API Performance Terms

Response time:

- Response time refers to the amount of time it takes for an API to process a request and provide a response to the client. This includes the time taken to authenticate the request, fetch and process the data, and generate the response.
- The response time is measured in milliseconds or seconds. A faster response time results in a better user experience.

Latency:

- API Latency refers to the time it takes for an API to process a request and send a response, including any network or processing delays. It is delay or lag when a request is sent and when a response is received.
- Latency can be affected by various factors, such as the speed of the network connection, the processing time of an API, and the amount of data being transferred.
- Low latency is desirable as it means the API is able to respond quickly to requests.

Availability:

- This metric measures the percentage of time that the API is available for use.
- A high availability rate indicates that the API is reliable and stable.
- The availability metric is calculated by dividing the number of successful responses by the total number of requests made during a specific time period. For **example**, if an API receives 1000 calls in a day and 980 of those requests are successful, then the availability rate would be 98%.
- An availability rate of 99.9% is generally considered an acceptable rate for a critical API.

Error Rate:

- API error rate is a performance metric that measures the percentage of requests made to an API that result in errors.
- A low error rate is an important indicator of the API's reliability and performance.

A Few things to consider when measuring the API error rate:

- Error types: Client errors (400 Bad Request, etc.) and server errors (500 Internal Server Error).
- Error frequency: Monitoring the frequency of errors can help identify trends and patterns that could indicate underlying issues.
- Error handling
- Error Monitoring
- Root cause analysis

Throughput:

- Throughput refers to the number of requests that an API can process in a given time period and is an important metric for measuring API capacity to handle a large number of requests.
- The throughput of an API is affected by various factors, such as the network bandwidth, the size of the data being transferred, the processing power of the server, and the efficiency of the API implementation.

- Throughput is measured in terms of requests or transactions per second. A higher throughput generally indicates a more responsive and scalable API.

Requests Per Second (RPS) OR Transactions Per Second (TPS):

- RPS: The number of HTTP requests that an API can handle in a second. RPS measures how many requests an API can handle per second, including both successful and failed requests.
- TPS: Measures the number of complete transactions that an API can handle in a second, where a transaction typically involves a series of API requests and responses necessary to complete a particular action.
- A high RPS or TPS rate indicates that an API can handle a large number of requests or transactions.

Rate Limiting:

- API rate limiting involves setting a limit on the number of requests a user can make to an API within a certain period of time. The limit is typically defined in terms of a maximum number of requests per second, minute, or hour. If the limit is exceeded, the API will respond with an error message indicating that the user has reached their rate limit.
- The purpose of rate limiting is to prevent a single user from overwhelming the API with requests, thereby affecting the traffic for other users.
- Rate Limiting is implemented at the user or client level.
- The 'Token Bucket Algorithm' is commonly used for rate limiting.

Ways to Optimize API Performance:

Load Balancing: Load Balancing distributes the incoming requests across various servers, which helps to evenly distribute the load and prevent any single server from becoming overloaded. This can be achieved using tools like NGINX or a cloud load balancer like the AWS Elastic Load Balancer.

Rate Limiting / Throttling: The number of transactions per second (TPS) is measured, and API clients making excessive requests are limited or cut off.

Caching: Implement server-side caching to reduce the number of requests hitting the databases.

- API requests that frequently produce the same response; a cached version of the response avoids excessive database queries.
- A cache is a memory buffer where frequently accessed data is temporarily stored to be accessed quicker. The cached data is then retrieved without having to access the origin. Caching will improve your app's response time and even reduce some costs, such as bandwidth and data volumes.
- Frequently accessed data can be cached.
- Caching helps reduce the number of requests to the server and improve the response time. This can be achieved by using in-memory caching or distributed caching solutions like Redis or Memcached.

DB connection:

- Optimized queries: Optimizing database queries can help improve API performance by reducing the time taken to retrieve data from the database.
- Connection pool has optimal configuration.
- Faster response from the DB.
- Add appropriate indexes to the DB tables to increase performance.