SPRING & DOCKER WORKSHOP

What is Spring and how to use Docker?

Created by Daniel Pollack

ZU MIR

- ~1.5y bei ITS
- vorher Student
- Projekt FB
- komme aus Gotha Thüringen
- studiert in Weimar (Medien-Informatik)

DISCLAIMER

- keine Zertifikate
- kein XML
- learning by doing Ansatz
- keine Kaufentscheidungen wegen mir
- Spring/Docker entwickelt sich zu schnell um aktuell zu sein

SPRING

Was ist Spring?

- Application Framework
- bringt IoC (Inversion of Control)
- bringt Dependency Injection

Was macht Spring beim Start?

- erstellen aller notwendigen Beans
- Autowiring/Injecting der Dependencies
- was immer der Entwickler will

Bean Lifecycle

1. Framework factory lädt Bean Definition u. erstellt sie
2. Befüllen der Properties
3. Erfüllen von Abhängigkeiten

Beans erstellen

```
@Component
public class Foo {}
```

```
@Configuration
public class SomeConfigurationClass{
  @Bean
  public SomeClass produceThisClass(){return new SomeClass();}
}
```

Alternativen zu *@Component*

- `@Service`
- `@Repository`
- `@Controller`
- `@RestController`
- etc.

Bean Scopes

- Singleton (***default***)
- Prototype
- Request
- Session
- GlobalSession

Abhängigkeiten bestimmen

```
@Autowired
private ISomeClass someClass
```

```
public class SomeOtherClass{

  private ISomeClass someClass;

  @Autowired
  SomeOtherClass(ISomeClass someClass){
    this.someClass=someClass;
  }
}
```

Autowire Varianten

1. byName
   - Field Name ~= Bean Class Name
2. byType
   - Field Type == Bean Type
3. byConstructor
   - byName | byType als Constructor Param

Property Injection

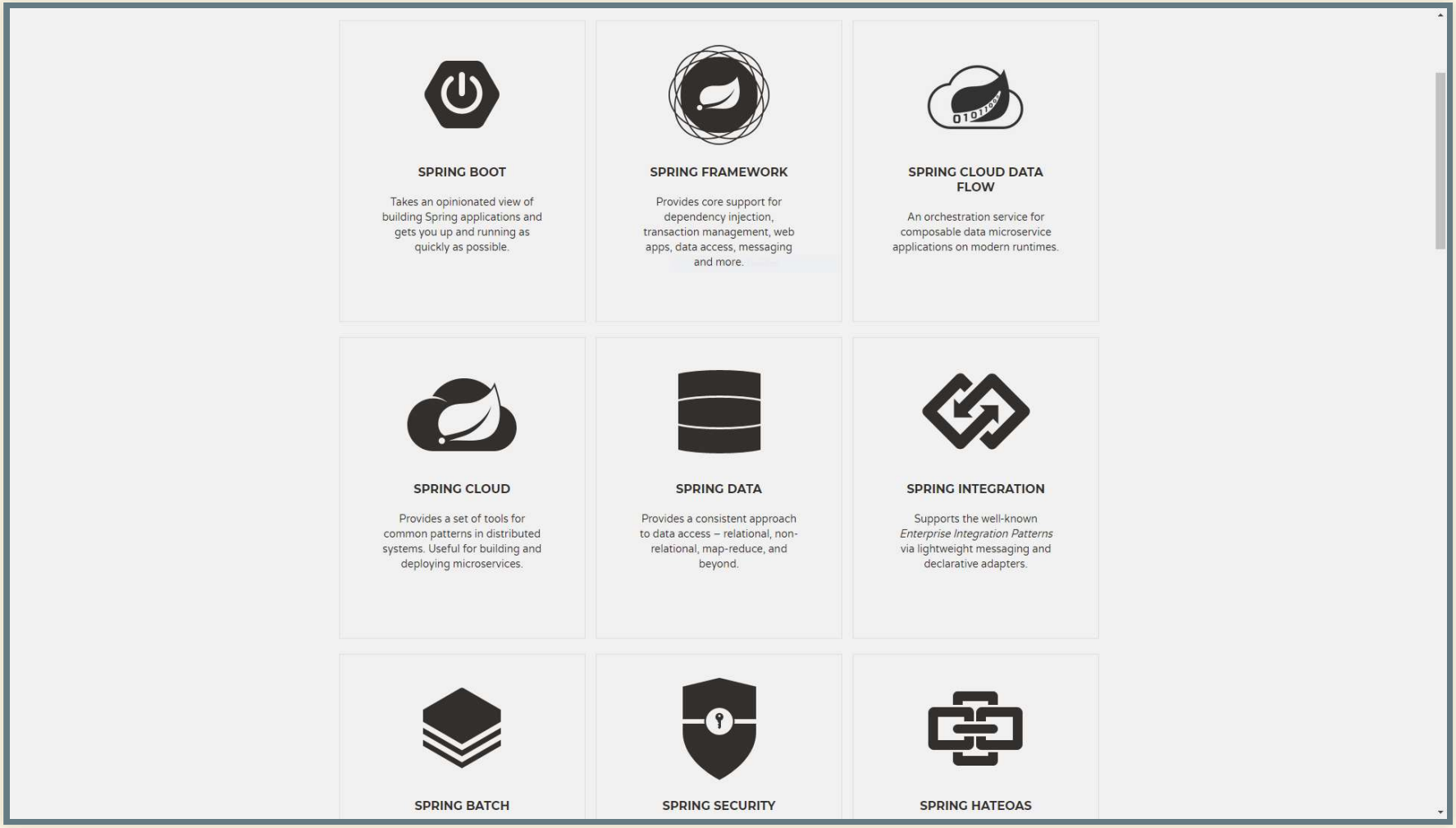
```
@Value("${some.path.value}")
public Long solutionToSomething;
```

```
# src/java/resources/application.yml
some:
  path:
    value: 42.0
```

```
# src/java/resources/application.properties
some.path.value=42.0
```

Spring Ökosystem

- Spring ist modular
- alle Projekte sollten miteinander funktionieren



SPRING-BOOT

Was ist Spring-Boot?

*"Takes an **opinionated** view of building production-ready Spring applications. Spring Boot favors **convention over configuration** and is designed to get you up and running as quickly as possible."*

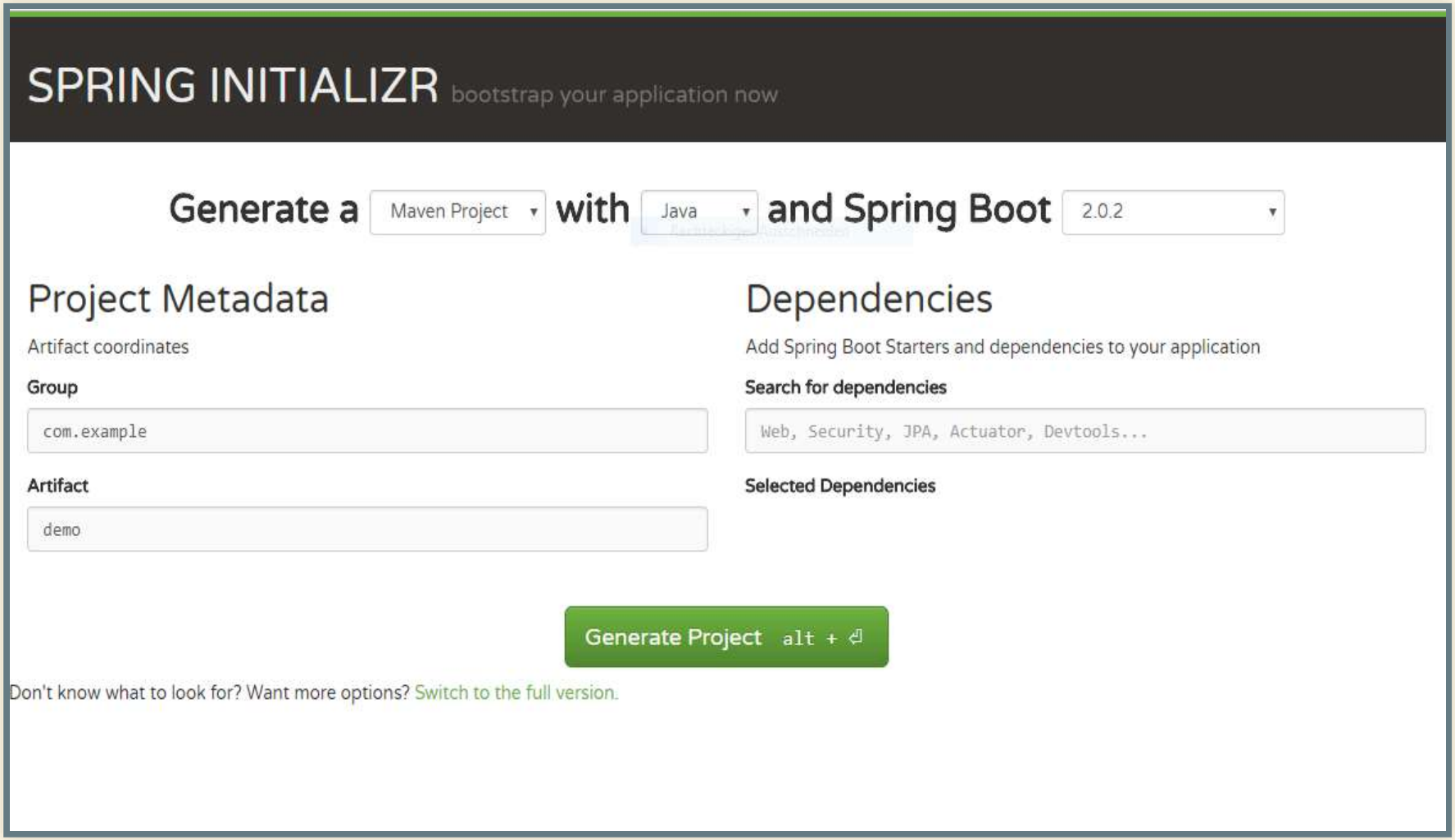
https://projects.spring.io/spring-boot/

Features

- stand-alone Spring applications
- embedded Tomcat (keine WAR files nötig)
- stellt **opinionated 'starter' POMs** um Maven Konfiguration zu vereinfachen
- konfiguriert Spring automatisch, wann immer möglich
- stellt "production ready" Features wie Metrics, Health Checks und externe Konfiguration
- keine XML Konfiguration oder Code-Generierung notwendig

Wie baue ich eine Spring-Boot-Anwendung von Null auf?

- https://start.spring.io



TASK

Erstellt eine Spring-Boot Anwendung

- Maven
- Java8
- Dependencies:
  - Web

TASK

- erstellt
  - einen `@RestController`
  - eine Methode die mit HTTP Get aufgerufen wird (`@GetMapping`)
    - return `"Hello World"`

Zusatz:

- Controller bezieht Grußformel aus der application property (`@Value`)

TESTING SPRING-BOOT

- Unit-Test
- Integration Test

Unit-Test

```
@Test
public void stuff(){assertTrue(true);}
```

Integration Test

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class TestClass{
  @Test
  public void someTest(){
    ...
  }
}
```

- **Mock** - eine fake Implementation einer Klasse
- **Spy** - zum Teil ein Mock, erlaubt das überladen von Methoden der Klasse

Run Tests

- Unit Tests laufen mit Surefire Plugin (enabled by default)
  - führt alle Tests in Files mit Test Sufix aus (`*Test.java`)
- Integration Tests laufen mit Failsafe Plugin (missing by default)
  - führt alle Tests in Files mit Sufix IT oder IntegerationTest aus (`*IT.java` | `*IntegrationTest.java`)

Failsafe Plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <version>2.20</version>
  <configuration>
    <includes>
      <include>**/*IT.java</include>
    </includes>
    <additionalClasspathElements>
      <additionalClasspathElement>${basedir}/target/classes</addi
    </additionalClasspathElements>
    <parallel>none</parallel>
  </configuration>
  <executions>
    <execution>
      <goals>
```

TASK

- Testet den Controller

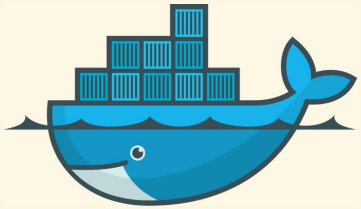
```
        <dependency>
            <groupId>com.jayway.restassured</groupId>
            <artifactId>rest-assured</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>
```

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RA
@RunWith(SpringRunner.class)
public class HelloControllerIT {

    @LocalServerPort
    public int port;

    @Before
    public void init() {
        RestAssured.port = port;
    }

    @Test
    public void testHelloControllerResponse() {
        ...
    }
```

Nützliche Annotationen für Tests

- `@Test(expected=RuntimeExceptio.class)`
- `@RunWith(SpringRunner.class)`
- `@TestConfiguration`
- `@MockBean`
- `@SpyBean`
- `@ActiveProfiles`
- `@ContextConfiguration`
- `@DirtiesContext`
- `@Transactional`

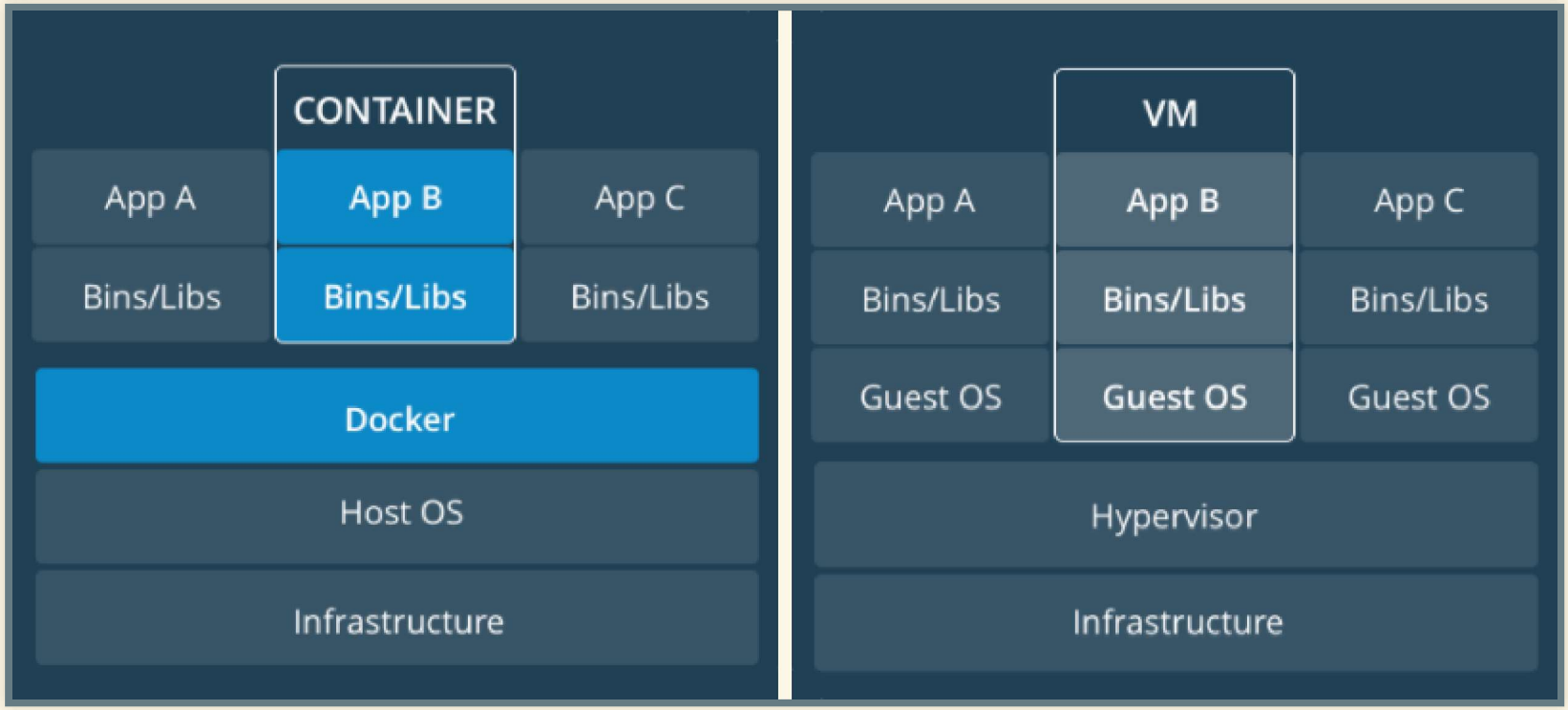
DOCKER



Was ist Docker?


WORKED FINE IN DEV
OPS PROBLEM NOW
memegenerator.net

Container vs. VM



TASK

```
$ docker run hello-world

$ docker run -p 8080:80 nginx
```

Begriffe

IMAGE & LAYER

> *"A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only."*

https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/

Begriffe

REGISTRY

> *"A registry is a storage and content delivery system, holding named Docker images, available in different tagged versions."*

https://docs.docker.com/registry/introduction/

Begriffe

CONTAINER

> *"The major difference between a container and an image is the top writable layer."*

https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/#container-and-layers

Begriffe

DOCKER COMPOSE:

- Tool zum Definieren und Ausführen von Multi-Container Definitionen

DOCKER MACHINE:

- Docker Host auf mehren Plattformen starten

DOCKER HUB:

- Dockers registry für Images

Was macht Docker beim Start?

- Resourcen reservieren
- Host Kernel ankoppeln
- Befehl ausführen

Wie setze ich von Null an eine Dockerumgebung auf?

- Installiere Docker auf einem Computer
- Besorge eine Registry

Wie verpacke ich meine Anwendung in Docker?

- In einem Dockerfile

Dockerfile

```
FROM ibmjava:8-sfj-alpine

CMD ["echo","hello world"]
```

```
FROM ibmjava:8-sfj-alpine

ADD target/demo.jar demo.jar

EXPOSE 8080

CMD ["java","-jar","demo.jar"]
```

Dockerfile build

- `$docker build .`
- `$docker build . -t greeter`
- `$docker build . -t greeter:stable`

Docker Kommandos

- docker run
- docker build
- docker logs
- docker exec
- docker stop
- docker kill

Docker Kommandos

- docker rm
- docker ps
- docker top

TASK

- erstellt ein `Dockerfile` und verpackt eure App

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
      <finalName>${project.name}</finalName>
      <goal>packaging</goal>
      <mainClass>de.workshop.demo.DemoApplication</mainC
  </configuration>
</plugin>
```

Docker best practices?

- Dockerfiles versionieren
- .dockerignore File nutzen
- nur einen Prozess/Befehl pro Container starten
- zustandslos halten
- schmale BaseImages nutzen

Java in Docker

PROBLEM:

JVM weiß nichts von Containern und sieht den Docker Host und seine Resourcen

LÖSUNG:

- JVM Parameter
- Seit Java8 optional container aware

-->

DOCKER-COMPOSE

Problem

```
docker image pull redis:alpine
docker image pull russmckendrick/moby-counter
docker network create moby-counter
docker container run -d --name redis --network moby-counter redis
docker container run -d --name moby-counter --network moby-counte
  -p 8080:80 russmckendrick/moby-counter
```

LÖSUNG

```yaml
version: "3"
services:
  redis:
    image: redis:alpine
    volumes:
    - redis_data:/data
    restart: always
  mobycounter:
    depends_on:
      - redis
    image: russmckendrick/moby-counter
    ports:
      - "8080:80"
    restart: always
    volumes:
      redis_data:
```

```
#build all images defined within compose file
docker-compose build
docker-compose build --no-cache

#run all images defined within compose file
docker-compose up
docker-compose up -d

#stop all running images
docker-compose down

#pull all images from remote registry
docker-compose pull

#push all images to remote registry
docker-compose push
```

TASK

- erstellt ein compose file

SPRING SECURITY

Authorization

Darf der Nutzer das?

Authentication

Hier bin ich und ich darf das.

Spring-Security

- bietet default Einstellungen
  - Login Page
  - User Roles

TASK

- bindet Spring-Security in eure App ein
- legt einen weiteren Nutzer und meldet euch an

TRÆFIK

WAS IST TRÆFIK?

- HTTP reverse Proxy
- Loadbalancer
- unterstützt Docker

TASK

- besorgt Træfik: `docker pull traefik`
  - tag ist egal
- erweitert euer docker-compose file