CLOUD COMPUTING

REST and Web Services

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:
Significant information in the slide deck presented through the Unit 1 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

Service oriented architecture (SOA)

SOA, or **service-oriented architecture**, defines a way to make software components reusable via service interfaces. These interfaces utilize common communication standards in such a way that they can be rapidly incorporated into new applications without having to perform deep integration each time.

Each service in an SOA embodies the code and data integrations required to execute a complete, discrete business function. The service interfaces provide loose coupling, meaning they can be called with little or no knowledge of how the integration is implemented underneath. The services are exposed using standard network protocols—such as SOAP (simple object access protocol)/HTTP or JSON/HTTP—to send requests to read or change data. The services are published in a way that enables developers to quickly find them and reuse them to assemble new applications.

Two major service-oriented architecture styles :

1. **REST** (REpresentational State Transfer)
2. **SOAP based WS** (Web Services)

https://www.ibm.com/cloud/learn/soa, https://www.geeksforgeeks.org/service-oriented-architecture/

REST(REpresentational State Transfer)

- Cloud architecture involves number of distributed autonomous systems or components or applications frequently communicating or interacting between themselves for performing an application request.
- REST is an architectural (sometimes also called programming) style for providing a set of rules to be used for building computer systems on the web, and making it easier for systems to communicate with each other.
- REST helps in building a client-friendly distributed systems that are simple to understand and simple to scale or following the REST principles while designing your application (distributed system), you will end up with a system that exploits the Web's architecture to your benefit
- An API which adheres to these constraints are considered RESTful and helps getting benefits of a Client Server distributed architecture by reducing the complexities of distributed systems (often called RESTful systems) and allows the benefits to be achieved more easily.
- REST has been employed throughout the software industry and is a widely accepted set of guidelines for creating stateless, reliable web APIs. RESTful web APIs are typically loosely based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data.

REST(REpresentational State Transfer) - Design Principles

The following are some of the mandatory constraints (principles) for designing a RESTful system.

1. **Client Server Constraint (Separation of concerns)**

   This sets the constraint on the system to be built in such a fashion where the client can change and evolve without having to change anything on the server, and also on the contrary different aspects of the Server should also be able to be changed without breaking clients. **Or**

   Its about how the client sends the server a message, and how the server rejects or responds to the client

2. **Stateless Constraint**

   The communication between the client and the server must remain stateless between requests. Each request the client makes should contain all information needed for the server to answer successfully. All of the state information should be transferred back to the client as part of the response and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client. This is the “S” and “T” in REST; we are always giving the state back to the client. Not keeping the state in the Server impacts performance which are being addressed by the other concerns.

3. **Cache Constraint**

   In order to improve network efficiency, cache constraints are added require that the data within a Server’s response to a request, be implicitly or explicitly labeled/marked as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

REST(REpresentational State Transfer)

Constraints to designing a RESTful system (Cont.)

4. **Uniform Interface Constraint**

   A uniform interface makes it generic and improves the overall visibility of interactions on how the client and server exchange requests and responses. Implementations are decoupled from the services they provide. This could impact performance for non web based data interactions. REST is defined by 4 interface constraints

   a. Resource and Resource Identification
   b. Manipulation of Resources through Representations
   c. Self-descriptive messages
   d. Hypermedia as the engine of application state

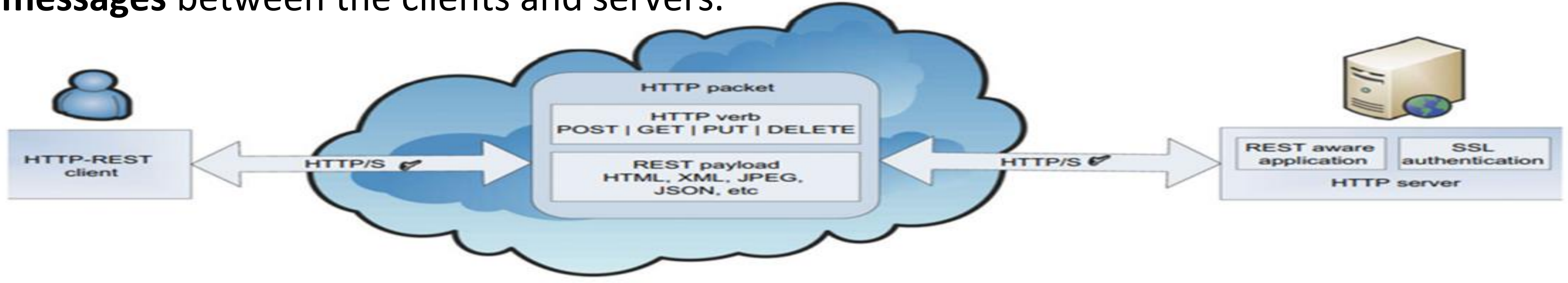
   These will be discussed further.

5. **Layered System Constraint**

   Messages between a client and server can go through a hierarchy of intermediate components (could be load balancers, proxy servers, firewalls, or other types of devices) where these intermediate components should not be able to "see" the next layer. Thus the interaction between the client and server should not be affected by these devices and thus ensures independence and bounding of complexity. All communication should remain consistent, even if one of the layers is added or removed https://www.redhat.com/en/blog/rest-architecture

REST(REpresentational State Transfer) Functioning

- The REST architectural style is designed for network-based applications, specifically client-server applications.
- Clients can only access resources using **URIs**.
- Client requests a resource using a URI and the server responds with a **representation** of the resource.
- To ensure responses can be interpreted by the widest possible number of client **applications** a representation of the resource is sent in **hypermedia** format.
- Thus, a resource is manipulated through hypermedia representations transferred in **messages** between the clients and servers.



Ref - https://dev.to/flippedcoding/what-is-the-difference-between-a-uri-and-a-url-4455

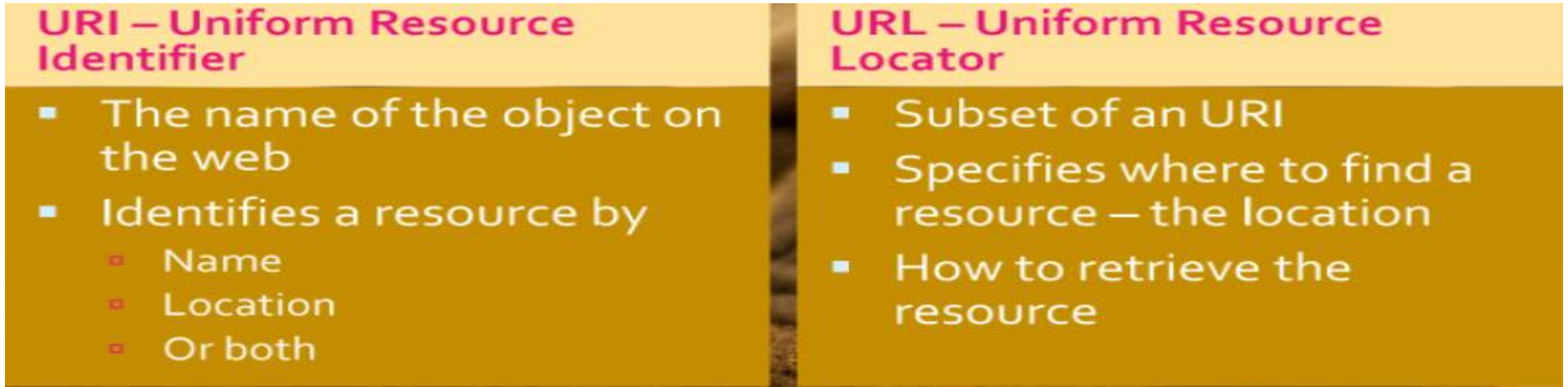
REST Architectural Elements – Resources and Resource Identification through URIs (T2)

Resource and Resource Identification: The key abstraction of information in REST is a resource. A resource is a "Thing". Any information that can be named can be a resource, such as a document or image or a temporal service.

Eg. Light on board in seminar hall. If you need to control it, you name it

/pesu/b-block/groundfloor/seminarhall/board/light/1

The RESTful web service exposes a set of resources which identify targets of interaction with its clients. Each particular resource is identified by a unique name, or more precisely, a Uniform Resource Identifier (URI) providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery. The URIs can be bookmarked or exchanged via hyperlinks, providing more readability.



URI – Uniform Resource Identifier
- The name of the object on the web
- Identifies a resource by
  - Name
  - Location
  - Or both

URL – Uniform Resource Locator
- Subset of an URI
- Specifies where to find a resource – the location
- How to retrieve the resource

REST Principles - Uniform, Constrained API Interface (T2)

Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol.

Resources are manipulated using a fixed set of four CRUD (create, read, update, delete) Interfaces which supports operations as below :

GET — retrieve a specific resource or a collection of resources

POST — Create or partial update of a resource

PUT — Create or update resource by replacement.

DELETE — remove a resource

So typically in your application clients say for a book lending library something like

- GetListofBooks()
- AddBookToShoppingCart()

Need to define these operations. No standard style exists. These operations are implemented using the Interfaces described earlier.

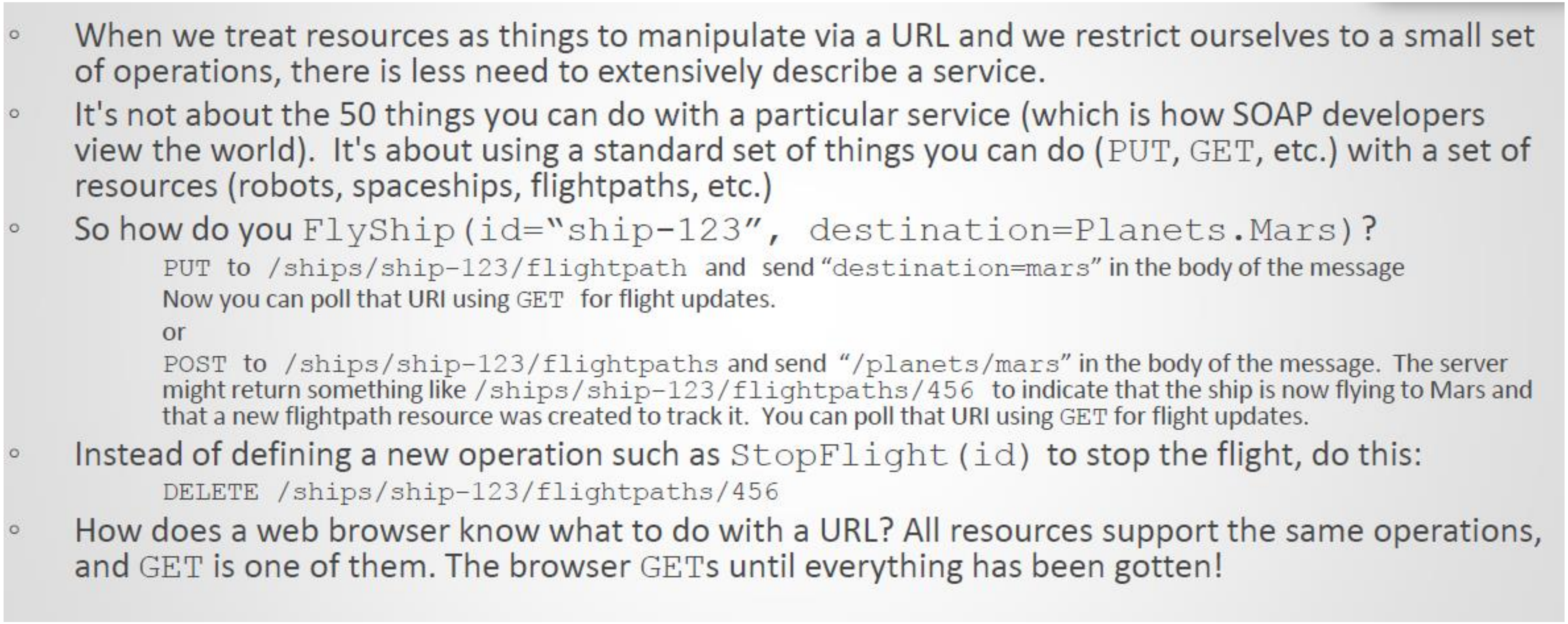
REST Principles – Operations – another view

- REST Operations
  - GET
    - Retrieve representation of a resource
  - PUT
    - Create or update resource by replacement.
  - POST
    - Create or partial update of a resource
  - DELETE
    - Remove a resource

- Usage
  - GET
    - https://bblock/seminarhall/board/light1.xml
    - To check if light is ON
  - POST to
    - https://bblock/seminarhall/board/light1.xml
    - To turn on/off the light
    - Put ON in the body of the message

REST Principles - Operations

CLOUD COMPUTING : Operations



- When we treat resources as things to manipulate via a URL and we restrict ourselves to a small set of operations, there is less need to extensively describe a service.
- It's not about the 50 things you can do with a particular service (which is how SOAP developers view the world). It's about using a standard set of things you can do (`PUT`, `GET`, etc.) with a set of resources (robots, spaceships, flightpaths, etc.)
- So how do you `FlyShip(id="ship-123", destination=Planets.Mars)`?

  `PUT to /ships/ship-123/flightpath` and send "`destination=mars`" in the body of the message
  Now you can poll that URI using `GET` for flight updates.
  or
  `POST to /ships/ship-123/flightpaths` and send "`/planets/mars`" in the body of the message. The server might return something like `/ships/ship-123/flightpaths/456` to indicate that the ship is now flying to Mars and that a new flightpath resource was created to track it. You can poll that URI using `GET` for flight updates.
- Instead of defining a new operation such as `StopFlight(id)` to stop the flight, do this:

  `DELETE /ships/ship-123/flightpaths/456`
- How does a web browser know what to do with a URL? All resources support the same operations, and `GET` is one of them. The browser `GET`s until everything has been gotten!

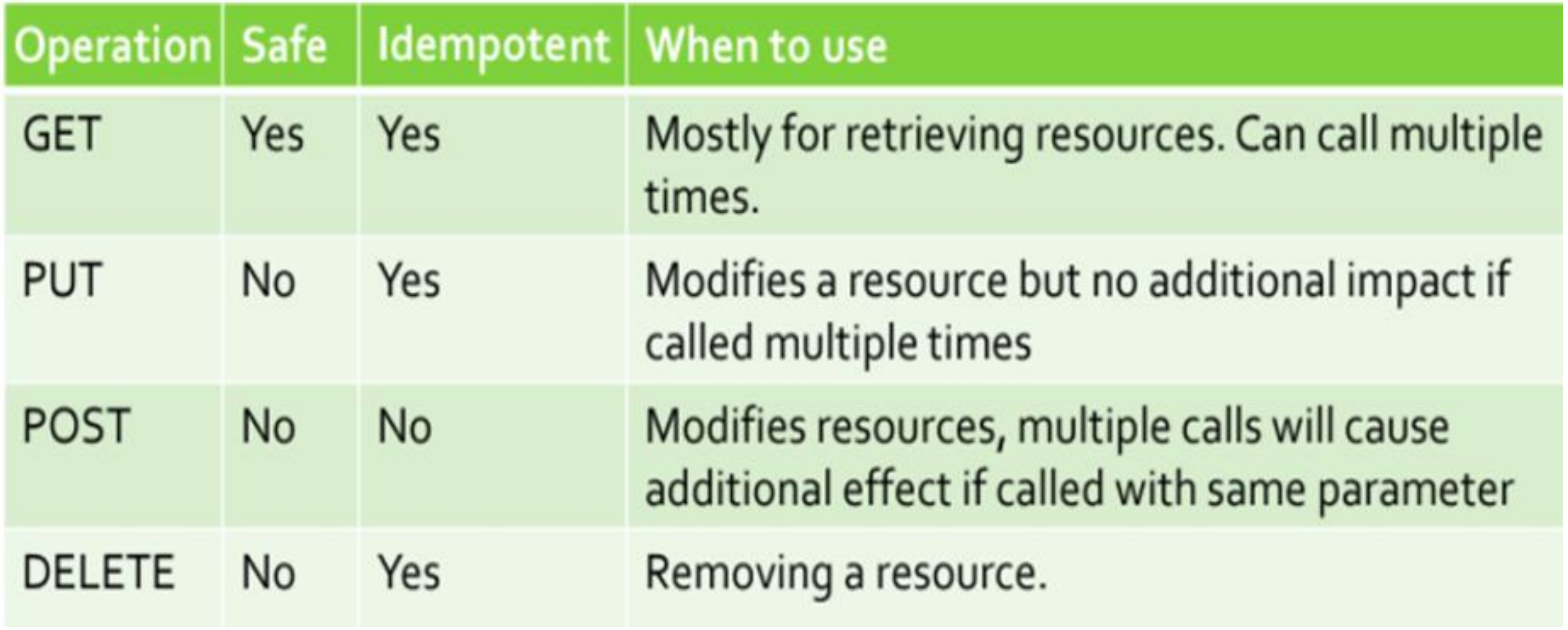
CLOUD COMPUTING : Characteristics of these Operations: Safe - Idempotent

- Safe
  - Does not modify the resources
  - Example
    - GET

- Idempotent
  - Idempotent has no additional effect if it is called more than once with ***same input parameters***
  - Can repeatedly perform operations.
  - No effect on servers
    - How would you like to pay for a seat multiple times?

REST Principles - Operations

CLOUD COMPUTING : Operations with characteristics



Operation	Safe	Idempotent	When to use
GET	Yes	Yes	Mostly for retrieving resources. Can call multiple times.
PUT	No	Yes	Modifies a resource but no additional impact if called multiple times
POST	No	No	Modifies resources, multiple calls will cause additional effect if called with same parameter
DELETE	No	Yes	Removing a resource.

REST Principles - Stateless Interactions

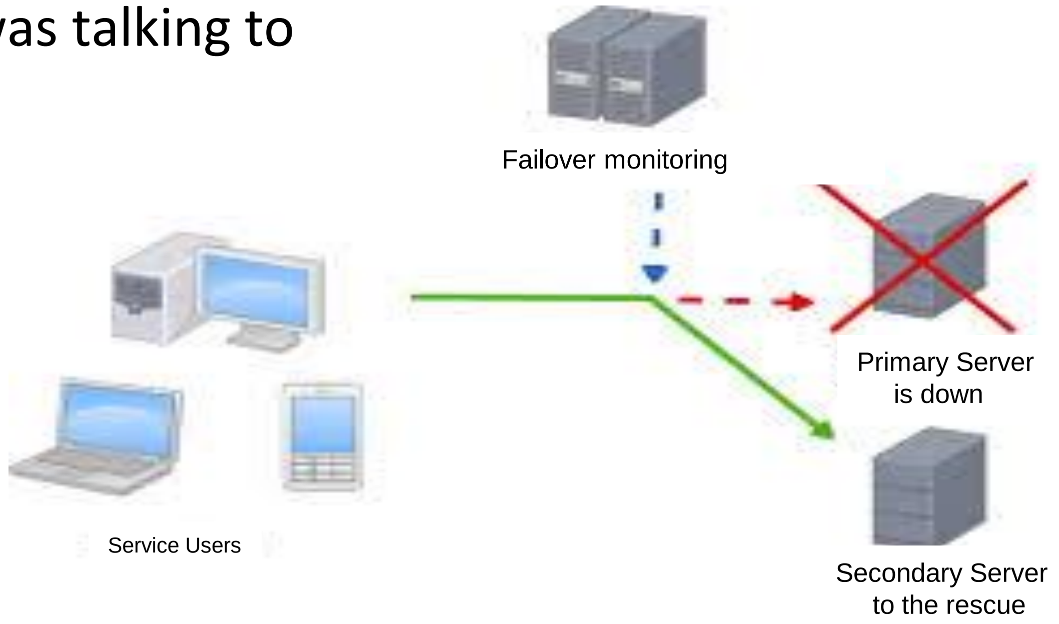
(T2)

Stateless Interactions: Systems that follow the REST paradigm are stateless, meaning that the server does not need to know anything about what state the client is in and vice versa. In this way, both the server and the client can understand any message received, even without seeing previous messages. Each client request is treated independently.

Statelessness benefits :

1. Clients isolated against changes on server
2. Promotes redundancy - unlocks performance:
    a. don't really need to know which server client was talking to
    b. No synchronization overhead

No state saved on server, so even if server fails, the client connects to another server and continue



Ref - https://dev.to/flippedcoding/what-is-the-difference-between-a-uri-and-a-url-4455

REST Principles - Representations

Representation : A representation is a snapshot in time of the state of a given resource. It's a sequence of bytes made up of the data, plus representation metadata to describe those bytes.

- This captures the current or intended state of that resource and helps transferring that representation between the interacting components.
- The metadata could be in the form of binary or textual key-value pairs.
- The data format of a representation is known as a media type and this type identifies a specification that defines how a representation is to be processed.
- The message type is ***hypermedia***, which refers to any content that contains links to other forms of media such as images, movies, and text. hypermedia links in the API response contents. It allows the client to dynamically navigate to the appropriate resources by traversing the hypermedia links.
- Navigating hypermedia links is conceptually the same as browsing through web pages by clicking the relevant hyperlinks to achieve a final goal.
- All the response representation need to be self-descriptive

https://restfulapi.net/

REST Principles - Self-Descriptive Message

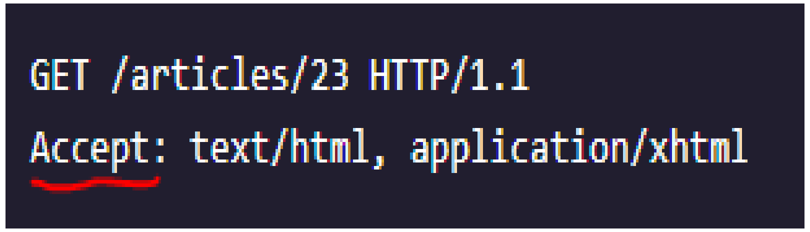
(T2)

A REST message includes enough information to describe how to process the message. This enables intermediaries to do more with the message without parsing the message contents.

In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats (e.g., HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.) i.e. the resources can have **multiple representations**. RESTful systems empowers client to ask for data in a form they understand.

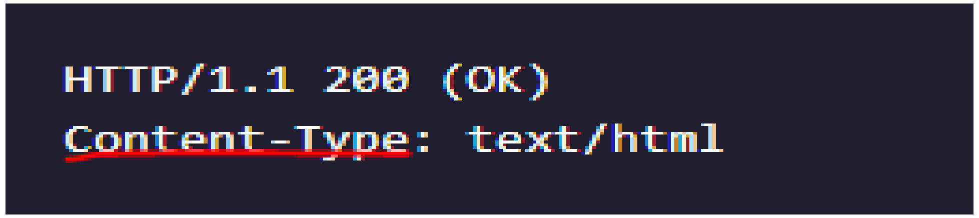
Example:

media type (also known as a Multipurpose Internet Mail Extensions or MIME type) **indicates the nature and format of a document, file, or assortment of bytes**. MIME types are defined and standardized in IETF's RFC 6838



```
GET /articles/23 HTTP/1.1
Accept: text/html, application/xhtml
```

Client request



```
HTTP/1.1 200 (OK)
Content-Type: text/html
```

Server Response

Metadata about the resource is available and can be used for various purposes, such as cache control, transmission error detection, authentication or authorization, and access control.

More points on REST

1. **Rest is not a standard** :

   It is a design and architectural style for large scale distributed systems

2. **Rest is not same as http:**

   Http can also be used in non-RESTful way. Example : SOAP services that use http to transport data.

Web Services

(T2)

Web Service: a software system designed to support interoperable machine-to-machine interaction over a network.

The term “web service” is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web.

Once a web service is deployed, other applications and other web services can discover and invoke the deployed service. Other systems interact with the web service in a manner prescribed by its description.

A web service is one of the most common instances of an SOA implementation.

The two prominent ways of implementing Web Services. This could be using the approach of :

1. Simple Object Access Protocol (SOAP)
2. REST (we have discussed this at length)

Simple Object Access Protocol (SOAP)

SOAP is a protocol which was designed before REST came into the picture. The main idea behind creating SOAP was to ensure that programs built on different platforms and programming languages could securely exchange data.

SOAP provides a structure for transmission of XML documents over various Internet protocols, such as SMTP, HTTP, and FTP.

A SOAP message consists of element called **envelope**, which is used to encapsulate all of the data in the SOAP message which contains a ***Header*** element that contains header information such as ***authentication credentials*** which can be used by the calling application, and a ***body*** element that carries the payload of the message.

The content of the payload will be marshaled by the sender's SOAP engine and un-marshaled at the receiver side.

WSDL or web service description language describes the functionality of the SOAP based web service

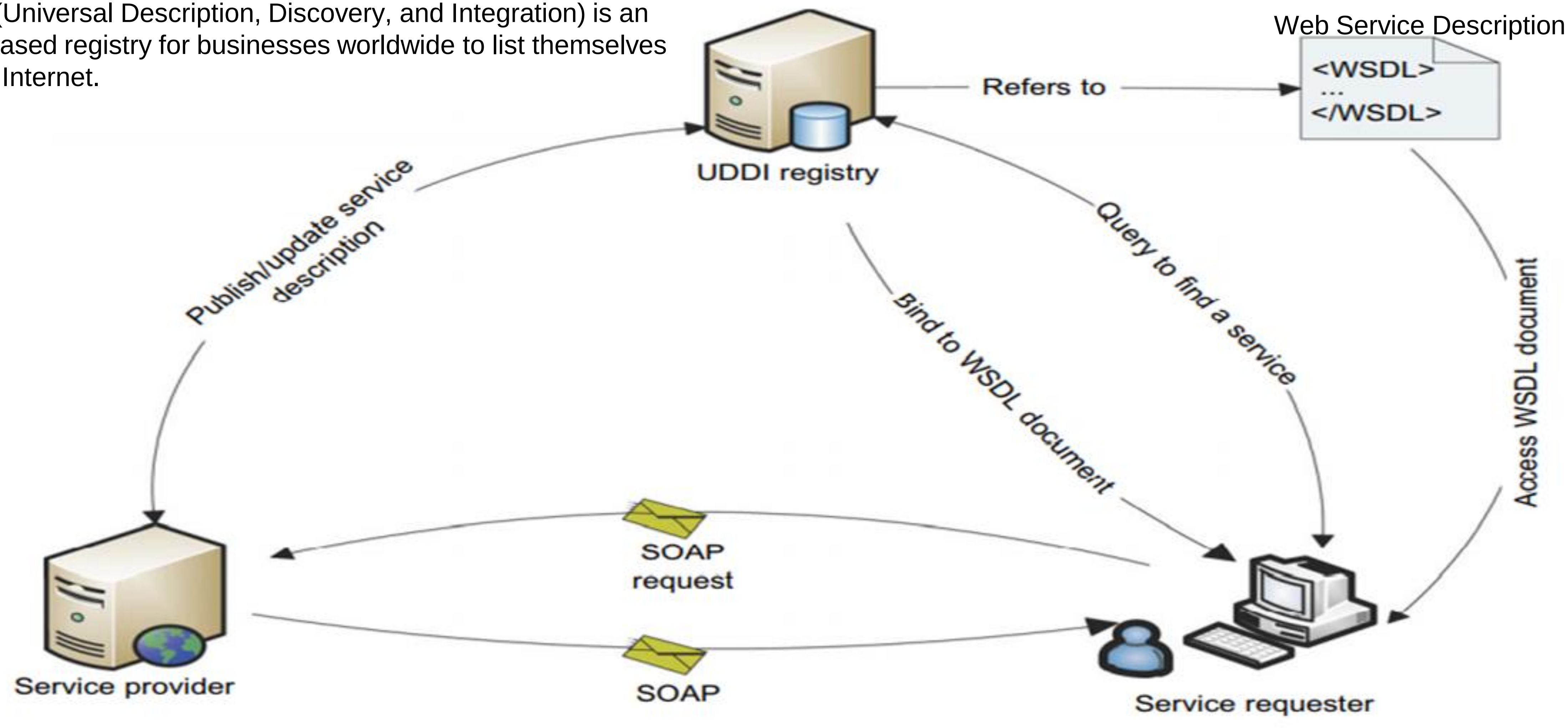
Ref - https://www.guru99.com/soap-simple-object-access-protocol.html

Web Services in action say using SOAP

(T2)

UDDI (Universal Description, Discovery, and Integration) is an XML-based registry for businesses worldwide to list themselves on the Internet.

Web Service Description Language)



A simple web service interaction among provider, user, and the UDDI registry.

Web Services in action

(T2)

Refer to Figure 5.2(previous slide):

Step 1 : The service provider publishes its location and description to the UDDI registry.

Step 2: The service requester queries the UDDI registry using names, identifiers, or the specification. The UDDI registry finds the corresponding service and returns the WSDL document of the service.

Step 3: The service requester reads the WSDL document and forms a SOAP message binding to all constraints in the WSDL document.

Step 4: This SOAP message is sent to the web service as the body of an HTTP/SMTP/FTP request.

Step 5: The web service unpacks the SOAP request and converts it into a command that the application can understand and executes the command.

Step 6: The web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP/SMTP/FTP request

Step 7: The client program unpacks the SOAP message to obtain the results

THANK YOU

Prafullata Kiran Auradkar
Department of Computer Science and Engineering
prafullatak@pes.edu