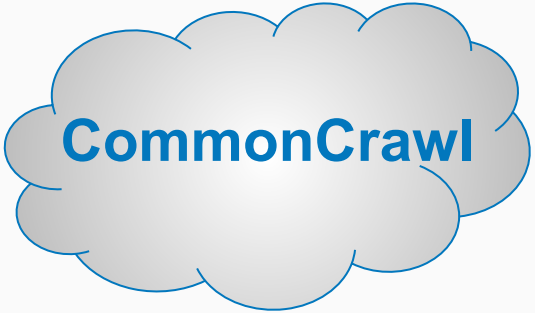
Group 2: RESTful APIs

TUM JS Technology Seminar 2017

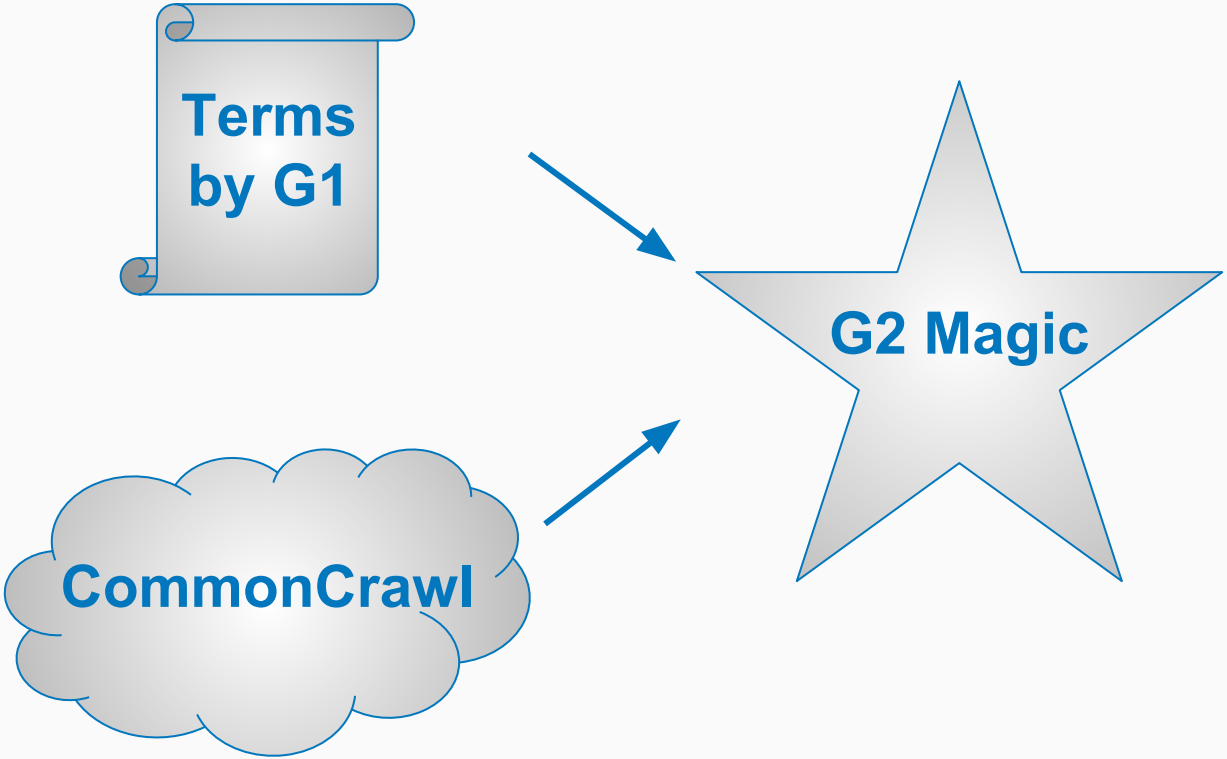
Felix, Lukas, Anshul, Nikita

Unstructured Data
Overview

Unstructured Data - Basic Overview

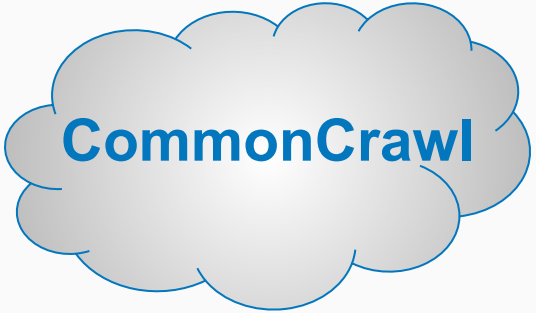


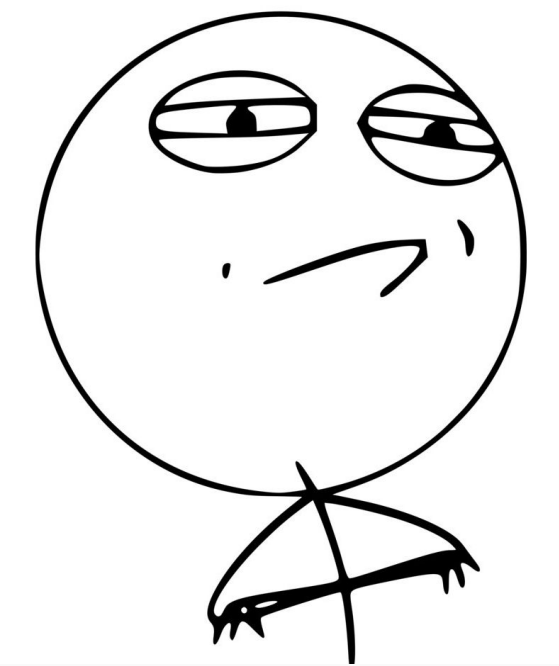
Unstructured Data - Basic Overview



Unstructured Data - Basic Overview

CommonCrawl in Details

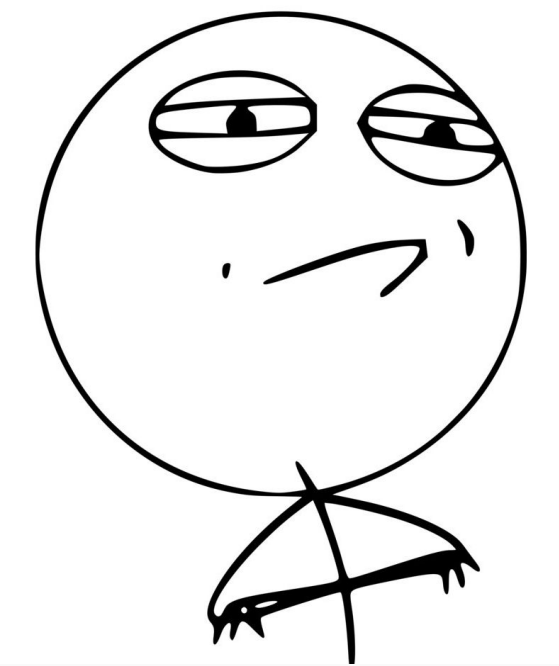




CommonCrawl in Details

WET files (no <tags>)

- 57800 files x 5000000+ lines
- Compressed: 9 TB
- Just for the latest crawl
- 33 crawls in total



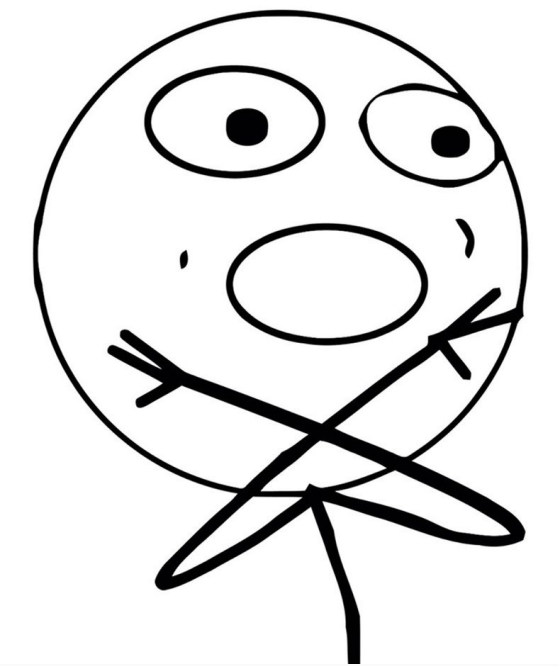
CommonCrawl in Details

WET files (no <tags>)

- 57800 files x 5000000+ lines
- Compressed: 9 TB
- Just for the latest crawl
- 33 crawls in total

WARC files (with <tags>)

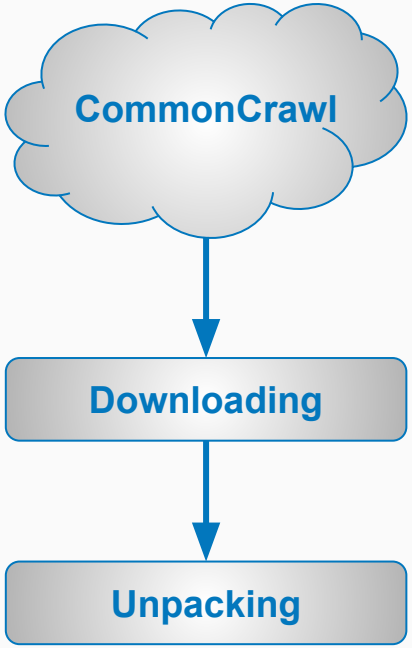
- Compressed: 54 TB
- We **don't** use them



Group 2 in Details

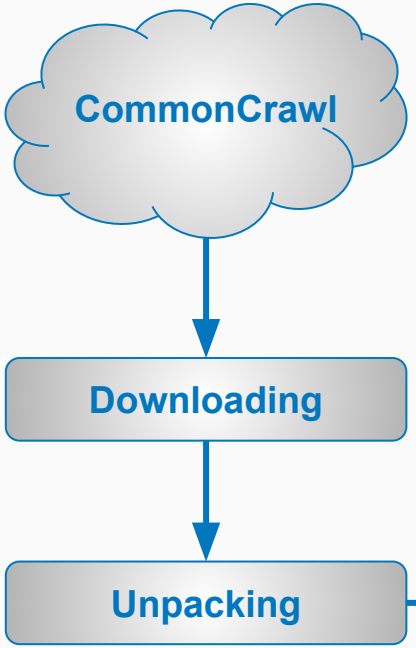
Group 2 in Details

Preprocessing

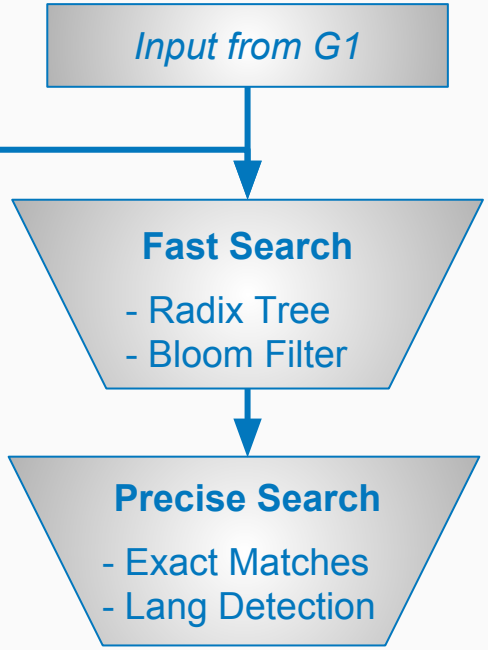


Group 2 in Details

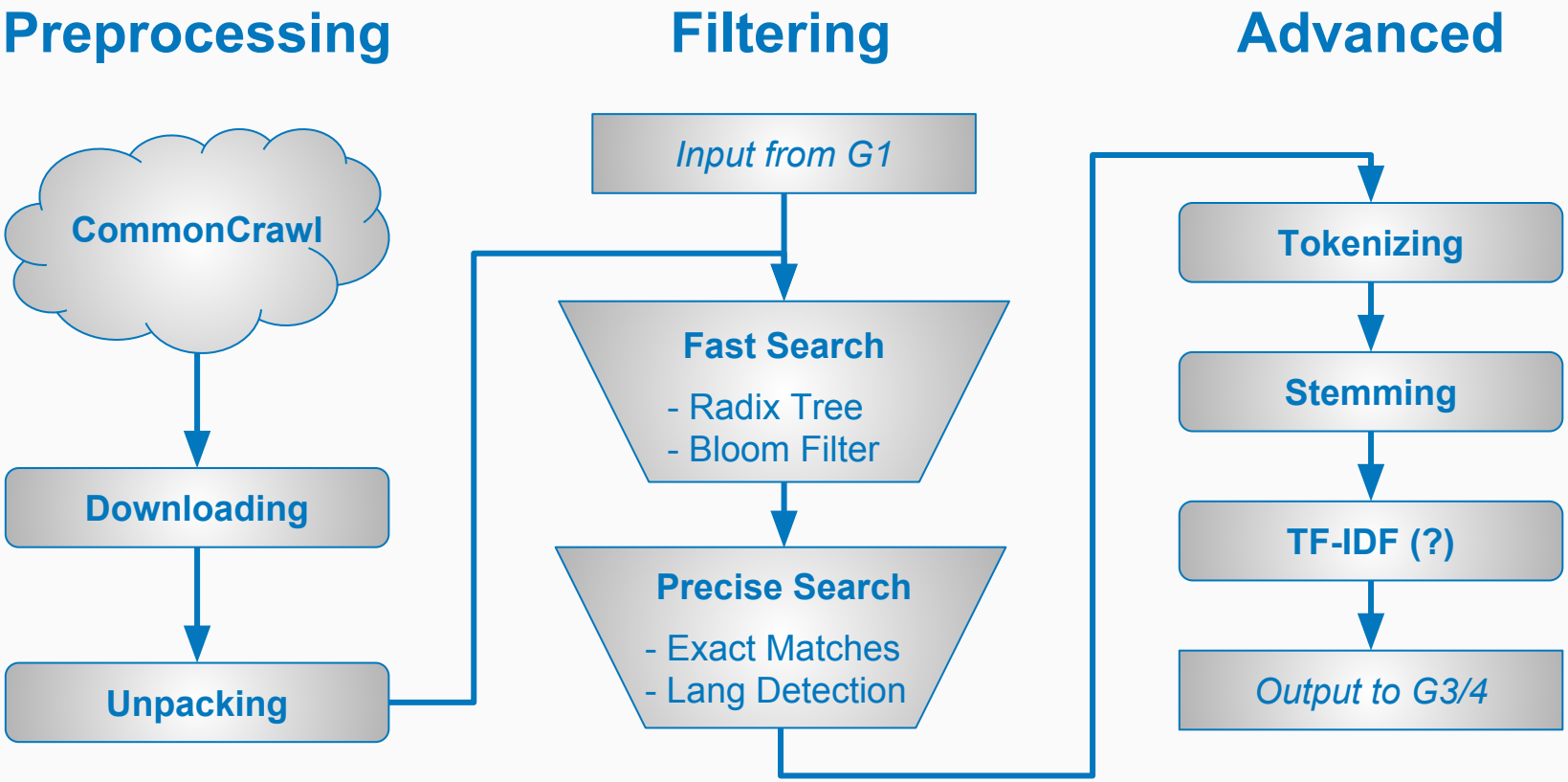
Preprocessing



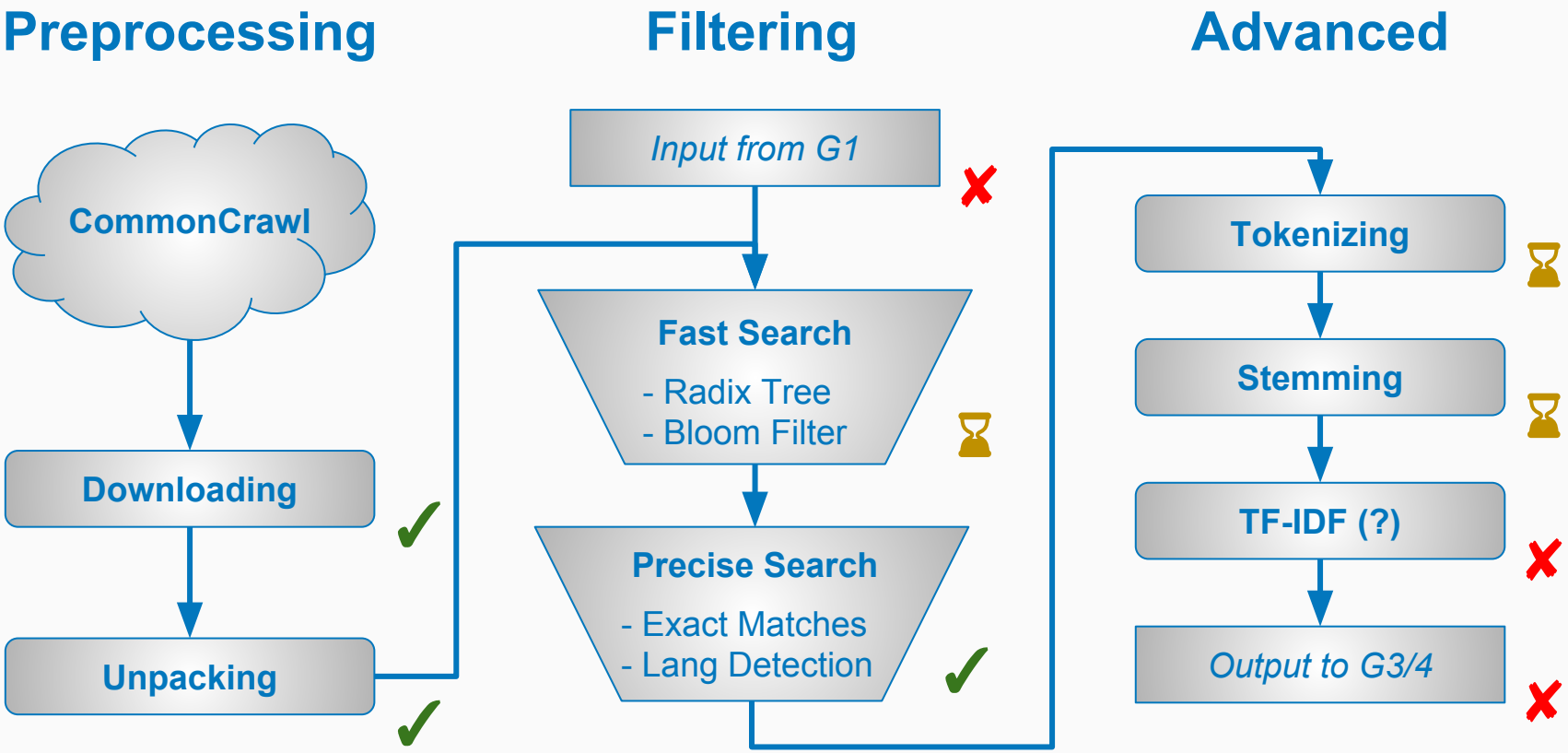
Filtering



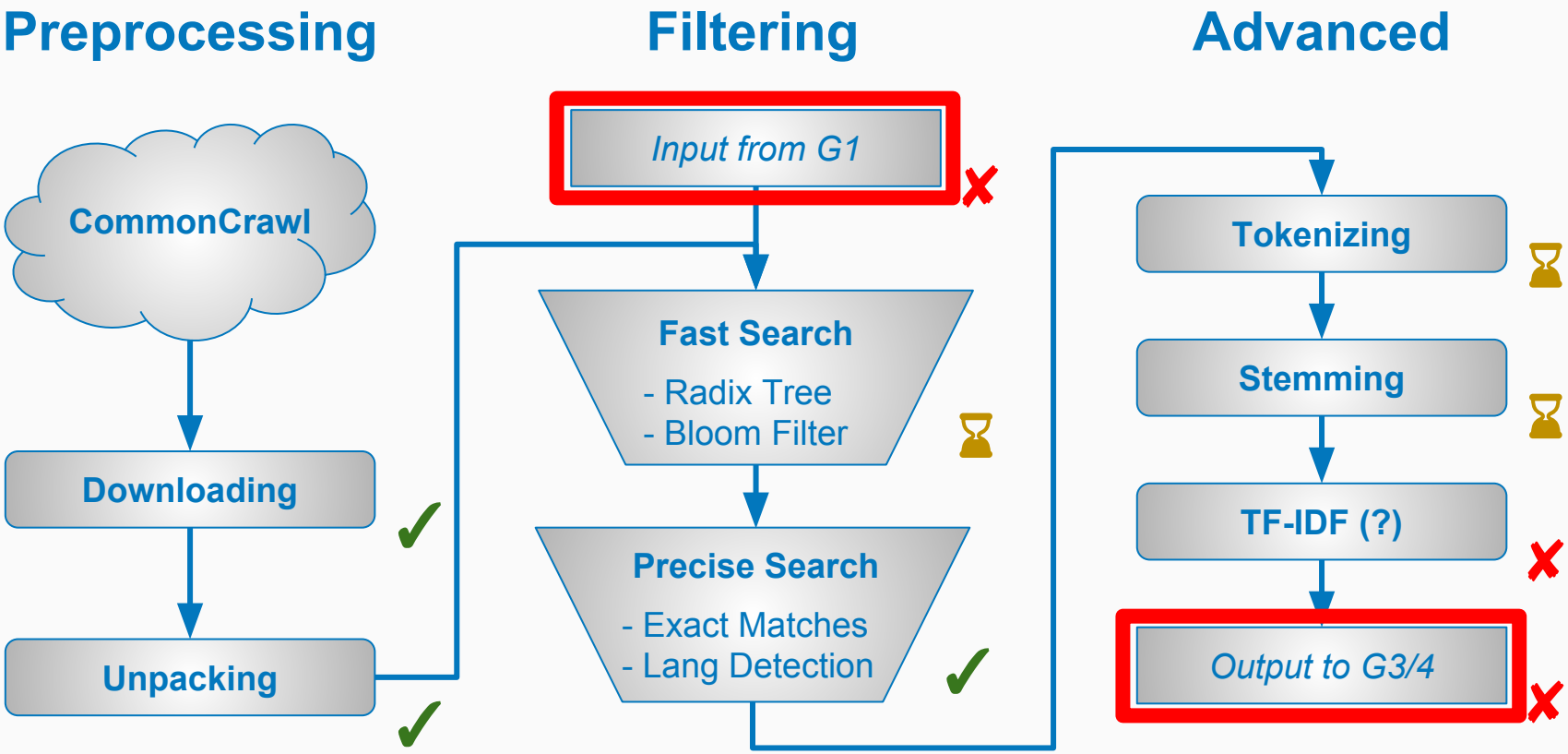
Group 2 in Details



Group 2 in Details

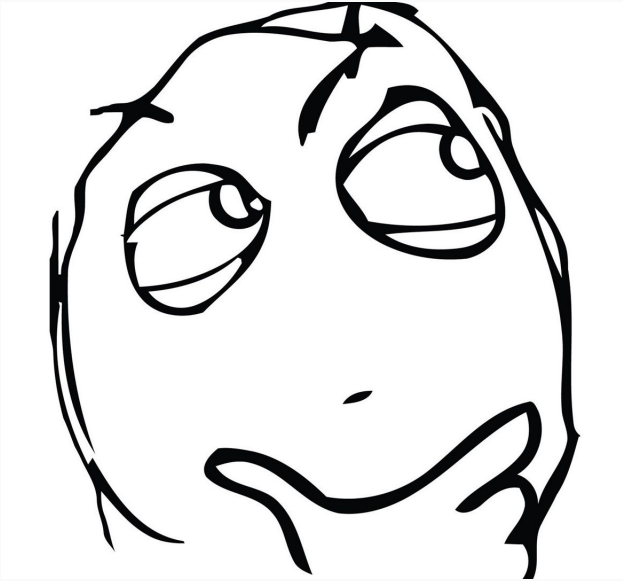


Group 2 in Details



To API or not to API

The Question



Why/What?

How can six groups exchange data?

What do we need to know?

Advantages? Disadvantages? Performance?

Let’s take a look at REST APIs!

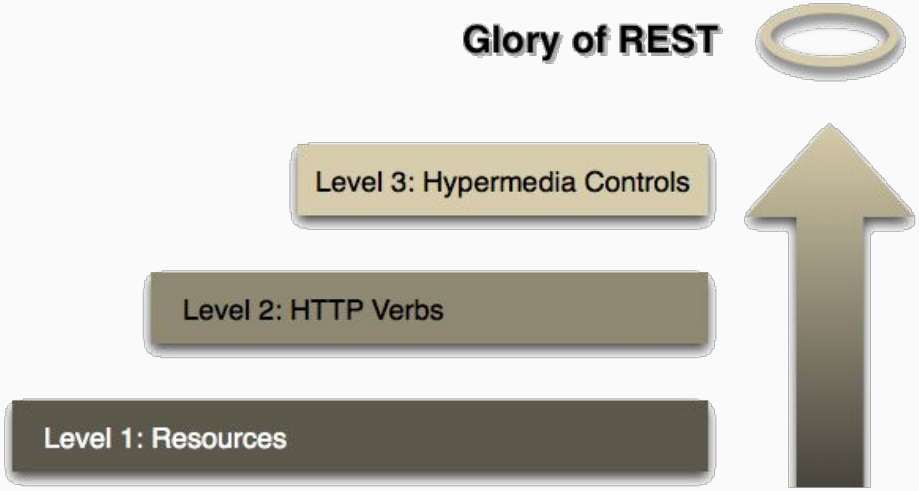
REST
Definition & Main Ideas

Overview

- REST: Representational State Transfer
- Mainly used for Machine to Machine communication
- Always uses HTTP/HTTPS
- Revolves around resources that can be accessed using standard HTTP methods
- Resources can have different representations: Text, JSON, XML, anything is possible (even images). Today the most commonly used format is JSON

Richardson Maturity Model

The Richardson Maturity Model defines a simple layered model to explain REST principles. It breaks down the REST approach into three steps



Source: martinfowler.com

Level 1 - Resources

- Requests address resources
- Resources are identified by their URIs

There are different ways to access resources:

- Collections:
  e.g. /musicians ➡ collection of musicians
- Single resources:
  e.g. /musicians/5 ➡ musician with id value 5

Level 2 - HTTP Verbs

HTTP method defines which actions should be undertaken on a resource

- GET: Retrieve information about a resource
  ➡ Should not have any side effects
- POST: Create a new resource
- PUT: Update/Replace a resource
- DELETE: Guess what? Delete a resource

Level 3 - Hypermedia Controls (HATEOAS)

- Responses contain information on how to proceed
- Common practice:
  Using attributes that contain type of action and URI for the action
- Client no longer needs to know the entire URI scheme beforehand
- URI scheme may be changed without breaking the clients

Architectural Constraints

- Client-Server
- Stateless
- Cacheable
- Uniform Interfaces
- Layered System

Example: simple REST APIs for musicians and their pieces

- Resources: musicians and pieces
- Methods: Adding, Updating, Retrieving and Deleting
- Possible URI scheme:
  - /musicians: collection of all musicians
  - /musicians/:id: single musician
  - /musicians/:id/pieces: collection of musicians pieces
  - /musicians/:id/pieces/:pieceId: single piece of the musician

Frameworks
Overview

restify

express

Express vs. Restify

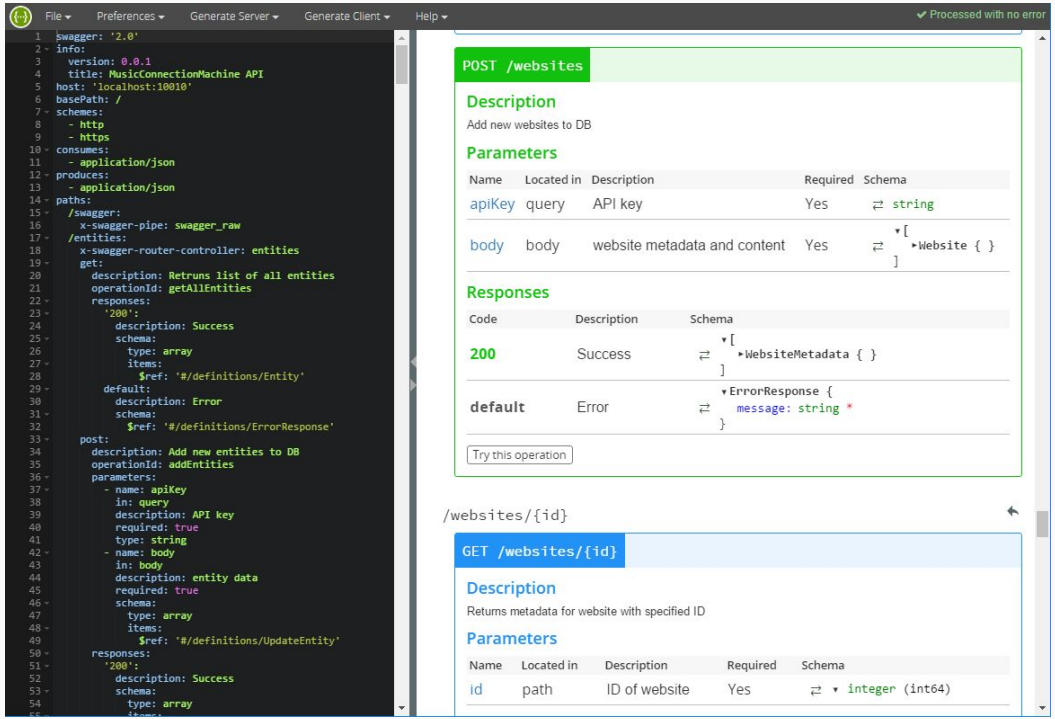
- Full blown server framework vs. purpose built for RESTful APIs
- Easy to get going
- Gets really hard to work with the bigger your project gets
- Only recommended for smaller projects

Swagger 2.0

- NPM package version 0.7.5
- Framework for documenting and implementing RESTful APIs
- Centered around swagger.yaml configuration file …
- … which can get really long real quick
- Swagger Node can be used on top of a lot of frameworks
  - Including Express, Restify, Hapi, …
- Not restricted to Node.js however
  - Support for Spring, Rails5, Python Flask, Go Server, ...

Swagger editor

- Generates documentation on the fly
- Ability to generate server and client side code
- Validates code against Swagger spec
- Very generic error messages ¯_(ツ)_/¯



Getting started with Swagger Node

1. $ npm install -g swagger
2. $ swagger project create my-awesome-project-name

```
? Framework? (Use arrow keys)
> connect
  express
  hapi
  restify
  sails
```

3. $ swagger project edit
4. $ swagger project start

Swagger.yaml explained

```yaml
swagger: "2.0"
info:
  version: "0.0.1"
  title: Hello World App
host: localhost:10010
basePath: /
schemes:
  - http
  - https
consumes:
  - application/json
produces:
  - application/json
```

- Swagger version
- Project details
- Host
- Base path all defined paths are relative to
- Accepted request protocols
- Accepted request bodies
- Response bodies

Swagger.yaml explained

```yaml
paths:
  /hello:
    x-swagger-router-controller: hello_world
    get:
      description: Returns 'Hello' to the caller
      operationId: hello
      parameters:
        - name: name
          in: query
          description: Name of the person to whom to say hello
          required: false
          type: string
      responses:
        "200":
          description: Success
          schema:
            $ref: "#/definitions/HelloWorldResponse"
        default:
          description: Error
          schema:
            $ref: "#/definitions/ErrorResponse"
```

- Path
- Controller file
- HTTP method
- Controller function
- Accepted parameters
- Responses

Swagger.yaml explained

```yaml
definitions:
  HelloWorldResponse:
    required:
      - message
    properties:
      message:
        type: string
  ErrorResponse:
    required:
      - message
    properties:
      message:
        type: string
```

- Definitions to be referenced within the project with `$ref: "#/definitions/<ClassName>"`
- Frequently used request / response bodies

File structure and controllers

```
.
├── api
    ├── controllers
        ├── hello_world.js
        ├── ...
        └── ...
    ├── ...
    └── swagger
        └── swagger.yaml
└── ...
```

```
module.exports = {
  hello: hello
};
function hello(req, res) {
  var name = req.swagger.params.name.value || 'stranger';
  var hello = 'Hello, ' + name + '!';
  res.json(hello);
}
```

Versioning in Rest APIs

Is “Versioning” RESTful ?

Roy Fielding in an presentation in August 2013

REST

What is
the best practice for
versioning
a REST API?

DON'T

Versioning an interface
is just a "polite" way
to kill deployed applications

Why do we need it?

How to version a REST API

Using the URI

- **.../YourAPIhere/version/2**
- **.../YourAPIhere/v2/**
- **.../YourAPIhere/version/2/Resource/version/3**

Pros:

- Visibility !
- Developer friendly
- Versioning specific Resources

Cons:

- URI should represent only the resource in a REST API.
- New versions are not compatible with existing implementation

URI Parameter

.../YourAPIHere/Resource?version=1

Pros:

- If no version is specified, you can provide the latest version of the resource.
- Doesn’t break existing URIs.

Cons:

- Parameters should specify the properties of a resource rather than its implementation

Accept Header

Accept: application/vnd.YourAPIHere.v2 + json

Pros:

- No new URI routing rules

Cons:

- Testing isn’t easy
- Hidden

Custom Request Header

YourAPIHere_Version : 2

Cons :

- In some cases older routers may reject a request with a non standard header. In this case debugging becomes very hard.
- Again is hidden like the accept header

Authentication/Authorization in Rest APIs

Methods for authentication on REST API

HTTP Authentication

- Basic
- OAuth 1.0
- OAuth 2.0

API Key Authentication

Basic Authentication

GET /Resource/Resource_ID HTTP/1.1

Host: Hostname.com/YourAPIHere

Authorization: Basic Base64.(UserName:Password)

Basic Authentication

Pros :

- Simple to use

Cons :

- Requires HTTPS
- Can be exploited with Man in the Middle attacks

OAuth 1.0

- Is usually used when a resource owner, needs to authorize a 3rd party (client) to access resources on its behalf stored on the server
- Protects the user (Resource owner) from disclosing the details to the 3rd party
- Server allows the client to access the resources after the resource owner has allowed the access, and grants it an Access_Token

OAuth 1.0 Header Example

```
GET /accounts/1234 HTTP/1.1
Host: Hostname.com
Authorization: OAuth realm="YourAPIHere",
    oauth_consumer_key="dpf43f3p2l4k3l03",
    oauth_signature_method="HMAC-SHA1",
    oauth_timestamp="137131200",
    oauth_nonce="wIjqoS",
    oauth_callback="YourLinkHere",
    oauth_signature="74KNZJeDHnMBp0EMJ9ZHt%2FXKycU%3D"
```

Reference : https://www.slideshare.net/stormpath/rest-api-security?from_action=save

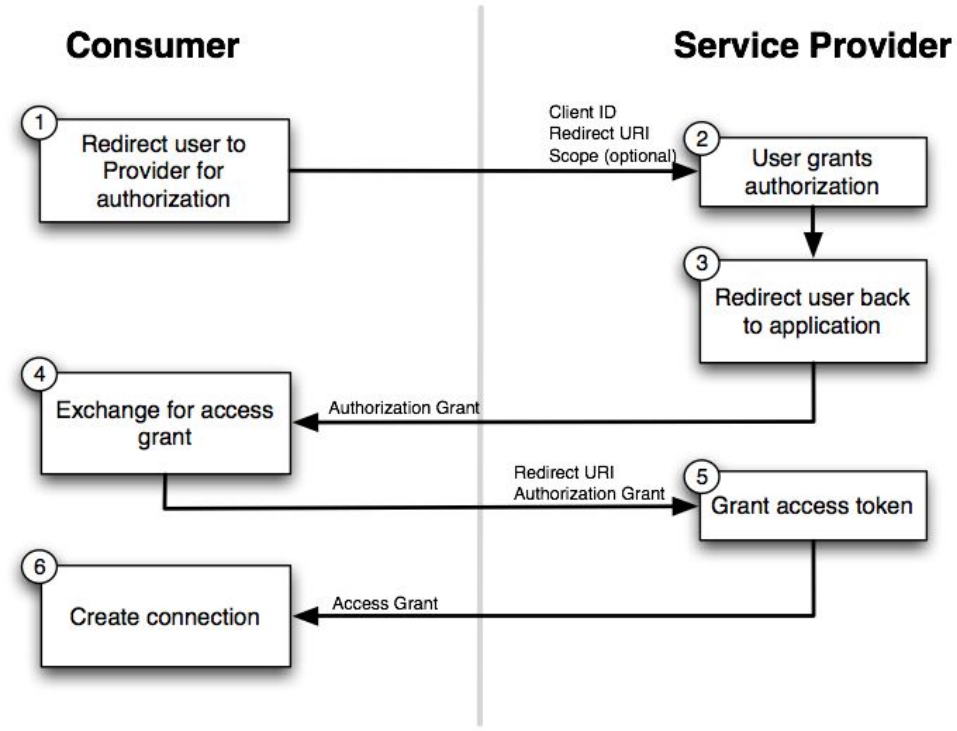
OAuth 2.0

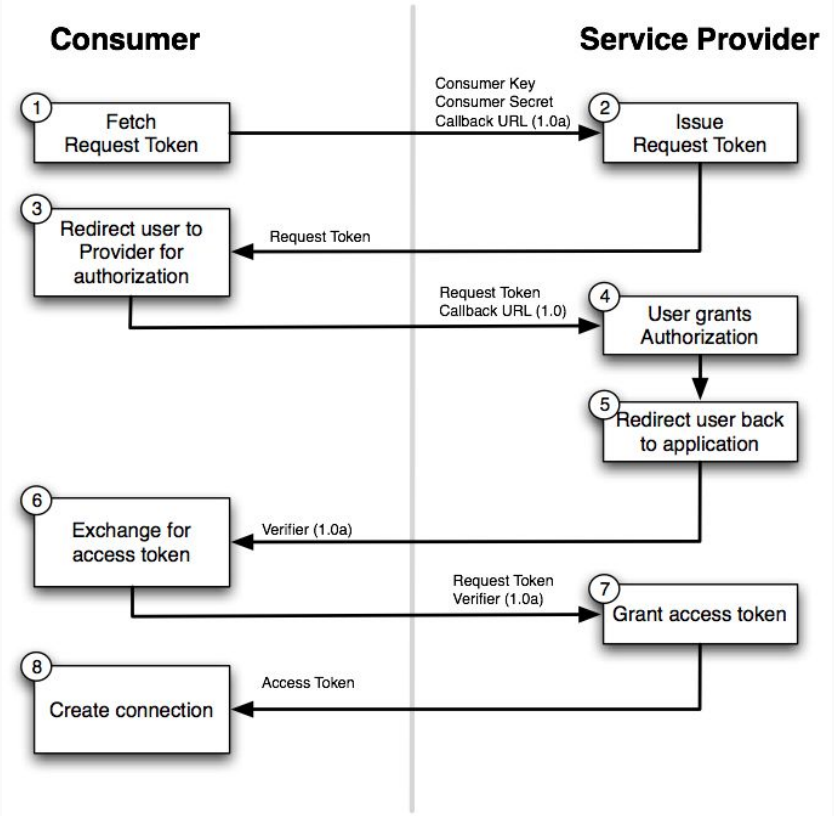
GET /Resource/Resource_ID HTTP/1.1

Host: Hostname.com/YourAPIHere

Authorization: Bearer Access_Token

OAuth 1.0 v/s OAuth 2





Reference: http://docs.spring.io/spring-social/docs/1.1.0.RELEASE/reference/htmlsingle/#section_oauth2ServiceProviders

API Keys

A string of generated characters, created to allow access to resources that were defined during it’s creation.

We can use API Keys instead of our traditional Username and Password for authentication.

API Keys and Why you should use them

- Entropy
- Security
- Independence
- Speed

*Reference: https://stormpath.com/blog/top-six-reasons-use-api-keys-and-how

Q & A

Questions?

SPEAK NOW

OR

FOREVER HOLD YOUR PEACE

Hacking time
Workshop