QAWARE
SOFTWARE ENGINEERING

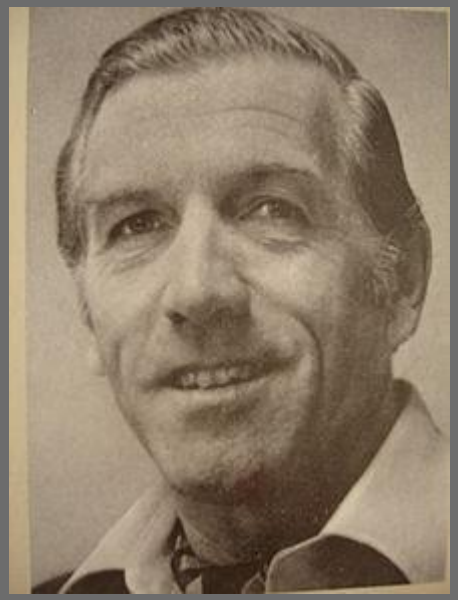
Cloud Computing

Kapitel 1: Kommunikationssysteme im Internet

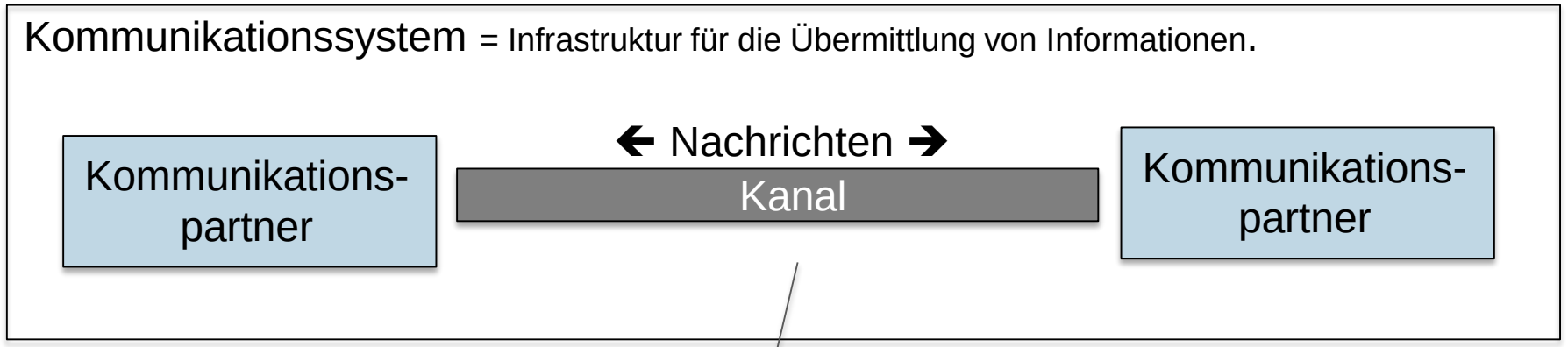
Dr. Simon Bäumler

Grundlagen von Kommunikationssystemen im Internet

„Man kann nicht nicht kommunizieren!“
Paul Watzlawik



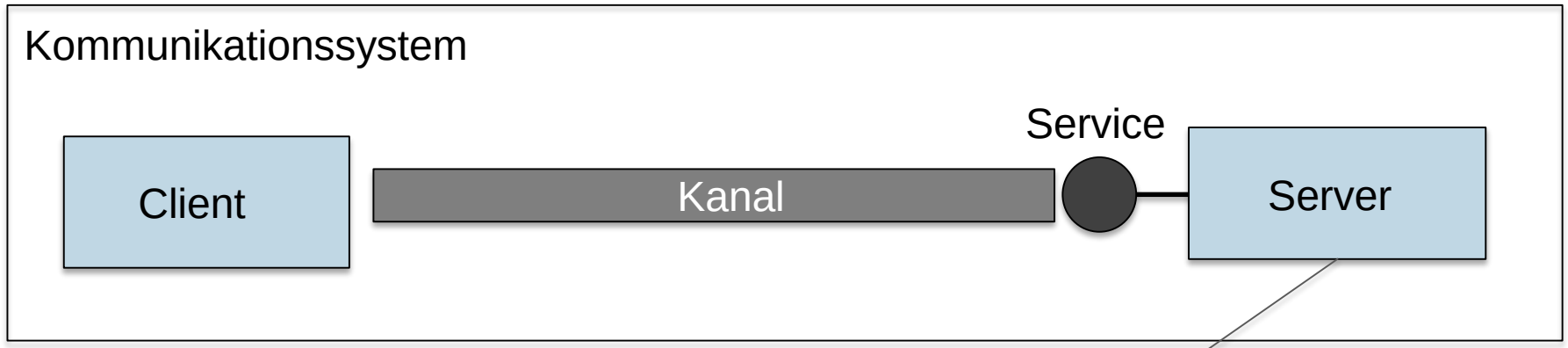
Ein allgemeines Kommunikationsmodell im Internet. Angelehnt an das Modell von Shannon/Weaver.



Typische Kanaleigenschaften:

- **Richtung**
- **Datenformat**/Codierung
- Synchronität (synchron/asynchron)
- Zuverlässigkeit und Garantien
- Sicherheit
- Performance (Latenz, Bandbreite)
- Overhead (Nutzlast / Gesamtlast)

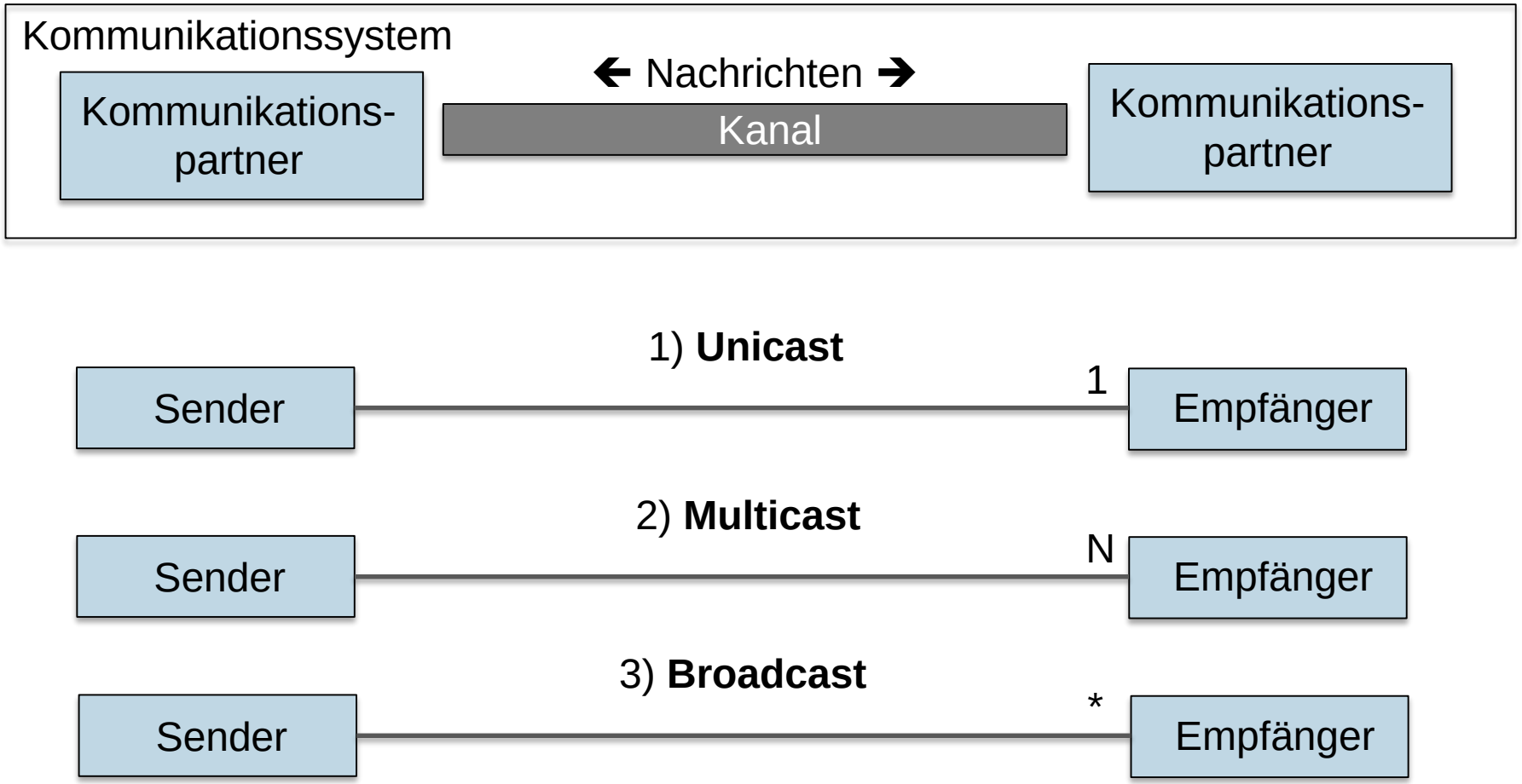
Service-Orientierung in einem Kommunikationssystem.



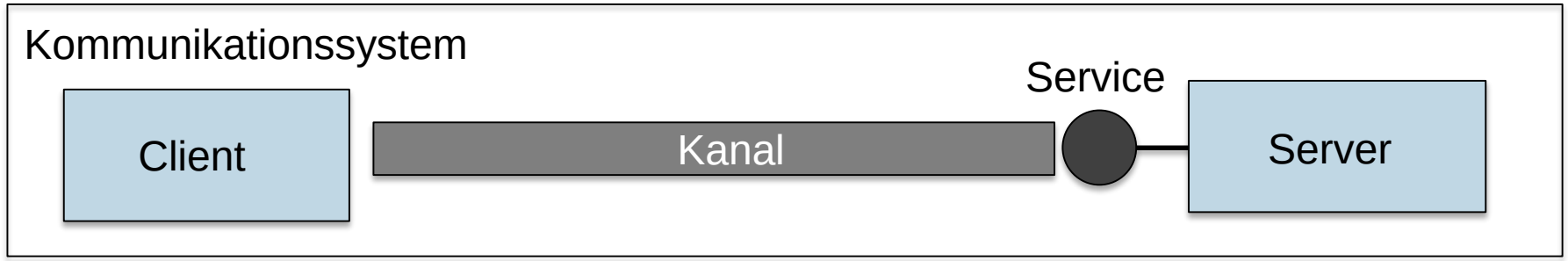
Ein **Service** ist eine Funktionalität, die über eine definierte Schnittstelle zur Verfügung stellt. Jeder Service ist definiert durch eine **Serviceschnittstelle**.

Eine **Serviceschnittstelle** ist ein Vertrag zwischen Nutzer und Anbieter über Syntax und Semantik der Service-Nutzung und enthält optional Zusicherungen in Hinblick auf den **Quality of Service**.

Klassifikation von Kommunikationssytemen: Kardinalität der Empfänger einer Nachricht.



Klassifikation von Kommunikationssytemen: (B) Wer beginnt mit der Kommunikation?



Client beginnt die Kommunikation.

Request/Response

Anfrage mit der Bitte um Antwort.

Benachrichtigung in eine Richtung.

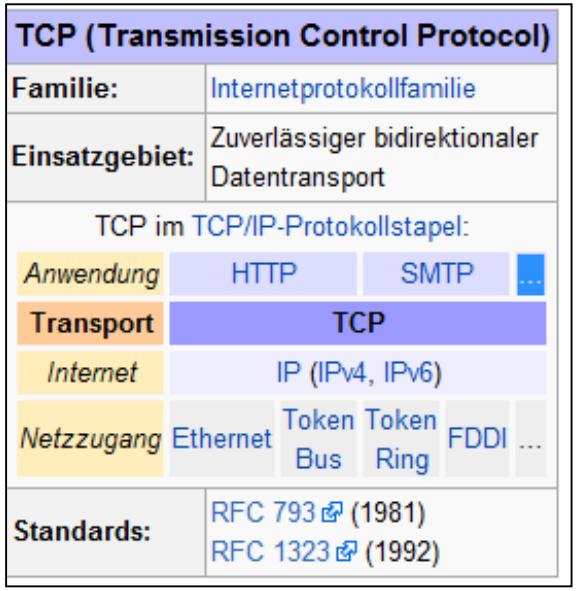
Push

Server beginnt die Kommunikation.

Peer-to-Peer

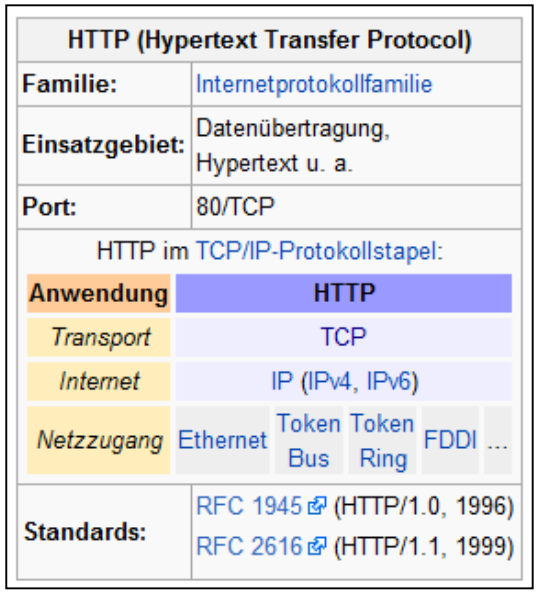
Beide Kommunikationspartner können die Kommunikation beginnen (Ambivalenz der Rollen Client und Server).
Nachrichtenaustausch in beide Richtungen.

Basis aller Cloud-Kommunikationstechnologien ist TCP und teilweise HTTP.



TCP (Transmission Control Protocol)						
Familie:	Internetprotokollfamilie					
Einsatzgebiet:	Zuverlässiger bidirektionaler Datentransport					
TCP im TCP/IP-Protokollstapel:						
Anwendung	HTTP	SMTP	...			
Transport	**TCP**					
Internet	IP (IPv4, IPv6)					
Netzzugang	Ethernet	Token Bus	Token Ring	FDDI	...	
Standards:	RFC 793 (1981) RFC 1323 (1992)					

- Ab 1973 entwickelt und 1981 standardisiert.
- Zuverlässige Voll-Duplex Ende-zu-Ende Verbindung.
- Ein Endpunkt ist eine IP + Port.

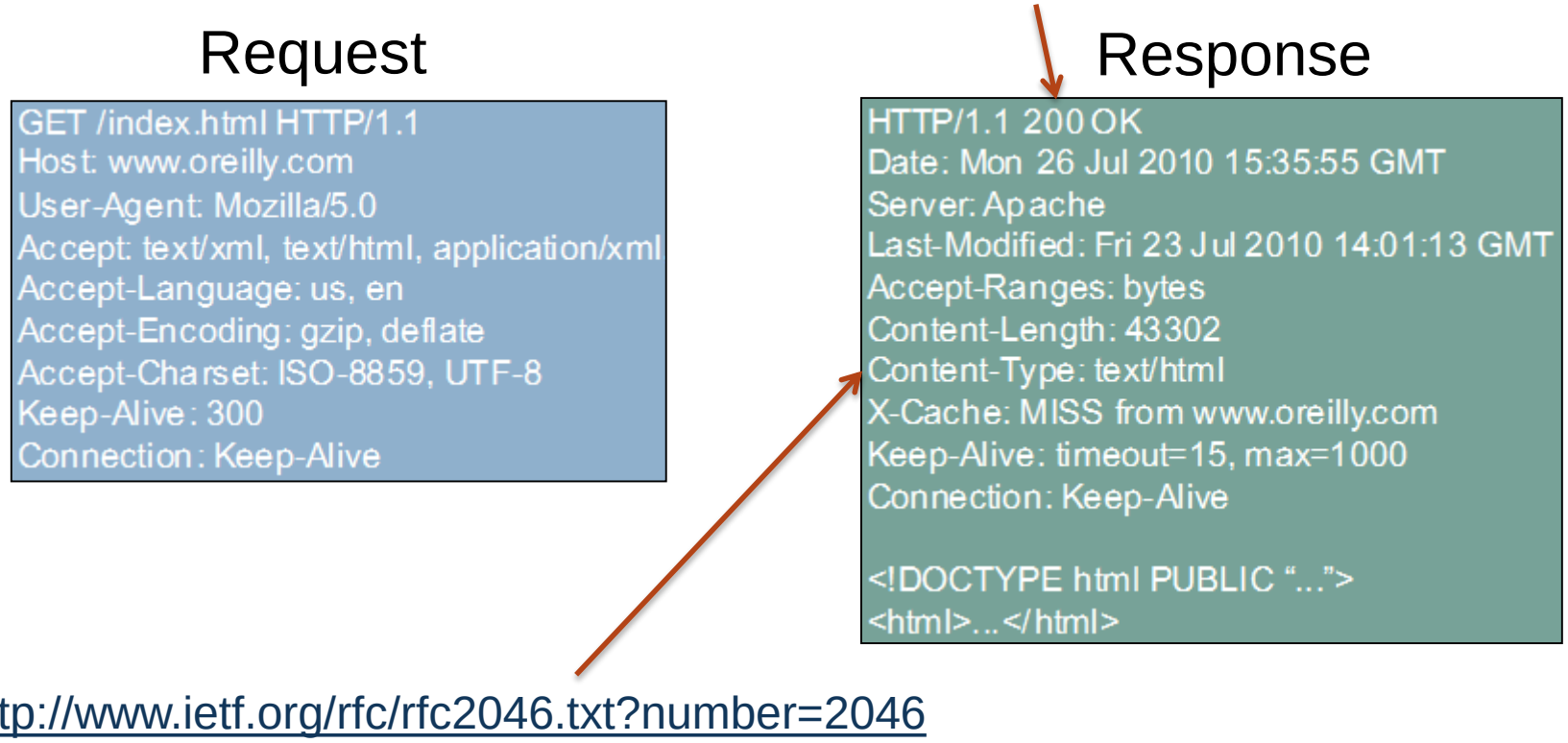


HTTP (Hypertext Transfer Protocol)						
Familie:	Internetprotokollfamilie					
Einsatzgebiet:	Datenübertragung, Hypertext u. a.					
Port:	80/TCP					
HTTP im TCP/IP-Protokollstapel:						
Anwendung	**HTTP**					
Transport	TCP					
Internet	IP (IPv4, IPv6)					
Netzzugang	Ethernet	Token Bus	Token Ring	FDDI	...	
Standards:	RFC 1945 (HTTP/1.0, 1996) RFC 2616 (HTTP/1.1, 1999)					

- HTTP 1.0: 1989 am CERN entwickelt.
- HTTP 1.1: Connection Pooling / Keepalive, HTTP-Pipelining, Methoden PUT und DELETE.
- HTTP 2.0: Binär-Stream, Multiplexing, Verschlüsselung als Standard, div. Performance-Optimierungen, Push. (siehe https://http2.github.io)

Ein Beispiel für eine HTTP-Kommunikation.

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html



http://www.ietf.org/rfc/rfc2046.txt?number=2046

Typische Datenformate im Internet: (1) XML

```
<Kreditkarte
  Herausgeber="Xema"
  Nummer="1234-5678-9012-3456"
  Deckung="2e+6"
  Waehrung="EURO">
  <Inhaber
    Name="Mustermann"
    Vorname="Max"
    maennlich="true"
    Alter="42"
    Partner="null">
    <Hobbys>
      <Hobby>Reiten</Hobby>
      <Hobby>Golfen</Hobby>
      <Hobby>Lesen</Hobby>
    </Hobbys>
    <Kinder />
  </Inhaber>
</Kreditkarte>
```

- XML = eXtensible Markup Language
  (Daten und ihre Beschreibung)
- MIME-Typ: *text/xml, application/xml* bzw. *application/<xml-lang>+xml*.
- Schema-Sprachen: XML Schema, DTD, Relax NG
- Datentypen
  - Elemente
  - Attribute
  - Textknoten
  - Listen, Sequenzen, Auswahlen
  - 19 primitive Datentypen (string, integer, bool, …)
  - 25 abgeleitete Datentypen (ID, IDREF, URI, …)

Typische Datenformate im Internet: (2) JSON

- JSON = JavaScript Object Notation
  (Daten pur). Auch in Binärcodierung (BSON – Binary JSON).
- MIME-Typ: *application/json*
- Schema-Sprachen: JSON Schema (http://json-schema.org)
- Datentypen
  - Nullwert: **`null`**
  - bool'scher Wert: **`true`**, **`false`**
  - Zahl: **`42`**, **`2e+6`**
  - Zeichenkette: **`"Mustermann"`**
  - Array: **`[1,2,3]`**
  - Objekt mit Eigenschaften: **`{"Name":"Mustermann"}`**

Service-orientierte Request/Response-Kommunikation mit REST

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 (PUT = erzeugen, POST = 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.

Beispiele für REST-Aufrufsyntax: Schnittstellenentwurf über Substantive.

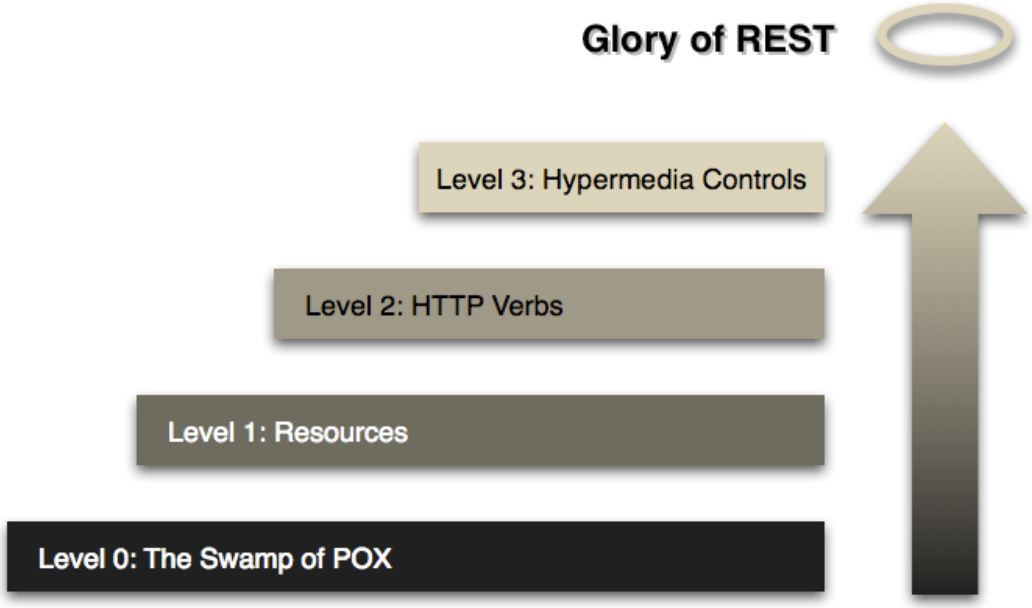
- Produkte aus der Kategorie Spielwaren:
  http://www.service.de/produkte/spielwaren
- Bestellungen aus dem Jahr 2008
  http://www.service.de/bestellungen/2008
- Liste aller Regionen, in denen der Umsatz größer als 5 Mio. Euro war
  http://www.service.de/regionen/umsatz/summe?groesserAls=5M
- Gib mir 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

Mit dem REST Maturity Model kann bewertet werden, wie RESTful ein HTTP-basierter Service ist.



http://martinfowler.com/articles/richardsonMaturityModel.html

http://www.service.de/customers/4711/orders
Eingabe: *Order* per PUT
Ausgabe: *HTTP-Code (200, 500)* +
Link auf geänderte Order

http://www.service.de/customers/4711/orders
Eingabe: *Order* per PUT
Ausgabe: *HTTP-Code (200, 500)*

http://www.service.de/customers/4711/orders
Eingabe: *Order* per POST
Ausgabe: *UpdateOrderConfirmation (OK)*

http://www.service.de/updateOrderService
Eingabe: *UpdateOrderRequest* per POST
Ausgabe: *UpdateOrderConfirmation (OK)*

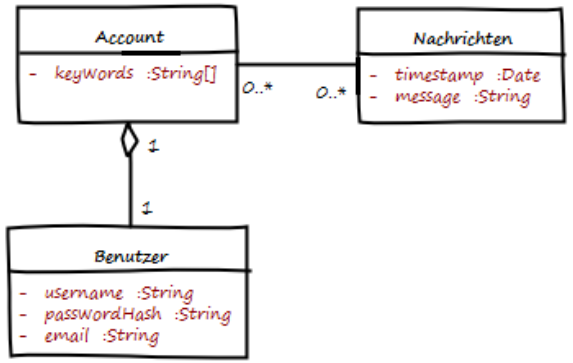
HATEOAS (Hypermedia as the Engine of Application State)

Entwicklung von REST APIs

Anwendungsfälle erheben

- 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

Entitätenmodell erstellen



REST Schnittstelle umsetzen

Var. 1

Top-down Ansatz:

- REST Schnittstelle definieren
- REST Schnittstelle generieren
- REST Schnittstelle implementieren

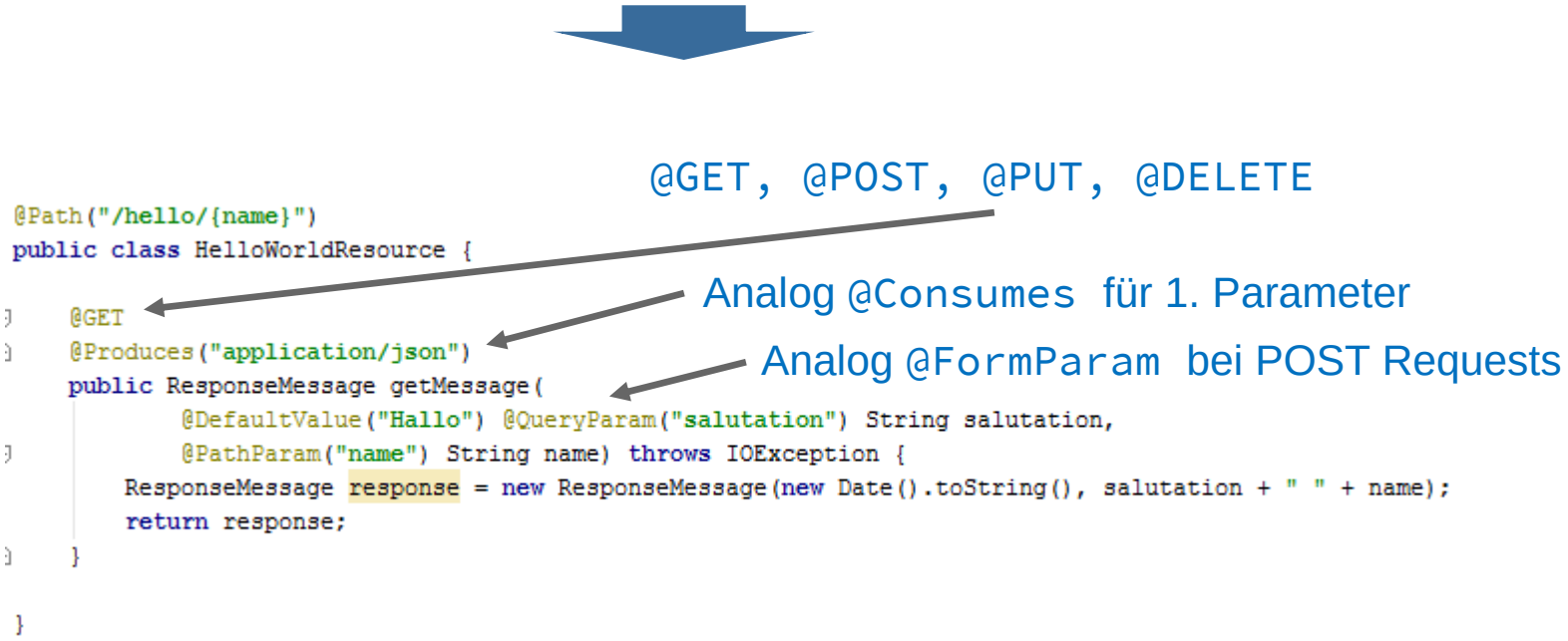
Var. 2

Bottom-up Ansatz:

- REST Schnittstelle implementieren
- Definition REST-Schnittstelle generieren

REST-Webservices mit JAX-RS.

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



Die effizienten Alternativen: Binärprotokolle

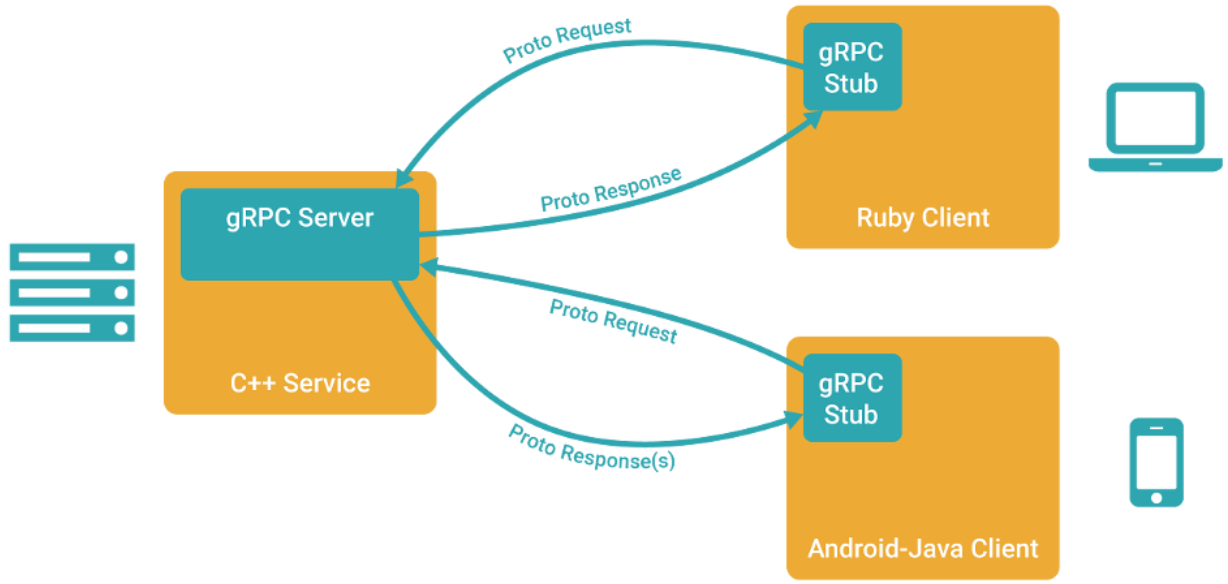
- Binärprotokolle sind eine sinnvolle Alternative zu REST, wenn eine effiziente und programmiersprachennahe Kommunikation erfolgen soll.
  - Encoding der Payload als komprimiertes Binärformat
  - Separate Schnittstellenbeschreibungen (IDLs, *Interface Definiton Languages*) aus denen dann Client- und Server-Code in mehreren Programmiersprachen generiert werden können
- Kandidaten
  - gRPC / Protocol Buffers
  - Apache Avro
  - Apache Thrift
  - Hessian
- Binärprotokolle können auch mit REST kombiniert werden: Als Content-Type und damit als Payload wird eine Binär-Codierung verwendet. Beispiel: Protocol Buffers over REST.

gRPC

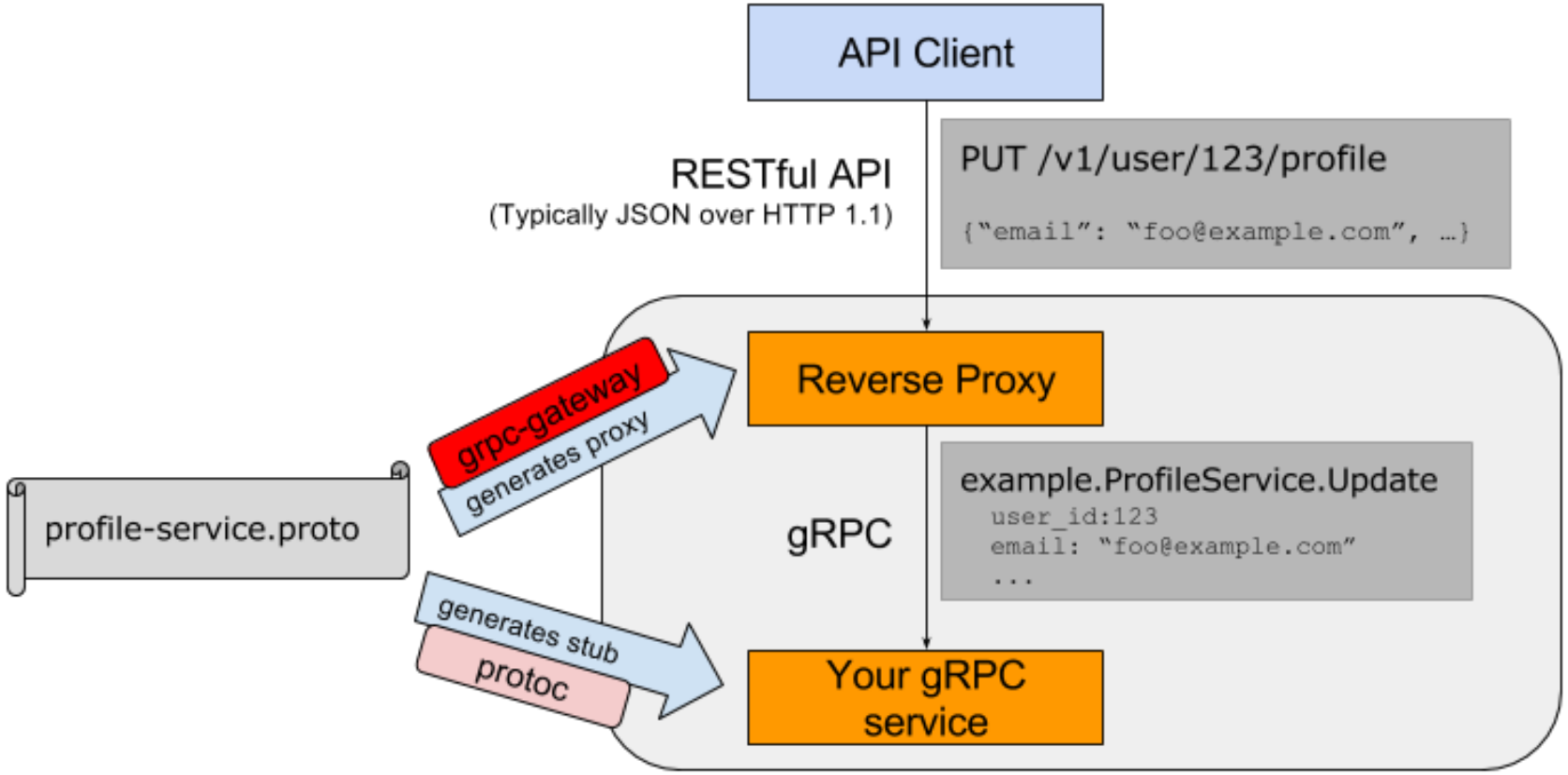
- Open-Source-Binärprotokoll von Google auf Basis der Protocol Buffers Binärcodierung (http://www.grpc.io/docs)
- Flexibel erweiterbarer Generator (`protoc`) für Server- und Client-Code (*Skeleton* und *Stubs*).

```
syntax = "proto3";

option java_package = "io.grpc.examples";

package helloworld;

// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

`protoc`



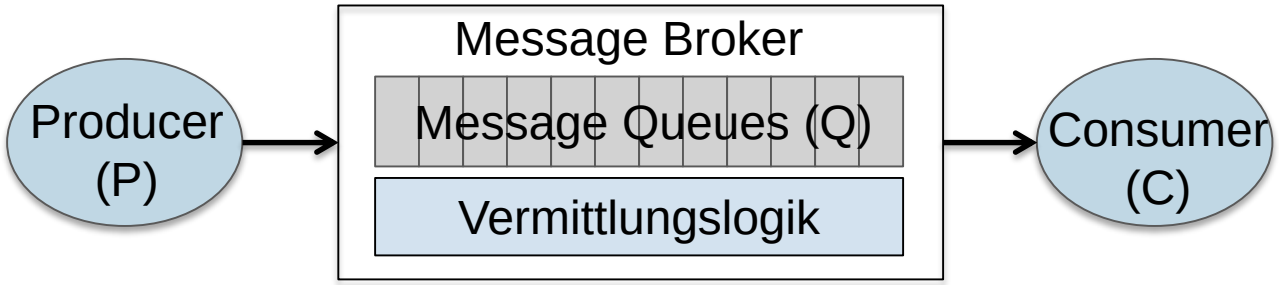
Dank HTTP/2 Multiplexing kann eine Anwendung auf dem selben HTTP-Port sowohl ein Binärprotokoll als auch REST anbieten.



https://github.com/gengo/grpc-gateway

Flexible Kommunikationsmuster mit Messaging

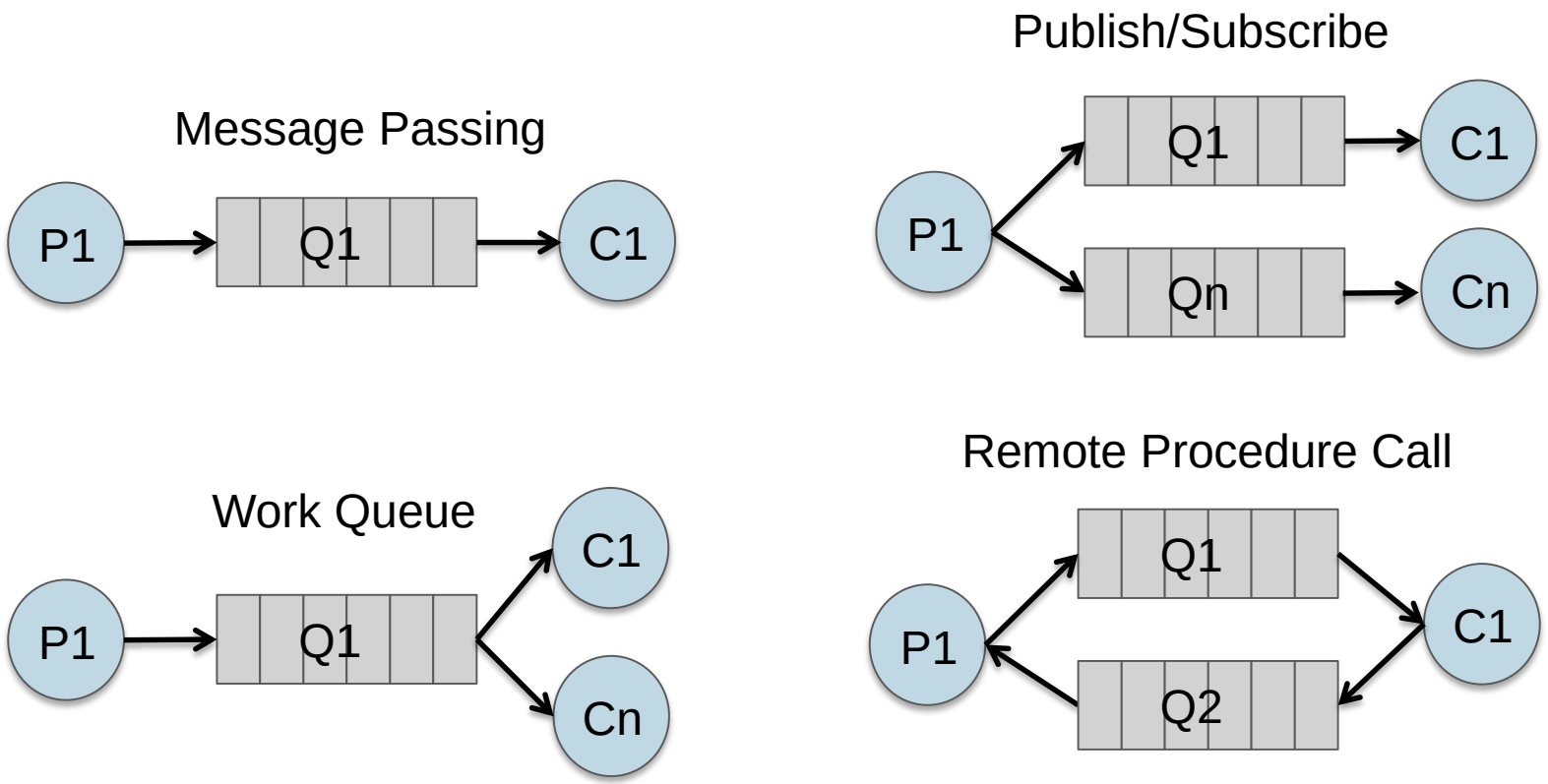
Messaging ist zuverlässiger, asynchroner Nachrichtenaustausch.



- **Entkopplung von Producer und Consumer.**
  Die Serviceschnittstelle ist lediglich das Format der Nachricht. Message Broker machen zum Format keinen Einschränkungen. Sende-Zeitpunkt und Empfangs-Zeitpunkt können beliebig lange auseinander liegen.
- **Skalierbarkeit.** Die Vermittlungslogik entscheidet zentral …
  - … an wie viele Consumer die Nachricht ausgeliefert wird (horizontale Skalierbarkeit),
  - an welchen Consumer die Nachricht ausgeliefert wird (Lastverteilung),
  - wann eine Nachricht ausgeliefert wird (Pufferung von Lastspitzen),

  auf Basis von konfigurierten Anforderungen an die Vermittlung:
  - Maximale Zustelldauer bzw. Lebenszeit der Nachricht
  - Geforderte Zustellgarantie (mindestens 1 Mal, exakt 1 Mal, an alle) und Transaktionalität
  - Priorität der Nachricht
  - Notwendige Einhaltung der Zustellreihenfolge

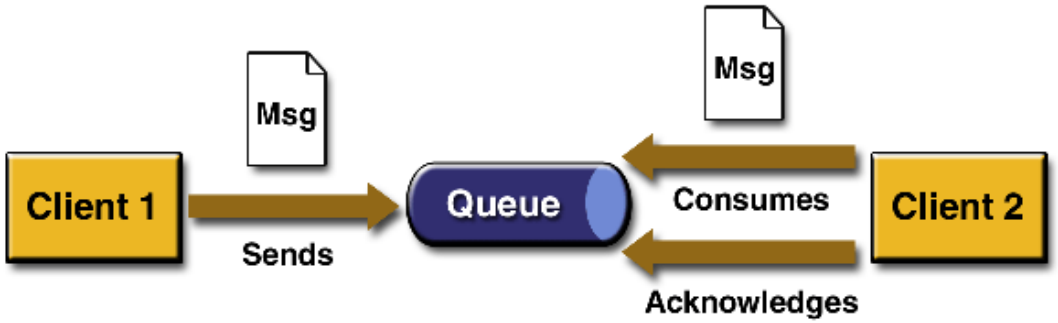
Messaging ist eine flexible Kommunikationsart, mit der sich vielfältige Kommunikationsmuster umsetzen lassen.



JMS

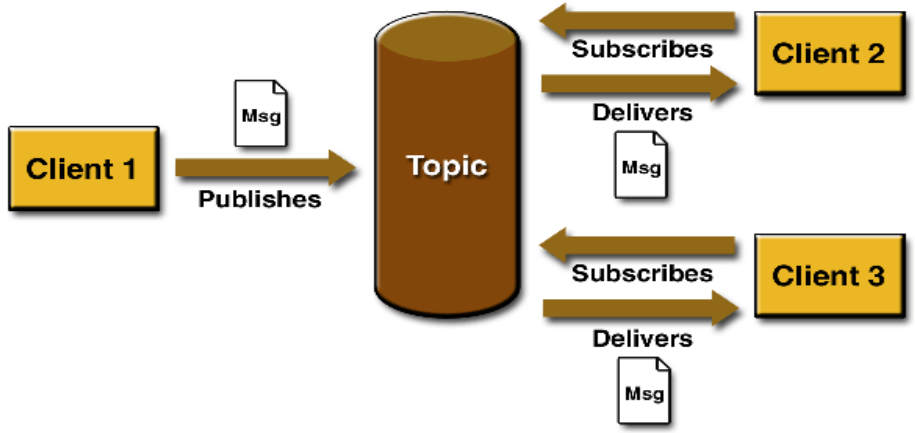
- JMS = Java Messaging Service. Standardisierte API im Rahmen der Java-Enterprise-Edition-Spezifikation. Standardisiert nicht das Messaging-Protokoll.
  - 2002-2013: Version 1.1. Sehr stabil und weit verbreitet in der Java-Welt.
  - Seit Mai 2013: Version 2.0 als Teil der JEE 7 Spezifikation
- Unterstützte Kommunikationsmuster:

Message Passing:



- Ein Consumer pro Message
- Der Erhalt einer Nachricht wird bestätigt

Publish / Subscribe:



- Mehrere Consumer pro Message

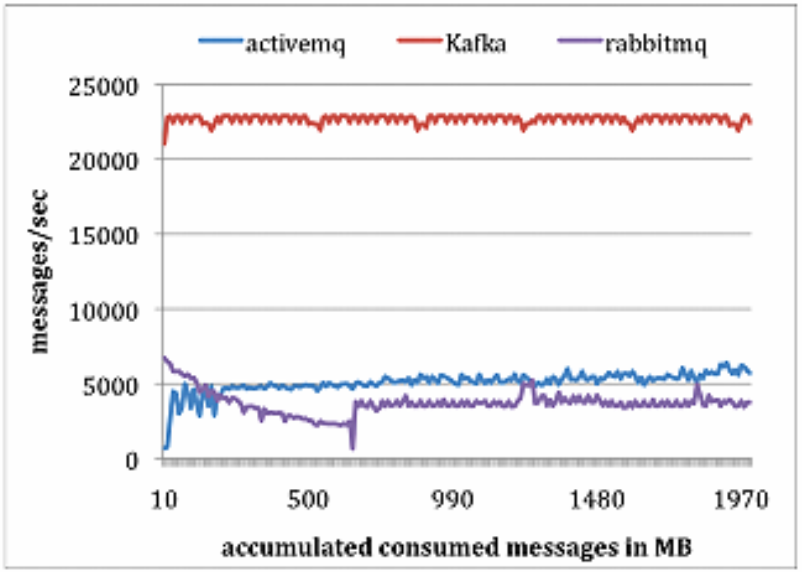
AMQP: Ein Standard-Protokoll für Messaging-Systeme.

- **Problem**: Message Broker sind intern proprietär aufgebaut (Beispiel: IBM MQSeries mit 80% Marktanteil im kommerziellen Bereich). Sie sind nicht zueinander interoperabel, wie man es z.B. von SNMP-Servern her kennt. Das ist besonders beim Messaging über Firmengrenzen und Technologie-Stacks hinweg ein Problem.

- **Lösung AMQP**: Standardisierung eines interoperablen Protokolls für Messaging-Broker. AMQP steht seit Ende 2011 in der Version 1.0 zur Verfügung.
  - Im Standardisierungsgremium sind u.A. Cisco, Microsoft, Red Hat, Deutsche Börse Systems, IONA, Novell, Credit Suisse, JPMorganChase.
  - Standardisiert ein Netzwerk-Protokoll für die Kommunikation zwischen den Clients und den Message Brokern.
  - Standardisiert ein Modell der verfügbaren APIs und Bausteine für die Vermittlung und Speicherung von Nachrichten (Producer, Exchange, Queue, Consumer).
  - Unterstützung aller bekannter Messaging-Muster.

Kafka

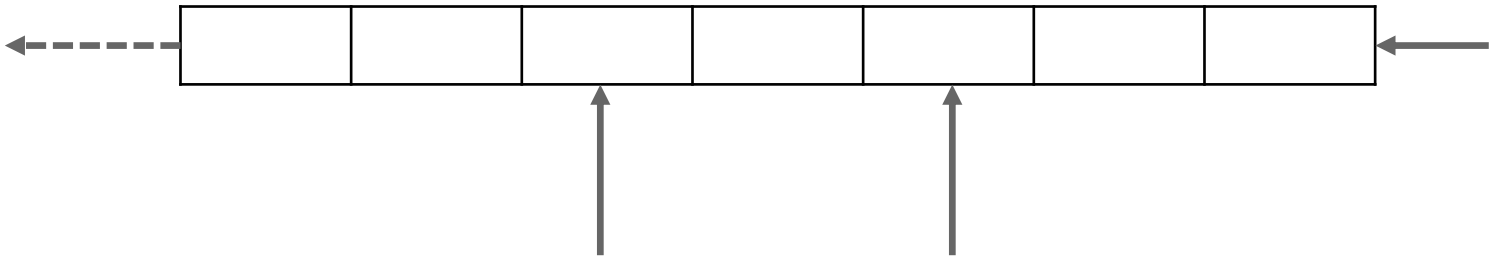
- Entwickelt bei LinkedIn und 2011 als Open Source Projekt veröffentlicht
- Kafka hat sich zum de-facto Standard in der Cloud für Messaging entwickelt, da Kafka hochgradig verteilbar und deutlich schneller als vergleichbare Lösungen ist:



- Kafka ist so schnell, da es Betriebssystem-Mittel intelligent nutzt, ein effizientes Codierungsformat für Nachrichten besitzt und den Auslieferungszustand in den Clients hält.
- Kafka ist in Java und Scala geschrieben. Die Kafka API ist proprietär und orientiert sich an keinem Messaging-Standard.

Kafka basiert auf dem Konzept eines Event-Logs. Jeder Consumer hat einen eigenen Lese-Zeiger im Log.

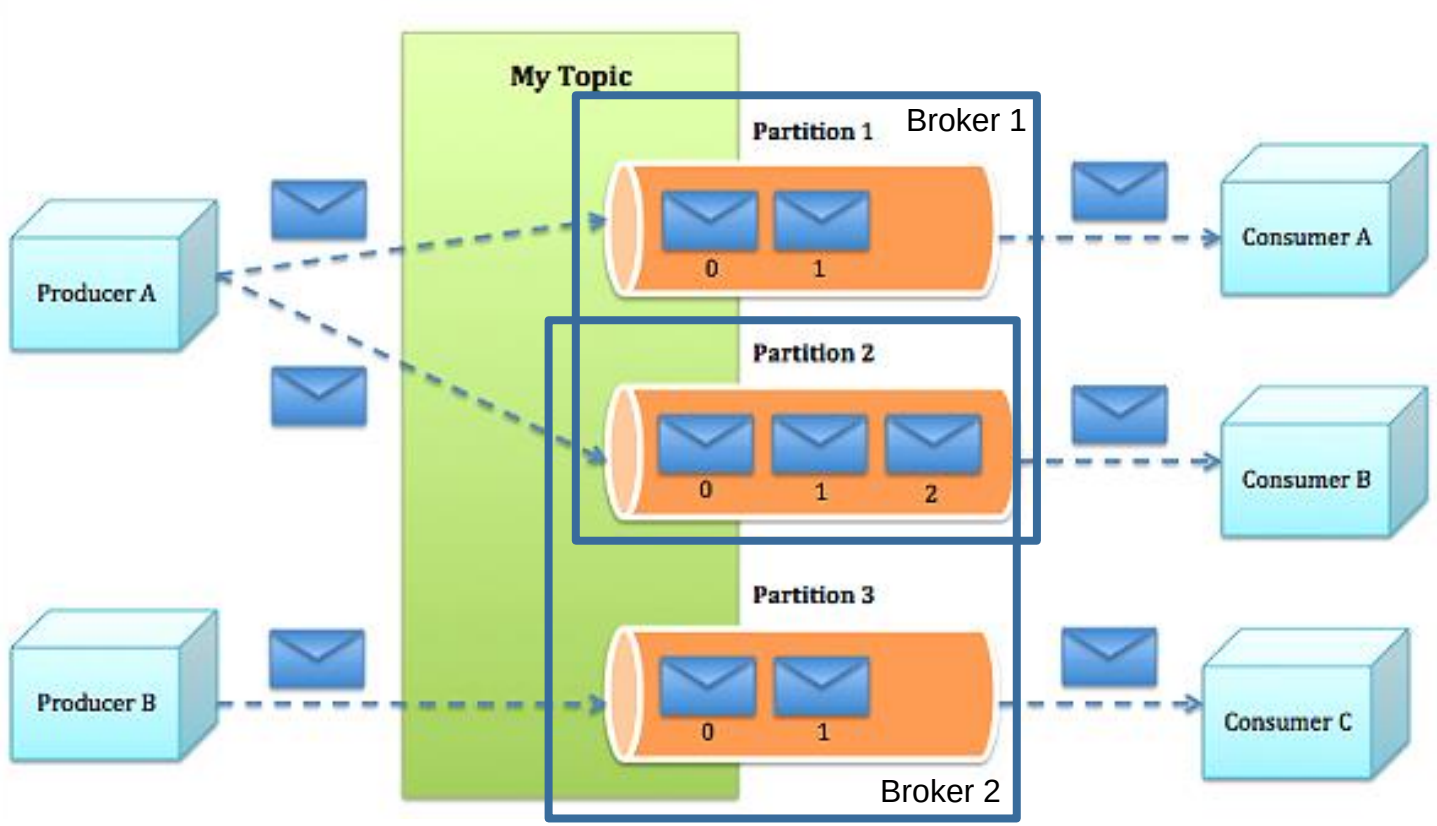
Alte Events werden gemäß definierter Kriterien gelöscht (z.B. Alter, max. Topic-Größe)



Neue Events werden immer am Ende des Topics angehängt

Zeiger auf letztes gelesenes Event eines Clients (verwaltet der Client selbst).

Der Event-Log in Kafka ist hochgradig verteilt.



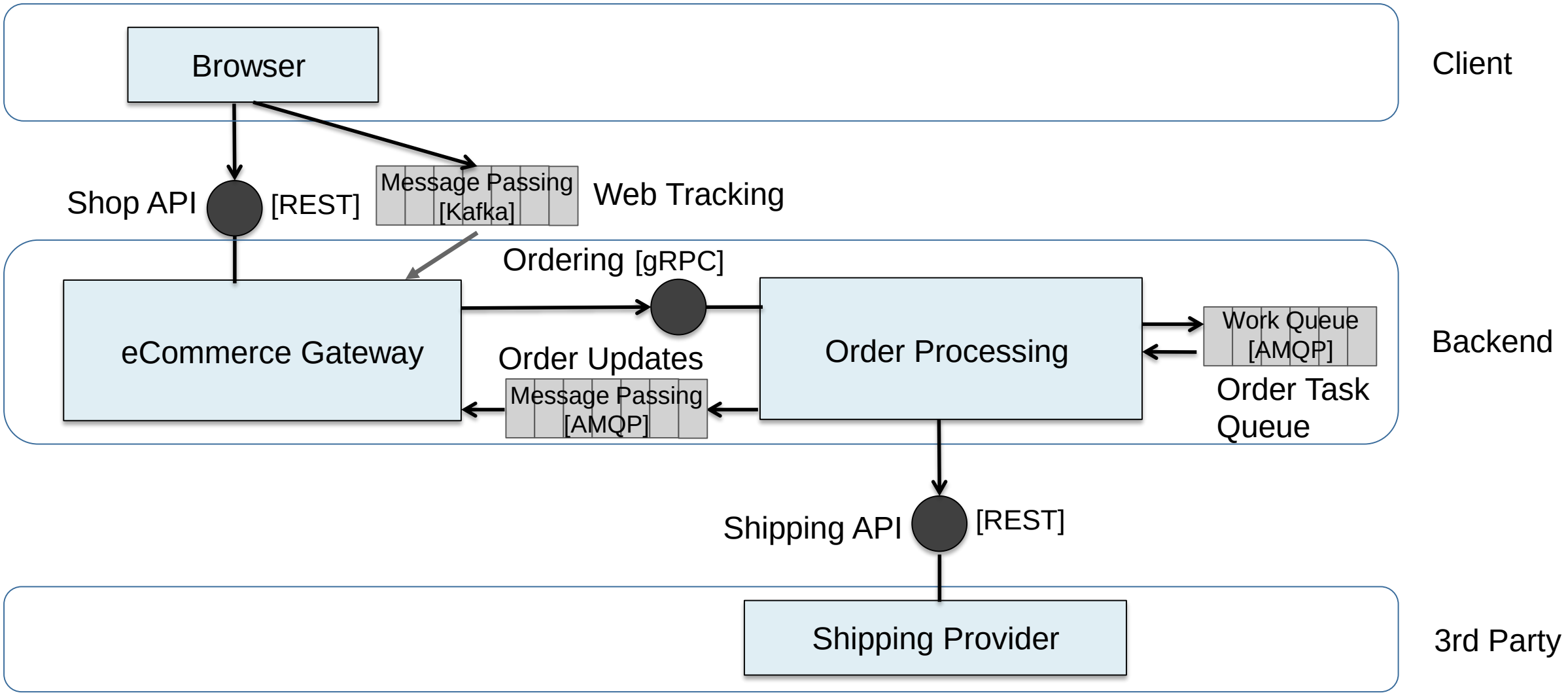
- Die Events in einem Topic werden aufgeteilt in Partitionen
- Die Partitionen werden verteilt auf die verfügbaren Broker-Instanzen
- Partitionen werden zur Fehlertoleranz repliziert

siehe:

- http://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node
- http://www.infoq.com/articles/apache-kafka

Architekturaspekte

Putting it all together…



Literatur

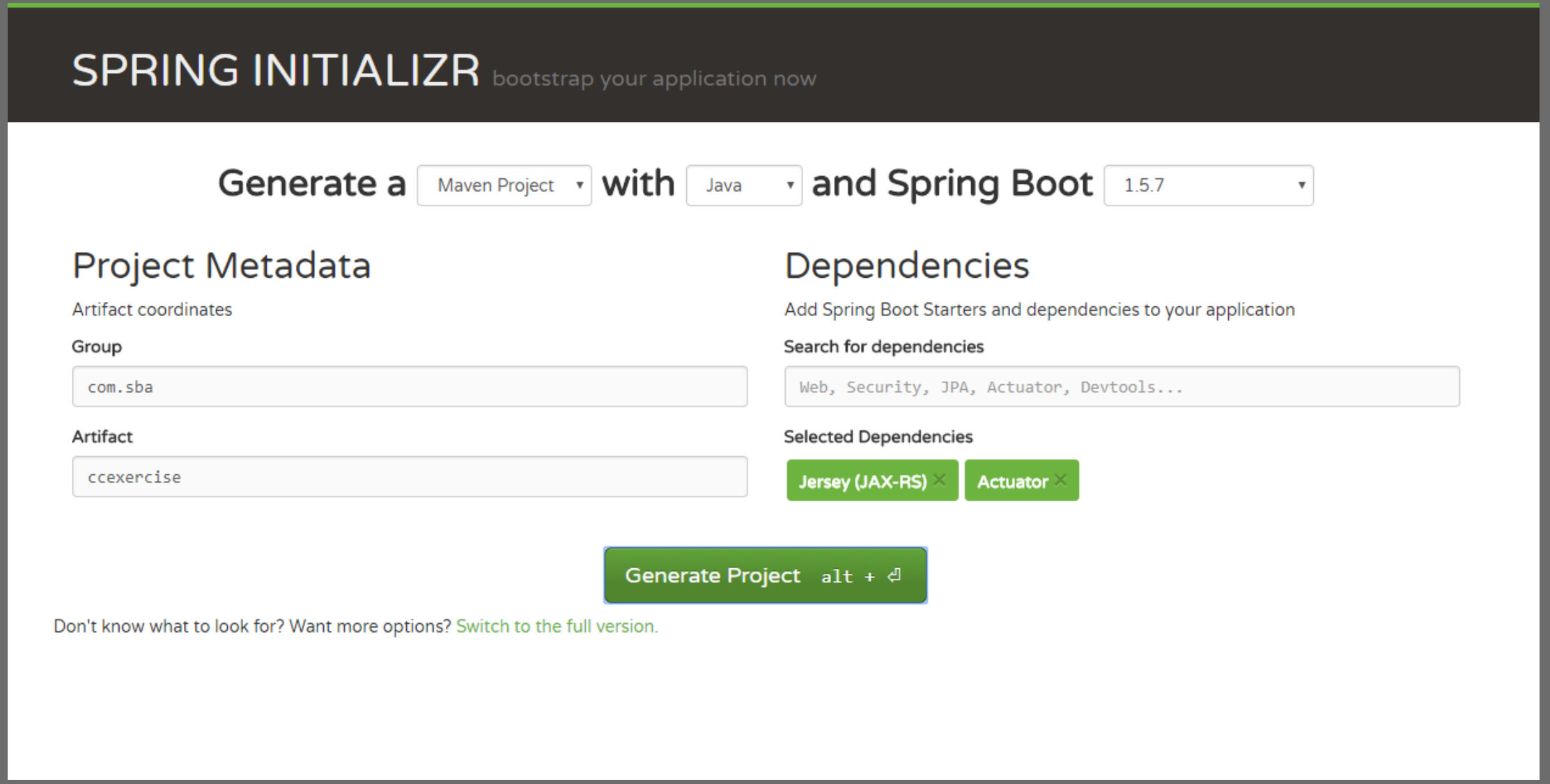
Literatur

- **Bücher:**
  - Patterns of Enterprise Application Architecture, Martin Fowler, 2002
  - Computer Networks, Andrew Tanenbaum, 2010
  - Inter-Process Communication, Hephaestus Books, 2011

- **Internet:**
  - Dissertation von Roy Fielding zu REST (http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
  - RESTful Webservices (http://www.ibm.com/developerworks/webservices/library/ws-restful)

Prolog zur Übung

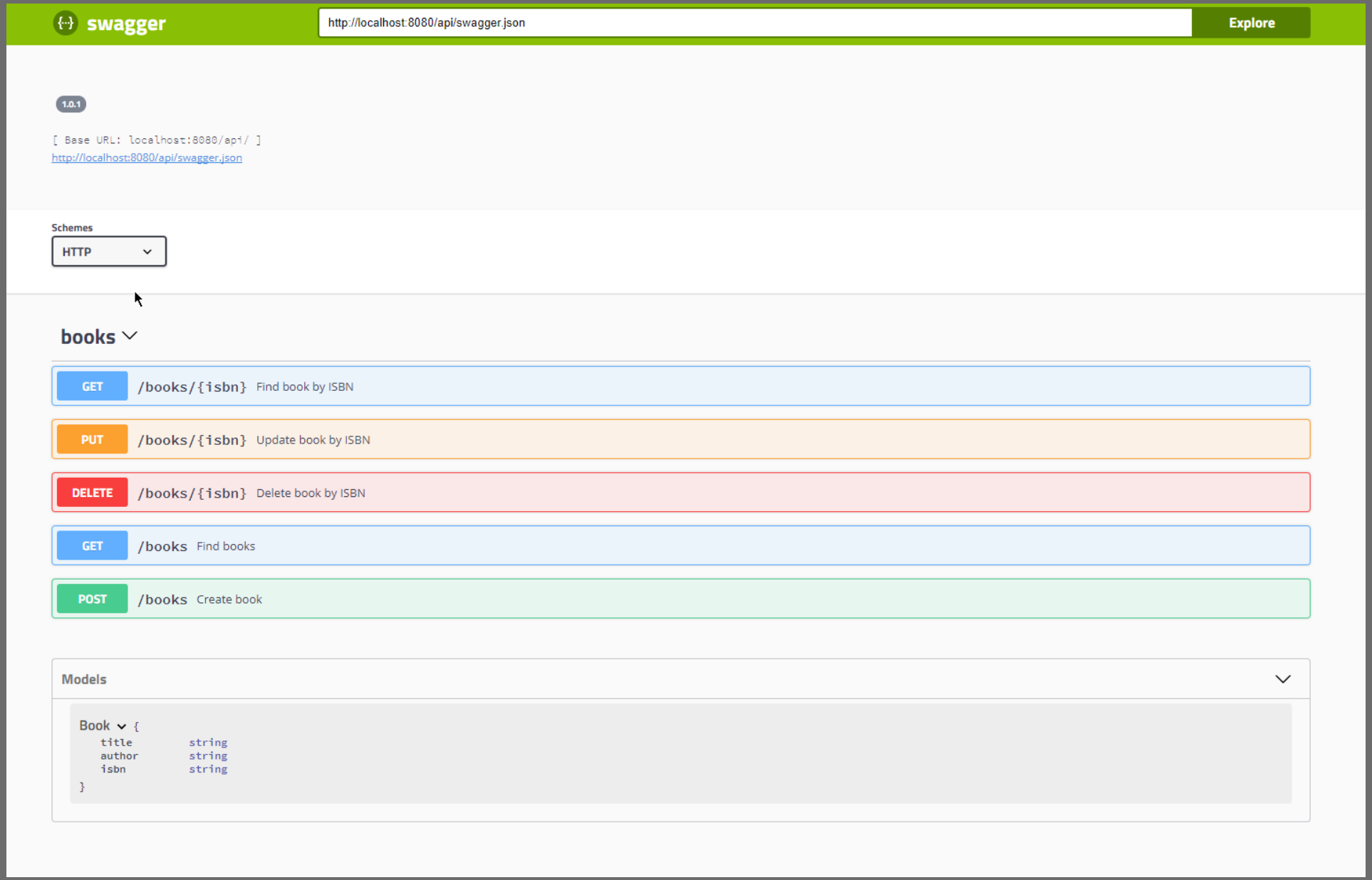
Technische Basis: Spring Boot



Ein REST-Server mit Spring Boot / Jersey

```java
/**
 * The REST resource for the books.
 */
@Component
@Path("/books")
@Api(value = "/books", description = "Operations about books")
@Produces(MediaType.APPLICATION_JSON)
public class BookResource {

    @Autowired
    private Bookshelf bookshelf;

    @GET
    @ApiOperation(value = "Find books", response = Book.class, responseContainer = "List")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "Found all books")
    })
    public Response books(@ApiParam(value = "title to search")
                          @QueryParam("title") String title) {
        Collection<Book> books = bookshelf.findByTitle(title);
        return Response.ok(books).build();
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "Create book")
    @ApiResponses(value = {
            @ApiResponse(code = 201, message = "Created the book"),
            @ApiResponse(code = 409, message = "Book already exists")
    })
    public Response create(Book book) {
        boolean created = bookshelf.create(book);
        if (created) {
            return Response.created(URI.create("/api/books/" + book.getIsbn())).build();
        } else {
            return Response.status(Response.Status.CONFLICT).build();
        }
    }
}
```

Technische Basis: Swagger



A POWERFUL INTERFACE TO YOUR API

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 ist:

- eine Schnittstellen-Spezifikations-Sprache für REST-APIs im JSON-Format. Dieses Format ist mittlerweile an ein Standardisierungsgremium übergeben, der Open API Initiative. Die Sprache wir in einer Community weiterentwickelt (http://swagger.io/specification).
- ein Satz von Werkzeugen für den Umgang mit REST-APIs in der Entwicklung:

- Alternativen: WADL, RAML, Blueprint API, WSDL

Annotation von Quellcode


```
@GET
 @Path("/{petId}")
 @Produces({ "application/json", "application/xml" })
 @ApiOperation(
         value = "Find pet by ID",
         notes = "Returns a pet when ID < 10.  ID > 10 or nonintegers will simulate API error
         conditions",
         response = Pet.class,
         authorizations = {
             @Authorization(value = "api_key")
         },
         tags={ "pet" }
 )
 @ApiResponses(value = {
     @ApiResponse(code = 200, message = "successful operation", response = Pet.class),
     @ApiResponse(code = 400, message = "Invalid ID supplied", response = Pet.class),
     @ApiResponse(code = 404, message = "Pet not found", response = Pet.class) })

 public Pet getPetById(
     @ApiParam(value = "ID of pet that needs to be fetched", required=true)
     @PathParam("petId")
       Long petId,
     @Context
       SecurityContext securityContext) throws NotFoundException {
         //...
 }
```

JSON API-Spezifikation


```
{
   "swagger":"2.0",
   "info":{
      "description":"This is a sample server Petstore server.  Yo
(http://swagger.io/irc/).  For this sample, you can use the api k
      "version":"1.0.0",
      "title":"Swagger Petstore",
      "termsOfService":"http://swagger.io/terms/",
      "contact":{
         "email":"apiteam@swagger.io"
      },
      "license":{
         "name":"Apache 2.0",
         "url":"http://www.apache.org/licenses/LICENSE-2.0.html"
      }
   },
   "host":"petstore.swagger.io",
   "basePath":"/v2",
   "tags":[
      {
         "name":"pet",
         "description":"Everything about your Pets",
         "externalDocs":{
            "description":"Find out more",
            "url":"http://swagger.io"
         }
      },
      {
         "name":"store",
         "description":"Access to Petstore orders"
      },
      {
         "name":"user",
         "description":"Operations about user",
         "externalDocs":{
            "description":"Find out more about our store",
            "url":"http://swagger.io"
         }
      }
   ],
   "schemes":[
      "http"
   ],
```

```
   "/pet/{petId}":{
      "get":{
         "tags":[
            "pet"
         ],
         "summary":"Find pet by ID",
         "description":"Returns a single pet",
         "operationId":"getPetById",
         "produces":[
            "application/xml",
            "application/json"
         ],
         "parameters":[
            {
               "name":"petId",
               "in":"path",
               "description":"ID of pet to return",
               "required":true,
               "type":"integer",
               "format":"int64"
            }
         ],
         "responses":{
            "200":{
               "description":"successful operation",
               "schema":{
                  "$ref":"#/definitions/Pet"
               }
            },
            "400":{
               "description":"Invalid ID supplied"
            },
            "404":{
               "description":"Pet not found"
            }
         },
         "security":[
            {
               "api_key":[
```

API Web-UI

swagger http://petstore.swagger.io/v2/swagger.json Authorize Explore

Swagger Petstore

This is a sample server Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

Find out more about Swagger

http://swagger.io
Contact the developer
Apache 2.0

pet : Everything about your Pets Show/Hide | List Operations | Expand Operations

POST	/pet	Add a new pet to the store
PUT	/pet	Update an existing pet
GET	/pet/findByStatus	Finds Pets by status
GET	~~/pet/findByTags~~	Finds Pets by tags
DELETE	/pet/{petId}	Deletes a pet
GET	/pet/{petId}	Find pet by ID
POST	/pet/{petId}	Updates a pet in the store with form data
POST	/pet/{petId}/uploadImage	uploads an image

store : Access to Petstore orders Show/Hide | List Operations | Expand Operations

user : Operations about user Show/Hide | List Operations | Expand Operations

[BASE URL: /v2 , API VERSION: 1.0.0]

GET /pet/{petId} Find pet by ID

Implementation Notes
Returns a single pet

Response Class (Status 200)
successful operation

Model Example Value

```
<?xml version="1.0"?>
<Pet>
  <id>1</id>
  <Category>
    <id>1</id>
    <name>string</name>
  </Category>
  <name>doggie</name>
  <photoUrl>
    <photoUrl>string</photoUrl>
```

Response Content Type application/xml

Parameters

Parameter	Value	Description	Parameter Type	Data Type
petId	(required)	ID of pet to return	path	long

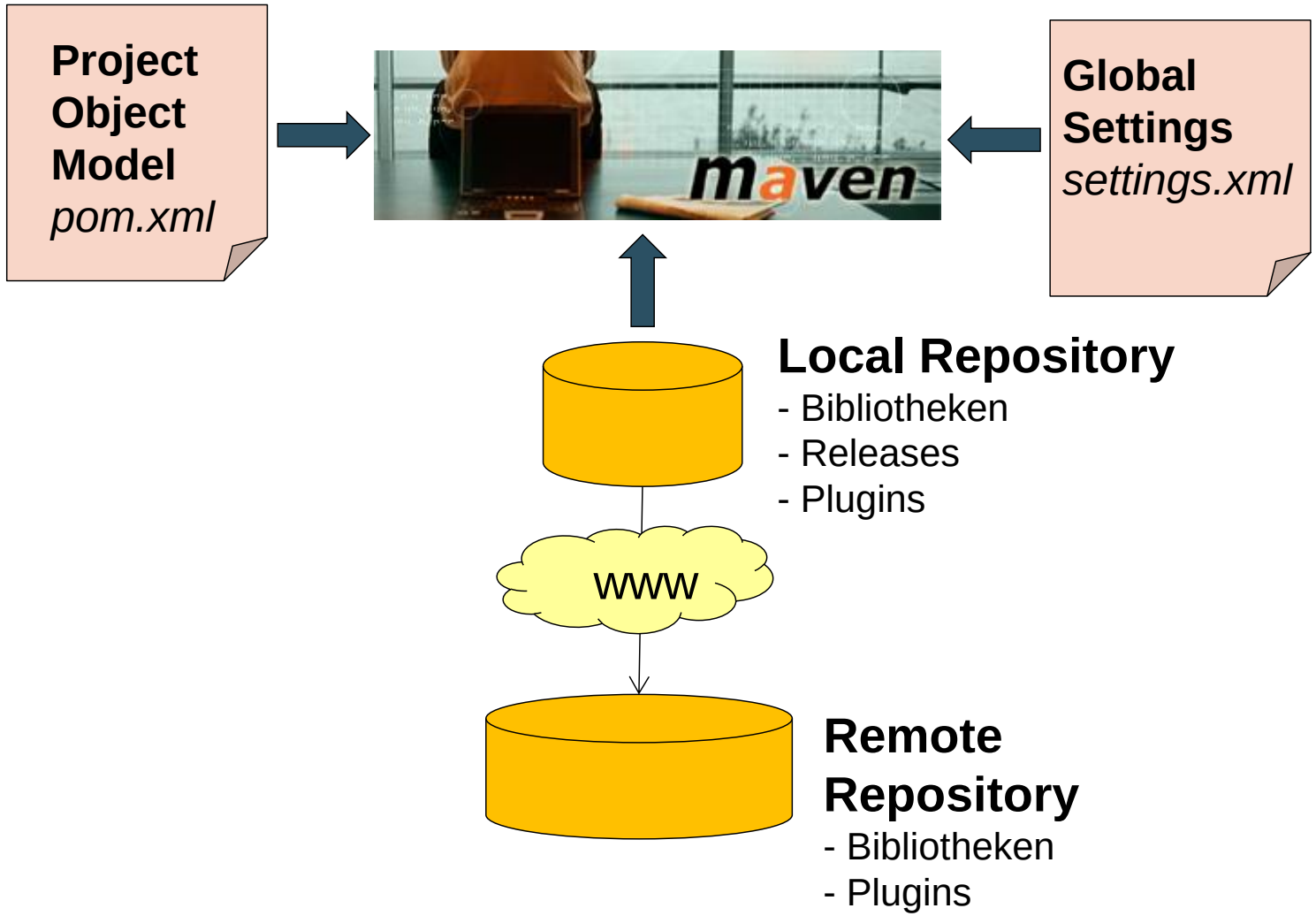
Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	Invalid ID supplied		
404	Pet not found		

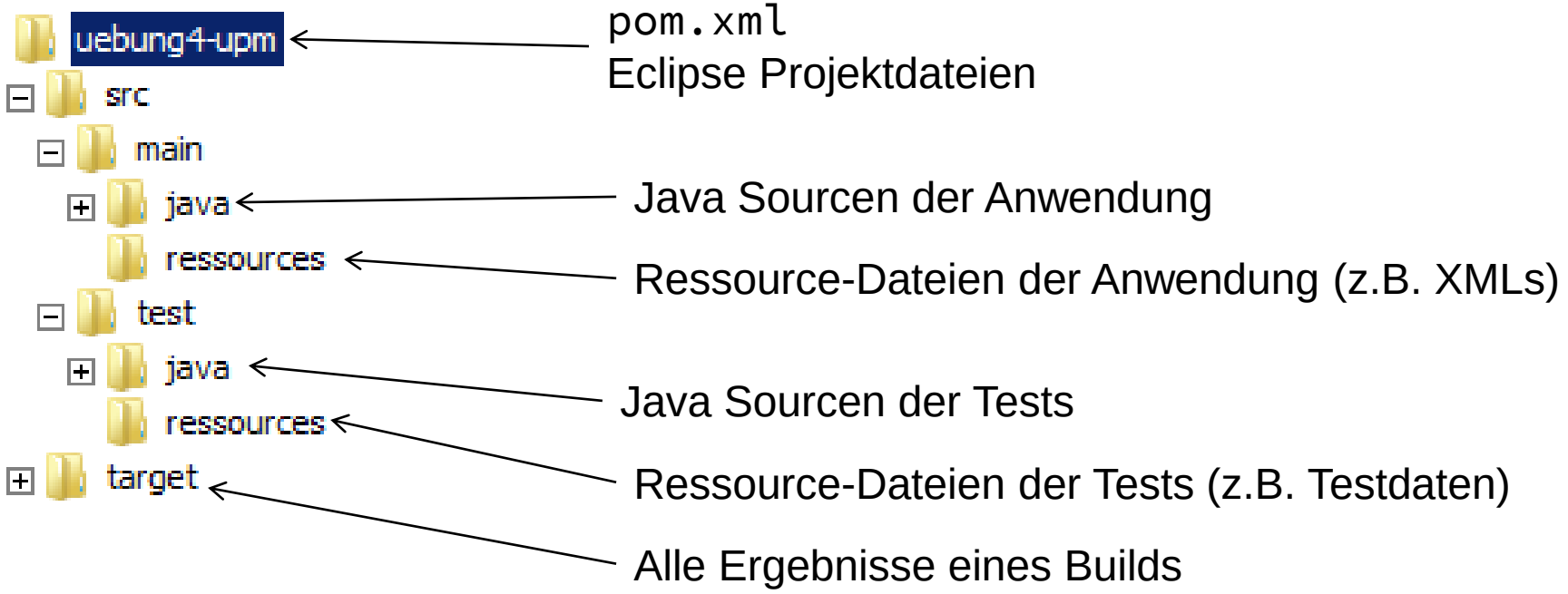
Try it out!

Entwicklungswerkzeuge: Maven

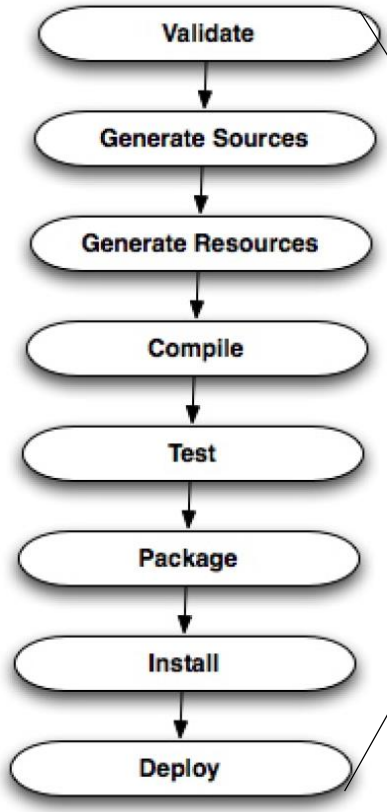
Maven: Übersicht



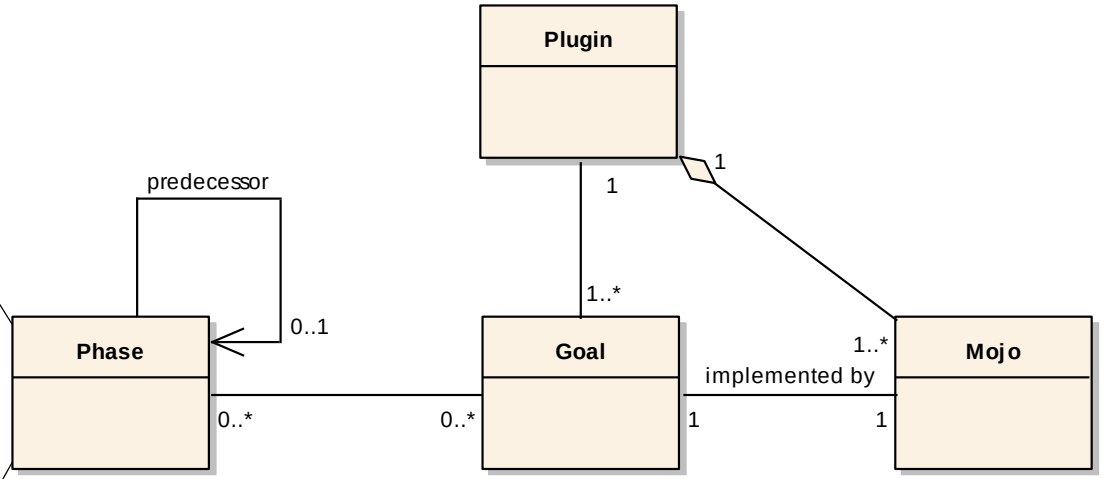
Die Maven Standard-Verzeichnisstruktur



Maven: Buildstruktur

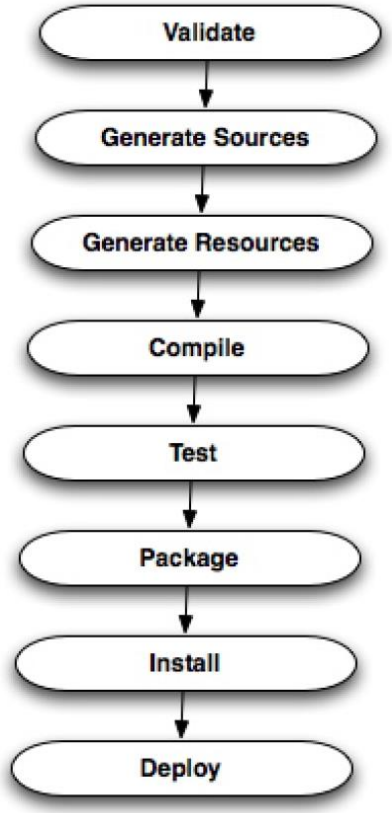


Build Lifecycle



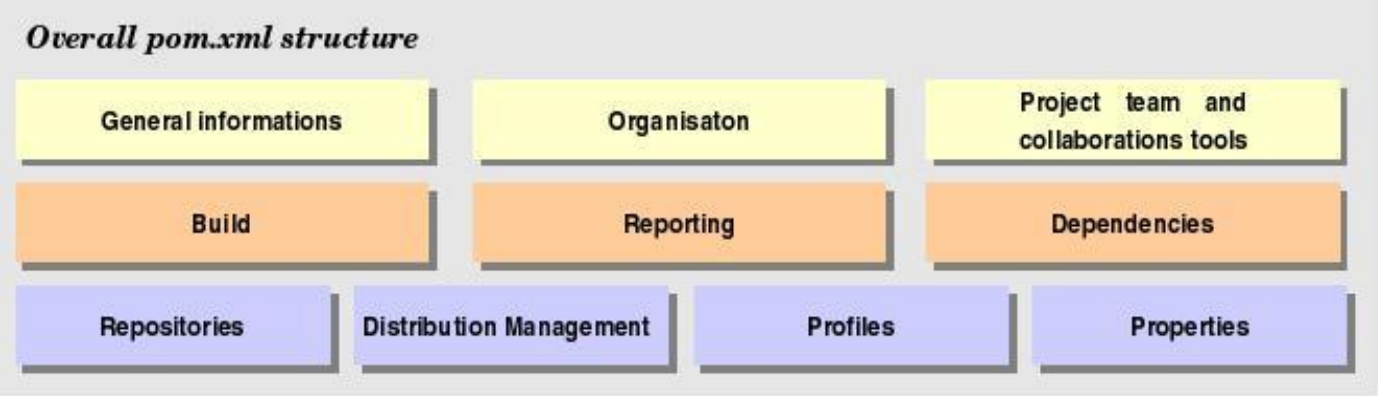
- Phase
  - Einzelne Buildphase.
  - Abarbeitung Build bis zu einer Phase:
    `mvn <phase>`
- Goal
  - Einzelne Buildaufgabe. (z.B. Testüberdeckung messen)
  - Direkter Aufruf eines einzelnen Goals:
    `mvn <plugin>:<goal>`

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. Diese können z.B. per http://mvnrepository.com recherchiert werden.

```
<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.