TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Leon Hess
Lehrstuhl für Rechnernetze

Im Team Software entwickeln

Service and Cloud Computing // 29.10.2019

Gliederung

1. Einführung
    1. Szenario
    2. Herausforderungen
    3. Lösungen
    4. REST-Service mit Python Flask
2. Git
    1. Was ist Git?
    2. Download Git & more
    3. Wie funktioniert Git
    4. Commandline vs. GUI
3. Docker
    1. Was ist Docker?
    2. Komponenten von Docker
    3. Download Docker
    4. Dockerzyklus
    5. Dockerfile
    6. Vom Dockerfile zum laufenden Container
    7. Client und Service verbinden

1. Einführung

Szenario:

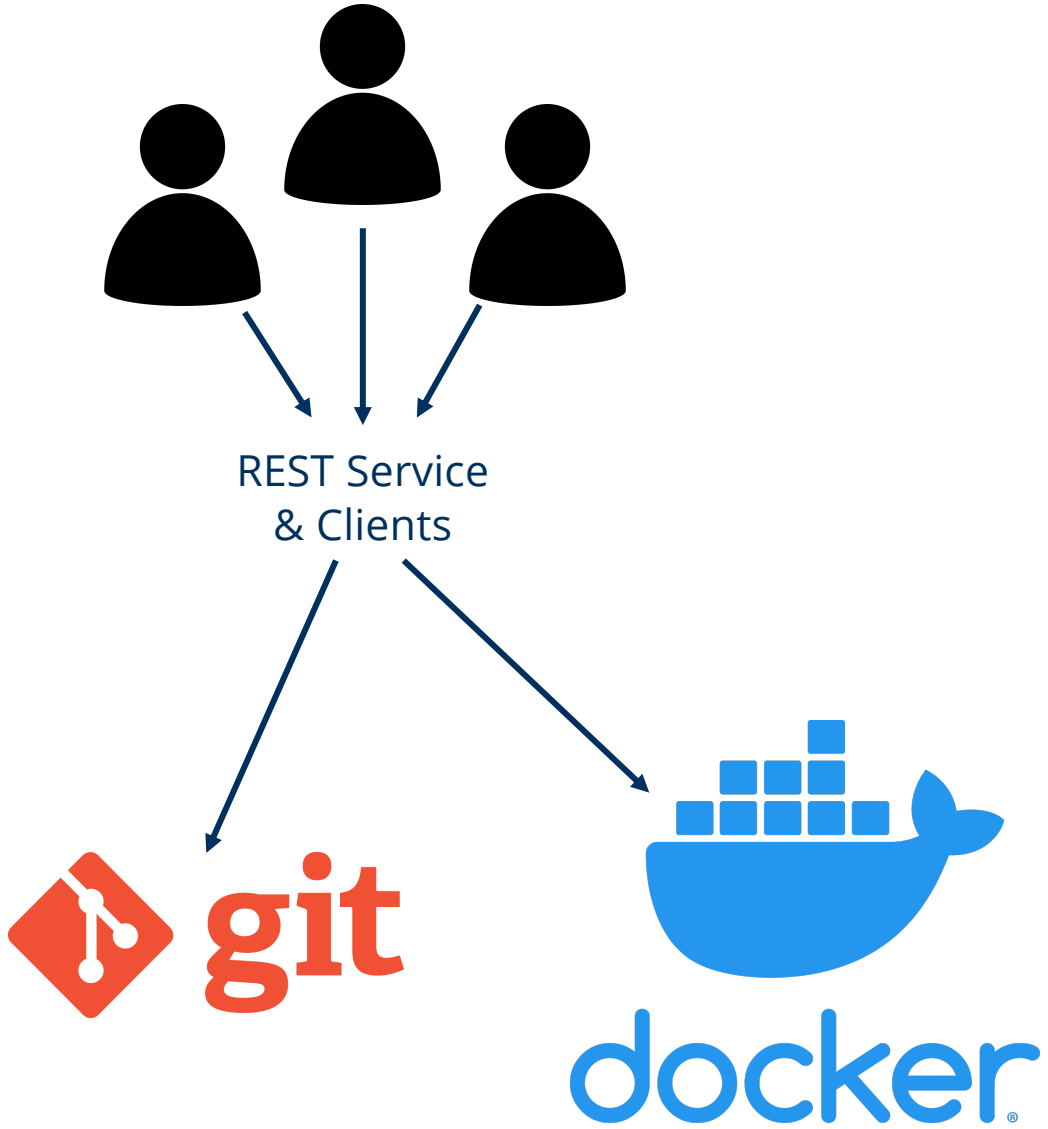
- Ein Entwicklerteam soll einen REST Service und mehrere Clients bauen

Herausforderungen:

- Wie verwaltet man die enstandene Software?
- Wie kann man die Software überall und schnell ausführen?

Lösungen:

- Version Control System (VCS) – konkret Git
- Container – konkret Docker



REST Service mit Python Flask

Studentenverwaltung:

- Alle Studenten ausgeben
- Einen Studenten ausgeben
- Student hinzufügen
- Student aktualisieren
- Student löschen

REST HTTP Methoden:

- GET – fordert Daten an
- POST – übermittelt Daten
- PUT – ändert bestehende Daten
- DELETE – löscht besthende Daten

```
students = [
    {
        "id": 1,
        "first_name": “Max",
        "last_name": “Mustermann",
        "mat_nr": 12345
    },
    {
        "id": 2,
        "first_name": “Kai",
        "last_name": “Müller",
        "mat_nr": 54321
    }
]
```

REST Service mit Python Flask

GET Methoden:

- Alle Studenten ausgeben

```
@app.route('/api/students', methods=['GET'])
def get_students():
    return jsonify({"students": students})
```

- Einen Studenten ausgeben

```
@app.route('/api/student/<int:student_id>', methods=['GET'])
def get_student(student_id):
    student = [student for student in students if student["id"] == student_id]
    if len(student) == 0:
        abort(404)
    return jsonify({"student": student[0]})
```

REST Service mit Python Flask

POST Methoden:

- Student hinzufügen

```
@app.route('/api/students', methods=['POST'])
def create_student():
    if not request.json:
        abort(400)
    if len(students) != 0:
        id = students[-1]["id"] + 1
    else:
        id = 1

    student = {
        "id": id,
        "first_name": request.json["first_name"],
        "last_name": request.json["last_name"],
        "mat_nr": request.json["mat_nr"]
    }
    students.append(student)
    return jsonify({"student": student})
```

REST Service mit Python Flask

PUT Methoden:

- Student aktualisieren

```
@app.route('/api/students/<int:student_id>', methods=['PUT'])
def update_student(student_id):
    student = [student for student in students if student['id'] == student_id]
    if len(student) == 0:
        abort(404)
    if not request.json:
        abort(400)

    student[0]["first_name"] = request.json.get("first_name", student[0]["first_name"])
    student[0]["last_name"] = request.json.get("last_name", student[0]["last_name"])
    student[0]["mat_nr"] = request.json.get("mat_nr", student[0]["mat_nr"])
    return jsonify({"student": student[0]})
```

REST Service mit Python Flask

DELETE Methoden:

- Student löschen

```
@app.route('/api/students/<int:student_id>', methods=['DELETE'])
def delete_student(student_id):
    student = [student for student in students if student["id"] == student_id]
    if len(student) == 0:
        abort(404)
    students.remove(student[0])
    return jsonify({"result": True})
```

Routen:

/api/students	← Alle Studenten anzeigen, Student hinzufügen
/api/students/<student_id>	← Student anzeigen, aktualisieren, löschen

2. Git

Was ist Git?

- Version Control System → verschiedene Versionen werden in Repositories abgespeichert
- Ermöglicht das Verwalten einer Vielzahl von verteilten Dokumenten/Dateien
- Automatisches zusammenführen (merge) von Änderungen
- Einfaches Zurückrollen auf eine ältere Version

Download Git & more

Debian/Ubuntu:

```
sudo apt install git-all
```

Windows:

https://git-scm.com/download/windows

MacOS:

https://git-scm.com/download/mac

Download the REST Service and Client:

https://github.com/leonhess/rest-uebung/archive/master.zip

Make an Account on GitHub:

https://github.com/join

Wie funktioniert Git

Repository wird initialisiert	`git init`
Erster commit	
1. Show repository status	`git status`
2. Add files to staging	`git add src/rest-service`

lokal
master ⟶⟵ HEAD

Wie funktioniert Git

Repository wird initialisiert

```
git init
```

Erster commit

1. Show repository status

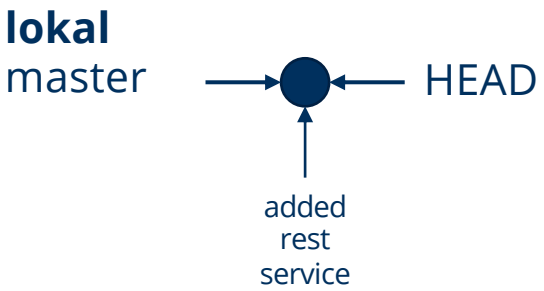
```
git status
```

2. Add files to staging

```
git add src/rest-service
```

3. Commit new files

```
git commit –m “added rest service”
```



Wie funktioniert Git

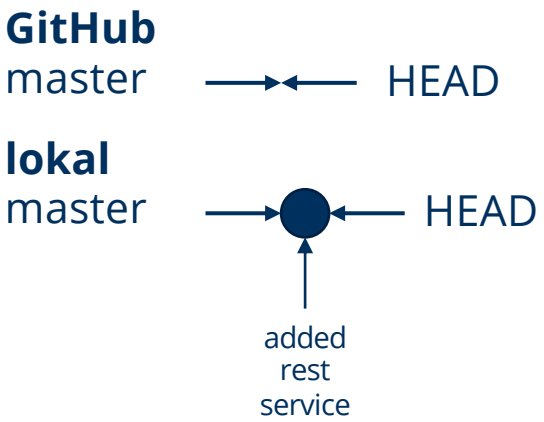
Mit Github repository verbinden

1. Add remote repo url

```
git add origin <url>
```

2. Show all remote repos

```
git remote -v
```



Wie funktioniert Git

Mit Github repository verbinden

1. Add remote repo url

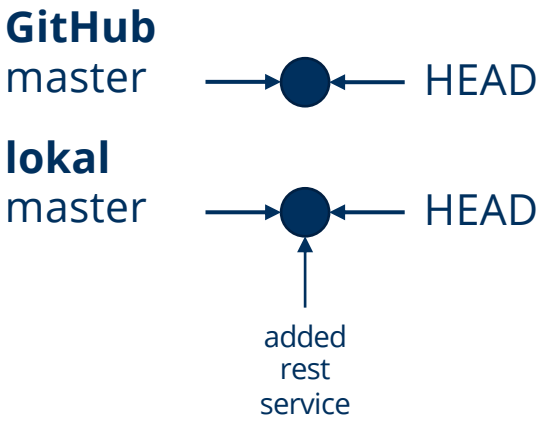
```
git add origin <url>
```

2. Show all remote repos

```
git remote -v
```

3. Push first commit to GitHub

```
git push origin master
```

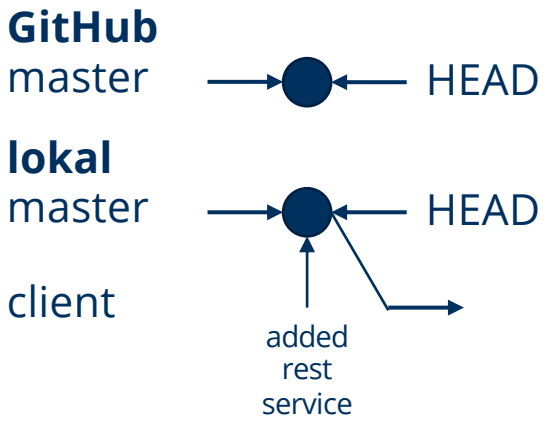


Wie funktioniert Git

REST-Client über Branch hinzufügen

1. Create a new Branch

```
git branch client
```



Wie funktioniert Git

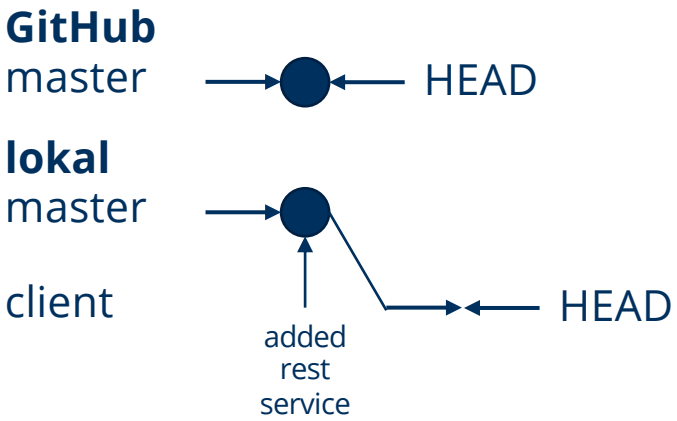
REST-Client über Branch hinzufügen

1. Create a new Branch

```
git branch client
```

2. Switch to new branch

```
git checkout client
```



Wie funktioniert Git

REST-Client über Branch hinzufügen

1. Create a new Branch

```
git branch client
```

2. Switch to new branch

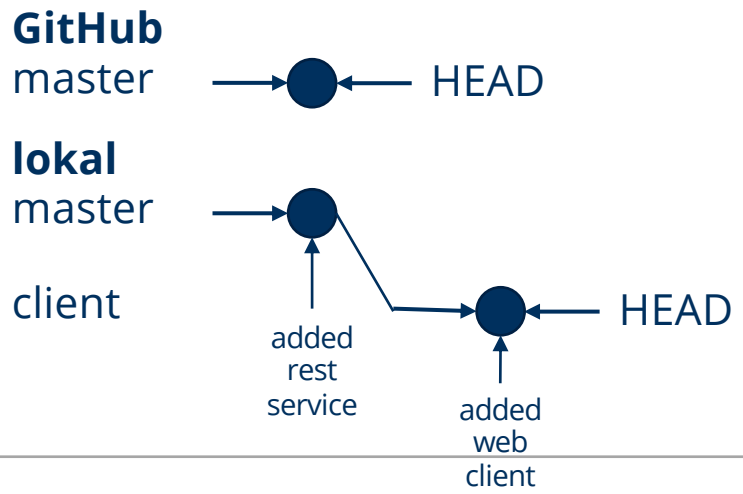
```
git checkout client
```

3. Add client files

```
git add src/web-client
```

4. commit

```
git commit –m “added web client”
```

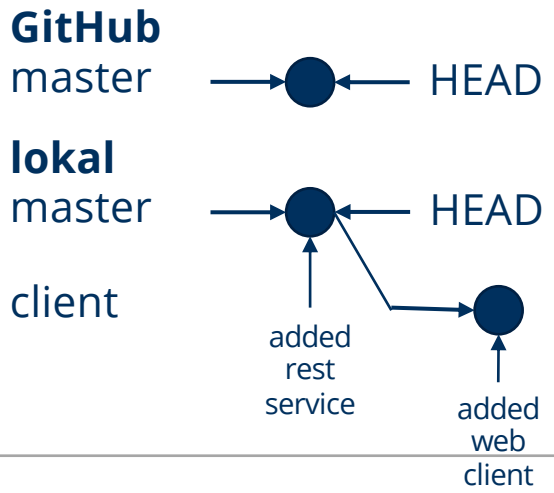


Wie funktioniert Git

Client Branch mit Master Branch mergen

1. Switch back to master

```
git checkout master
```



Wie funktioniert Git

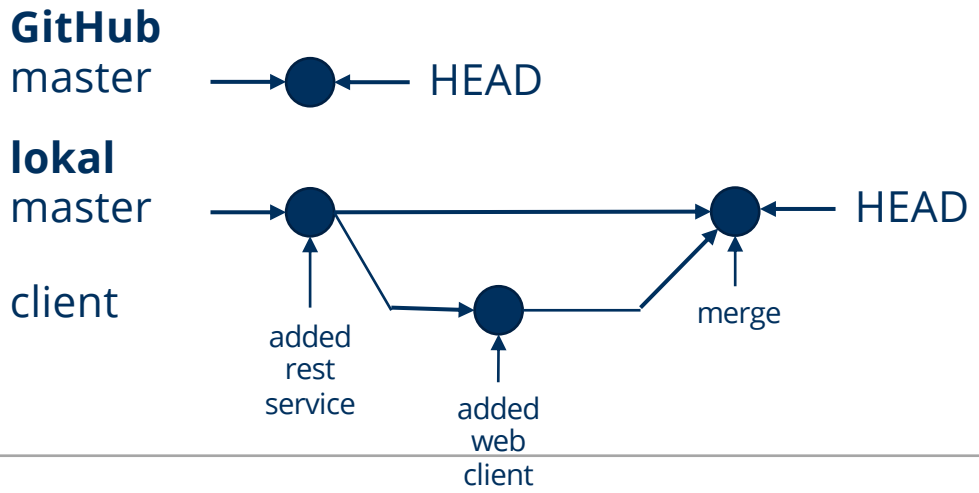
Client Branch mit Master Branch mergen

1. Switch back to master

```
git checkout master
```

2. Merge Client into Master

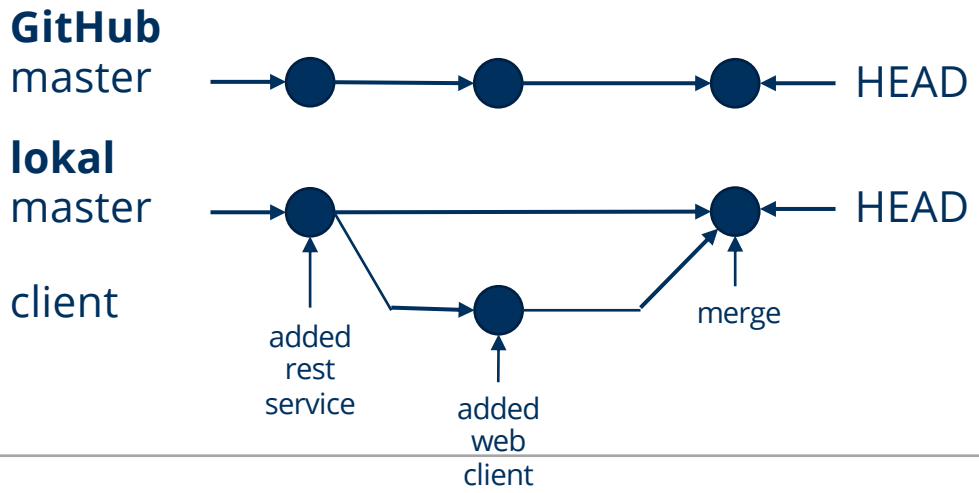
```
git merge client
```



Wie funktioniert Git

Client Branch mit Master Branch mergen

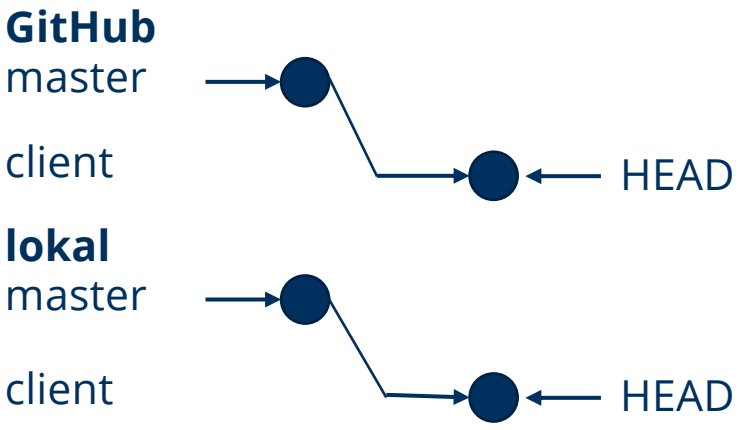
1. Switch back to master — `git checkout master`
2. Merge Client into Master — `git merge client`
3. Push changes to GitHub — `git push origin master`



Wie funktioniert Git

Client Branch mit Master Branch mergen - **Alternative**

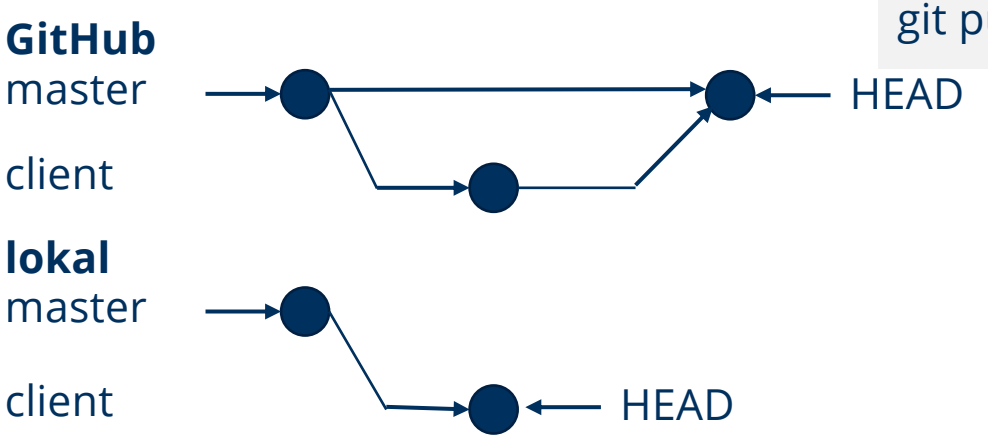
1. Create new branch	git branch slides
2. Switch to new branch	git checkout slides
3. Add slides	git add Im\ Team\ Software\ entwickeln.pdf
4. Commit slides	git commit -m “added slides”
5. Push client Branch to GitHub	git push origin slides



Wie funktioniert Git

Client Branch mit Master Branch mergen - **Alternative**

1. Create new branch
```
git branch slides
```
2. Switch to new branch
```
git checkout slides
```
3. Add slides
```
git add Im\ Team\ Software\ entwickeln.pdf
```
4. Commit slides
```
git commit -m “added slides”
```
5. Push client Branch to GitHub
```
git push origin slides
```



6. Pull Request öffnen und mergen

Wie funktioniert Git

Client Branch mit Master Branch mergen - **Alternative**

1. Create new branch

```
git branch slides
```

2. Switch to new branch

```
git checkout slides
```

3. Add slides

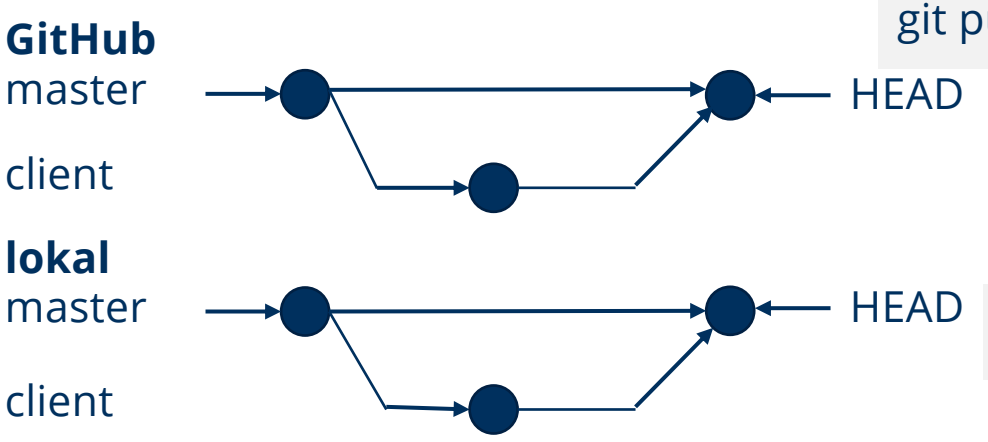
```
git add Im\ Team\ Software\ entwickeln.pdf
```

4. Commit slides

```
git commit -m “added slides”
```

5. Push client Branch to GitHub

```
git push origin slides
```



6. Pull Request öffnen und mergen
7. Pull merge to local

```
git pull origin master
```

Commandline vs GUI

- Commandline unterstützt 100% aller Funktionen
- Teilweise recht umständlich, kompliziert

- GUIs unterstützen nicht immer alle Funktionen

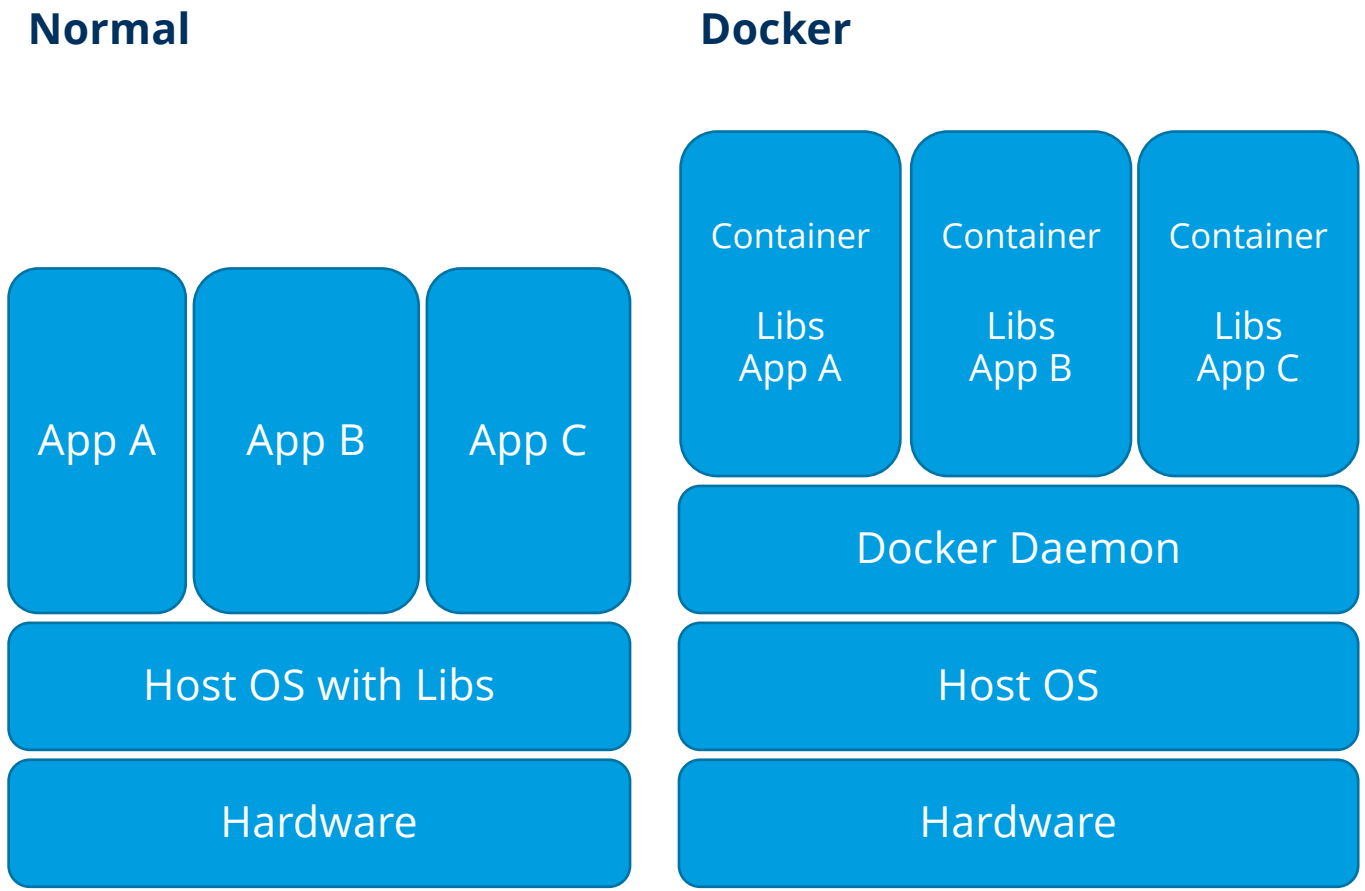
- Je nach Situation is commandline bzw. eine GUI schneller

- Bsp. für GUIs:
  - GitHub Desktop
  - Sourcetree
  - GitKraken

3. Docker

3.1 Was ist Docker

- Standardisiertes Environment für jede Applikation
- Container können überall schnell gestartet werden
- Vermeidet Konflikte zwischen Apps



3.2 Komponenten von Docker

Image

- Kapsel, welche Libraries und eine App enthält
- Wird gebaut und abgespeichert

Container

- Ein Image, welches mit bestimmten Parametern vom Docker Daemon ausgeführt wird

Dockerfile

- “Rezept” für ein Image
- Meist wird bereits vorhandes Image erweitert

Download Docker

Windows: https://download.docker.com/win/stable/Docker%20for%20Windows%20Installer.exe

Mac: https://download.docker.com/mac/stable/Docker.dmg

Linux:

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

```
sudo sh get-docker.sh
```

Dockerzyklus

Dockerfile

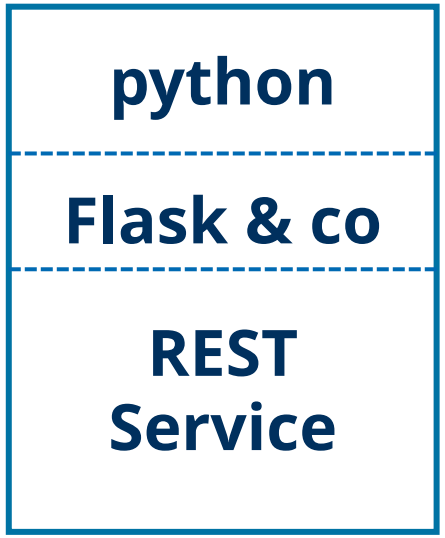
```
FROM python:latest

ADD app.py .
WORKDIR .

RUN pip3 install flask

CMD ["python3", "app.py"]
```

Image



Eigenschaften:

Name: rest-service

Container

Eigenschaften:

Name: rest-service
Image: rest-service
Port: 80:80
Detached: True

Dockerfile

```
FROM python3:latest

ADD app.py .
WORKDIR .

RUN pip install flask
EXPOSE 2000

CMD [“python3”, “app.py”]
```

← Spezifiziert das Basisimage mit den python3 libs

← Fügt die Datei mit dem Rest-Service zum Image hinzu

← Setzt den aktuellen Ordner als *working directory*

← Installiert das python flask modul

← Schaltet den Port 2000 im Docker Netzwerk frei

← Sagt Docker was beim start des Image als Container gemacht werden soll. Also *python3* mit der Datei *app.py ausführen*

```
FROM <image>:<tag>
ADD <Quelldatei> <Zielpfad>
WORKDIR <Pfad>
RUN <shell kommando>
CMD [“<programm>”, ”<datei>”]
```

Vom Dockerfile zum laufenden Container

- Im Ordner wo das Dockerfile liegt:

```
docker build –t  <image name> .
```

```
docker build –t rest-service .
```

- Alle lokalen Images anzeigen:

```
docker images
```

- Image zum upload taggen:

```
docker tag <source image>:<tag> <user>/<target image>:<tag>
```

```
docker tag rest-service leonhess/rest-service:latest
```

- Image zu DockerHub pushen:

```
docker push <user>/<target image name>:<tag>
```

```
docker push leonhess/rest-service:latest
```

Vom Dockerfile zum laufenden Container

- Container mit image starten:

```
docker run <options> <image>:<tag>
```

```
docker run –d –p 2000:2000 –-name rest-service rest-service
```

- Laufende Container anzeigen:

```
docker ps
```

-p <host port>:<container port>	Leitet traffic vom host port and entsprechenden container port weiter
-d	Startet den Container im Hintergrund
--name <container name>	Gibt dem Container einen Namen
--restart <no, always, on-failure, unless-stopped>	Legt fest wann ein container neugestartet werden soll

Client und Service verbinden

- Service container stoppen:

```
docker kill rest-service
```

```
docker rm rest-service
```

- Client container bauen:

```
docker build -t rest-client .
```

- Netzwerk anlegen:

```
docker network create --driver bridge rest-net
```

- Service und Client mit Netzwerk starten:

```
docker run -d --network rest-net --name rest-service rest-service
```

```
docker run -d --network rest-net -p 30000:30000 --name rest-client rest-client
```

Fragen?

Jetzt hier!

oder

leon.hess@mailbox.tu-dresden.de

Folien: github.com/leonhess/rest-uebung

Quellen

Folie 4:

Docker logo: https://www.docker.com/sites/default/files/d8/2019-07/vertical-logo-monochromatic.png

Git logo: https://git-scm.com/images/logos/downloads/Git-Logo-1788C.png

Person: https://toastmasters-reutlingen.de/wp-content/uploads/2018/12/person-622x800.png