The Twelve-Factor App

Regeln für das Betreiben von Software in der „Cloud“

DHBW

Was bedeutet „12-factor-app“?

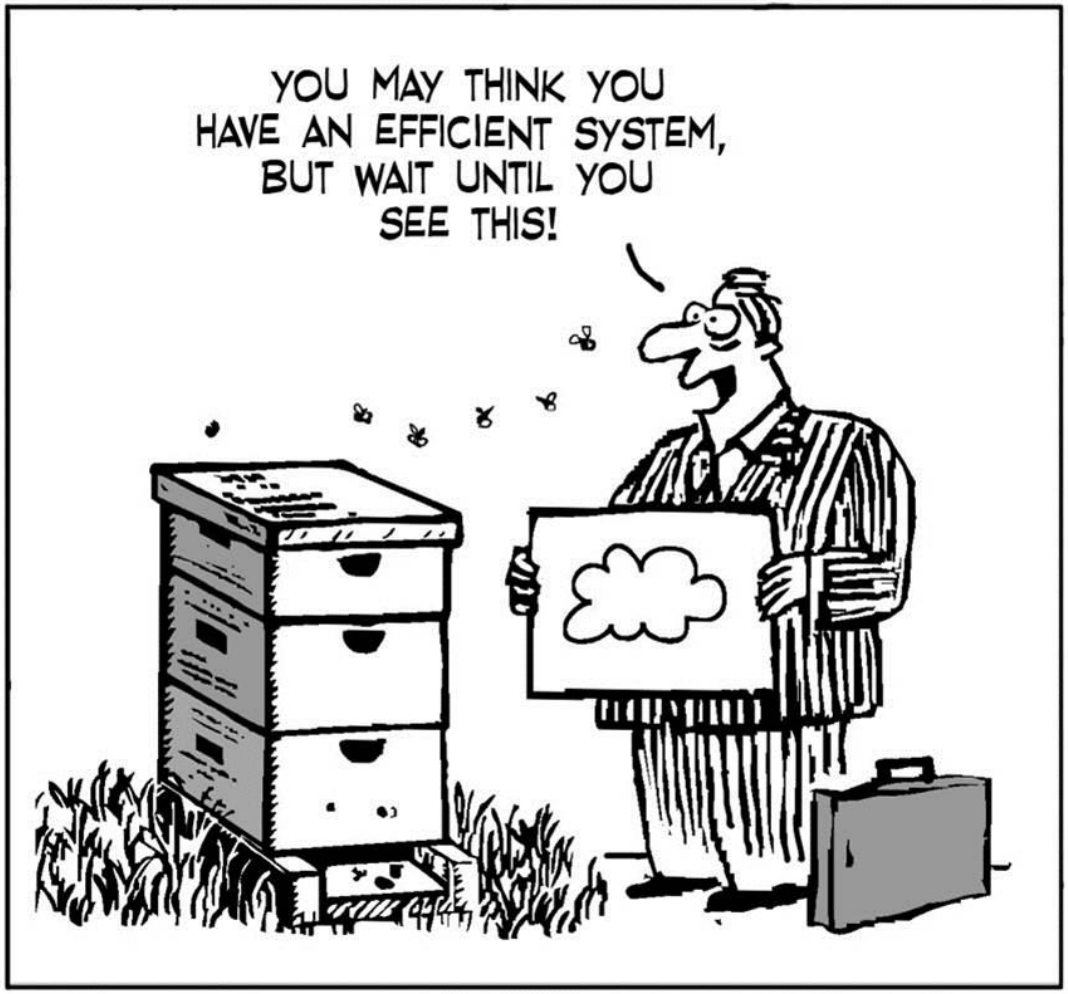
- 12 allgemeine, sprachunabhängige Empfehlungen für das Erstellen von Anwendungen, die „in der Cloud“ deployed werden
- Ursprünglich als Richtlinien für Heroku-kompatible bzw. PaaS-Anwendungen gedacht, jedoch größtenteils generell anwendbar

Despite being partly self-serving (apps built like this will translate more naturally to running on Heroku), there’s a lot of meaty best-practices worth examining.

- WILL KOFFEL, *http://www.clearlytech.com/2014/01/04/12-factor-apps-plain-english/, 28.02.2016*

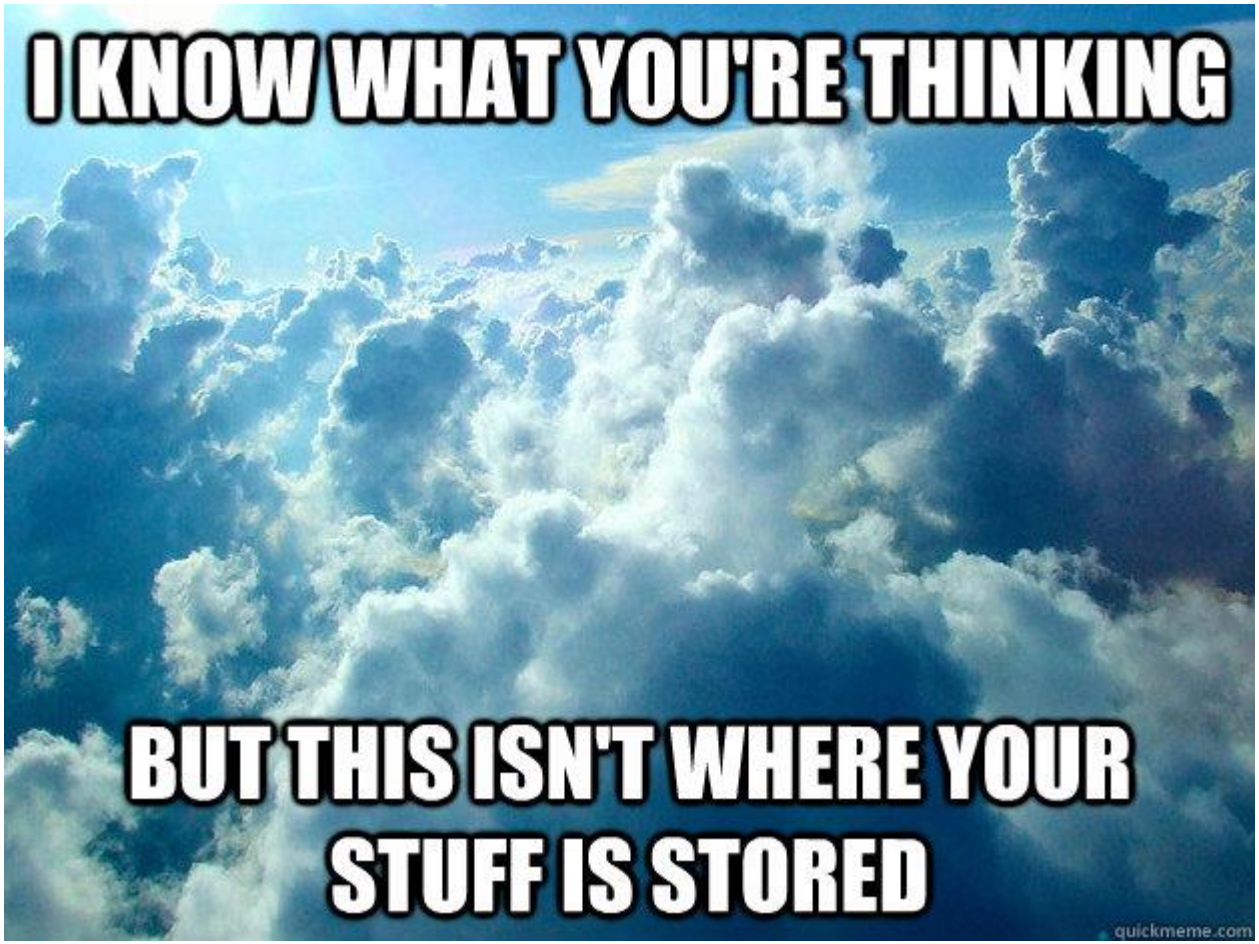
DHBW

Die „Cloud“



DHBW

Die „Cloud“


I KNOW WHAT YOU'RE THINKING
BUT THIS ISN'T WHERE YOUR STUFF IS STORED
quickmeme.com

Was bedeutet „lauffähig in der Cloud“

„With cloud services, failure is, in fact, inevitable”

-https://blogs.microsoft.com/cybertrust/2014/08/12/designing-for-failure-the-changing-face-of-reliability/

DHBW

Was bedeutet „lauffähig in der Cloud“

- Agnostisch gegenüber Infrastruktur
- Portierbar
- (kontinuierlich und schnell) deploybar
- Skalierbar
- Robust gegenüber Ausfällen
- Anwendungen, die sich nicht darauf verlassen, dass die Infrastruktur immer verfügbar ist und funktioniert

DHBW

Portabilität

- Anwendung kann schnell auf einen anderen Knoten ungezogen werden
- Plattformunabhängigkeit

DHBW

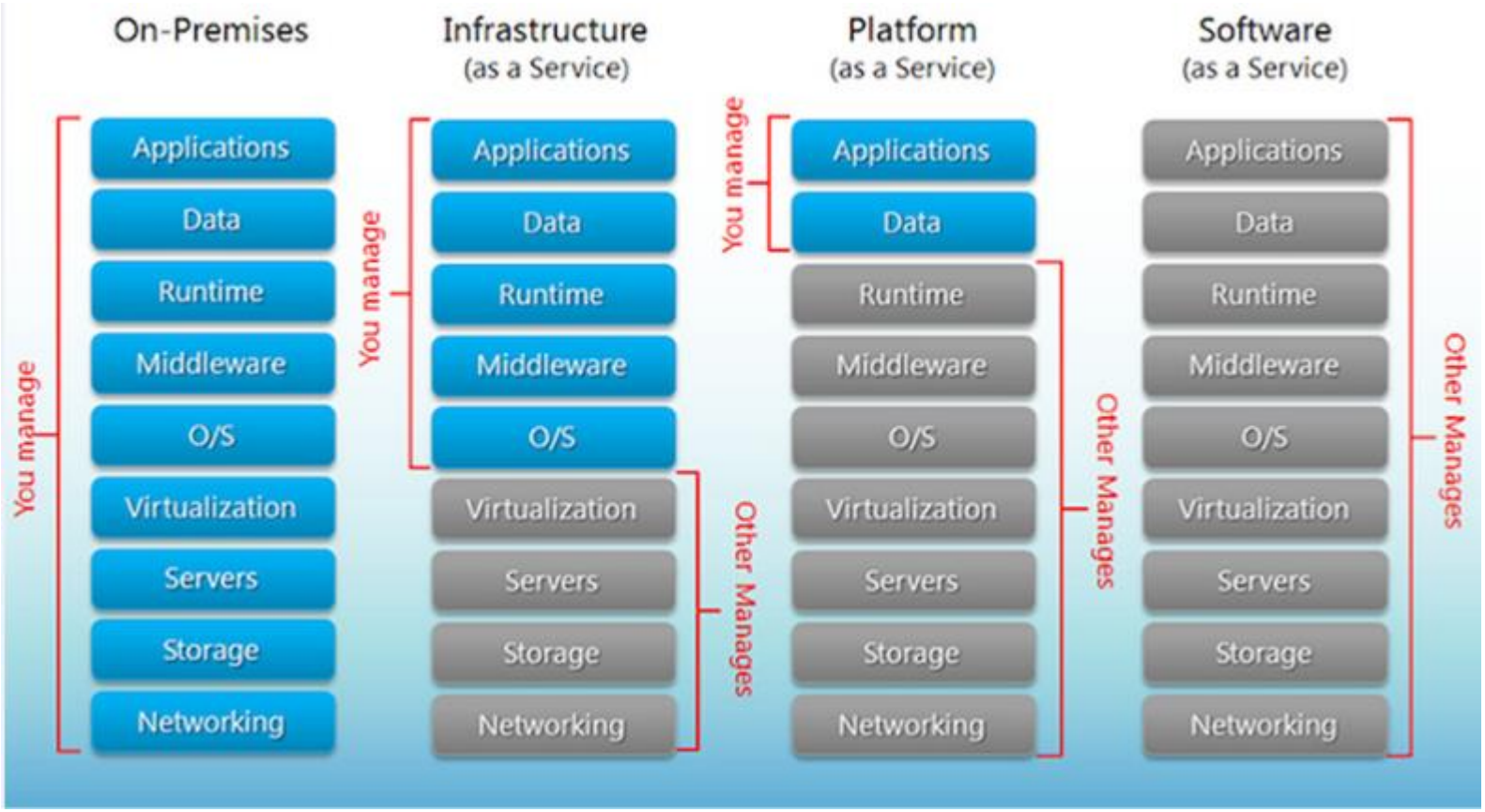
Skalierbarkeit

- Fähigkeit eines Systems, auf höhere Last durch das Hinzufügen von Ressourcen zu reagieren
- **Vertikale Skalierung (scale up)**
  - Hinzufügen von Ressourcen zu einem Knoten des Systems (CPU, Arbeitsspeicher, Festplatte, …)
  - Keine Code-Änderung nötig
  - limitiert
- **Horizontale Skalierung (scale out)**
  - Hinzufügen weiterer Knoten zum System
  - Architektur- bzw. implementierungsabhängig (nicht mit jeder Anwendung möglich)
  - Theoretisch unlimitiert

Vgl. https://de.wikipedia.org/wiki/Skalierbarkeit, abgerufen am 29.12.2015

DHBW

As-a-Service-Modelle



Ziele dieser Vorlesung

- Workshop-Charakter
- Sie denken darüber nach, warum Sie bestimmte Dinge so tun, wie Sie sie tun
- Sie nehmen Ideen mit, wie Sie bestimmte Dinge anders tun könnten
- Sie treffen in Zukunft informierte(re) Entscheidungen bei der Entwicklung von Anwendungen

DHBW

Die 12 Faktoren

1. Eine code-basis pro app
2. Abhängigkeiten
3. Konfigurationsmanagement
4. Unterstützende dienste
5. Build, Deploy, run trennen
6. Unabhängige Prozesse
7. Port-bindung
8. Nebenläufigkeit
9. Verfügbarkeit
10. Gleichheit der umgebungen
11. Logs
12. Admin-prozesse

DHBW

1. Eine Codebasis pro App

„One codebase tracked in revision control, many deploys“

DHBW

1. Eine Codebasis pro App

„One codebase tracked in revision control, many deploys“

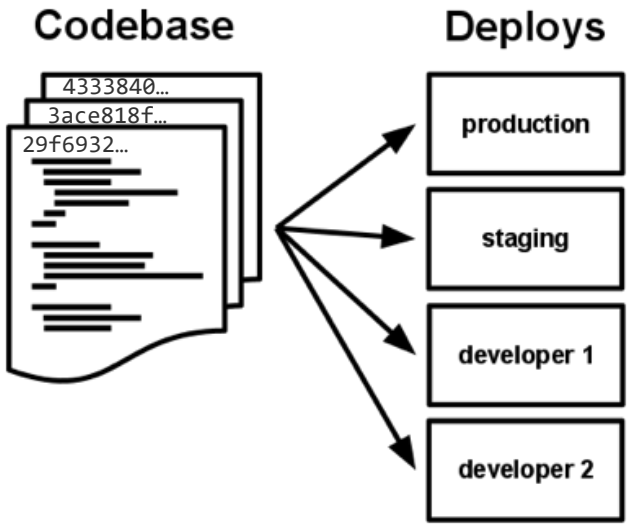
- Eine 12FA wird in einer Versionsverwaltung versioniert (Codebase = Repository)
- Jede App hat genau eine Codebase
  - Mehrere Codebases/Repositories = mehrere Apps
  - Nie mehr als eine App pro Codebase

DHBW

1. Eine Codebasis pro App

„One codebase tracked in revision control, many deploys“

- Alle Deployments stammen aus der selben Codebase
- Deployment: Installation und Konfiguration einer Anwendung in einer bestimmten Zielumgebung



DHBW

1. Eine Codebasis pro App

„One codebase tracked in revision control, many deploys“

Begründung:

„Everyone does this, and developers will laugh at you if you aren’t.”

- WILL KOFFEL, *http://www.clearlytech.com/2014/01/04/12-factor-apps-plain-english/, 28.02.2016*

DHBW

1. Eine Codebasis pro App

„One codebase tracked in revision control, many deploys“

Kritik:

- Es gibt Anwendungen, die dies nicht betrifft
  - Weil Artefakte starken Bezug untereinander haben
  - UND kein Artefakt in mehrere unterschiedlichen Lösungen verwendet wird
- „ein Repository pro **Lösung**“
- Empfehlung: informierte Entscheidung treffen

DHBW

2. Abhängigkeiten

„ Explicitly declare and isolate dependencies“

2. Abhängigkeiten - Theorie

„ Explicitly declare and isolate dependencies“

- Eine 12FA verlässt sich nie darauf, dass benötigte Abhängigkeiten in einer Zielumgebung implizit vorhanden sind, sondern **deklariert alle Abhängigkeiten explizit**
- eine 12FA teilt ihre Abhängigkeiten nie mit anderen Anwendungen auf dem gleichen System, sondern **isoliert diese vom Rest des Systems**

DHBW

2. Abhängigkeiten

„ Explicitly declare and isolate dependencies“

Deklaration von Abhängigkeiten

Mögliche Abhängigkeiten:

- Pakete/Bibliotheken/Module
- Laufzeitumgebung/Interpreter/Application Container
- Berechtigungen
- Verzeichnisstruktur
- Systemwerkzeuge
- Ressourcen (Network, Grafikkarte, …)
- Dienste (Webserver, …)

->alles, was eine Anwendung zusätzlich zum Code benötigt, um zu funktionieren

DHBW

2. Abhängigkeiten

„ Explicitly declare and isolate dependencies“

- Idealerweise sind alle Abhängigkeiten in der Anwendung enthalten – dies ist aber nicht immer möglich bzw. nicht mit vertretbarem Aufwand realisierbar
- Mit **Dependency-Management-Tools** (Maven, Bundler, Composer, …) und **Infrastructure-as-Code-Tools** (Puppet, …) können die meisten Abhängigkeiten deklariert werden
- Wo dies NICHT möglich ist, MUSS die App beim Start sicherstellen, dass alle nicht mit auslieferbaren Abhängigkeiten in der korrekten Version vorhanden sind

DHBW

2. Abhängigkeiten

„ Explicitly declare and isolate dependencies“

Isolation von Abhängigkeiten

- Abhängigkeiten gelten immer nur für eine Anwendung
- Abhängigkeiten können nicht von außen geändert werden
- Abhängigkeiten werden nicht geteilt

DHBW

Abhängigkeiten
Empfehlungen

- Höchste Form der Kontrolle, wenn Anwendung als Container ausgeliefert wird oder auf einem dedizierten Server läuft
- Volle Isolation oft nicht gegeben/möglich
- Man muss nicht alles ausliefern, aber alles deklarieren oder prüfen – wenigstens beim Start der Anwendung

  = Abhängigkeiten „explizit“ machen

3. Konfigurationsmanagement

„Store config in the environment”

DHBW

3. Konfigurationsmanagement - Theorie

„Store config in the environment”

- In einer 12FA sind **Code und Konfiguration strikt** voneinander **getrennt**
  - Eine 12FA speichert niemals Konfiguration in der Codebase
  - Eine 12FA verwendet niemals Konstanten für Konfigurations-Einstellungen
  - Das selbe Binary(Artefakt) wird in allen Umgebungen verwendet
- Stattdessen wird eine 12FA über **Umgebungsvariablen** konfiguriert

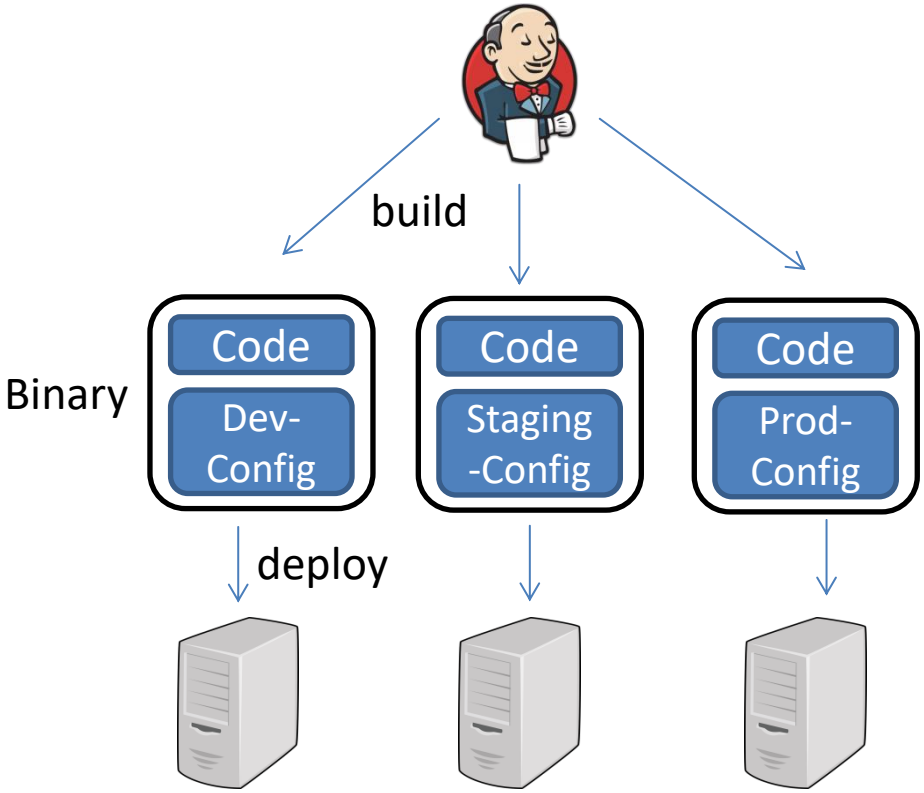
DHBW

Was ist Konfiguration?

- Alles, was sich zwischen **verschiedenen Umgebungen ändert**
  - Credentials
  - Verzeichnisse (logs, tmp, …)
  - Angaben zu Ressourcen wie DB (siehe „Backing Services“)
- Probe:
  - **muss ich committen, um eine Konfigurationseinstellung zu verändern?**
  - **Kann ich das Repository veröffentlichen, ohne die Sicherheit zu gefährden?**
- Bezieht sich **nicht auf interne Konfiguration** wie URL-Mappings/Routen, Dependency Injection usw.

Verschiedene Binaries, viele Umgebungen

- Konfiguration liegt in Versionsverwaltung
- Binaries können sich unterscheiden

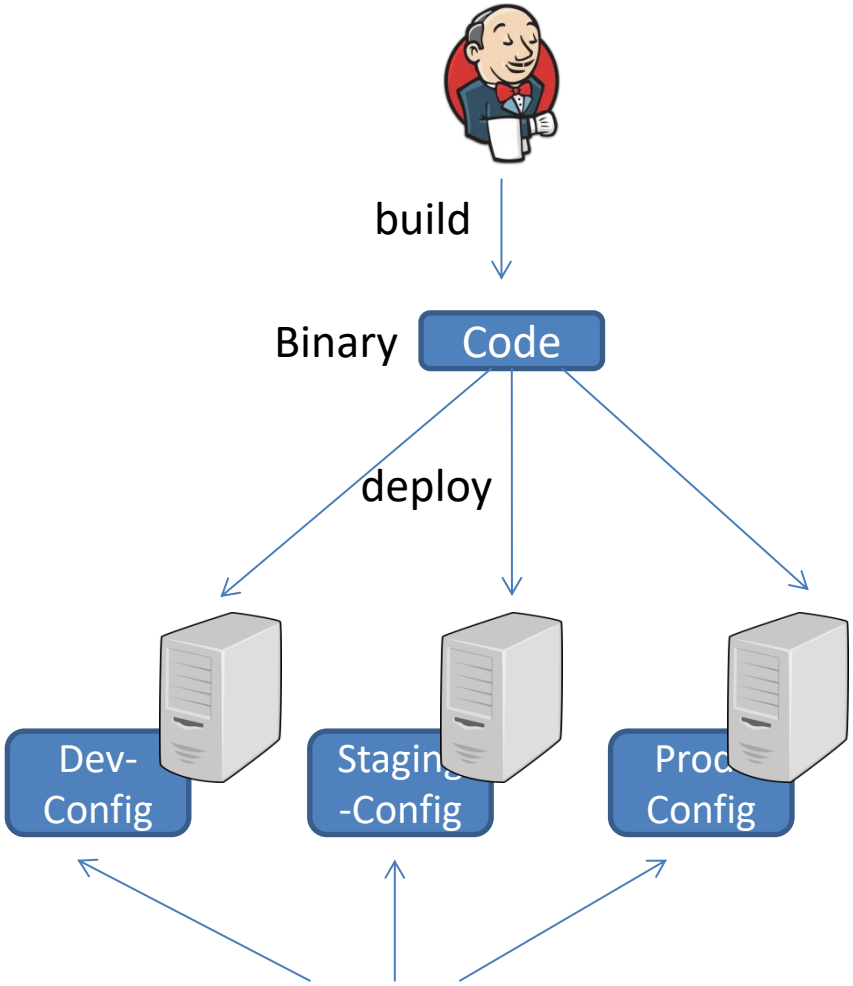


Anti-Pattern!!!

DHBW

Eine Binary, viele Umgebungen

- Ideal:



Problem: wie kommt Konfig auf den Server?

DHBW

Wie können Binary und Konfiguration getrennt werden?

- Die umgebungsabhängigen Konfigurationseinstellungen werden nur auf dem Zielsystem abgelegt:
  - Als Umgebungsvariablen
  - Als Konfig-Datei
  - …
- Die Anwendung lädt die Einstellungen beim Start und validiert diese

DHBW

Pro und Kontra: Konfiguration über Umgebungsvariablen

- Einfach zu verändern
- sprach- und betriebssystemunabhängig
- Nur einfache key/value-Paare erlaubt
- „sickern“ leicht nach draußen
- Nicht immer setzbar (fehlende Rechte)
- Erlaubte/notwendige Konfiguration nicht offensichtlich

DHBW

Konfigurationsmanagement Empfehlungen

- Sollte außerhalb des Projektes liegen und es schwierig machen, unabsichtlich Konfig in die Versionsverwaltung zu committen
- Sollte einfach eingelesen werden können (sprach/tool-unabhängig)
- Eingeschränkte Ausdrucksfähigkeit (keine komplexen, verschachtelten Konfigurationen)

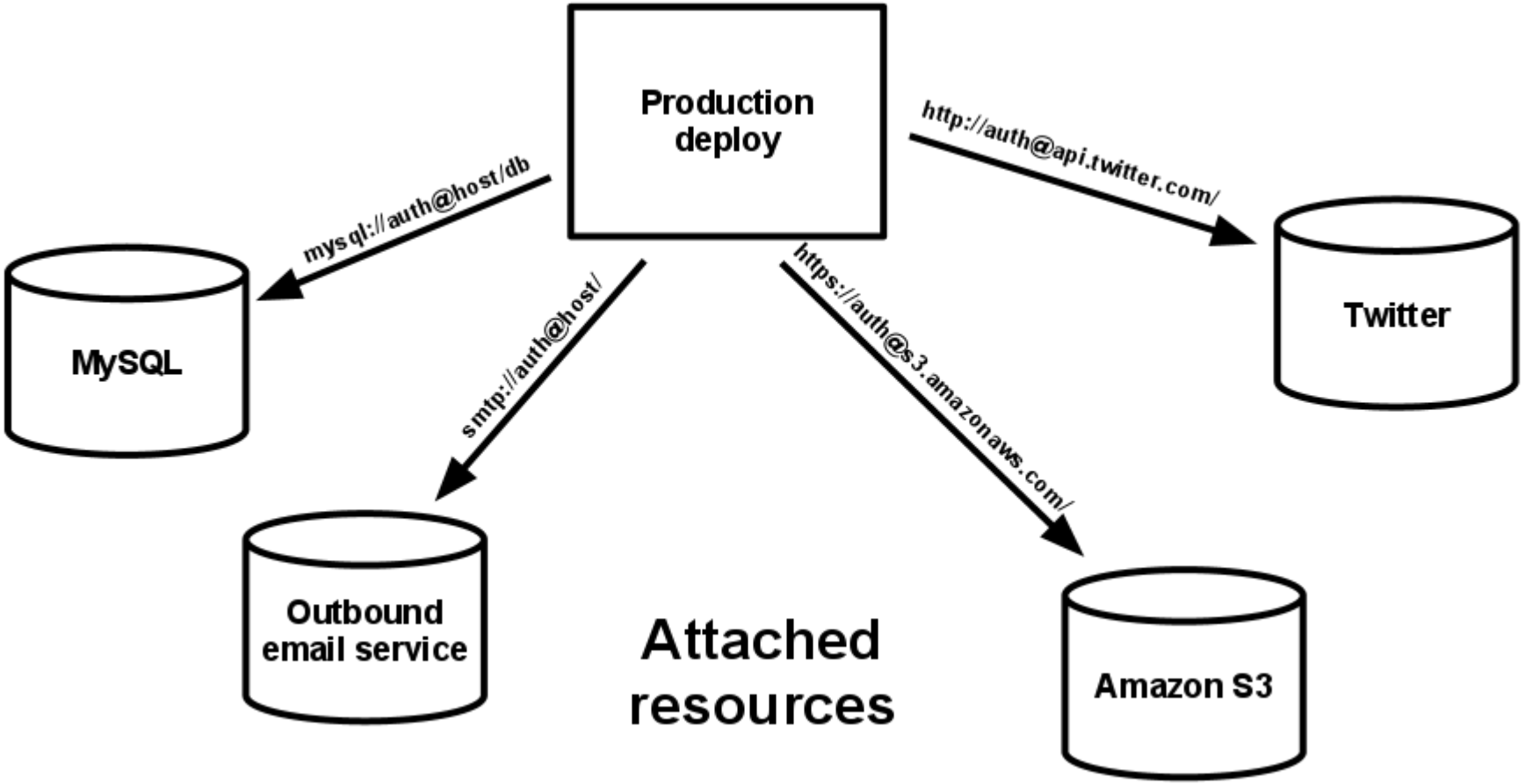
Konfigurationsmanagement Empfehlungen

- Anwendung sollte ihre Konfiguration loggen
- Anwendung muss beim Start prüfen, ob alle nötigen Einstellungen vorhanden sind
- Es muss nachgelesen werden können, welche Einstellungen möglich/nötig sind
- Niemals Credentials in VCS einchecken

DHBW

4. Backing Services

„Treat backing services as attached resources“



Quelle: http://12factor.net/backing-services

DHBW

4. Backing Services

„Treat backing services as attached resources“

- In einer 12FA können **unterstützenden Dienste** („backing service“) einfach **angehängt (attached) oder abgehängt (detached) werden**
- Eine 12FA **unterscheidet nicht** zwischen **lokalen** oder **entfernten** Diensten
- Der Zugriff auf unterstützende Dienste erfolgt über eine **URL**
- unterstützende Dienste können durch **Änderungen an der Konfiguration ausgetauscht** werden

DHBW

4. Backing Services

„Treat backing services as attached resources“

- Beispiele für unterstützende Dienste:
  - DB
  - Messaging
  - SMTP
  - Cache (bspw. Memcached)
  - Twitter
  - Google Maps
  - ElasticSearch
  - NewRelic, …

DHBW

4. Backing Services

„Treat backing services as attached resources“

Begründung:

- Anwendung wird flexibler
  - Beispiel: fehlerhafte DB-Instanz abhängen und neue Instanz aus aktuellem Snapshot anhängen
- leichter skalier- und portierbar
- Anwendung wird robuster gegen Ausfall eines Dienstes

DHBW

4. Backing Services

„Treat backing services as attached resources“

Praxistipp:

- Anwendung sollte beim Start prüfen, ob alle benötigten Dienste ansprechbar sind

DHBW

5. Build, Release, Deploy, Run

„Strictly separate build and run stages”

DHBW

5. Build, Release, Deploy, Run

„Strictly separate build and run stages”

- Jede Änderung an einer 12FA muss durch die Phasen **build, release, deploy und run**
- Jedes Release soll eine **eindeutige ID** erhalten
- Ein Release kann **nach Erstellung nicht geändert** werden - jede Änderung muss zu einem neuen Release führen

DHBW

5. Build, Release, Deploy, Run

„Strictly separate build and run stages”

Build:

- Erzeugen aller Artefakte (deliverables)

Release:

- Artefakt mit eindeutiger, nachverfolgbarer Version versehen

Deploy:

- Artefakt und Konfiguration kombinieren
  - Zielumgebung aufsetzen und konfigurieren
  - Artefakte in Zielumgebung kopieren

Run:

- Artefakt als Prozess starten

DHBW

5. Build, Release, Deploy, Run

„Strictly separate build and run stages”

Begründung:

- Code-Änderungen am Produktivsystem unterbinden
- Einfacher Rollback (zum zuletzt funktionierenden Release) möglich
- Komplexere Aufgaben finden in build/release/deploy statt -> Start der Anwendung (run) so simpel wie möglich halten
- Meint im Grunde: continuous integration/continuous delivery

6. Unabhängige Prozesse

„Execute the app as one or more stateless processes“

- Grafik

6. Unabhängige Prozesse

„Execute the app as one or more stateless processes“

- Eine 12FA wird als ein oder mehrere Prozesse ausgeführt
- 12FA-Prozesse sind zustandslos (stateless) und „shared-nothing“

DHBW

6. Unabhängige Prozesse

„Execute the app as one or more stateless processes“

Zustandslosigkeit

- Keine Session
- Jede Anfrage (Request) ist unabhängig von allen anderen
- Wenn etwas gespeichert wird, dann in einem persistenten Speicher

Let’s say you have a signup workflow, where a user has to enter 3 screens of information to create their profile. One (wrong) model would be to store each intermediate state in the running code, and direct the user back to the same server until the signup process is complete. The right approach is to store intermediate data in a database or persistent key-value store, so even if the web server goes down in the middle of the user’s signup, another web server can handle the traffic, and the system is none-the-wiser.

DHBW

6. Unabhängige Prozesse

„Execute the app as one or more stateless processes“

Begründung:

- Hängt mit der Art und Weise zusammen, wie PaaS-Anbieter wie Heroku Anwendungen skalieren (nämlich auf Prozess-Ebene)
- Zustandslosigkeit und Shared-Nothing-Architektur ist Voraussetzung für horizontale Skalierbarkeit
- Zustandslose Apps begünstigt allgemein Robustheit einer Anwendung

DHBW

7. Port Binding

„Export services via port binding“

GRAFIK

7. Port Binding

„Export services via port binding“

Das sagen die Regeln:

- Eine 12FA ist eigenständig („self-contained“) und nicht auf Webserver/Application Server/etc. angewiesen
- Eine 12FA bindet sich selbst beim Start an einen (konfigurierbaren) Port und nimmt Anfragen über ein bestimmtes Protokoll an (zum Beispiel Port 80 und http)

DHBW

7. Port Binding

„Export services via port binding“

Begründung:

- Unabhängigkeit von externen Diensten (apache, nginx, tomcat, …) macht Anwendungen leichter skalier- und portierbar
- Anwendung wird einfacher und schlanker
- Trägt zu Regel 2 bei
- Kommunikation über Port macht Anwendung zum **Backing-Service** für andere Anwendungen
- Anwendung existiert für die „Außenwelt“ nur in Form von Port, Protokoll und API und kann daher einfach(er) durch Neuimplementierung ausgetauscht werden, so lange Port/Protokoll/API gleich bleiben

DHBW

8. Nebenläufigkeit

„Scale out via the process model”

- Grafik

8. Nebenläufigkeit

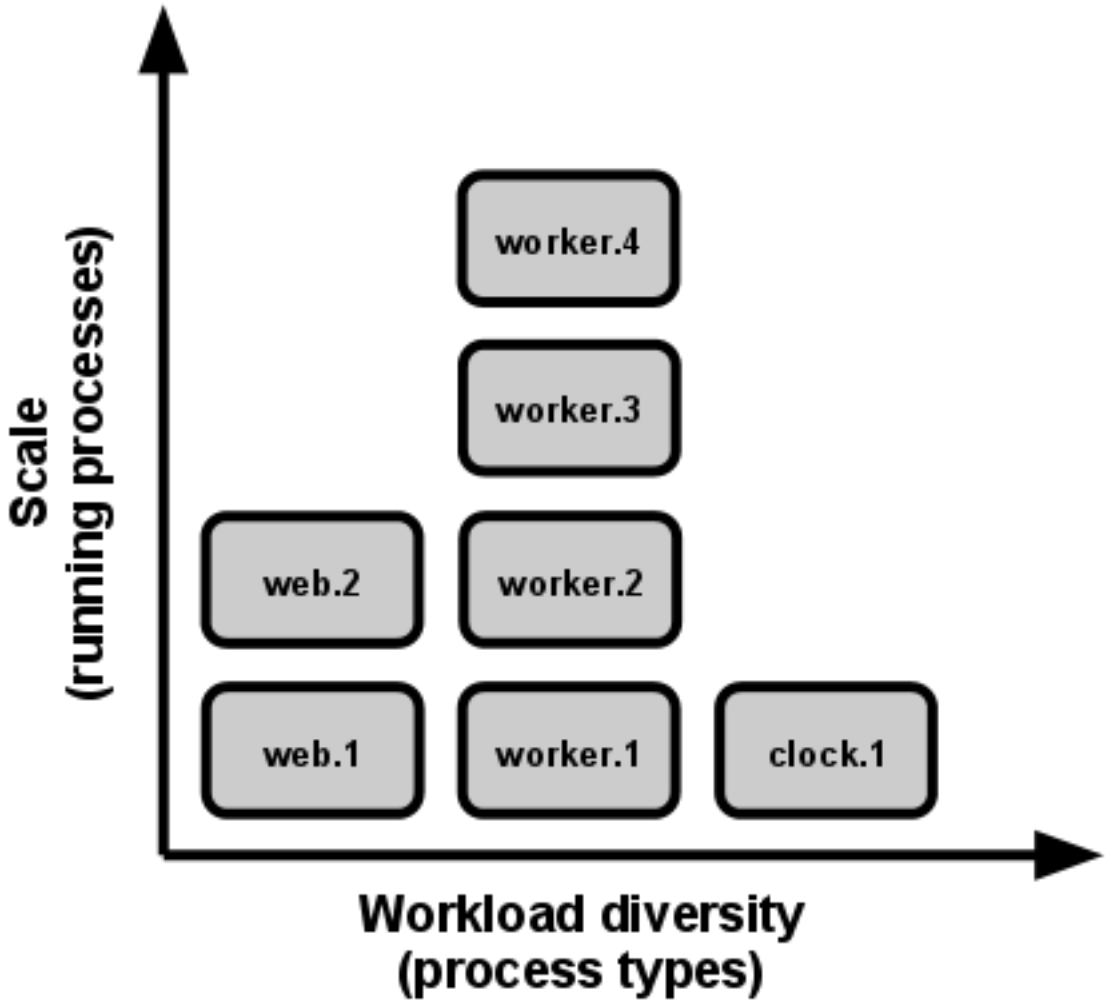
„Scale out via the process model”

- Eine 12FA skaliert horizontal über zusätzliche Prozesse
- Bei stärkerer Last werden mehr Prozesse gestartet

DHBW

8. Nebenläufigkeit

„Scale out via the process model”



8. Nebenläufigkeit

„Scale out via the process model”

Begründung:

- Hängt mit der Art und Weise zusammen, wie PaaS-Anbieter wie Heroku Anwendungen skalieren
- Im Wesentlichen ein Ergebnis der Einhaltung der anderen Regeln

DHBW

9. Verfügbarkeit

„Maximize robustness with fast startup and graceful shutdown”

- GRAFIK

9. Verfügbarkeit

„Maximize robustness with fast startup and graceful shutdown”

Das sagen die Regeln:

- Eine 12FA lässt sich schnell starten/stoppen
- Eine 12FA lässt sich über einfache Unix-Signale beenden (SIGTERM)
- Eine 12FA fährt kontrolliert herunter
- Eine 12FA ist robust gegenüber Abstürzen
  - Bspw. aufgrund von Hardwarefehlern

DHBW

9. Verfügbarkeit

„Maximize robustness with fast startup and graceful shutdown”

Begründung:

- Schnelle Start-/Stoppbarkeit
  - Begünstigt Skalierbarkeit, Deployment und Änderungen an Konfiguration
- Beenden über Unix-Signale
  - SIGTERM: Standard-Weg zum sauberen Beenden von Prozessen (auf POSIX-kompatiblen Betriebssystemen)
  - Anwendung hat Gelegenheit, offene Arbeit zu beenden/kontrolliert abzubrechen

```
myhost:~$ kill 4711
```

  - Funktioniert am Besten mit self-contained Apps

DHBW

9. Verfügbarkeit

„Maximize robustness with fast startup and graceful shutdown”

Begründung:

- Kontrolliertes Herunterfahren /robust gegenüber Abstürzen
  - Anwendung nicht in undefiniertem Zustand belassen
  - Manuelle Aufräum-Arbeiten dürfen niemals nötig sein
  - Beispiel Herunterfahren einer Webapp:
    1. Keine neuen Requests annehmen
    2. Laufende Requests sauber beenden (**nicht** abbrechen)
    3. exit

DHBW

Exkurs: Crash-only Design

Grundidee:

- Eine Anwendung kann gar nicht heruntergefahren, sondern nur abgebrochen („gecrasht“) werden
- Dadurch soll die Fehlerbehandlung von der Ausnahme zur Regel werden

“Properly implemented, crash-only software produces higher quality, more reliable code; poorly understood it results in lazy programming.”

– *http://lwn.net/Articles/191059/, 04.03.2016*

DHBW

Exkurs: Design for Failure

- Es ist unmöglich, eine Anwendung gegen alle möglichen Fehler abzusichern
- Statt dessen: mit Fehlern rechnen und Anwendungen/Systeme so designen, dass Fehler sich nur begrenzt auswirken
- Nachteil: muss im Voraus in Design-Überlegungen einbezogen werden, hinterher schwieriger

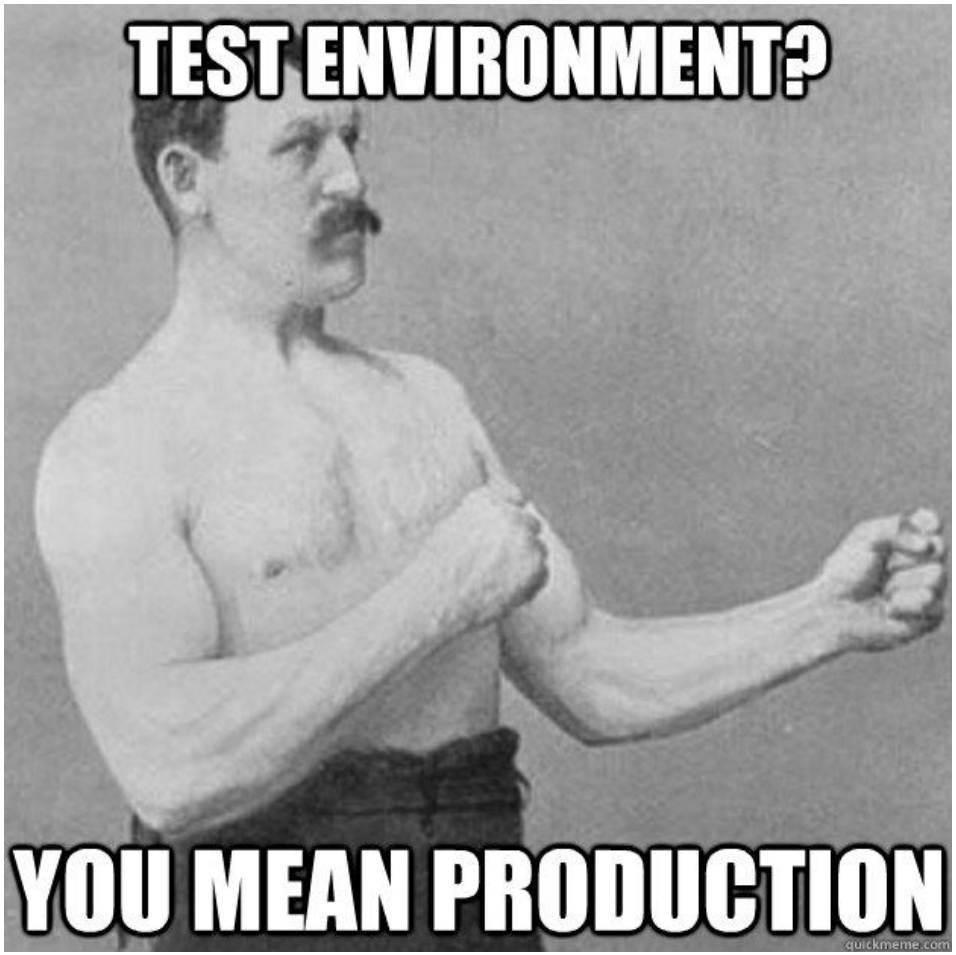
"…"design for failure" requires you to design for failure. It therefore significantly constrains your application architecture."

– *http://broadcast.oreilly.com/2011/04/the-aws-outage-the-clouds-shining-moment.html, 04.03.2016*

DHBW

10. Gleichheit der Umgebungen

„Keep development, staging, and production as similar as possible”



DHBW

10. Gleichheit der Umgebungen

„Keep development, staging, and production as similar as possible”

- Bei der Entwicklung einer 12FA ist die „Kluft“ zwischen verschiedenen Umgebungen so gering wie möglich
  - Zeitliche Kluft
  - Menschliche Kluft
  - Infrastruktur-Kluft
- Eine 12FA verwendet in der Entwicklung möglichst die selben unterstützenden Dienste wie in der Produktion

DHBW

10. Gleichheit der Umgebungen

„Keep development, staging, and production as similar as possible”

Zeitliche Kluft:

- Code ist jetzt fertig, wird aber erst nach Tagen/Wochen/Monaten deployed

Menschliche Kluft:

- Entwickler entwickeln, Administratoren deployen

Infrastruktur-Kluft:

- Unterschiedliche Tools/Backing Services/OS in Entwicklung und Produktion

DHBW

10. Gleichheit der Umgebungen

„Keep development, staging, and production as similar as possible”

	Traditional app	Twelve-factor app
Time between deploys	Weeks	Hours
Code authors vs code deployers	Different people	Same people (DevOps)
Dev vs production environments	Divergent	As similar as possible

10. Gleichheit der Umgebungen

„Keep development, staging, and production as similar as possible”

Begründung:

- Fehler sind leichter reproduzierbar
- Fehler werden früher gefunden
- Abstraktion (bspw. Hibernate) kann Unterschiede meist nicht vollständig beseitigen
  - „you run against Oracle, you develop against Oracle“

DHBW

10. Gleichheit der Umgebungen

„Keep development, staging, and production as similar as possible”

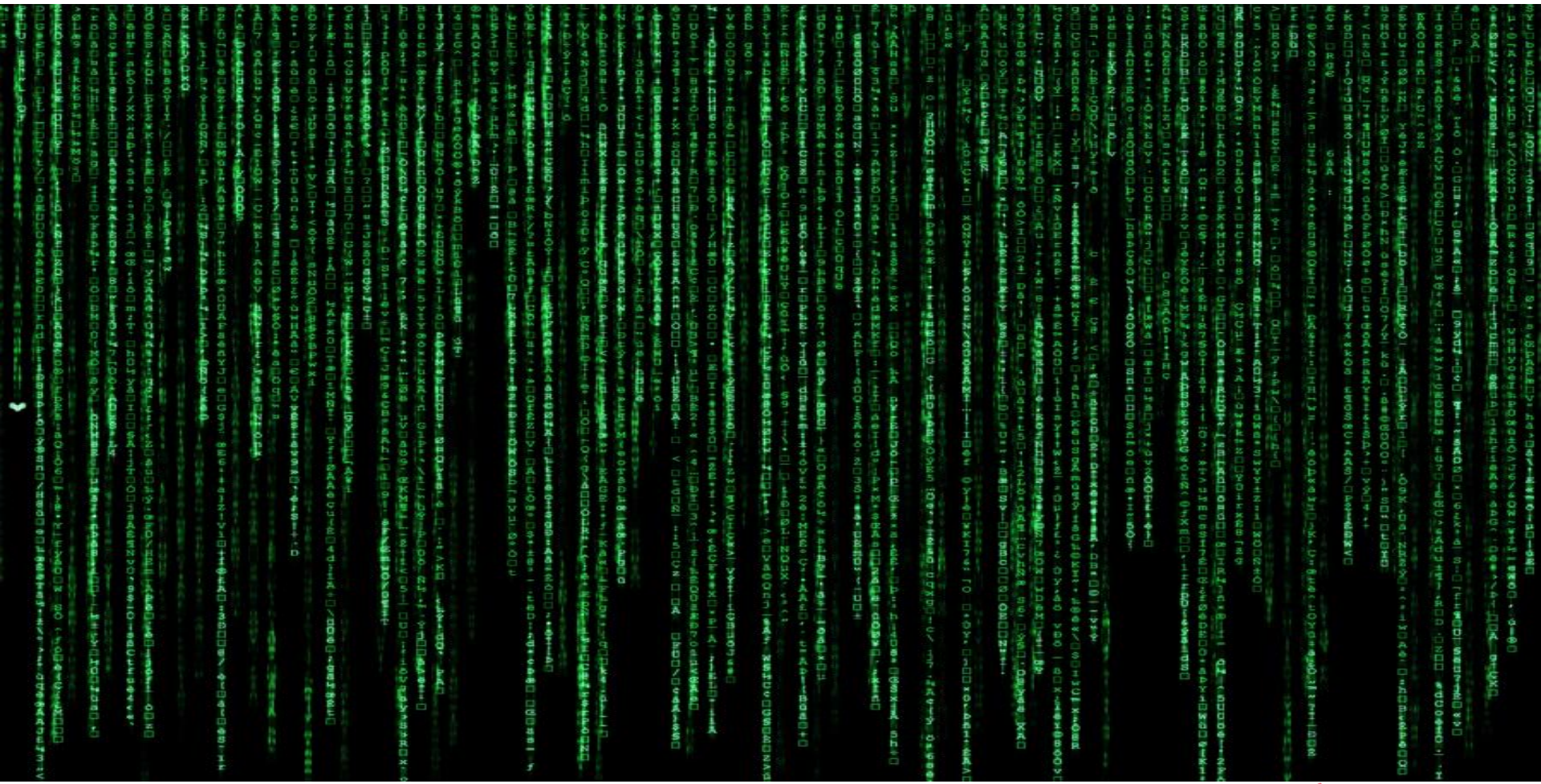
Kritik:

- Im Zeitalter von Virtualisierung und Containern sind gleiche Umgebungen wesentlich einfacher geworden
- Kosten/Nutzen abwägen
  - Unterschiedlicher Cache ok
  - unterschiedliche DB nicht ok
- Manchmal gar nicht möglich (bspw. besondere Hardware)

DHBW

11. Logging

„Treat logs as event streams“



11. Logging

„Treat logs as event streams“

- Eine 12FA kümmert sich nicht um das Speichern von Logs
- eine 12FA schreibt Log-Ausgaben immer direkt auf stdout
- Die Verarbeitung der Log-Ausgaben erfolgt durch die Infrastruktur
- -> **Logging ist Teil der Infrastruktur, nicht Teil der Anwendung!**

DHBW

11. Logging

„Treat logs as event streams“

Standard-Datenströme:

- Standard Input (stdin)
- Standard Output (stdout)
- Standard Error (stderr)

11. Logging

„Treat logs as event streams“

Begründung:

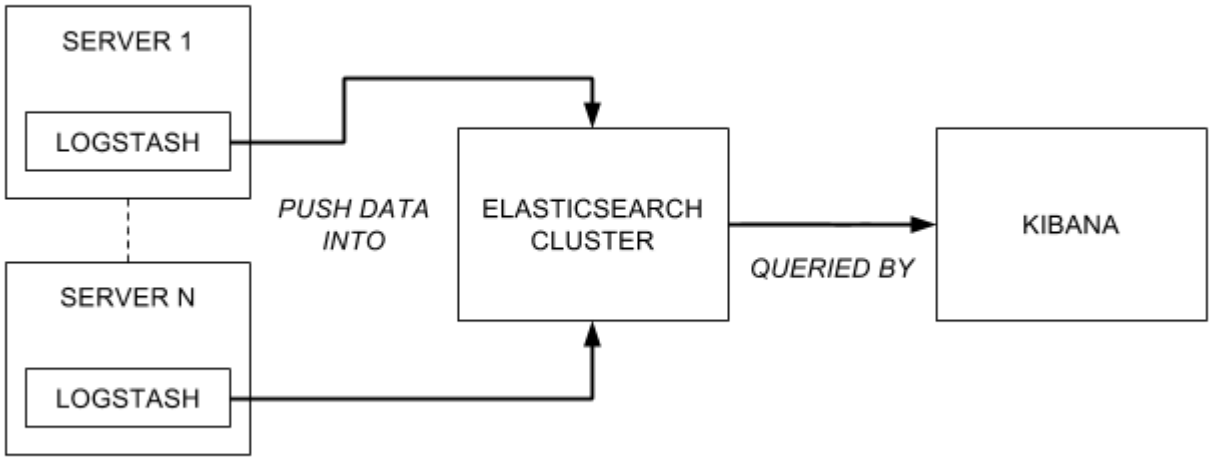
- Sammeln/Auswerten von Logs wird als Backing Service betrachtet
  - besser skalierbar
  - Höhere Flexibilität
  - App wird vereinfacht (weniger Verantwortung)
- Beispiel: systemd-Journal, splunk, ELK

Exkurs: ELK-Stack

- Elasticsearch
  - Suchmaschine/Index auf Basis von Lucene
- Logstash
  - Logs auf Server sammeln, normalisieren und an Elastic weiterleiten (“push”)
- Kibana
  - Visualisierungstool/UI für Elasticsearch

DHBW

Exkurs: ELK-Stack



DHBW

12. Administrations-Prozesse

12. Administrations-Prozesse

„Run admin/management tasks as one-off processes”

- Administrationsaufgaben sollten als **eigenständige Prozesse** ausgeführt werden, unabhängig von der eigentlichen Anwendung
- Administrationsaufgaben/-funktionen werden als **Teil der Codebase** verwaltet (Teil des Release)
  - >Für Admin-Aufgaben gelten die Regeln aus Punkt 2
- Administrationsaufgaben werden **in der selben Umgebung ausgeführt, in der die Anwendung läuft**
- Entwickler haben über **REPL** oder **Terminal Zugriff** auf Umgebung

DHBW

12. Administrations-Prozesse

„Run admin/management tasks as one-off processes”

Beispiele für Administrationsaufgaben:

- Migrationen
- Cache leeren
- Statusprüfung
- Interaktive Analyse

DHBW

12. Administrations-Prozesse

„Run admin/management tasks as one-off processes”

Beispiel DB-Migration:

- Migrations-Skript wird in Repository eingecheckt (**Teil der Codebase)**
- Migration wird im Zuge des Deployments ausgeführt - NICHT beim Start der Anwendung (**eigenständige Prozesse**)
- Migration wird auf dem Zielsystem gestartet, nicht vom lokalen Rechner gegen die Produktiv-DB (**selbe Umgebung, Zugriff)**

DHBW

12. Administrations-Prozesse

„Run admin/management tasks as one-off processes”

Begründung:

- Einflussmöglichkeiten auf Anwendung explizit machen (Skript/Task/…)
- Manuelle Einflussnahme (löschen von Verzeichnissen, Änderungen an DB, …) unterbinden
- Eigentlichen Anwendungsprozess nicht mit Admin-Aufgaben belasten

DHBW

Weiterführende Links

- http://highscalability.com/
- http://stackoverflow.com/questions/907260/how-to-design-scalable-applications
- http://broadcast.oreilly.com/2011/04/the-aws-outage-the-clouds-shining-moment.html
- https://www.usenix.org/legacy/events/hotos03/tech/full_papers/candea/candea.pdf
- https://www.innoq.com/de/podcast/018-twelve-factor-app/

DHBW