Ship It

Web Development

Vertiefungsmodul im Medieninformatik Bachelor

13.06.2019, Dirk Breuer

Planning

Photo by Manny Pantoja on Unsplash

Planning

Photo by Manny Pantoja on Unsplash

Coding

Photo by Steve Halama on Unsplash

Planning

Photo by Manny Pantoja on Unsplash

Coding

Photo by Steve Halama on Unsplash

Profit

Photo by Sharon McCutcheon on Unsplash

Schritte zum Erfolg

1. Software muss genutzt werden!

Ziel

Ziel

Anforderungen ermitteln

Ziel

Anforderungen ermitteln

Umsetzung der Anforderungen

Ziel

Anforderungen ermitteln

Umsetzung der Anforderungen

Auslieferung

Ziel

Anforderungen ermitteln

Umsetzung der Anforderungen

Auslieferung

Verifikation

Ziel

Anforderungen ermitteln

Umsetzung der Anforderungen

Auslieferung

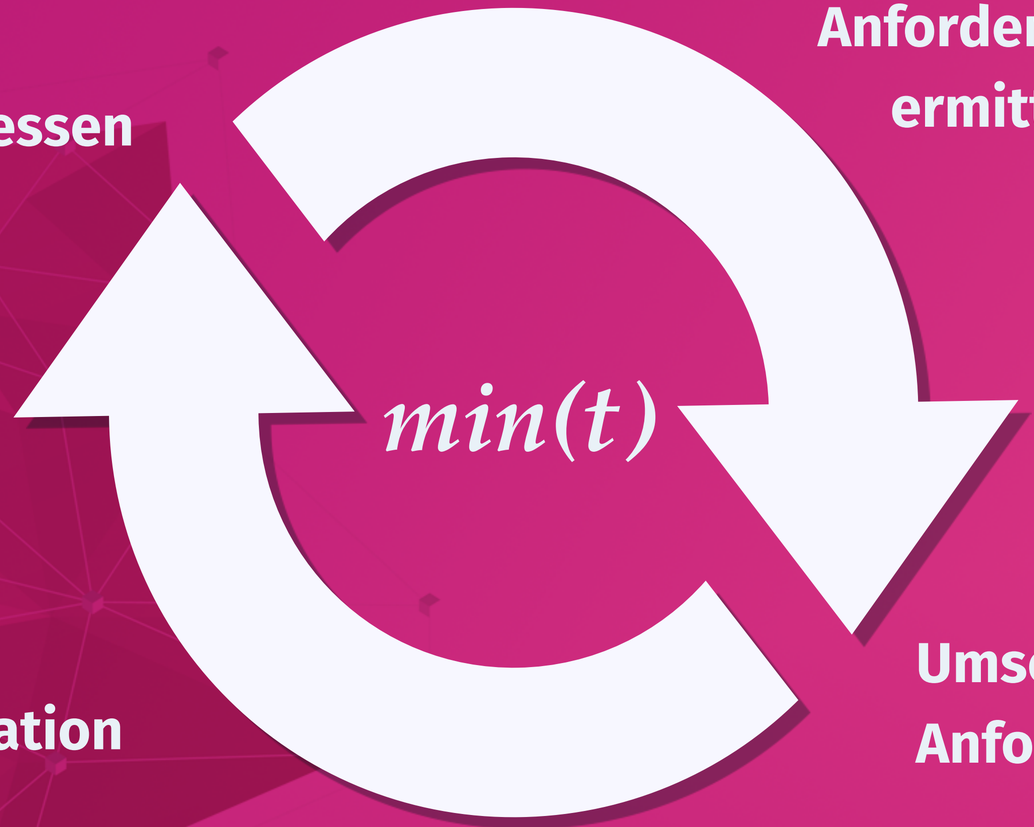
Verifikation

Erfolg messen

Ziel


Anforderungen ermitteln
Umsetzung der Anforderungen
Auslieferung
Verifikation
Erfolg messen

Ziel


Anforderungen ermitteln
Umsetzung der Anforderungen
Auslieferung
Verifikation
Erfolg messen
min(t)

1 Release / Quartal

VS

3 Releases / Tag

Aber…

- …ohne negativen Auswirkungen auf Nutzer
- …mit konstanter Flusszeit

Unterschiedliche Anforderungen

- Risiko eines Problems
- Auswirkungen eines Problems
- "Time to Recovery"
- Bsp.: Privates Blog gegenüber Reaktorsteuerung 😉

Software nutzen

- Bisher "lief" die Software auf der Umgebung der Entwickler
- Entwicklungsumgebung in der Regel nicht für breite Masse zugänglich
- Lösung: Software auf einer allgemein zugänglichen Umgebung "laufen" lassen

allgemein zugängliche Umgebung =

Produktionsumgebung

–

Software laufen lassen =

Betrieb (Operations, Ops)

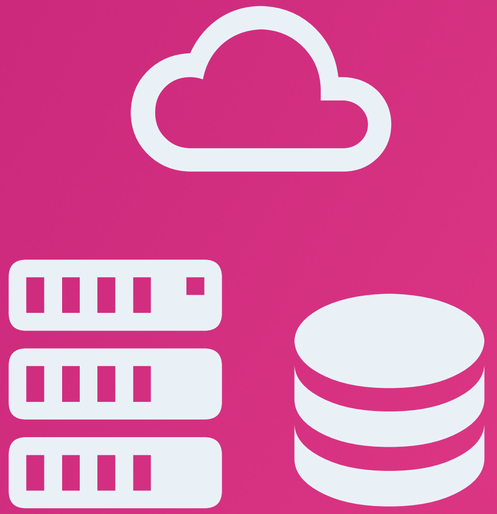
⚠ Disclaimer ⚠

Fokus auf "Webbasierte Anwendungen"

Ein Software Release Prozess sieht unterschiedlich für Mobile Anwendungen, Embedded Systeme oder Desktop Anwendungen aus.

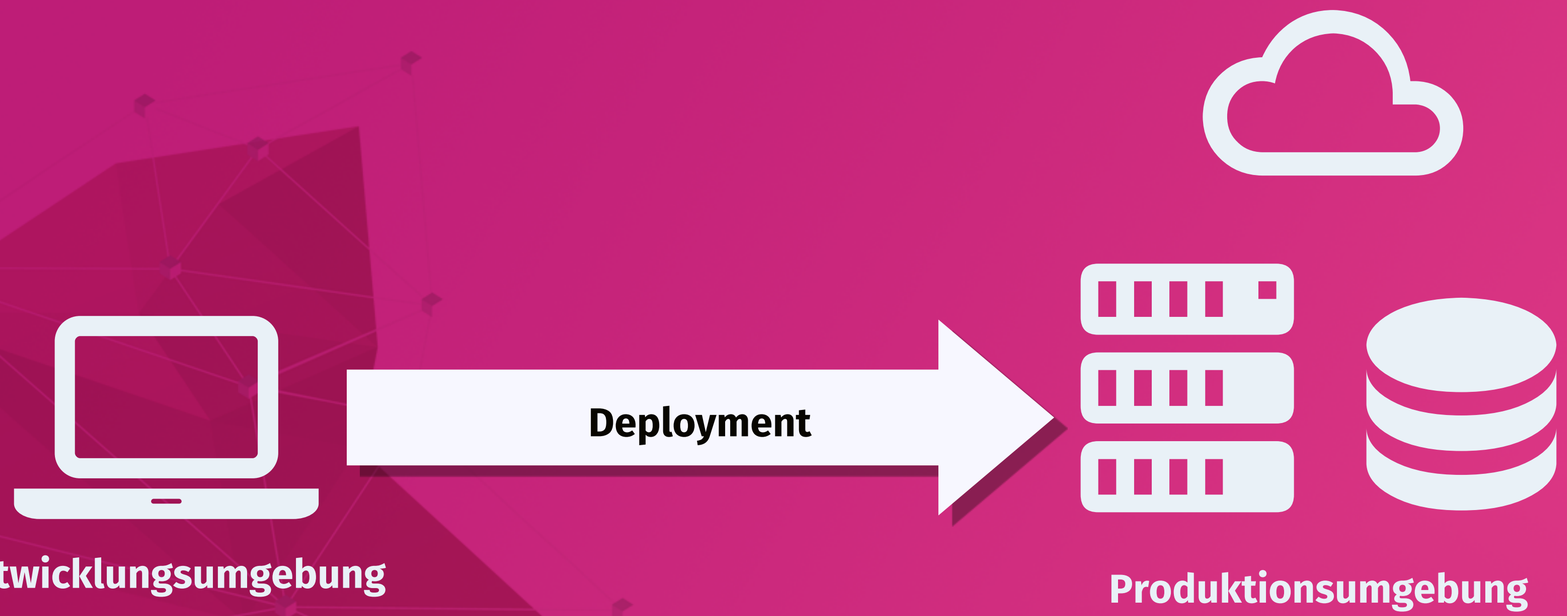
Entwicklungsumgebung

Entwicklungsumgebung



Produktionsumgebung


Entwicklungsumgebung
Produktionsumgebung


Deployment
Entwicklungsumgebung
Produktionsumgebung

Entwicklungsumgebung

- Entwicklungswerkzeuge
- Desktop-Betriebssystem
- Desktop-Hardware
- Aktuellste Software
- Zweifelhafte Backups
- Ausführung von Tests und Möglichkeiten zum Debugging
- Testdaten

Entwicklungsumgebung

- Entwicklungswerkzeuge
- Desktop-Betriebssystem
- Desktop-Hardware
- Aktuellste Software
- Zweifelhafte Backups
- Ausführung von Tests und Möglichkeiten zum Debugging
- Testdaten

Produktionsumgebung

- Nur das nötigste installiert
- Server-Betriebssystem
- Server-Hardware
- Stabile Software
- Umfangreiche Backups
- Keine Möglichkeit zur Ausführung von Tests / kein Debugging
- Echte Daten

Produktionsumgebung

Betriebssystem

Persistenz

Security

Anwendung

Logging

Performance

Updates

Fail-Over

Scaling

Produktionsumgebung

Abhängigkeiten

Web-Server

Zugriffskontrolle

Verteilung

Alerting

Monitoring

Datenbank

Backup

Deployment

“All activities required to make a software system available for use.”

Deployment

“Inbetriebsetzung”

Release

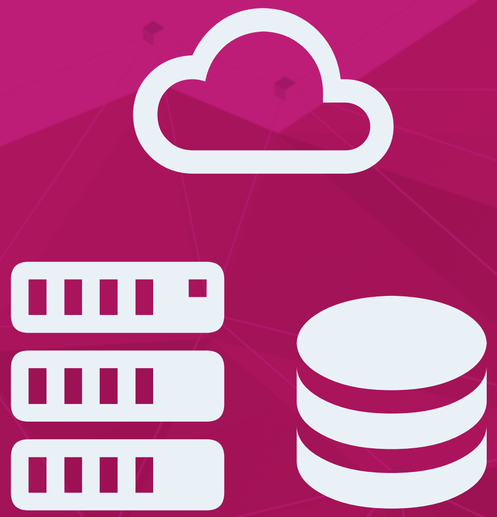
“Eine Version eines Softwaresystems mit definiertem Funktionsumfang und allen benötigten Abhängigkeiten.”

Release

≠

Deployment

n-Umgebungen

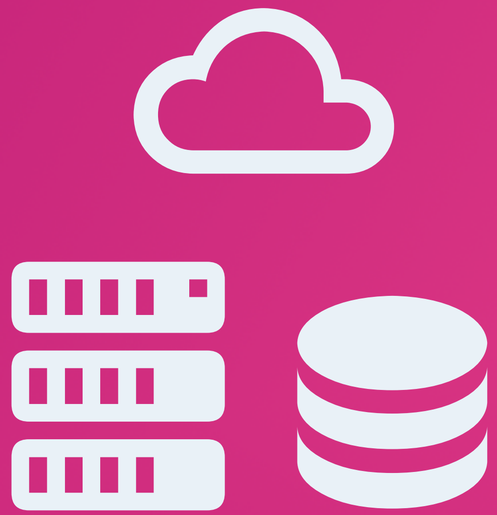


Produktionsumgebung

n-Umgebungen

Produktionsumgebung

Stagingumgebung



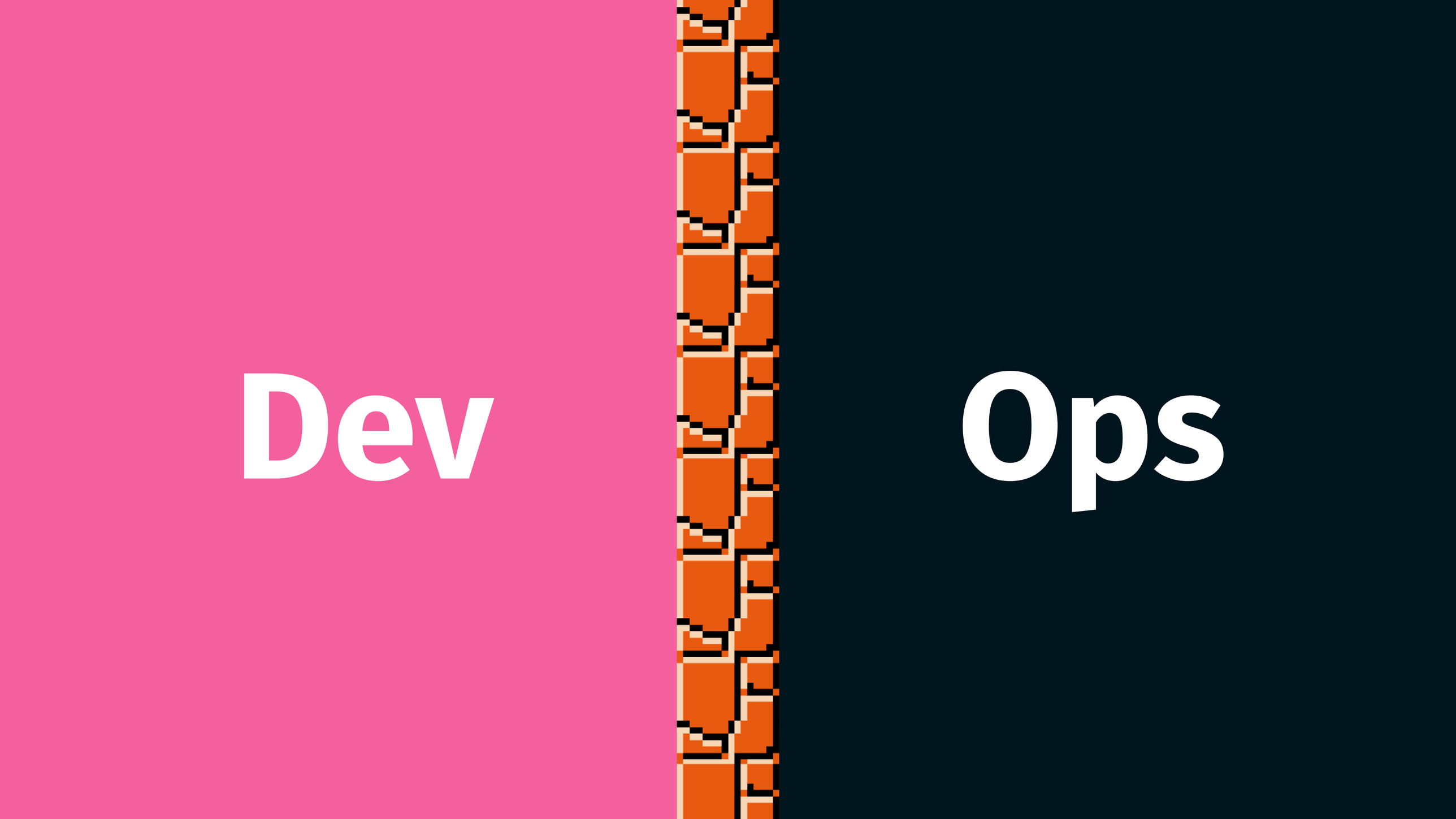
Testumgebung

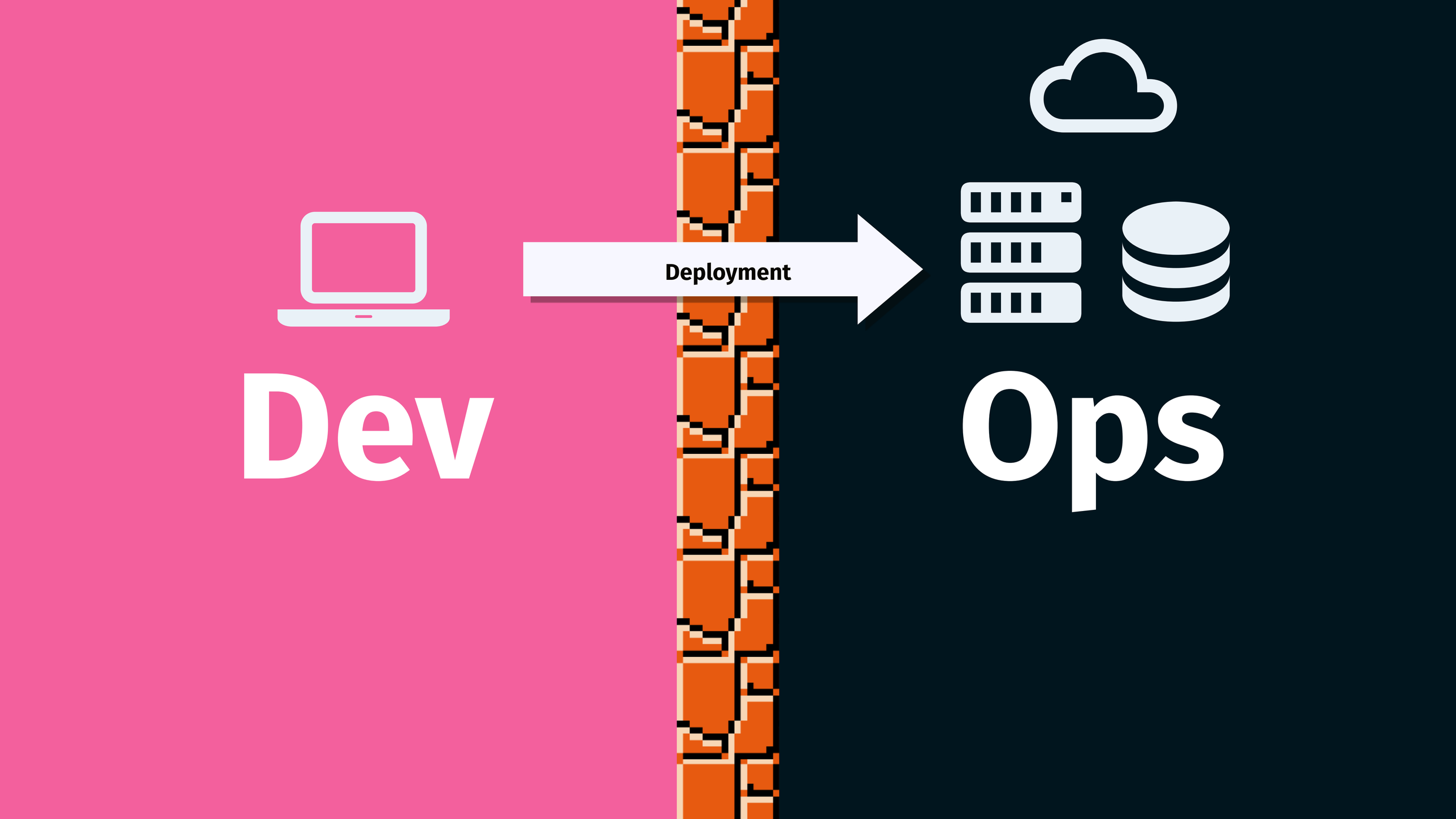
- Verantwortung über Infrastruktur
- Verantwortung über Betrieb
- 24/7 Support
- Eher konservativ

Ops

Dev

- Verantwortung über Entwicklung
- Umsetzung neuer Features
- Beheben von Fehlern
- Weit weg vom Betrieb
- Eher progressiv


Dev
Ops


Deployment
Dev
Ops

DevOps

DevOps

+ Bessere Kommunikation
+ Geteilte Verantwortung über Betrieb
+ Geteiltes Wissen über Anwendung und Infrastruktur
+ Adaption gegenseitiger Praktiken / Denkweisen

DevOps

- Geteilte Verantwortung über Betrieb
- Gegensätzliche Ziele: Sicherer Betrieb vs. Moderne Technologien

1. Paket bauen
2. Deployment durchführen
3. Profit!

Paket bauen

- JAR-File
- Debian Paket
- Docker Image
- git Commit
- …
- https://12factor.net/

1. Codebase – *One codebase tracked in revision control, many deploys*
2. Dependencies – *Explicitly declare and isolate dependencies*
3. Config – *Store config in the environment*
4. Backing services – *Treat backing services as attached resources*
5. Build, release, run – *Strictly separate build and run stages*
6. Processes – *Execute the app as one or more stateless processes*
7. Port binding – *Export services via port binding*
8. Concurrency – *Scale out via the process model*
9. Disposability – *Maximize robustness with fast startup and graceful shutdown*
10. Dev/prod parity – *Keep development, staging, and production as similar as possible*
11. Logs – *Treat logs as event streams*
12. Admin processes – *Run admin/management tasks as one-off processes*

1. Codebase – *One codebase tracked in revision control, many deploys*
2. Dependencies – *Explicitly declare and isolate dependencies*
3. Config – *Store config in the environment*
4. Backing services – *Treat backing services as attached resources*
5. Build, release, run – *Strictly separate build and run stages*
6. Processes – *Execute the app as one or more stateless processes*
7. Port binding – *Export services via port binding*
8. Concurrency – *Scale out via the process model*
9. Disposability – *Maximize robustness with fast startup and graceful shutdown*
10. Dev/prod parity – *Keep development, staging, and production as similar as possible*
11. Logs – *Treat logs as event streams*
12. Admin processes – *Run admin/management tasks as one-off processes*

Produktionsumgebung

Dev/Prod Parity

- Alle Umgebungen so gleich wie möglich halten
- Erleichtert die Entwicklung
- Reduzierte Komplexität
- Möglichkeiten
  - Zentrale Entwicklungsumgebung
  - Virtuelle Maschinen
  - Docker

- Dockerfile
- docker-compose.yml

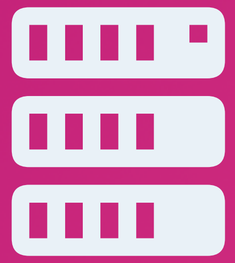
1. Paket bauen
2. Deployment durchführen
3. Profit!

1. Paket bauen
2. Deployment durchführen
3. Profit!

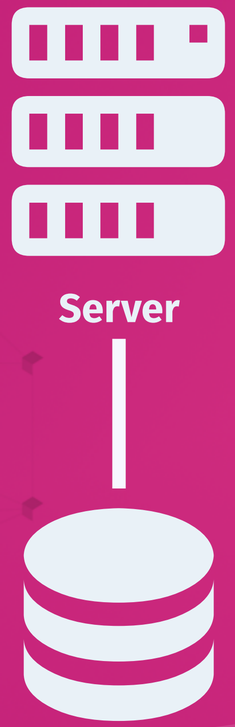
Und wo kommt die Umgebung her auf der das Deployment durchgeführt werden soll?

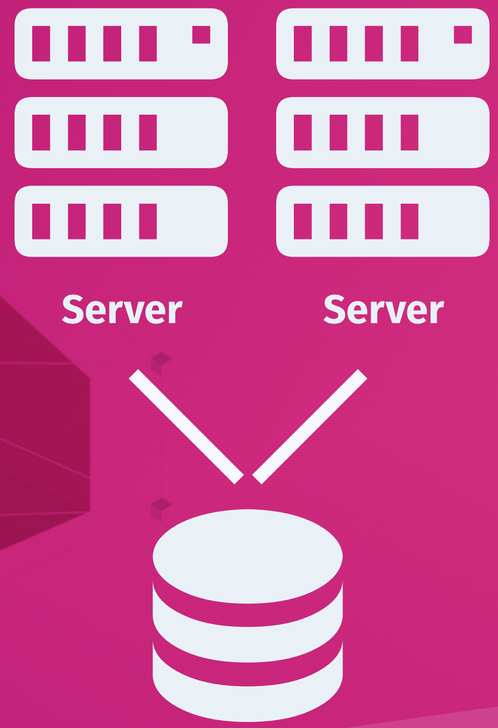
0. Bereitstellung einer Umgebung

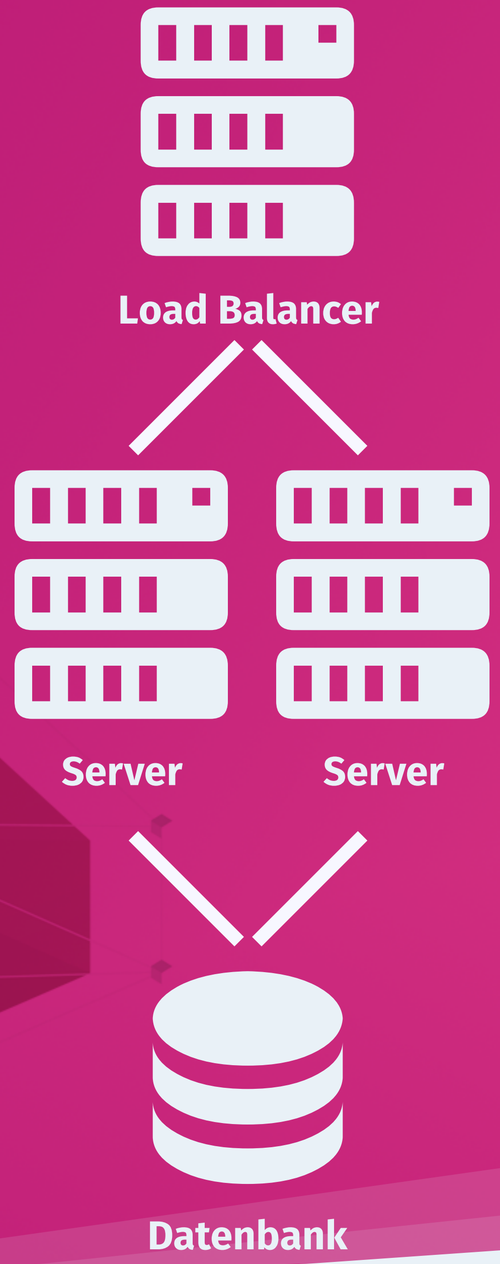
- Welche Anforderungen hat die Anwendung?
- Welche Abhängigkeiten hat die Anwendung?
- Welche Anforderungen an Sicherheit hat die Anwendung / das Unternehmen?
- Welche Anforderungen an Performance / Skalierbarkeit gibt es?

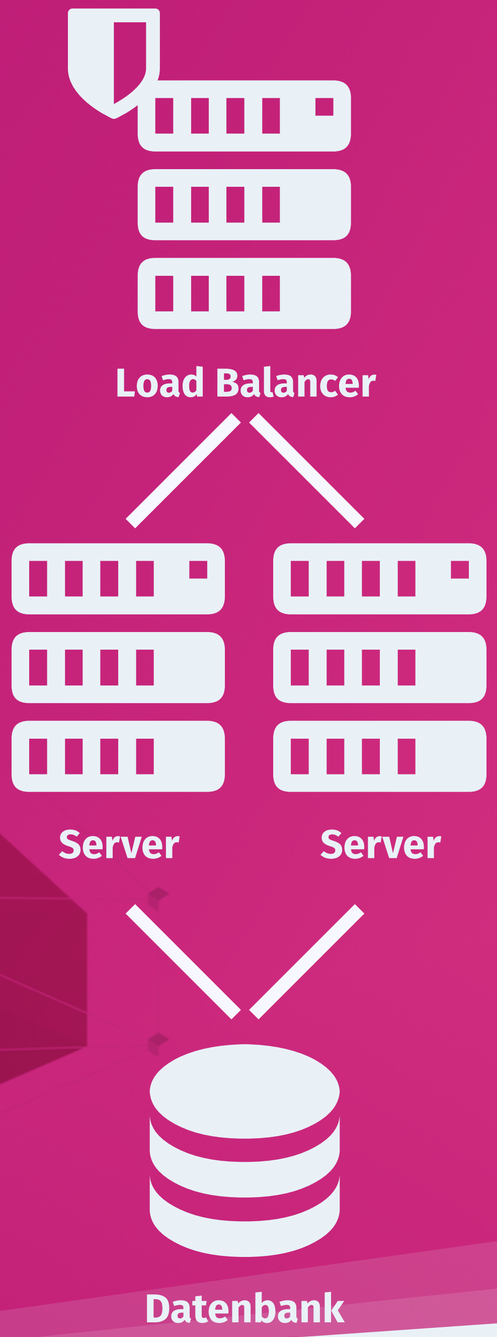


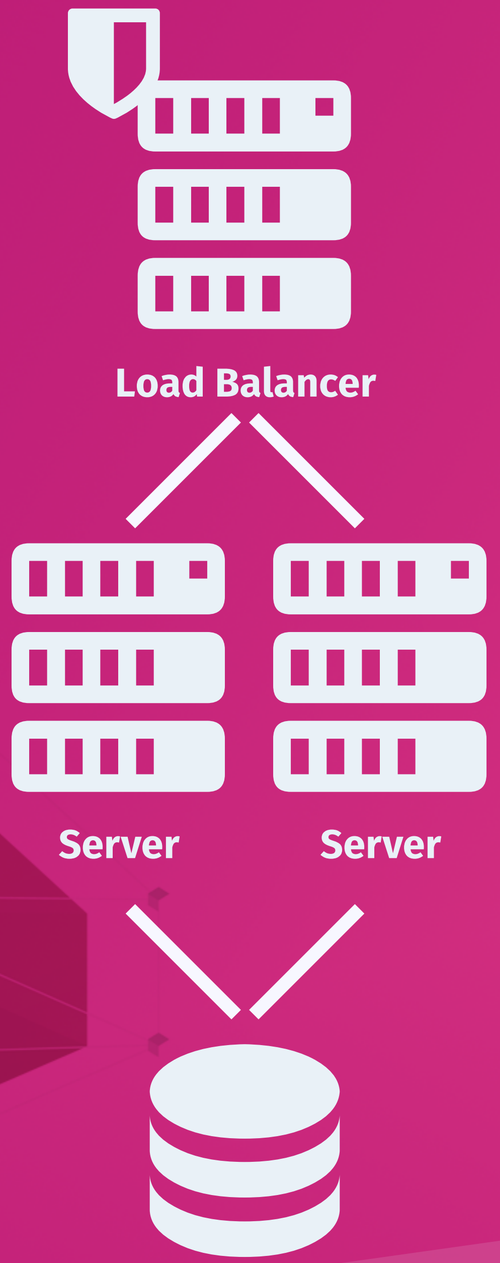
Server


Server
Datenbank

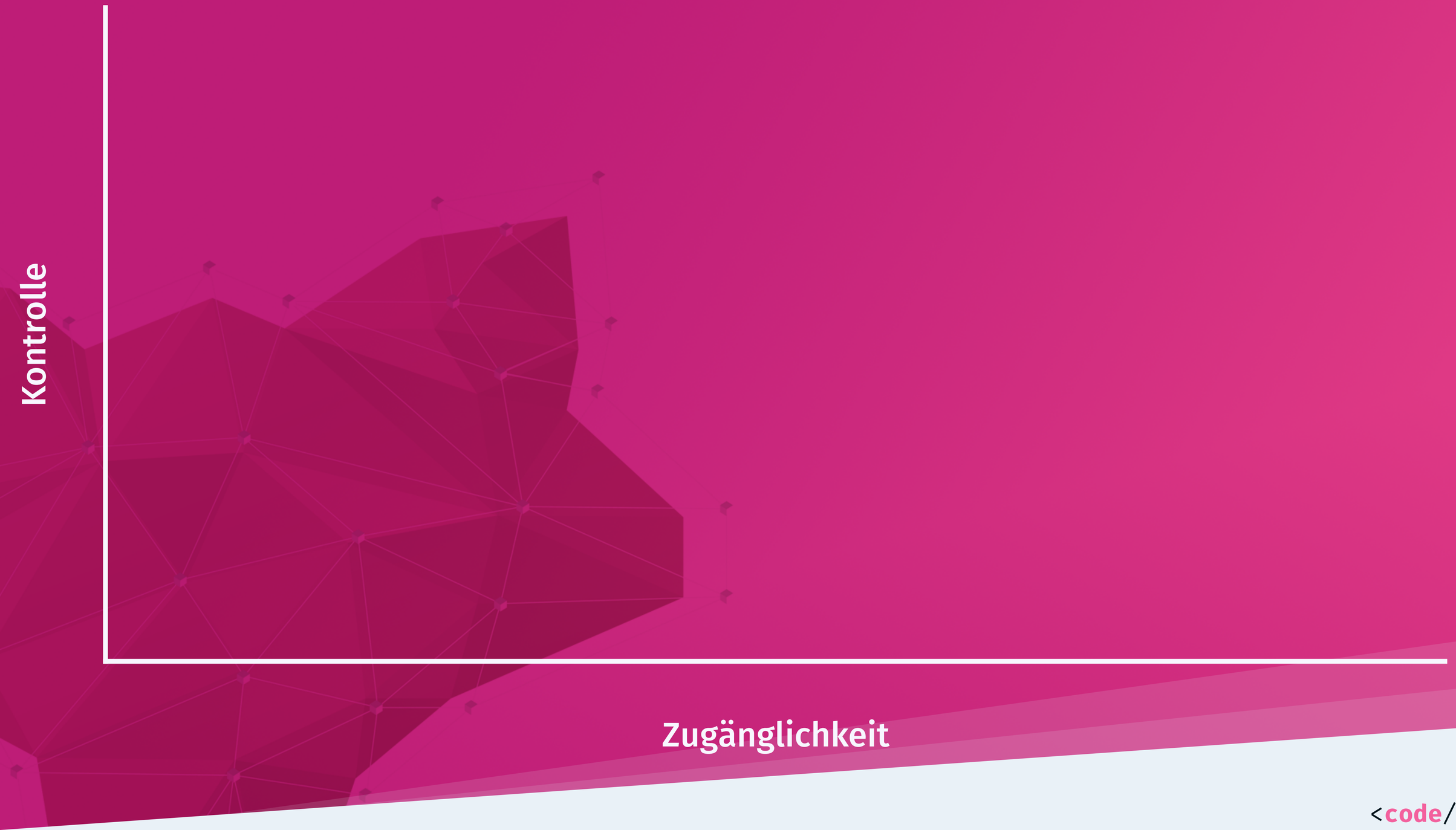

Server
Server
Datenbank


Load Balancer
Server
Server
Datenbank

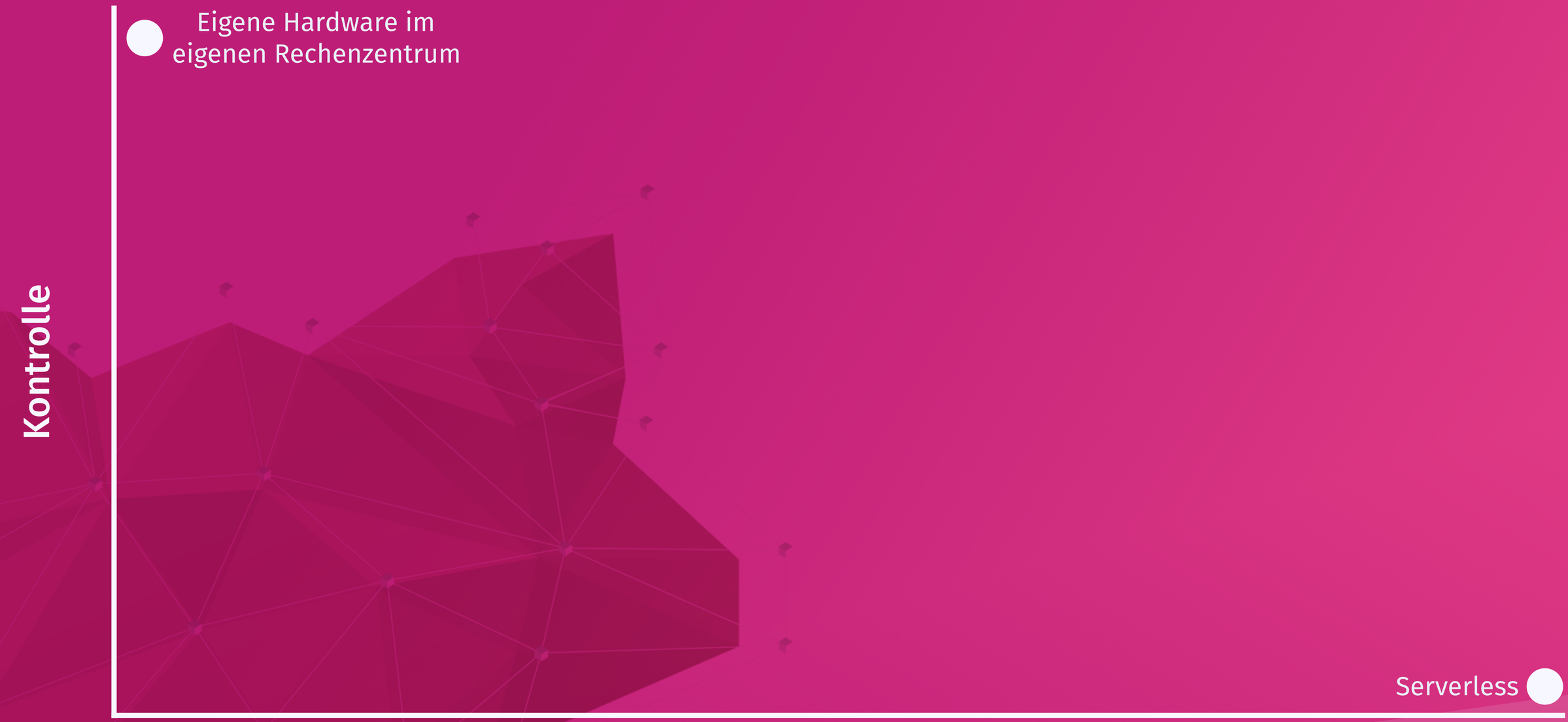

Load Balancer
Server
Server
Datenbank



- Logging
- Zugriffskontrolle
- Monitoring
- Alerting
- Backups
- Caches
- ...


Kontrolle
Zugänglichkeit


Eigene Hardware im
eigenen Rechenzentrum
Kontrolle
Zugänglichkeit


Eigene Hardware im
eigenen Rechenzentrum
Kontrolle
Serverless
Zugänglichkeit


Eigene Hardware im
eigenen Rechenzentrum
Root Server
Kontrolle
Serverless
Zugänglichkeit


Eigene Hardware im
eigenen Rechenzentrum
Root Server
Virtual
(Cloud) Server
Kontrolle
Serverless
Zugänglichkeit


Eigene Hardware im
eigenen Rechenzentrum
Root Server
Virtual
(Cloud) Server
Infrastructure
as a Service
Serverless
Kontrolle
Zugänglichkeit


Eigene Hardware im
eigenen Rechenzentrum
Root Server
Virtual
(Cloud) Server
Infrastructure
as a Service
Plattform as a
Service
Serverless
Kontrolle
Zugänglichkeit

Demo

Demo-Anwendung

- Todo-API
- Node.js mit Express.js
- PostgreSQL Datenbank

- Zielumgebung Ubuntu 19.04
- Installation von Node.js
- Installation von Webserver
- Installation von PostgreSQL
- Deployment der Demo-Anwendung
- TLS

https://www.digitalocean.com/community/tutorials

https://www.digitalocean.com/community/tutorials

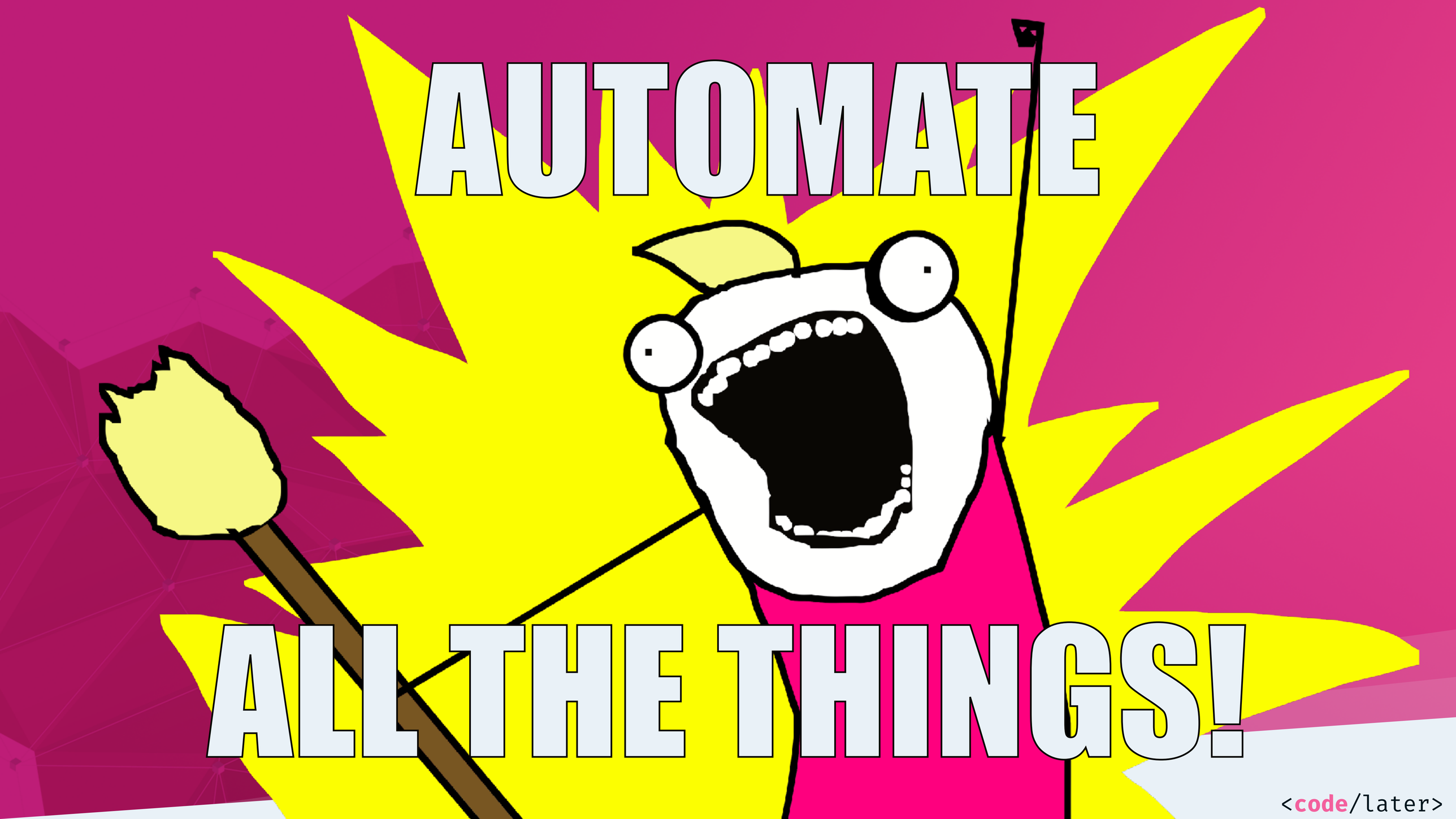
Aber...

- ...noch kein Backup
- ...kein zentrales Logging
- ...kein Monitoring
- ...kein Alerting
- ...keine Redundanz

Updates der Infrastruktur?!?

Probleme

- Manuell ausgeführte Schritte (meist von kopierten Befehlen)
- Repetitiv und damit Anfällig für Fehler
- Selbst sorgfältig geführte Checklisten helfen kaum
- Schlecht zu testen
- Kein (leichtes) "Rückgängig"
- Am Ende wird jeder Server ein bisschen anders sein


AUTOMATE
ALL THE THINGS!

Infrastructure as Code

“[…] is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.”

–https://en.wikipedia.org/wiki/Infrastructure_as_code

Beispiel: Ansible

```
- name: Add apt-key from Phusion
  apt_key: keyserver=keyserver.ubuntu.com id=561F9B9CAC40B2F7

- name: Add Phusion apt repo (for nginx)
   apt_repository:
     repo: 'deb https://oss-binaries.phusionpassenger.com/apt/passenger {{ ansible_distribution_release }} main'
     state: present
  notify:
    - apt-get update

- name: Install nginx
  apt: name=nginx-extras

- name: Start nginx service
  service: name=nginx state=started enabled=yes

- name: Remove default nginx site
  file: path=/etc/nginx/sites-enabled/default state=absent

- name: Configure /nginx_status endpoint for metrics
  template: src=roles/common/templates/etc/nginx/sites-enabled/nginx_status.conf.j2 dest=/etc/nginx/sites-enabled/nginx_status
  notify:
    - reload nginx
```

Beispiel: Packer

```
---
variables:
  aws_access_key_id: "{{env `AWS_ACCESS_KEY_ID`}}"
  aws_secret_access_key: "{{env `AWS_SECRET_ACCESS_KEY`}}"
builders:
- type: amazon-ebs
  access_key: "{{user `aws_access_key_id`}}"
  secret_key: "{{user `aws_secret_access_key`}}"
  region: eu-central-1
  source_ami: ami-05af84768964d3dc0
  instance_type: c4.2xlarge
  iam_instance_profile: packer-ami-builder
  ssh_username: ubuntu
  ami_name: "web-{{ isotime | clean_ami_name }}"
  tags:
    Name: Application Server
provisioners:
- type: ansible-local
  playbook_file: packer/ansible/application_server.yml
  inventory_file: packer/ansible/inventory
  playbook_dir: packer/ansible
  galaxy_file: packer/ansible/requirements.yml
  extra_arguments:
  - "-e main_user=ubuntu"
min_packer_version: 1.3.4
```

Beispiel: Terraform

```
provider "aws" {
  access_key = "${var.aws_access_key_id}"
  secret_key = "${var.aws_secret_access_key}"
  region = "${var.aws_region}"
}

resource "aws_instance" "web" {
  ami = "${var.web_ami}"
  instance_type = "${var.web_instance_type}"
  vpc_security_group_ids = ["${aws_security_group.web.id}"]
  subnet_id = "${var.public_subnets[0]}"
  monitoring = true

  tags {
    Name = "${var.environment}-web"
    stage = "${var.environment}"
  }
}
```

Beispiel: Dockerfile

```
FROM node:11

# Create app directory
WORKDIR /usr/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
COPY package*.json ./

RUN npm install --quiet

# Bundle app source
COPY . .

EXPOSE 3000

CMD [ "npm", "start" ]
```

Beispiel: docker-compose.yml

```yaml
version: '3'
services:
  web:
    build: .
    command: npm start
    volumes:
      - .:/usr/app/
      - node_modules:/usr/app/node_modules
    ports:
      - "3000:3000"
    stdin_open: true
    tty: true
    depends_on:
      - postgres
    environment:
      PORT: 3000
      DATABASE_URL: postgres://todoapp:supersecretpassword@postgres/todos
  postgres:
    image: postgres:11
    volumes:
      - postgres:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: todoapp
      POSTGRES_PASSWORD: supersecretpassword

volumes:
  postgres:
  node_modules:
```

Infrastructure as Code

- Infrastruktur als (ausführbaren) Code
- Änderungen an der Infrastruktur sind Änderungen am Code
- Code kann / sollte idempotent ausführbar sein
- Nachhalten der Änderungen in Versionskontrolle
- Infrastruktur lässt sich reproduzier von 0 auf konfigurieren
- Als Code unterliegt er den bekannten Regeln: **Reviews & Tests**

- Shell-Skripte
- Ansible
- Puppet
- Terraform
- Saltstack
- Packer
- Chef
- Docker
- Kubernetes (k8s)
- …

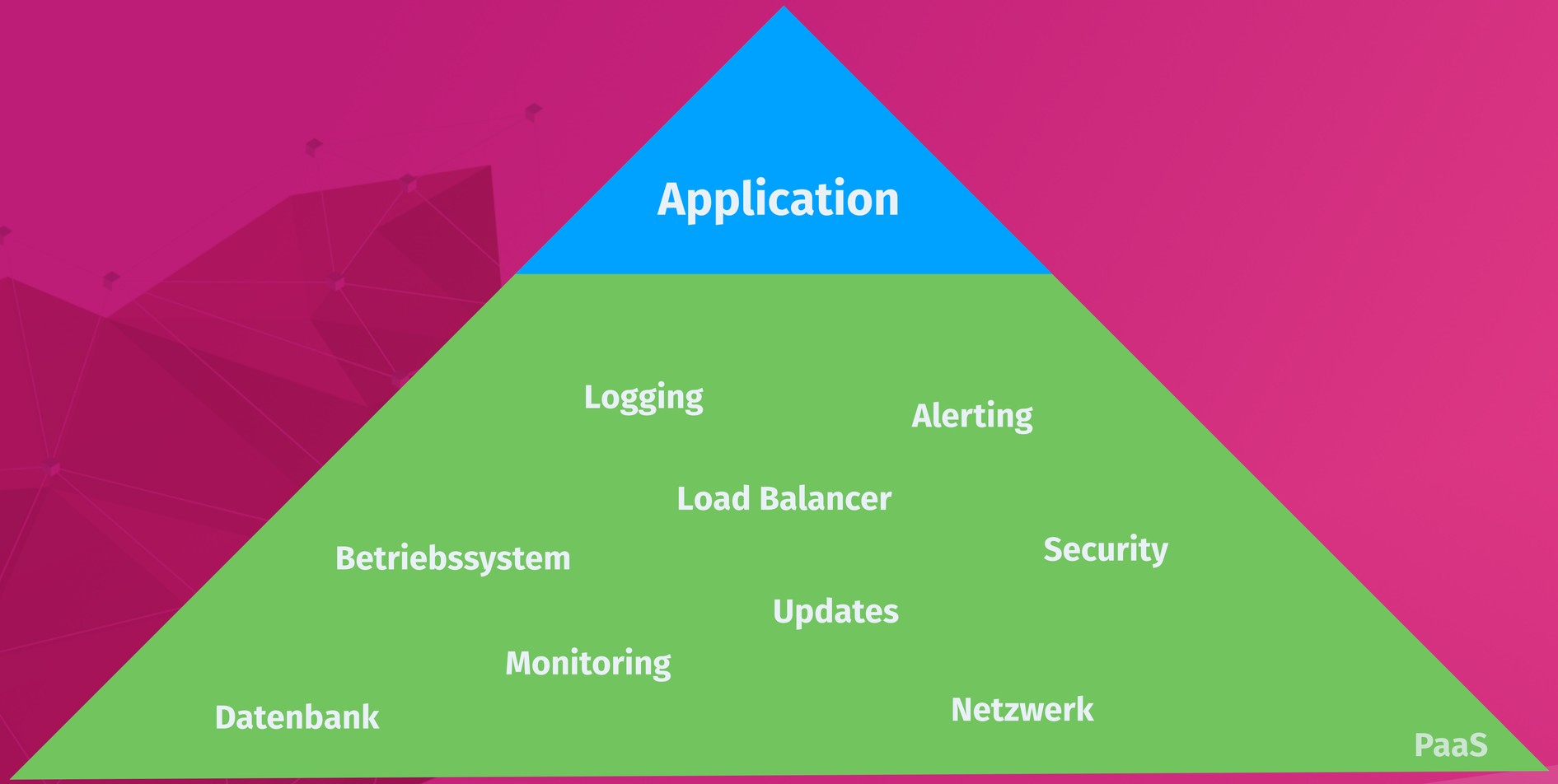

IS THAT THE BEST YOU CAN DO?


IS THAT THE BEST YOU CAN DO?

PaaS

PaaS

- Abstrahiert alle wesentlichen Komponenten der Infrastruktur
- Bietet Dienste zur Nutzung an
- Verwaltet Ressourcen
- Stellt Garantieren über genutzte Dienste aus (Verfügbarkeit, Performance, etc)


Application
Logging
Alerting
Load Balancer
Betriebssystem
Security
Updates
Monitoring
Datenbank
Netzwerk
PaaS

What if...?

```
$ git push paas master
```

HEROKU

Hands-On

- Heroku Account erstellen
- Beispiel-Anwendung forken
  - https://github.com/code-later/todo-demo.codelater.de
- Deployment nach Heroku

```
$ cd todo-demo
$ heroku login
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/browser/12444874-35c3c-41fb-9415-6b647ga6c080b
Logging in... done
Logged in as johndoe@me.com
$ heroku git:remote -a todo-demo-prod
set git remote heroku to https://git.heroku.com/todo-demo-prod.git
git push heroku master
Enumerating objects: 31, done.
Counting objects: 100% (31/31), done.
Delta compression using up to 8 threads
Compressing objects: 100% (27/27), done.
Writing objects: 100% (31/31), 16.27 KiB | 3.25 MiB/s, done.
Total 31 (delta 9), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:        https://todo-demo-prod.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/todo-demo-prod.git
 * [new branch]      master → master
```

```
$ http --json https://todo-demo-prod.herokuapp.com/todos
HTTP/1.1 500 Internal Server Error
Connection: keep-alive
Content-Length: 49
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Jun 2019 13:52:44 GMT
Etag: W/"31-N3CMxmHVH2QRVY7BhlpI3H4+usU"
Server: Cowboy
Via: 1.1 vegur
X-Powered-By: Express

{
    "message": "connect ECONNREFUSED 127.0.0.1:5432"
}
```

```
$ heroku addons:create heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on ⬢ todo-demo-prod... free
Database has been created and is available
 ! This database is empty. If upgrading, you can transfer
 ! data from another database with pg:copy
Created postgresql-horizontal-58429 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
```

```
$ http --json https://todo-demo-prod.herokuapp.com/todos
HTTP/1.1 500 Internal Server Error
Connection: keep-alive
Content-Length: 49
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Jun 2019 13:52:44 GMT
Etag: W/"31-N3CMxmHVH2QRVY7BhlpI3H4+usU"
Server: Cowboy
Via: 1.1 vegur
X-Powered-By: Express

{
    "message": "relation \"todos\" does not exist"
}
```

```
$ heroku run npx db-migrate up
Running npx db-migrate up on ⬢ todo-demo-prod... up, run.7109 (Free)
[INFO] Processed migration 20190608110952-createTodos
[INFO] Done
```

```
$ http --json https://todo-demo-prod.herokuapp.com/todos
HTTP/1.1 500 Internal Server Error
Connection: keep-alive
Content-Length: 49
Content-Type: application/json; charset=utf-8
Date: Mon, 10 Jun 2019 13:52:44 GMT
Etag: W/"31-N3CMxmHVH2QRVY7BhlpI3H4+usU"
Server: Cowboy
Via: 1.1 vegur
X-Powered-By: Express

[]
```


Anforderungen ermitteln
Umsetzung der Anforderungen
Auslieferung
Verifikation
Erfolg messen

What if...?

- Änderungen am Code durchführen
- `git push origin awesome-new-feature`
- Automatisierte Tests laufen
- App wird automatisch auf eine temporäre Umgebung deployed
- Funktionale Abnahme und Code Review
- `git merge awesome-new-feature`
- Änderungen werden auf Produktionsumgebung deployed

Continuous Delivery / Deployment

- In kurzen Zyklen abgeschlossene Zustände einer Software produzieren
- Jeder Stand ist funktional (ggf. auch nicht-funktional) verifiziert worden
- Kann jederzeit an Kunden ausgeliefert werden
- (Vollständige) Automatisierung des Deployment-Prozess

Hands-On

- Einfachen Test hinzufügen
- TravisCI einrichten
- Pull-Request Branch automatisch auf Heroku deployen
- Nach Merge in master Branch Deployment auf Heroku

Schritt 1

- Aktuelle `server.js` aufspalten
- Business Logik nach `app.js`
- HTTP in `server.js` belassen (`listen()`)

Schritt 2

- `npm install --save-dev supertest jest`
- Konfiguration von jest

Schritt 3

- Einfachen Test:
  - Nur HTTP Status Code prüfen
  - **Keine** Inhalte
  - Warum? Brauchen eine zweite Datenbank für die Tests.

Schritt 4

- `.travis.yml` hinzufügen
- TravisCI einrichten
- Branch pushen
- PR erstellen

Schritt 5

- Heroku Preview Apps einrichten
- Automatische Deployments auf Heroku einrichten

Schritt 6

- Der Anwendung eine `DELETE` Funktion hinzufügen
- PR erstellen
- Preview App testen
- In `master` mergen
- Automatisches Deployment testen

Fragen?