# Microservices

2016

F 340 E R01 2015-02

ALTRAN

# 1. Vorstellung

Vorname, Name : Stefan Draeger
Titel : Consultant
Hobby‘s : App Entwicklung, Arduino, Garten

seit 01.05.2015 bei Altran S.A.S Co. KG

# 2. Warum Microservices?

## 2.1 Monolithische Software-Architektur

Die Monolithische Software-Architektur findet sich in großen Gewerken wieder welche die funktionalen Elemente bildet.
Dieses bedeutet dass, die Elemente

- untrennbar,
- homogen

sind.

# 2. Warum Microservices?

## 2.2 Microservices

Microservices
- trennen die Architektur,
- ermöglichen einen einfachen Technologiewechsel,
- macht es einfacher die Vorteile von verschiedenen Programmiersprachen zu nutzen,
- werden in kleinen Teams mit spezialisierten Kenntnissen entwickelt.

Der Ausfall eines Services bedeutet nicht gleich ein Ausfall der gesamten Applikation, des Weiteren können die verschiedenen Services zu unterschiedlichen Zeiten deployt und auf verschiedenen Servern gehostet werden.

# 3. Was sind Microservices?

Microservices
- sind kleine in sich abgeschlossene Funktionalitäten,
- können einzeln
  - getestet
  - deployt, und
  - scaliert

  werden
- aufrufe geschehen über HTTP REST (JAX RS) Calls

# 4. Ein Beispiel

Als Beispiel dient hier ein einfacher Taschenrechner welcher die Funktionen

- Addition,
- Subtraktion,
- Division,
- Multiplikation

beherrscht.

## 4.1 Framework

Es werden diverse Werkzeuge benötigt um ein Microservice zu erstellen, unteranderem das JAX-RS Framework, einen Application Server usw.

Das kann man alles selbst zusammenstellen oder man verwendet DropWizard.

# 4. Ein Beispiel

## 4.2 DropWizard

DropWizard enthält :
- Jetty, a high-performance HTTP server.
- Jersey, a full-featured RESTful web framework.
- Jackson, the best JSON library for the JVM.
- Metrics, an excellent library for application metrics.
- Guava, Google's excellent utility library.
- Logback, the successor to Log4j
- Hibernate Validator,

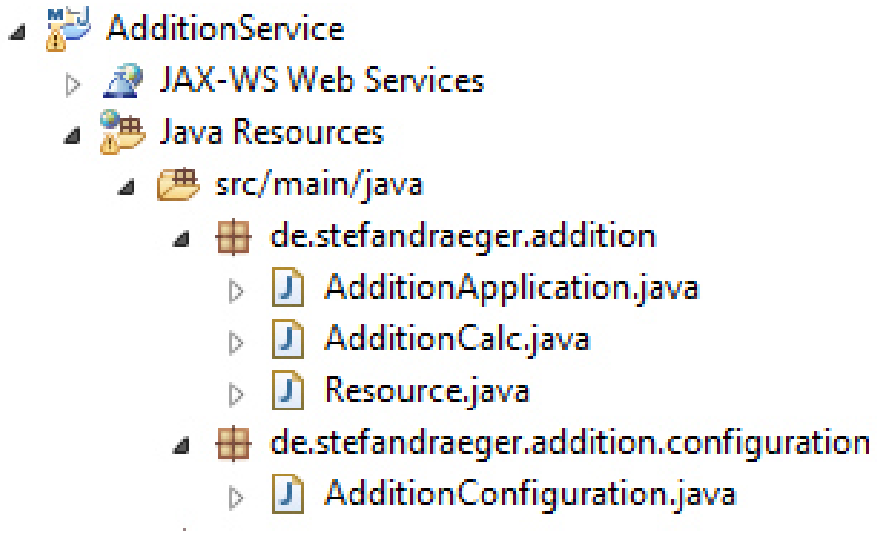
Diese „Features" werden mit einer Dependency in die POM geladen.

```
<dependency>
    <groupId>io.dropwizard</groupId>
    <artifactId>dropwizard-core</artifactId>
    <version>0.9.2</version>
</dependency>
```

# 5 DropWizard

## 5.1 Erstellen eines Microservices mit DropWizard

Für ein Microservice mit dem Framework DropWizard werden folgende Klassen benötigt:



# 5 DropWizard

## 5.1 Erstellen eines Microservices mit DropWizard

Des Weiteren wird eine Konfigurationsdatei benötigt in welcher zbsp. Parameter für den Service vorbelegt werden können wenn diese nicht über die URL mitgeliefert werden.

Es kann auch der Server über die Datei konfiguriert werden.
In meinem Beispiel habe ich für jeden Microservice einen eigenen HTTP Port eingerichtet.

# 5 DropWizard

## 5.2 Konfiguration

Konfiguration des Servers erfolgt per YAML Datei.
!! Keine Tabulator (Tabs) in der Datei verwenden !!

config.yml

```
zahl1: 0
zahl2: 0

server:
  applicationConnectors:
    - type: http
      port: 8080
  adminConnectors:
    - type: http
      port: 8081
```

# 5 DropWizard

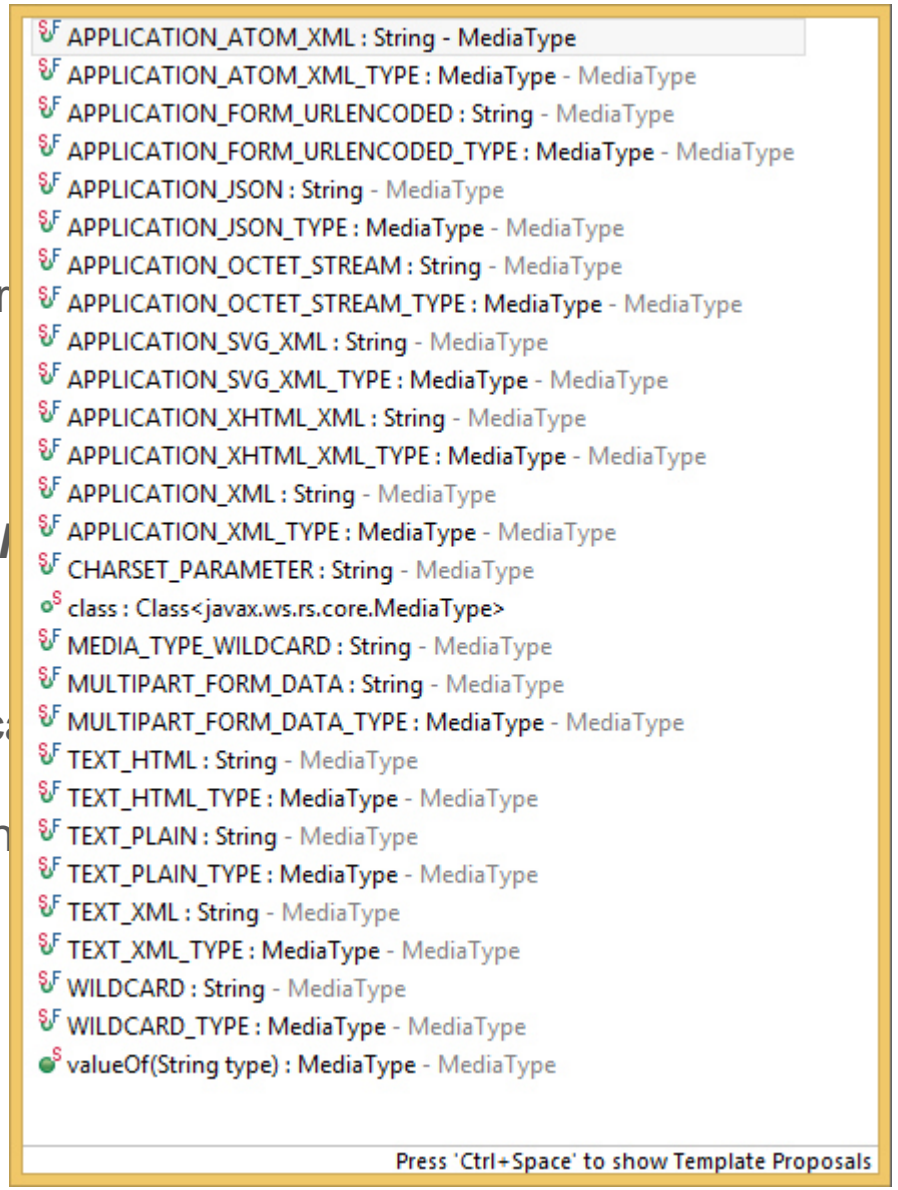
## 5.3 Resource Klasse

Die Klasse Resource definiert das „Ser
angesprochen werden kann.

```
@Path("/addition")
@Produces(MediaType.APPLI
public class Resource
```

@Path -> Das „Servlet“ zbsp. http://loc

@Produces -> Das Format des Ergebn



Erklären des Beispiels „Taschenrechner“!

# 5 DropWizard

## 5.4 Cross-Domain Origin

Die „CORS“ Richtlinie verbietet zugriffe auf andere Domains somit kann nicht von http://www.foo.com auf http://www.bar.com zugegriffen werden. Um dieses zu erlauben muss der Header im Server und im Client dieses explizit erlauben.

Wenn man mit JavaScript und jQuery dieses machen möchte so geht dieses mit folgendem Ajax Beispiel:

```
$.ajax({
        type: 'GET',  url: address,
        crossDomain: true, dataType: 'json',
        })
        .done(function( msg ) {
            //positiv
        })
        .fail(function( msg ) {
            alert(msg.statusText );
        });
```

# 5 DropWizard

## 5.4 CORS Filter für Jetty

Die Microservices auf dem DropWizard Framework laufen im Jetty WebContainer, hier nun das Beispiel wie CORS für die Funktionen „GET“ , „PUT“ , „POST“ usw. erlaubt werden.

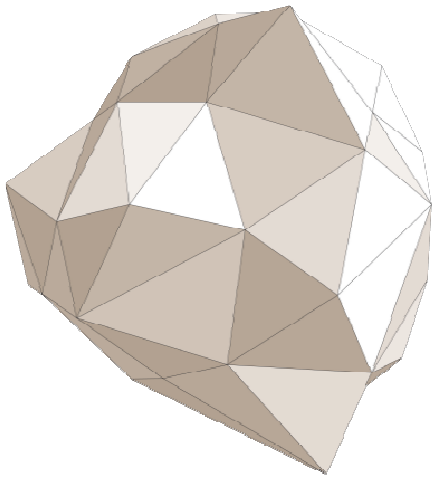
```
private void addCors(Environment environment) {
final FilterRegistration.Dynamic cors = environment.servlets().addFilter("CORS",
(Class<? extends Filter>) CrossOriginFilter.class);

cors.setInitParameter("allowedOrigins", "*");
cors.setInitParameter("allowedHeaders", "X-Requested-With,Content-
Type,Accept,Origin");
cors.setInitParameter("allowedMethods",
"OPTIONS,GET,PUT,POST,DELETE,HEAD");

// Add URL mapping
cors.addMappingForUrlPatterns(EnumSet.allOf(DispatcherType.class), true, "/*");
}
```

# 6. Quellen

http://www.jug-ostfalen.de/assets/articles/2016/microservice-technologie-enabler.pdf

http://www.dropwizard.io/

# INNOVATION MAKERS







ALTRAN