Lesson 6 -> Horizontal vs vertical scaling

First, what does scaling mean?

Scaling means:

> increasing system capacity to handle more load
> (more users, more data, more requests, more computation)

There are **two fundamental ways** to do this:

- Vertical scaling
- Horizontal scaling

1️⃣ Vertical Scaling (Scale Up)

What it means

You **make one machine more powerful**.

- Add more CPU
- Add more RAM
- Add more disk
- Add better GPU

Same server, bigger machine.

Simple example

You have:

- 4 CPU
- 16 GB RAM

System is slow → you upgrade to:

- 16 CPU
- 64 GB RAM

No architecture change. Just a bigger box.

Real-world examples

- Increasing EC2 instance size
- Adding more RAM to a database server
- Moving from single GPU to bigger GPU

Pros of vertical scaling

1. Very simple
   No code changes, no architectural changes.
2. Low operational complexity
   No load balancing, no distributed coordination.
3. Predictable performance
   No network hops, no distributed latency.

Cons of vertical scaling

1. Hard limits
   You can't scale infinitely. Machines have a max size.
2. Single point of failure
   If that machine goes down → system is down.
3. Expensive
   Cost grows non-linearly with size.
4. Downtime often required
   Upgrading hardware may require restart.

ML-specific view

Vertical scaling works well for:

- Model training on a single GPU
- Early-stage inference
- Prototyping

Fails when:

- QPS grows
- Latency requirements tighten
- High availability is needed

2️⃣ Horizontal Scaling (Scale Out)

What it means

You **add more machines**, not bigger machines.

Instead of 1 powerful server:

- Run multiple smaller servers
- Distribute traffic across them

Simple example

Instead of:

- 1 server with 16 CPU

You run:

- 4 servers with 4 CPU each

Traffic is split among them.

Real-world examples

- Web servers behind a load balancer
- Multiple ML inference pods in Kubernetes
- Distributed training nodes

Pros of horizontal scaling

1. Near-infinite scalability
   Add machines as load grows.
2. High availability
   One machine failure doesn’t bring down system.
3. Cost-efficient
   Cheaper to add commodity machines.
4. Elastic
   Scale up and down automatically.

Cons of horizontal scaling

1. Architectural complexity
   Requires:

- Load balancers
- Stateless services
- Distributed coordination

2. Network overhead
   Inter-service communication adds latency.
3. Data consistency challenges
   State must be shared or externalized.
4. Debugging complexity
   Failures are harder to diagnose.

ML-specific view

Horizontal scaling is ideal for:

- Real-time inference
- High-QPS recommendation systems
- Fault-tolerant ML services

Harder for:

- Large model training (requires coordination)
- Stateful models without careful design

3️⃣ Key Comparison (Intuition)

Vertical scaling:
“Make one brain stronger”

Horizontal scaling:
“Add more brains”

4️⃣ Which one do real systems use?

Almost always **both**, but in different layers.

Example:

- Database → vertical scaling first
- Web layer → horizontal scaling
- ML inference → horizontal scaling
- Training → mix (big GPU + multiple nodes)

5️⃣ Concrete ML Example (Retail)

Scenario:
Recommendation service starts slow.

Vertical approach:

- Move from 1 CPU to 8 CPU
- Works temporarily
- Hits limit

Horizontal approach:

- Run 10 inference replicas
- Load balancer distributes requests
- One replica fails → system still works

Production systems choose **horizontal**.

6️⃣ Common Misconception

“Vertical scaling is bad”

False.

Vertical scaling is:

- Faster to start
- Easier to manage
- Often cheaper initially

But it doesn’t scale indefinitely.

7️⃣ Interview-Ready One-Liners

Vertical scaling improves capacity by upgrading a single machine, while horizontal scaling increases capacity by adding more machines.

Vertical scaling is simpler but limited and risky, while horizontal scaling is complex but scalable and fault-tolerant.

8️⃣ Architect-Level Rule of Thumb

- Start vertical
- Design for horizontal
- Move horizontal when limits are hit

This applies to **ML systems, data platforms, and web systems**.