



Cloud Performance Optimization

Inefficient Code or Algorithms

- Poorly optimized code or inefficient algorithms can lead to excessive CPU usage and slow processing.
- **Solution:** Profile and optimize code. Refactor inefficient algorithms and use asynchronous programming where possible to improve throughput.

Limited Bandwidth and High Network Latency

- Data-intensive operations can overwhelm network capacity, leading to slow response times.
- **Solution:** Optimize data transfer by compressing data, using efficient protocols, or moving computations closer to data storage (edge computing). Implement CDN for static resources. Load balancing. Implement efficient data retrieval and pagination in your APIs.

Inefficient Communication Between Services

- Your system becomes slow because a lot of time is spent communicating between services.
- **Solution:** keep the number of microservices and communication between them limited

Inefficient Backend-Frontend Communication

- For example, doing autosave on every character changed in a text field, constantly checking if user is still logged in via backend requests.
- **Solution:** use a smarter autosave mechanism that only saves after stopping typing for a few seconds. Don't constantly check if the user is logged in, but get the expected cookie expire date once via a backend call, then show a dialog or redirect to logging page when the cookie expires.

Inefficient Database Queries

- If you have many requests at the same time or ask for huge amounts of data from the database, this can cause slowdowns. Backend might be setup to do lots of queries (i.e. GraphQL n+1 problem).
- **Solution:** Go through queries and make sure they're optimal - use the power of SQL to filter results. Implement caching using for example Redis or Memcached.

Overloaded or Under-Provisioned Servers

- Servers may become a bottleneck if they are not provisioned according to demand.
- **Solution:** Implement auto-scaling to dynamically adjust the number of active server instances based on current load. Regularly review performance metrics to adjust baseline provisions (i.e. horizontal scaling). Horizontal scaling also has other benefits, such as adding redundancy and improving fault tolerance.

Inefficient Cloud Resource Utilization

- Doing heavy compute tasks in a REST backend, blocking the simpler requests to go through.
- **Solution:** move these tasks to a separately scaleable service so they don't interfere with simple REST API service (i.e. vertical scaling). Vertical scaling does have limitations, like potential downtime and reaching upper hardware limits.

Excessive Logging

- Excessive or unoptimized logging can consume a significant amount of resources.
- **Solution:** Implement log sampling or log levels to reduce the volume of logs generated. Ensure efficient log management and storage.

Single Points of Failure

- System components without redundancy can become single points of failure, leading to system outages.
- **Solution:** Design for redundancy and high availability. Use multiple instances, load balancing, and failover strategies to ensure uptime.