architektur SPICKER

Übersichten für die konzeptionelle Seite der Softwareentwicklung

MEHR WISSEN IN KOMPAKTER FORM:
Weitere Architektur-Spicker gibt es als kostenfreies PDF unter www.architektur-spicker.de

NR. 7

Continuous Delivery

Zeitgemäße Techniken aus Continuous Integration (CI) und Continuous Delivery (CD) unterstützen wichtige Architekturziele wie Stabilität und Reaktionsfähigkeit. Dieser Spicker zeigt den Aufbau einer passenden CI/CD-Kette.

IN DIESER AUSGABE

Worum geht's? (Herausforderungen/Ziele)

- Neue Features in eure Lösung zu integrieren ist aufwändig und fehleranfällig. Wie minimiert ihr dieses Risiko?
- Moderne Architekturansätze wie Microservices haben hohe Anforderungen bzgl. Integration und Verteilung. Welche Wechselwirkungen bestehen zwischen Architekturstil und CI/CD?
- Manuelle, wiederkehrende Tätigkeiten binden Kräfte und lassen sich nicht in gleichbleibender Qualität wiederholen. Wie eliminiert das Team diese monotonen Aufgaben?
- Auswirkungen von Änderungen in Quelltext, Technologie und Konfiguration werden erst spät im Entwicklungsprozess erkannt. Wie erhaltet ihr rasch Feedback?

Die Continuous Delivery Perlenkette

CD automatisiert die Integrations- und Verteilungsprozesse von der Codierung bis zur lauffähigen Software, um schnell und verlässlich zu liefern. Die folgende Abbildung fädelt die wesentlichen Elemente zu einer CD-Kette zusammen:

1. **Version Control System**
   Code-Versionierung zur Zusammenarbeit und zur Nachvollziehbarkeit
   Implementierungen und Tools: Git, Mercurial, Subversion, ...
2. **Continuous Build**
   Build-Automatisierung zur reproduzierbaren Erstellung der Auslieferungsartefakte
   Implementierungen und Tools: Maven, Gradle, Sbt, NPM, MSBuild, Grunt, ...
3. **Continuous Testing**
   Test-Automatisierung zur Sicherstellung der Funktionen, auch im Zusammenspiel
   Implementierungen und Tools: xUnit, JBehave, Selenium, Jasmin, ...
4. **Continuous Database Integration**
   Automatisierte DB-Migration für Flexibilität auch in der relationalen Welt
   Implementierungen und Tools: Flyway, Liquibase, ...
5. **Konfigurationsmanagement**
   Automatisierung für eine schnelle Bereitstellung von Infrastruktur
   Implementierungen und Tools: Terraform, Packer, CloudFormation, ... Ansible, Puppet, Chef, ...
6. **Automatisierte Verteilung**
   Automatisierung für eine stressarme Installation der Softwareartefakte
   Implementierungen und Tools: Ansible, Salt, Skripte, ... Nomad, Kubernetes, ...

Continuous Inspection
Statische Codeanalyse für die innere Qualität (→ Spicker 2)
Implementierungen und Tools: SonarQube, SpotBugs, Checkstyle, Jacoco, JSLint, ...

Continuous Controlling/Observation
Kontinuierliches Monitoring und Simulation zur Qualitäts- und Zielüberwachung
Implementierungen und Tools: Prometheus, Icinga, Elastic Stack, Simian Army, ...

1 Versionskontrollsystem

Ein Versionskontrollsystem (VCS) dient als Basis jeder CI- (und später CD-) Umgebung. Mit ihm kehrt ein Team bei Bedarf (z.B. im Fehlerfall) zuverlässig auf einen früheren, funktionierenden Stand zurück.

BEST PRACTICES

- Das Projekt im VCS besitzt eine konsistente Ordnerstruktur.
- Source Code ist an einer zentralen/offiziellen Stelle abgelegt.
- Zum Source Code gehören: Build-Skripte, Programm-Code, Konfigurationen (zu Beginn, später im Konfigurationsmanagement), Deployment-Skripte, Datenbank-Skripte

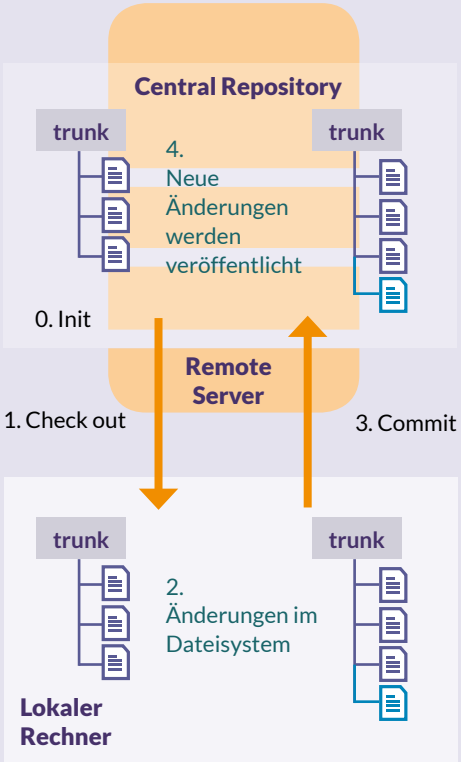
SO ARBEITEN EURE TEAMS

- Entwickler checken Code häufig ein.
- Teams entscheiden sich für einen VCS Workflow und passen ihre Arbeitsweise daran an.
- Jedes Team ist für seinen Quelltext gemeinsam verantwortlich (Collective Ownership).

Arten der Versionsverwaltung

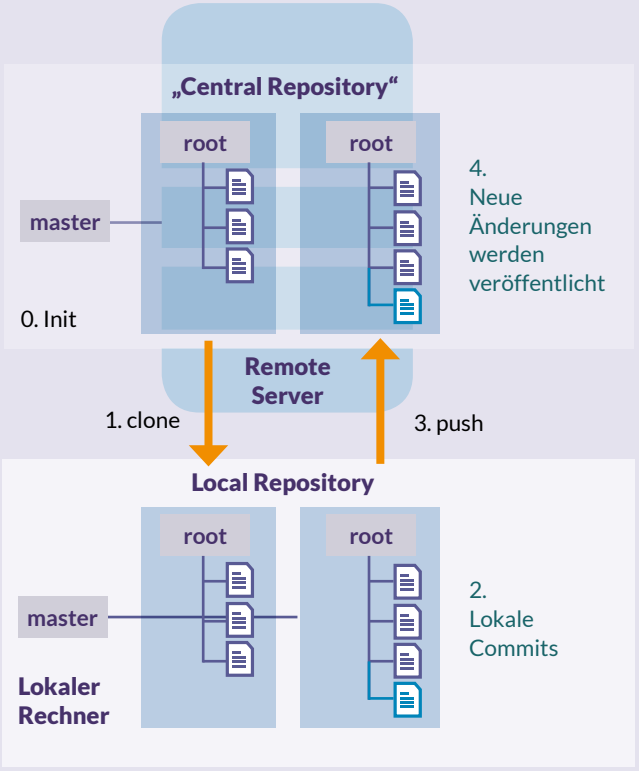
Zentrale Versionsverwaltung

Realisiert durch eine Client-Server-Architektur, wobei ein Repository mit samt seiner Historie zentral im Server abgelegt ist. Die Entwickler arbeiten mit einer Kopie des Repositories



Verteilte Versionsverwaltung

Konzeptionell existiert kein zentrales Repository mehr, sondern jeder Entwickler erhält sein eigenes Repository inklusive der gesamten Historie. Es wird aber üblicherweise ein Repository zum offiziellen Repository ernannt, von denen alle Beteiligten sich ein Repository klonen können.



2 Continuous Build

Der erste Automatisierungsschritt etabliert einen kontinuierlichen Build-Lauf. Jede Änderung im VCS zieht das Bauen nach sich. So bleibt die Software durchgängig mindestens kompilierfähig.

BEST PRACTICES

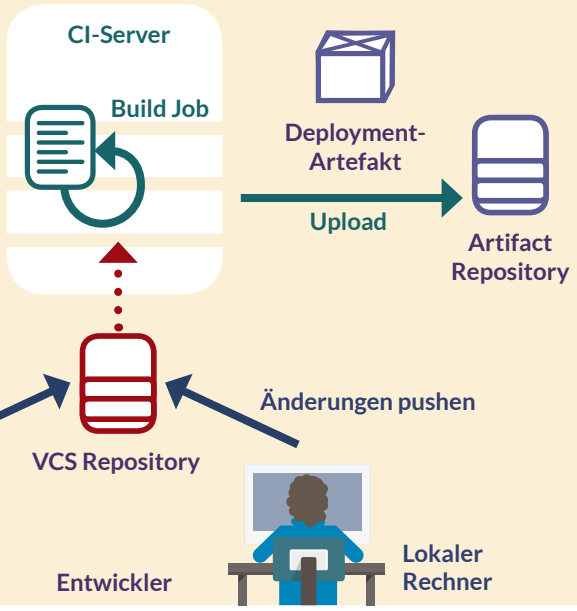
- Keine manuellen Schritte – Builds laufen automatisiert ab.
- Das Abhängigkeitsmanagement erfolgt über das Buildwerkzeug.
- Das Resultat eines Build sind fertige, umgebungsunabhängige Deployment-Artefakte.
- Alleinige Wahrheit über den Zustand des Builds ist eine dedizierte Build Integration Maschine (CI-Server) – kein „It works on my machine.“

SO ARBEITEN EURE TEAMS

- Sie fixen einen fehlgeschlagenen Build auf dem Server unverzüglich.
- Entwickler lassen den Build auch auf ihrem lokalen Rechner laufen.

Was sollte bei der Build-Automatisierung beachtet werden?

- Bevorzuge existierende Werkzeuge vor selbstgeschriebenen.
- Passe deine Ordnerstruktur in deinem VCS an das Werkzeug an.
- Das Bauen der Software ist unabh. von seiner Umgebung und von anderen Werkzeugen (auch von der IDE).
- Builds schlagen schnell fehl, wenn etwas schief läuft.
- Langsame Builds verhindern ein schnelles Feedback.
- Dauer des Build-Lauf ist angemessen zur Projektgröße.



Ob der Build-Lauf schnell genug ist hängt maßgeblich von der Architektur ab. (Microservice vs. Monolith)

3 Continuous Testing

Automatisierte und regelmäßige Tests auf mehreren Ebenen verkürzen die Feedbackzeiten zum Zustand des Systems – über das bloße Bauen hinaus.

BEST PRACTICES

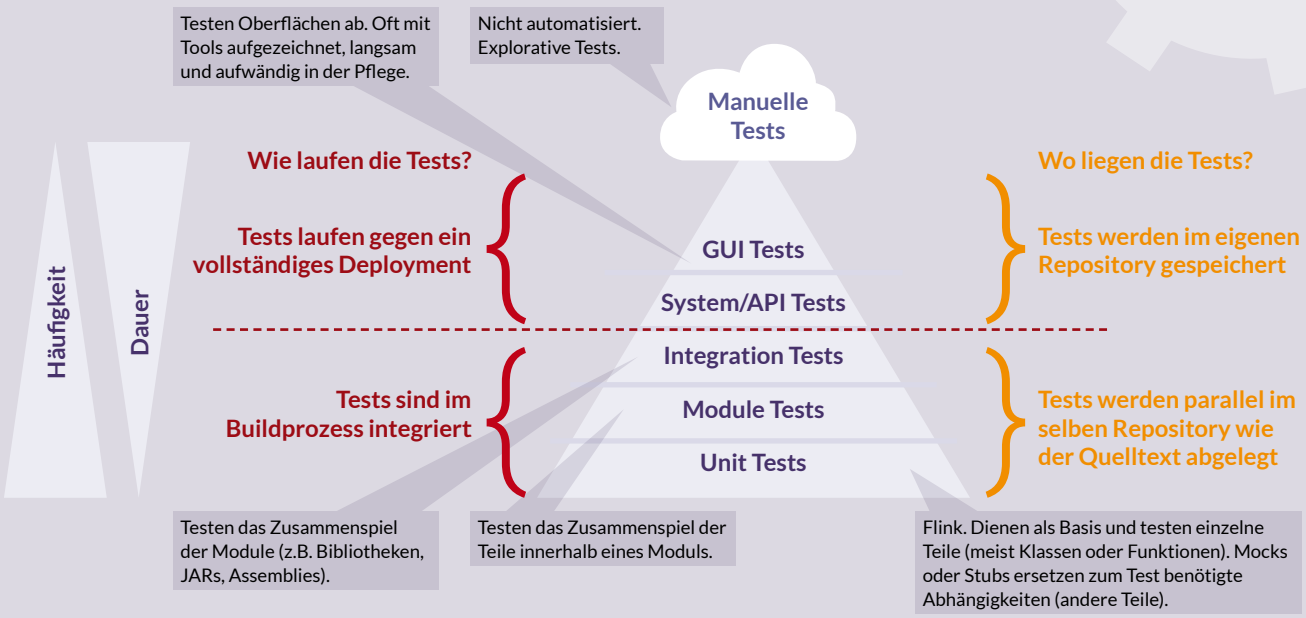
- Tests sind automatisiert, wiederholbar und laufen voneinander unabhängig.
- Tests liegen entweder in eigenen Repositories oder parallel zum Source Code.
- Tests sind kategorisiert (siehe Testpyramide).
- Für frühes Feedback laufen schnelle Tests zuerst.

SO ARBEITEN EURE TEAMS

- Entwickler lassen die Tests auch auf ihren lokalen Rechnern laufen.
- Sie schreiben automatisierte Entwicklertests und integrieren sie in den Build.
- Teammitglieder checken keine Tests ein, die fehlschlagen.
- Die Teams dokumentieren auftretende Softwarefehler anhand von Tests.

Ideale Testpyramide

Die Testpyramide bündelt Best Practices. Sie dient eurem Vorhaben als Maßstab und Zielbild für Häufigkeit (Anzahl) und Dauer (Ausführungszeit) unterschiedlicher Tests.



Ideal? Jede Testpyramide sieht anders aus. Die ideale Pyramide dient vorrangig der Kommunikation im Team.
Bei Microservices z.B. können Modul- und Integrationstest zusammenfallen und gegen ein volles Deployment laufen.

4 Continuous Database Integration

Änderungen an der Persistenzstruktur erfordern mitunter die Erneuerung der Datenbank und ihrer Daten. Eine relationale Datenbank benötigt, anders als eine NoSQL-Datenbank, hierzu explizite Migrationsschritte.

BEST PRACTICES

- Das Datenbankschema muss wiederholt aufsetzbar sein, am besten automatisiert.
- Eine Änderungshistorie dokumentiert alle ausgeführten Migrationsschritte.
- Die Testdatenbanken ähneln den Produktionsdatenbanken.
- Jede Änderung an den Datenbank-Skripten wird getestet.

SO ARBEITEN EURE TEAMS

- Jeder Entwickler hat seine eigene Datenbank.
- Niemand macht manuelle Änderungen am Datenbankschema.
- Jede Änderung erfolgt über Datenbank-Skripte.
- Die Entwickler behandeln die Datenbank-Skripte wie normalen Source-Code.
  - Sie versionieren DDL-Skripte, DML-Skripte, Konfigurationen, Testdaten und Skripte mit Stored Procedure, Functions usw. in einem VCS.
  - Sie testen die Datenbank-Skripte.

5 Konfigurationsmanagement

Das Ops in DevOps automatisiert die Provisionierung (Vorbereitung) der einzelnen Server. Dazu hält das Team die einzelnen Konfigurationen eines Servers in einem Skript fest.

➔ **Server-Provisionierung**

eine Menge an Schritten, um einen Server mit Daten und Software vorzubereiten

- Ressourcen zuweisen und konfigurieren
- Middleware installieren und konfigurieren
- Anwendungen installieren und konfigurieren

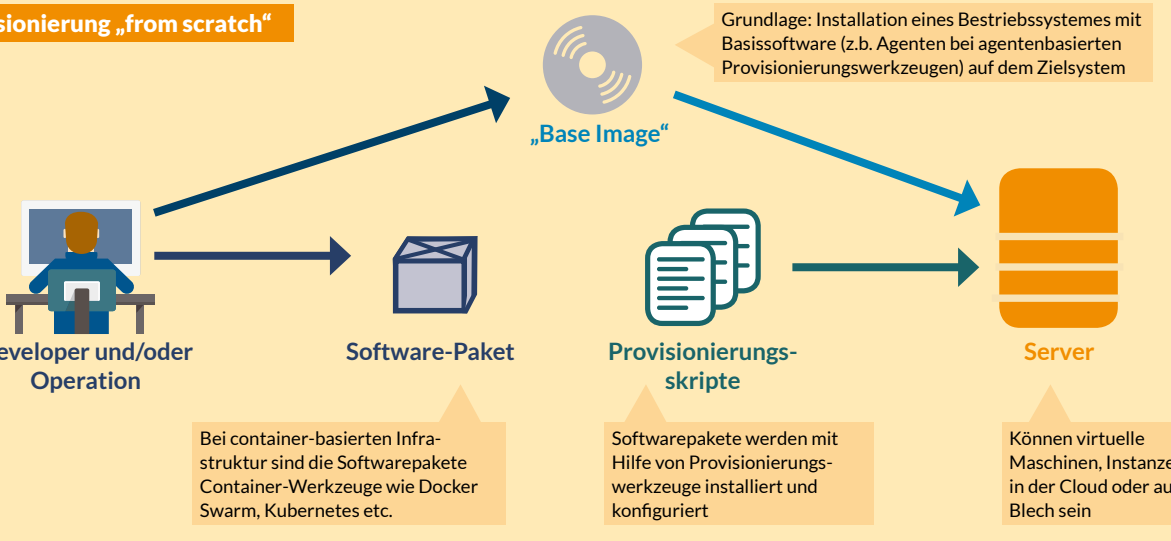
BEST PRACTICES

- Idealerweise kommt ein Provisionierungswerkzeug zum Einsatz; bei einfachen Fällen (Shell)-Skripte.
- Provisionierungsskripte sind idempotent (mehrfach hintereinander ausgeführt liefert es das gleiche Ergebnis wie bei einer einzigen Ausführung)
- Alle Änderungen erfolgen über Provisionierungs-skripte.

SO ARBEITEN EURE TEAMS

- Teammitglieder machen keine manuellen Änderungen an den Serverkonfigurationen.
- Sie behandeln ihre Provisionierungsskripte wie „normalen“ Source Code., versionieren sie in einem VCS und testen sie.

Serverprovisionierung „from scratch“



6 Automatisierte Verteilung

In diesem Schritt automatisiert das Team die Installation der Anwendung. Dazu verteilt sie die Deployment-Artifakte und die Konfigurationen auf die vorbereiteten Server.

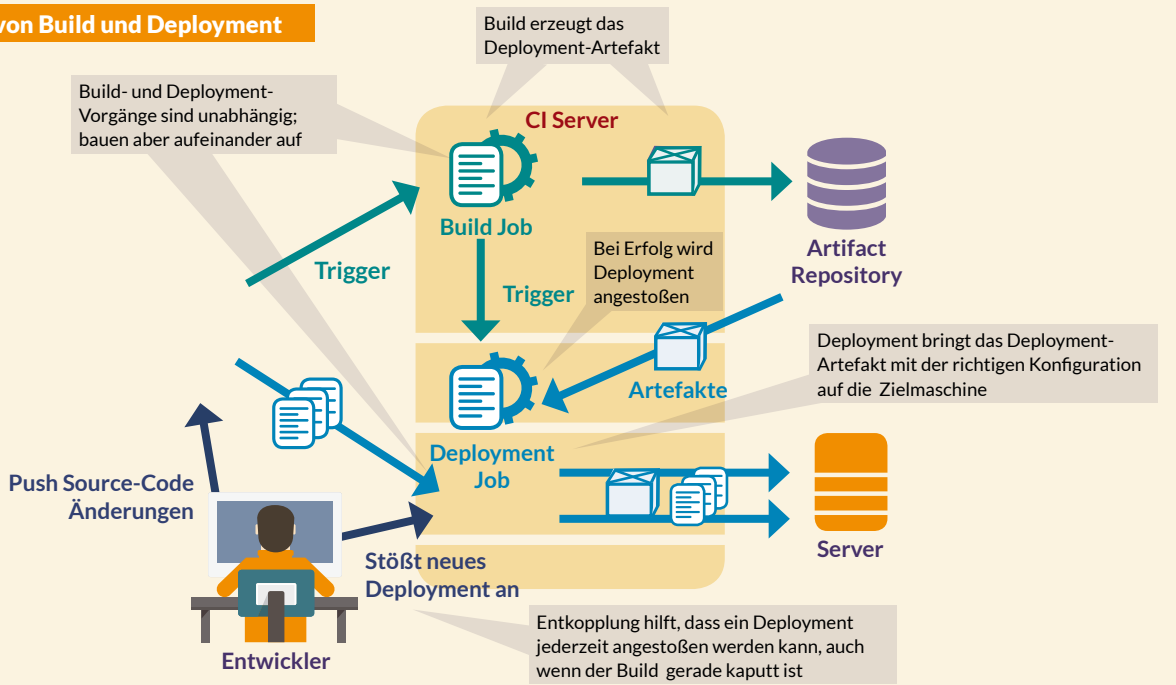
BEST PRACTICES

- Das Deployment-Artefakt basiert für alle Umgebungen (Dev, Test ... Prod) auf *einer* Code-Basis.
- Dasselbe Deployment-Artefakt wird auf allen Umgebungen verteilt.
- Jede Umgebung hat seine separate Konfiguration.
- Konfigurationen sind nicht Teil des Deployment-Artefaktes.
- Die Verteilung eines Deployment-Artefaktes inklusive der passenden Konfiguration ist automatisiert.
- Der Deploymentvorgang kann unabhängig vom Buildvorgang erfolgen.

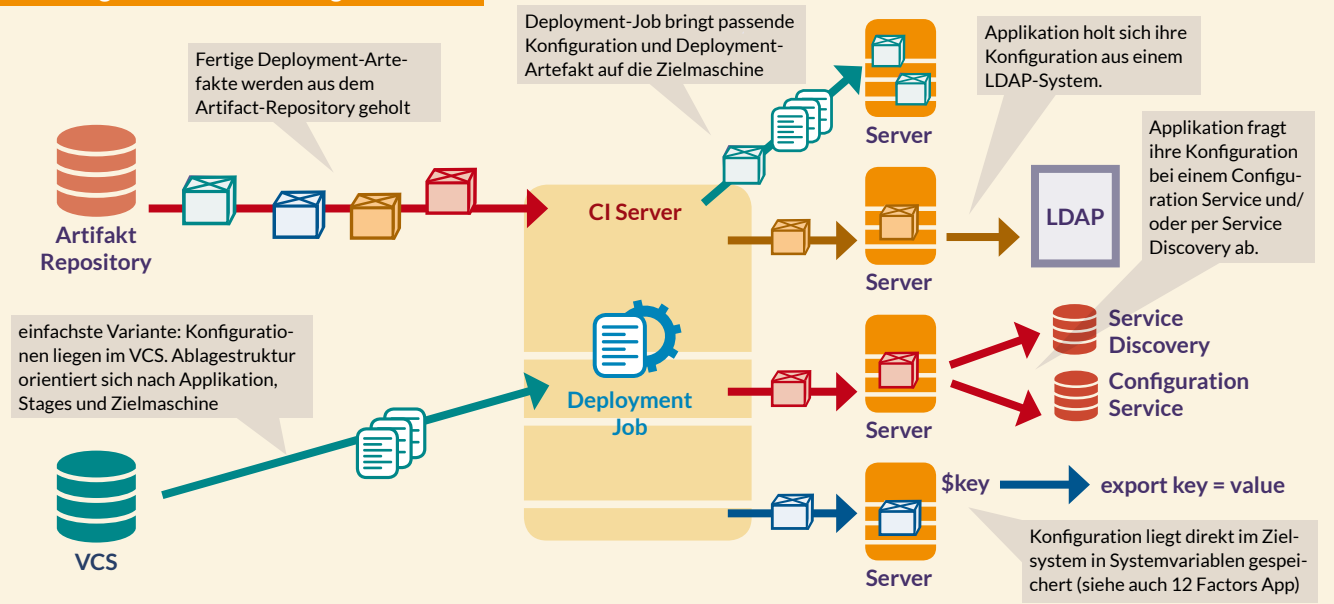
SO ARBEITEN EURE TEAMS

- Die Teams machen explizit, welche Schritte und Artefakte sie zu einem erfolgreichen Deployment benötigen.
- Sie nutzen bereits existierende Provisionierungswerkzeuge; bei einfachen Aufgaben reichen auch (Shell-)Skripte.

Trennung von Build und Deployment



Verwaltung von Software-Konfigurationen



Starthilfe

Dieser Spicker gibt viele Hinweise zu den einzelnen Perlen einer CI-Kette. Aber wo anfangen? Hier Empfehlungen zu typischen Zielen (und empfohlenen ersten Schritten) rund um Continuous Delivery als Startpunkt.

RELEASE-ZYKLUS VERKÜRZEN

- Ablauf eines Release als Wertkette (Value Stream) visualisieren, Bottlenecks und Verschwendung identifizieren, Anzahl Übergaben verringern.
- Euer Testkonzept gegen ideale Test-Pyramide halten. Abweichungen kritisch diskutieren. Manuelle Tests verringern, auf Automatisierung setzen.
- Perspektivisch: Umfang einzelner Lieferungen verringern. Teams befähigen unabhängig voneinander zu entwickeln, zu testen, zu releasen.

DOWNTIME BEIM DEPLOYMENT VERKÜRZEN

- Einzelne Arbeitsschritte im Deployment explizit machen (auch hier: Value Stream)
- Manuelle Schritte eliminieren, auch um Fehler (und damit Verzögerungen) zu vermeiden
- Perspektivisch: Deployment-Strategie anpassen (Big Bang vs. Blue Green, ggf. Cloud als Ziel), kleinteiliger deployen können

ANWENDUNG IN DIE CLOUD BRINGEN

- Zielsetzung für die Migration in „die Cloud“ herausarbeiten (z.B. mit schwankender Last besser umgehen, unterbrechungsfrei Deployen können ...) und weiteres Handeln danach ausrichten.
- Liefer- und Servicemodell für die Cloud auswählen, Top-Probleme und Hindernisse beim Migrieren (s. Spicker Nr. 5 „Cloud-Anwendungen“) mit Proof of Concept adressieren
- Fokus auf Konfigurationsmanagement und automatische Verteilung legen („Infrastructure as Code“), s. S. 4 + 5
- Perspektivisch: Evolutionäre Anwendungsarchitektur (Makro) für Cloud-Anwendungen etablieren (Stichwort Cloud-Prinzipien, 12 Factors ...)

ZUSAMMENARBEIT IN UND ZWISCHEN DEN TEAMS VERBESSERN

- Umgebungen auf Knopfdruck bereitstellen können („Self- Service“), statt auf Antragsformularbasis
- Gemeinsam genutzte Services anbieten (Repos, Build-Server, Deployment-Pipelines, Dashboards für Telemetrie, ...)
- Perspektivisch: Cross-funktionale Teams bilden, Verantwortung für Deployment und Betrieb an Entwicklungsteams geben (DevOps-Kultur)

TEAMS UNABHÄNGIG ARBEITEN LASSEN

- Zusammenarbeit zwischen Anwendungsteilen/ Teams in einer Context Map (Domain-driven Design) explizit machen.
- Test-Strategie an Abhängigkeiten zwischenTeilen/ Kollaborationen anpassen (API-Tests, Contract Testing)
- Anwendungsarchitektur fachlich und technisch loser koppeln, Deployment-Monolithen ggf. mit dem Strangler Pattern knacken.
- Perspektivisch: Team- und Anwendungsstruktur in Einklang bringen (Conway‘s Law)

Weitere Informationen

Bücher

- Gene Kim et al: The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win, IT Revolution Press 2014
- Jez Humble et al: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation, Addison-Wesley 2010
- Paul M. Duvall et al: Continuous Integration: Improving Software Quality and Reducing Risk 2007, Addison-Wesley 2007
- Gene Kim et al: The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations, IT Revolution Press 2016

Autoren dieses Spickers

- **Sandra Parsick** (info@sandra-parsick.de) berät Kunden rund um ihre Schwerpunkte Java-Enterprise-Anwendungen, Software Craftsmanship und der Automatisierung von Softwareentwicklungsprozessen.
- **Stefan Zörner** (stefan.zoerner@embarc.de) unterstützt Vorhaben und Teams dabei, passende Lösungsansätze wirkungsvoll in deren Software zu verankern.

Wir freuen uns auf Ihr Feedback: spicker@embarc.de

https://architektur-spicker.de

embarc Software Consulting GmbH
https://www.embarc.de
info@embarc.de

SIGS DATACOM
FACHINFORMATIONEN FÜR IT-PROFESSIONALS
https://www.sigs-datacom.de
info@sigs-datacom.de