

 tarent solutions GmbH

# Continuous Deployment mit Microservices



# Continuous Deployment mit Microservices

Tim Steffens

t.steffens@tarent.de

@tim\_127001

<https://github.com/tmstff>

(gibt's auf der letzten Folie nochmal zu sehen ;-)



Photo by [Cyrus Pellet](#) on [Unsplash](#)

An aerial photograph showing a narrow, winding body of water, likely a river or a coastal inlet, curving through a landscape of dark green vegetation and patches of brown, sandy soil. The water has a textured appearance with white foam or spray along its edges.

*Unterbrecht mich!*

Warum kürzer besser ist





Warum Continuous  
Deployment hilft  
glücklicher zu sein





Warum Microservices und Continuous Deployment so gut zusammenpassen

Warum kürzer besser ist





# Feedback Cycle

*"Feedback occurs when **outputs** of a system are **routed back as inputs** as part of a chain of cause-and-effect that forms a **circuit or loop**."*

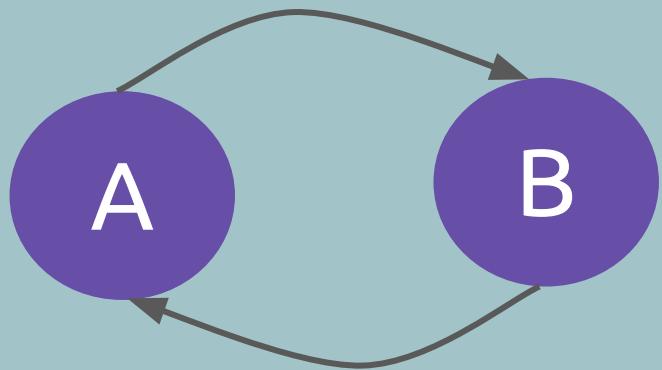
- Wikipedia



# Feedback Cycle

"Feedback occurs when **outputs** of a system are **routed back as inputs** as part of a chain of cause-and-effect that forms a **circuit or loop**."

- Wikipedia



# Feedback Cycle

*"Feedback occurs when **outputs** of a system are **routed back as inputs** as part of a chain of cause-and-effect that forms a **circuit or loop**."*

- Wikipedia

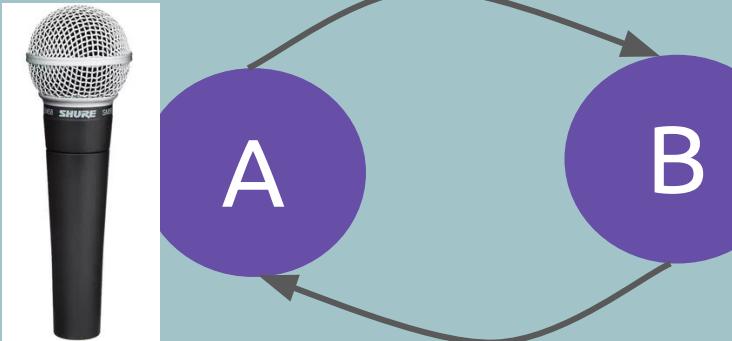


Photo by Cyrus Pellet on Unsplash

# Feedback Cycle

*"Feedback occurs when **outputs** of a system are **routed back as inputs** as part of a chain of cause-and-effect that forms a **circuit or loop**."*

- Wikipedia

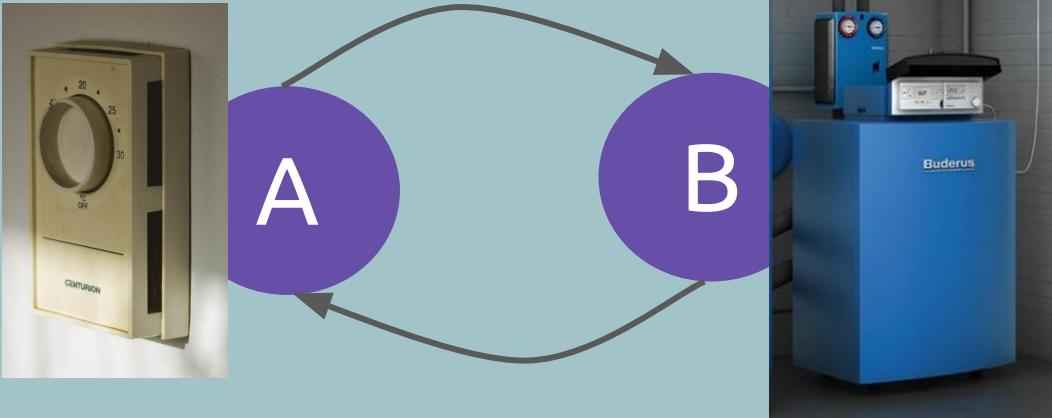


Photo by [Cyrus Pellet](#) on [Unsplash](#)



Continuous Deployment mit Microservices

# Feedback Cycle

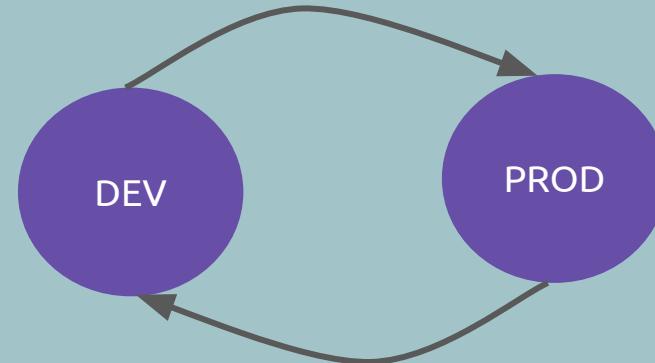
- Feedback ermöglicht **kontinuierliche Verbesserung**
  - mit Experimenten **Erkenntnisse** gewinnen
  - anpassen an **neue Situationen**



Continuous Deployment mit Microservices

# Feedback Cycle

- Feedback ermöglicht **kontinuierliche Verbesserung**
  - mit Experimenten **Erkenntnisse** gewinnen
  - anpassen an **neue Situationen**



# Feedback Cycle

Warum sind kurze Zyklen wichtig?



Photo by [Cyrus Pellet](#) on [Unsplash](#)



# Feedback Cycle

Warum sind kurze Zyklen wichtig?

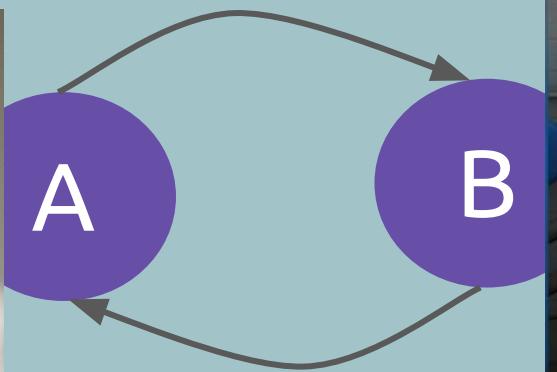


Photo by [Cyrus Pellet](#) on [Unsplash](#)



Continuous Deployment mit Microservices

# Beispiel: Chatfunktion in Onlineshop

Ein Onlineshop erhofft sich einen **höheren Umsatz**, wenn Kunden sich per **Chat** austauschen und Produkte empfehlen können



Continuous Deployment mit Microservices

# Beispiel: Chatfunktion in Onlineshop

**Kurzer** Feedback Cycle:

- **Minimaler Chat** wird implementiert
- wenn **kein User** den Chat aufruft (messen!), braucht man **keinen weiteren Aufwand** investieren



# Beispiel: Chatfunktion in Onlineshop

**Langer** Feedback Cycle:

- Release nur **vierteljährlich, manuelle QA**
- **Entwicklung** läuft noch **mehrere Wochen weiter**, bevor man mitbekommt dass das Feature nicht taugt



# Beispiel: Lang laufender Test

- Ausgangslage:
  - automatischer **UI-Test** läuft **4 Stunden**
  - der Build **läuft gerade**
- oft gesehen:
  - Horst checkt ein
  - Heinz checkt 30 min später ein
  - Hans checkt 60 min später ein
  - der **nächste Build schlägt** fehl
- **Welche Änderung** hat den Build kaputt gemacht?



Photo by [Cyrus Pellet](#) on [Unsplash](#)



## Beispiel: schnelle Bugfixes (Traumvorstellung)

- Ein **Kunde** stellt einen **Bug** fest und meldet sich beim Support
- Der Support meldet den **Bug** an das **Entwicklungsteam**
- Das Entwicklungsteam **findet** den **Fehler schnell**, behebt ihn und deployt den **Fix nach Produktion**
- Der Support meldet dem Kunden nach **30 Minuten**, dass das **Problem behoben** ist
- Der Kunde ist **begeistert** und schreibt einen **positiven Post** auf Facebook / Twitter / xyz:
  - *“Problem bei XY gemeldet - nach 30 min behoben :-O Warum bekommen die anderen das nicht hin?”*



# Beispiel: schnelle Bugfixes (Traumvorstellung)

- Warum waren für das Beispiel kurze **Feedback Zyklen wichtig?**
  - Der **Bug-Report** vom Kunden kam **schnell** beim **Entwicklungsteam** an (Support war direkt erreichbar, hat direkt reagiert und hatte einen direkten Draht zum Entwicklungsteam)
  - Das Team hatte kurz zuvor ein **einzelnes Feature** ausgerollt und konnte daher das Problem **schnell eingrenzen**
  - Die **automatisierten Tests** konnten dem Entwicklungsteam schnell bestätigen, dass der **Fix funktioniert** - und dass **keine neuen Probleme** entstanden sind



Photo by [Cyrus Pellet on Unsplash](#)



# Feedback Cycle

## Warum ist kürzer besser?

- Menschen sind sehr **schlecht** darin, die **Zukunft vorher zu sagen!**
- **Annahmen** aufstellen, **kleine Schritte** machen, stets **verifizieren!**
  - **Minimiert** das **Risiko!**
  - **Frühzeitiges** & präzises **Feedback**
  - schnelle **Reaktionsfähigkeit, Umlenken** falls nötig
- Voraussetzung: **Annahmen & Maße / Indikatoren** zum Überprüfen!



Photo by [Foto Garage AG](#) on [Unsplash](#)



Warum Continuous  
Deployment hilft  
glücklicher zu sein





# Continuous Integration

- Nach **Checkin** ins VCS wird der Code **automatisch**
  - **kompiliert**
  - **Unit-** (& ggf. Component-) **getestet**
  - **“gemessen”** (d.h. es werden Metriken ermittelt)
  - in installierbaren **Paketen verpackt**
  - auf einer **(Test-)Umgebung installiert**
  - **Integrations-getestet**
- Diese Schritte werden häufig **Build Pipeline** genannt und von einem Build Server ausgeführt
- Tests & Messungen dienen als **Quality Gates**





# Quality Gates

- “**Quality Gates** sind **Punkte** im Ablauf eines Entwicklungsprojekts, bei denen anhand von im Voraus eindeutig bestimmten **Qualitätskriterien** über die **Freigabe** des nächsten **Projektschrittes** entschieden wird.”

- Wikipedia



# Quality Gates

- **Automatisiert**
  - **Compiler**
  - **Linter / Metriken** (ESLint, FindBugs, PMD ...)
  - **Tests (auch UI!)** (Junit, Spock, Selenium / Geb, Galen ...)
  - **Security** (Clair, snyk, Find Security Bugs, ...)
- **Manuell**
  - **Code Review**
  - **explorativer Test**



# Continuous Delivery / Deployment

- **Continuous Delivery** = CI + auf Knopfdruck **installierbares Paket**
- **Continuous Deployment** = Continuous Delivery + **automatische installation auf Prod**
  - Jeder **commit** auf **master** landet auf **Produktion**
    - (wenn die Quality Gates ihn durchließen)





# Angst Produktivdeployment

- Viele Entwickler haben sehr **großen Respekt** vor einem **Deployment in Produktion**
  - Ein **Fehler** würde sich direkt auf den **Kunden auswirken**
- **Bedenken**
  - Ist die Software wirklich **ausreichend getestet?**  
**Funktioniert alles**, wie es soll?
  - Kann ich beim Deployment etwas **falsch machen?**
  - Bekomme ich **Ärger**, wenn ich einen Fehler deploye?
  - Ist es nicht **doof**, wenn **unfertige Features Live** gehen?





# Angst Produktivdeployment

- **Resultat**
  - **Vermeidung**, "erstmal abwarten"
  - Software wird **später released**
  - Software wird **seltener released**, dafür mit mehr Features pro Release
- D.h.
  - **Feedback Cycle** wird **länger!**
  - **Welche Änderung** führte zu einem Problem?
    - => **schwierig!**





# Angst Produktivdeployment

- **Lösungen**

- **Vertrauen schaffen**
  - z.B. durch gute Testautomatisierung & Automatisierung des Deployment
- Nur **fertige Features** sichtbar machen
  - z.B. durch Branching / Feature Toggles
- **schnell mitbekommen** wenn etwas **schief geht**
  - z.b. durch **Monitoring & Alerting**





# Feature Toggles

- AKA **Feature Flags**
- Erlauben es zur **Laufzeit** festzulegen ob ein **Feature aktiv** sein soll
  - bzw. **welche Implementierung** eines Features verwendet wird
  - ggf. in Abhängigkeit von **User, Standort, IP-Adresse ...**





# Feature Toggles

- Anwendungsfälle:
  - **unfertige Features verstecken**
  - **Tests auf dem Produktivsystem** für ausgewählte Benutzergruppen
  - **Schnelles "Rollback"**
  - **Deployment Dependencies entkoppeln**
  - **A / B Testing**
  - **Canary Releases**





# Warum Continuous Deployment hilft glücklicher zu sein

- der **Feedback Zyklus** wird minimiert 😊
- **Delta Test & Prod System** reduziert (!!!) 😊
- **händische Fehler** weitgehend **ausgeschlossen** 😊
- **Vertrauen aufgebaut**, dass neue & alte **Features funktionieren** 😊
- schnellere **Time to Market** und **schnelle Rekationszeit** (da alles automatisiert) 😊
- **Finden von Fehlern** wird **beschleunigt** (da weniger Änderungen als Problemquelle in Frage kommen) 😊





Warum Microservices und Continuous Deployment so gut zusammenpassen



Continuous Deployment mit Microservices

# Was ist ein Microservice?

- eine "kleine" **Applikation**
- Klar **abgegrenzte Fachlichkeit**
- **Self-Contained**
  - enthält alle Abhängigkeiten (Libs, ggf. http-Server)
  - Kann als Prozess gestartet werden



# Was ist ein Microservice?

- **Separat Deploybar** (& Skalierbar)
- **Eigene Codebase**
- **Eigene Datenhaltung** (Decentralized Data Management)
- Kommunikation über **definierte Schnittstellen** (z.B. REST)





Continuous Deployment mit Microservices

# Warum nutzt man Microservices?

- **Unabhängiges Entwickeln & Deployen** von Services
- **Organisationsstruktur** richtet sich nach **Fachlichkeit**
  - Silo Teams vs **Cross-Funktionale Teams**
- **Decentralized Governance**
- **Service wegwerfen & neu schreiben** bei Fehlentscheidungen oder neuen Erkenntnissen
- **gezieltere Skalierbarkeit &** (im Optimalfall) **höhere Resilienz**



# Microservices + CD = ❤

- **Mircoservices** brauchen i.d.R. **nicht lange um gebaut und getestet** zu werden 😊
  - **starten schnell** 😊
  - und werden **schnell deloyt** 😊
    - **Cloud deployment** häufig anzutreffen 😊
- => **super für kurze Feedbackzyklen** 😊





# Microservices + CD = ❤️

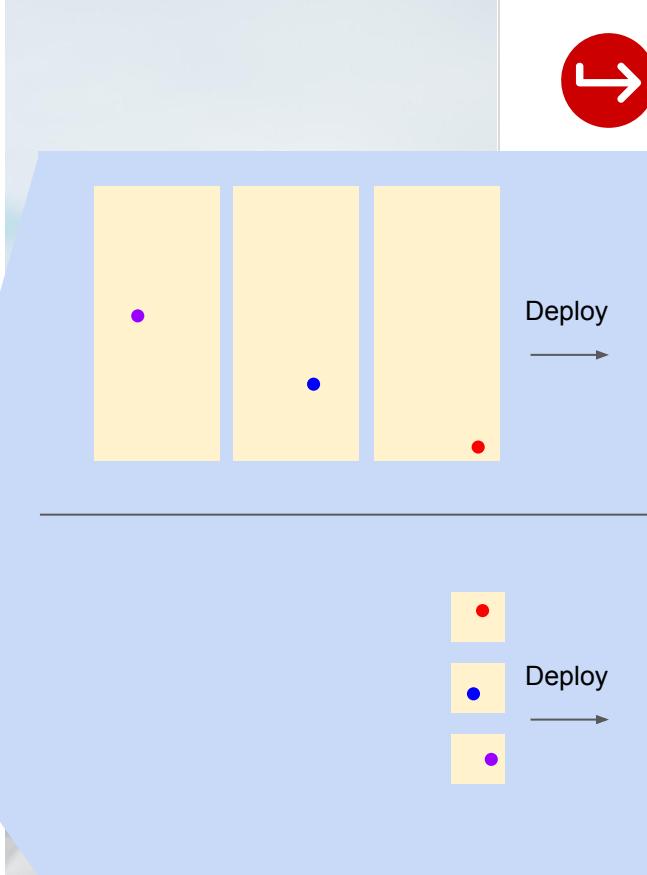
- Deployment betrifft nur **eine fachliche Komponente => minimiert Risiko 😊**
- **Unabhängige Build- & Deployjobs**, in einem großen Projekt mit vielen Deployments **Queuen sich weniger Build-Jobs 😊**
- **You Build it you run it 😊**





# Microservices + CD = ❤

- Deployment betrifft nur **eine fachliche Komponente => minimiert Risiko 😊**
- **Unabhängige Build- & Deployjobs**, in einem großen Projekt mit vielen Deployments **Queuen sich weniger Build-Jobs 😊**
- **You Build it you run it 😊**





# Microservices + CD = ❤️

- Deployment betrifft nur **eine fachliche Komponente => minimiert Risiko 😊**
- **Unabhängige Build- & Deployjobs**, in einem großen Projekt mit vielen Deployments **Queuen sich weniger Build-Jobs 😊**
- **You Build it you run it 😊**



Photo by [Pablo Heimplatz](#) on [Unsplash](#)



tarent solutions GmbH

Vielen Dank ❤️

### Kontakt

Tim Steffens

<https://github.com/tmstff>  
@tim\_127001  
t.steffens@tarent.de

[www.tarent.de](http://www.tarent.de)

