REST and intro to NodeJs

Learning objectives

- Upon successfully completing this section, you will be able to:
  1. Explain how to structure data using JSON
  2. Explain the structure of a URL
  3. Explain the concept and operation of a web API
  4. Explain the concept of RESTful APIs
  5. Design an API URL with path and query parameters
  6. Create and test a NodeJs / Express API

Communication (http)

Public internet
Azure private network
Docker private network
Virtual machine
Application container
Rabbitmq combined container

80 / 443
kong
8445
public IP:
private IP:

/categories
/question/:category
/moderate
/categories
/categories
/submit
/docs

4000
question
3000
/categories

4100
moderate
3100
/categories
submitted question

4200
submit
3200
categories

4002
MongoDB
27017

4002
MySQL
3306
mysql
vol

categories

4102
rmq-admin
15672

4201
SUBMITTED_QUESTIONS
5672

4202
rmq-admin
15672

public IP:
private IP:

mongo
vol

4001
etl
3001

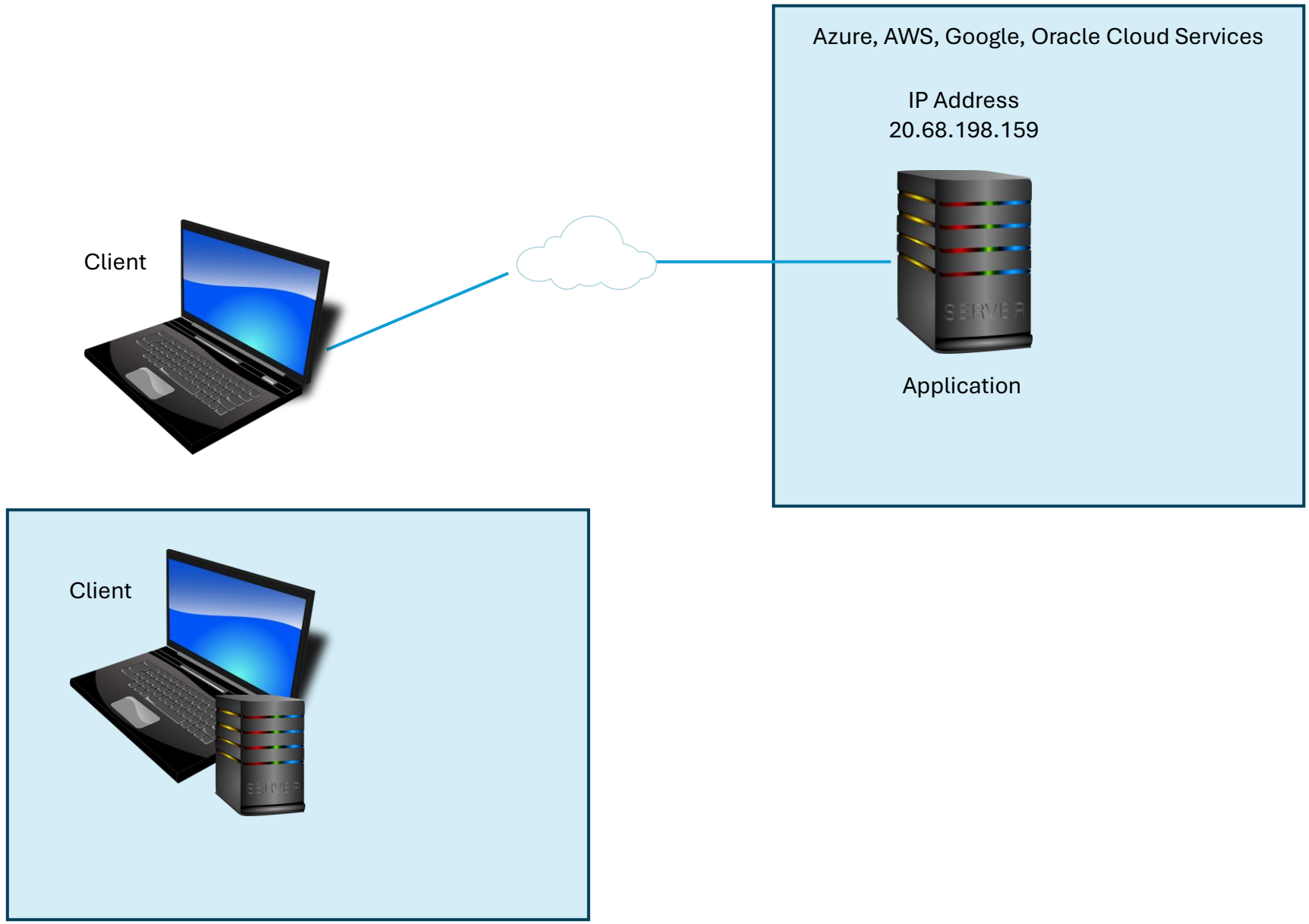
moderated questions

4101
MODERATED_QUESTIONS
5672

public IP:
private IP:

public IP:
private IP:

Container port number
Application port number


Azure, AWS, Google, Oracle Cloud Services
IP Address
20.68.198.159
Application
Client
Client

Communication using http

- Monolithic (single app) and distributed (multiple sub-apps), web apps need network communication
- http is the most popular protocol used to get data from ‘a’ to ‘b’, machine to machine
  The data structure passed from ‘a’ to ‘b’ is typically **H**yper**T**ext **M**arkup **L**anguage (HTML), text, e**X**tensible **M**arkup **L**anguage (XML) or **J**ava**S**cript **O**bject **N**otation (JSON)
- If you use JavaScript for the backend too, then json is ideal for data transfer in both directions
- The key thing is that these data formats are all text – i.e. characters. This avoids incompatibility between different programming languages, operating systems, processor types or whatever as they all support a basic character set, so data transferred as characters is ubiquitous (there are exceptions)

JSON – JavaScript Object Notation

ID	First Name	Last Name	Age
1	Jim	Smith	35
2	Joe	Soap	52
3	Jane	Doe	55

- Syntax
  Data is comma delimited **name:value** (sometimes called key:value) pairs. E.g. “firstName”:”John”
- We need the double quotes for anything other than numeric values
- JSON equivalent of the table discussed previously:

```
[
  { "ID" : 1, "firstName":"Jim", "lastName":"Smith", "age":35},
  { "ID" : 2, "firstName":"Joe", "lastName":"Soap", "age":52},
  { "ID" : 3, "firstName":"Jane", "lastName":"Doe", "age":55}
]
```

- Syntax is native JavaScript, almost no conversion is necessary to send or receive and interpret the data. Structure at both ends still needs to be known. The example will map directly to an array of objects
- JSON syntax diagrams are clearly explained here: **json.org**

Web API

- Typically use http or https to get a web page
  - http://20.26.222.217/index.html, or https://great-site.com/start.html
  - An API is an application programming interface. i.e. a well-defined interface contract (e.g. function name, parameters, types) enabling one program to use functionality of another - e.g. calling Windows OS API functions

```
#include <windows.h>
int main() {
  PlaySound(TEXT("C:\\Windows\\Media\\my_sound.wav"), NULL, SND_ASYNC);
  return 0;
}
```

  - Windows is written in C & C++, but can be called from other languages such as Python by mapping a python function call and types to the Windows C call based on contract
  - A web API is similar but the contract is with a remote server. The contract this time is passed in a url but the principle is the same

URL

- scheme://host[:port][/path][?query] where [ ] indicates optional. Default port = 80 for http, 443 for https
  - http(s)://something.com/something.html
  - http(s)://someAPI.com:3000/weather?city=preston&country=uk
  - http(s)://someAPI.com:3000/weather/{country}/{city}?hours=24&type=json
  - scheme and domain are case insensitive, but the rest should be case sensitive
  - query parameter is search or filter term
  - you can use 0-9, Aa-Zz, -, ., _ and ~ Other chars are for some form of control and should be escaped with %
  - to escape, use % followed by the hexadecimal character code. For example, you are not allowed a space character in a url. If you want one, you would use %20 which is the hex char code for space
  - e.g. quaint café would need to be written quaint%20caf%C3%A9
    - utf8 charset is here: https://www.fileformat.info/info/charset/UTF-8/list.htm
- The browser will make a request to the server, the server returns a response; this is ***synchronous*** - browser has to wait
- Could return html for rendering in a browser or data, typically in json format
- Web api example (not very good RESTful url):
  - http://universities.hipolabs.com/search?country=united%20kingdom
  - http://universities.hipolabs.com/search?country=united%20kingdom&name=university%20of%20central%20lancashire

So how does a web API work?

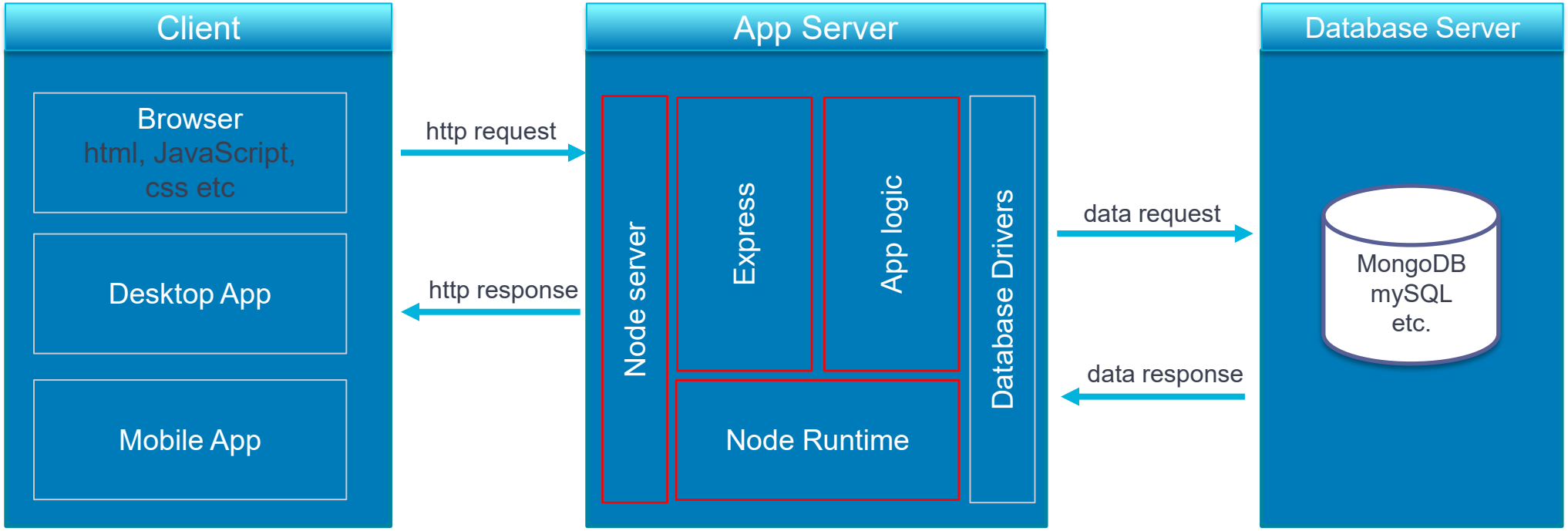
- A server at IP address, e.g., 20.123.80.240 listens for a request on a specific port number
- A url is used to form the contract with the api
- Server parses the request … e.g. http://x.com:3000/books/{type}/{author}?count=20
- Node.JS is a JavaScript runtime environment
- To deal with restful APIs, we need a framework to avoid us having to write all this parsing and routing code, so we can install express.js. This has been the de facto framework for Node for many years; other frameworks are available …
- Frameworks and libraries avoid you reinventing the wheel. You may have your favourite, or your organisation may stick to one for consistency. It really doesn't matter

- REST is currently the most popular approach to web API calls so we will focus on that (legacy may use SOAP)
- REST (REpresentational State Transfer) was introduced in 2000 by Roy Fielding in his PhD. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf
- HTTP has 9 methods. These are not methods in the function() sense, but just tokens the client can sent to indicate a required action so are sometimes called http verbs
- Of the 9 http methods, we typically use five: GET to get data, POST to submit data – e.g. create a new row in a relational database, PUT to update **all** columns in a row, PATCH to update some of the columns, and DELETE to delete a whole row - this enables us to implement all CRUD actions on a database
- These methods don't do anything other than inform the server of what action the client is requesting
- An http request contains headers and sometimes a body. The body carries the "payload" i.e. data being sent or returned. The header is more about control or additional information other than the data we are sending or receiving
- We don't need to worry about many of the headers but just to ensure you know what I'm talking about when I say "http header" a few examples are:
  - cookies: small text files that are used to remember state and other things
  - content-type: e.g. application/json if that's the format of the payload to inform the client
  - content-length: number of bytes in the body
  - user-agent: which web browser you are using
  - last-modified: date the file was modified. If server has same, it returns 304 and no data. i.e. no change so use cached version

Terminology

- **URL**: full address used to access a resource. e.g., http://api.example.com:3000/users/123?count=10
- **Origin**: the scheme, domain and port. e.g., http://api.example.com:3000
- **Path**: the part of the URL that comes after the domain. e.g., /users/123 123 is a path parameter
- **Query String**: acts as a filter or limiter of returned data. e.g., ?count=10
- **Endpoint**: a specific URL (or path) that represents a resource or action in an API. Usually with method
  - e.g. GET /users/123 or DELETE /users/123 are different endpoints
- **Route**: In backend frameworks (like Express.js, Spring boot, Flask), a route is a **server-side mapping** between a URL pattern and a function or controller

Architecture



- So we need Node.js to run the JavaScript
- We need Express.js to simplify creating the server and to process requests and return responses
- We need to write some logic – i.e. our app and data access and processing
- We need an appropriate database driver / SDK to enable our code to access the database
- We will focus on the areas in red first - the app server to create web APIs

Node.JS and NPM intro

- Node is just a runtime environment so doesn't have a graphical UI as it is more of a machine to machine type environment. We can see results in the console window of what is returned to the browser
- console.log() is the method to output to the terminal
- Rather than bloating the app with loads of libraries you may not use, you can import what you need
- For example, if we want functionality to parse file paths, we can use the node built-in module 'path'
  - const path = require('path');
- If you require functionality that isn't included with node, you can install packages created by others who share their code via the node package manager (npm)
- npm documentation is at https://docs.npmjs.com/
- Search packages at https://www.npmjs.com/

Create App Server and API

- When creating a new project that depends on external packages, those dependencies are recorded in a package.json file. You can create that manually but it's easier to get npm to set it up before installing any
  - Enter **npm init** then answer the questions. You can stick to the defaults – just press enter. You can edit later - or use **npm init -y** to answer yes to everything
  - This way, when storing your code in say git hub, or passing it to someone else, you don't need to save all the packages as they are recorded in the package.json file for npm to install later
- You will notice the packages.json and node_modules appear in the file explorer. There can be quite a lot of packages as the packages may themselves include packages
- If you have just the source files, say clone from git, type **npm install** on its own and npm will read package.json and install all the dependencies listed in that file

demos/api/minimalAPI

demos/API/firstAPI/app.js

demos/API/simpleAPI/app.js