
API Implementation

Sprint 3 - Week 5

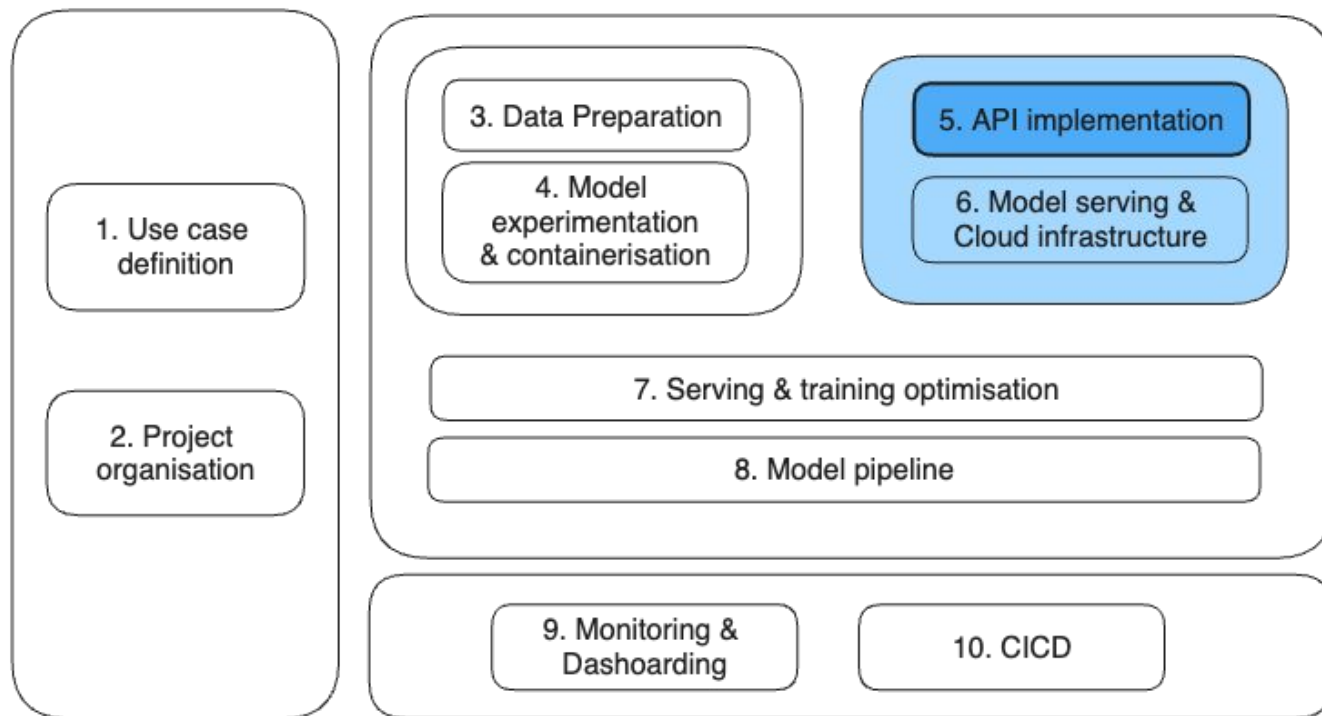
INFO 9023 - Machine Learning Systems Design

2024 H1

Thomas Vrancken (t.vrancken@uliege.be)

Matthias Pirlet (matthias.pirlet@uliege.be)

Status on our overall course roadmap



Agenda

What will we talk about today

Lecture (30min)

1. API
2. REST
3. RPC

Guest lecture (45min)

4. Connexion

Lab (45min)

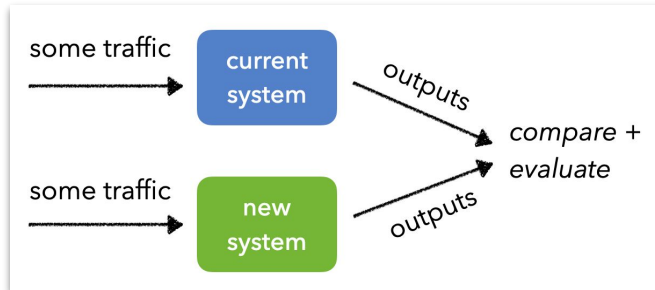
5. Flask



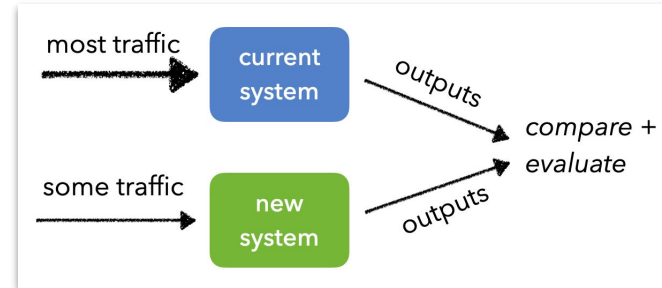
Recap last week

Online testing

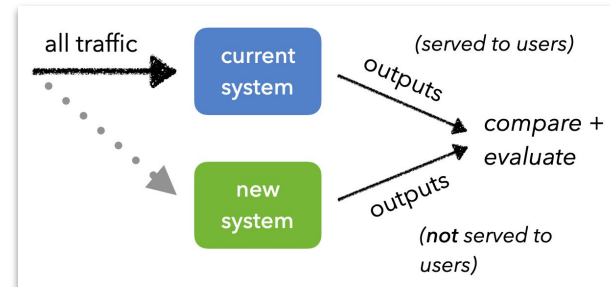
A / B testing



Canary testing



Shadow testing



Why do we need kubernetes?

Virtual Environment

Pros: remove complexity
Cons: does not isolate from OS

Virtual Machines

Pros: isolate OS guest from host
Cons: intensive use hardware

Containers

Pros: lightweight
Cons: issues with security, scalability, and control



How to manage microservices?

Goal: find effective ways to deploy our apps (more difficult than we might initially imagine) and to break down a complex application into smaller ones (i.e. microservices)

... Follow-up on VMs...

If you are sadly stuck with a Windows machine, fear no more!

*“Developers can access the power of both Windows and Linux at the same time on a Windows machine. The **Windows Subsystem for Linux (WSL)** lets developers install a Linux distribution (such as Ubuntu, OpenSUSE, Kali, Debian, Arch Linux, etc) and use Linux applications, utilities, and Bash command-line tools directly on Windows, unmodified, without the overhead of a traditional virtual machine or dualboot setup.”*

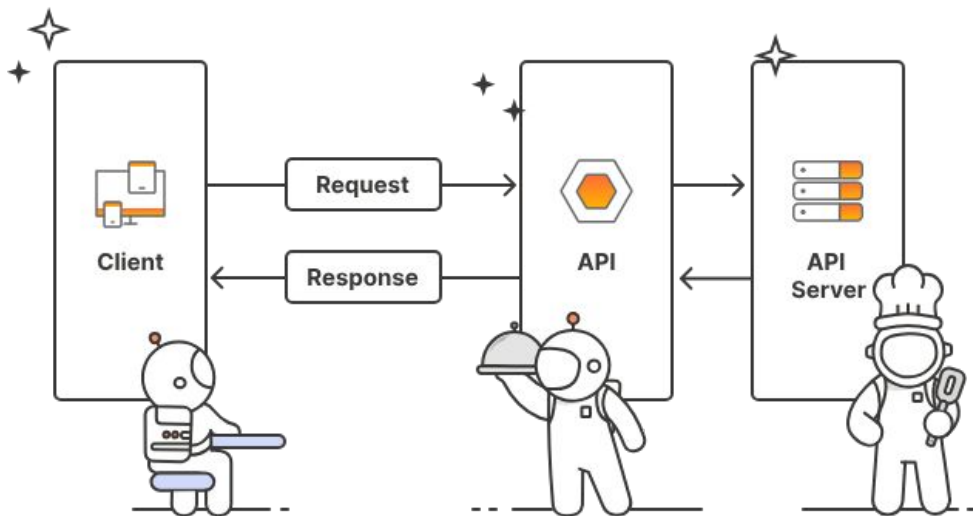
⇒ Let's you run Linux bash commands directly on your Windows OS





API

Application Programming Interface (API)



An **Application Programming Interface (API)** is a set of protocols that enable different software components to communicate and transfer data.

Developers use APIs to bridge the gaps between small, discrete chunks of code in order to create applications that are powerful, resilient, secure, and able to meet user needs.

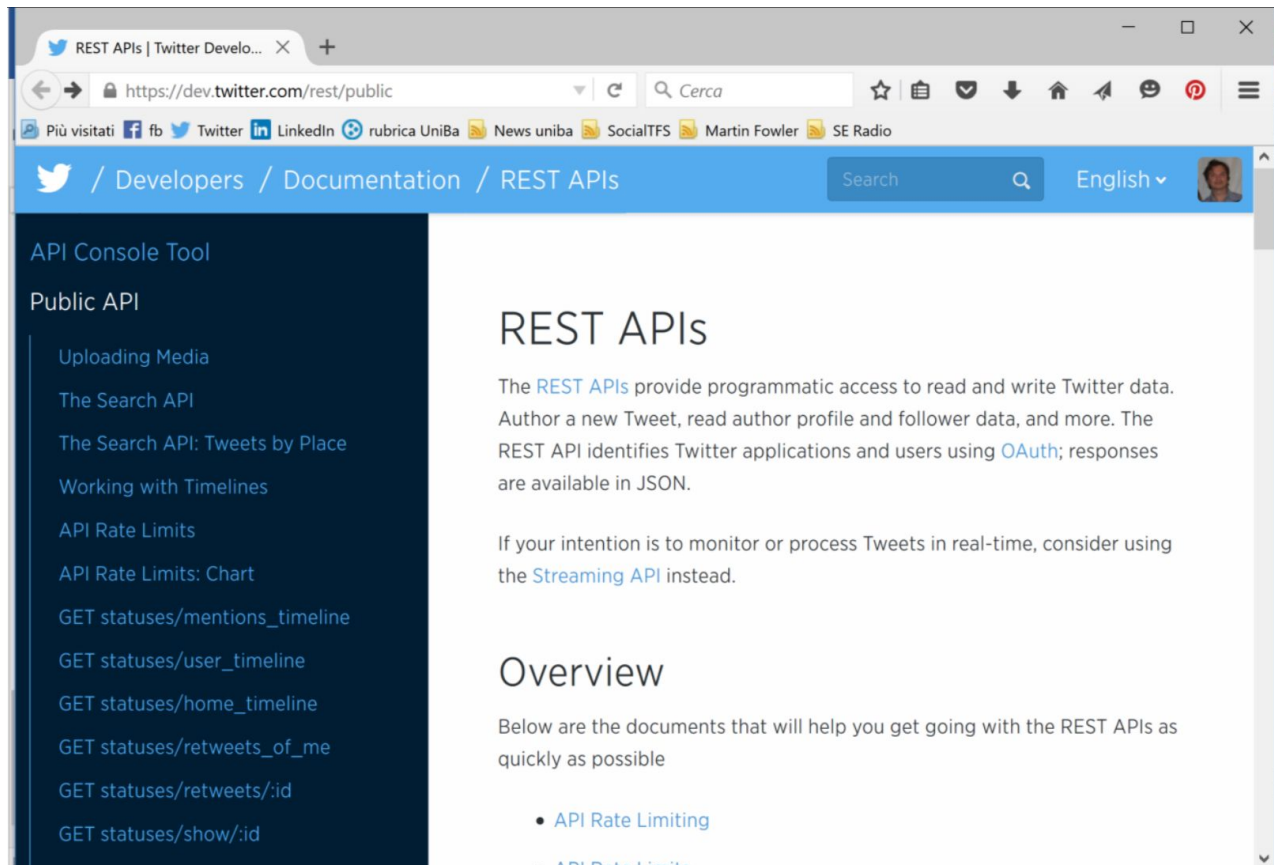
Client sends a **requests** and the API sends a **response**.

API

API EVERYWHERE

imgflip.com

APIs examples



The screenshot shows a web browser window with the URL `https://dev.twitter.com/rest/public`. The page is titled "REST APIs | Twitter Develo..." and features a blue header with navigation links: "/ Developers / Documentation / REST APIs". A search bar and a language dropdown set to "English" are also present. On the left, a dark blue sidebar lists various API endpoints under the heading "Public API", including "Uploading Media", "The Search API", "The Search API: Tweets by Place", "Working with Timelines", "API Rate Limits", "API Rate Limits: Chart", and several GET endpoints for timelines and retweets. The main content area is titled "REST APIs" and contains an introductory paragraph: "The [REST APIs](#) provide programmatic access to read and write Twitter data. Author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using [OAuth](#); responses are available in JSON." Below this, a section titled "Overview" states: "Below are the documents that will help you get going with the REST APIs as quickly as possible". A bulleted list follows, with the first item being "• [API Rate Limiting](#)".

REST APIs | Twitter Develo... X +

https://dev.twitter.com/rest/public

Più visitati fb Twitter LinkedIn rubrica UniBa News uniba SocialTFS Martin Fowler SE Radio

/ Developers / Documentation / REST APIs Search English

API Console Tool

Public API

- Uploading Media
- The Search API
- The Search API: Tweets by Place
- Working with Timelines
- API Rate Limits
- API Rate Limits: Chart
- GET statuses/mentions_timeline
- GET statuses/user_timeline
- GET statuses/home_timeline
- GET statuses/retweets_of_me
- GET statuses/retweets/:id
- GET statuses/show/:id

REST APIs

The [REST APIs](#) provide programmatic access to read and write Twitter data. Author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using [OAuth](#); responses are available in JSON.

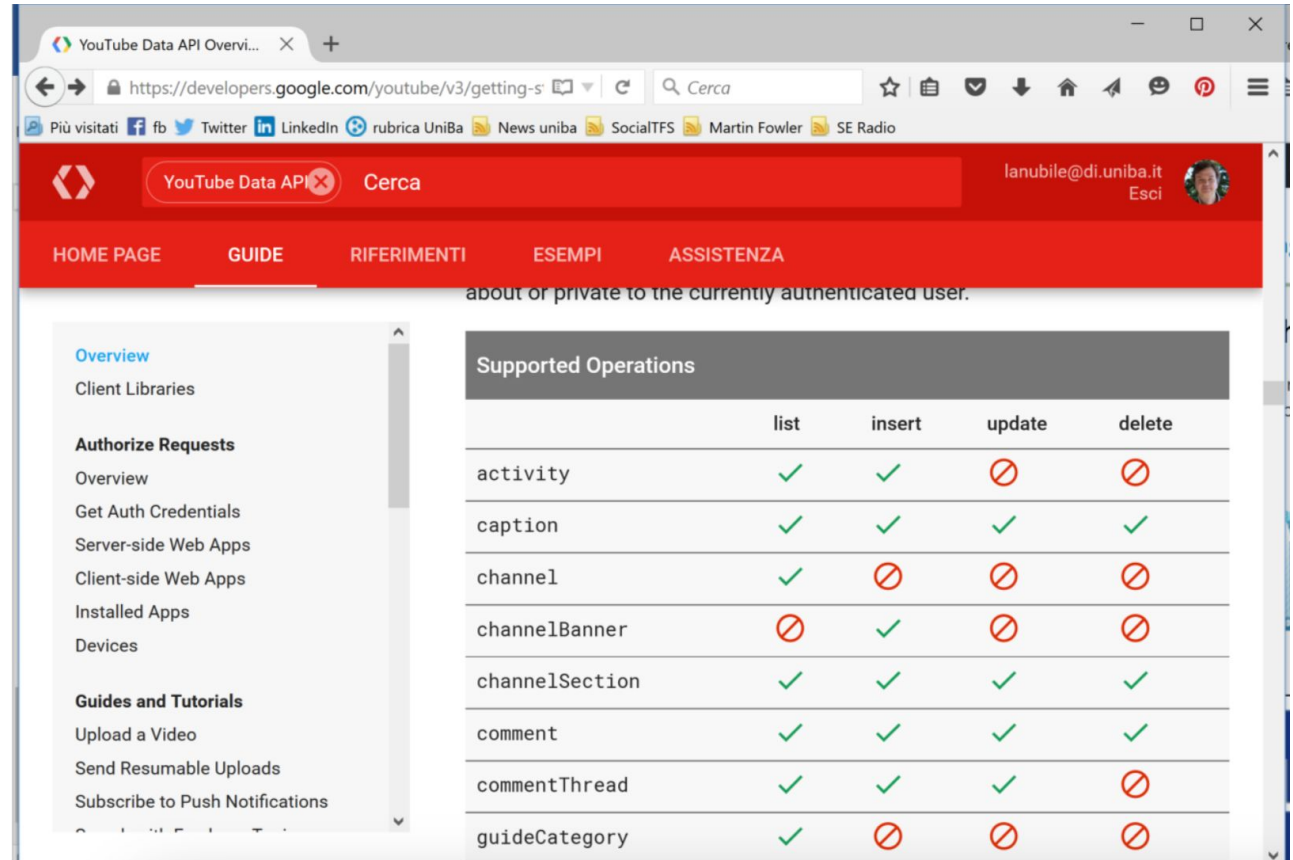
If your intention is to monitor or process Tweets in real-time, consider using the [Streaming API](#) instead.

Overview

Below are the documents that will help you get going with the REST APIs as quickly as possible

- [API Rate Limiting](#)

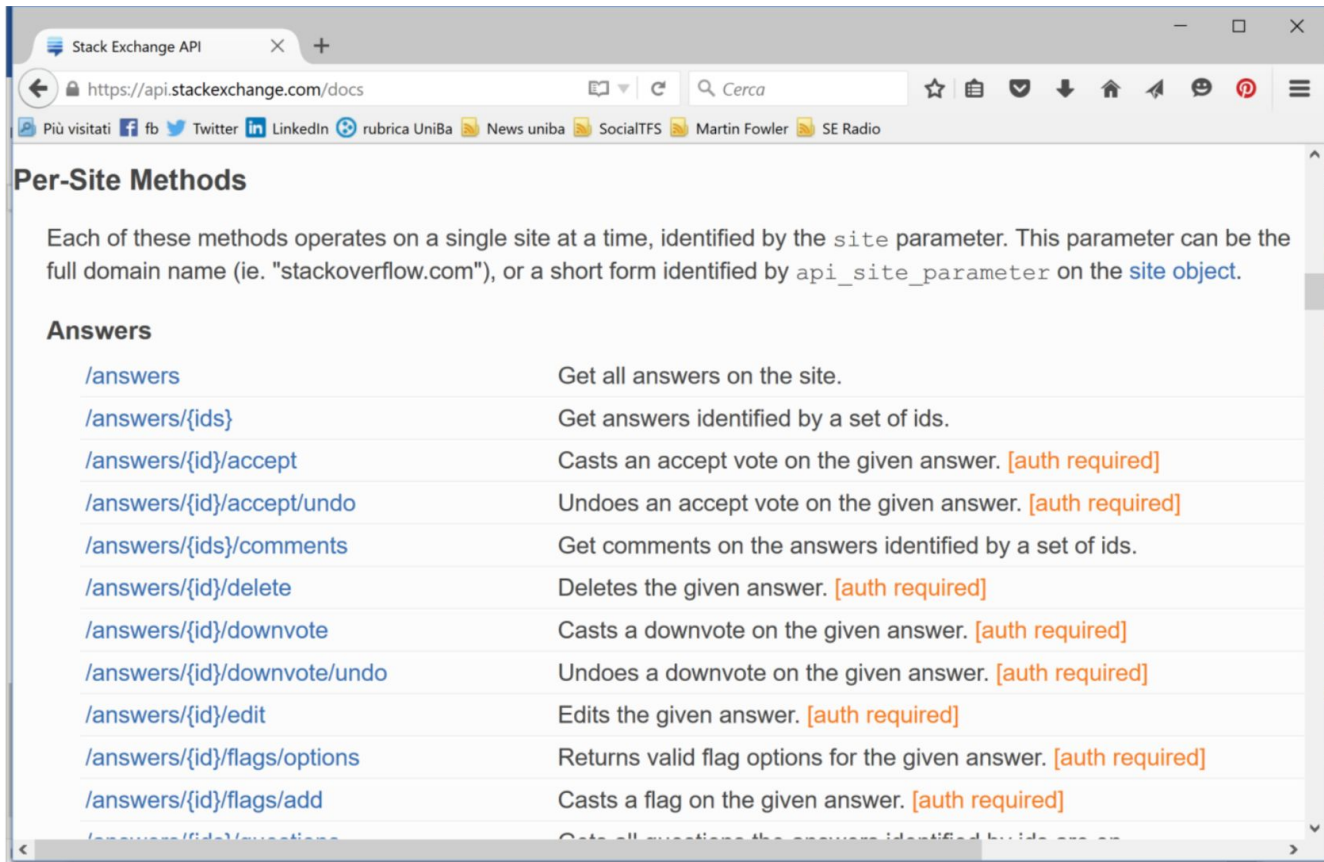
APIs examples



The screenshot shows the 'YouTube Data API Overview' page in a web browser. The page has a red header with the Google logo, a search bar, and a user profile for 'lanubile@di.uniba.it'. Below the header is a navigation bar with links: HOME PAGE, GUIDE (selected), RIFERIMENTI, ESEMPI, and ASSISTENZA. The main content area is divided into two columns. The left column contains a sidebar with links to Overview, Client Libraries, Authorize Requests, Guides and Tutorials, and other resources. The right column displays the 'Supported Operations' table, which lists various API endpoints and their supported HTTP methods (list, insert, update, delete).

	list	insert	update	delete
activity	✓	✓	✗	✗
caption	✓	✓	✓	✓
channel	✓	✗	✗	✗
channelBanner	✗	✓	✗	✗
channelSection	✓	✓	✓	✓
comment	✓	✓	✓	✓
commentThread	✓	✓	✓	✗
guideCategory	✓	✗	✗	✗

APIs examples



The screenshot shows a web browser window with the title "Stack Exchange API" and the URL "https://api.stackexchange.com/docs". The page content is titled "Per-Site Methods" and explains that each method operates on a single site, identified by the `site` parameter. Below this, a section titled "Answers" lists various API endpoints and their descriptions.

Per-Site Methods

Each of these methods operates on a single site at a time, identified by the `site` parameter. This parameter can be the full domain name (ie. "stackoverflow.com"), or a short form identified by `api_site_parameter` on the [site object](#).

Answers

/answers	Get all answers on the site.
/answers/{ids}	Get answers identified by a set of ids.
/answers/{id}/accept	Casts an accept vote on the given answer. [auth required]
/answers/{id}/accept/undo	Undoes an accept vote on the given answer. [auth required]
/answers/{ids}/comments	Get comments on the answers identified by a set of ids.
/answers/{id}/delete	Deletes the given answer. [auth required]
/answers/{id}/downvote	Casts a downvote on the given answer. [auth required]
/answers/{id}/downvote/undo	Undoes a downvote on the given answer. [auth required]
/answers/{id}/edit	Edits the given answer. [auth required]
/answers/{id}/flags/options	Returns valid flag options for the given answer. [auth required]
/answers/{id}/flags/add	Casts a flag on the given answer. [auth required]
/answers/{ids}/questions	Get all questions the answers identified by ids are on

APIs can be accessed differently



Private APIs

- ✦ Used to connect different software components within a single organization
- ✦ Not available for third-party use
- ✦ Some applications may include dozens or even hundreds of private APIs



Public APIs

- ✦ Provide public access to an organization's data, functionality, or services
- ✦ Can be integrated into third-party applications
- ✦ Some public APIs are available for free, while others are offered as billable products



Partner APIs

- ✦ Enable two or more companies to share data or functionality in order to collaborate
- ✦ Not available to the general public
- ✦ Leverage authentication mechanisms to restrict access

Main types of APIs

- **SOAP** (Simple Object Access Protocol). Client and server exchange messages using XML. This is a less flexible API that was more popular in the past.
- **Websocket**: Websocket API is another modern web API development that uses JSON objects to pass data. A WebSocket API supports two-way communication between client apps and the server. The server can send callback messages to connected clients, which is not possible in REST API.
- **REST** (Representational State Transfer): Widely used today. The client sends requests to the server as data. The server uses this client input to start internal functions and returns output data back to the client.
- **RPC** (Remote Procedure Calls). The client completes a function (aka method or procedure) on the server, and the server sends the output back to the client. “Action centric”, when a complex action needs to happen on the server side. Can be stateful, which allows you to interact with an object on the server side as if you could access it locally.

RPC focuses on **functions** or **actions**, while REST focuses on **resources** or **objects**.

Focus for today!



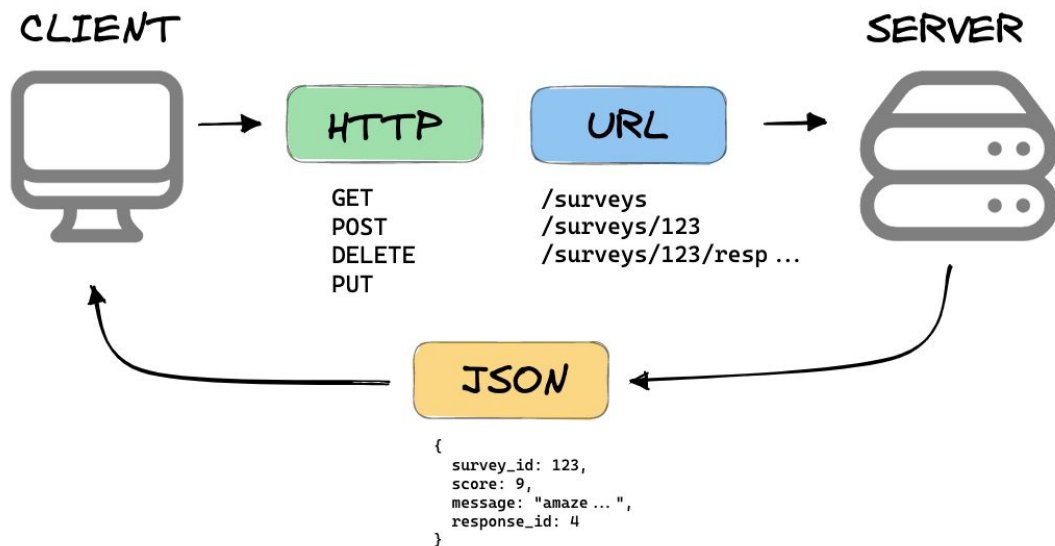
REST APIs

REST API

- **REpresentational State Transfer (REST)** API is an interface that two computer systems use to exchange information securely over the internet
- Architectural style for designing decentralized systems
 - Based on the original web architecture as a distributed hypermedia system
- Roy Fielding's PhD thesis: Architectural Styles and the Design of Network-based Software Architectures
 - <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- Central framework in IT. Mostly used to read and write data.

REpresentational State Transfer (REST) API

Roy Fielding's PhD thesis: [Architectural Styles and the Design of Network-based Software Architectures](https://mannhowie.com/rest-api)

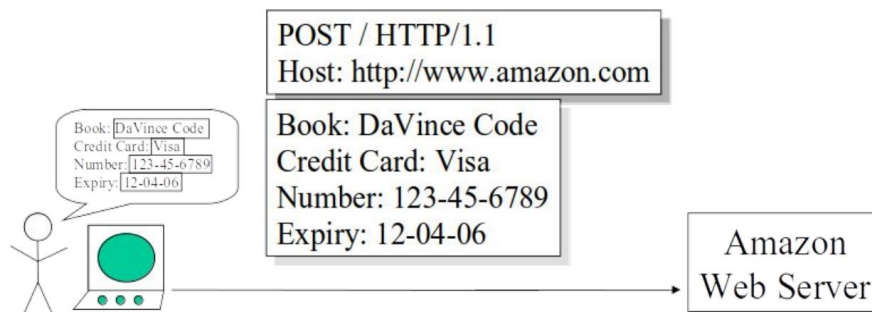


Retrieving Information (GET)



- User types into the browser: <http://www.amazon.com>
- The browser creates an HTTP request (no body)
- The HTTP request identifies:
 - The desired action: GET
 - The target machine (`www.amazon.com`)

Updating Information (POST)



- The user fills in a form on a webpage.
- The browser creates an HTTP request with a body containing form data.
- HTTP request identifies:
 - The action: POST ("here is some updated info")
 - The target machine (amazon.com)
- The body contains data being POSTed (form data)

REST API: HTTP requests

HTTP is a communication protocol for networked systems

⇒ How your computer accesses a webpage.

REST is based on **CRUD** operations: Able to **Create**, **Read**, **Update** and **Delete** resources

There are 4 basic **verb** commands when making a HTTP request

- **POST**: Create a new resource based on the payload given in the body of the request
- **GET**: Read data from a single or a list of multiple resources.
- **PUT/PATCH**: Update a specific resource (by ID)
- **DELETE**: Delete the given resource based on the id provided

Anatomy of an HTTP Request

VERB	URI	HTTP Version
Request Header		
Request Body		

- **Verb** (one of PUT, GET, POST, DELETE, ...)
- **Endpoint (URI)** Identifies the resource upon which the operation will be performed (e.g. `/users/123`)
- The **Request Header** contains metadata such as
 - Collection of key-value pairs of headers and their values.
 - Information about the **message** and its **sender** like client type, the formats client supports, format type of the message body, cache settings for the response, and more.
- **Request Body** is the actual message content (JSON, XML)

Anatomy of an HTTP Request

VERB	URI	HTTP Version
Request Header		
Request Body		

GET:

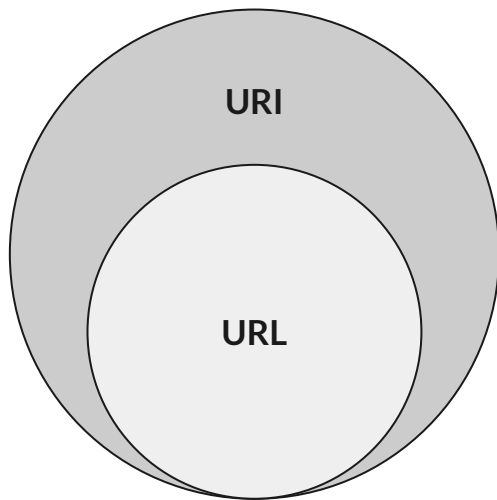
```
GET http://www.w3.org/Protocols/rfc2616/rfc2616.html HTTP/1.1
Host: www.w3.org, Accept: text/html,application/xhtml+xml,application/xml; ...
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 ...
Accept-Encoding: gzip,deflate,sdch, Accept-Language: en-US,en;q=0.8,hi;q=0.6
```

POST:

```
POST http://MyService/Person/ HTTP/1.1
Host: MyService, Content-Type: text/xml; charset=utf-8, Content-Length: 123
<?xml version="1.0" encoding="utf-8"?>
<Person><ID>1</ID><Name>M Vaqqas</Name>
<Email>m.vaqqas@gmail.com</Email><Country>India</Country></Person>
```

What's the difference between an URI and an URL

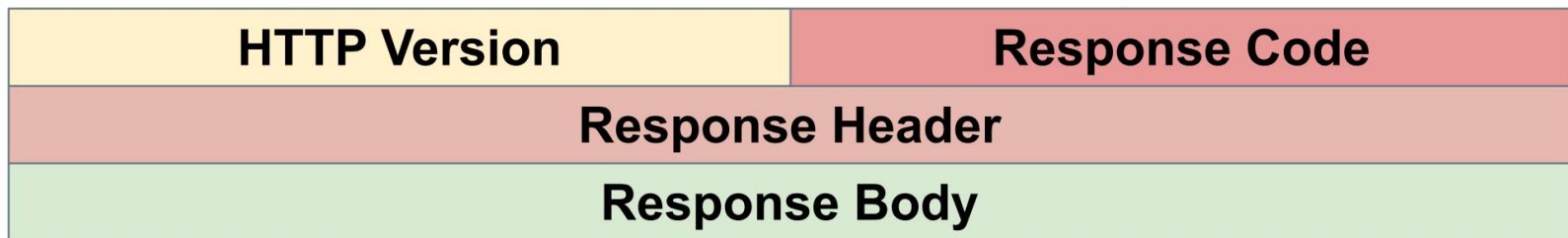
What's the difference between an URI and an URL



URI (Uniform Resource Identifier) Is just the *identifier* of the resource (e.g. [/users/123](#))

URL (Uniform Resource Locator) A URL is a specific type of URI that not only identifies a resource but also provides a means of locating the resource by describing its primary access mechanism (e.g., its network "location"). Essentially, all URLs are URIs, but not all URIs are URLs. (e.g. <https://api.example.com/users/123>)

Anatomy of an HTTP Response



- **Response code** contains request status. 3-digit HTTP status code from a pre-defined list.
- **Response Header** contains metadata and settings about the response message.
- **Response Body** contains the requested resource or data - if the request was successful. Can be as a JSON, XML or other formats such as an HTML or just text.

Anatomy of an HTTP Response

HTTP Version	Response Code
Response Header	
Response Body	

HTTP/1.1 200 OK

Date: Sat, 23 Aug 2014 18:31:04 GMT, Server: Apache/2, Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT, Accept-Ranges: bytes, Content-Length: 32859, Cache-Control: max-age=21600, must-revalidate, Expires: Sun, 24 Aug 2014 00:31:04 GMT, Content-Type: text/html; charset=iso-8859-1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> <html
xmlns="http://www.w3.org/1999/xhtml"> <head><title>Hypertext Transfer
Protocol -- HTTP/1.1</title></head> <body> ...
```

Response codes

Status code	Meaning
200 OK	Request was successful.
301 Moved Permanently	For SEO purposes when a page has been moved and all link equity should be passed through.
401 Unauthorized	Server requires authentication.
403 Forbidden	Client authenticated but does not have permissions to view resource.
404 Not Found	Page not found because no search results or may be out of stock.
500 Internal Server Error	Server side error. Usually due to bugs and exceptions thrown on the server side code.
503 Server Unavailable	Server side error. Usually due to a platform hosting, overload and maintenance issue.

Response co

Status c
200 OK
301 Moved Per
401 Unauthori
403 Forbidder
404 Not Found
500 Internal S
Error
503 Server Un



ould be passed
server side code.
intenance issue.

Looking at more endpoints examples

URL endpoint resource	GET	POST	PUT	DELETE
/surveys	Retrieve all surveys	Create a new survey	Bulk update surveys (not advised)	Remove all surveys (not advised)
/surveys/123	Retrieve the details for survey 123	Error	Update the details of survey 123 if it exists	Remove survey 123
/surveys/123/responses	Retrieve all responses for survey 123	Create a new response for survey 123	Bulk update responses for survey 123 (not advised)	Remove all responses for survey 123 (not advised)
/responses/42	Retrieve the details for response 42	Error	Update the details of response 42 if it exists	Remove response 42

Looking at more endpoints examples: GET

The server receives the HTTP request and returns a response that includes a HTTP status code and a response usually in JSON format. Here is an example request and response to a REST API server asking to read all the responses for survey 123:

Request:

```
GET http://example.com/surveys/123/responses
```

Response:

```
// HTTP status code: 200
{
  survey_id: 123,
  survey_title: "nps onboarding survey",
  responses: [
    {
      response_id: 42,
      nps_score: 9,
      feedback: "love the service",
      respondent_id: 42
    }
    ...
  ]
}
```

Looking at more endpoints examples: POST and Payload

For example, we want to create a new response for the survey 123 which we captured which includes a positive Net Promoter Score of 9, a short feedback message and the unique id of the respondent who gave the feedback.

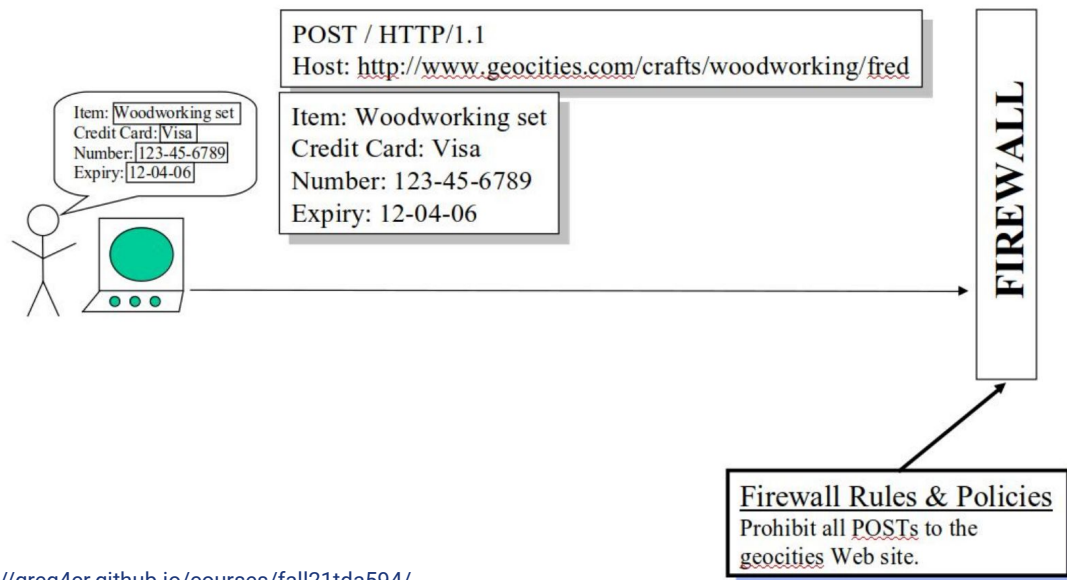
```
// Request:  
POST example.com/surveys/123/responses  
  
// Body payload:  
{  
  survey_id: 123,  
  nps_score: 9,  
  feedback: "love the service",  
  respondent_id: 42  
}
```


Elements of Web Architecture

- **Firewalls** decide which HTTP messages get out, and which get in.
 - These components enforce web *security*.
- **Routers** decide where to send HTTP messages.
 - These components manage web *scalability*.
- **Caches** decide if saved copy of resource used.
 - These components increase web *performance*.

Firewalls

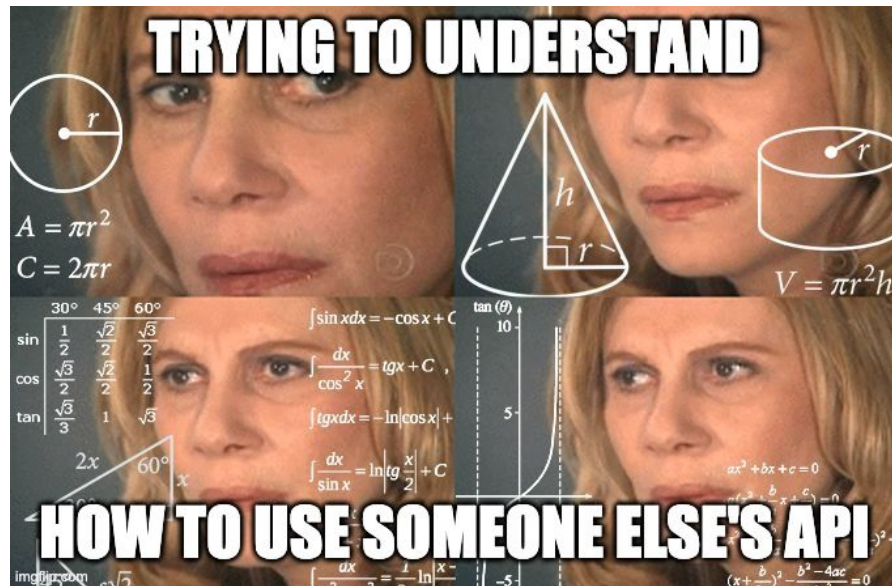
- Firewall decides whether a HTTP message can pass through
- **Important:** All decisions based purely on the **HTTP header**. The firewall **never looks in the payload**
- This message is rejected



Best practices for RESTful APIs

- Avoid using cryptic resource names
- Nouns not verbs when naming resources
 - `GET /users` not `GET /get_users`
- Plural nouns
 - `GET /users/{userId}` not `GET /user/{userID}`
- Dashes in URIs for resources and path parameters but use underscores for query parameters
 - `GET /admin-users/?find_desc=super`
- Return appropriate HTTP and informative messages to the user

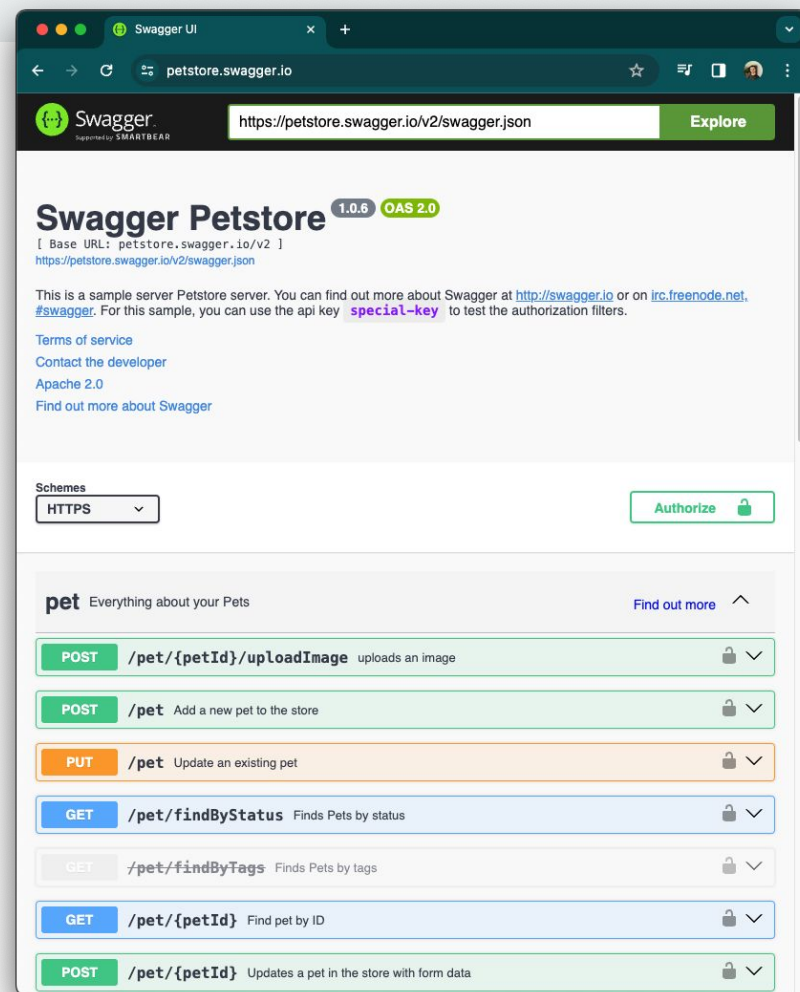
OpenAPI & Swagger



OpenAPI & Swagger

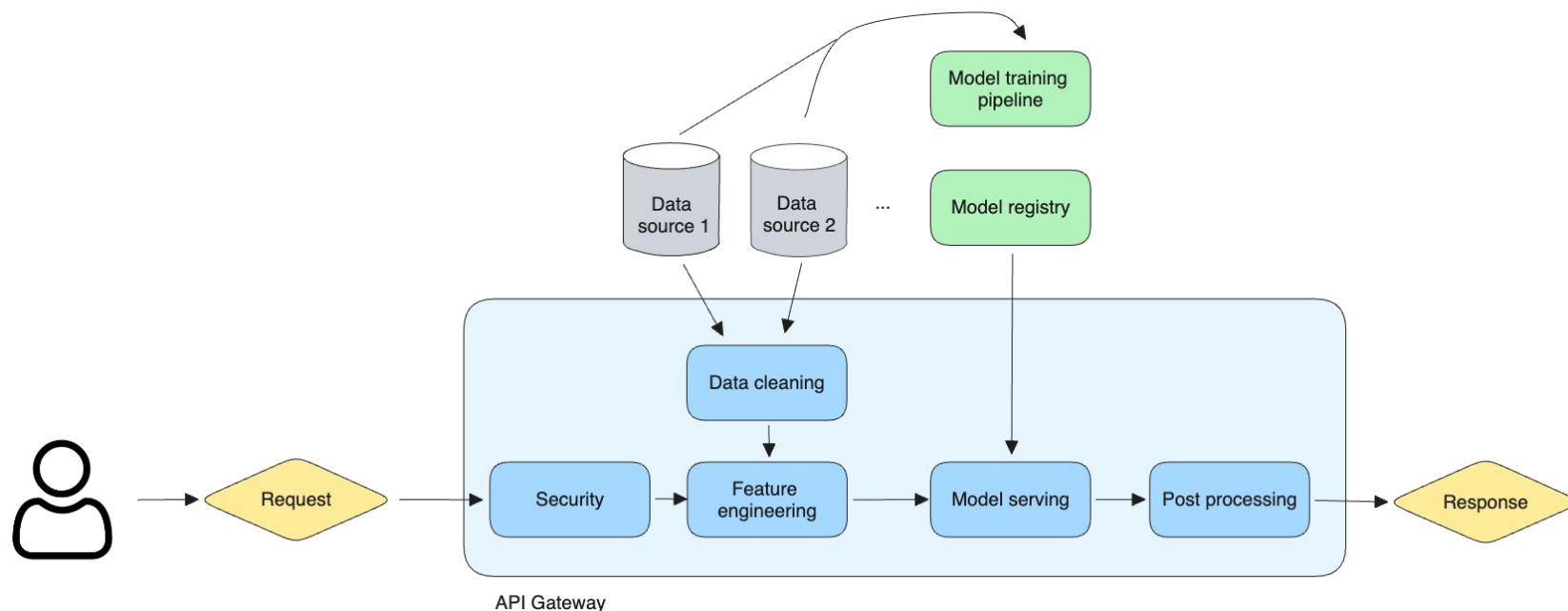
OpenAPI Specification (OAS): Specification language for HTTP APIs that provides a standardized means to define your API to others.

Swagger: Tool to implement your specifications.



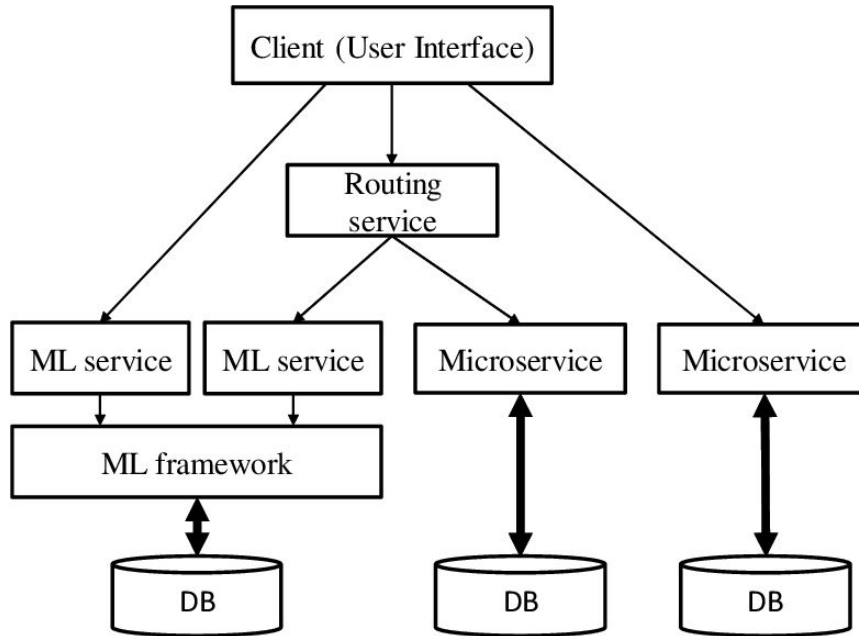
ML API: Might include different logic parts

More components than purely the model serving



You might implement many different independent services:

Microservices





RPC

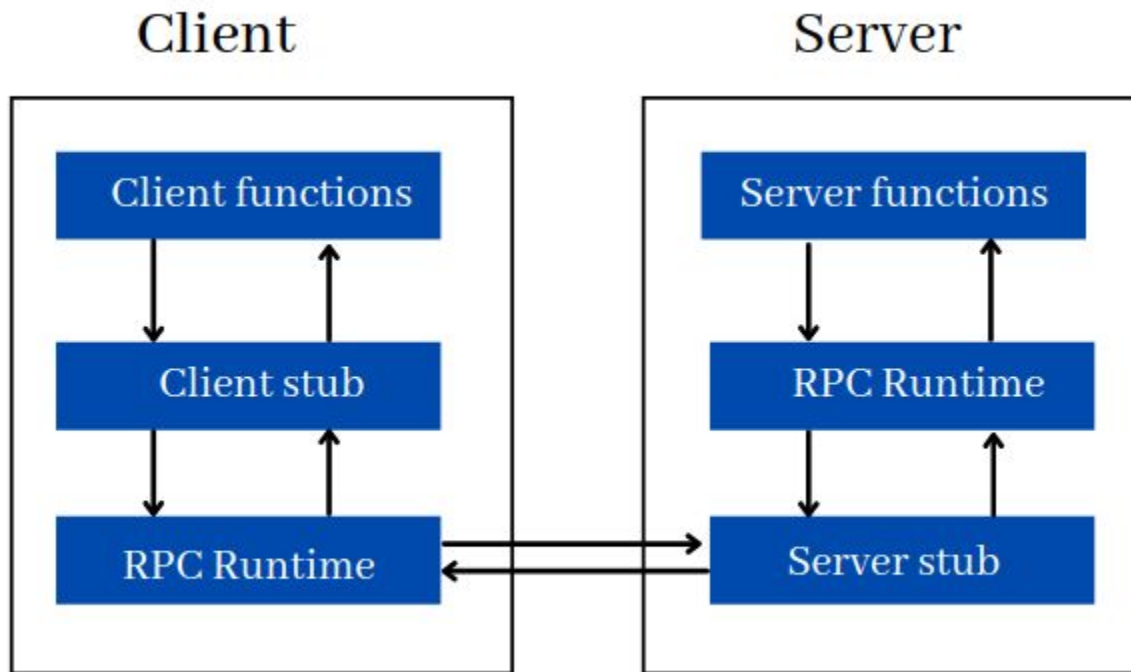
RPC is less frequently used but still relevant

RPC (Remote Procedure Calls). The client completes a function (aka method or procedure) on the server, and the server sends the output back to the client. “Action centric”, when a complex action needs to happen on the server side. Can be stateful, which allows you to interact with an object on the server side as if you could access it locally.

“While REST web APIs are the norm today, Remote Procedure Call (RPC) hasn't disappeared. A REST API is typically used in applications as it is easier for developers to understand and implement. However, RPC still exists and is used when it suits the use case better.”

<https://aws.amazon.com/compare/the-difference-between-rpc-and-rest>

How does RPC work?



An example of an RPC call

```
GET /getMovie/12 HTTP/1.1
Host: api.moviedb.com
Content-Type: application/json
```

Python

```
/* Function definition */
def getMovie(id) {
    // ...
}

/* Function call */
getMovie(id)
```

Python Copy

Stateless vs Stateful

- **Stateless API** does not maintain information about previous requests
 - 👉 REST APIs
- **Stateful API** maintains information about previous requests
 - For most complex systems
 - RPC APIs *can* be stateful

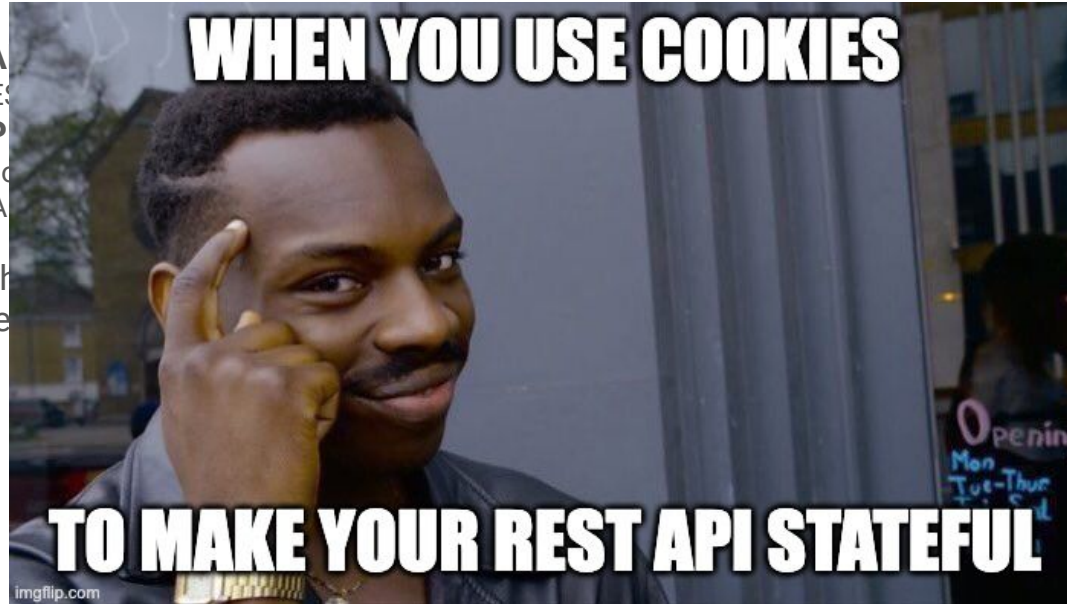
Stateful APIs are helpful for complex workflows where previous interactions need to be considered, while stateless APIs are useful for simpler and more scalable systems.

Stateless vs Stateful

- Stateless API
 - 👉 RES
- Stateful API
 - For mo
 - RPC A

Stateful APIs are h
stateless APIs are

considered, while



RPC principles

Remote invocation: An RPC call is made by a client to a function on the remote server as if it were called locally to the client.

Passing parameters: The client typically sends parameters to a server function, much the same as a local function.

Stubs: Function stubs exist on both the client and the server. On the client side, it makes the function call. On the server, it invokes the actual function.

IDL (Interface Definition Language): Language used to describe the interface that an RPC service provides. It is a way to define the functions, methods, and data types that are available for remote invocation, ensuring that both the client and server understand the format and types of data that will be sent and received.

gRPC

- Not a spelling mistake
- A more modern version of RPC developed by Google
- Includes key improvements such as
 - Protocol Buffer (protobuf) as IDL
 - use of HTTP/2
 - Bidirectional streaming and flow control
 - Authentication

Summary of differences: RPC vs. REST

	RPC	REST
What is it?	A system allows a remote client to call a procedure on a server as if it were local.	A set of rules that defines structured data exchange between a client and a server.
Used for	Performing actions on a remote server.	Create, read, update, and delete (CRUD) operations on remote objects.
Best fit	When requiring complex calculations or triggering a remote process on the server.	When server data and data structures need to be exposed uniformly.
Statefulness	Stateless or stateful.	Stateless.
Data passing format	In a consistent structure defined by the server and enforced on the client.	In a structure determined independently by the server. Multiple different formats can be passed within the same API.



Guest lecture connexion



Lab: Flask



Wrap-up

Lecture summary

Topic	Concepts	To know for...	
		Project	Exam
API	<ul style="list-style-type: none">• API• REST• RPC		Yes
Connexion			
Flask		Possibly	

Project objective for sprint 3

#	Week	Work package	Requirement
3.1	W05	Build an API to serve your model and any extra logic that is needed to serve it (e.g. using Flask). You should be able to run the API locally.	Required
3.2	W05	Package your model serving API in a Docker container . This too should be run locally.	Required
3.3	W06	Deploy your model serving API in the Cloud. You should be able to call your model to generate new predictions from another machine. <u>Attention</u> : This can incur Cloud costs . Make sure to use a platform where you have credits and not burn through them. You can ask for support from the teaching staff in that regard.	Required

It's feedback time!



Next week...

- Today we saw how an API works and how to build a local Flask API
- Next week we will see how you can deploy an API in the Cloud
 - Use Cloud infrastructure (GPUs, ...)
 - Serve your ML model so it can be used by other services (e.g. an application)



Next week...

Guest lecture: [Philippe Modard](#)



Experience



Google

7 yrs 10 mos



Tech Lead

Full-time

Aug 2023 - Present · 8 mos

Brussels, Brussels Region, Belgium · Hybrid

Kaggle Models team (kaggle.com/models)



Tech Lead

Full-time

Jul 2022 - Present · 1 yr 9 mos

New York, United States · Hybrid

Kaggle Models team (kaggle.com/models)



[Find Pre-trained Models | Kaggle](#)



Tech Lead

Full-time

Feb 2019 - Sep 2023 · 4 yrs 8 mos

Greater New York City Area

Kaggle Notebook backend team (Google Cloud AI).

Next week...

Guest lecture: [Philippe](#)



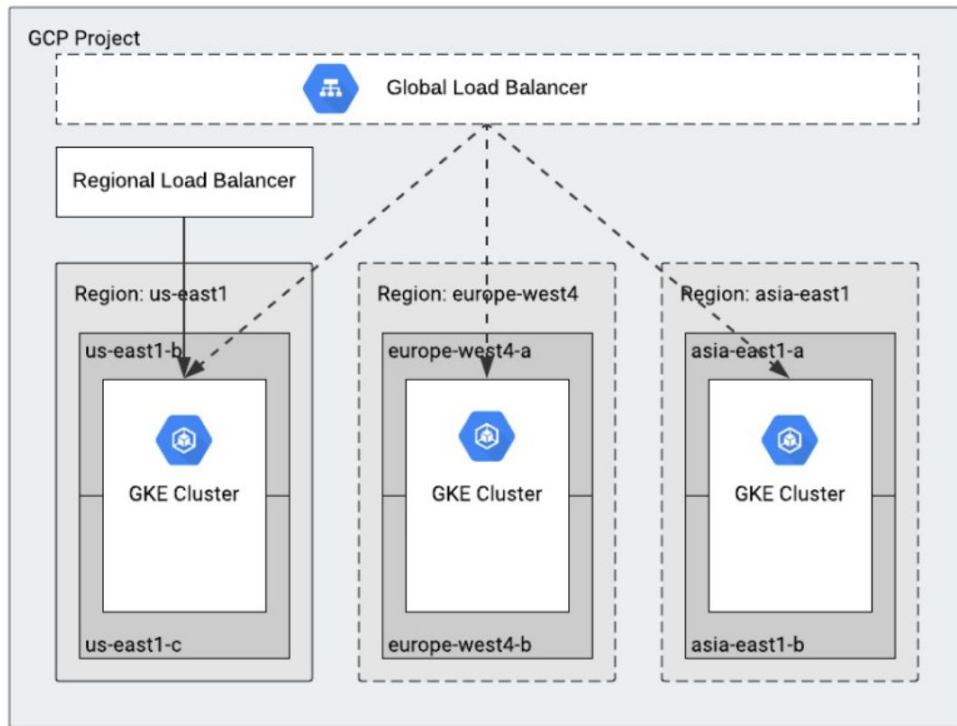
Philippe Modard • 1st

Tech Lead @ Google | Backend Development, Cloud Computing, Ma...

4yr •

...

How Kaggle load-balances a gRPC application across multiple GKE clusters.



A multi-cluster gRPC architecture on GKE

[bit.ly](#)



APPENDIX

Main types of APIs

Types of APIs:

- **SOAP:** These APIs use Simple Object Access Protocol. Client and server exchange messages using XML. This is a less flexible API that was more popular in the past.
- **RPC:** (Remote Procedure Calls). The client completes a function (or procedure) on the server, and the server sends the output back to the client.
- **Websocket:** Websocket API is another modern web API development that uses JSON objects to pass data. A WebSocket API supports two-way communication between client apps and the server. The server can send callback messages to connected clients, making it more efficient than REST API.
- **REST:** These are the most popular and flexible APIs found on the web today. The client sends requests to the server as data. The server uses this client input to start internal functions and returns output data back to the client.
 - *Focus for today!*

REST API

Three main components:

- **URL endpoint:** Is a URL link that represents the resources we want to access. Resources can be text, images, documents or any data entry. For example, `example.com/surveys` allows us to view or create survey templates and `example.com/surveys/123/responses` allows us to do the same for all responses of survey 123.
- **HTTP verb:** Tells the server what we want to do with the URL endpoint resource. For example, a POST request means we want to create a new survey template and a GET request means we want to view an existing survey template.
- **Body message:** Is an optional custom payload which contains a message with the properties and values we want to use to create or update a given resource.

Could include more content based on the “Designing Services with REST” section of <https://greg4cr.github.io/courses/fall21tda594/Lectures/Fall-21-Lecture9-APIs.pdf>

Wrapping ML functionality

- Invoking a ML model does not always fit within CRUD primitives
 - E.g., Translating sentences, detecting sentiment
- Solution: HTTP RPC-style APIs
 - All methods with URL paths in the form `POST https://{service-endpoint}/v1/{endpoint}:predict`
 - not RESTful APIs but still based on HTTP
 - Compatible with frameworks and specification languages for RESTful APIs
 - Example: the Slack Web API
 - `https://api.slack.com/web`

WSGI and other frameworks

https://www.tutorialspoint.com/flask/flask_overview.htm

Jinja2

REST vs RESTful

#1. Definitions

REST



It is used to develop APIs which enable interaction between the client and the server. It should be used to get a piece of data when the user connects any link to the particular URL.

RESTful



It is a web application that follows the REST infrastructure which provides interoperability between different systems on the entire network.