From monolith to microservices

When, why, how: draw the boundary today, split the deployment once a measured signal arrives.

Muhammet Şafak • October 3, 2026



SECTION 01

Boundary

Modular monolith: one codebase, one deploy, one database; clear module boundaries inside.

THREE APPROACHES

Where does the boundary sit?

APPROACH	BOUNDARY
Classic monolith	No boundary. Everything reaches everything, and over time it becomes a "big ball of mud".
Microservices	There is a boundary, but the boundary is also a network call.
Modular monolith	There is a boundary, but it lives inside the process; it is a method call.

THE REAL ISSUE

Boundary, service: two decisions.

Draw the boundary today, split the deployment when it is really needed.

THE BOUNDARY IN CODE

One public surface per module

A module calls only the `*Api.php` class of another module. It never runs a `SELECT` directly against another module's table and never ties a foreign key to it.

app/Modules

```
app/
└── Modules/
    ├── Billing/
    │   ├── Domain/          # module's entities, value objects
    │   ├── Application/     # use cases
    │   ├── Infrastructure/ # repositories, external adapters
    │   └── BillingApi.php  # the module's ONE public surface
    ├── Catalog/
    └── Notification/
```

WHY I START HERE

Five concrete reasons

Cheap to fix

A wrongly drawn boundary is fixed with one refactor.

One transaction

Two modules update consistently inside a single DB transaction.

Less operations

One deploy, one log stream, one thing to monitor.

Refactoring tools

The compiler verifies a method call.

Cheap migration

Extracting a module with a clear boundary into a service is mechanical work.

DISCIPLINE

Four practices that keep the boundary standing

- The dependency rule runs in CI; a violation breaks the build (deptrac)
- Each module's public API is a single class
- Relations between modules are an ID + a public call, not a foreign key
- Review question: does this change cross a module boundary without permission?



ALL THE VALUE IS IN THE DISCIPLINE

No CI check, no boundary.

If the boundary isn't enforced by CI, what you have isn't a modular monolith, just a monolith with tidy folder names.

SECTION 02

Cost

Microservices are not a solution, they are a trade-off.

02

ONE CHANGE

A method call turns into a network call.

What you get is independent deployment and independent scaling. What you pay in return is long.

TRADE-OFF

The microservices bill

	In microservices	In a monolith
CALL	Serialization, latency, partial failure	✓ In-process, free
CONSISTENCY	Saga, outbox, eventual consistency	✓ A single DB transaction
OBSERVABILITY	Distributed tracing required	✓ One log stream
DEPLOY	Versioned, backward-compatible contracts	✓ One deploy
REFACTOR	A contract nobody verifies	✓ The compiler verifies

WRONG REASONS

The wrong medicine for the right disease

REASON	THE REAL FIX
"The codebase is too big."	Module boundaries, file layout, dead code cleanup
"Deploys are slow and scary."	Fast pipeline, tests, gradual rollout, rollback
"I need to scale independently."	Monoliths scale out too: N copies + load balancer
"This is modern architecture."	Modern is not an architectural reason
"Netflix does it this way."	Not the problem, only the operational load gets copied

CODE SIZE

Same code, plus a network.

A 200-thousand-line monolith and 200 thousand lines split into 20 services are the same amount of code; the second one just has a network in between.

SECTION 03

Signal

At least one of these signals must have been measured.

03

MEASURED SIGNALS

Four signals that justify the move

Scaling profile

One part consumes resources very differently from the rest.

Ownership

Teams wait on each other's deploys; the boundary is organizational.

Runtime

A measured bottleneck: the workload is clearly cheaper in Go or Rust.

Fault isolation

One module's crash must not take the system down; it can't be achieved in-process.

THRESHOLD

Not on a graph? Not a signal.

If a signal doesn't show up on a graph, it is not yet a signal.

SECTION 04

Migration

Not everything: split out the single module that gives the signal.

04

STRANGLER PATTERN

Taking one module out

1

Pick the module

the one module giving the signal

2

Build the service

the new service comes up

3

Shift the traffic

calls are routed gradually

4

Delete the old

the old code is deleted

When to split, and how not to

BIG-BANG Splitting everything at once is the most expensive migration and the one that fails most often.

TEAM When several teams are squeezed into the same deploy unit.

RESOURCE When one part's resource profile holds the rest of the system hostage.

CRASH When one component's crash regularly takes down the whole system.

BORING ARCHITECTURE

The complexity budget is limited

PHP + Laravel

or Symfony; the application layer.

PostgreSQL

Relational database.

Redis

In-memory data store.

Nginx

Web server.

Linux

Process management with systemd + Supervisor.

New tool

My threshold for boring: 12–18 months of active use in production.

Thank you

muhammetsafak.com/engineering/

Draw the boundaries early, deploy late. Wait for the signal; let measurement make the decision, not fashion.

