OBJEKTspektrum

DevOps: Agile Software-Entwicklung und IT-Betrieb

Hochschule **Rosenheim**
University of Applied Sciences

Inhaltliche Entwicklung: Prof. Dr. Gerd Beneken, Martin Kucich, Felix Hummel
„Innovationslabor" der Hochschule Rosenheim

DevOps

DevOps vereinigt Betrieb und Entwicklung auf der Ebene der Kultur, der Systeme, der Praktiken und der Werkzeuge. Ziel ist es, schneller und häufiger Nutzen (value) an den Kunden zu liefern [...].

DevOps ist agiles Liefern von IT, notwendig um die Rhythmen des IT-Betriebs und der agilen Entwicklung auf einander anzupassen.[...]
[Rob England]

http://www.itskeptic.org/content/define-devops

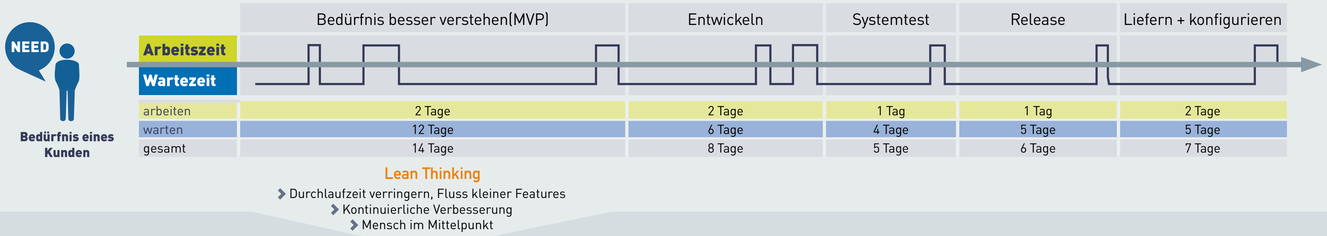
46x	**Häufiger liefern** (Deployment Frequency)
440x	**Schnellere Bearbeitungszeit** für Änderungen
96x	**Schnelleres Erholen** nach ausfällen (MTTR)
5x	**Geringere Fehlerrate** bei Änderungen

https://puppet.com/system/files/2017-06/2017-state-of-devops-report_3.pdf

Lean Thinking

Analyse des Value-Stream über Value-Stream Maps: Durchlaufzeit verkürzen: Wartezeiten abschaffen!

Durchlaufzeit verringern
- Automatisierung aller Schritte (soweit möglich)
- Halbfertige Features vermeiden (Work in Progress begrenzen)
- Pull Prinzip und Just in Time
- Engpässe identifizieren und abbauen
- Kleine Features (small batch sizes)
- Qualität einbauen (Nacharbeiten vermeiden)



Kontinuierliche Verbesserung
- Fehler systematisch abstellen (Root-Cause-Analysis)
- Probleme schnell sichtbar machen (→ Band anhalten)
- Transparenz, Visuelles Management
- Ständiges Üben macht den Meister

Lean Thinking
- Durchlaufzeit verringern, Fluss kleiner Features
- Kontinuierliche Verbesserung
- Mensch im Mittelpunkt

Cross-Functional

Gemeinsam für das laufende System verantwortlich
You Build it, You Run it! [Werner Vogels, Amazon]

Kontinuierlicher Fluss kleiner Arbeitspakte/Features (Low Risk Releases)

Feedback Verstärken und Beschleunigen!

- Fehlerrate im Test?, Code Qualität
- Testabdeckung, Testlaufzeiten
- Performance Kennzahlen Ressourcenverbrauch
- Erkenntnisse aus Penetration Test
- Ressourcenverbrauch?
- Probleme beim Liefern (z.B. wg. Datenmigration)?
- Gibt es kritische (vermeidbare) Ausfälle?
- Wie groß ist die Wiederanlaufzeit nach einem Ausfall? (MTTR)
- Wie elastisch skaliert das System?
- Von wem wird das Feature verwendet?
- Wie oft wird es benutzt?
- Wird es so verwendet? Wie gedacht?
- Was macht der Kunde mit dem System?

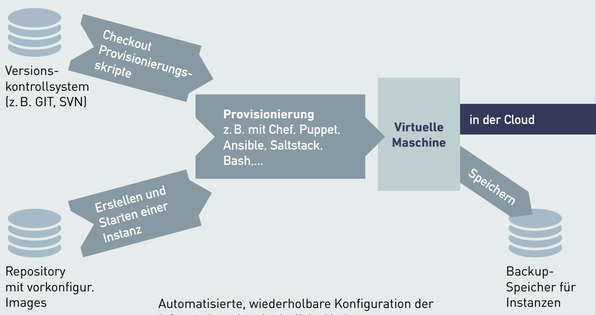
(Quelle: E. Ries: Lean Startup und G. Kim et al: DevOps Handbook)

Infrastructure as Code

- Wiederherstellbare Konfigurationen
- Infrastruktur auch als Einheit des Konfigurationsmanagements
- Einheitliche Maschinen in Dev, Stage und Prod

Infrastructure As Code ist ein Ansatz um Infrastrukturen automatisiert zu erstellen und zu betreiben basierend auf Praktiken der Software-Entwicklung. Er betont konsistente, wiederholbare Routinen zur Provisionierung und zum Ändern von Systemen und ihrer Konfiguration. Änderungen werden an den Routinen durchgeführt und dann auf die Systeme ausgerollt in einem vollständig automatisierten Prozess, der sich selbst validiert. [K. Morris: Infrastructure as Code , O Reilly 2016]

Konfigurationsmanagement über Provisionierung

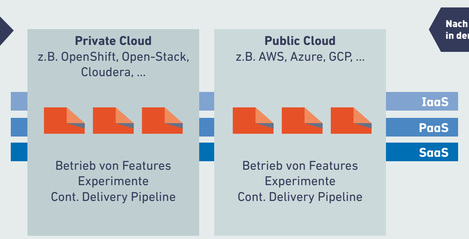


Automatisierte, wiederholbare Konfiguration der Infrastruktur, kontinuierliche Verbesserung.

Cloud Computing

- Selfservice Infrastruktur statt IT-Beschaffung
- Abrechnung nach Rechenleistung
- Elastische Skalierbarkeit, leichte Erweiterbarkeit
- Weltweite Verfügbarkeit, hohe Verfügbarkeit
- Transparenter Betrieb/Monitoring

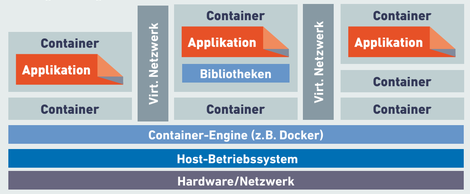
Cloud Computing ist ein Modell, das überall bequem und nach Bedarf Netzwerkzugriff auf einen gemeinsamen Vorrat an konfigurierbaren IT-Ressourcen ermöglicht. Das sind beispielsweise Netzwerke, Server, Speicher, Anwendungen und Dienste. Diese können schnell provisioniert und in Betrieb genommen werden, mit minimalem Aufwand bzw. Interaktion mit dem Provider. [NIST, http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf]



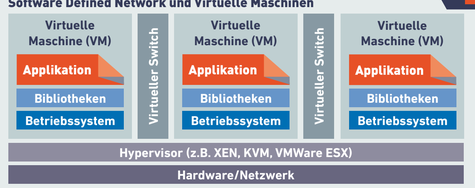
Virtualisierung, Container und Software Defined Networks

- Reproduzierbarkeit einer Installation als VM/Container
- Immutable Infrastructure (z.B. über Container-Stapel)
- Flexibler Betrieb vieler (verschiedener) Maschinen/Container
- Bessere Auslastung der Hardware

Leichtgewichtige Container



Virtualisierung im Betrieb
Software Defined Network und Virtuelle Maschinen



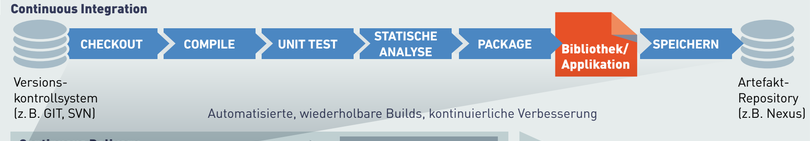
Betrieb der Pipeline und der Produktion

Continuous Delivery

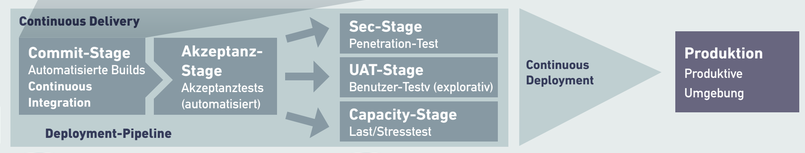
- Automatisierter und damit wiederholbarer/optimierbarer Build
- Automatisierte und damit wiederholbare/optimierbare Delivery Pipeline
- Kontinuierliche Qualitätssicherung, Kontinuierliche Verbesserung

Continuous Integration ist eine Praktik der Software-Entwicklung bei der die Teammitglieder ihre Arbeit (ihren Code) häufig integrieren. Üblicherweise integriert jede Person mindestens täglich. Das führt zu vielen Integrationen pro Tag. Jede Integration wird über einen automatisierten Build verifiziert, dieser enthält automatisierte Tests um Integrationsfehler so schnell wie möglich zu erkennen. [M. Fowler]

Continuous Integration



Continuous Delivery



Continuous Delivery ist eine Disziplin der Software-Entwicklung. Die Software wird kontinuierlich integriert, und automatisiert getestet, sodass sie zu jedem Zeitpunkt released und in Produktion genommen werden kann. Der Prozess ist weitgehend automatisiert. Ein geeignetes Tool hierfür ist bspw. XL Release von XebiaLabs

Continuous Deployment bedeutet, dass jede Änderung durch die Deployment Pipeline läuft und automatisiert produktiv gesetzt wird. Das führt zu vielen Lieferungen in die Produktion jeden Tag. Continuous Delivery bedeutet nur, dass man in der Lage ist, häufige Lieferungen durchzuführen, aber entscheiden kann es nicht zu tun. Üblicherweise weil die Fachabteilungen eine niedrigere Lieferrate bevorzugen. [M. Fowler]