APIs

API

- Stands for Application Programming Interface
- These are messengers (intermediary b/w 2 applications) that allow different software applications(systems or services) to communicate with each other.
- They define the methods/ rules and data formats that applications should use to request and exchange information or functionalities.

Components of an API

1. Endpoints:
   - Endpoints are URLs that the API exposes for interacting with the service. For example, https://api.example.com/users might be an endpoint to access user data.
   - Each resource or a functionality (that you intend to expose) is associated with an endpoint.
2. Requests:
   - The way you communicate with the API. Typically involves sending an HTTP request (GET, POST, PUT, DELETE).
3. Responses:
   - The data or the outcome returned by the API in response to a request. This is usually in the form of JSON or XML.

   ANALOGY:
   - You - one application
   - Restaurant - another application which provides/ exposes services like food services, drinks services
   - Server - API. You interact with the server to obtain certain service (either food service or food service. Each of these have some endpoint say FOOD or DRINK).
     - You can then place a certain type of request(say order, cancel order, update order) to the server on one or more the exposed endpoints.
     - The server then responds to the service: either complete the request or reject the request.

Types of APIs:

1. Web APIs: Allow communication over the web using HTTP/HTTPS. Examples include REST (Representational State Transfer) and SOAP (Simple Object Access Protocol).
   - **NOTE: All of the above content is wrt Web APIs.**
2. Library APIs: Allow interaction with software libraries, providing functions and methods that the library exposes.
3. Operating System APIs: Provide functions for interacting with the underlying operating system. For example, Windows API or POSIX.

Where the different components related to APIs are executed?

1. API Server:

- **Location:** The API server is where the actual code that processes API requests resides. This is often hosted on a web server, cloud service, or on-premises infrastructure.
  - During development APIs can also be run on a local machine or development server, allowing the developers to test and debug their APIs before deploying them to a production environment(I.e, the actual API server)
- **Responsibilities:** It listens for incoming API requests, processes them, and sends back responses. This server can be running on physical hardware or virtual machines in a cloud environment.

2. Client Applications:

- **Location:** Client applications, which make requests to the API, can run on various devices including web browsers, mobile apps, or desktop applications.
- **Responsibilities:** They initiate requests to the API, handle the responses, and integrate the data or services provided by the API into their own functionalities.

3. API Gateway:

- **Location:** An API gateway is often a separate service or component that sits between clients and API servers. It can be hosted in the cloud or on-premises.
- **Responsibilities:** It handles tasks such as routing requests to the appropriate backend services, rate limiting, authentication, and logging. It helps manage and streamline the interaction between clients and the API server.

4. Service Server (Backend Service):
   - **Location:** This server can be separate from the API server and might be hosted on different infrastructure. For example, it could be on another server, in a different part of the cloud, or even on a separate microservice architecture.
   - **Role:** The service server (or backend service) is where the core business logic, data processing, and storage operations occur. It performs the actual work needed to fulfill the API requests, such as querying databases, running algorithms, or interacting with other services.
   - **Functionality:** It might involve interacting with databases, performing computations, or communicating with other services. The API server delegates tasks to this backend service and may handle complex logic and orchestration.

Architecture Examples:

1. Microservices Architecture:
   - **API Gateway:** Acts as the entry point for all client requests and routes them to the appropriate microservice.
   - **Microservices:** Each microservice is responsible for a specific piece of functionality and may interact with its own database or other services.
2. Load Balancing and Scaling:
   - **API Server:** Handles incoming requests and distributes them across multiple backend services or instances.
   - **Backend Services:** Can be scaled independently based on demand. For example, you might have multiple instances of a backend service to handle high load while the API server remains a separate component.
3. Service-Oriented Architecture (SOA):
   - **API Server:** Provides a unified interface to interact with different services.
   - **Backend Services:** Implement specific business functions and may be distributed across different servers or environments.

How They Work Together:

- Client Request: A client sends a request to the API server.

- API Server Processing: The API server processes the request, which may involve querying or invoking backend services.
- Backend Service Execution: The backend service performs the required operations, such as accessing data or running computations.
- Response: The backend service sends the result back to the API server, which then formats and sends the final response to the client.