Load Balancing & Scaling Notes

I. The Why and How of Scaling

Scaling is the ability of a system to handle increased load (more users, more data, more requests) by adding resources.

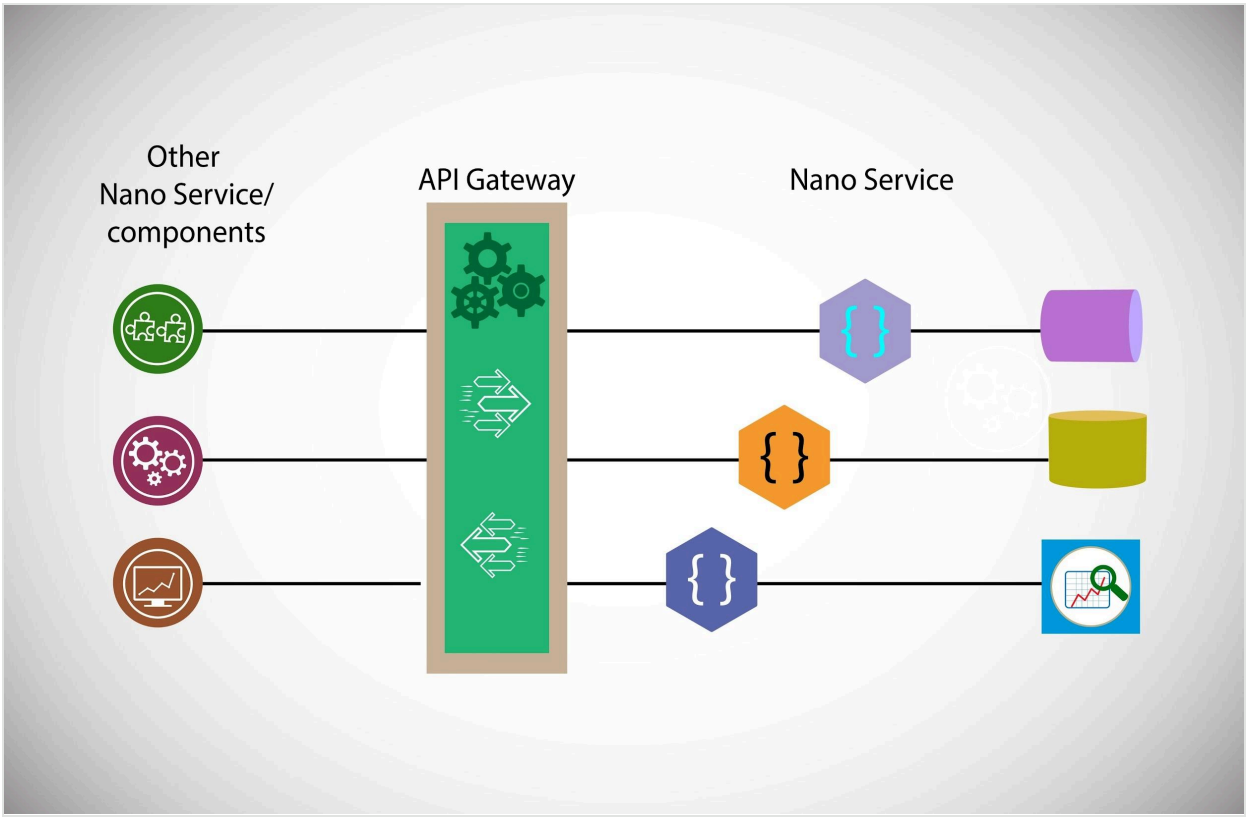
1. Vertical vs. Horizontal Scaling

- **Vertical Scaling (Scaling Up):** Adding more CPU/RAM to a single machine.
    - *Analogy:* Upgrading from a bicycle to a truck.
    - *Pros:* Simple, no changes to application logic.
    - *Cons:* Hard hardware limit; "Single Point of Failure" (if the truck breaks, everything stops).
- **Horizontal Scaling (Scaling Out):** Adding more machines to your pool.
    - *Analogy:* Hiring a fleet of bicycles to deliver packages.
    - *Pros:* Virtually infinite scale; High availability (if one bike breaks, 99 others still work).
    - *Cons:* Requires a Load Balancer and complex networking logic.

> Interview Talk Track:
>
> "While vertical scaling is easier to implement initially, modern distributed systems almost always prefer Horizontal Scaling. It allows us to treat our servers like 'cattle' rather than 'pets'—if one goes down, we just replace it without bringing down the entire service. It’s the only way to handle millions of concurrent users efficiently."

II. The Load Balancer (LB)

The Load Balancer is the "Traffic Cop" of your architecture. It sits between the user and the servers, directing incoming requests to ensure no single server is overwhelmed.



Shutterstock

1. Core Responsibilities

- **Request Distribution:** Spreading the load.
- **Health Checks:** Periodically "pinging" servers to see if they are alive.
- **SSL Termination:** Handling the heavy lifting of decrypting HTTPS requests so the backend servers don't have to.

2. Common Algorithms

- **Round Robin:** Requests are sent to servers in order (1, 2, 3... then back to 1). Best for servers with equal power.
- **Least Connections:** Sends the request to the server currently handling the fewest active users. Best for requests that take varying amounts of time.
- **IP Hash:** Uses the user's IP address to decide which server they go to. Ensures a specific user always hits the same server.

Interview Talk Track:

"I view a Load Balancer as a critical gateway. For a simple web app, Round Robin usually suffices. However, if some requests are heavy (like video processing) and

others are light (like fetching a profile), I would opt for Least Connections to prevent a single server from getting bogged down while others sit idle."

III. Stateless vs. Stateful Architecture

This is a high-level concept that determines how well your system scales.

- **Stateful:** The server remembers the user's session data locally.
    - *Problem:* If the Load Balancer sends the user to a different server next time, the user is "logged out" because the new server doesn't know them.
- **Stateless:** The server stores *nothing* locally. All session data is kept in a shared Database or Cache (like Redis).
    - *Benefit:* Any server can handle any request at any time. This is a requirement for **Auto-Scaling**.

Interview Talk Track:

"To achieve true horizontal scalability, the application tier must be Stateless. If we store session info in the server's local memory, we create 'Sticky Sessions,' which makes scaling difficult. By moving state to a distributed cache like Redis, we can spin up or shut down servers dynamically based on traffic without affecting the user experience."

IV. Eliminating the Single Point of Failure (SPOF)

A SPOF is any part of the system that, if it fails, stops the entire system from working.

- **Server Level:** Use multiple servers + Load Balancer.
- **LB Level:** Use two Load Balancers (one active, one passive) so if the primary fails, the secondary takes over.
- **Data Level:** Use Database Replicas.

Practice Interview Questions (Day 7–8)

1. "How does a Load Balancer know if a server is down?"

- **Answer:** Through **Health Checks**. The LB sends a small request (like a heartbeat) to a specific endpoint (e.g., `/health`) every few seconds. If the server fails to respond with a `200 OK` after a few tries, the LB stops sending traffic to it.

2. "What happens if the Load Balancer itself fails?"

- **Answer:** That becomes a Single Point of Failure. To prevent this, we use **High Availability (HA)** pairs. We have two LBs; they share a virtual IP. If the active one dies, the passive one detects it and takes over the traffic immediately.

3. "Why is 'IP Hash' algorithm useful despite the push for Stateless architecture?"

- **Answer:** Even in a stateless system, IP Hashing can be useful for **caching**. If a user always hits the same server, that server might already have their data in its local 'L1 cache,' making the response even faster. However, we shouldn't *rely* on it for the app to function.

4. "Your auto-scaling group just added 5 new servers. How does the Load Balancer know they exist?"

- **Answer:** The new servers 'register' themselves with the Load Balancer upon startup. The LB then begins health checks and, once they pass, starts including them in the traffic rotation.

5. "We are scaling horizontally, but our Database is now the bottleneck. What should we do?"

- **Answer:** Since DBs are harder to scale horizontally than web servers, we should first implement **Read Replicas** (for read-heavy apps), **Caching** (to reduce DB hits), or eventually **Sharding** (splitting the data).