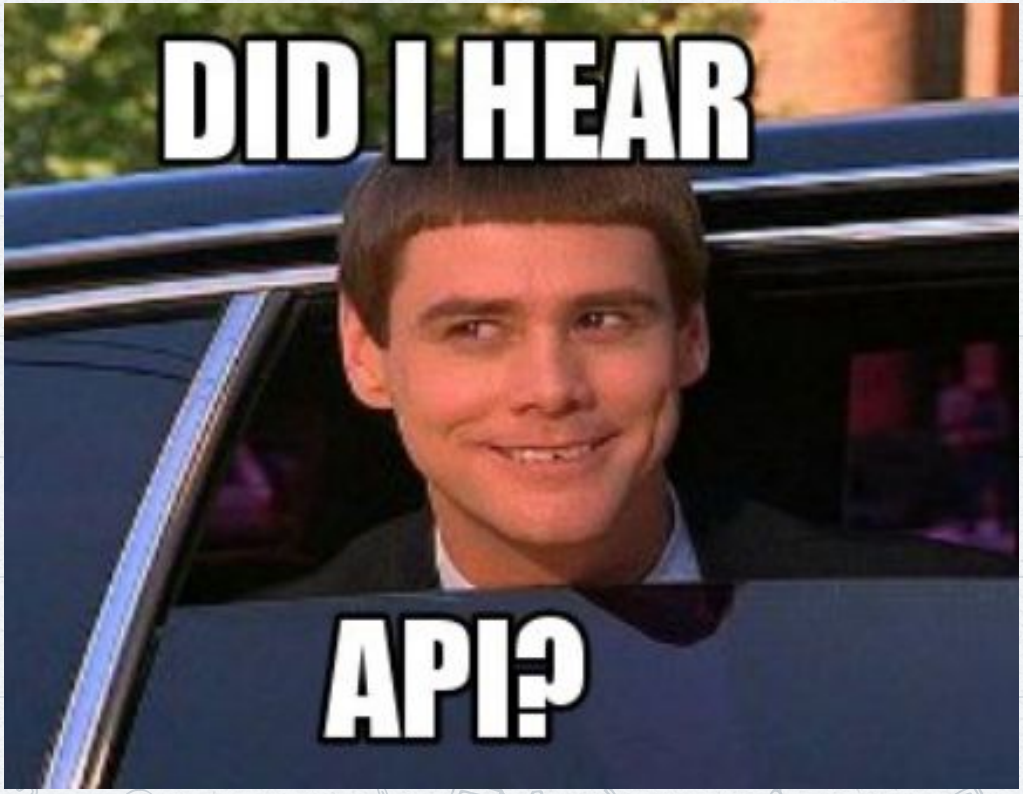
REST & gRPC


DID I HEAR
API?

Who invented REST?

- **REST** is a key architectural principle of the **World Wide Web**, and received a large amount of attention.
- Involved in the development of and **Uniform Resource Identifiers**.
- Co-founder of the **Apache HTTP Server** project and was a member of the interim **OpenSolaris** Boards until he resigned from the community in 2008.

Roy Fielding

REST APIs

Verb	Path	action	used for
GET	/photos	index	display a list of all photos
GET	/photos/new	new	return an HTML form for creating a new photo
POST	/photos	create	create a new photo
GET	/photos/:id	show	display a specific photo
GET	/photos/:id/edit	edit	return an HTML form for editing a photo
PUT	/photos/:id	update	update a specific photo
DELETE	/photos/:id	destroy	delete a specific photo

Why REST ?

1. Any browser can be used because the REST approach uses the standard GET, PUT, POST, and DELETE verbs.
2. Totally stateless operations.
    - If you need stateless CRUD (Create, Read, Update, and Delete) operations.
3. Caching situations.
    - As REST is stateless, but we get to store data as cache.

```
<Head>
  <UserName>Ada Lovelace</UserName>
</Head>
<Body>
  <Action>Login</Action>
  <Password>WerdToThePass</Password>
</Body>
```


```
{
  "userName": "Ada Lovelace",
  "password":"WerdToThePass"
}
```


YOU STILL USE SOAP?
BETTER TAKE A REST

REST Concepts

1. Resources
    - A resource is basically a data object that we need to modify using a REST API. These resource objects often contain a type, associated data, relationships to other resources, and a set of methods that operate on it.

    - It is similar to an object instance in an object-oriented programming language.

2. Paths
    - Each resource in a REST API has a defined path that can be used to do operations on that resource. In most cases, this is a concatenation of the base path of the API and the resource name.

3. Request/Response Headers
    - Extra information sent to the server and back to specify the type of request/response, encoding, and content type, customized parameters, Etc.

4. Response Code
    - These codes returned with the response and signify the status of the request sent to the server. Standard HTTP response codes are used.

Representational State Transfer - (rest)

V2

1. REST is essentially an "architecture for networked applications."

2. REST is an architectural style as well as an approach for communications purpose that is often used in various web services development.

3. It's a set of standards that describe how computers should communicate with each other and with applications across a network.

4. REST defines certain specific operations that applications should be able to do in order to satisfy all of the CRUD (create, read, update, delete) requirements.

5. HTTP is the protocol most frequently used to implement the REST architecture, supplying operations like PUT, POST, GET, DELETE, and HEAD.

https://www.quora.com/What-are-RESTful-APIs-and-how-do-they-work

Representational State Transfer - (rest)

V2

Application state is server-side data which servers store to identify incoming client requests, their previous interaction details, and current context information.

Resource state is the current state of a resource on a server at any point of time – and it has nothing to do with the interaction between client and server. It is what you get as a response from the server as API response. You refer to it as resource representation.

https://www.quora.com/What-are-RESTful-APIs-and-how-do-they-work

REST Principles

1. Stateless
    - Stateless means the server does not remember anything about the user who uses the API.
    - Each individual request contains all the information the server needs to perform the request, regardless of other requests made by the same API user.

2. Uniform Interface
    - Every resource in the server should have only one logical URI and should expose methods to access this resource.
    - Paths in the API should follow standard naming conventions.

3. Client-server separation
    - The client and the server implementation should be independent.

4. Layered system
    - We should divide our web service into different layers that cannot see beyond their immediate layer. E.g. authentication layer, data access layer, etc.

5. Cacheable
    - Due to statelessness, caching is very easy.The server should send information about whether or not the data is cacheable, in a response.

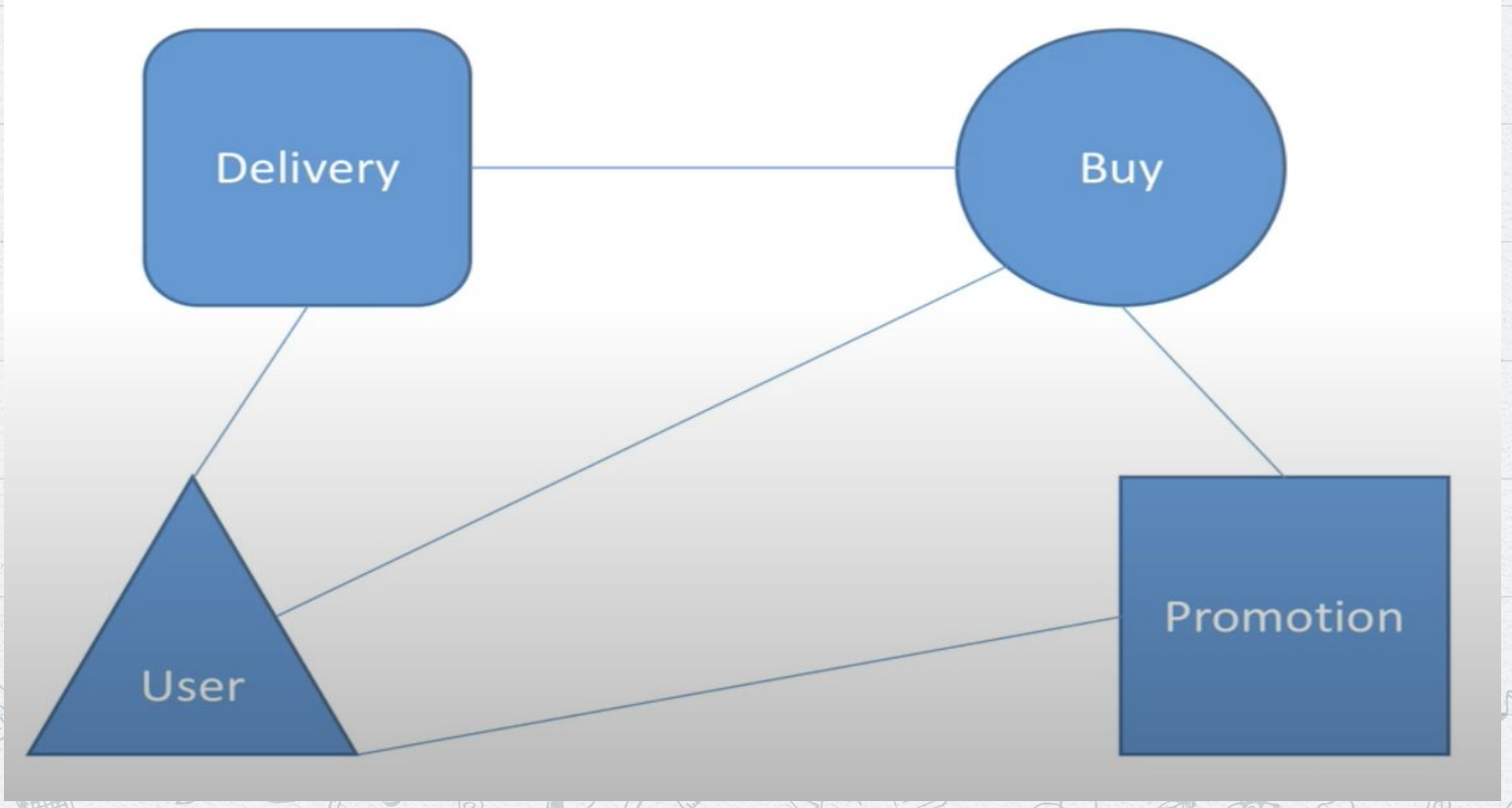
Representational State Transfer

V1

- The REST style is an abstraction of the architectural elements within a distributed hypermedia system.
- REST provide interoperability between computer systems on the Internet
- REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components.
- It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application.

MICRO SERVICES

V1



MICRO SERVICES

V1

- API to exchange data
- Data formats
- Error patterns
- Load balancing
- Many other things

MICRO SERVICES

V1

- Need to think about data model
  - JSON
  - XML
  - Something Binary?
- Need to think about the endpoint
  - GET /api/v1/user/123/post/456
  - POST /api/vl/user/123/post

- Need to think about how to invoke it and handle errors
  - API • Error

Introduction to gRPC

- gRPC is a modern open source high performance RPC framework that can run in any environment.
- gRPC is a very popular open source RPC framework with support of languages such as C++, Java, Python, Go, Ruby, Node.js, C#, Objective-C and PHP.
- It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication.
- It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services.

So, What's RPC?

- Remote procedure call (RPC) architecture is popular in building scalable distributed client/server model based applications.

- RPC client and server run-time stubs take care of the network protocol and communication so that you can focus on your application.

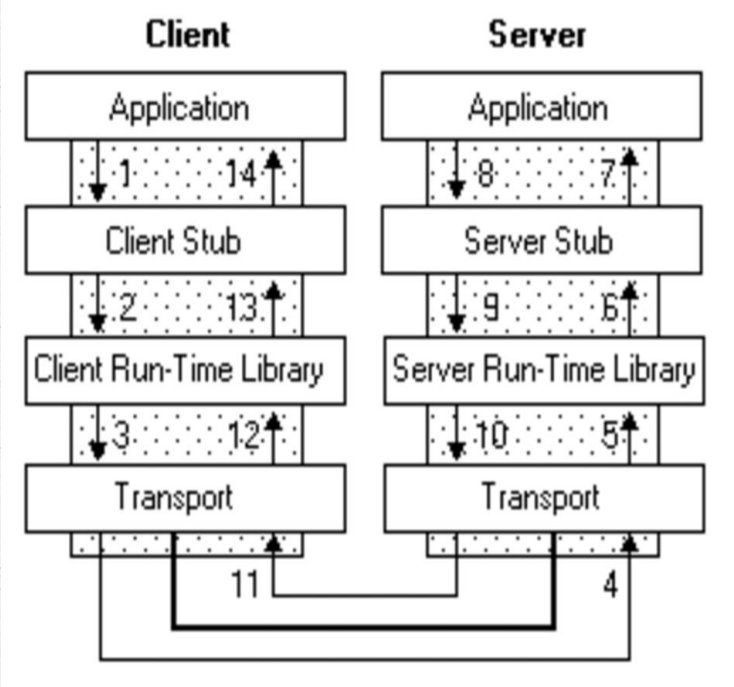
Client Code
(any language)

```
{code}
...
server.CreateUser(user)
...
{code}
```

Server Code
(any other language)

```
// function creating users
def CreateUser(User user){
...
}
```

1. Client App makes a local procedure call to client stub containing the parameters to be passed on to the server.
2. Stub is a piece of code which packages the parameters into a package which makes it suitable to transfer and store.
3. Then client stub serializes the parameters (process called ***marshalling***), and forwards the request to local client run-time, which send it to server runtime
4. Server runtime passes it to server stub, which unmarshalls, (opens up the package and take out the parameters) and then calls the actual procedure on server.



How does RPC works?

- We implement REST to expose API endpoints, through and API gateway, to external applications (mostly web apps in AngularJS and ReactJS). Inter service communication is then handled using RPC.

- Remote procedure call's data serialization into binary format for inter-process (server to server) communication makes it ideal for building out scalable microservice architecture by improving performance.

RPC vs REST

- Data Format
  - REST:- JSON, which is Human-readable, flexible format that is easily debuggable. The simple format works with all programming languages. The primary drawback is that data requires additional memory to store and uses extra space over the network since it is stored in a friendly human-readable format.

  - RPC:- Machine-only data format that conforms to a specification that can be hand-coded or can be a generic data format like protocol buffers. Requires a special library to interpret. Library availability depends on the popularity of the data format and also the programming language used. Good binary formats tend to be much more compact and easier to parse.

RPC vs REST

- gRPC is faster than REST in most tests.
- Those tests include Heavy, Normal, Small Payloads, along with multiple calls and Parallel calls.
- The only test that REST won, was the tests where the payload was small and several clients made a server call at the same time.
- When creating a web service that expects several client-calls at the same time, and uses a small payload as input and output, REST might be the better choice.

#gRPC walks into a bar and orders a drink. After a few minutes the bartender hands #gRPC their drink.

#REST walks in.

#REST walks in and orders a drink.

#REST walks in and asks if their drink is ready.

#REST walks in and asks if their drink is ready.

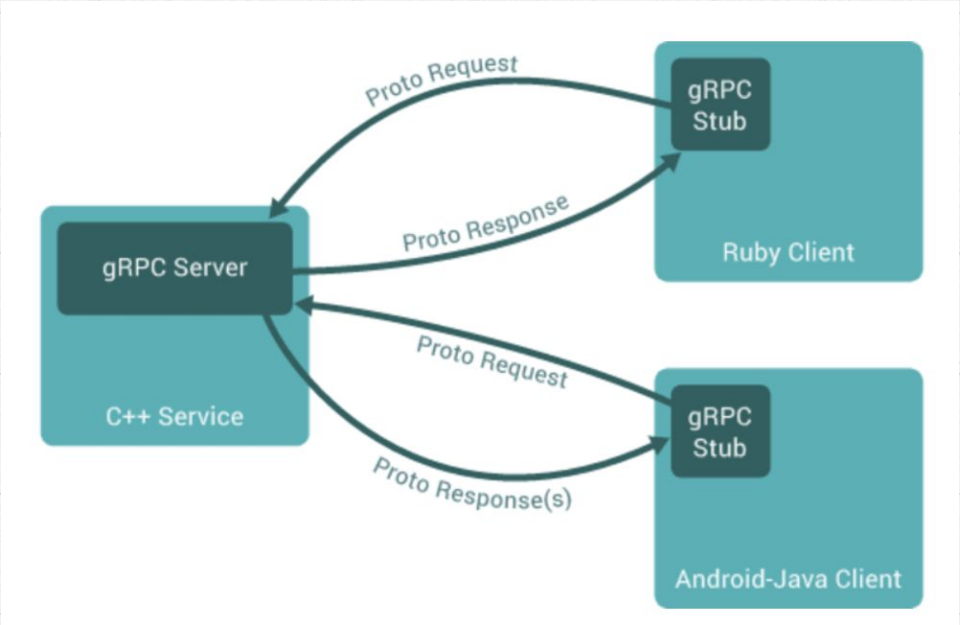
#REST walks in and ...

- In all other occasions, gRPC was faster. We can conclude that gRPC is our overall winner, but besides the speed, gRPC and REST both have their own advantages:
  - gRPC can use protocol buffer for data serialization. This makes payloads faster, smaller and simpler.
  - gRPC uses HTTP/2 to support highly performant and scalable API's and makes use of binary data rather than just text which makes the communication more compact and more efficient. gRPC makes better use of HTTP/2 then REST.
  - gRPC is also type-safe. This basically means that you can't give an apple while a banana is expected. When the server expects an integer, gRPC won't allow you to send a string because these are two different types.

- REST can be used cross-language which makes these web services flexible and scalable.
- One of the main advantages of REST is that it does not need to setup a client.
- A lot of people have experience with it and a lot of other web services (and clients) use REST. Having a REST web service makes it easier for other people to interact with your web service.
- Communication often happens using a JSON, which is human readable. This makes it easier for developers to determine if the client input is send correctly to the server, and back.

Now, how gRPC works?

- With gRPC, a client application can directly call methods on a server application on a different network as if the method was local.
- The beauty about RPC is that it is language agnostic, which means grpc server can be written in Java handling client calls from node.js, PHP and Go.



Advantages of using gRPC?

- gRPC is built on HTTP/2 under it's BSD license. What this means is that the procedure calls get the goodness of HTTP/2 to build real time applications taking advantages of features such as bidirectional streaming, flow control, header compression and multiplexing requests.

- gRPC's default serialization protocol, Protocol Buffer, also transmits data in binary format which is smaller and faster as compared to good old JSON and XML.

- Protocol buffer's latest version proto3 which makes it easy to define services and automatically generate client libraries

Who's using gRPC?

In our initial use of gRPC we've been able to extend it easily to live within our opinionated ecosystem. Further, we've had great success making improvements directly to gRPC through pull requests and interactions with Google's team that manages the project. We expect to see many improvements to developer productivity, and the ability to allow development in non-JVM languages as a result of adopting gRPC.

With support for high performance bi-directional streaming, TLS based security, and a wide variety of programming languages, gRPC is an ideal unified transport protocol for model driven configuration and telemetry.

Our switch from a home-grown RPC system to gRPC was seamless. We quickly took advantage of the per-stream flow control to provide better scheduling of large RPCs over the same connection as small ones.