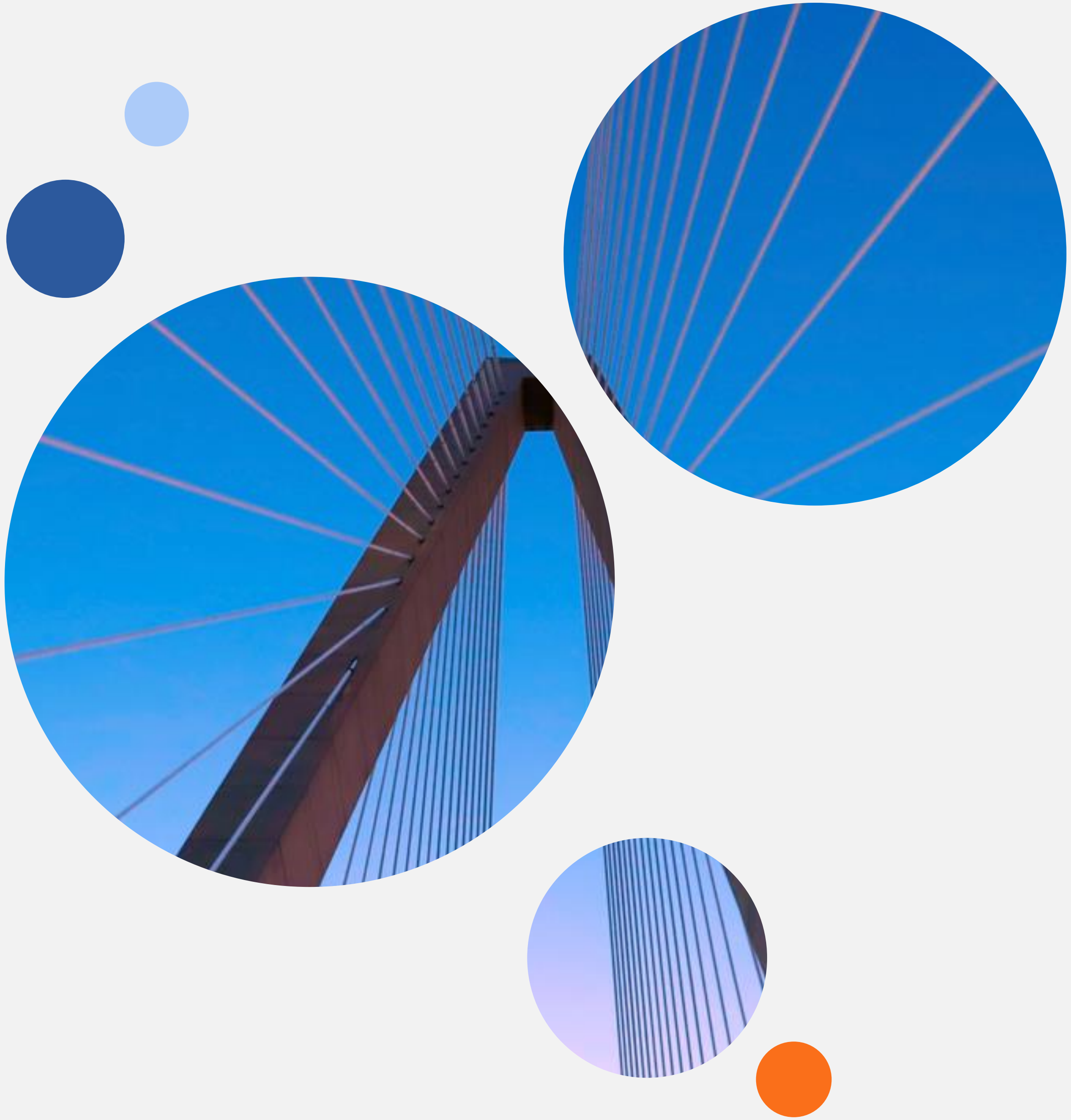
Microservices Architecture

By
Ahmed Abouzeid

Agenda

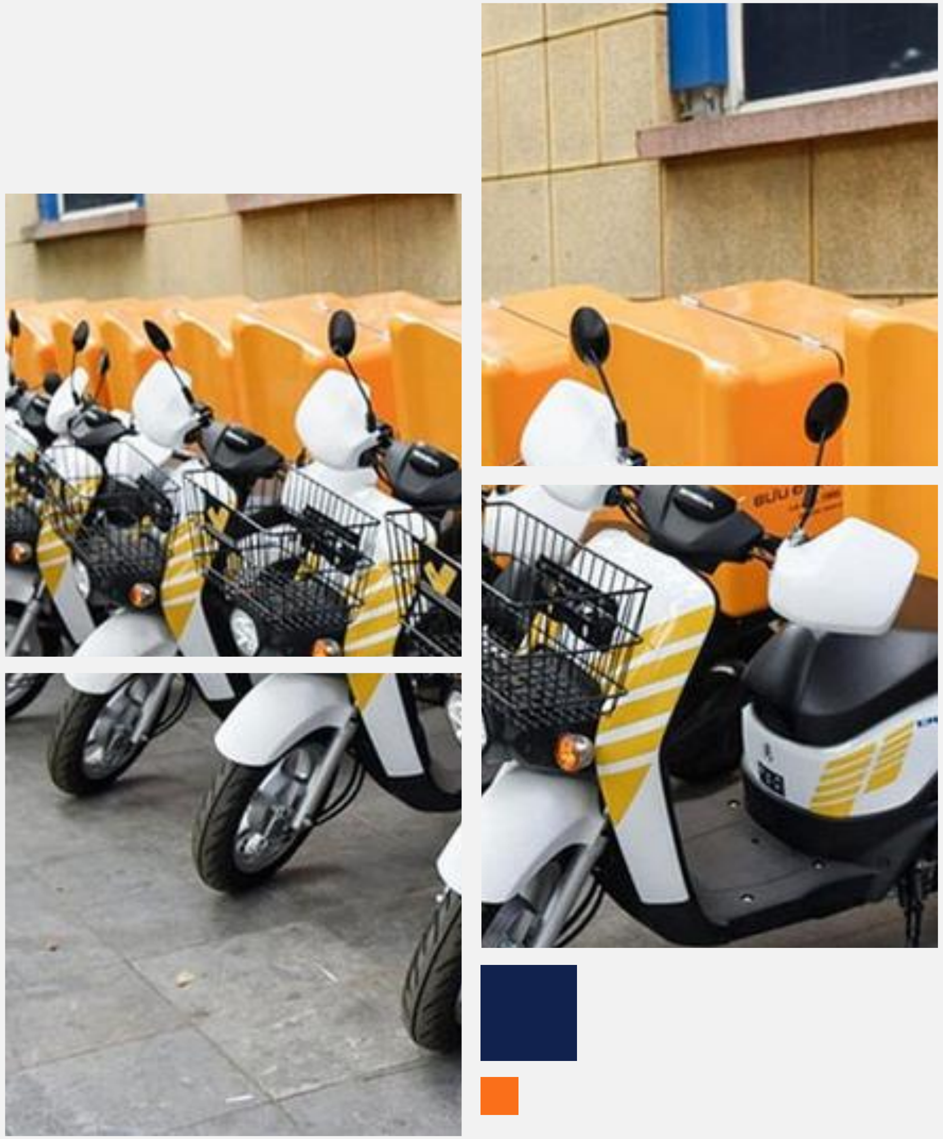
- Business Case
- Tackling the business case
- What is Microservices architecture?
- Benefits and Challenges
- Domain Modeling for Microservices
- Compute options
- Interservice communication
- API Gateway
- Data Management



Business Case

Delivery Service

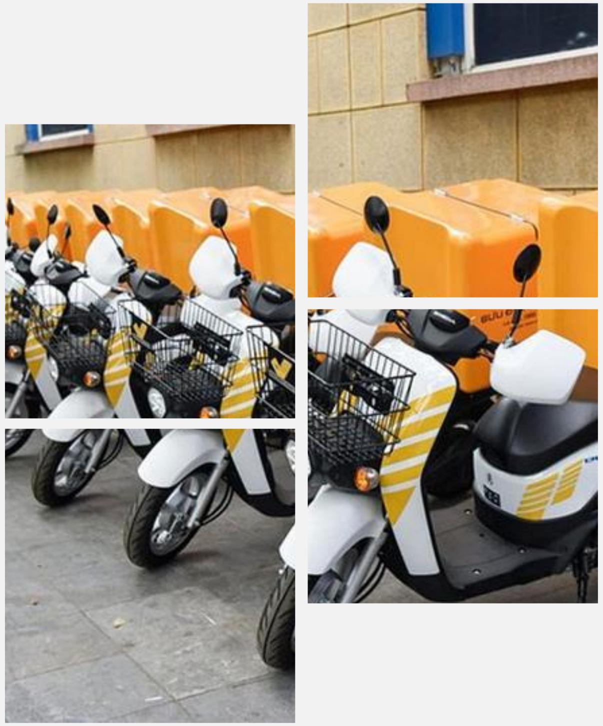
- The company manages a fleet of vehicles
- Businesses register with the service, and users can request a vehicle to pick up goods for delivery
- When a customer schedules a pickup, a backend system assigns a vehicle and notifies the user with an estimated delivery time
- The customer can track the location of the vehicle, with a continuously updated ETA

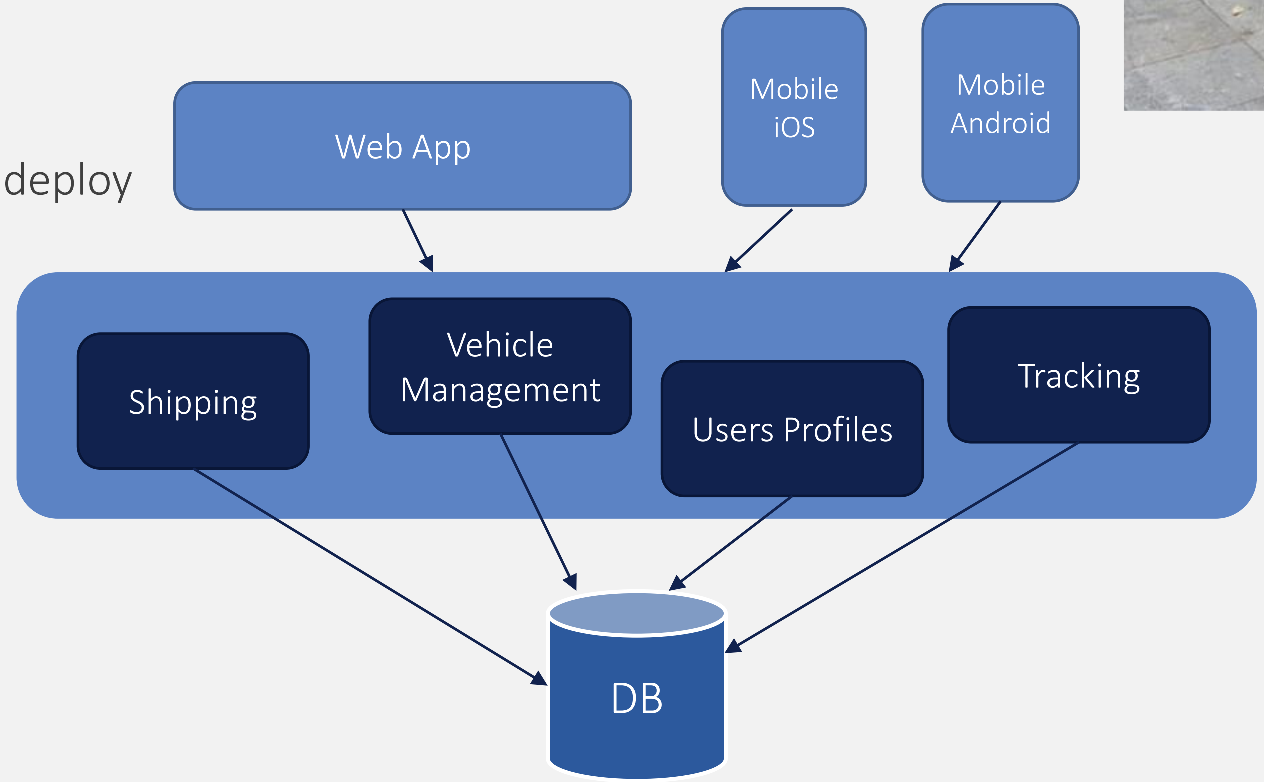


Tackling the business case

Monolithic Architecture

- The traditional unified model for the design of a software program.
- Multiple components are combined into one large application.
- Motivations:
  - Well Known
  - Easy to understand
  - Easy to develop, test and deploy



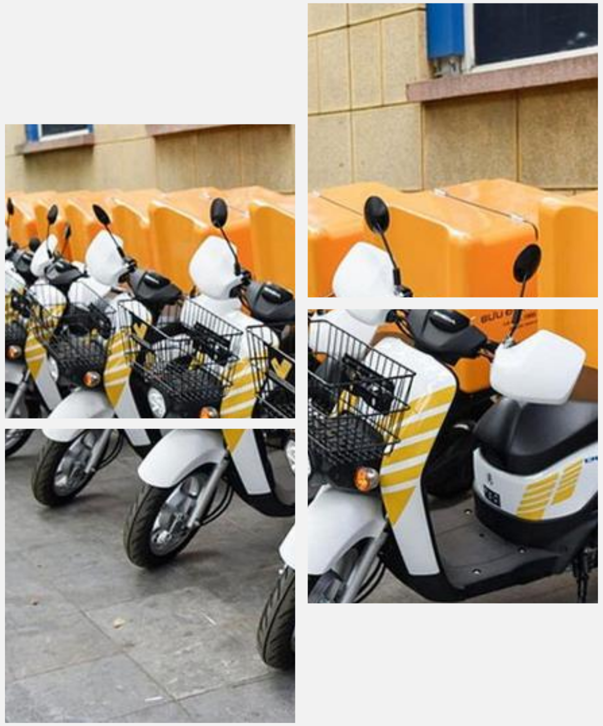


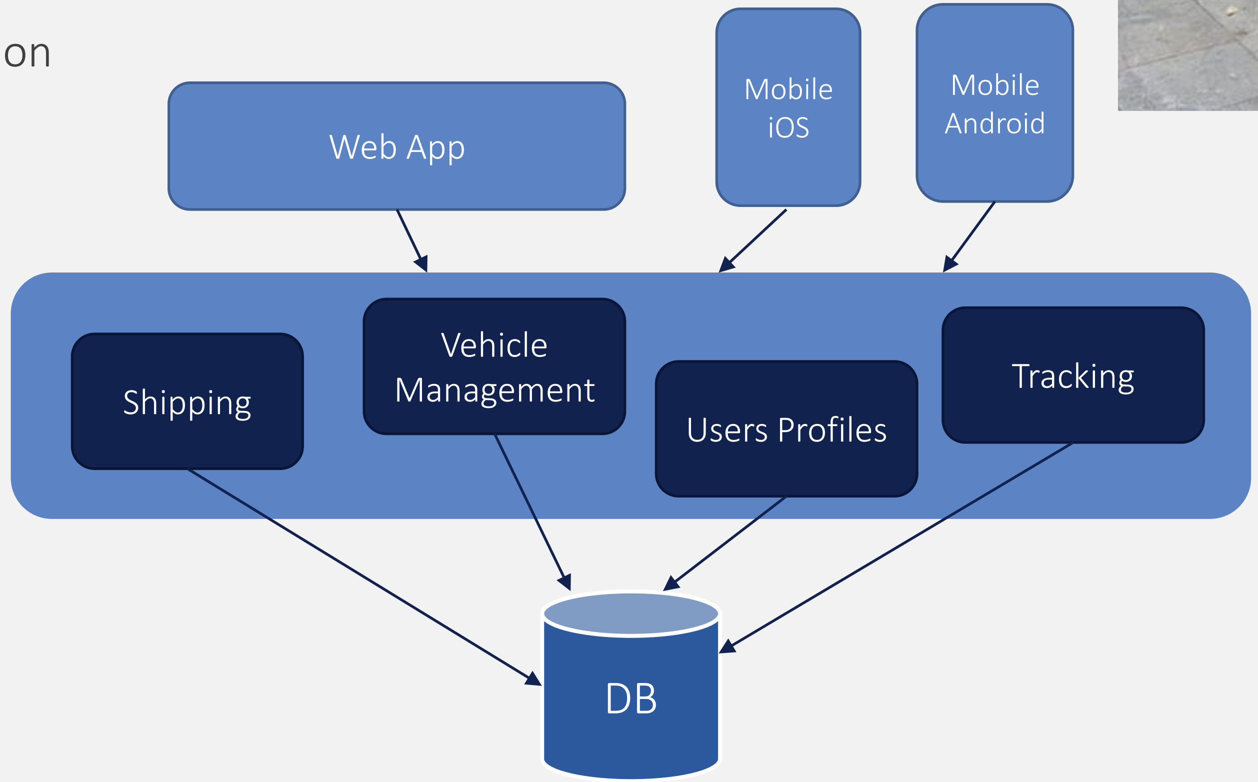
Tackling the business case

Monolithic Architecture

Downsides

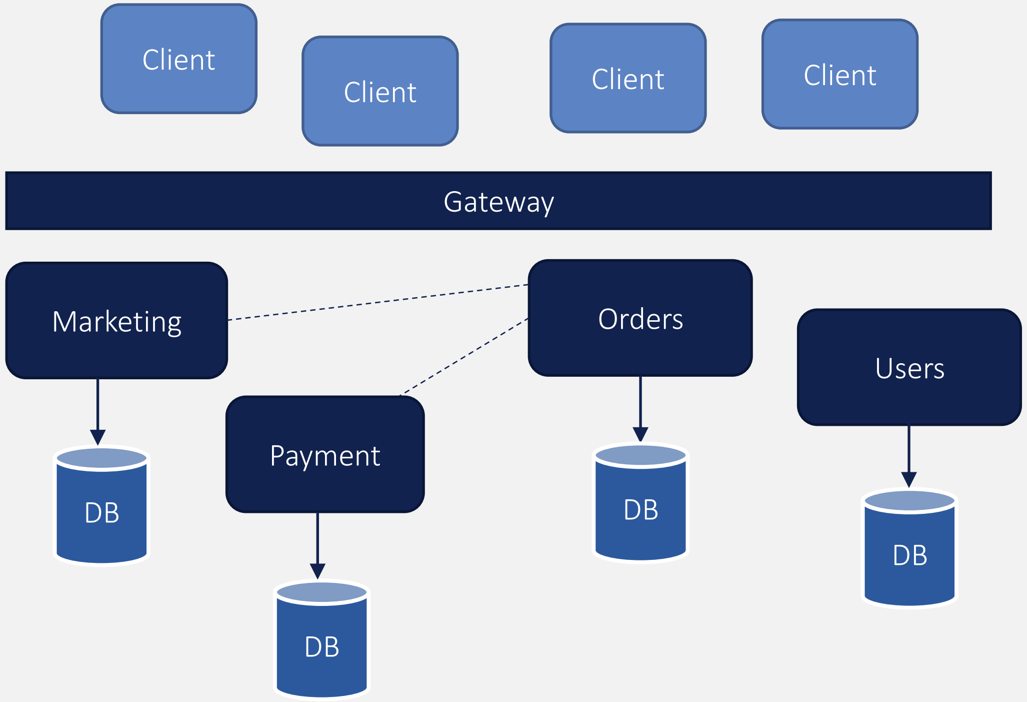
- Hard to maintain the modularity
- Bigger == more complex
- Scale out the whole application
- Fault tolerance
- Update to new technology
- Multiple teams





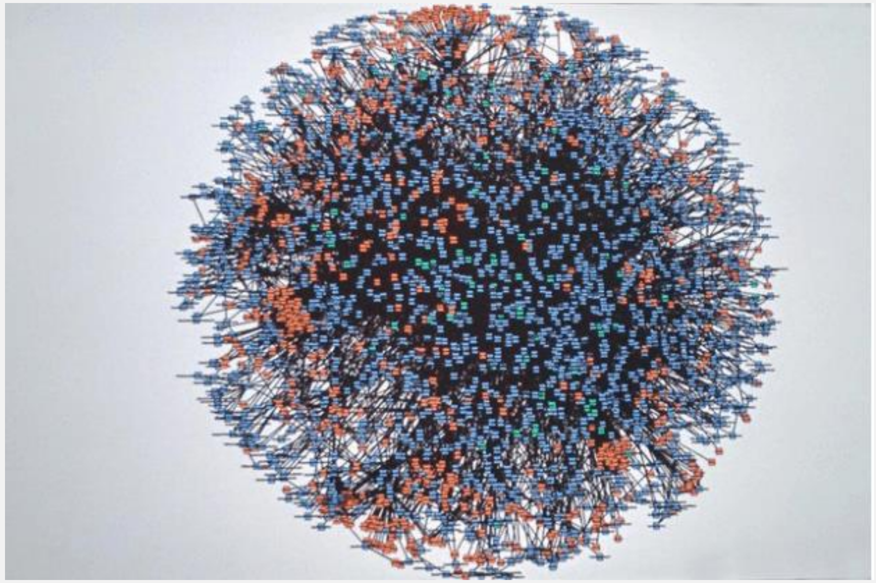
What is Microservices architecture?

- Microservices are small, independent, and loosely coupled
- Each service is a separate codebase, which can be managed by a small development team
- Services can be deployed independently
- Services are responsible for persisting their own data or external state
- Services communicate with each other by using well-defined APIs
- Services communicate with each other by using well-defined APIs



Microservices Examples

- Amazon



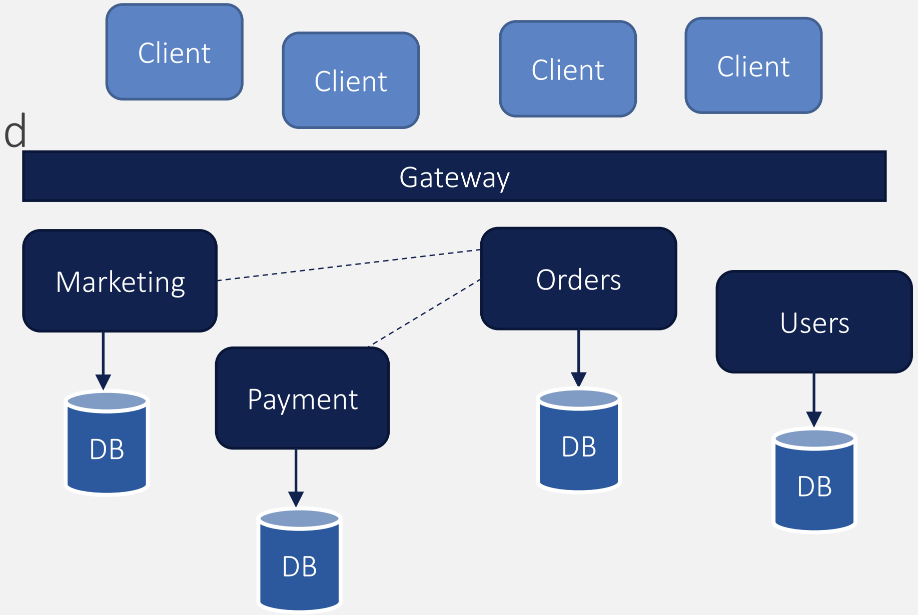
This is a 2008 graphic of Amazon’s microservices infrastructure

- **Netflix**: 110 million subscribers, 1 billion hours of video each week, 5 million new subscribers per quarter. Uses over 700 loosely coupled microservices
- **Uber**: 1300 microservices

Benefits and Challenges

Benefits

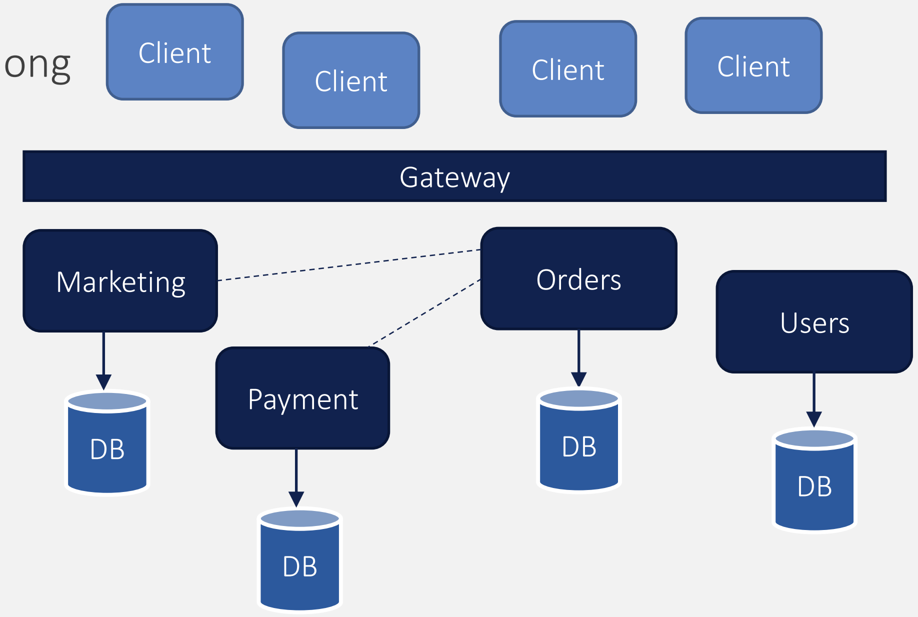
- **Small, focused teams**: A microservice should be small enough that a single feature team can build, test, and deploy it.
- **Agility**: easier to manage bug fixes and feature releases.
- **Small code base**
- **Mix of technologies**
- **Fault isolation:** as long as any upstream microservices are designed to handle faults correctly
- **Scalability:** scale out subsystems that require more resources, without scaling out the entire application
- **Data isolation:** easier to perform schema updates



Benefits and Challenges

Challenges

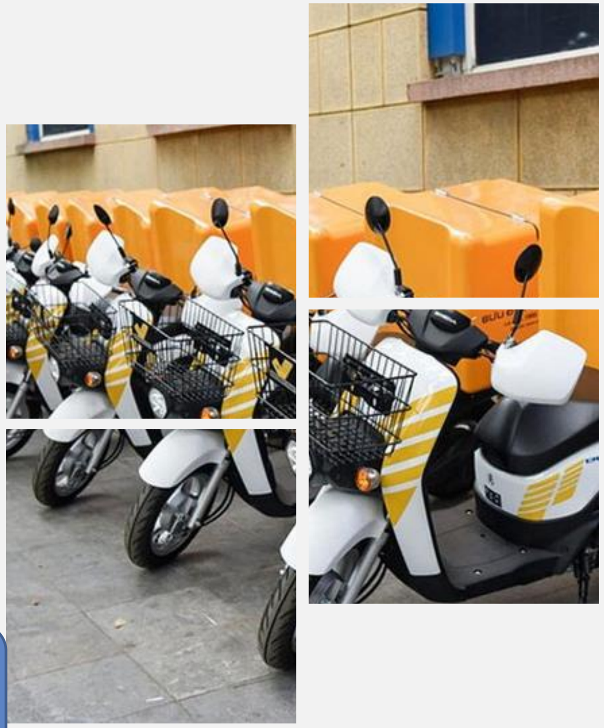
- **Complexity**: Each service is simpler, but the entire system as a whole is more complex.
- **Development and testing**: rely on other dependent services requires a different approach, refactoring across services, integration tests
- **Lack of governance:** many different languages and frameworks. It may be useful to put some project-wide standards
- **Network congestion and latency:** chain of service dependencies gets too long (service A calls B, which calls C...), avoid overly chatty APIs
- **Data integrity:** data consistency can be a challenge
- **Management:** requires a mature DevOps culture
  Correlated logging across services
- **Versioning:** Updates to a service must not break services
- **Skill set**

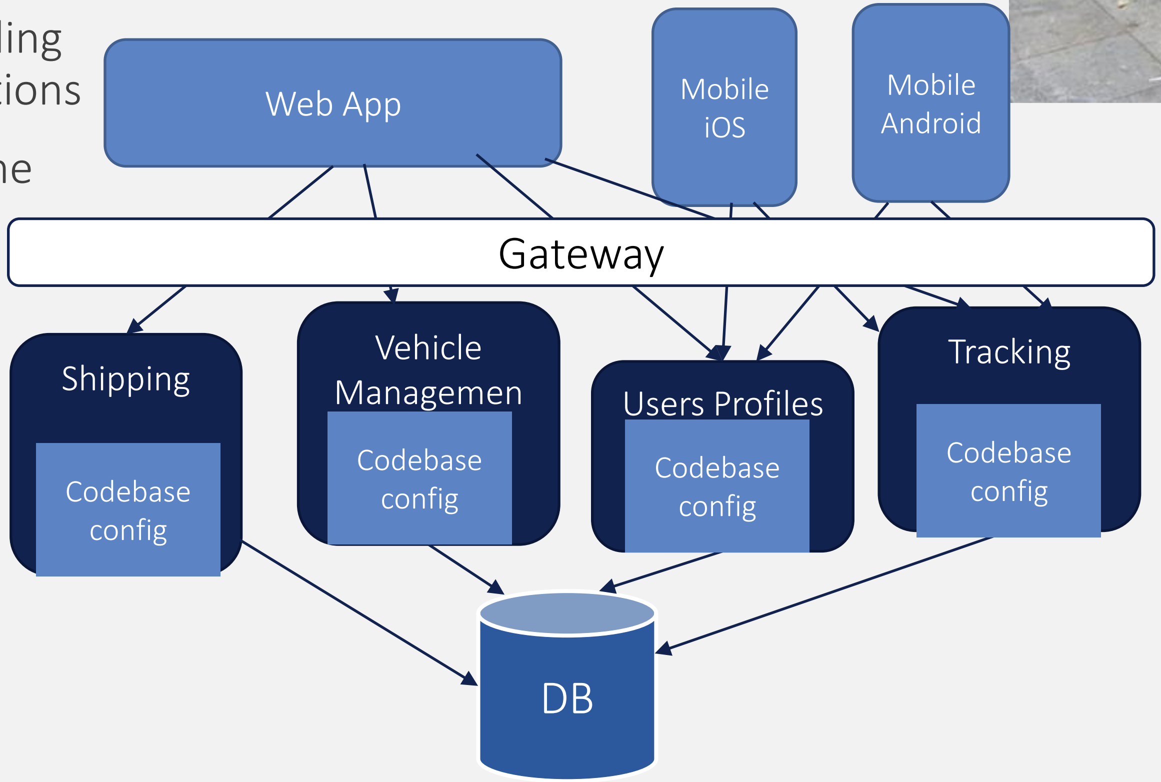


Tackling the business case (cont.)

Distributed Monolithic Architecture

- Independent services
- Shared database
- Must be deployed and released at once
- Bundled into single package including codebase , libraries and configurations
- Introduce API Gateway to overcome clients' maintenance



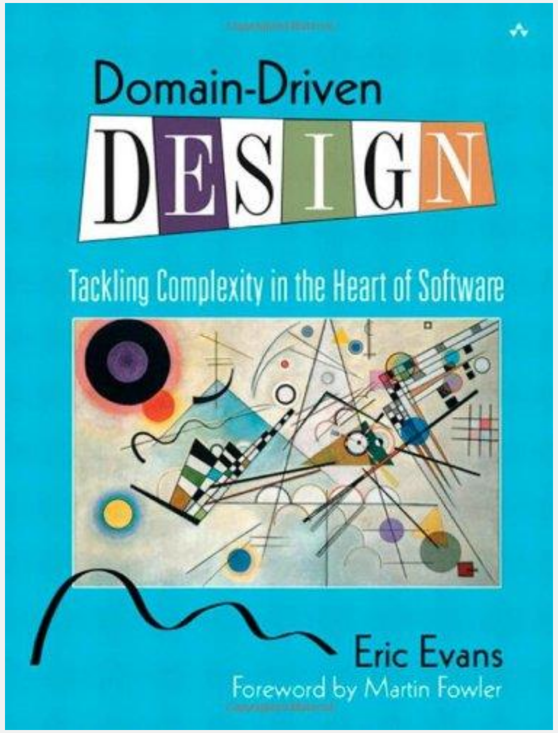


Domain Modeling for Microservices

Software Complexity Types

- Technical complexity
- Amount of data
- Performance
- Business logic complexity → DDD

Domain Driven Design can handle complex business logic complexity in such away that it would be possible to extend, maintained and keep it simple

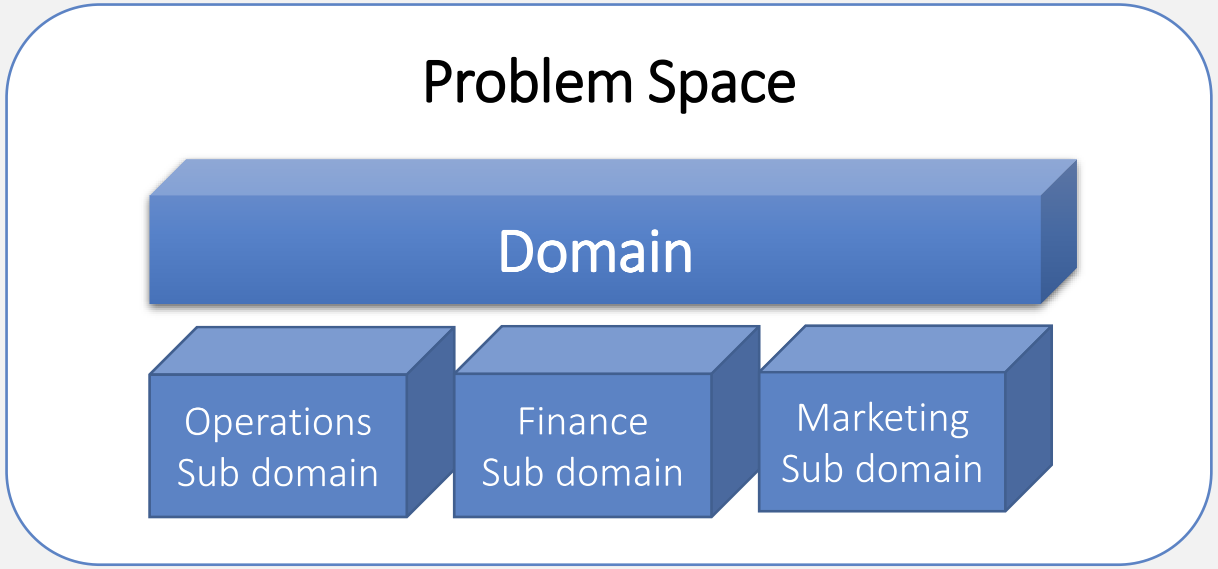

Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Domain Modeling for Microservices

Domain

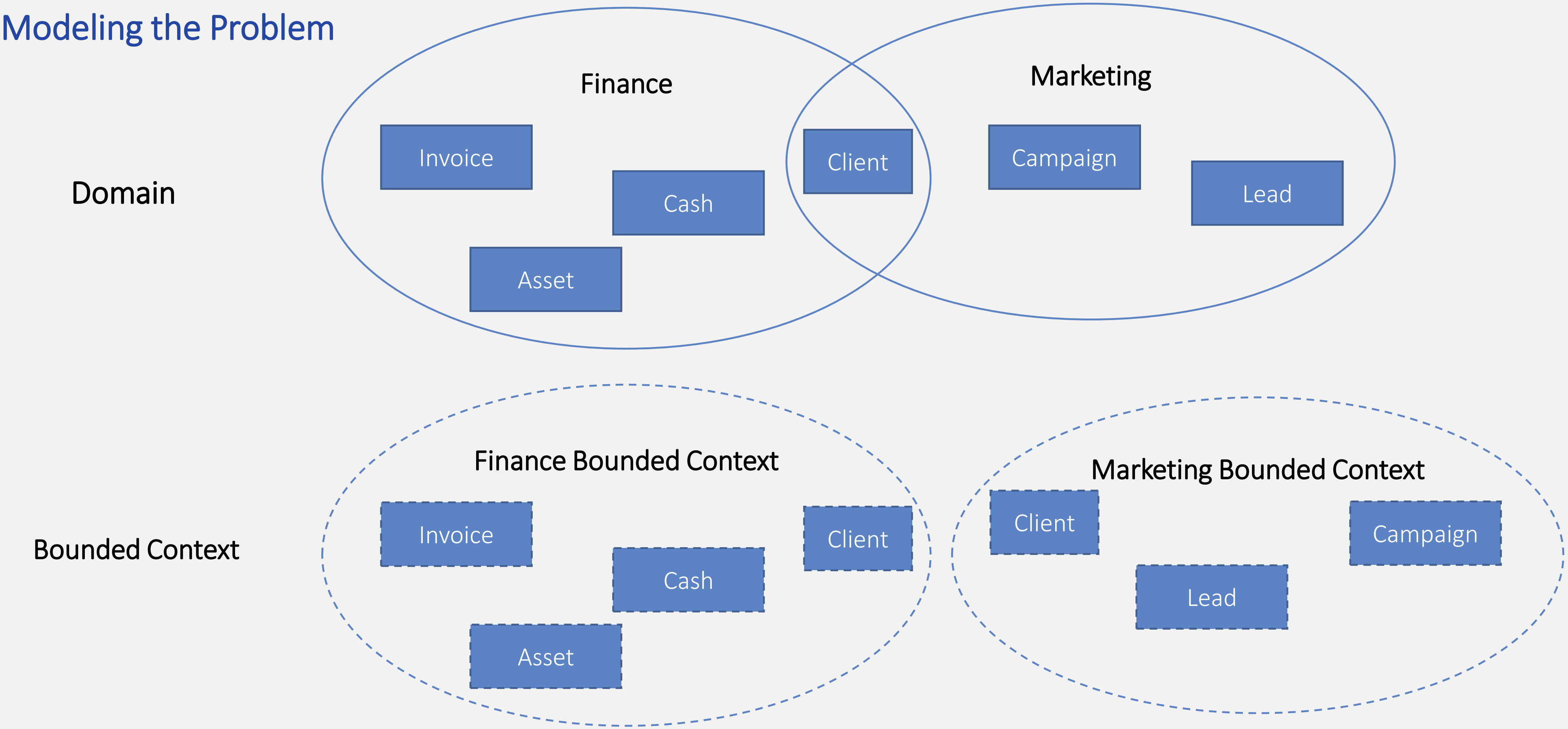
Field of study that defines a set of common requirements, terminology, and functionality for any software program

Ubiquitous Language



Domain Modeling for Microservices

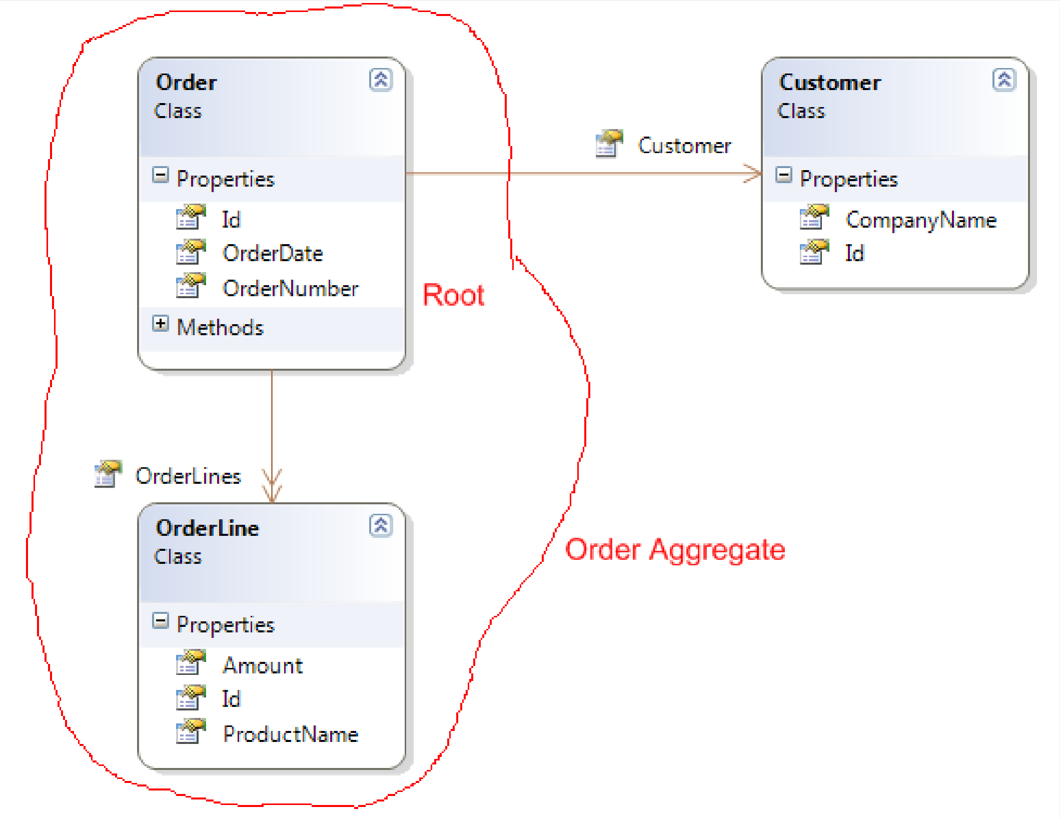
Modeling the Problem



Domain Modeling for Microservices

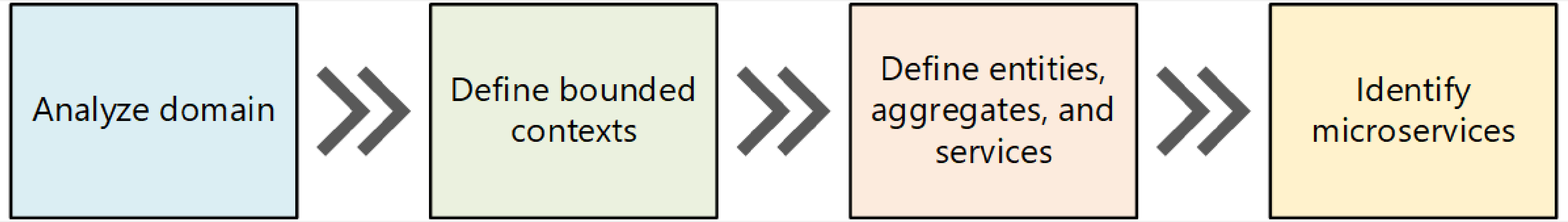
Aggregate

A DDD aggregate is a cluster of domain objects that can be treated as a single unit



Domain Modeling for Microservices

Domain-driven design (DDD) provides a framework that can get you most of the way to a set of well-designed microservices

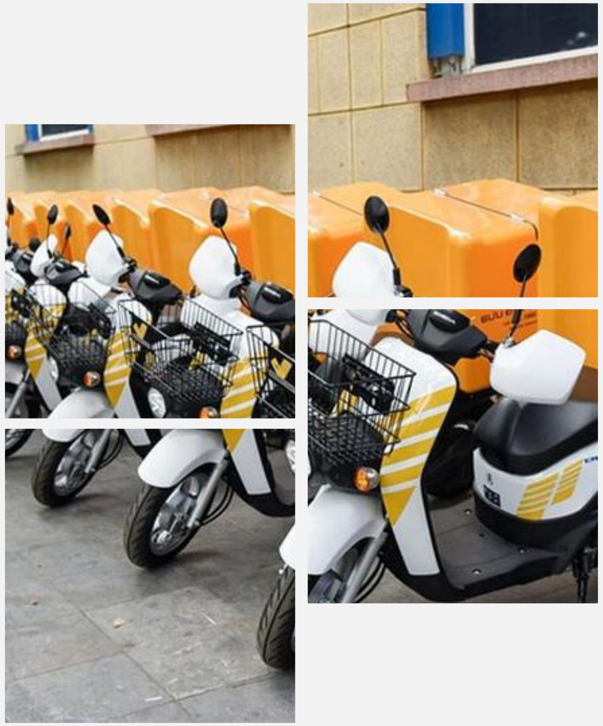


Analyze domain

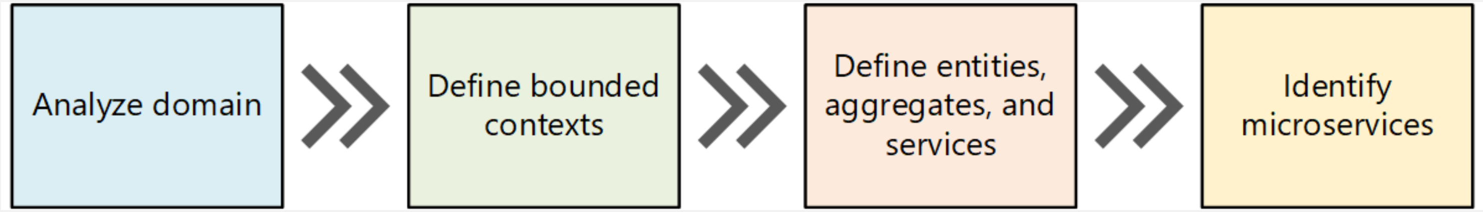
Define bounded contexts

Define entities, aggregates, and services

Identify microservices



Domain Modeling for Microservices



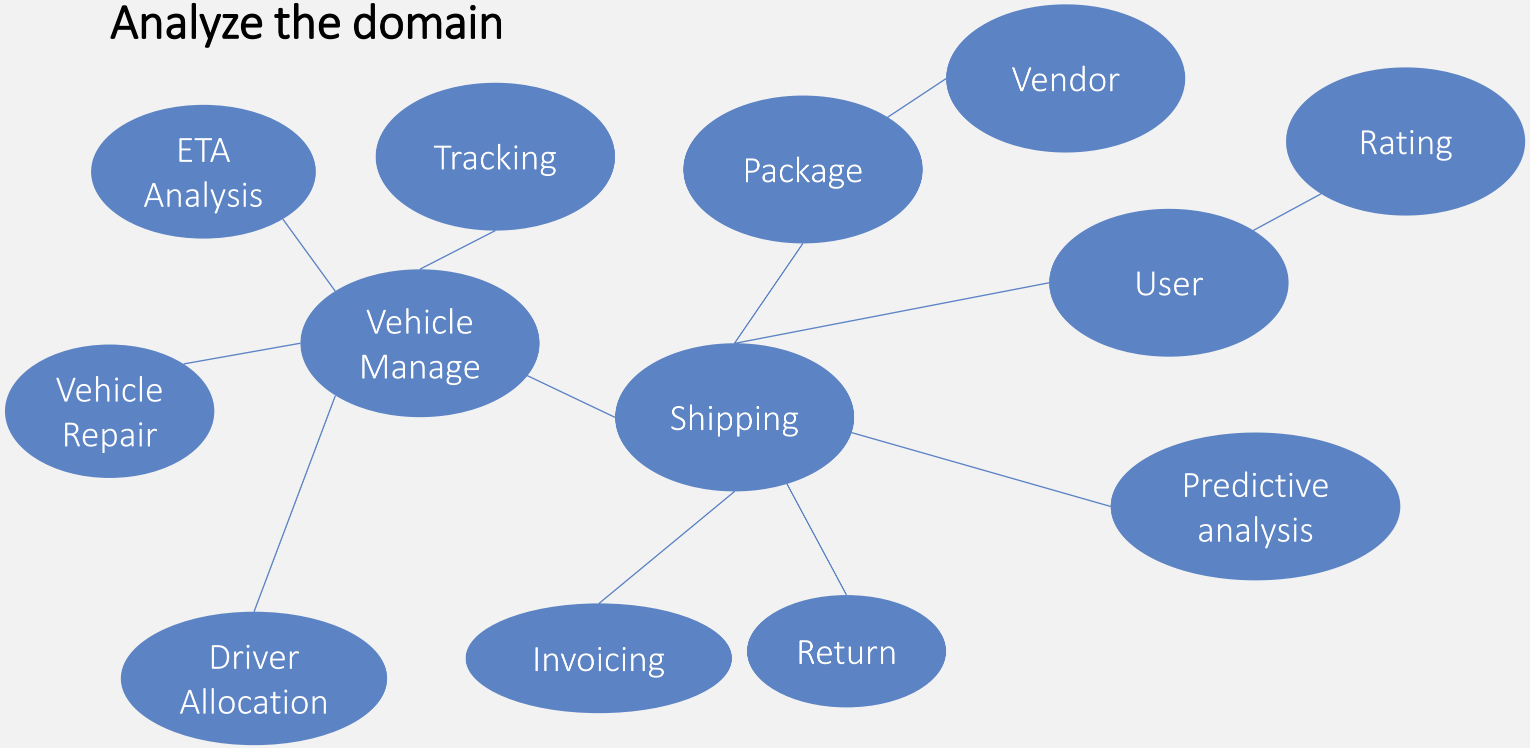
Analyze domain

Define bounded contexts

Define entities, aggregates, and services

Identify microservices

Analyze the domain



ETA Analysis

Tracking

Package

Vendor

Rating

User

Vehicle Manage

Vehicle Repair

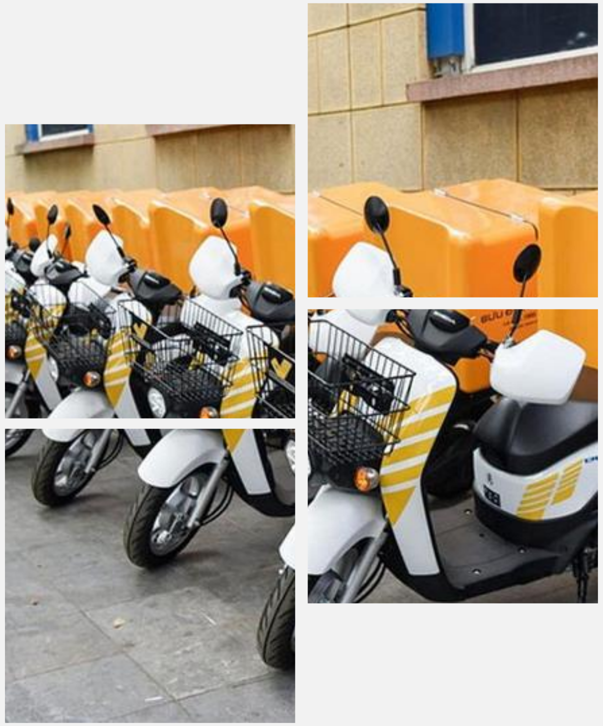
Shipping

Predictive analysis

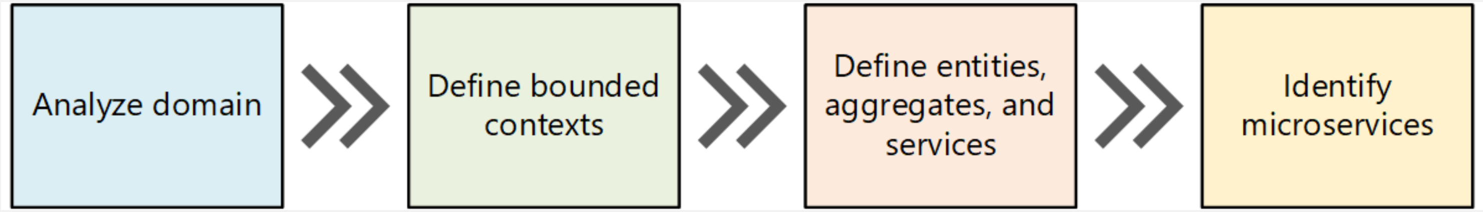
Driver Allocation

Invoicing

Return



Domain Modeling for Microservices

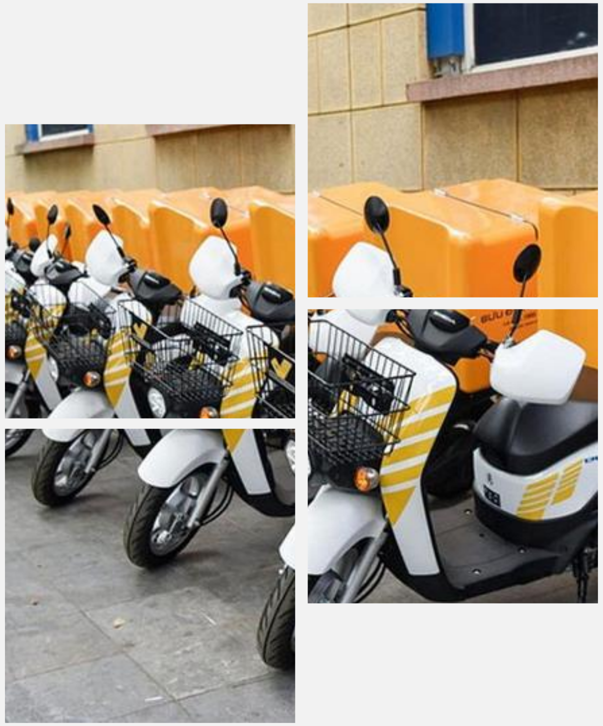


Analyze domain

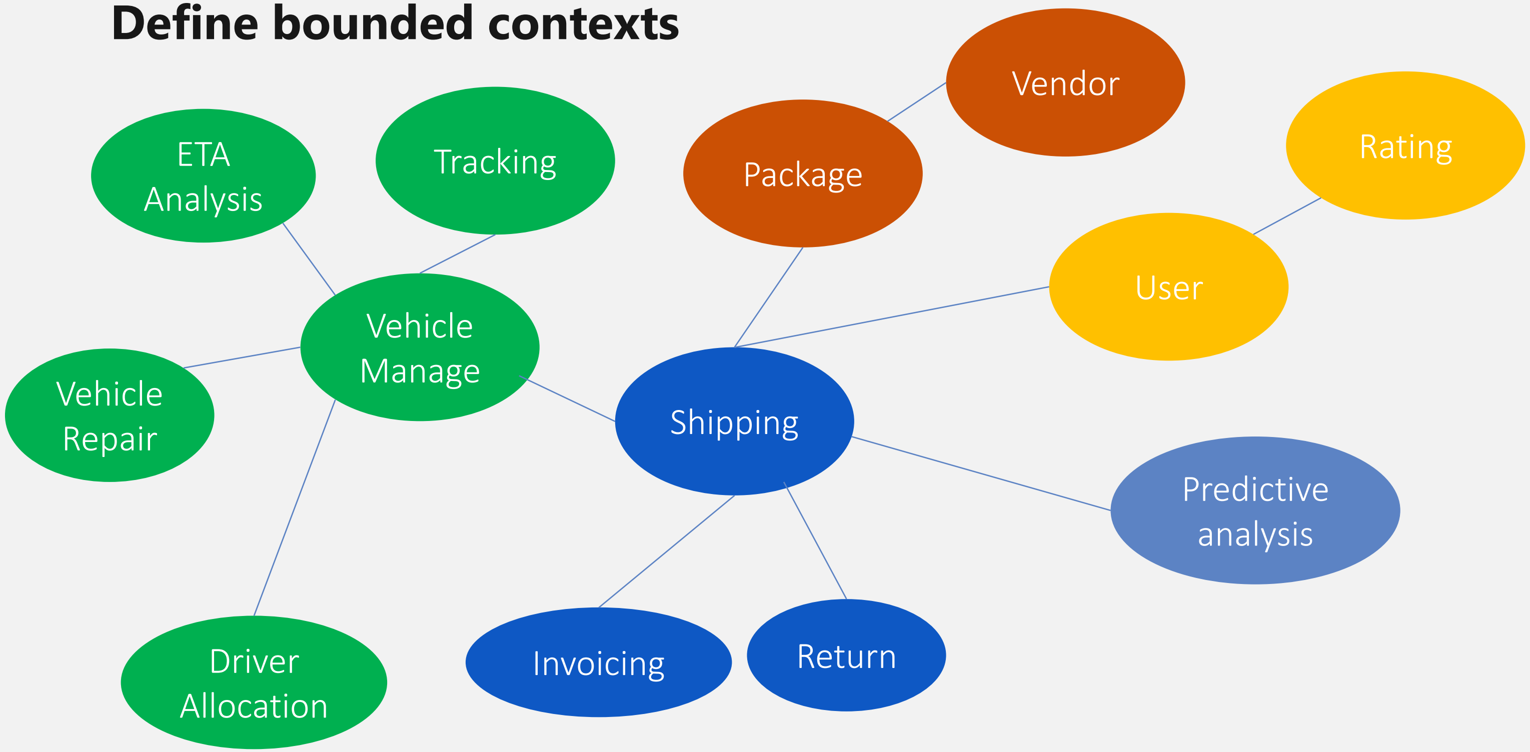
Define bounded contexts

Define entities, aggregates, and services

Identify microservices



Define bounded contexts



Vendor

ETA Analysis

Tracking

Package

Rating

User

Vehicle Manage

Vehicle Repair

Shipping

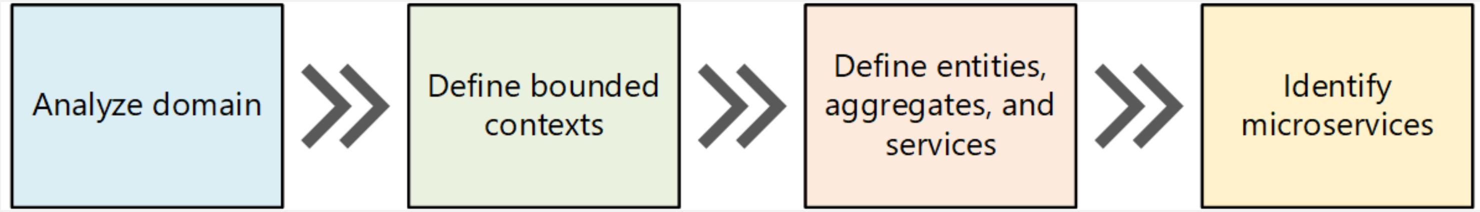
Predictive analysis

Driver Allocation

Invoicing

Return

Domain Modeling for Microservices

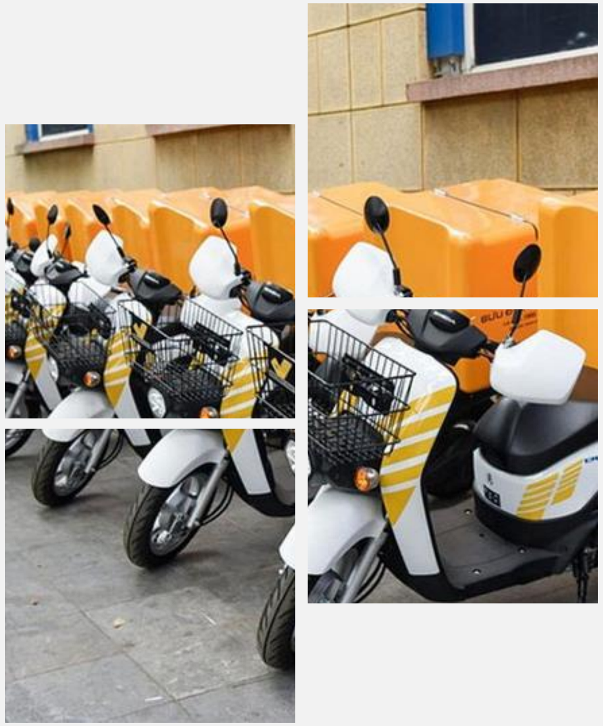


Analyze domain

Define bounded contexts

Define entities, aggregates, and services

Identify microservices

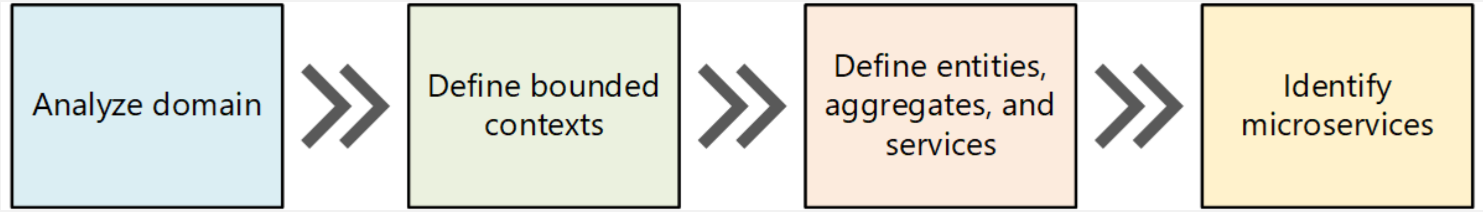


Define entities, aggregates, and services

- A customer can request a vehicle to pick up goods.
- The sender generates a tag (i.e. barcode) to put on the package.
- Pick up and deliver a package from the source location to the destination location
- The customer is notified when the delivery is completed.
- The sender can request delivery confirmation from the customer, in the form of a signature or finger print.

Shipping

Domain Modeling for Microservices



Analyze domain

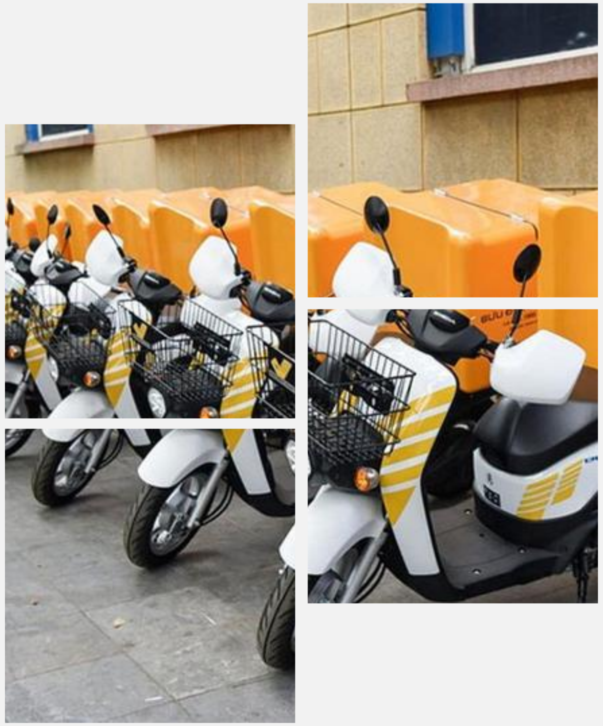
Define bounded contexts

Define entities, aggregates, and services

Identify microservices

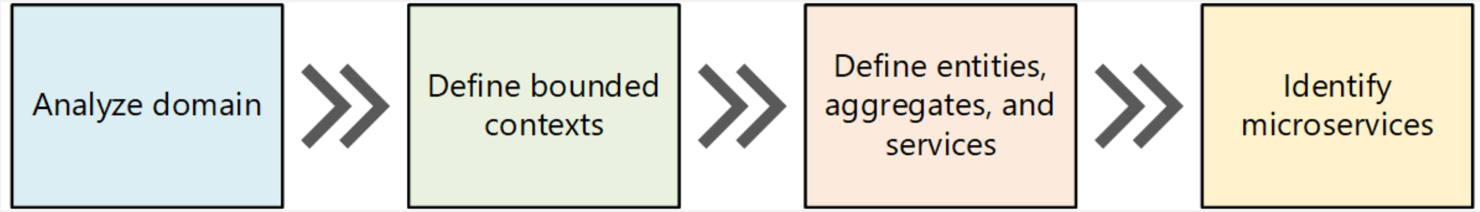
Define entities, aggregates, and services

- **Entities**:
  - Delivery
  - Package
  - Vehicle
  - Account
  - Confirmation
  - Notification
  - Tag
- **Aggregates**: Delivery, Package, Vehicle, and Account



Shipping

Domain Modeling for Microservices

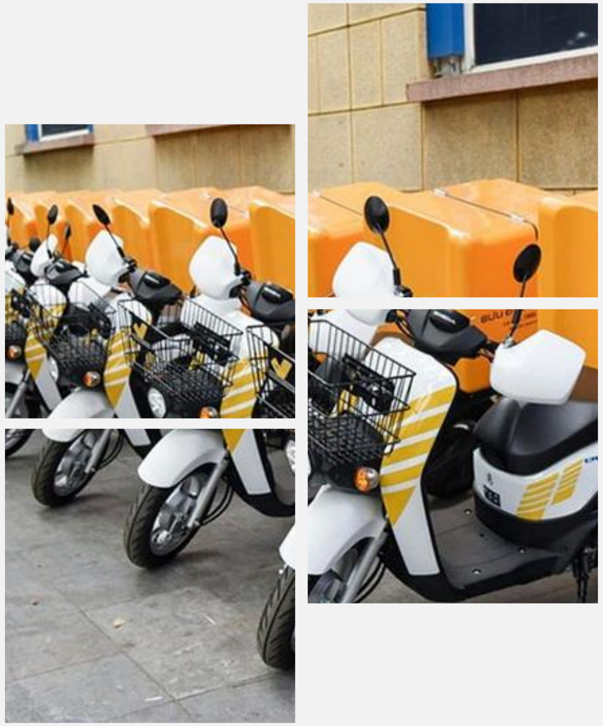


Analyze domain

Define bounded contexts

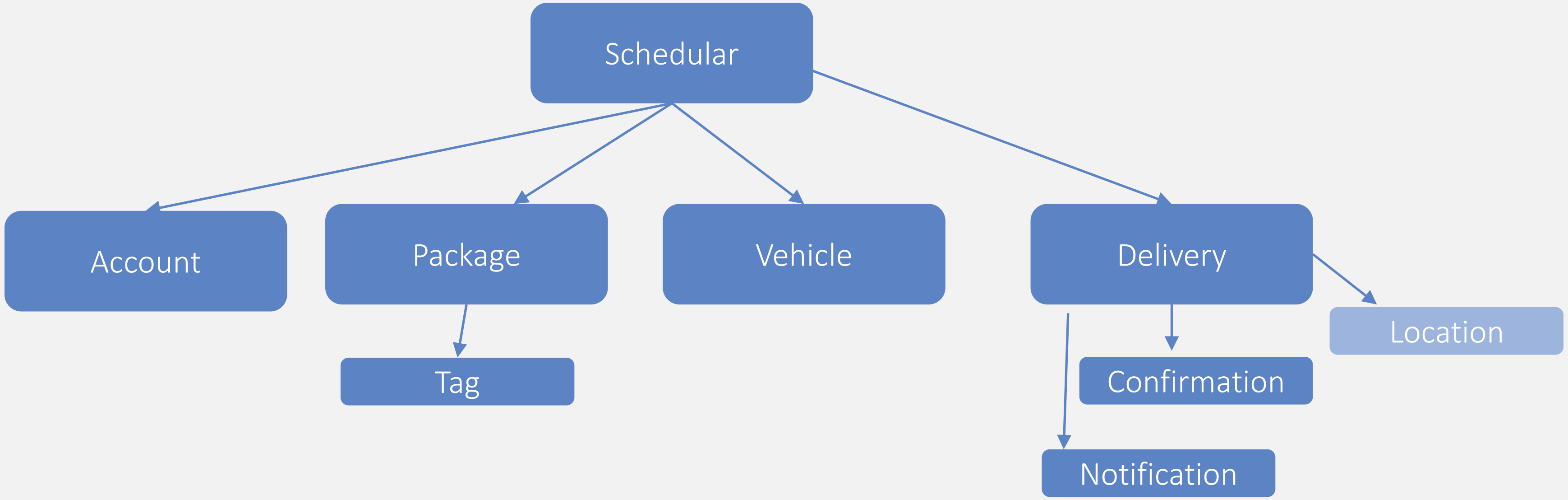
Define entities, aggregates, and services

Identify microservices



Define entities, aggregates, and services

- **Domain Service**: Schedular



Schedular

Account

Package

Vehicle

Delivery

Tag

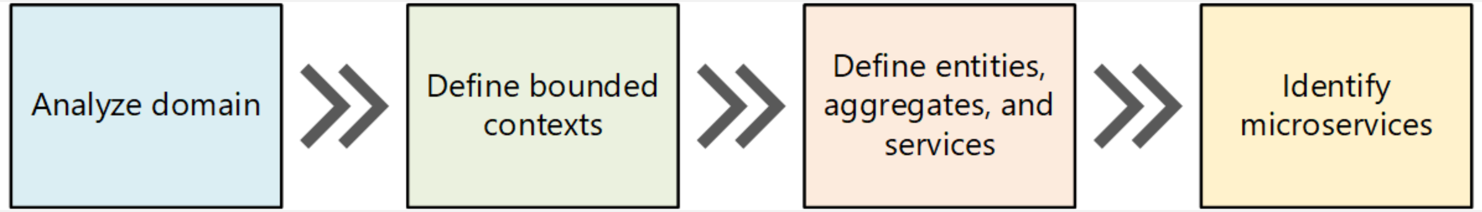
Location

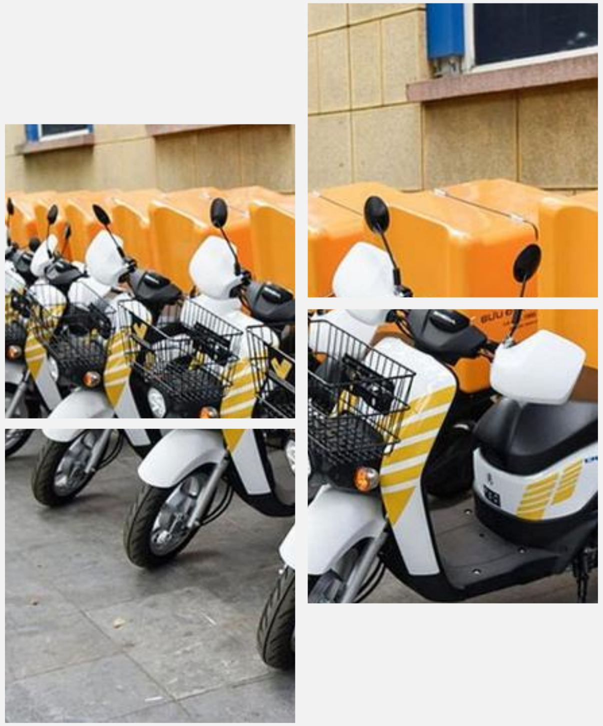
Confirmation

Notification

Shipping

Domain Modeling for Microservices

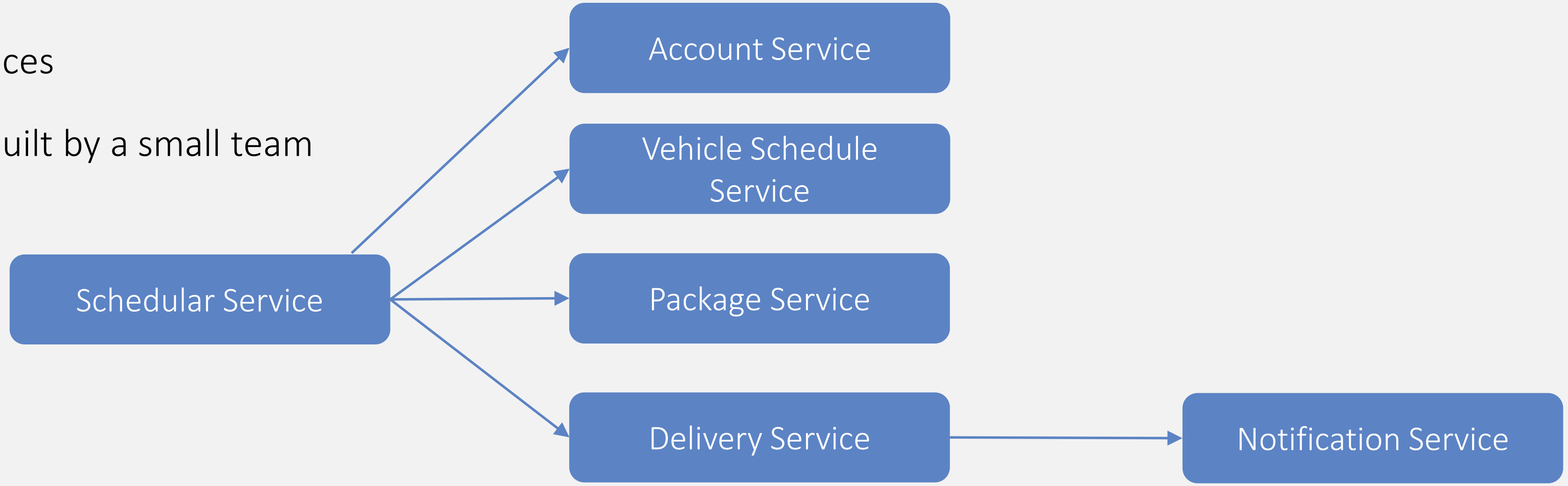




Identify microservices

"not too big and not too small"

- Each service has a single responsibility.
- No chatty calls between services
- Small enough that it can be built by a small team


Account Service
Vehicle Schedule Service
Schedular Service
Package Service
Delivery Service
Notification Service