

Lesson 06 — Software Architecture

Monolith / microservices / serverless / event-driven — what these words actually mean.

IT Landscape

Module 2

What does “architecture” mean?

How you arrange the parts of your software, when it gets bigger than one file.

Same problems buildings have:

- **Load** — how many users at once?
 - **Traffic** — how do requests flow?
 - **Repair** — how do you fix things at 3 AM?
 - **Evolution** — how do you add a feature without breaking what's there?
-

Pattern 1 — Monolith

One big app, one codebase, one deploy.

Pros

- Simple to reason about
- One place to debug
- No network between modules

Cons

- One bug can bring everything down
- Teams step on each other
- Scaling means scaling all of it

How almost every successful product was built first.

Pattern 2 — Microservices

Many small apps, each owning one slice, talking via APIs (or queues).

Pros

- Teams ship independently
- Scale only the hot parts
- Different languages per service

Cons

- Every API call is a network call (can fail!)
 - Operational complexity explodes
 - 10× services to monitor, deploy, secure
-

Pattern 3 — Serverless

You ship a **function**; the cloud provider runs it on demand.

- **AWS Lambda** — the original.
- **Cloudflare Workers** — edge-first.
- **Google Cloud Functions, Azure Functions.**

Pros

- No servers to manage
- Pay-per-invocation
- Scales to zero

Cons

- Cold-start latency
 - Vendor lock-in
 - Surprising bills at scale
-

Pattern 4 — Event-driven

Components don't call each other — they **publish events** to a **broker**; other components **subscribe**.

- Events: `OrderPlaced`, `PaymentReceived`.
- Brokers: **Kafka**, **RabbitMQ**, **AWS SQS**, **Kinesis**.

Pros

- Decoupled producers and consumers
- Great audit trail
- Replayable

Cons

- Hard to follow one request across N services
- Eventual consistency by default

Worked example — an e-commerce site, 4 ways

Same product, four shapes.

Monolith

One Rails / Django app. One database. Catalog, cart, checkout, admin — all in one process. One deploy.

Microservices

`catalog`, `cart`, `orders`, `payments`, `users` — each its own service, its own DB, talking over HTTP/gRPC.

Serverless

An API gateway routes each endpoint (`/cart`, `/checkout`) to a Lambda function. DynamoDB holds state. Scales to zero overnight.

Event-driven

Checkout publishes `OrderPlaced`. Payments, inventory, email, and analytics all subscribe and react.

Trade-offs at a glance

Shape	Team size that fits	Ops complexity	Scale story	Bill at low traffic	Best for
Monolith	1–30	Low	Bigger box	Low (one VM)	Startups, most products
Microservices	50+	High	Scale hot services only	High (cluster + ops)	Many teams, big orgs
Serverless	Any	Medium	Auto, to thousands	Near zero	Spiky, low-volume APIs
Event-driven	Any	Medium–high	Consumers scale per topic	Medium (broker)	Pipelines, audit, integrations

Anti-pattern — the distributed monolith

Microservices that all have to deploy together. Worst of both worlds.

This is what happens when teams split a monolith too early or along the wrong lines — e.g., by **technical layer** ("the auth service", "the email service") instead of by **business domain** ("the orders service").

- You pay the ops cost of microservices.
 - You keep the development friction of a monolith.
 - The fix is usually long and political.
-

Cost models

- **Monolith on one VM** — fixed monthly bill, scales by buying bigger boxes.
- **Microservices on Kubernetes** — fixed cluster bill plus ops staff to run it.
- **Serverless** — pay per invocation; cheapest at low traffic, can become expensive at very high traffic.

Architecture is also a **finance decision**. The cheapest shape at 100 users/day is rarely the cheapest at 100M users/day.

How to read an architecture diagram

- **Boxes** = services / processes.
- **Arrows** = “calls” (sync) or “publishes to” (async).
- **Cylinders** = databases.
- **Trapezoids** = queues / brokers.
- **Clouds** = managed services / SaaS.

Tip: always read along the arrows from “user action” forward. If you can't trace a request from the user box to the database and back, the diagram (or the system) is hiding something.

Recap

- **Architecture is shape**, not technology.
- **Monolith first** is fine and often right.
- The four words are **tools, not destinations**.

Coming up: Lesson 07 — where the data lives.

