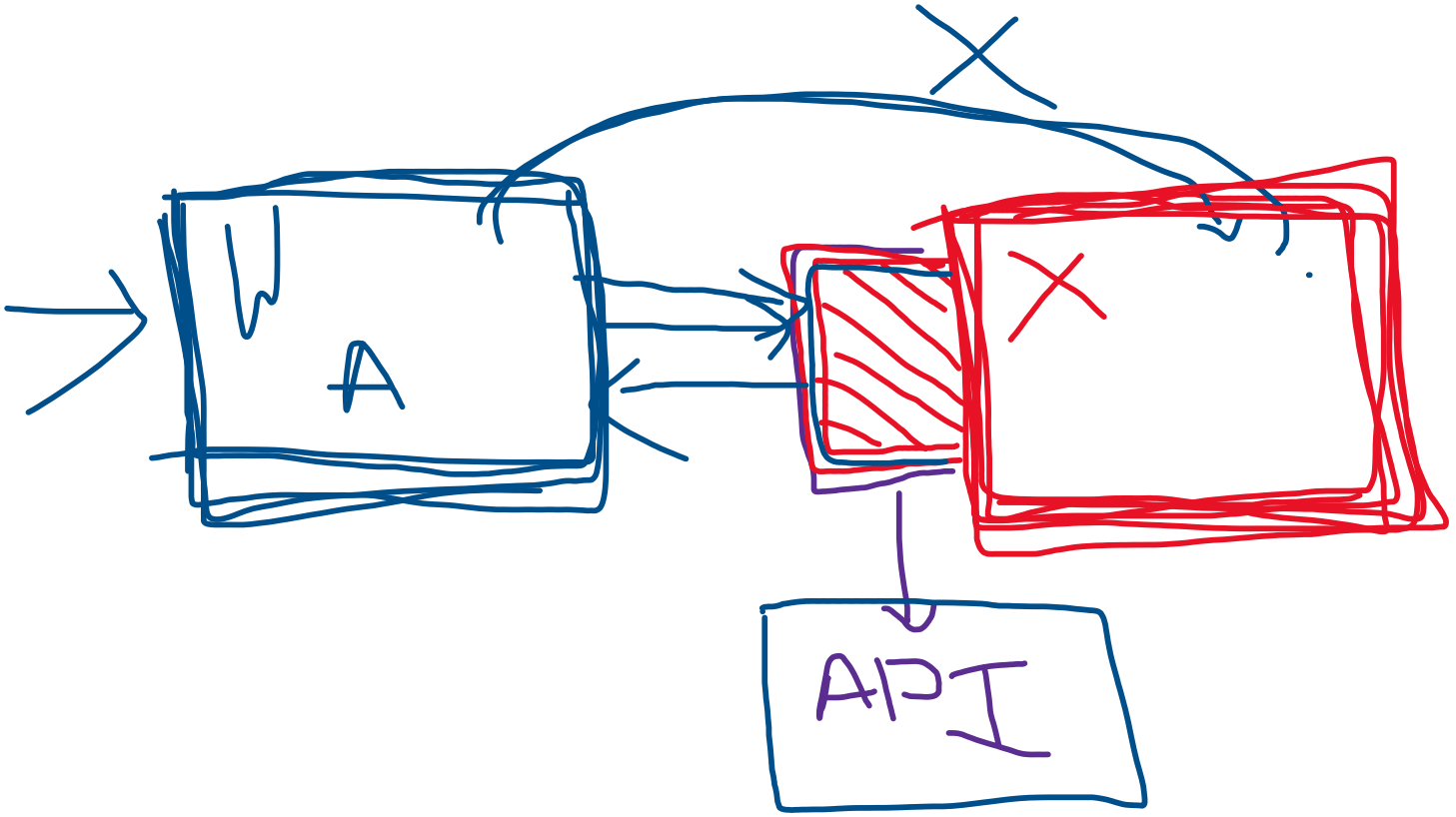
1. Definition

Sunday, January 19, 2025 7:51 AM

- An API (Application Programming Interface) is essentially a set of rules and protocols that allows different software applications to communicate with each other
- It can be thought of as a bridge that connects different systems or services, enabling them to share data and functionalities without directly interacting with each other's underlying code
- Ex: Weather app



2. Key Features

Sunday, January 19, 2025 8:01 AM

- Enable different systems, platforms, and applications to communicate and share data effortlessly
- Significantly reduce the time and effort required to develop applications by leveraging pre-built functionalities
- Enable systems to scale and adapt to growing or changing requirements
- APIs contribute to creating dynamic and responsive applications that offer better user experiences
- Allow organizations to expand their ecosystem and collaborate with external developers and partners
- APIs provide controlled and secure access to data, allowing organizations to share information with clients, partners, or the public
- APIs provide mechanisms to enforce security policies and ensure compliance with industry standards
- APIs can help organizations reduce operational and development costs

3. Types of APIs

Sunday, January 19, 2025 8:05 AM

1. **Web API:**
   - APIs that are accessible over the web using HTTP/HTTPS protocols
   - Allow applications to communicate over the internet
   - Data is usually shared in JSON/XML formats
   - Examples:
     - *Fetching weather data from OpenWeather API*
     - *Embedding Google Maps on a website*

2. **Library API:**
   - APIs provided by libraries or frameworks to expose specific functionality for developers
   - Allow developers to utilize predefined functions or methods without implementing them from scratch
   - Examples:
     - *NumPy API: Provides functions for numerical computations*
     - *TensorFlow API: Offers tools for building and training ML/DL models*
     - *Matplotlib API: Allows creating visualizations programmatically*

3. **Remote API:**
   - APIs designed to interact with systems located on a different network or server, often through the internet or intranet
   - Web API is a subset of Remote API, i.e., all Web APIs are Remote APIs, but not all Remote APIs are Web APIs
   - Makes use of Intranet APIs, which can only be accessible within a private network
   - Examples:
     - *Deploying virtual machines using AWS EC2 API*
     - *Retrieving remote files stored in Google Drive through its API*

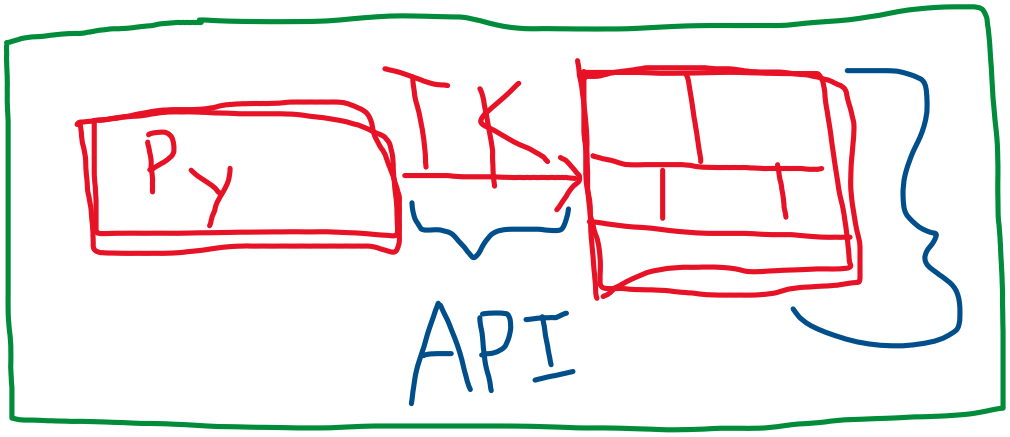
4. **Database API:**
   - APIs that allow applications to interact with databases
   - Provide a structured way to perform CRUD (Create, Read, Update, Delete) operations on a database
   - Examples:
     - *MySQL Connector API: Enables interaction with MySQL databases*
     - *MongoDB API: Allows CRUD operations on NoSQL data*
     - *Firebase Realtime Database API: Facilitates real-time database interactions*

5. **Hardware API:**
   - APIs that enable software to interact with physical hardware devices

- Provide an abstraction layer to control and retrieve data from devices without needing low-level programming
- Examples:
  - *GPU APIs like CUDA or OpenCL for performing parallel computations*
  - *APIs for IoT devices such as sensors or smart appliances*
  - *Using APIs to control drones or robots programmatically*

6. GUI API:

- APIs designed for building and interacting with graphical user interfaces
- Allow developers to create and manage GUI components programmatically
- Examples:
  - *Java Swing API: Provides tools to create desktop GUIs in Java*
  - *Tkinter: A Python library for creating GUI applications*
  - *Android SDK: Enables developers to build mobile app interfaces*



4. API Protocols

Sunday, January 19, 2025 8:31 AM

1. **REST:**
   - Abbreviation for Representational State Transfer
   - It's a lightweight architectural style that uses HTTP for communication
   - Follow a set of principles, such as statelessness and resource representation using URLs
   - Each API call is independent of the other
   - Common HTTP methods: GET, POST, PUT, DELETE
   - Use Cases: Web applications, mobile applications, and public APIs (e.g., Twitter API, GitHub API)

```
GET /users/123 HTTP/1.1
Host: example.com
```

```
{
  "id": 123,
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```

2. **SOAP:**
   - Abbreviation for Simple Object Access Protocol
   - A protocol that relies on XML messaging for communication
   - Designed for more structured and secure interactions
   - Has built-in error handling and security (e.g., WS-Security)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetUserDetails>
      <UserId>123</UserId>
    </GetUserDetails>
  </soap:Body>
</soap:Envelope>
```

3. **GraphQL:**
   - Allows clients to request only the data they need, reducing over-fetching and under-fetching
   - Clients specify the structure of the response
   - All queries are handled at one endpoint

```
{
  user(id: "123") {
    name
    email
    posts {
      title
      comments {
        text
      }
    }
  }
}
```

4. **gRPC:**
   - Abbreviation for Google Remote Procedure Call
   - A high-performance RPC framework by Google using Protocol Buffers (Protobuf) for message serialization
   - Operates over HTTP/2, providing bi-directional streaming
   - Works across various programming languages and minimizes payload size

```
service UserService {
  rpc GetUserDetails (UserRequest) returns (UserResponse);
}
message UserRequest {
  int32 user_id = 1;
}
message UserResponse {
  int32 user_id = 1;
  string name = 2;
  string email = 3;
}
```

5. **WebSocket:**
   - A protocol providing full-duplex communication over a single TCP connection
   - Enables continuous exchange of data without re-establishing the connection
   - Ideal for real-time use cases due to its low latency
   - Use Cases: Chat applications, online gaming, live data feeds

```
const socket = new WebSocket("ws://example.com/stocks");
socket.onmessage = (event) => {
  console.log("Stock update:", event.data);
};
```

Feature	REST	SOAP	GraphQL
Data Format	JSON, XML, etc.	XML only	JSON only
Flexibility	High	Low	Very High

Feature	REST	SOAP	GraphQL
Data Format	JSON, XML, etc.	XML only	JSON only
Flexibility	High	Low	Very High
Performance	Fast	Slower	Efficient
Use Case	Modern web APIs	Enterprise applications	Dynamic client needs

5. Working

Sunday, January 19, 2025 9:50 AM

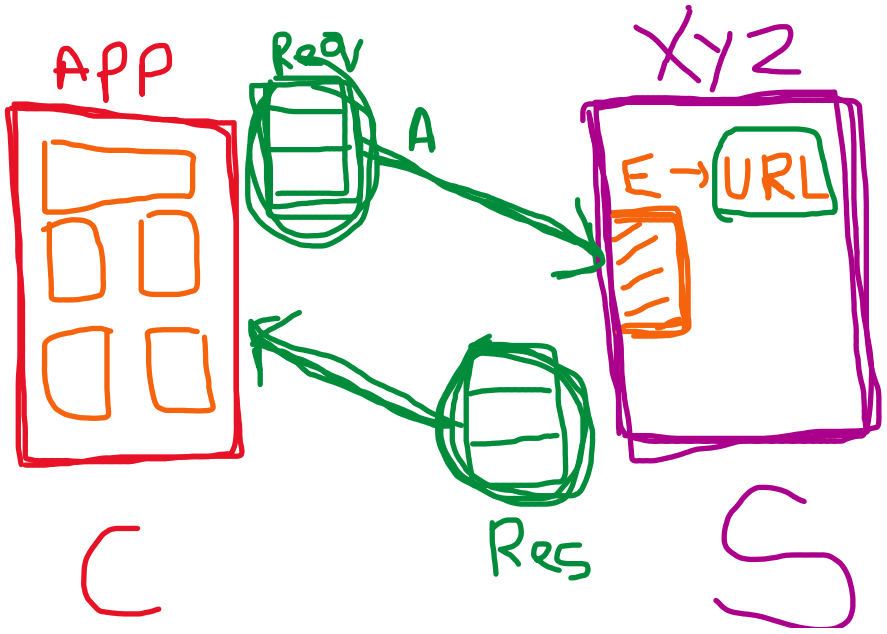
1. **Request Initiation:**
   - The process begins when a client (like a web browser, mobile app, etc.) initiates a request to the API
   - This request is usually made using HTTP methods such as GET, POST, PUT, or DELETE

2. **API Endpoint:**
   - The request is sent to a specific URL known as an API endpoint
   - This endpoint is like a door that the API has opened for requests

3. **Request Processing:**
   - The server receives the request and processes it
   - This involves verifying the request parameters, checking authentication, and accessing databases, etc.

4. **Response Generation:**
   - After processing the request, the server generates a response
   - This response typically includes the requested data or a confirmation of the action performed

5. **Response Delivery:**
   - The response is sent back to the client
   - The client receives the response and uses the data to perform the desired action or display information to the user



6. API Components

Sunday, January 19, 2025 10:08 AM

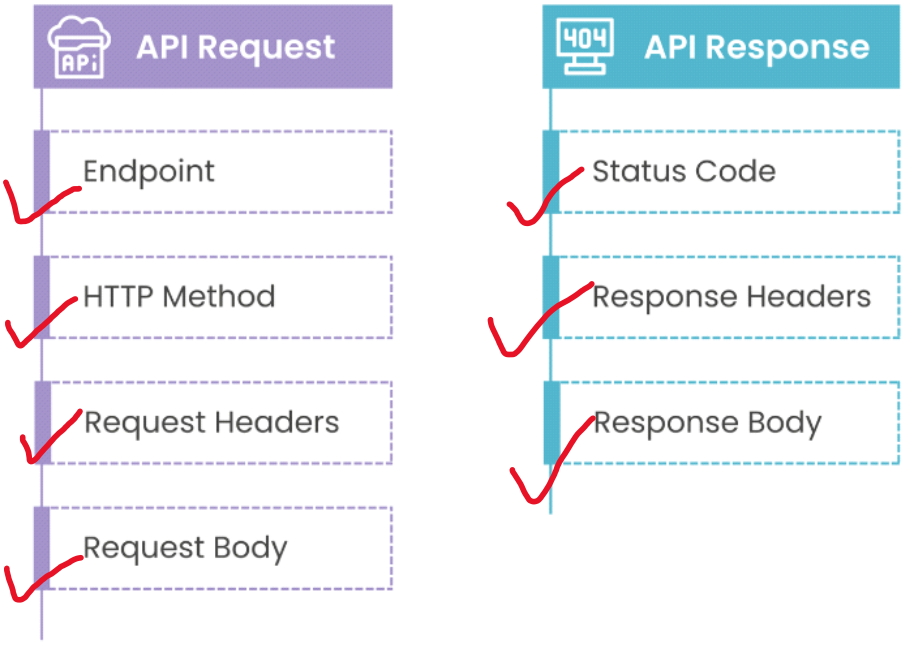
1. **Endpoint:**
   - It's a specific URL where the API can be accessed by a client to perform a certain action
   - Acts as the communication touchpoint between the client and the server
   - Usually include path parameters and query parameters

2. **Request:**
   - Message sent by a client to the API to ask for information or perform an operation
   - Key components of a request:
     - Method: Specifies the type of operation (e.g., GET, POST, PUT, DELETE)
     - Headers: Provide metadata about the request, such as content type
     - Body: Contains the data being sent to the server

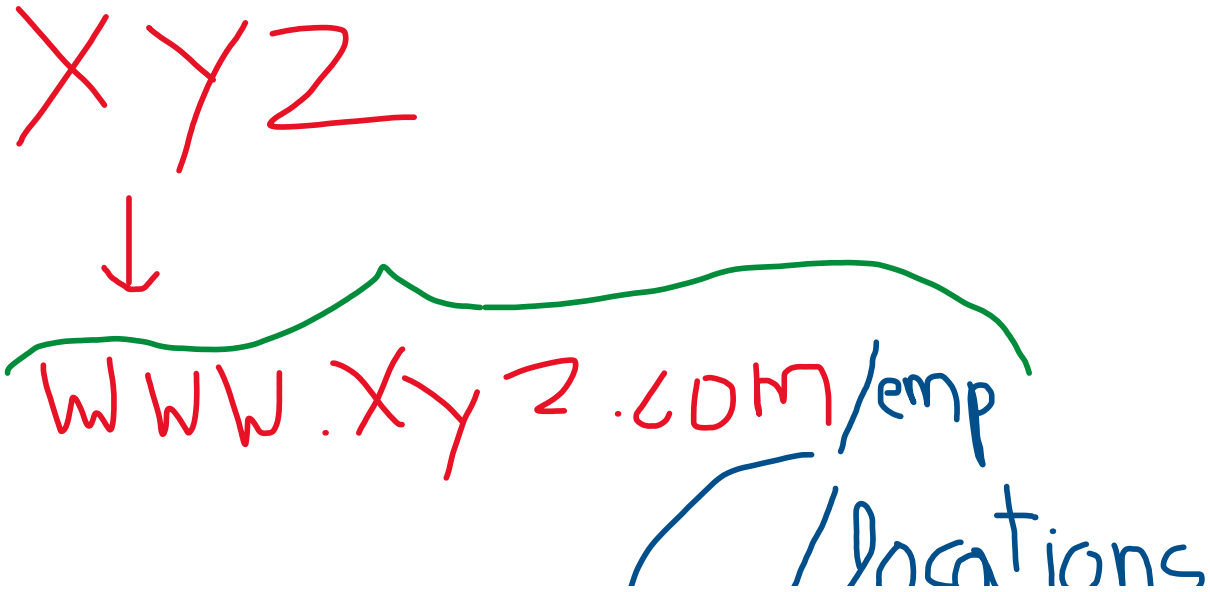
```
POST /users HTTP/1.1
Host: api.example.com
Content-Type: application/json
Authorization: Bearer token123
Body:
{
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```

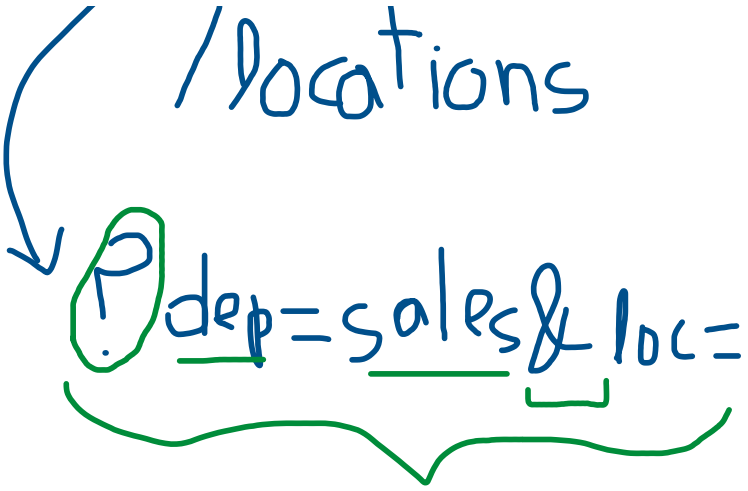
3. **Response:**
   - message sent back by the API after processing a client's request
   - Key components of a response:
     - Status Code: Indicates the outcome of the request (e.g., success, error)
     - Headers: Metadata about the response, such as content type or caching instructions
     - Body: The actual data being returned, typically in JSON, XML

```
HTTP/1.1 200 OK
Content-Type: application/json
Body:
{
  "id": 123,
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```



4. **Rate Limiting and Quotas:**
   - Mechanisms to control the number of requests a client can make within a specified time
   - Mainly to prevent abuse and ensure fair usage
   - Headers for implementation:
     - X-RateLimit-Limit: Maximum requests allowed
     - X-RateLimit-Remaining: Requests remaining in the current window
     - Retry-After: Time to wait before retrying




/locations
?dep=sales&loc=

7. API Lifecycle

Sunday, January 19, 2025 10:24 AM

- The API lifecycle refers to the comprehensive process of designing, developing, deploying, managing, and eventually retiring an API
- It is a structured framework that ensures APIs are effective, scalable, secure, and meet business objectives
- Understanding the API lifecycle is critical for maintaining high-quality API products and enhancing the user experience

1. **Planning and Design:**
   - Identify business goals, user needs, and the problems the API will solve
   - Determine the target audience: developers, partners, or internal teams
   - Define endpoints, request/response formats, and data models
   - Follow design methodologies like REST, GraphQL, or gRPC
   - Ensure adherence to API design best practices, such as intuitive endpoints, consistent naming, and proper versioning
   - **Tools:** Postman, SwaggerHub, and Stoplight for API design and documentation

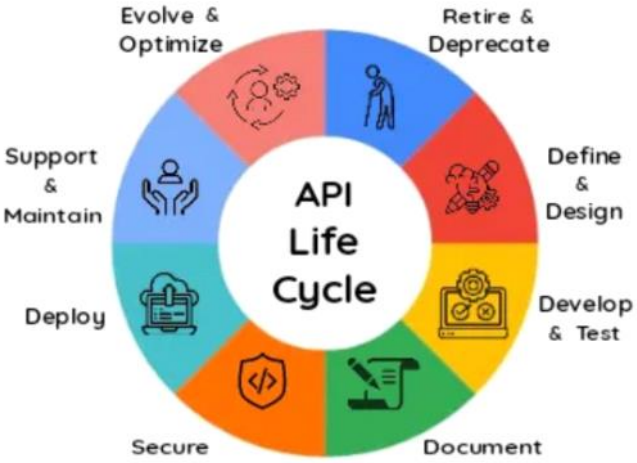
2. **Development:**
   - The development phase involves implementing the API's backend logic and infrastructure
   - Write the server-side code using programming frameworks like Flask, FastAPI, or Express.js
   - Integrate authentication and security mechanisms
   - Perform unit testing to validate individual functions and methods
   - **Tools:** Git, Jenkins, Docker, or Kubernetes for CI/CD pipelines

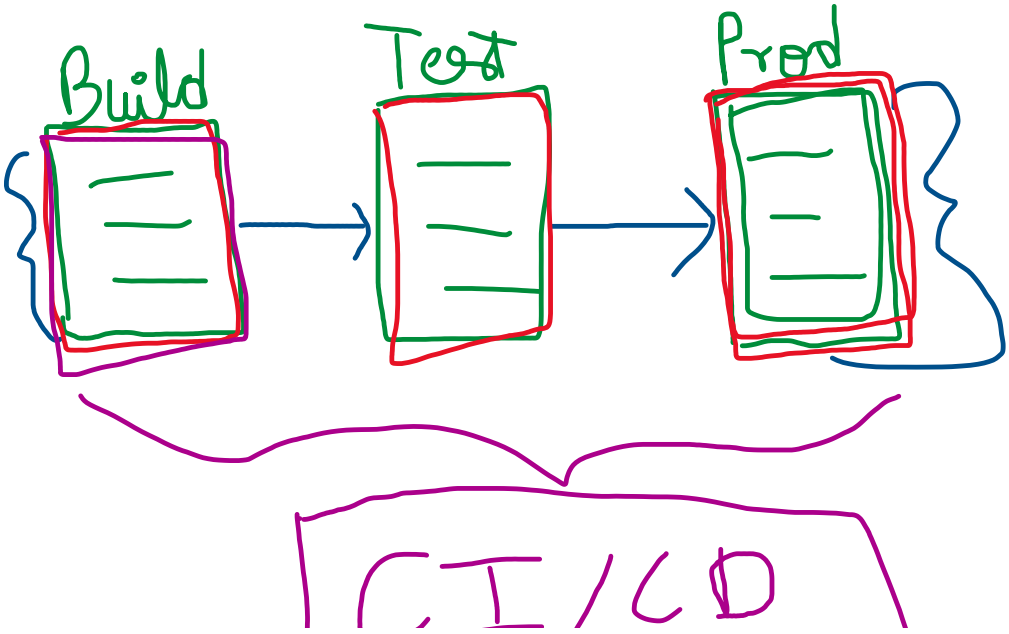
3. **Deployment:**
   - Involves releasing the API into a production environment where users can access it
   - Deploy the API to staging, testing, and production environments
   - Use cloud services like AWS, Azure, or GCP for scalability and reliability
   - Set up an API gateway to manage routing, caching, and load balancing
   - Popular gateways include AWS API Gateway, Kong, and Apigee

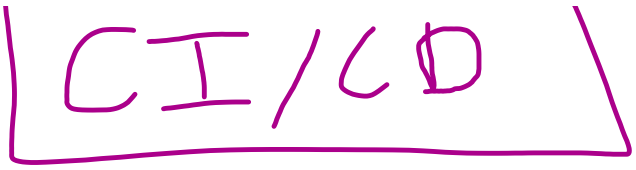
4. **Monitoring and Management:**
   - Track performance metrics like response times, uptime, and error rates
   - Gather usage data to understand traffic patterns, popular endpoints, and user behavior
   - Identify areas for improvement or optimization
   - Enforce rate limiting to prevent abuse
   - **Tools:** Datadog, New Relic, and Grafana

5. **Updates and Versioning:**
   - Ensure new updates do not break existing clients
   - Provide clear migration paths if breaking changes are unavoidable
   - Notify users in advance of deprecated features or versions
   - Provide timelines and guidelines for transition

6. **Retirement:**
   - When an API no longer serves its purpose, it is retired or deprecated
   - Inform stakeholders and users about the API's retirement well in advance
   - Provide tools or resources to help users transition to newer APIs
   - Archive or securely dispose of any stored data associated with the API






CI/CD

8. Authentication & Authorization

Sunday, January 19, 2025 11:01 AM

- When building an API, especially one that will be exposed to the public or clients, managing who can access it and ensuring t he security of data is paramount
- Authentication and authorization are two critical concepts that help in securing APIs and ensuring that only legitimate users can access resources

Authentication:

- Process of verifying the identity of a user or system
- It answers the question: ***Who are you?***

Authorization:

- process of determining what an authenticated user is allowed to do
- It answers the question: ***What can you do?***

Types of Authentication Mechanisms:

1. **API Keys:**
   - They work by sending a key (usually a long string of characters) with each request to identify the client
   - Typically used for public APIs or services where the user is not required to log in manually
   - Easy to implement and widely supported
   - Can be intercepted if not transmitted over HTTPS and does not provide strong user verification
2. **OAuth (Open Authorization):**
   - Open standard for access delegation commonly used for third-party authentication
   - Allows a user to grant a third-party application access to their resources without sharing their credentials
   - Secure and allows fine-grained permissions
   - Comparatively more complex to implement
3. **JWT (JSON Web Tokens):**
   - JWT is a compact and self-contained method for securely transmitting information between parties as a JSON object
   - It's commonly used for handling user authentication in stateless applications
   - No need to store session information server-side
   - Tokens can become stale or vulnerable if not properly managed
4. **Bearer Tokens:**
   - They are a form of access token commonly used in API requests to authorize access to protected resources
   - These tokens are typically provided by an OAuth server or after a successful login
   - Secure and stateless authentication method

Types of Authorization Mechanisms:

1. **Role-Based Access Control (RBAC):**
   - Common authorization method used to assign permissions based on user roles
   - The system defines roles (e.g., Admin, User, Manager) and each role has certain permissions associated with it
   - After a user is authenticated, the API checks the user's role and grants or denies access based on predefined role permissions
   - Simple to manage when there are clear roles

2. **OAuth 2.0 and OpenID Connect (OIDC):**
   - OAuth 2.0 is a widely used framework for authorization
   - OIDC is an identity layer on top of OAuth 2.0, which provides authentication features, in addition to OAuth's authorization features
   - OAuth 2.0 is typically used for delegating access to APIs on behalf of a user

Best Practices for API Authentication and Authorization:

- Ensure all authentication and authorization requests are transmitted over HTTPS to prevent **man-in-the-middle (MITM)** attacks
- Use hashing algorithms (e.g., bcrypt, Argon2) to store user passwords
- Set appropriate expiration times for tokens and use refresh tokens to allow users to renew their sessions without needing to re-authenticate
- Implement **multi-factor authentication (MFA)** where possible to increase the security of APIs
- Return generic error messages for failed authentication or authorization attempts to avoid exposing sensitive information
- Regularly review API logs to detect suspicious activity and to identify potential security breaches