Web Services

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

Contents

Introduction

What is a Service?

A service is software that provides runtime capabilities to other software.

You already have local services on your computer that provide OS-level capabilities. e.g. printer, logging.

We're specifically going to talk about remote services, running on a different computer, over a network.

Most applications use remote services.

- Your online DB is just a service!

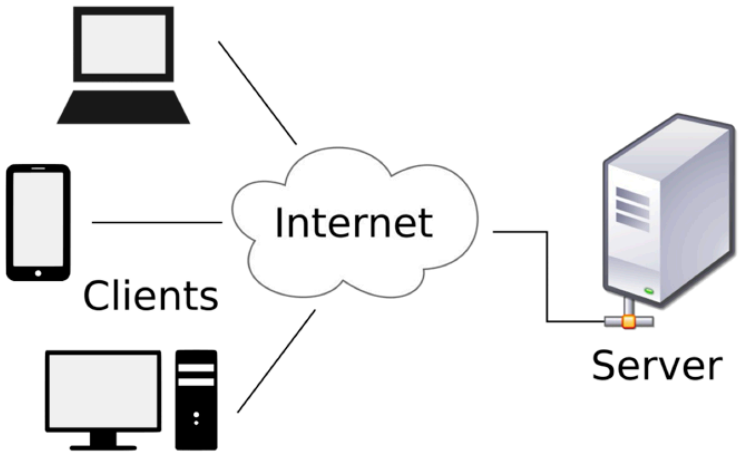

Figure 1: A server might service dozens or hundreds of other systems (aka clients).

Benefits of a Service

Data Sharing

- Services that are accessible over the internet allow us to share data between users, or between devices.
  - e.g., a TODO list showing up on both my phone and desktop.
  - e.g., ability to share TODO items with other users (multiple users).

Performance

- If designed correctly, distributing our work across systems can allow us to grow our system to meet high demand
  - e.g., Amazon AWS “spins up” services as needed and shuts them down again when demand subsides.

Expanding Capabilities

- There are actions that your application cannot perform on its own.
  - e.g., using a remote GenAI model to provide capabilities to your application.

Accessing a Service

How do client and server communicate?

We have protocols for packaging, transmitting and interpreting information.

- **Communication protocols**, for transferring data.
  - These include HTTP for web traffic, FTP for sending files over a network, or TCP/IP for data packets.
- **Security protocols** to establish secure channels for the transmission of data.
  - These include SFTP and SSH.
- **Network management protocols** for controlling and managing devices.
  - These include SMNP and ICMP.

Accessing a Service

TCP/IP

TCP/IP stands for Transmission Control Protocol/Internet Protocol, and is a suite of communication protocols used to interconnect network devices on the internet.

These two protocols work together:

- Transmission Control Protocol (TCP) defines how applications can create channels of communication across a network. It also manages how a message is assembled into smaller packets before they are then transmitted over the internet and reassembled in the right order at the destination address.
- Internet Protocol (IP) defines how to address and route each packet to make sure it reaches the right destination. Each gateway computer on the network checks this IP address to determine where to forward the message.

Accessing a Service

Addressing

For computers to communicate and transmit data between them, they need to be able to locate one another. To facilitate this, every device on a network is assigned a unique identifier, or IP address.

This is usually presented as four numbers (0-255), dot separated e.g., here's the IP address of my computer as I'm typing this:

```
$ ipconfig getifaddr en0
192.168.2.147
```

What if you have multiple services on that computer? We can supply a second identifier, a port, a 16-bit unsigned integer that represents an endpoint.

- Servers usually have multiple ports that they monitor for different services.
- Some ports are reserved e.g., port 80 is a web server, port 21 is an ftp server.
- Refer to a specific system and port by using `address:port`
  - e.g., https://www.google.com:80.

Web Services

Introduction

The world-wide-web is an example of an early service design.

- A web browser (front-end) can request content from a web server (back-end) hosted on a different system.
- Requests are sent using the Hypertext Transfer Protocol (HTTP).
  - HTTP is a request-response model, where the client requests data.
  - The URL of a website describes the protocol, address, and document. e.g., https://uwaterloo.ca/about/

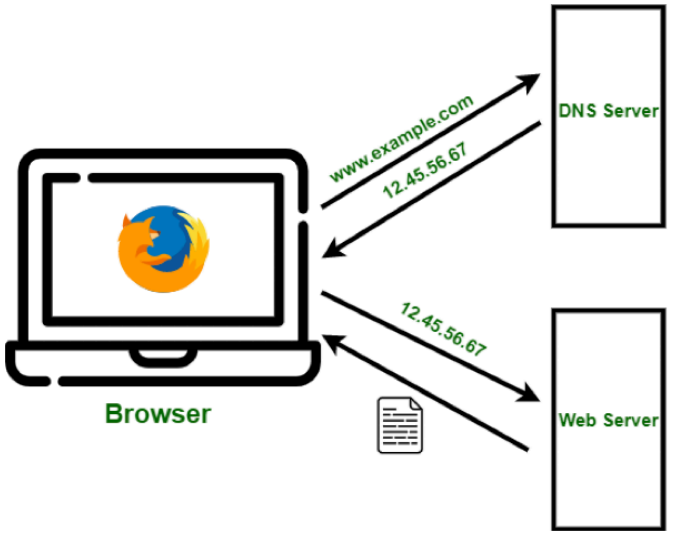

Figure 2: A browser and web server interact.

HTTP Protocol

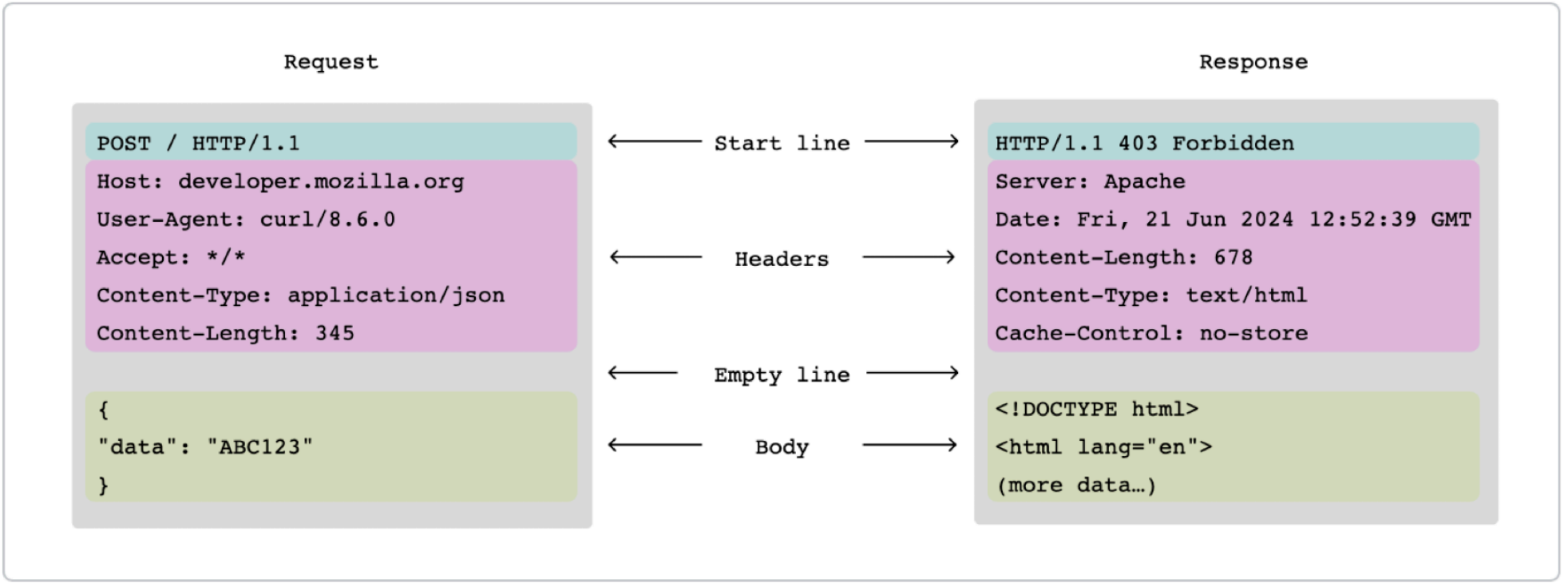

Figure 3: An example of a HTTP request/response. The request includes a header with information about the request, and a body that contains any required data. The response includes relevant header information and data, usually HTML or JSON.

HTTP Protocol

Here's the raw data from an HTTP response. It's just structured text (HTML).

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
avif,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9
Accept-Language: en-US,en;q=0.9
Connection: keep-alive

<!doctype html>
<html lang="en">
<head><title>Example Domain</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<body><div><h1>Example Domain</h1>
<p>This domain is for use in documentation examples</p>
</body></html>
```

HTTP Protocol

HTTP specifies that there are different types of client requests. The HTTP protocol supports the following types of request methods.

- **GET**: Requests that the target resource transfers a representation of its state. These requests only retrieve data.
- **HEAD**: Requests that the target resource transfers a representation of its state, like for a GET request, but without the data. Uses include checking if a page is available or finding the size of a file.
- **POST**: Requests that the target resource processes the representation enclosed in the request according to the semantics of the target resource. e.g., post a message on a forum, or complete an online transaction.
- **PUT**: Requests that the target resource creates or updates its state with the state defined by the representation enclosed in the request.
- **DELETE**: Requests that the target resource deletes its state.

HTTP Protocol

Web Services

A web service is a simply a service that is built using web technologies, and which serves up content using web protocols and data formats.

We are using HTTP as the basis for a more generalized service protocol that can serve up a broader range of data than just HTML.

A service can be written in almost any language. The web server e.g. Apache, nginx, handles the actual request and delegate work. This is just an extension of what web servers were originally designed to do.

We are also leveraging the ability of web servers to handle HTTP requests efficiently, with the ability to scale to very large numbers of requests.

HTTP Protocol

REST

Representational State Transfer (REST), is a software architectural style that defines a set of constraints for how the architecture of an Internet-scale system, such as the Web, should behave.

REST was created by Roy Fielding in his doctoral dissertation in 2000 [1].

- It has been widely adopted and is considered the standard for managing stateless interfaces for service-based systems.
- The term “RESTful Services” is commonly used to describe services built using standard web technologies that adheres to these design principles.

HTTP Protocol

REST Principles

- **Client-Server**. By splitting responsibility into a client and service, we decouple our interface and allow for greater flexibility.
- **Layered System**. The client has no awareness of how the service is provided, and we may have multiple layers of responsibility on the server. i.e. we may have multiple servers behind the scenes.
- **Stateless**. The service does not retain state i.e. it's idempotent. Every request that is sent is handled independently of previous requests. That does not mean that we cannot store data in a backing database, it just means that requests are processed individually.
- **Cacheable**. With stateless servers, the client has the ability to cache responses under certain circumstances which can improve performance.
- **Uniform Interface**. Our interface is consistent and well-documented. Using the guidelines below, we can be assured of consistent behaviour.

HTTP Protocol

Applying REST

For your service, you define one or more HTTP endpoints (URLs). Think of an endpoint as a function that you interact with to make a request to the server.

- e.g., https://localhost:8080/messages
- e.g., https://cs.uwaterloo.ca/asis

To use a service, you format and send a request using one of these request types and send that request to an endpoint.

- GET: Use this method to read data. Such requests are safe and idempotent.
- POST: Use this method to store data i.e. create a new record in the database, or underlying data model.
- PUT: Use this method to update existing data.
- DELETE: Use this method to delete existing data.

HTTP Protocol

HTTP Response

The HTTP response needs to return data in a machine-readable format, that can be transmitted as part of the response body (UTF-8). JSON is often used as a standard data format.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 51
{
“status”: “success”,
“message”: “Data retrieved successfully”
}
```

Figure 4: An HTTP response, in JSON format.

Ktor

What is Ktor?

Ktor is an application framework for building networked applications. It's developed and supported by JetBrains alongside Kotlin.

You can use it for anything network and service related.

- e.g., fetch a web page
- e.g., connect to a service using HTTP requests (GET, POST).

You can use it:

- on the client side i.e., to make application requests, or
- on the server side to build a web service which handles GET/POST requests for clients.

What is Ktor?

How do we make requests to a service? Kotlin includes libraries that allow you to structure and execute requests from within your application.

This is all is takes to fetch the results from a simple GET request:

```
  val response = URL("https://google.com").readText()
  println(response)
```

Results:

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width />
  <meta name="robots" content="index, follow" />
  <title>Course Description - CS 346 Winter 2026</title>
  <meta name="description" content="Introduction to full-stack
   application design and development." ... />
</head>
```

What is Ktor?

GET Request

The Ktor `HttpRequest` class uses a builder to let us supply as many optional parameters as we need. Here's a simple GET example:

```
fun get(): String {
  val client = HttpClient.newBuilder().build()
  val request = HttpRequest.newBuilder()
        .uri(URI.create("http://127.0.0.1:8080"))
        .GET()
        .build()

  val response = client.send(request, HttpResponse.BodyHandlers)
  return response.body()
}
```

What is Ktor?

POST Request

Here's a POST method that sends an instance of our Message class to the service that we've defined and returns the response. We use serialization to encode it as JSON.

```
fun post(message: Message): String {
  val string = Json.encodeToString(message)
  val client = HttpClient.newBuilder().build()
  val request = HttpRequest.newBuilder()
        .uri(URI.create("http://127.0.0.1:8080"))
        .header("Content-Type", "application/json")
        .POST(HttpRequest.BodyPublishers.ofString(string))
        .build()

  val response = client.send(request, HttpResponse.BodyHandlers);
  return response.body()
}
```

What is Ktor?

Processing JSON

The web API returns a response, where the body is a JSON object This is from the Domain lecture (`courses` demo).

```
val result = CoroutineScope(IO).async {
  query("https://openapi.data.uwaterloo.ca/$term/$courseID")
}
val response = result.await()

if (response.status == HttpStatusCode.OK) {
  val body = response.body<String>()
  Json.decodeFromString<List<SectionDto>>(body)
} else {
  emptyList<SectionDto>()
}
```

Bibliography

[1] R. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” 2000. [Online]. Available: https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf