇ Aber: Unterschiedliche Anforderungen berücksichtigen

# Prinzipien

# Prinzipien

Es gibt keine verbindlichen Prinzipien

Aber: Bestimmte Begriffe werden häufiger zu Erklärung verwendet

- ✧ The Three Ways

- ✧ CALMS

# The Three Ways

Aus dem Bereich des Lean Management abgeleitet

Ziel: Möglichst kurze Deployment Lead Time

"The Phoenix Project" von Gene Kim, Kevin Behr, George Spafford

# The Three Ways

## **The First Way: The Principles of Flow**

Technologischer Value Stream: Arbeit fließt von der Entwicklung in den Betrieb

- ◇ Arbeit sichtbar machen (z.B. globales Kanban-Board)
- ◇ Work in Progress begrenzen
- ◇ Arbeitspakete reduzieren
- ◇ Anzahl der Übergaben reduzieren

# The Three Ways

## **The Second Way: The Principles of Feedback**

Schnelles Feedback ist bei der Arbeit mit komplexen Systemen wichtig

- ◇ Informationen werden in allen Arbeitsschritten gesammelt
- ◇ Beschleunigung durch Automatisierung der Build-, Integrations- und Testprozesse
- ◇ Keine langen Freigabe-Ketten; Stattdessen z.B. Peer-Reviews
- ◇ Monitoring des Produktivsystems
- ◇ Alle an Problemlösung beteiligen



# The Three Ways

## **The Third Way: The Principles of Continual Learning**

Schaffung einer Kultur des ständigen Lernens

- ◇ Vorschläge dürfen nicht ignoriert werden
- ◇ Fehler und das Hinweisen auf Probleme dürfen keine negativen Folgen haben
- ◇ Zeit reservieren, um die Arbeitsabläufe zu verbessern
- ◇ Stresstests des Produkts; Fehler finden, bevor der Kunde es tut
- ◇ Neues Wissen in der Organisation verbreiten

# CALMS

Grundwerte, auf denen DevOps aufbaut

Geprägt von Damon Edwards und John Willis  
Von Jez Humble um das L erweitert

- ◊ Culture
- ◊ Automation
- ◊ Lean
- ◊ Measurement
- ◊ Sharing

DevOps ist nicht...

# ...kein Werkzeug

- ◇ Aber: Werkzeuge können bei der Entwicklung einer DevOps-Kultur helfen
- ◇ Entscheidend ist, *wie* ein Werkzeug verwendet wird

# ...kein Team

- ◊ Und auch kein Job
- ◊ Kein Vermittler zwischen Entwicklung und Betrieb
- ◊ Die Kompetenzen bleiben bei den jeweiligen Teams
- ◊ Die Zusammenarbeit soll optimiert werden
- ◊ Gemeinsames Team zur Einführung für erste Projekte möglich

# ...nicht abschätzbar

- ◇ Einzelne Verbesserungen sind messbar
- ◇ Aber: Eine Änderung der Kultur kann nicht zeitlich geplant oder abgeschätzt werden
- ◇ Iterativer Prozess
- ◇ Geschwindigkeit individuell unterschiedlich

# ...kein Vorgehensmodell

- ◇ Kein festes Regelwerk
- ◇ Alles ist "DevOps", was Geschwindigkeit und Qualität der Produktion verbessert

# ...nicht nur für Start-ups und Internetfirmen

- ◊ Diese Firmen spüren den Druck zur Optimierung besonders stark:
  - ◊ Kurze Release-Zeiten sind ein entscheidender Vorteil gegenüber der Konkurrenz
  - ◊ Fehler betreffen sofort einen großen Teil der Kunden
- ◊ Aber: Höhere Geschwindigkeit und Qualität sind immer positiv



# Culture

# Culture

- ◊ Sozialer Aspekt einer Organisation
- ◊ Raum für ständiges Lernen schaffen
- ◊ Kommunikation
- ◊ Vertrauen
- ◊ Oft vernachlässigt

# Dokumentation

## Situation

- ✧ Entwickler leitet Informationen an Dokuabteilung weiter
- ✧ Die Informationen werden in proprietäres Tool eingepflegt und das Ergebnis dem Entwickler zur Korrektur gegeben
- ✧ Solange wiederholt, bis alle zufrieden...

# Dokumentation

## Änderungen

- ◇ Neues gemeinsames Datenformat: Markdown
- ◇ Versionierung und Kollaboration über GitHub
- ◇ Automatisierung des Publishing
- ◇ Gleiche Tools für lokale Arbeit und Publish-Pipeline (Docker)

# Dokumentation

## Ergebnis

- ◇ Schnelleres Einpflegen von neuen Informationen
  - ◇ Keine Konvertierung notwendig
  - ◇ Alle können Änderungen vornehmen
- ◇ Bessere Zusammenarbeit zwischen Entwicklung und Dokumentation
- ◇ Einbeziehung von Dritten (z.B. Support, Kunden)
- ◇ Aber: Kontrolle bleibt bei Dokuteam!
- ◇ Beispiel: <https://docs.microsoft.com/de-de/dotnet/>

# GitHub

## Situation

- ◇ Codeverwaltung über Subversion oder CVS
- ◇ Tägliche Arbeit stockt oft, da viele Aktionen eine Verbindung mit dem SVN-Server erfordern
- ◇ Erstellen und Mergen von Branches kompliziert, daher...
  - ◇ Gemeinsame Entwicklung im Hauptbranch
  - ◇ Branching nur bei Release *durch Experten*

# GitHub

## Änderungen

- ◇ Umstellung auf Git
  - ◇ Branchen und Mergen wird von allen beherrscht und täglich durchgeführt
  - ◇ Jede Entwicklung findet ungestört im eigenen Branch statt
  - ◇ Hauptbranch kann *immer* released werden
- ◇ Wichtigste Änderung:  
Umzug auf **GitHub**

# GitHub

## Ergebnis

- ◊ Keine gegenseitige Störung bei paralleler Entwicklung
- ◊ GitHub bietet **Pull Requests**:
  - ◊ Möglichkeit, über Änderungen am Code zu diskutieren
  - ◊ Persistente, nachvollziehbare Kommunikation
- ◊ **Beispiel**
- ◊ GitHub macht es einfach, Projekte in Open Source umzuwandeln



# GitHub

## Anti-Pattern

- ◊ Gitolite

- ◊ Günstige Alternative: **GitLab**

# Automation

# Automation

- ◊ Beschleunigt Prozesse
- ◊ Vermeidet Fehler
- ◊ Immer betrachten: Ist Gewinn größer als Aufwand?
- ◊ Auf Wartbarkeit achten
- ◊ Ständige Suche nach Möglichkeiten

# Build- & Testinfrastruktur

## Situation

- ◇ Der Hauptteil der Tests wird vor Release manuell durchgeführt – *oft nur auf einer Plattform*
- ◇ Es werden keine neuen VMs verwendet, sondern "bereits eingerichtete"
- ◇ Erst *nach* den Tests werden die endgültigen Installationspakete für die Kunden gebaut
- ◇ Code wird während des Release-Vorgangs weiterhin geändert, aber *nicht neu getestet*
- ◇ Dauer eines Releases: Mehrere Monate

# Build- & Testinfrastruktur

## Änderungen

- ◊ Automatisches Erzeugen der *endgültigen* Installationspakete
- ◊ QS arbeitet *nur* mit diesen Paketen
- ◊ Anteil der automatisierten Tests drastisch erhöht
  - ◊ jMeter
  - ◊ Cucumber
  - ◊ Selbst geschriebene Tests
- ◊ Bei jedem Durchlauf werden neue Tests-VMs erzeugt
- ◊ Automatisierung mittels **Vagrant** und **Terraform**
- ◊ Alle unterstützten Plattformen werden getestet
- ◊ Sowohl Cloud als auch interne Systeme werden verwendet

# Build- & Testinfrastruktur

## Ergebnis

- ◇ Automatischer Teil des Releasevorgangs innerhalb von Stunden vollständig durchlaufen
- ◇ Nutzung der Cloud macht automatisierten Testaufbau sehr einfach
- ◇ Sicherheit so hoch, dass bereits ins firmeneigene Produktivsystem deployed wird
- ◇ Viel Zeit für Entwicklung, aber auch viel Einsparung

# Build- & Testinfrastruktur

## Anti-Pattern

- ◇ Testinfrastruktur muss nicht unbedingt selbst bereitgestellt werden
  - ◇ Gerade für kurzlebige Testdurchläufe kann die Cloud günstig sein
  - ◇ Cloud besonders für Tests geeignet, die die Ressourcen stark belasten
  - ◇ Automatisierung mittels Terraform u.ä. sehr einfach
  - ◇ Oft wird der Wartungsaufwand bei eigenen Lösungen unterschätzt

# Lean



# Lean

- ◇ Schlanke Prozesse
- ◇ Globale Sicht: Auch Übergabe zwischen Teams betrachten
- ◇ Auch mit Hilfe von Automatisierung
- ◇ Wirkt sich auf den Aufbau der Software aus

# The Twelve-Factor App

**Best practices** für die Entwicklung von Software as a Service (SaaS)

- ◇ Nicht nur geeignet für das Deployment in die Cloud
- ◇ Möglichst reibungsloser Übergang zwischen Entwicklung und Betrieb (Continuous Deployment)
- ◇ Einfache Skalierung im laufenden Betrieb

# The Twelve-Factor App

## Die 12 Faktoren

### 1. Codebase

Eine im Versionsmanagementsystem verwaltete Codebase, viele Deployments

### 2. Abhängigkeiten

Abhängigkeiten explizit deklarieren und isolieren

### 3. Konfiguration

Die Konfiguration in Umgebungsvariablen ablegen

### 4. Unterstützende Dienste

Unterstützende Dienste als angehängte Ressourcen behandeln

# The Twelve-Factor App

## Die 12 Faktoren

### 5. Build, release, run

Build- und Run-Phase strikt trennen

### 6. Prozesse

Die App als einen oder mehrere Prozesse ausführen

### 7. Bindung an Ports

Dienste durch das Binden von Ports exportieren

### 8. Nebenläufigkeit

Mit dem Prozess-Modell skalieren

# The Twelve-Factor App

## Die 12 Faktoren

### 9. Einweggebrauch

Robuster mit schnellem Start und problemlosen Stopp

### 11. Logs

Logs als Strom von Ereignissen behandeln

### 10. Dev-Prod-Vergleichbarkeit

Entwicklung, Staging und Produktion so ähnlich wie möglich halten

### 12. Admin-Prozesse

Admin/Management-Aufgaben als einmalige Vorgänge behandeln

# The Twelve-Factor App

... und CALM(S)

## Culture

- 1. Codebase
- 5. Build, release, run
- 6. Prozesse

## Automation

- 2. Abhängigkeiten
- 3. Konfiguration
- 4. Unterstützende Dienste
- 9. Einweggebrauch
- 12. Admin-Prozesse

## Lean

- 8. Nebenläufigkeit

## Measurement

- 7. Bindung an Ports
- 10. Dev-Prod-Vergleichbarkeit
- 11. Logs

# API-Schnittstellen

## Situation

- ◇ Ein System besteht aus Prozessen, die von verschiedenen Teams entwickelt werden
- ◇ Keine einheitliche Schnittstelle zwischen den Komponenten
  - ◇ Historisch gewachsen: Von selbst erfundenen Protokollen bis zu SOAP
- ◇ Keine konsistente Dokumentation dieser Schnittstellen
- ◇ Nur die *Experten* können Änderungen vornehmen

# API-Schnittstellen

## Änderungen

- ◊ Einführung von REST, wo möglich
- ◊ Schulung der Mitarbeiter, so dass *jeder* mit REST vertraut ist
- ◊ Definition der Schnittstellen via **OpenAPI**
- ◊ Dafür wird **Swagger** eingesetzt
- ◊ Änderungen werden durch die Codeverwaltung versioniert



# API-Schnittstellen

## Ergebnis

- ◇ HTTPS anstelle von unsicheren, selbst entwickelten Formaten
- ◇ Verbreiterung der Wissensbasis: Jeder kann die Spezifikation verstehen
- ◇ Spezifikation dient auch Kunden als Quelle für eigene Anbindungen
- ◇ Automatische Code-Generierung möglich

# API-Schnittstellen

## Anti-Pattern

- ◊ Nicht in jedem Fall ist REST geeignet
  - ◊ z.B. zu hohe Last durch Polling der Clients
- ◊ Aber: Immer prüfen, ob die Voraussetzungen geschaffen werden können

# Measurement

# Measurement

- ◇ Nur, was man messen kann, kann man optimieren
- ◇ Basis für Entscheidung, ob Erfolg
- ◇ Betrifft Prozesse, aber auch das Produkt selbst
- ◇ Möglichst schnelles Feedback
- ◇ Benötigte Metriken können sich mit der Zeit ändern

# Logging & Monitoring

## Situation

- ◇ Monitoring des Produkt ist wichtig
  - ◇ Erfolg von Änderungen
  - ◇ Kritische Situationen erkennen
- ◇ Logdaten können wertvolle Hinweise liefern
- ◇ Aber: Meist nur für Post-Mortem Analysen verwendet
  - ◇ Dateizugriff ist oft nur schlecht möglich
  - ◇ Bei verteilten Systemen weit verstreut

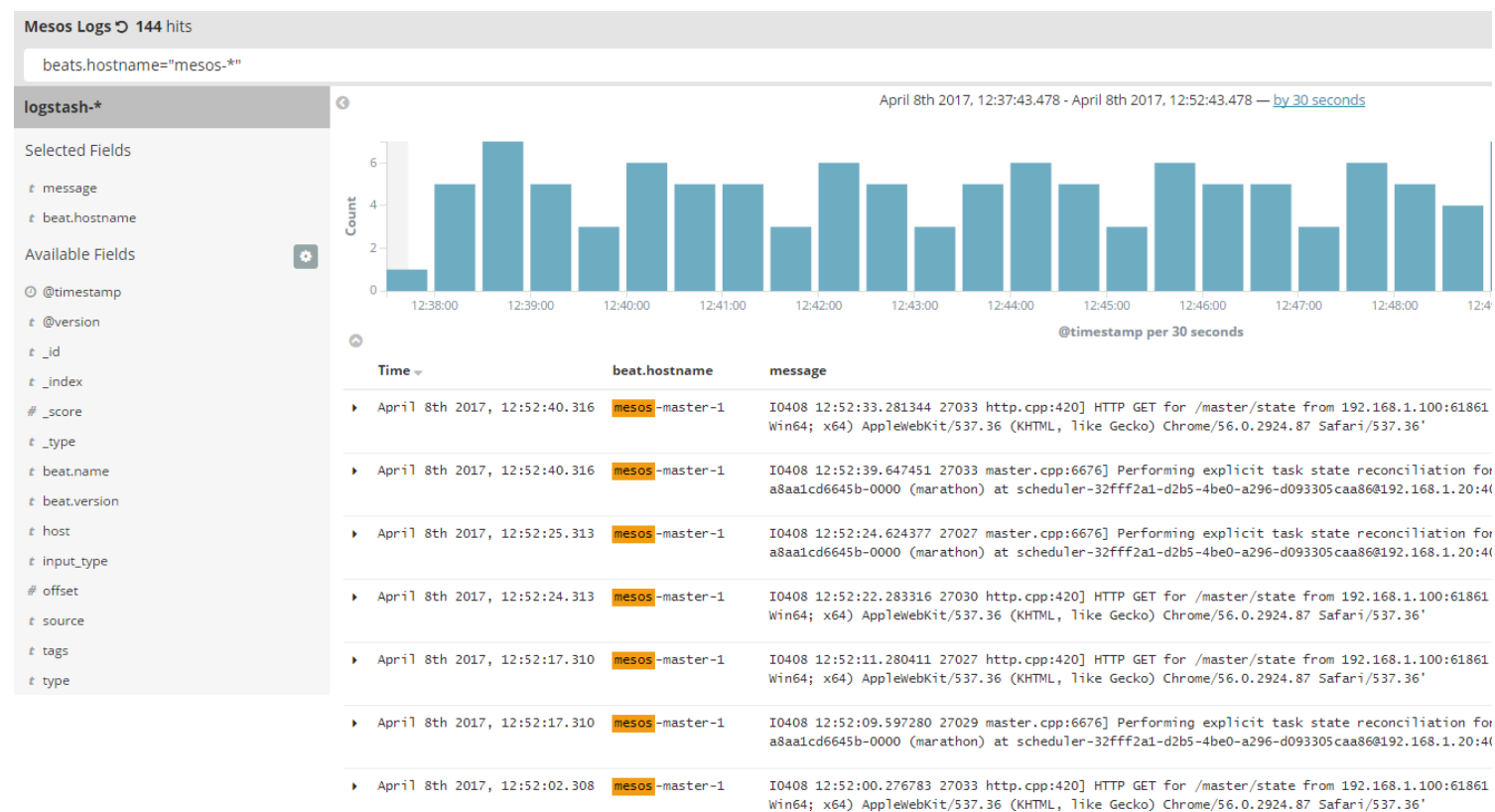
# Logging & Monitoring

## Änderungen

- ◇ Sammeln aller Logdaten an einer zentralen Stelle
- ◇ **ELK-Stack** (Elasticsearch, Logstash, Kibana)
- ◇ Auch als SaaS verfügbar

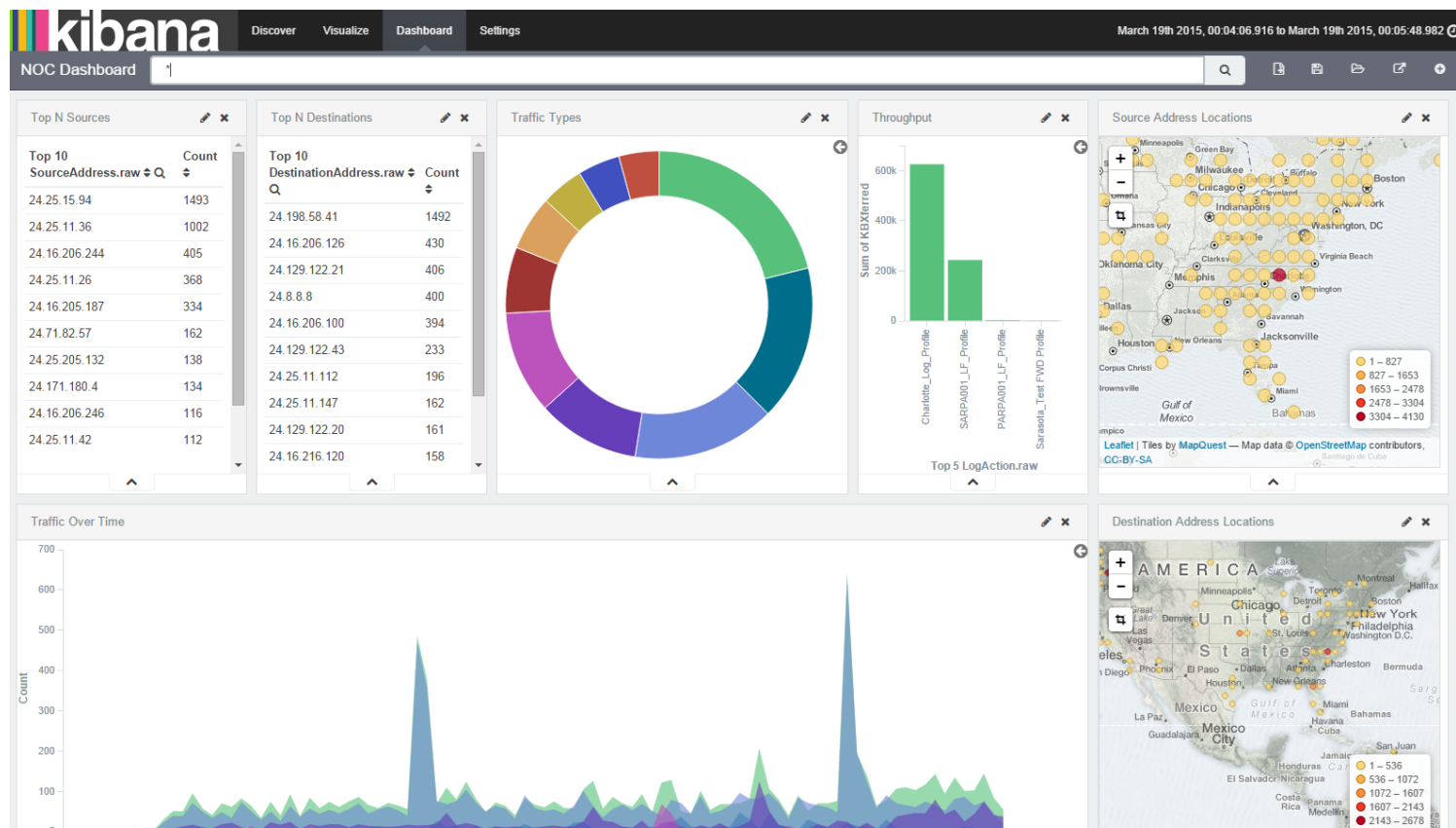
# Logging & Monitoring

## Kibana - Log View



# Logging & Monitoring

## Kibana - Dashboard





# Logging & Monitoring

## Ergebnis

- ◇ Besseres Monitoring ohne Änderung des Programms!
- ◇ Auswertung in Echtzeit
  - ◇ Frühere Reaktion auf Fehlersituationen möglich
- ◇ Neue Auswertungen einfach durch Schreiben zusätzlicher Daten ins Log

# Logging & Monitoring

## Anti-Pattern

- ✧ Ausfall des zentralen Loggings darf nicht das System zum Stillstand bringen

# Funktionale Tests

## Situation

- ◊ Black-Box-Tests
- ◊ Häufig manuell
- ◊ Wenn automatisiert, schwer zu verstehen da sehr implizit und technisch

# Funktionale Tests

## Änderungen

- ◊ Cucumber
- ◊ Gemeinsame Festlegung von Tests (PO, QS, Dev, Ops)
- ◊ Docker für den schnellen Aufbau der Anwendung
  - ◊ Tests können sich nicht beeinflussen
  - ◊ Notwendige Anpassungen an Konfiguration wird explizit

# Funktionale Tests

## Beispiel für einen Cucumber-Test

```
return_to_stock.feature
1 Feature: Returns go to stock
2   In order to keep track of stock
3   As a store owner
4   I want to add items back to stock when they're returned
5
6   Scenario: Refunded items should be returned to stock
7     Given a customer previously bought a keyboard from me
8     And I currently have three keyboards left in stock
9     When he returns the keyboard for a refund
10    Then I should have four keyboards in stock
11
12   Scenario: Replaced items should be returned to stock
13     Given that a customer buys a white telephone
14     And I have two white telephones in stock
15     And three black telephones in stock.
16     When he returns the telephone for a replacement in black
17     Then I should have three white telephones in stock
18     And two black telephones in stock
19
```

# Funktionale Tests

## Ergebnis

- ◊ Nicht nur für Experten, sogar für Kunden geeignet
- ◊ Dient als Diskussionsgrundlage für neue Features
- ◊ Änderungen im System wirken sich nicht auf Beschreibung der Testfälle aus
- ◊ Trennung der Verantwortlichkeiten
  - ◊ QS für Beschreibung der Testfälle
  - ◊ Dev für Testcode

# Funktionale Tests

## Anti-Pattern

- ✧ Eine (Programmier-)Sprache für alle
- ✧ Auf richtige Abstraktion der Beschreibungen achten
- ✧ Zu technische Beschreibungen vermeiden

Der Kunde muss es verstehen können

# Erfolg messen

**Nur was messbar ist, kann man optimieren**

## ◇ Produkt

Wächst die Kundenzufriedenheit?

## ◇ Interaktionen

Gibt es mehr Austausch zwischen Mitarbeitern und Teams?

## ◇ Prozesse

Wird der Prozess beschleunigt?

Wird er stabiler?

## ◇ Werkzeuge

Welche Probleme löst das Werkzeug?

Welche neuen Probleme kommen dazu?



# Erfolg messen

- ◊ Oft unterschiedliche Definitionen von Erfolg
- ◊ Notwendig:
  - ◊ Gemeinsames Ziel
  - ◊ Transparenz

# Blameless Post-Mortem

## Umgang mit Misserfolg

- ◊ Fehler können nicht ausgeschlossen werden
- ◊ Basis für das Lernen aus Fehlern
- ◊ Stattdessen häufig:
  - ◊ Suche nach einem Schuldigen
  - ◊ Überhaupt keine Analyse
- ◊ Gewonnene Erkenntnisse müssen verbreitet werden

# Sharing

# Sharing

Intern:

- ◊ Chat Channels
- ◊ Lightning Talks
- ◊ Open Space
- ◊ Interessengruppen

Extern:

- ◊ Usergruppen / Meetups
- ◊ Konferenzen

# DevOps in traditionellen Firmen

# DevOps in traditionellen Firmen

- ◊ Keine Start-ups oder Internetfirmen
- ◊ Aber trotzdem Veränderungen:
  - ◊ Vorteile durch Nutzung neuer Technologien bei Erstellung und Betrieb von Software
  - ◊ Kunden fangen an, die Cloud zu nutzen
- ◊ Beschleunigung von Prozessen und Erhöhung der Produktqualität
- ◊ Zeit nötig für Änderung der Kultur, gerade bei gefestigten Strukturen

# Zusammenfassung

- ◊ DevOps ist eine Unternehmenskultur der Zusammenarbeit und der konstanten Optimierung
- ◊ DevOps ermöglicht schnelle Reaktionen auf Veränderungen
- ◊ Nicht von Zahlen beeindruckt lassen: DevOps ist auch sinnvoll, wenn nicht alle paar Minuten deployed wird
- ◊ Cloud ist Herausforderung aber auch *die* Gelegenheit für Einführung von DevOps
- ◊ Auch jenseits davon: Schnelle Prozesse und hohe Produktqualität sind immer wichtige Ziele

Vielen Dank. Many thanks. Merci beaucoup.  
Muchas gracias. Molte grazie. Hartelijk bedankt.  
Tack så mycket. çok teşekkür ederim. большое  
спасибо. 非常感谢. どうもありがとう. شكرا  
جزیلا.

✉ hello@thenativeweb.io | 🐦 thenativeweb | 🌐 thenativeweb