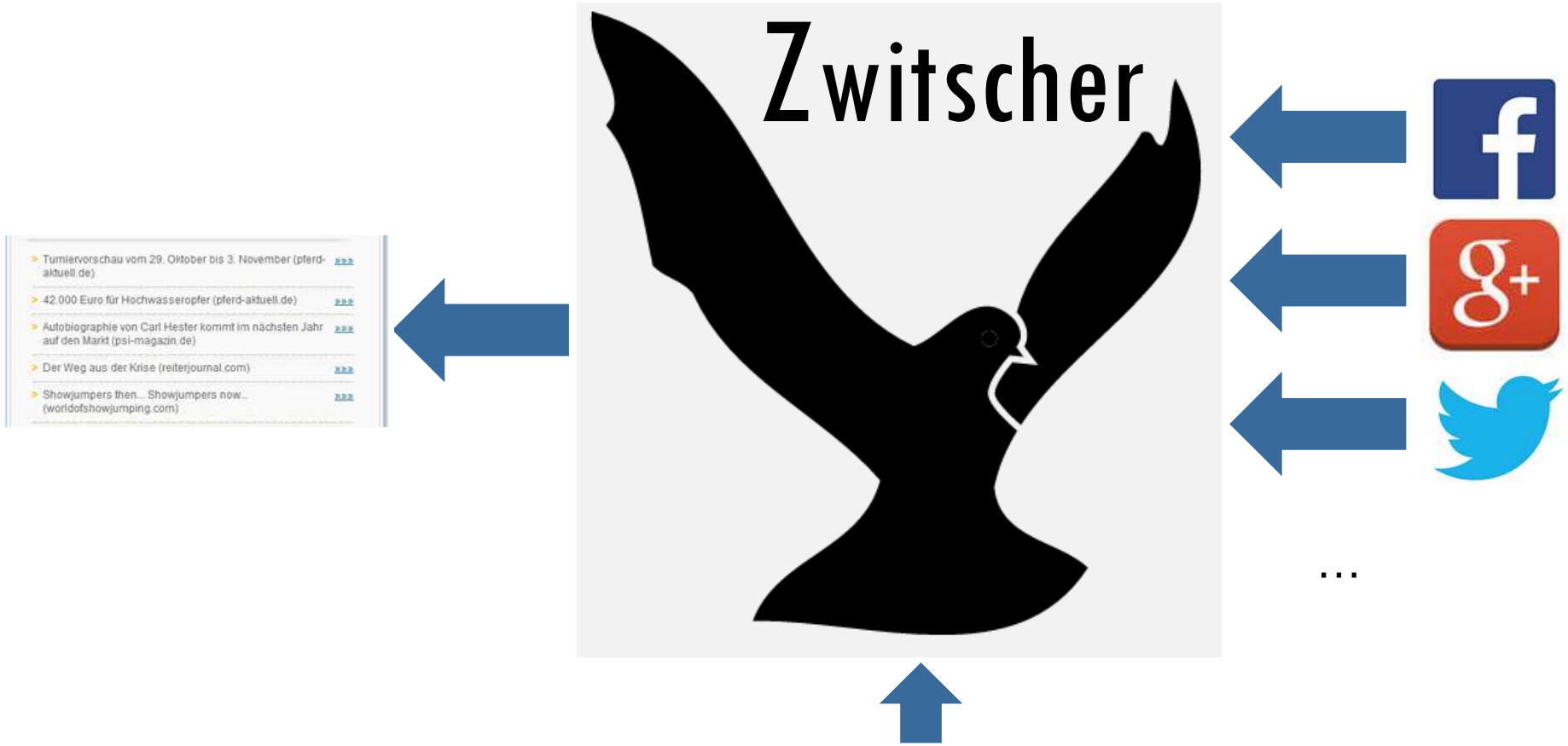
Cloud Computing: Übung

Kapitel 2: Kommunikationssysteme im Internet

Dr. Josef Adersberger

Unsere Cloud-Anwendung:



REST API Entwicklungsvorgehen

REST ist ein Paradigma für Anwendungsservices auf Basis des HTTP-Protokolls.

- REST ist eine Paradigma für den Schnittstellenentwurf von Internetanwendungen auf Basis des HTTP-Protokolls.
- REST wurde erstmalig in der Dissertation von Roy Fielding definiert: „Architectural Styles and the Design of Network-based Software Architectures", 2000, University of California, Irvine.
- **Grundlegende Eigenschaften:**
  - **Alles ist eine Ressource:** Eine Ressource ist eindeutig adressierbar über einen URI, hat eine oder mehrere Repräsentationen (XML, JSON, bel. MIME-Typ) und kann per Hyperlink auf andere Ressourcen verweisen. Ressourcen sind, wo immer möglich, hierarchisch navigierbar.
  - **Uniforme Schnittstellen:** Services auf Basis der HTTP-Methoden (POST = erzeugen, PUT = aktualisieren oder erzeugen, DELETE = löschen, GET = abfragen). Fehler werden über die HTTP Codes zurückgemeldet. Services haben somit eine standardisierte Semantik und eine stabile Syntax.
  - **Zustandslosigkeit**: Die Kommunikation zwischen Server und Client ist zustandslos. Ein Zustand wird im Client nur durch URIs gehalten.
  - **Konnektivität**: Basiert auf ausgereifter und allgegenwärtiger Infrastruktur: Der Web-Infrastruktur mit wirkungsvollen Caching- und Sicherheitsmechanismen, leistungsfähigen Servern und z.B. Web-Browser als Clients.

REST-Webservices mit JAX-RS.

http://www.service.de/hello/Josef?salutation=Servus

@GET, @POST, @PUT, @DELETE

Analog @Consumes für 1. Parameter

Analog @FormParam bei POST Requests

```
@Path("/hello/{name}")
public class HelloWorldResource {

    @GET
    @Produces("application/json")
    public ResponseMessage getMessage(
            @DefaultValue("Hallo") @QueryParam("salutation") String salutation,
            @PathParam("name") String name) throws IOException {
        ResponseMessage response = new ResponseMessage(new Date().toString(), salutation + " " + name);
        return response;
    }

}
```

Der Prozess im Überblick

Anwendungsfälle erheben

Entitätenmodell erstellen

REST Schnittstelle umsetzen

Var. 1

Top-down Ansatz:

REST Schnittstelle definieren

REST Schnittstelle generieren

REST Schnittstelle implementieren

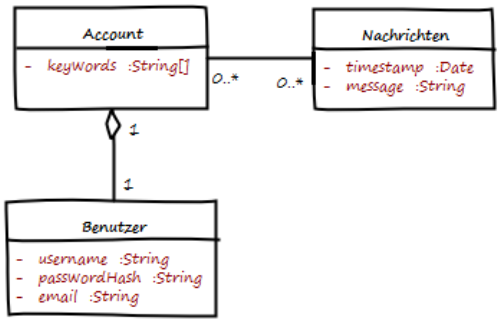
Var. 2

Bottom-up Ansatz:

REST Schnittstelle implementieren

Definition REST-Schnittstelle generieren

- Benutzer authentifizieren
- Nachricht einstellen
- Meine Nachricht löschen
- Meine Nachricht ändern
- Liste aller Nachrichten anzeigen
- Liste meiner Nachrichten anzeigen
- Liste der Nachrichten mit best. Text anzeigen
- Nutzerstatistik anzeigen
- Account erstellen (inkl. Benutzerinfos anlegen)
- Account ändern
- Account löschen



Beispiele für REST-Aufrufsyntax: Schnittstellenentwurf über Substantive.

- Produkte aus der Kategorie Spielwaren: http://www.service.de/produkte/spielwaren
- Bestellungen aus dem Jahr 2008 http://www.service.de/bestellungen/2008
- Liste aller Regionen, in denen der Umsatz größer als 5 Mio. Euro war http://www.service.de/regionen/umsatz/summe?groesserAls=5M
- Gib mir die zweite Seite aus dem Produktkatalog http://www.service.de/produkte/2
- Alle Gruppen, in den der Benutzer „josef.adersberger“ Mitglied ist. http://www.service.de/benutzer/josef.adersberger/gruppen

Gängige Entwurfsregeln:

- Plural, wenn auf Menge an Entitäten referenziert werden soll. Sonst singular.
- Pfad-Parameter, wenn Reihenfolge der Angabe wichtig. Sonst Query Parameter.
- Standard Query Parameter einführen (z.B. für Filter und Abfragen sowie seitenweisen Zugriff) und konsistent halten.
- Pfad-Abstieg, wenn Entitäten per Aggregation oder Komposition verbunden sind.
- Pfad-Abstieg, wenn es sich um einen gängigen Navigationsweg handelt.
- Ids als Pfad-Paramter abbilden.
- Fehler und Ausnahmen über Return Codes abbilden. Einen Standard-Code suchen, der von der Semantik her passt.

Siehe auch: http://codeplanet.io/principles-good-restful-api-design

Technische Basis: Dropwizard

Dropwizard Production-ready, out of the box.

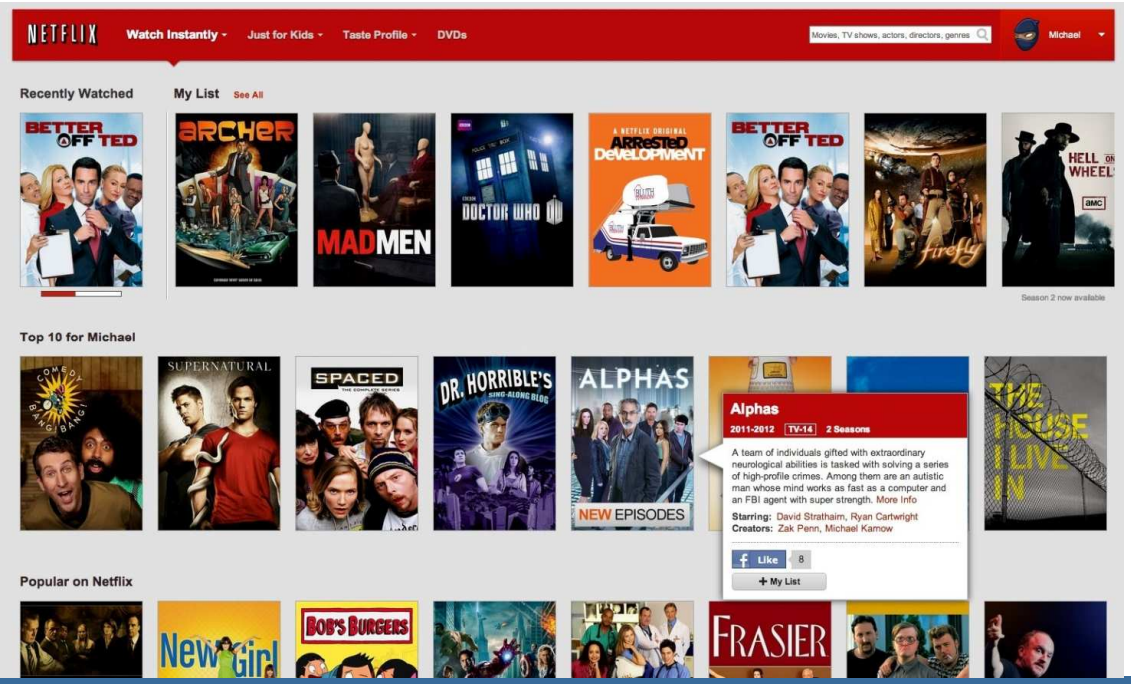
Einführung: Dropwizard, ein Micro Service Container

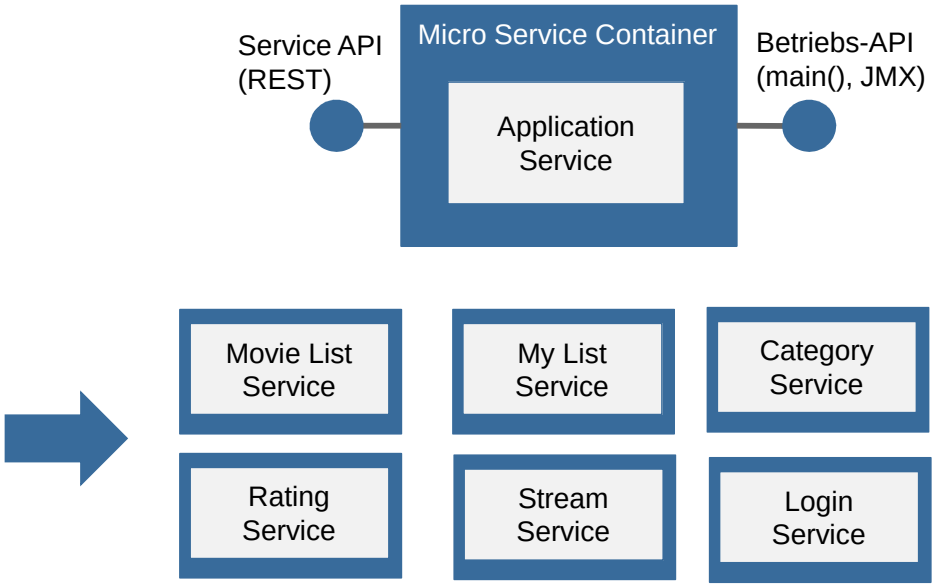
Dropwizard is a Java framework for developing ops-friendly, high-performance, RESTful web services.

Dropwizard pulls together **stable**, **mature** libraries from the Java ecosystem into a **simple**, **light-weight** package that lets you focus on *getting things done*.

Dropwizard has *out-of-the-box* support for sophisticated **configuration**, **application metrics**, **logging**, **operational tools**, and much more, allowing you and your team to ship a *production-quality* web service in the shortest time possible.

- Grundprinzip: *Micro Service Architektur* = Komponentenorientierung im Betrieb





Technische Basis: Swagger

Einführung: Swagger, eine Schnittstellen-Beschreibungs-Sprache

A POWERFUL INTERFACE TO YOUR API

JSON Struktur

Swagger is a simple yet powerful representation of your RESTful API. With the largest ecosystem of API tooling on the planet, thousands of developers are supporting Swagger in almost every modern programming language and deployment environment. With a Swagger-enabled API, you get interactive documentation, client SDK generation and discoverability.

SWAGGER TOOLS & RESOURCES

TOOLS

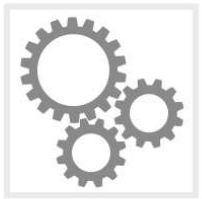
SWAGGER UI
Use a Swagger specification to drive your API documentation. **Demo** and **Download**.

SWAGGER EDITOR
An editor for designing Swagger specifications from scratch, using a simple YAML structure. **Demo** and **Source**.

SDK GENERATORS
Turn an API spec into client SDKs or server-side code with **Swagger Codegen**.

RESOURCES

SERVER INTEGRATIONS
Dozens of integration options for putting Swagger in your API, from both Reverb and the community. Pick your language, framework and **get started!**.



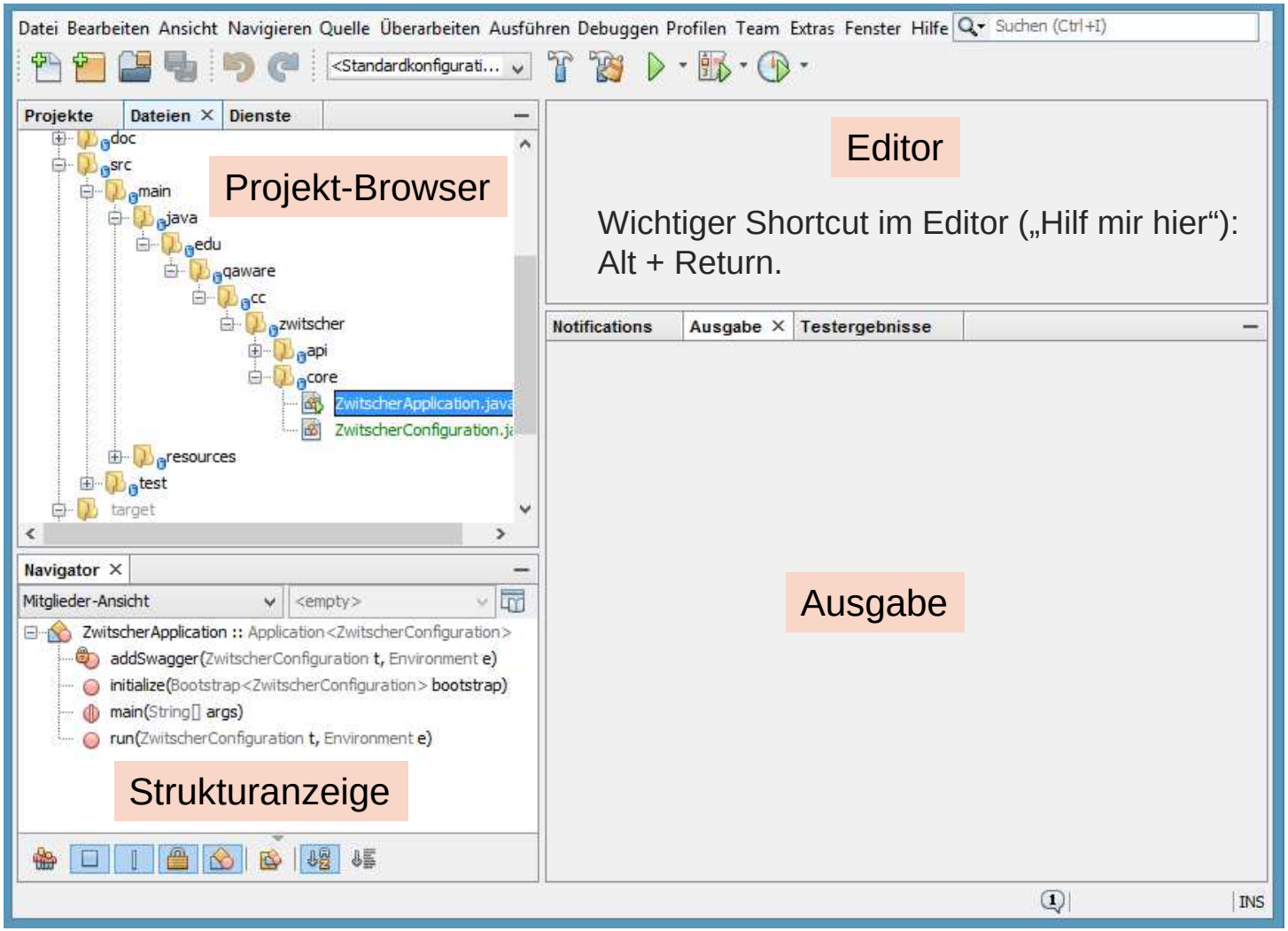
SERVICES
From API management to Platform as a Service, there are a number of services tightly integrated with Swagger. **See more**.

Alternativen:

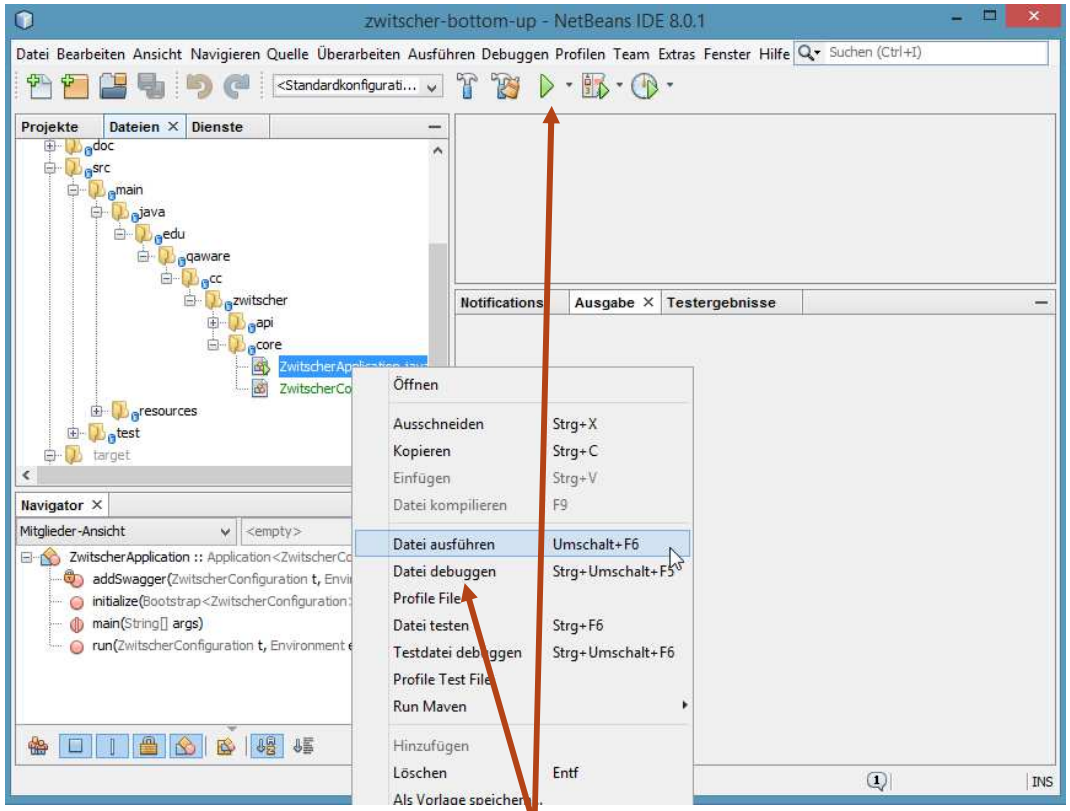
- WADL
- RAML
- Blueprint API
- WSDL

Entwicklungswerkzeuge: Netbeans und Maven

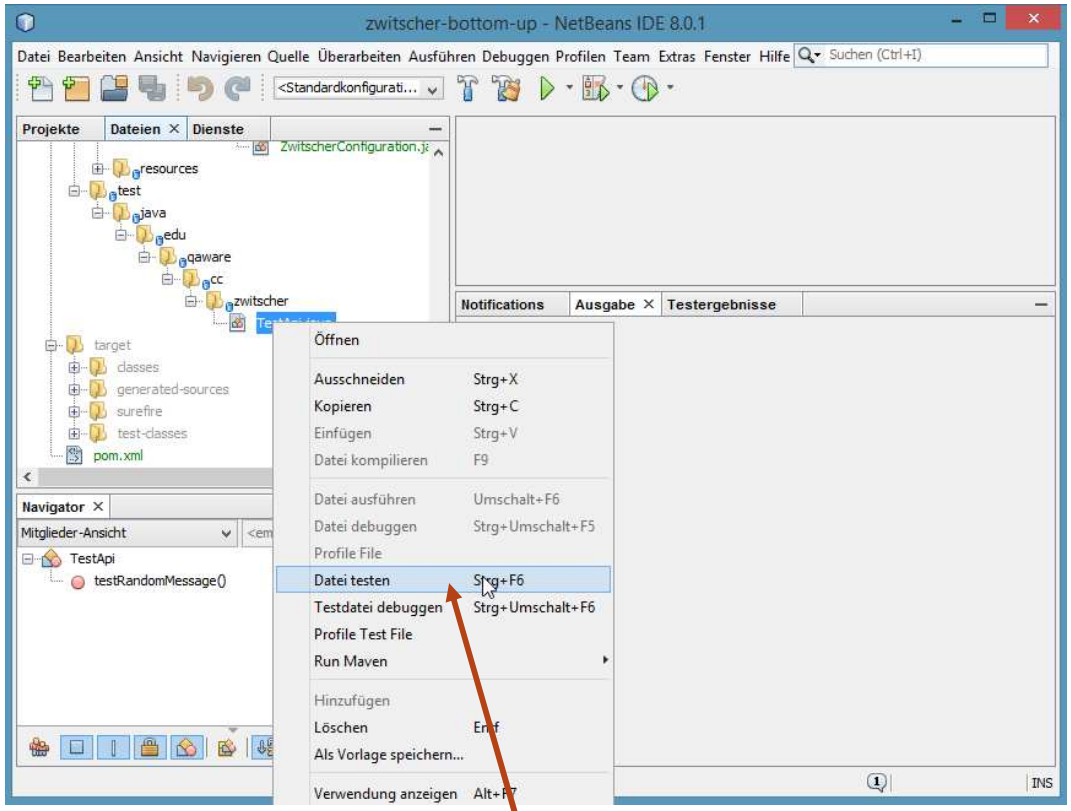
Einführung: Netbeans



Netbeans: Code starten

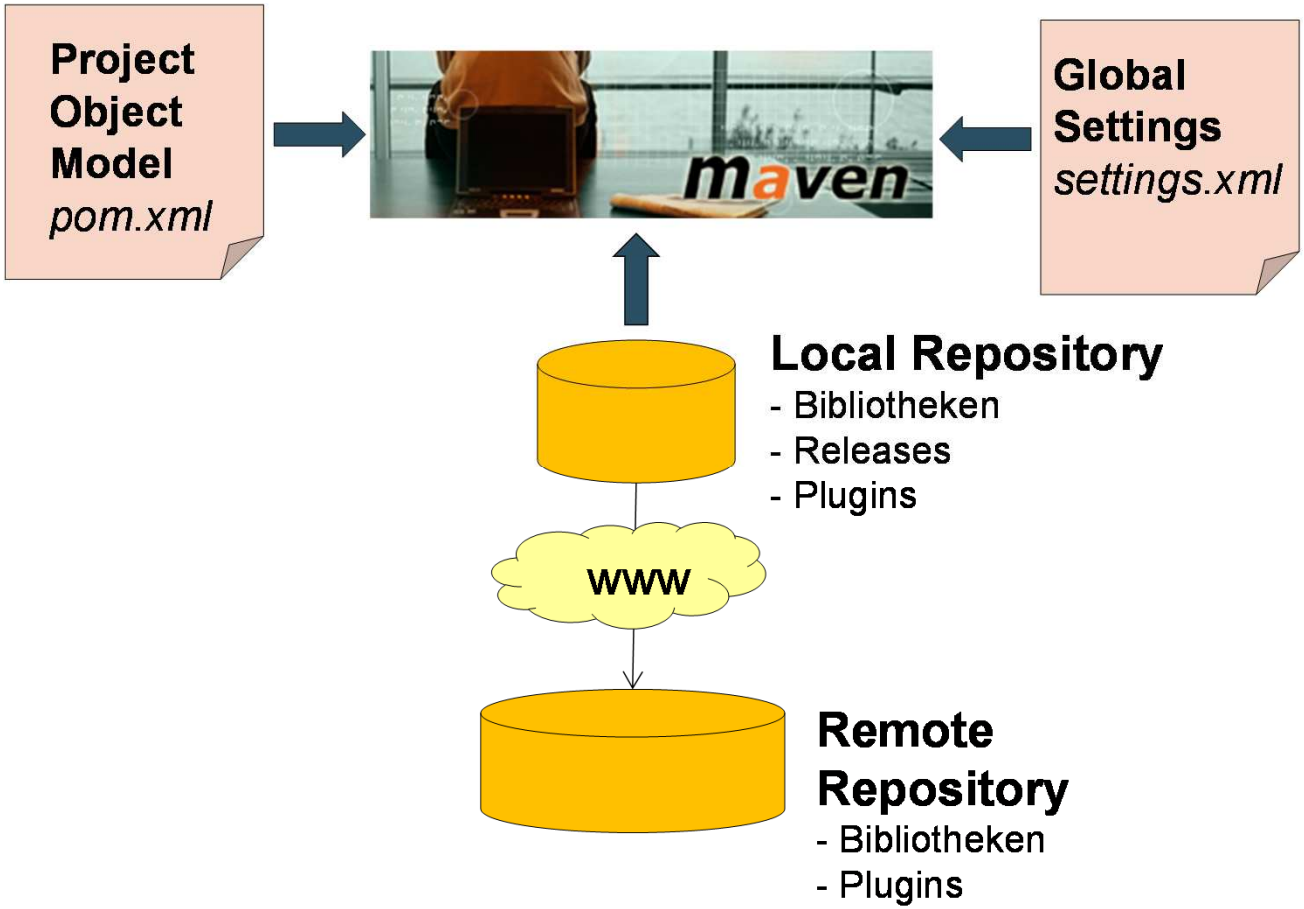


Datei mit *main()* Methode starten

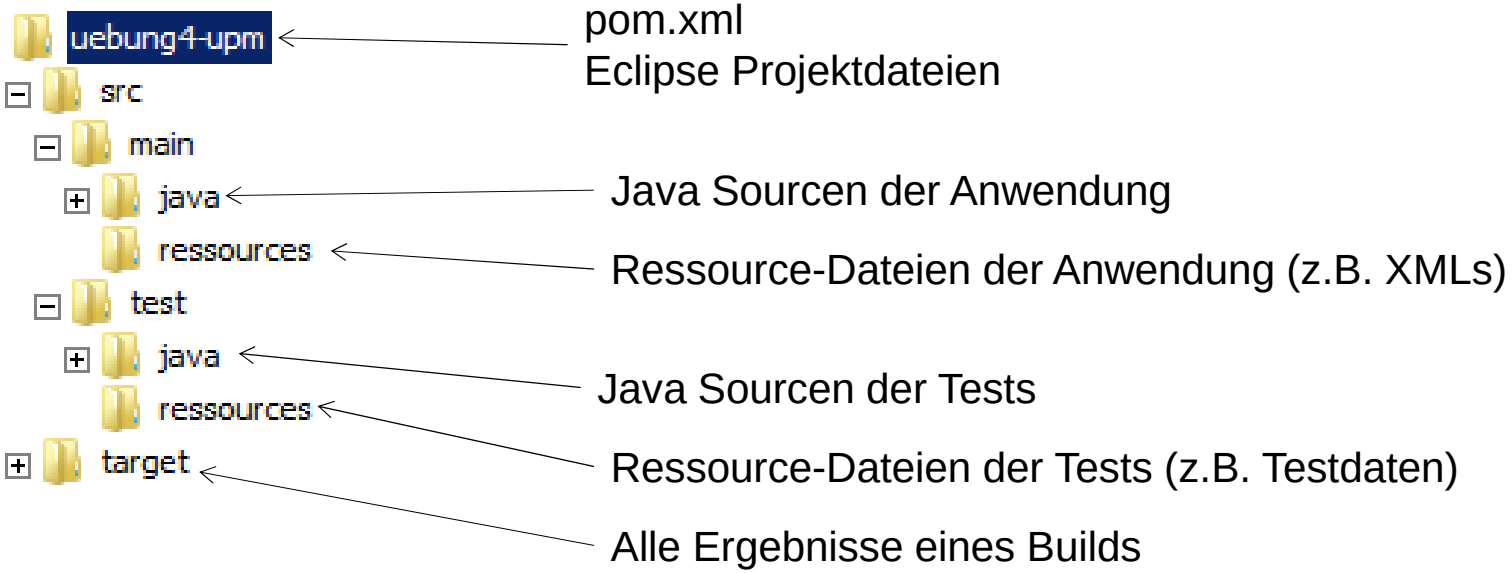


Datei mit Unit-Test starten

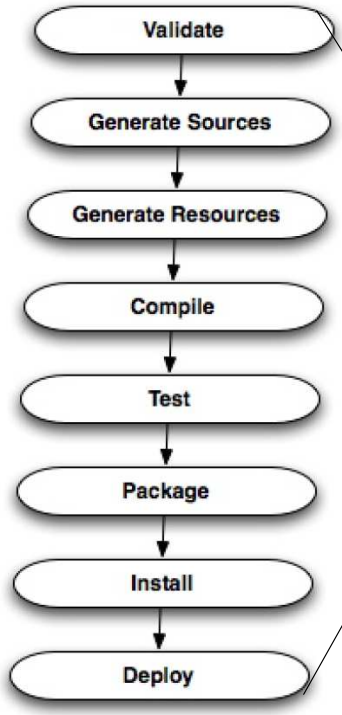
Maven: Übersicht



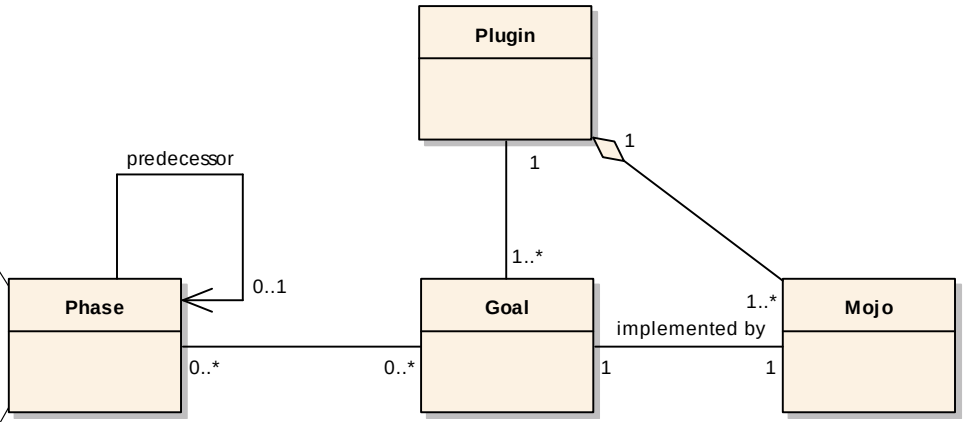
Die Maven Standard-Verzeichnisstruktur



Maven: Buildstruktur

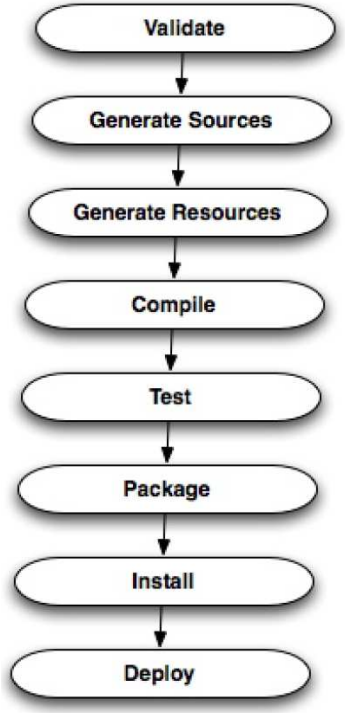


Build Lifecycle



- Phase
  - Einzelne Buildphase.
  - Abarbeitung Build bis zu einer Phase:
    `mvn <phase>`
- Goal
  - Einzelne Buildaufgabe. (z.B. Testüberdeckung messen)
  - Direkter Aufruf eines einzelnen Goals:
    `mvn <plugin>:<goal>`

Wichtige Befehle



- **`mvn clean`**: Räumt *target* Verzeichnis auf
- **`mvn compile`**: Kompiliert den Quellcode und lädt alle dafür notwendigen Bibliotheken herunter.
- **`mvn package`**: Erzeugt ein getestetes JAR/WAR/EAR.

Inhalt einer Maven POM, der für einen Build notwendigen Beschreibungsdatei.


```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.fhr.seu</groupId>
  <artifactId>uebung4-upm</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>uebung4-upm</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Definition von Abhängigkeiten

- Alle Abhängigkeiten werden über das Repository aufgelöst
  - **Maven-interne Abhängigkeiten**: z.B. Plugins
  - **Abhängigkeiten Projekt zu Drittbibliotheken**
  - **Abhängigkeit Projekt zu anderen (Teil-)Projekten**
- Angabe per Maven Coordinate

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>
```

  - Transitive Abhängigkeiten werden automatisch aufgelöst
- Zusätzlich Angabe Scope
  - **compile**: In allen Klassenpfaden verfügbar (default)
  - **runtime**: Wird nicht zur Kompilierung, aber bei der Ausführung benötigt
  - **test**: Wird nur für die Ausführung der Tests benötigt
  - **provided**: Nur zur zur Kompilierung benötigt. Wird zur Laufzeit z.B. durch einen Container bereitgestellt.

Netbeans: Umgang mit Maven