REST
Elasticsearch
Dropwizard

↳ tarent

REST

.. stands for 'Representational State Transfair'

- Strict usage of the HTTP protocol for data communication
- Two main concepts:

Resources: HTTP endpoints identified by an URL

HTTP Verbs, the essential ones are:

  - GET: Requests the representation of a resource
  - POST: Create a resource
  - PUT: Update a resource
  - DELETE: Deletes a resource

REST + JSON

.. stands for 'JavaScript object notation'

- Usage of the JavaScript syntax for data encoding

```
{
  "name" : "Sebastian",
  "age" : 35,
  "children" : [
     "Felix",
     "Nils",
     "Emil",
     "Linus"
  ]
}
```

REST + JSON (raw example)

```
curl -v -X POST --data @example.json
    -H "Content-type: application/json" http://127.0.0.1/
```

HTTP-Request:

```
POST /hello HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 94
Content-Type: application/json

{"name": "Sebastian",    ... }
```

HTTP-Reponse:

```
HTTP/1.1 200 OK
Date: Mon, 04 May 2015 13:33:16 GMT
Content-Type: text/plain
Content-Length: 12

Hello Sebastian
```

REST + JAX-RS

JAX-RS ist a Java based standard for REST on the server

Jersey is the reference implementation

```
@Path("/hello")
public class ExampleResource{

    @POST
    @Consumes("application/json")
    public String hello(Person person) {
        return "Hello "+ person.getName();
    }
}
```

REST + Java Client

Many Java REST clients out there.

My favorite: Spring RestTemplate

```
Person person =  new Person();
person.setName("Sebastian");

RestTemplate restTemplate = new RestTemplate();
String response restTemplate
     .postForObject("http://127.0.0.1:8080/hello", person, String.class);

assertThat(response).isEqualTo("Hello Sebastian");
```

NoSQL

NoSQL is a class of persistance systems which take a different approach than the relational model.

The main categories are:

- ↪ Key value stores
  e.g. redis
- ↪ Document oriented databases
  e.g. Couchdb, Elasticsearch
- ↪ Graph databases
  e.g. neo4j

NoSQL – Why?

Relational database are fine,

So why to we need other concepts?

Some reasons:

- Impedance missmatch:

  The relational normalisation is often in contrast to the use cases.

- Performance and scalability
- Special query types:

  e.g. graph queries, fulltext search

For web projects: Document based access makes many usecases very easy.

Elasticsearch

- Document oriented (document == JSON object)
- Slightly schema based
- HTTP REST interface and Java interface
- Focus on storing and searching within the database
- Based on the lucene search engine

    Every document is stored

    - As original JSON document
    - within the lucene search index
- Implemented in java (easy embeddable)
- Highly distributed
- Immutable data
    - every change produces a new revision

Elasticsearch – REST Api

Store or update a document:

```
curl -X POST --data @example.json
http://127.0.0.1:9200/persons/person/seb
```

```
{"_index":"persons",
  "_type":"person",
  "_id":"seb",
  "_version":1,
  "created":true
}
```

Elasticsearch – REST Api

Request a document by id:

```
curl http://127.0.0.1:9200/persons/person/seb
{
  "_source" : {
    "name" : "Sebastian",
    "age" : 35,
    "children" : [
      "Felix", "Nils", "Emil", "Linus"
    ]
  },
  "_version" : 1,
  "_index" : "persons",
  "found" : true,
  "_type" : "person",
  "_id" : "seb"
}
```

Elasticsearch – REST Api

List all documents from index and type:

```
curl http://127.0.0.1:9200/persons/person/_search
```

Search by search term:

```
curl http://127.0.0.1:9200/persons/person/_search
        ?q=name:sebastian
```

Fuzzy search:

```
curl 'http://127.0.0.1:9200/persons/person/_search
        ?q=name:seastian~'
```

Elasticsearch – REST Api

Delete a document by id:

```
curl -X DELETE http://127.0.0.1:9200/persons/person/seb
{
   "_type" : "person",
   "_version" : 2,
   "_index" : "persons",
   "_id" : "sebastian",
   "found" : true
}
```

Elasticsearch – REST Api

Delete the whole index:

```
curl -X DELETE http://127.0.0.1:9200/persons
{
   "acknowledged" : true
}
```

Dropwizard

- Java-Framework for RESTful Web Services
- Aggregation of stable standard libraries:
  - Jetty for HTTP
  - Jersey for REST
  - Jackson for JSON
  - Metrics for Health metrics
  - Hibernate for persistance
  - Logging
  - Simple configuration

Dropwizard features

- Micro-Services container
- Packages the app as single (fat) jar
- Startup in a few seconds
- Has everything for reliable operations
  - Shutdown with unix signals
  - Logging to stdout by default
  - SSL support
  - Management port
  - Simple security framework

Rapid development

- Create a new gradle (or maven) project
- Add the dropwizard dependency
- Configure the maven shade plugin for fat jar creation
- Create a config file
- Create the service main class
  - Add REST resources
  - Add health checks
- Start the service with

```
java -jar build/libs/dropwizard-example.jar server config.yaml
```

Configuration

- One central configuration file for everything
  - Jetty
    - SSL
  - Database
  - Logging
- Custom configurations
  - Simple mapping by annotations

Healthchecks

- Healhchecks and metrics information can be called on a seperate port
- Example Healthcheck:

```
public class Health extends HealthCheck {

    @Override
    protected Result check() {
        return Result.healthy("I'm ok ..");
        //return Result.unhealthy("there is a failure");
    }

}
```

Exercise

1. Install Elasticsearch and do some REST calls
2. Build the demo project
3. Use RestTemplate to do REST calls
4. Build and start dropwizard
5. Implement a HealthCheck within the service

   (test if the Elasticsearch is available)
6. Create a resource for managing books in a library

   (create, search, delete)

Thank you!