Backend Developer Roadtrip

Random Collection of Knowledge

EDUARDO CARDENAZ

Part I: Basics

1 APIs

Application Programming Interface (API) An API is basically an intermediary that allows two applications to talk to each other.

It's useful to think of API communication in terms of requests and responses between a client and a server. The application submitting the request is the client, and the server provides the response.

API Specification Is a document or standard that describes how to build or how to use an API. A system that meets this standard its said to be *implementing* or *exposing* an API. The term API can refer to both the implementation and the specification.

1.1 RESTful APIs

REST stands for Representational State Transfer, and it's an architectural style for designing networked applications. A RESTful API (Application Programming Interface) adheres to the principles of REST.

"The design rationale behind the Web architecture can be described by an architectural style consisting of the set of constraints applied to elements within the architecture." source

To understand REST, we'll be expanding and building on top of each constraint that composes this architecture.

Violating any constraint other than Code on Demand means that the service is not strictly RESTful.

1.1.1 Uniform Interface

Resource Based Individual resources are identified using URIs as resource identifiers. The resources themselves are conceptually different from the *representation* that is returned to the client. For example, the server doesn't return its database but rather some HTML, JSON or XML that represents some database records.

Manipulation of Resources Through Representations When a client holds a representation of a resource given by the server, including any metadata attached to it, it should have enough information to modify or delete said resource on the server, given it has the right permissions.

Self-Descriptive Messages Each message includes all the necessary information to describe how to handle that message.

1.1.2 Client-Server

Separation of Concerns is the principle behind this constraint. By separating the user interface (UI) concerns from the data storage concerns, we improve the portability of the UI across multiple platforms.

1.1.3 Stateless

This constraint stablish that the interaction between the client and the server must be stateless in nature, by that, meaning that any request made by the client must contain all the necessary information so that the server can understand the request. It shall not take advantage of any stored context.

Stateless Imagine that you're buying coffee at a shop. Each time you want to order a coffee you need to tell the cashier exactly what you want and how you want it. The cashier doesn't remember you or what you ordered the last time you went there. Each visit is like starting from scratch.

Stateful The cashier remembers you and remember what you usually ask for as you're a frequent client. So, you could say 'the usual' and they'd know exactly what you want.

1.1.4 Cacheable

Clients should can cache responses. Responses must, implicitly or explicitly, define themselves as cacheable or not. Clients should be able to negotiate wether to cache or not to prevent reusing stale or inappropriate data in response to further requests.

1.1.5 Layered System

A client cannot tell whether its connected to the main server or an intermediary along the way. The layered system style allows an application to be composed of hierarchical layers by constraining component behavior such that each component can't see beyond the immediate layer with which they are interacting.

1.1.6 Code on Demand

This is a kind-of unique thing about RESTful systems. Code on Demand is optional. Basically means that a server can temporarily extend functionality to a client by transferring logic to the client. As an example, a request can return client-side scripts such as Javascript code.

1.2 JSON APIs

1.3 SOAP APIs

1.4 GraphQL APIs

1.5 gRPC APIs

1.6 Authentication & Authorization

Authentication is the process that an individual, process or system goes through to **prove** their identity before gaining access to digital systems.

Authorization is the process of determining whether a user, system or application has the necessary permissions to perform a particular action within a system.

API Authentication validates the identity of the client attempting to make a connection by using an authentication protocol. The protocol sends the credentials from the remote client requesting access to the server for verification, usually the credentials are in either plain text or some form of encrypted format.

1.6.1 Session Authentication

Session A session, refers to a temporary, stateful interaction between a user and a server. A session, is a period of interaction between a user an a server, during which the user's actions and data are temporarily stored and managed by the server.

- **Duration**: A session begins when the user logs into the server and ends when they log out, or after a period of inactivity.
- **Session Data**: During the sessions, the server stores information relevant to the user's interaction, such as their authentication status, user ID, preferences, etc.
- **Session ID**: To uniquely identify each sessions, the server assigns a session identifier. This session id is used to to associate the Session Data with the User

Example When you log in (authenticate) into a web application, the server creates a session and then keeps track of it itself. Then it creates a Session ID and gives it to you, subsequently, the client (you) pass this Session ID to the server with each request. Then, the Server looks this Session ID up in its Session Log, and if it finds it, it knows who you are and what you're allowed to do.

How the client passes the Session ID to the server, depends on the implementation, but it's usually done through Cookies.

1.6.2 JWT

JWT or JSON Web Token, is a compact, URL-Safe means of representing claims between two parties. The claims in a JWT are encoded as JSON objects.

A JWT consists of three parts:

- **Header**: Contains the type of token and the signing algorithm being use. (e.g. HMAC SHA256, or RSA).
- **Payload**: Contains the Claims. Claims are statements about an entity (typically the user) and additional data.
- **Signature**: The signature is created by taking the combined encoded header and payload, and signing it using the algorithm specified in the header and a secret key or private key.

Claims In the context of JWT, claims refer to pieces of information that are included in the token. These pieces of information are **statements** about an entity. Claims are stored in the payload part of the JWT.

There are three types of Claims.

- **Registered Claims**: These are part of a set of predefined claims which are not mandatory but recommended to provide a set of useful, interoperable claims, like:
  - **iss** (issuer): Identifies the principal that issued the JWT.
  - **sub** (subject): Identifies the principal that is the subject of the JWT.
  - **aud** (audience): Identifies the recipients that the JWT is intended for.
  - **exp** (expiration time): Identifies the expiration time on or after which the JWT must not be accepted for processing.
  - **nbf** (not before): Identifies the time before which the JWT must not be accepted for processing.
  - **iat** (issued at): Identifies the time at which the JWT was issued.
  - **jti** (JWT ID): Provides a unique identifier for the JWT.
- **Public Claims**: They can be defined by developers to include custom information. These claims can be used publicly, meaning that they can be understood by anyone who processes the JWT.
- These are custom claims created to share information between parties that agree on using them. They are neither registered nor public claims, and they are meant to be used within a specific context between the parties involved.

1.6.3 Basic Auth

1.6.4 Token Auth

1.6.5 OAuth

1.6.6 Cookie Based

1.6.7 OpenID & SAML

HTTP

Hypertext Transfer Protocol (HTTP) Is a protocol for fetching resources over the internet. It is the foundation of the Web and its a client-server protocol. Clients, usually a web browser, and servers communicate by exchanging messages. The messages send by the client are called Requests and the messages sent by the server are called Responses.

Client-Server Architecture HTTP operates on a client-server model where the client (usually a web browser) send a request to the server, and the server responds with the requested resource (e.g. an HTML page, image, or JSON data)

Stateless Protocol Each HTTP request from a client to a server is independent; it carries all the information the server needs to fulfill the request. This stateless nature simplifies server design bc it doesn't need to retain information about previous requests.

1.7 HTTP Methods & Status Codes

1.7.1 HTTP Methods

GET Requests a representation of a specified resource. GET requests should only retrieve data.

POST Submits data to be processed to a specified resource, often causing a change in state or side effects on the server.

PUT Replaces all current representations of the target resource with the request payload.

DELETE Removes the specified resource.

PATCH Applies partial modifications to a resource.

1.7.2 HTTP Status Codes

Most common status codes.

1xx Informational

2xx Success

- 200 OK: The request has succeeded.
- 201 Created: The request succeeded, and a new resource was created as a result. This is typically the response for a POST request.
- 204 No Content: The request was successful, but there is no content tos end in the response.

3xx Redirection

- 301 Moved Permanently: The requested resource has been permanently moved to a new location.
- 302 Found: The requested resource has been temporarily moved to a new location.

4xx Client Errors

- 400 Bad Request: The server cannot process the request due to a client error.
- 401 Unauthorized: The client must authenticate itself to get the requested response.
- 403 Forbidden: The client does not have permission to access the requested resource.
- 404 Not Found: The server cannot find the requested resource.

5xx Server Errors

- 500 Internal Server Error: The server has encountered a situation it doesn't know how to handle.
- 502 Bad Gateway: The server, while acting as a gateway or proxy, received an invalid response from the upstream server. Meaning the server is down or being upgraded.
- 503 Service Unavailable: The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded.
- 504 Gateway Timeout: The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server.

1.8 Common HTTP Headers

HTTP Headers are key-value pairs sent between the client and the server in an HTTP request or response. They provide essential information about the request, the resposne, and the data being transferred. Examples of the most common headers include:

General Headers

- **Cache-Control**: Specifies directives for caching mechanisms in both requests and responses. Examples of directives include *no-cache*, *no-store*, *max-age*, and *must-revalidate*.
- **Connection**: Controls whether the network connection stays open after the current transaction finishes. Examples of values include *keep-alive* and *close*. Keep-alive allows the connection to be reused for multiple requests. Close forces the connection to close after the current request.
- **Date**: The date and time at which the message was sent.

Request Headers

- **Accept**: Specifies the media types that the client can understand. The server should respond with one of these media types. Example values include *text/html*, *application/json*, and *image/png*.
- **Accept-Encoding**: Specifies the encoding algorithms that the client can understand. The server should respond with one of these encoding algorithms. Example values include *gzip*, *deflate*, and *br*.
- **Accept-Language**: Specifies the natural languages that the client can understand. The server should respond with one of these languages. Example values include *en-US*, *es*, and *fr*.
- **Authorization**: Contains credentials for authenticating the client with the server.
- **User-Agent**: Contains information about the client making the request. This can include the client's operating system, browser, and version.
- **Content-Type**: Specifies the media type of the data being sent in the request.

Response Headers

- **Content-Type**: Specifies the media type of the data being sent in the response.
- **Content-Length**: Specifies the length of the response body in octets (8-bit bytes).
- **Set-Cookie**: Sets a cookie in the client's browser. Cookies are used to store information about the client's session.
- **Location**: Specifies the URL to redirect the client to. This is often used in 3xx redirection responses.
- **WWW-Authenticate**: Specifies the authentication method that the client must use to access the resource.

1.8.1 Content-Type

This header is a critical component that indicates the media type of the resource being sent. It tells the client or the server how to interpret the data in the body of the request or response.

The *Content-Type* header consists of:

- **Media Type**: The type of data being sent (e.g., text, image, application).
  - *text*: Represents textual information. Examples include HTML, CSS, and plain text.
  - *image*: Represents image data. Examples include PNG, JPEG, and GIF.
  - *application*: media types are used to denote formats that are usually processed by applications rather than directly read by humans. These types often require specific software to interpret or manipulate the data.
- **Subtype**: The specific format of the data (e.g., HTML, JSON, PNG).
- **Optional Parameters**: Additional information about the media type, often specifying the character encoding (e.g., charset=UTF-8).

application/x-www-form-urlencoded This is the default encoding for form data. It encodes form fields as key-value pairs, with special characters replaced by escape sequences. This encoding is commonly used in HTML forms and URL query strings. Note that the space character is encoded as *%20* or the plus sign (+).

```
name=John+Doe&age=25
```

application/json Commonly used in API requests and responses to transfer JSON data.

```
{
    "name": "John Doe",
    "age": 25
}
```

application/xml Used to transfer XML data.

```
<person>
    <name>John Doe</name>
    <age>25</age>
</person>
```

1.8.2 Content-Length

This headers specifies the size of the body of the request or response, measured in bytes. It is used by the client and server to understand the exact size of teh content being sent or received.

1.8.3 User-Agent

This headers provides a way to identify the client software that is making the request. This ehader contains a string that provides details about the user agent (browser, application, or other software) and the operating system it is running on. This information can be used by the server for various purposes, such as content negotiation, analytics and logging.

1.8.4 Authorization

This header is used to proved credentials that atuhenticate the client to the server. This header allows the server to verify the identity of the client and grant acces to protected resources. The header supports various authentication schemes.

- Basic Authentication: The client sends a base64-encoded username and password separated by a colon. The server decodes the credentials and verifies them against a user database.
- Bearer Token: The client sends a token that was previously obtained from the server. The server validates the token to authenticate the client.
- Digest Authentication: The client sends a hashed value of the username, password, and other information. The server verifies the hash to authenticate the client.
- OAuth: The client sends an access token that was previously obtained from an OAuth server. The server validates the token to authenticate the client.
- API Key: The client sends an API key that was previously obtained from the server. The server validates the key to authenticate the client.
- JWT: The client sends a JSON Web Token (JWT) that was previously obtained from the server. The server validates the token to authenticate the client.

Usually, the authorization header is in the format *Authorization : <type> <credentials>* for example:

```
Authorization: Bearer eyHstOSkasd...
```

1.8.5 Cache-Control

1.8.6 Cookie

1.8.7 Set-Cookie

1.8.8 Accept

1.8.9 Accept-Encoding

1.8.10 Host

1.8.11 Referer

1.9 Unicode & Character Encoding

UTF-8

ISO-8859-1

1.9.1 URL Encoding

1.10 CORS

Origins

Headers

Methods

Credentials

Preflight Requests

1.11 WebSockets

Protocol

Handshake

Frames

1.12 HTTP Versions

HTTP/1.1

HTTP/2

HTTP/3

1.13 HTTPS

TLS/SSL

Certificates

Handshake

1.14 HTTP Cookies

Creation

Attributes

SameSite

Secure

HttpOnly

2 Networking and Shit

2.1 The OSI Model

The Open Systems Interconnection model is a conceptual framework used to understand and implement standardized network protocols in seven layers. Each layer serves a specific function and communicates with the layers directly above and below it.

IP Addresses Unique identifier assigned to each device connected to a network. It allows devices to locate and communicate with each other.

- Public IP: Used on the internet.
- Private IP: Used within private networks (e.g., home or office)

Ports A port is a logical endpoint for communication, used to distinguish different types of network services running on the same IP address (device).

- Well-Known Ports (0-1023): Assigned to widely used services and protocols.
- Registered Ports (1024-49151): Assigned by IANA for specific services.
- Dynamic/Private Ports (49152-65535): Used for dynamic or temporary purposes.

Domain Name System (DNS) The DNS translates human-readable domain names (e.g., www.example.com) into IP addresses that computers use to identify each other on the network.

2.1.1 Application Layer

It s the topmost layer of the OSI model. It serves as the interface between the network services and the end-user applications. This layer provides the necessary protocols and functionalities for application to communicate over the network, enabling end-users to interact with networked systems.

Key Functions

- Network Services: Provides services like file transfers, email, remote login and web browsing.
- Data representation: Ensures data is presented in a readable format.
- Communication: Facilitates communication between software applicaitons and network services.

Common Protocols

- HTTP (Hypertext Transfer Protocol): Used for web browsing. Usually runs on port 80.
- HTTPS: Secure version of HTTP, using SSL/TLS encryption. Usually runs on port 443.
- FTP (File Transfer Protocol): Used for transferring files between systems. Usually runs on port 21.
- SMTP (Simple Mail Transfer Protocol): Used for sending emails. Usually runs on port 25.
- DNS (Domain Name System): Translates domain names to IP addresses. Usually runs on port 53.

Example Scenario Accessing a Website.

1. URL Input: You enter ‘www.example.com‘ in your browser.
2. DNS Lookup: the browser send a DNS query to find the IP address of ‘www.example.com‘. The DNS server replies with ‘93.184.216.34‘
3. Sending a Request: The browser sends an HTTP request to ‘93.184.216.34‘ on port 80.
4. Server Response: The web server listens on port 80, processes the request, and sends back the webpage content.
5. Receiving Data: The browser receives the data and renders the webpage for you to view.

2.1.2 Presentation Layer

The Presentation Layer is the sixth layer of the OSI model. It acts as a translator between the application layer and the lower layers of the OSI model. This layer is responsible for the syntax and the semantics of the data being transmitted, ensuring that the data is readable by the receiving system.

Key Functions

- Data Translation: Converts data both incoming to the appplication and outgoing from the application into a format that can be sent over the network.
- Data Encryption and Decryption: Encrypts data before it is sent over the network to ensure secure comms.
- Data Compression: Compresses and decompresses the data to reduce the size of the data being transmitted.
- Serialization and Deserialization: Converts data into a format that can be stored or transmitted and then converts it back to its original form.

Common Protocols

- SSL/TLS: Secure Sockets Layer and Transport Layer Security are protocols used to encrypt data transmitted over the network. Most commonly used for HTTPS.
- ASCII: American Standard Code for Information Interchange is a character encoding standard used for text files in computers and other devices.
- JPEG: Joint Photographic Experts Group is a standard for compressing images.

This layers is also able to provide encryption and compression if the application layer ask it to do so.

2.1.3 Session Layer

The Session Layer is the fifth layer of the OSI model. It is the responsible for managing and controlling the dialogs (sessions) between two devices. This layer establishes, maintains, and terminates connections between applications, ensuring that the data is properly synchronized and delivered.

Common Protocols

- NetBIOS: Network Basic Input/Output System is a protocol used for communication between computers on a local area network.
- PPTP: Point-to-Point Tunneling Protocol is used to create a virtual private network (VPN) over the internet.
- RPC: Remote Procedure Call is a protocol that allows a computer program to cause a subroutine or procedure to execute in another address space.

2.1.4 Transport Layer

The Transport Layer is the fourth layer of the OSI model. It is responsible for end-to-end communication between the source and destination devices. This layer ensures that data is delivered error-free, in sequence, and without loss or duplication.

Key Functions

- Segmentation and Reassembly: Breaks down large data into smaller segments for transmission and reassembles them at the destination.
- Error Detection and Correction: Detects errors in the data and retransmits it if necessary.
- Flow Control: Manages the flow of data between devices to prevent data loss due to congestion.

Common Protocols

- TCP (Transmission Control Protocol): Provides reliable, connection-oriented communication between devices. It ensures that data is delivered in the correct order and without errors.
- UDP (User Datagram Protocol): Provides connectionless communication between devices. It is faster than TCP but does not guarantee delivery or order of data.

2.1.5 Network Layer

The Network Layer is the third layer of the OSI model. It is responsible for routing data from the source to the destination device. This layer determines the best path for data to travel and handles addressing, packet forwarding, and routing.

Key Functions

- Logical Addressing: Assigns logical addresses (IP addresses) to devices on the network.
- Routing: Determines the best path for data to travel from the source to the destination device.
- Packet Forwarding: Forwards data packets from one device to another based on the destination address.

Common Protocols

- IP (Internet Protocol): Provides logical addressing and routing of data packets between devices on the network.
- ICMP (Internet Control Message Protocol): Used for error reporting and diagnostic functions in IP networks.
- ARP (Address Resolution Protocol): Maps IP addresses to MAC addresses on a local network.
- DHCP (Dynamic Host Configuration Protocol): Assigns IP addresses to devices on a network automatically.

2.1.6 Data Link Layer

Defines the format of data on the network.

2.1.7 Physical Layer

Defines the physical connection between devices. Transmits raw bit stream over the physical medium.

2.2 TCP/IP

2.2.1 Transmission Control Protocol (TCP)

Is a network protocol that lets two host connect and exchange data streams. TCP guarantees that the packages and data that are sent, come in the same order as they were sent.

TCP is a connection-oriented protocol, meaning that it establishes a connection between the two hosts before sending data. It also ensures that the data is delivered error-free and in the correct order.

Connection-Oriented Communication TCP uses a three-way handshake to establish a connection between the two hosts. The three steps are:

1. SYN (Synchronize): Device A (the client) wants to start a conversation with Device B (the server).
2. SYN-ACK (Synchronize-Acknowledge): The server responds with a SYN-ACK packet to acknowledge the connection request.
3. ACK (Acknowledge): The client sends an ACK packet to confirm the connection.

When it wants to terminate a connection, it uses a four-way handshake.

1. FIN (Finish): The client sends a FIN packet to the server to close the connection.
2. ACK: The server responds with an ACK packet to acknowledge the request.
3. FIN: The server sends a FIN packet to the client to close the connection.
4. ACK: The client responds with an ACK packet to acknowledge the request.

Datagrams Are the basic unit of information passed across the internet. They are packets of data that are sent and received over a network.

A datagram is a self-contained, independent packet of data that is sent over a network. It contains the source and destination addresses, as well as the data being transferred.

Internet Protocol (IP) Is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet.

2.3 UDP

2.4 DNS

3 Caching

3.1 Client Side

3.2 Server Side

3.3 Content Delivery Network (CDN)

3.4 Redis

3.5 Memcached

4 Web Security

4.1 Hashing Algorithms

4.2 API Security Best Practices

5 Testing

5.1 Unit Testing

5.2 Integration Testing

5.3 Functional Testing

6 CI/CD

7 Databases

8 Software Design & Architecture

8.1 Design and Development Principles

8.1.1 Separation of Concerns

8.1.2 Reusability

8.1.3 Keep It Simple Stupid (KISS)

8.1.4 Don't Repit Yourself (DRY)

8.1.5 Scalability

8.1.6 Security

8.2 GOF Design Patterns

8.3 Domain Driven Design

8.4 CQRS

8.5 Event Sourcing

9 Architectural Patterns

10 Message Brokers

10.1 RabbitMQ

10.2 Kafka

11 Containerization Vs Virtualization

11.1 LXC

11.2 Docker

11.3 Kubernetes

11.4 Elasticsearch

11.5 Solr

12 Web Servers

12.1 Server Sent Events

12.2 WebSockets

12.3 Long Polling

12.4 Short Polling

13 GraphQL

13.1 Apollo

14 NoSQL Databases

14.1 Document DBs - MongoDB

14.2 Time Series - InfluxDB

14.3 Realtime - Firebase

14.4 Column DBs - Cassandra

14.5 Key Value - Redis

14.6 Graph DBs - Neo4j

15 Building for Scale

15.1 Difference + Usage

15.2 Mitigation Strategies

Part II: Infraestructure Knowledge

16 Go Programming Language

16.1

16.2

17 Networking and Protocols

17.1

17.2

18 Docker

Container A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

A container is a way to package our applications including all the dependencies that it needs, including its configuration files. This makes them portable. Making development and deployment much faster.

Image A Docker Image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

An image is a packaging containing all the dependencies and the code. This is what's gonna be shared. Finally a container is an image we configured which its going to have all the dependencies running, alongside with environment variables, etc. A container, is basically layers upon layers of images, where usually the bottom layer is a Linux image.

A container is an instance of an image.

18.1 Virtualization

At its core, Docker is a virtualization tool.

Virtualization Is the process of creating a virtual version of something, such as an operating system, a server, a storage device or network resources.

Host Being the physical machine that runs the virtualization software.

Guest The virtual machine or OS that runs on the host.

Layers of Virtualization The concept of virtualization is based on three layers.

- **Hardware**: Where the physical hardware is located.
- **Kernel**: The kernel is the core of the operating system. It's the bridge between the hardware and the software.
- **Application**: This is your software, the applications you run on your computer.

A Virtual Machine (VM) is a software that emulates a physical computer. It creates a virtualized environment that behaves like a separate computer. When you run a VM, the Kernel and the Application layers are what's being virtualized.

As you may realize, this is very resource intensive.

Docker , on the other hand, lives on the Application layer. It doesn't virtualize the hardware, nor the kernel. It just virtualizes the application. This makes it much more lightweight than a VM.

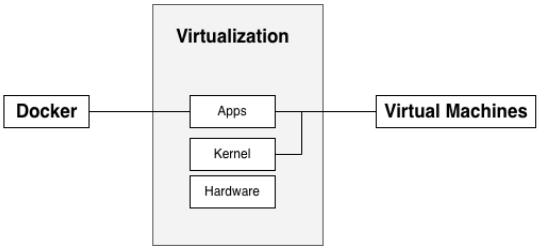

Figure 1: Docker vs VMs

Types of Virtualization Exists three types of virtualization:

- **Para-Virtualization**: In para-virtualization, the host is trying to give the guest most of their hardware.
- **Partial Virtualization**: Where some components of the host's hardware are virtualized and given to the guest.
- **Full Virtualization**: In full virtualization, the guest is given a full copy of the host's hardware.

For each of this cases, Docker is going to be superior, as Docker uses the kernel of the host directly.

18.2 Docker Network

Network A Network in simple terms is a group of two or more devices that can communicate with each other either physically or virtually.

Docker Network The Docker Network is a virtual network created by Docker to enable communication between Docker Containers. Container Networking refers to the ability for containers to connect to and communicate withe ach other, or to non-Docker workloads.

Containers have networking enabled by default, and they can make outgoing connections. A container has no information about what kind of network it's attached to, or whether their peers are also Docker workloads or not.

Port Mapping is a way to allow traffic to flow from the outside world into a container. Since containers are isolated environments, if you want to access a web application inside a container from your web browser, you need to map the port on which the application is running inside the container to a port on the host machine.

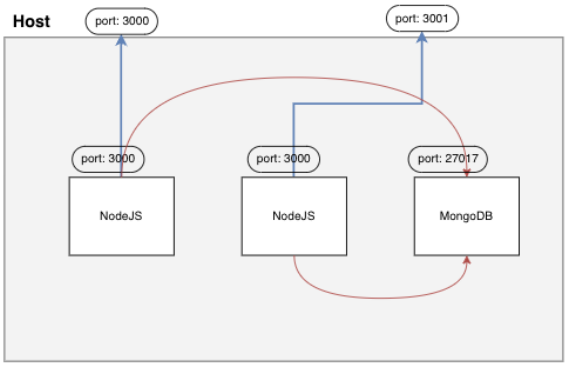

Figure 2: Docker Port Mapping

As per the figure above, the Node JS container is running on port 3000. To access this container from the host machine, we need to map the port 3000 of the container to a port on the host machine. In this case, we're mapping the port 3000 of the container to the port 3000 of the host machine.

Usually, you would find that in cases where you have multiple containers running on the same port, you would map the port of the container to a different port on the host machine. So, if you have two containers with Node JS each with a web application running on port 3000, you would map the port of the first container to port 3000 of the host machine, and the port of the second container to port 3001 of the host machine. Although this is not mandatory, it's a good practice to avoid conflicts.

18.3

18.4

19 Amazon Web Services

19.1

19.2

20 Terraform

20.1

20.2

21 Ansible

21.1

21.2

22 Github Actions

22.1

22.2