Microservices

Overview

- What are microservices?
- Monolith vs microservices
- Core building blocks
- API Gateway
- REST & Messaging
- Best practices & patterns
- When to Use & When NOT to Use
- Node.js project
- Q & A

What are Microservices ?

- Small, focused service → does one thing
- Owns its own code, DB, deployment
- Talks over network → REST or Messaging
- Independently deployable & scalable

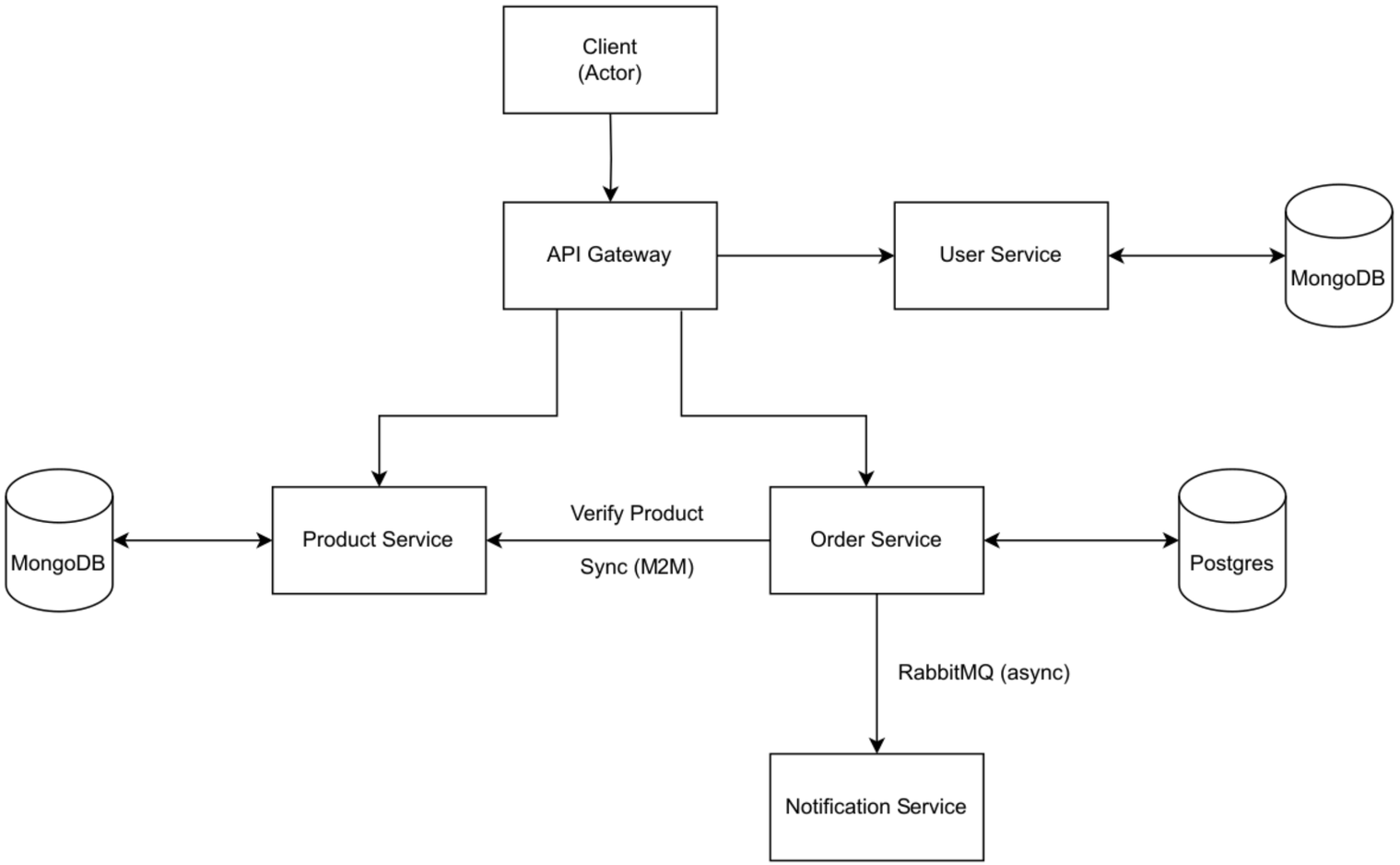
Why Microservices ?

- Scale only what grows
- Small code → easy to maintain
- Different teams → no conflicts
- Faster, safer deployments

Monolith vs Microservices

	Monolith	**Microservices**
App	All in one	Split into services
DB	Shared tables	DB per service
Scaling	Scale whole app	Scale one part
Tech freedom	One stack	Any stack per service

Our Microservice Architecture



API Gateway

- Single entry point
- Routes requests → correct service
- One door → all services behind it

Sync: REST + HTTPS

- Direct call → immediate response
- Easy for lookups
- Node.js → axios library

Async: Messaging with RabbitMQ

- Services publish events → others listen
- Loosely coupled

Each Service = Own DB

- No shared DB
- User Service → MongoDB
- Product Service → MongoDB
- Order Service → PostgreSQL
- Communicate only via APIs OR MQ

Patterns & Pitfalls

- API Gateway → single entry
- Sync + async → balance wisely
- Pub/Sub → decoupling

- Distributed monolith → too many sync calls
- No logs/trace → blind debugging
- Sharing DB → tight coupling

When to Use & When NOT to Use Microservices

✅ Use Microservices When:

- App is big & complex
- You need to scale parts separately
- You have multiple teams
- You want tech freedom (different stacks)
- You plan frequent deploys

🚫 Don’t Use When:

- App is small/simple → single team
- No clear domains yet
- You lack DevOps experience
- You need Strong ACID Transaction
- No clear need for independent scaling

THANK YOU!

Resource Page