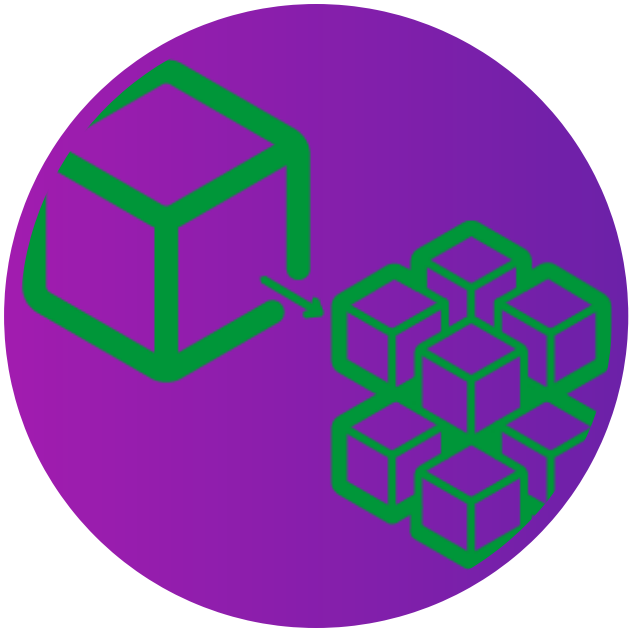


Microservices

Experience the Basics in 30 minutes

CPU * UNMER

Batch 3: Session 5

Richard Michael Coo

@myknbani

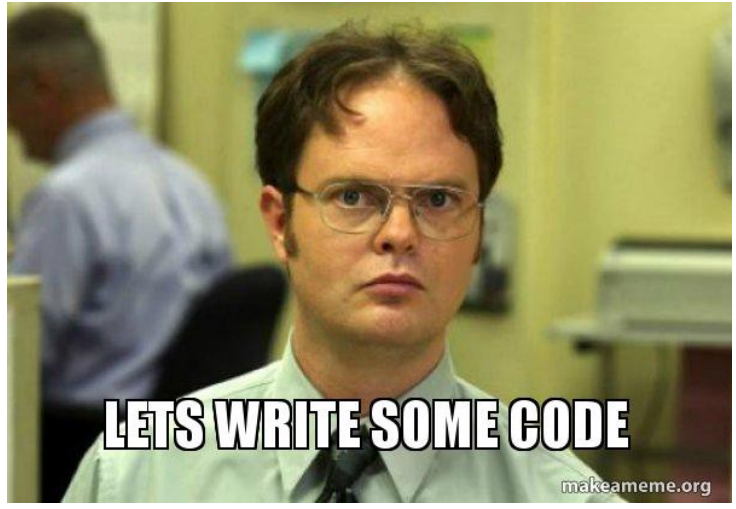
ABOUT ME

Richard Michael Coo

- ⏳ Full-time work:
  - Senior Node.js Developer at Dev Partners
- 🕛 Part-time engagements:
  - Consultant at Stacktrek
  - Instructor at Central Philippine University
    - *Software Engineering department*
- 👨‍💻 Specializes in:
  - Express.js
  - Nest.js
  - Typescript

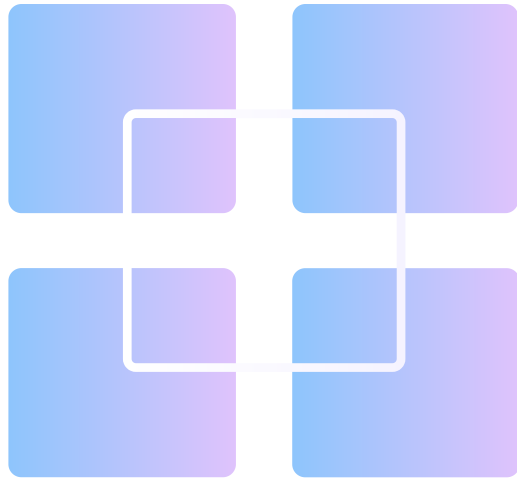
What to expect

- 🗣️ ~~boring talk about microservices, its patterns, antipatterns that you'll forget tomorrow~~
  - There are many good resources online for further study, e.g. https://microservices.io/
- Hands-on – "*get your* 🐾 *feet wet* 💦" – experience in coding your first microservices
  - we'll **not** do things the hard way though, due to time constriants

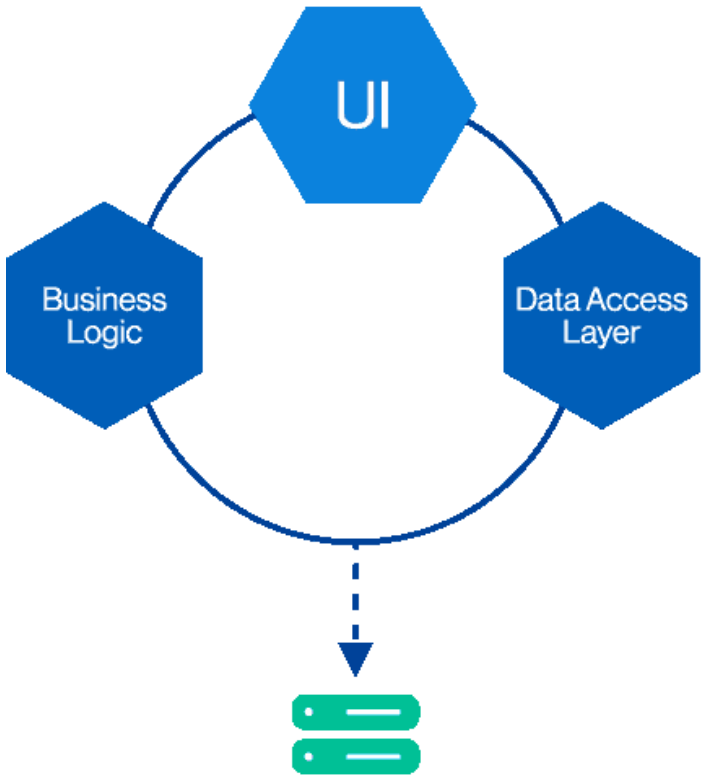


Agenda

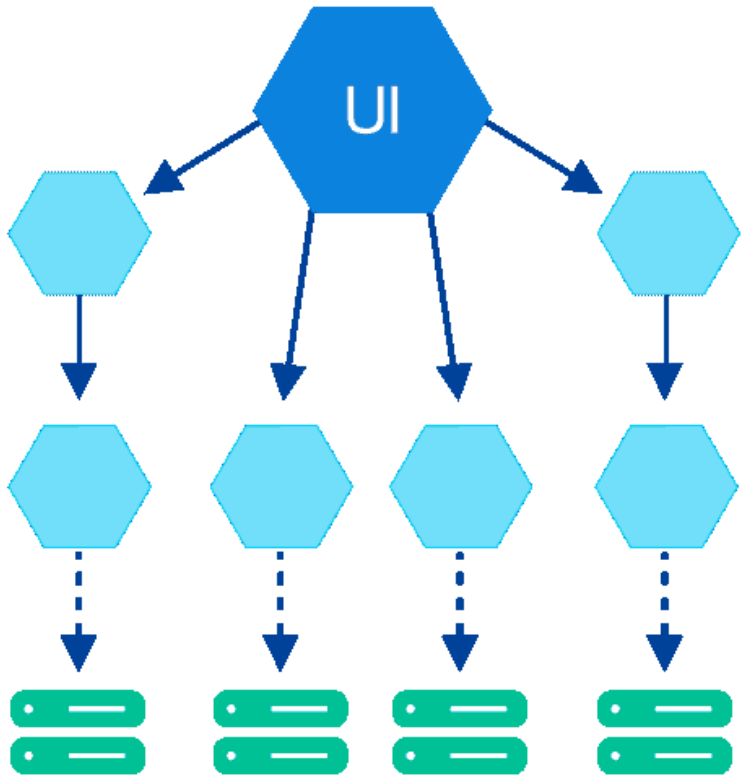
- What are microservices?
- Why should undergrads learn microservices?
- What to expect
- Microservice architecture patterns to use in this session
  - Decompose by business capability
  - Database per (micro)service
  - API Composition
- Hands-on workshop
- Microservice architecture antipatterns
  - My pick: "*trying to fly before you can walk*"
- Parting advice
- Q & A

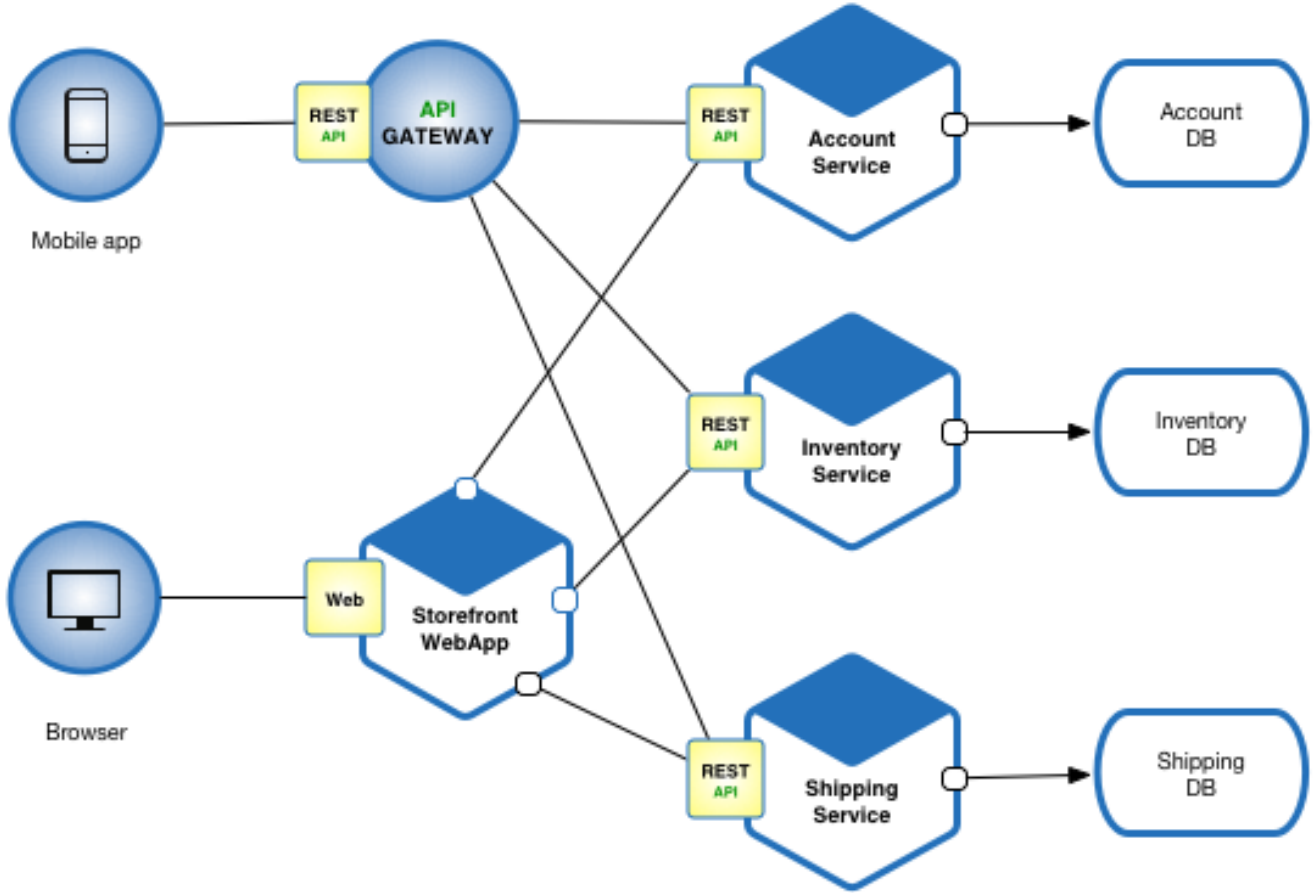


Monolithic Architecture



Microservices Architecture




Mobile app
REST API
API GATEWAY
REST API
Account Service
Account DB
REST API
Inventory Service
Inventory DB
Browser
Web
Storefront WebApp
REST API
Shipping Service
Shipping DB

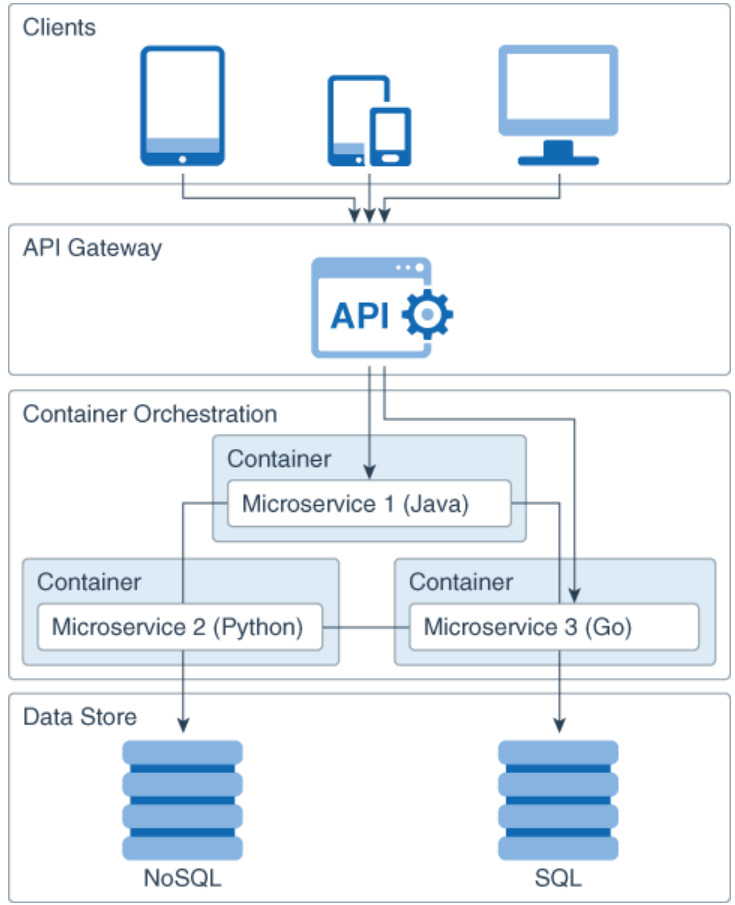
What are Microservices?

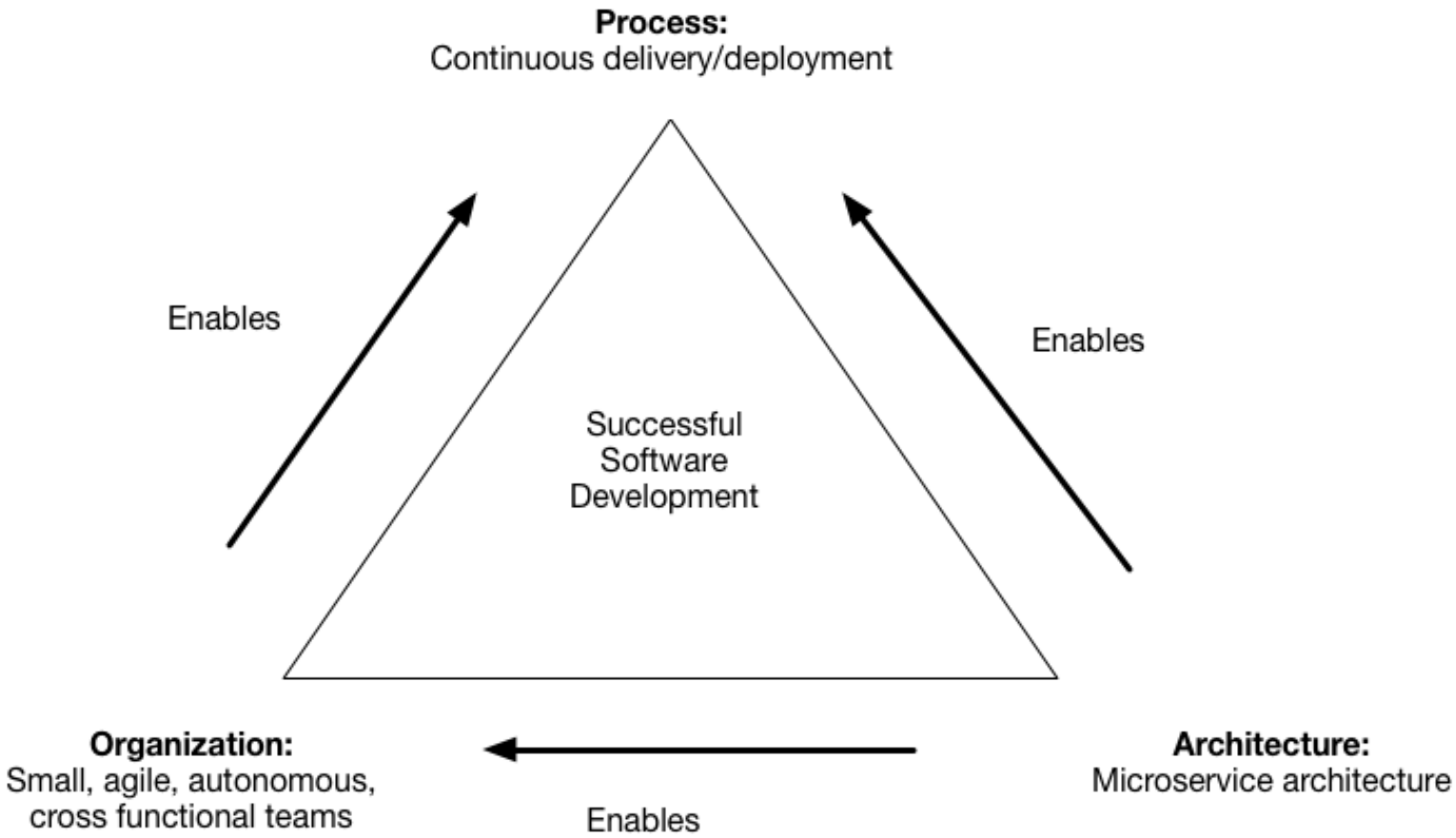
Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are:

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

BENEFITS:

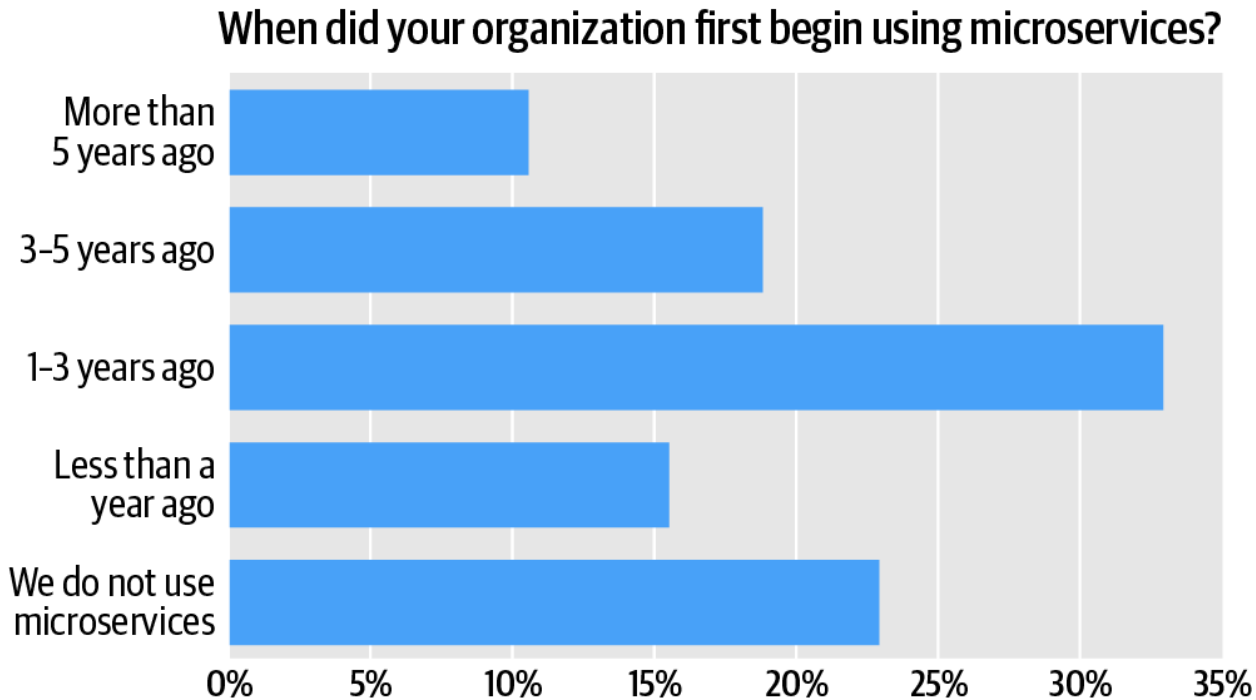
- enables the rapid, frequent, and reliable delivery of large, complex applications
- enables an organization to evolve its technology stack.




Process:
Continuous delivery/deployment
Enables
Enables
Successful
Software
Development
Organization:
Small, agile, autonomous,
cross functional teams
Enables
Architecture:
Microservice architecture

O’Reilly Microservices Survey 2020

1500 respondents



Should undergrads learn microservices?

30+ days ago · More...

Associate Backend Software Engineer

MediLink Network Inc. **3.1** ★

Remote in Makati

Fresh graduate +1 | **8 hour shift**

Easily apply | Responsive employer

Hiring multiple candidates

- Develop state of the art APIs, **microservices** and backend processes.
- The *Backend Software Engineer(CL1)* will help the company by helping the business and its...

Active 4 days ago · More...

Senior Frontend Software Engineer

MediLink Network Inc. **3.1** ★

Remote in Makati

Permanent +2 | **8 hour shift**

Java Enterprise Edition

Accenture ★★★★☆ 22,394 reviews

Manila

You must create an Indeed account before continuing to the company website to apply

Apply on company site

. Bachelor's degree or higher in Computer Engineering, Computer Sciences, IT, Information System, or related field.

. Fresh graduates are welcome to apply

. At least 2 years of experience in Java programming for Junior level

. Minimum 4 years of experience for Senior level Java-based language experience

. Familiar with Java Enterprise Edition (J2EE), Spring Boot, Microservices, OOP, Oracle database

. Ability to work in a team environment delivering quality software that meets requirements working to a timeline

Patterns

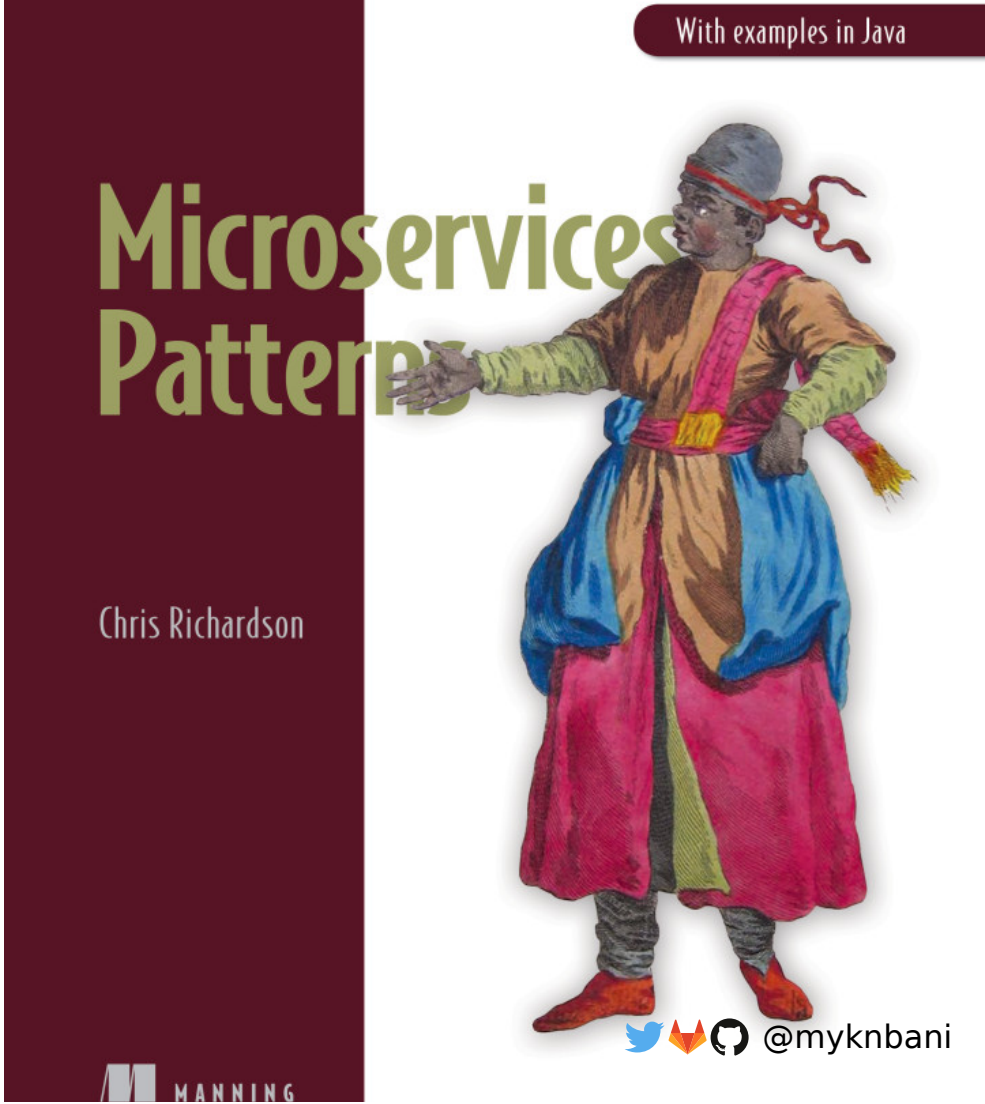
WHAT ARE PATTERNS?

In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design.

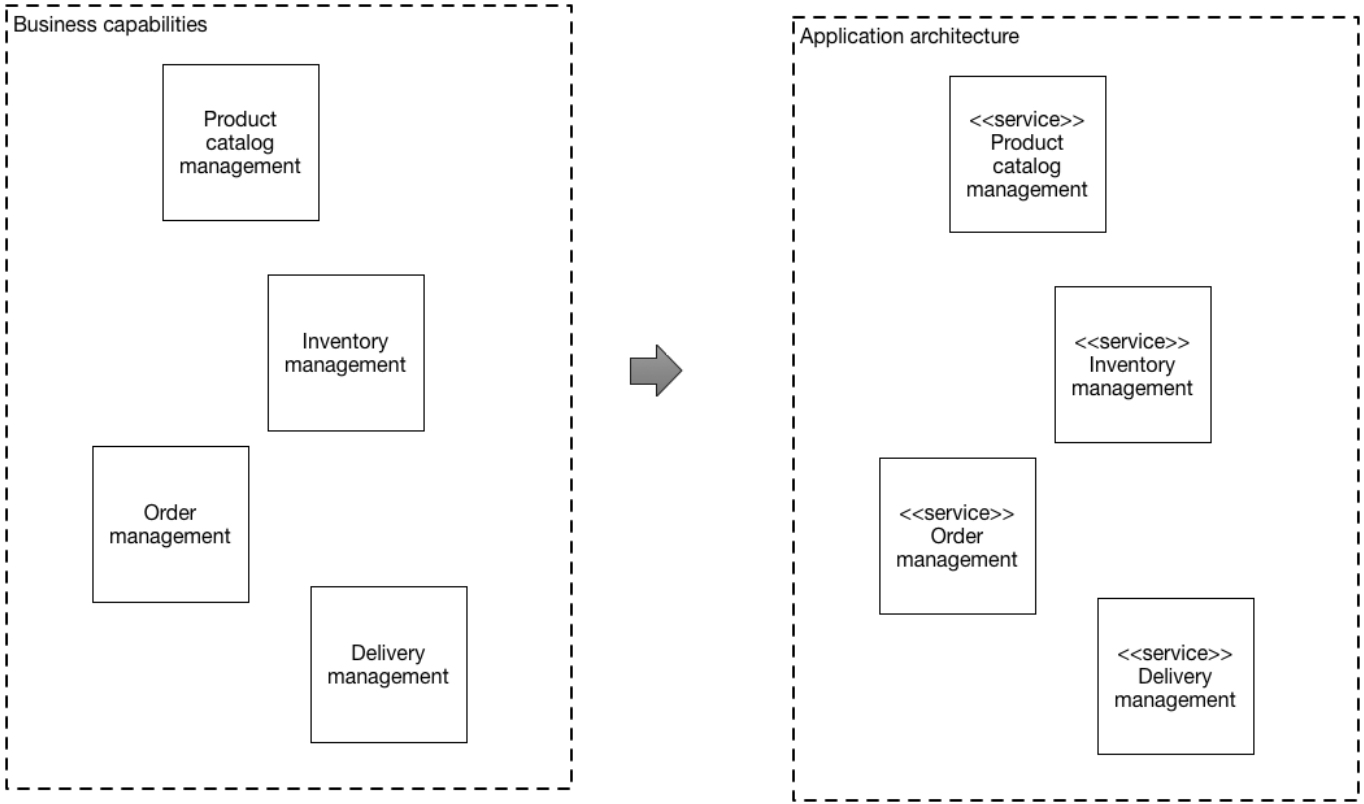
- General - *reusable*
  - experience reuse, **not** code reuse
- Repeatable solution - *proven, best-practice solution*

As a corollary:

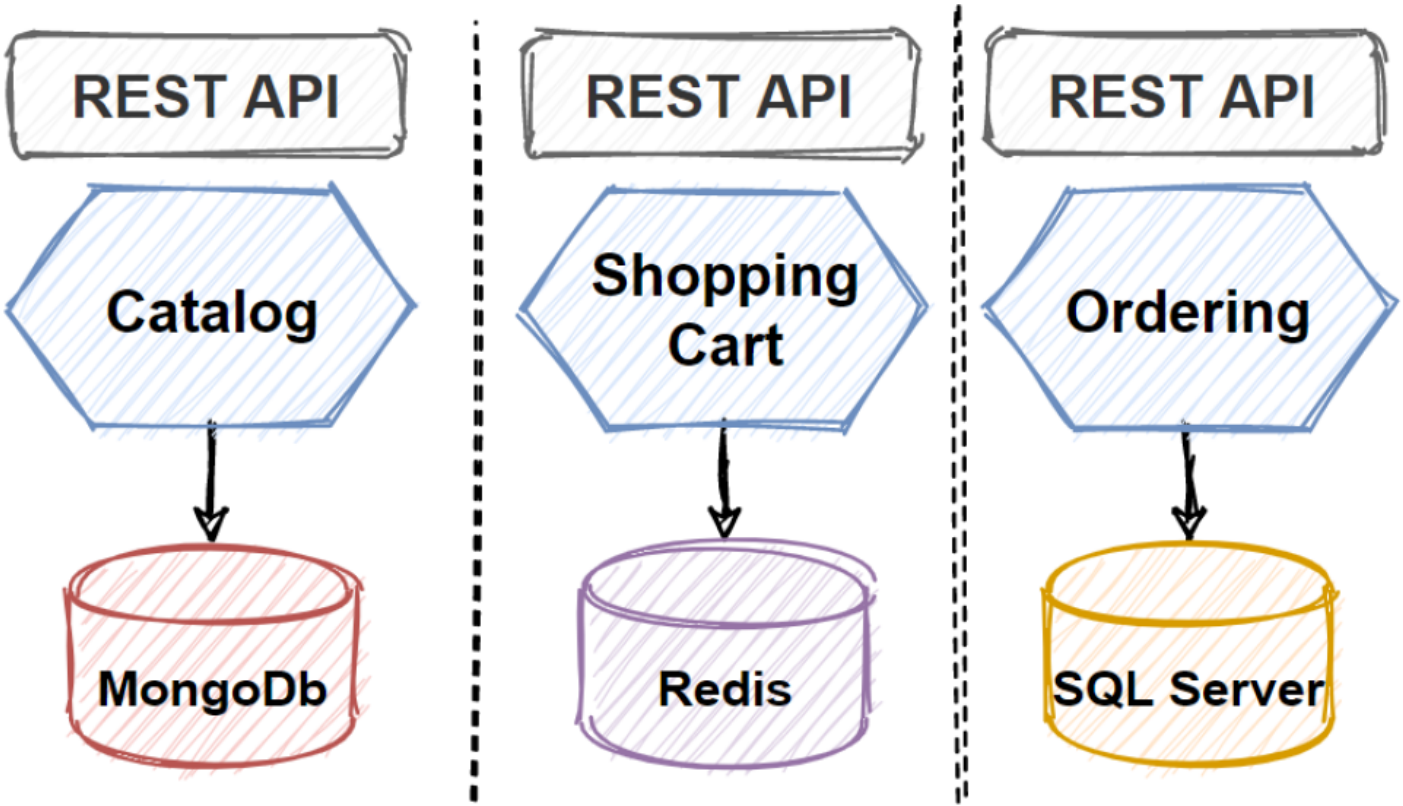
- *Architectural patterns*
- Microservice Patterns



Decompose by business capability

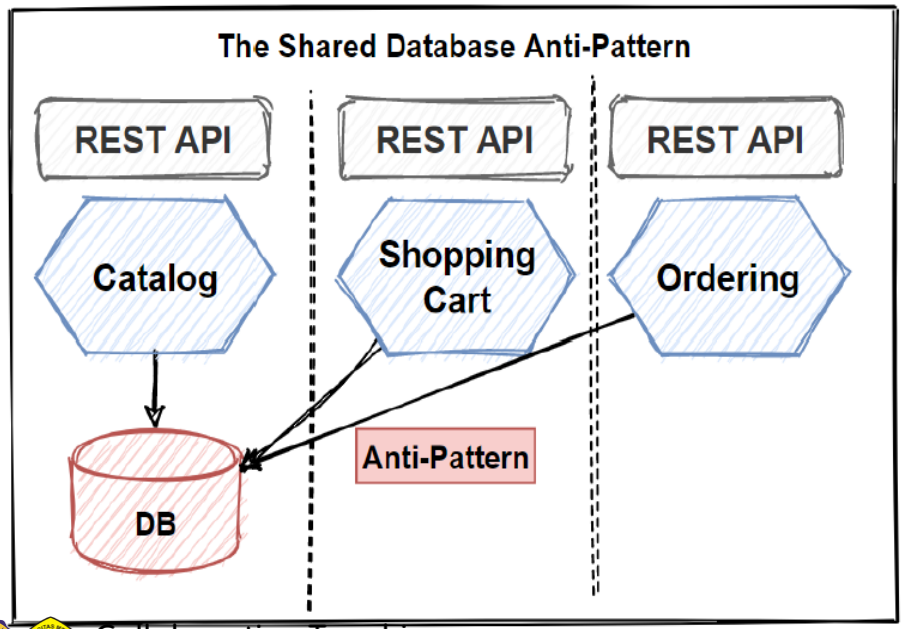


Database per service



Shared database pattern

- some consider this an *antipattern*
  - but it's more of a valid pattern with drawbacks



DRAWBACKS

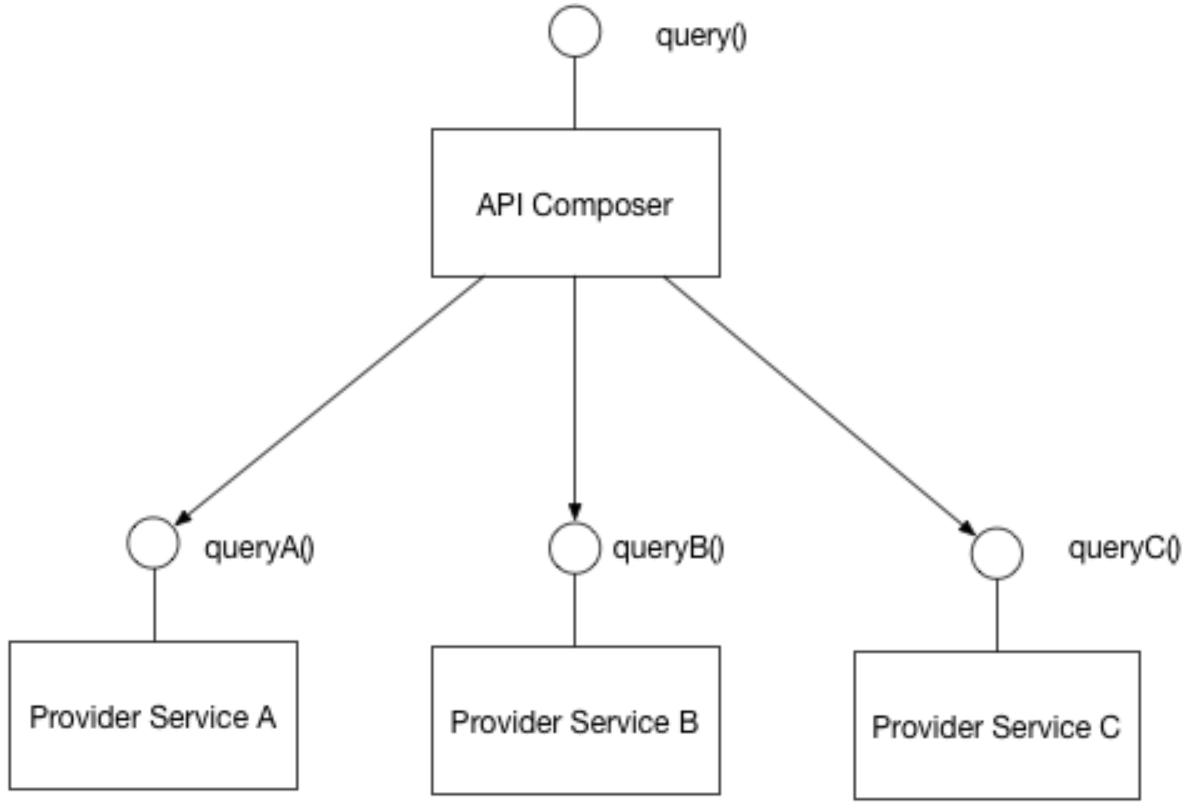
- Development time coupling
- Runtime coupling
- Single database may not be "*one-size-fits-all*"
- different libraries deal with databases differently
  - Migrations and migration tables
    - Knex.js?, Prisma, DBMate?

```
reddit_clone=# \dt
                List of relations
 Schema |          Name          | Type  |  Owner
--------+------------------------+-------+----------
 public | comments               | table | JonRamer
 public | knex_migrations        | table | JonRamer
 public | knex_migrations_lock   | table | JonRamer
 public | posts                  | table | JonRamer
 public | users                  | table | JonRamer
(5 rows)

reddit_clone=#
```

  - Metadata from ORMs, ODMs
    - Mongoose's `__v`
    - Hibernate/JPA locks

API composition

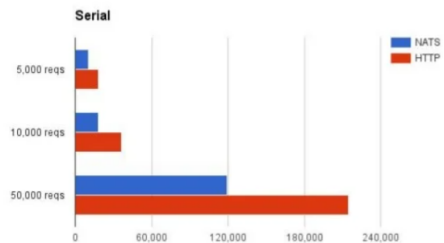


Hands-on Workshop

- Build a Microservice with Nest.js
  - using the NATS Transport
    - supports async *(mailbox pattern)*
    - supports at-least-once delivery
    - faster than HTTP + JSON
- Build an API Gateway with Nest.js
- Build a Microservice with and Panini

Benchmark Results

- Each request is executed serially
- Client times how long it takes to execute all requests



NATS.io
@nats_io · **Follow**

Much like #golang NATS is all about simplicity and performance. It doesn't get much simpler:

- Set up your Go environment

Instructions

1. Install the NATS server.

```
go get github.com/nats-io/gnatsd
```

2. Start the NATS server.

You can invoke the NATS server binary, with no options and no configuration file, to start a server with acceptable standalone defaults (no authentication, no clustering).

```
gnatsd
```

When the server starts successfully, you will see that the NATS server listens for client connections on TCP Port 4222:

```
[1] 2015/08/12 15:18:22.301550 [INF] Starting gnatsd version 0.6.4
[1] 2015/08/12 15:18:22.301762 [INF] Listening for client connections on 0.0.0.0:4222
[1] 2015/08/12 15:18:22.301769 [INF] gnatsd is ready
```

3. Start the NATS server with monitoring enabled.

The NATS server exposes a monitoring interface on port 8222.

```
gnatsd -m 8222
```

Antipattern – My pick:

Trying to fly before you can walk

- Trying to adopt microservices without practicing the basics of software development is likely to lead to disappointment.
- The microservice architecture requires good design skills and test automation. A badly designed microservice architecture that lacks automated tests is likely to be worse than a monolith.
- Messy code will reduce your ability to deliver software rapidly and frequently.

Better Approach

- Clean code
- Automated testing *(TDD is a bonus)*
- Design skills *(OOP, DDD)*

Parting advice

- Build ⚒️ and ship 🚀 something for practice!
- Try to learn it the hard way
  - *(instead of libraries and frameworks)*

FEEL FREE TO REACH OUT!

@myknbani

richard.michael.coo

Add me to your GC or Discord if you like