Web Services

2024/25 COMP3322 Modern Technologies on WWW

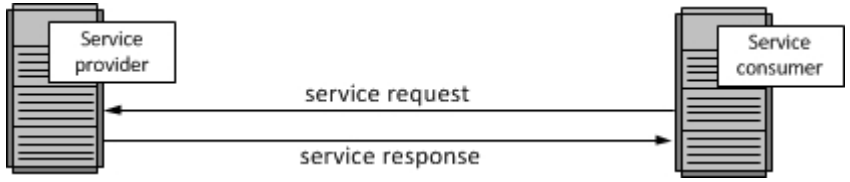
Contents

- What is Web Service?
- REST

What is Web Service?

- The term “Web Services” is confusing
  - Will you consider Moodle **a service** provided by the University for delivery of course contents to the students?
  - Will you consider Google Docs **a service** provided by Google for documents processing?
- These are not Web Services. These are Web Applications.
- W3C definition of a Web Service:
  - Software system designed to support **interoperable** **machine-to-machine** interaction over a network.
  - The keys are:
    - Web services are designed for machine-to-machine (or **application-to-application**) interaction
    - Web services should be interoperable - **Not platform dependent**
    - Web services should allow communication over a network

A Service



- A service is a software component provided through a network-accessible endpoint.
  - A service consumer sends a service **request** message to a service provider. The service provider returns a **response** message in rendering the service.
  - The requests and subsequent responses are **defined in some way** that is **understandable** to both the service consumer and service provider.
- A service is a function that is well-defined, self-contained, and does **not depend** on the context or state of other services.
- Web services are characterized by three factors:
  - What they do (the business **functionality** they expose).
  - Where they are (the **web site** which exposes that functionality).
  - How they can be accessed (the set of published **interfaces** necessary to use the exposed functionality).

Type of Web Service

- There are mainly two types of web services:
  - SOAP web services.
    - The Simple Object Access Protocol (SOAP) standard uses the XML language defining a message architecture and message formats.
    - A simple SOAP Message has the following elements:
      - The Envelope element - encapsulates the header, body and fault elements
      - The header element (optional)
      - The body element
      - The Fault element - for the error response (Optional)
    - SOAP services often contain a machine-readable description of the operations offered by the service, written in the Web Services Description Language (WSDL), an XML language for defining interfaces syntactically.
  - RESTful web services.

REST

- REST is used for building Web services that are lightweight, maintainable, and scalable in nature.
- REST stands for **Re**presentational **S**tate **T**ransfer, which defines a set of **architectural principles** by which you can design Web services that focus on **a system's resources**.
- A web service based on REST is called a RESTful web service.
- REST is not dependent on any protocol, but almost every RESTful service uses HTTP as its underlying protocol.

Resources and Representations

- **Any information** that can be named can be a resource:
  - a document, an image, a function, a collection of resources, etc.
  - A resource has an identifier, which is a URI that uniquely identifies that resource.
- The state of resource at any particular time (e.g., at the time of the request) is known as **resource representation**.
  - A representation consists of data, metadata describing the data and **hypermedia** links which can help the clients in transition to next desired state.
  - A representation should be able to completely represent a resource.
    - If there is a need to partially represent a resource, then you should think about breaking this resource into child resources.
- REST does not impose any restriction on the format of a resource representation.
  - Could be using JSON, XML or HTML.

REST Architecture Constraints

- An application or architecture considered RESTful or REST-style has the following characteristics:
  - Uniform Interface
  - Client-Server Separation
  - Be stateless
  - Cacheable
  - Layered

Uniform Interface

- A RESTful service uses a **directory structure-like URIs** to address its resources.
- <protocol>://<service-name>/<ResourceType>/<ResourceID>
- Another important attribute of a request is the 'VERB' which identifies the **operation to be performed** on the resource.
- REST asks developers to use HTTP methods explicitly and in a way that's **consistent** with the HTTP protocol.
- This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods.
  - To create a resource on the server, use POST.
  - To retrieve a resource, use GET.
  - To change the state of a resource or to update it, use PUT.
  - To remove or delete a resource, use DELETE.

Client-Server Separation

- The client and the server act independently, each on its own, and the interaction between them is only ***in the form of client-server request-response interaction***.
- The server just sits there waiting for requests from the client to come.

Be Stateless

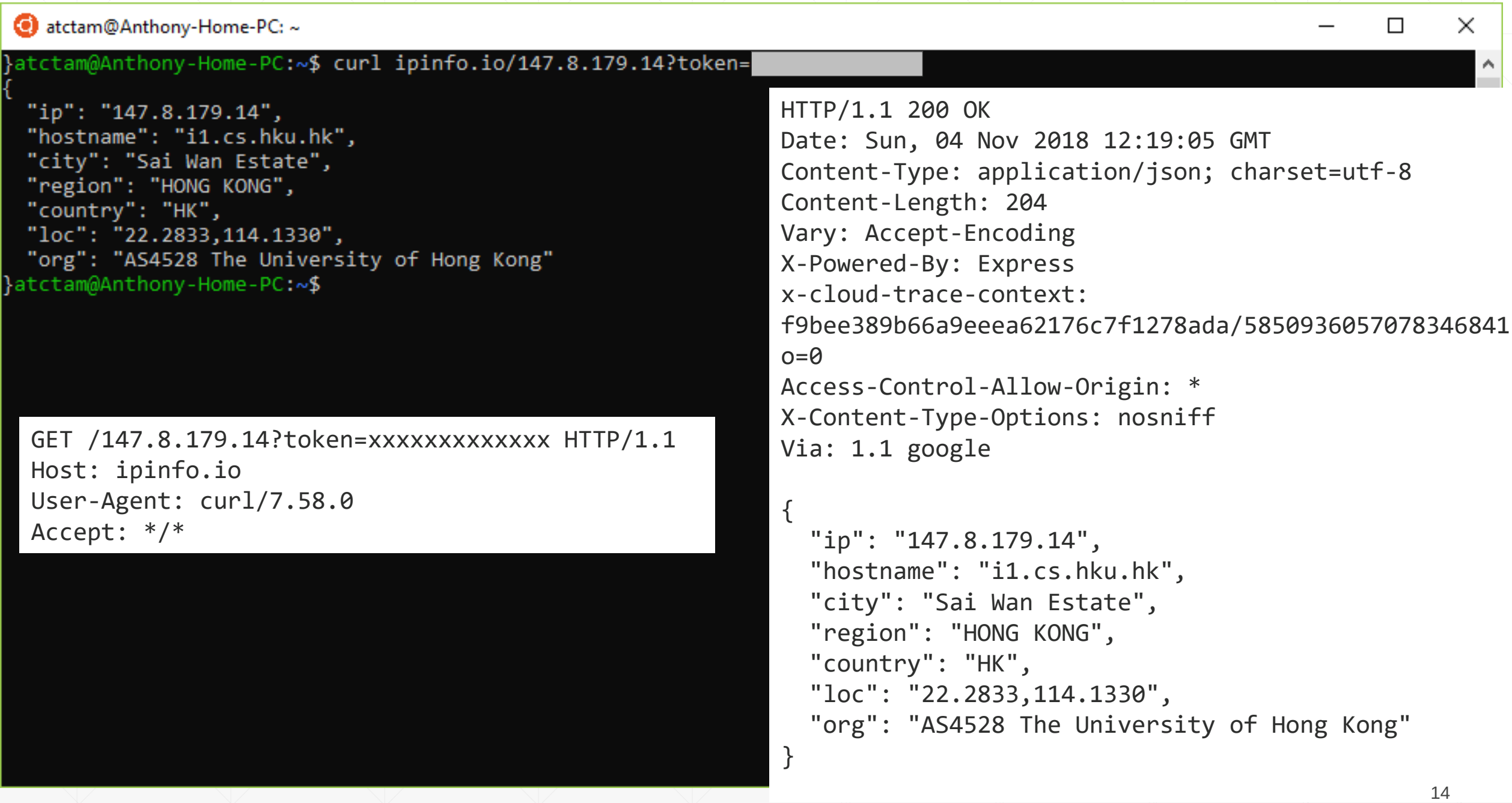
- Stateless means the server does not remember anything about the client.
- **Each individual request** contains **all the information** the server needs to perform the request and return a response, regardless of other requests made by the same user before.
- No client context shall be stored on the server between requests. The **client** is responsible for **managing the state** of the application.
- This constraint basically says that your RESTful API should not have a login and logout function, which would involve the use of sessions, and isn't allowed.
- If authentication is needed, the server will look for authentication information **within** each request.

Cacheable and Layered

- Cacheable
  - In REST, caching shall be applied to resources **when applicable** and then these resources MUST declare themselves cacheable.
  - The server does this by including a Cache-Control and Last-Modified (a date value) HTTP response header.
- Layered
  - Between the client and the server, there might be a number of servers in the middle.
  - These servers might provide a security layer, a caching layer, a load-balancing layer, or other functionality. Those layers should not affect the request or the response.

Example REST Web Service

- ipinfo.io - Find geolocation with ip address
  - https://ipinfo.io/


atctam@Anthony-Home-PC: ~
}atctam@Anthony-Home-PC:~$ curl ipinfo.io/147.8.179.14?token=
{
"ip": "147.8.179.14",
"hostname": "i1.cs.hku.hk",
"city": "Sai Wan Estate",
"region": "HONG KONG",
"country": "HK",
"loc": "22.2833,114.1330",
"org": "AS4528 The University of Hong Kong"
}atctam@Anthony-Home-PC:~$
GET /147.8.179.14?token=xxxxxxxxxxxxx HTTP/1.1
Host: ipinfo.io
User-Agent: curl/7.58.0
Accept: */*
HTTP/1.1 200 OK
Date: Sun, 04 Nov 2018 12:19:05 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 204
Vary: Accept-Encoding
X-Powered-By: Express
x-cloud-trace-context:
f9bee389b66a9eeea62176c7f1278ada/5850936057078346841;
o=0
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
Via: 1.1 google
{
"ip": "147.8.179.14",
"hostname": "i1.cs.hku.hk",
"city": "Sai Wan Estate",
"region": "HONG KONG",
"country": "HK",
"loc": "22.2833,114.1330",
"org": "AS4528 The University of Hong Kong"
}

Designing RESTful API

- In Rest, URI's should represent resources. URI's should be hierarchical and as **self descriptive** as possible.
- Always use HTTP Methods.
  - GET : Should not update anything. Should be idempotent (same result in multiple calls).
  - POST : Should create new resource. Ideally return JSON with link to newly created resource.
  - PUT : Update a known resource.
  - DELETE : Used to delete a resource.

https://restcountries.com/

References and Resources

- What are Web Services? Architecture, Types, Example
  - https://www.guru99.com/web-service-architecture.html
- Web Services Explained
  - https://www.service-architecture.com/articles/web-services/web_services_explained.html
- RESTful Web services: The basics
  - https://developer.ibm.com/articles/ws-restful/
- REST API Best Practices – REST Endpoint Design Examples
  - https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/
- APIs
  - toddmotto / public-apis (https://github.com/toddmotto/public-apis/blob/master/README.md)
  - Any API (https://any-api.com/)