BITS Pilani
Pilani Campus

Microservices Contd.

Akanksha Bharadwaj
Asst. Professor, CSIS DEpartment

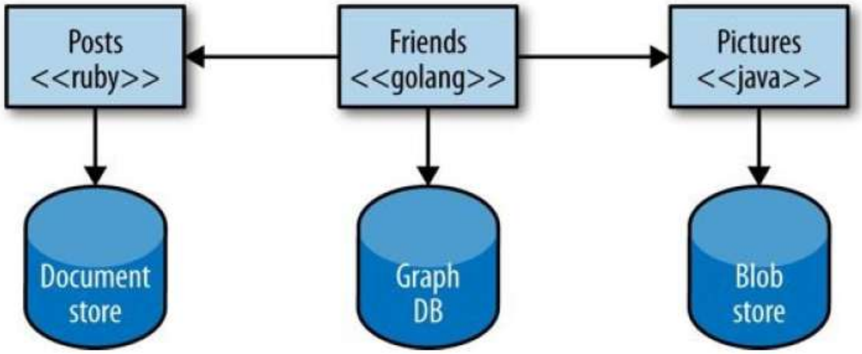
innovate achieve lead

BITS Pilani
Pilani Campus

SE ZG583, Scalable Services
Lecture No. 5

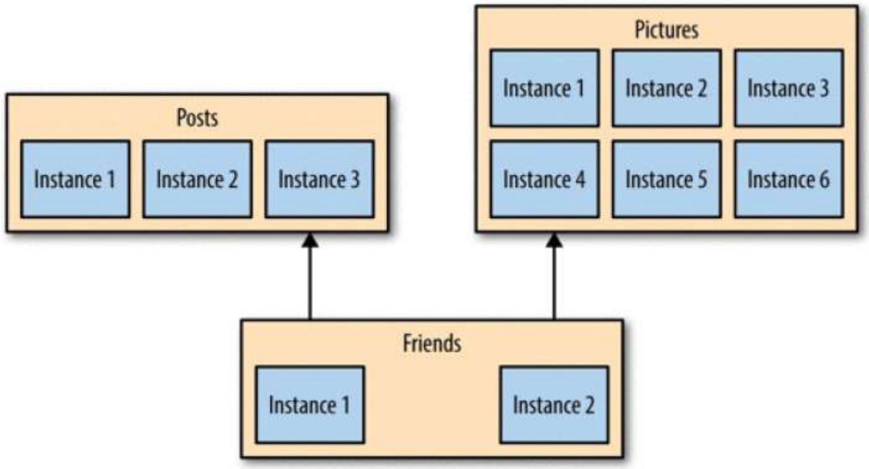
Advantages of Microservices

- **Technology Heterogeneity**
  - Different microservices can use different programming languages, databases, or frameworks.
  - Teams can choose the best technology for each service.



Advantages contd.

- **Resilience**: A failure in one microservice does not necessarily bring down the entire system.
- **Faster Development & Deployment:** Small, independent teams can develop, test, and deploy microservices independently
- **Scalability:** Each microservice can be scaled independently based on demand.



Advantages contd.

- **Better Maintainability & Modularity:** Codebases remain smaller and more manageable
- **Enhanced Security:** Sensitive functionalities can be isolated and secured independently.
- **Support for DevOps & Agile Methodologies:** Encourages a culture of continuous improvement and iteration

Microservices is not a silver bullet

- Increased Operational Complexity
- Inter-Service Communication Overhead
- Data Consistency & Distributed Transactions
- Security Risks
- Higher Infrastructure & Maintenance Costs
- Steep Learning Curve
- Not Always the Best Fit

Example Case Study

- **Spotify** uses a modular monolith rather than fully distributed microservices. This helps balance modularity with simplicity in operations.

- **GitLab** maintains a large monolithic codebase but with modular design principles to manage complexity and speed development

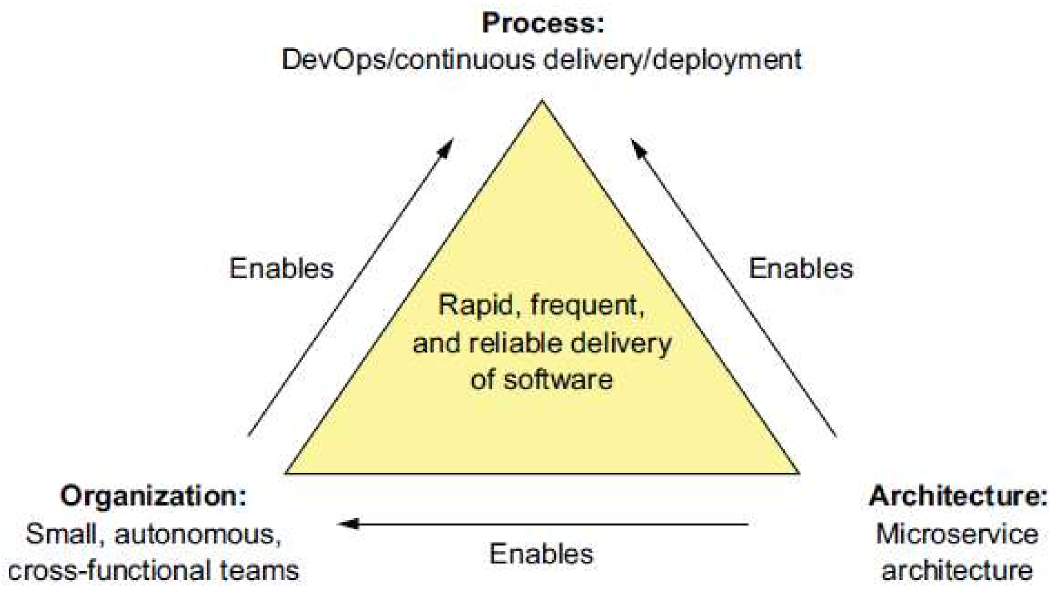
innovate achieve lead

BITS Pilani

Pilani Campus

Process and Organization

Introduction


Process:
DevOps/continuous delivery/deployment
Enables
Enables
Rapid, frequent, and reliable delivery of software
Organization:
Small, autonomous, cross-functional teams
Enables
Architecture:
Microservice architecture

Software development and delivery organization

- Success means that the engineering team will grow
- The solution is to refactor a large single team into a team of teams.
- The velocity of the team of teams is significantly higher than that of a single large team.
- Moreover, it’s very clear who to contact when a service isn’t meeting its SLA.

Software development and delivery process

- If you want to develop an application with the microservice architecture, it’s essential that you adopt agile development and deployment practices

- Practice continuous delivery/deployment, which is a part of DevOps.

- A key characteristic of continuous delivery is that software is always releasable

The human side of adopting microservices

- Ultimately, it changes the working environment of people and thus impact them emotionally

Three stage transition model

- Ending, Losing, and Letting Go
- The Neutral Zone
- The New Beginning

innovate achieve lead

BITS Pilani
Pilani Campus

Microservices Design Principles

Single Responsibility Principle (SRP)

- Sometimes it's important to maintain data consistency by putting functionality into a single microservice.
- Each microservice implements only one business responsibility from the bounded domain context.
- The rule of thumb is

“Gather together those things that change for the same reason, and separate those things that change for different reasons.” — Robert C. Martin

Abstraction and Information Hiding

- A service should only be consumed through a standardized API and should not expose its internal implementation details to its consumers

Loose Coupling & High Cohesion

- Services should be loosely coupled to minimize dependencies.
- High cohesion ensures that each service is self-contained and modular.

Resilience & Fault Tolerance

- Each service is necessarily fault tolerant so that failures on the side of its collaborating services will have minimal impact

Observability & Monitoring

- Implement logging, monitoring, and tracing (e.g., Prometheus, ELK Stack).
- Enable distributed tracing to track request flows across microservices.

Statelessness

- Services should be stateless and avoid storing session data.

- Use external caching mechanisms (e.g., Redis, Memcached) for performance optimization.

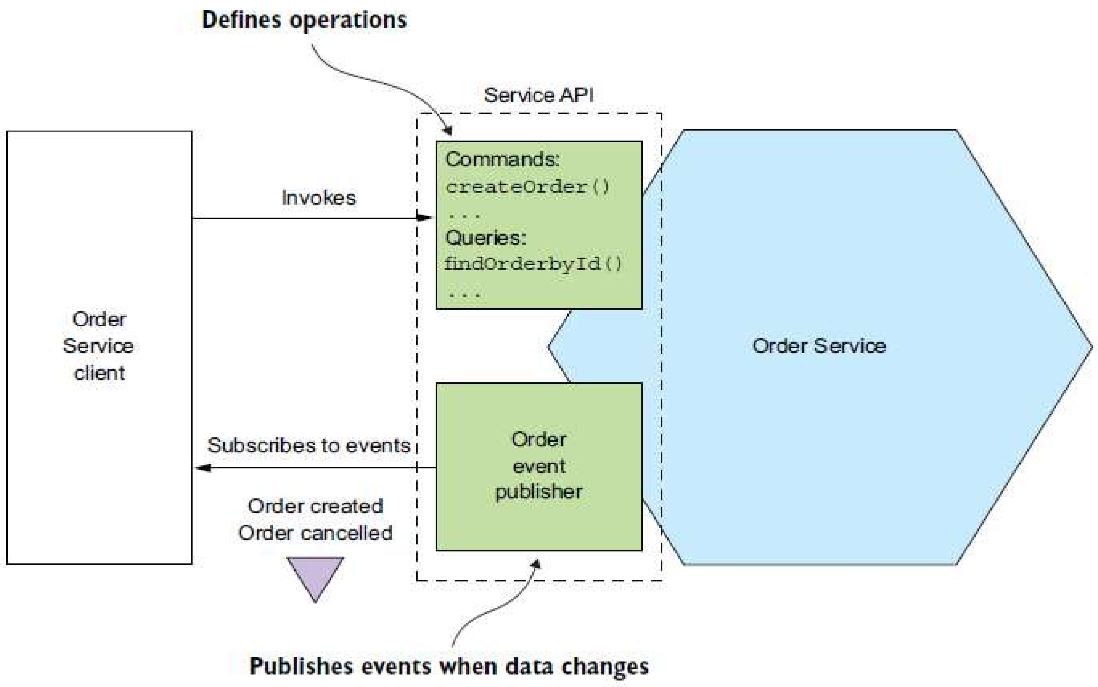
innovate achieve lead

BITS Pilani
Pilani Campus

Important concepts

What is a Service?

- It is a standalone, independently deployable software component that implements some useful functionality



Size of a Service

- Each service should be small enough to be independently developed, deployed, and scaled, yet large enough to encapsulate a meaningful business capability.

✓ **Too Small → Increased Complexity:** Leads to too many inter-service calls, higher network latency, and dependency issues.
✓ **Too Large → Monolithic Behavior:** Reduces scalability, reusability, and independent deployments.
✓ **Right Size → Business-Oriented, Autonomous, Scalable**

Services: The role of Shared Libraries

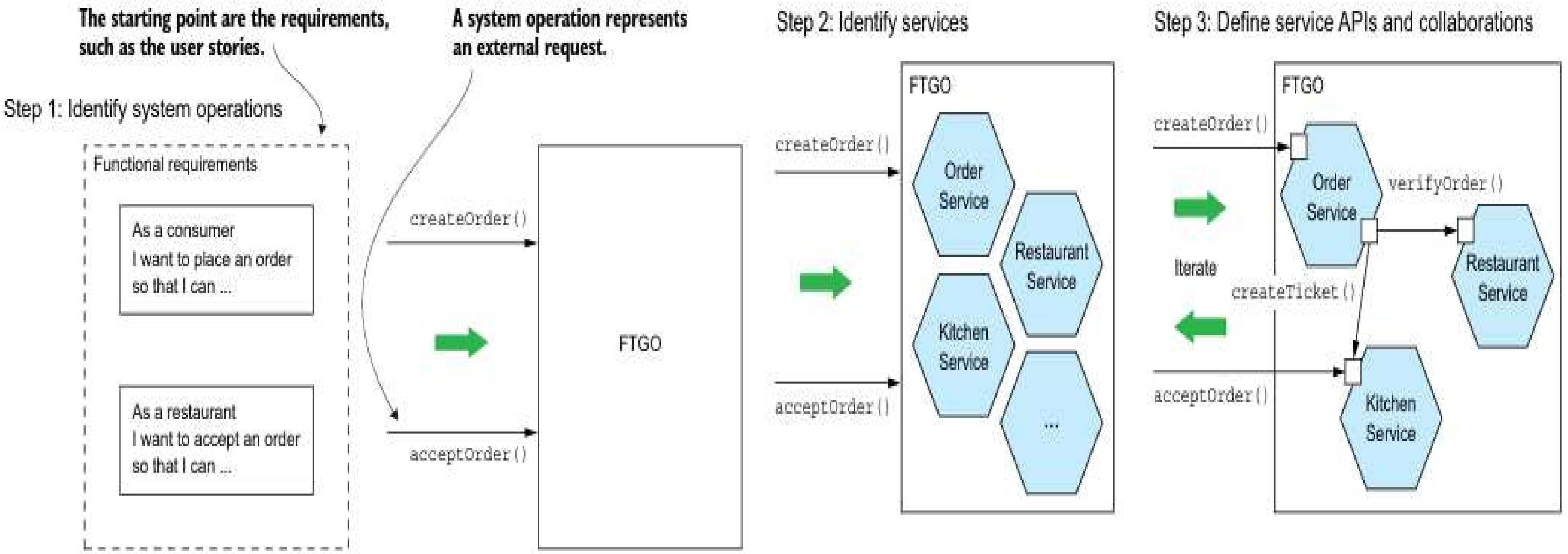
- On the surface, it looks like a good way to reduce code duplication in your services.
- But you need to ensure that you don't accidentally introduce coupling between your services.
- You should strive to use libraries for functionality that's unlikely to change.

Steps for defining an application's microservice architecture

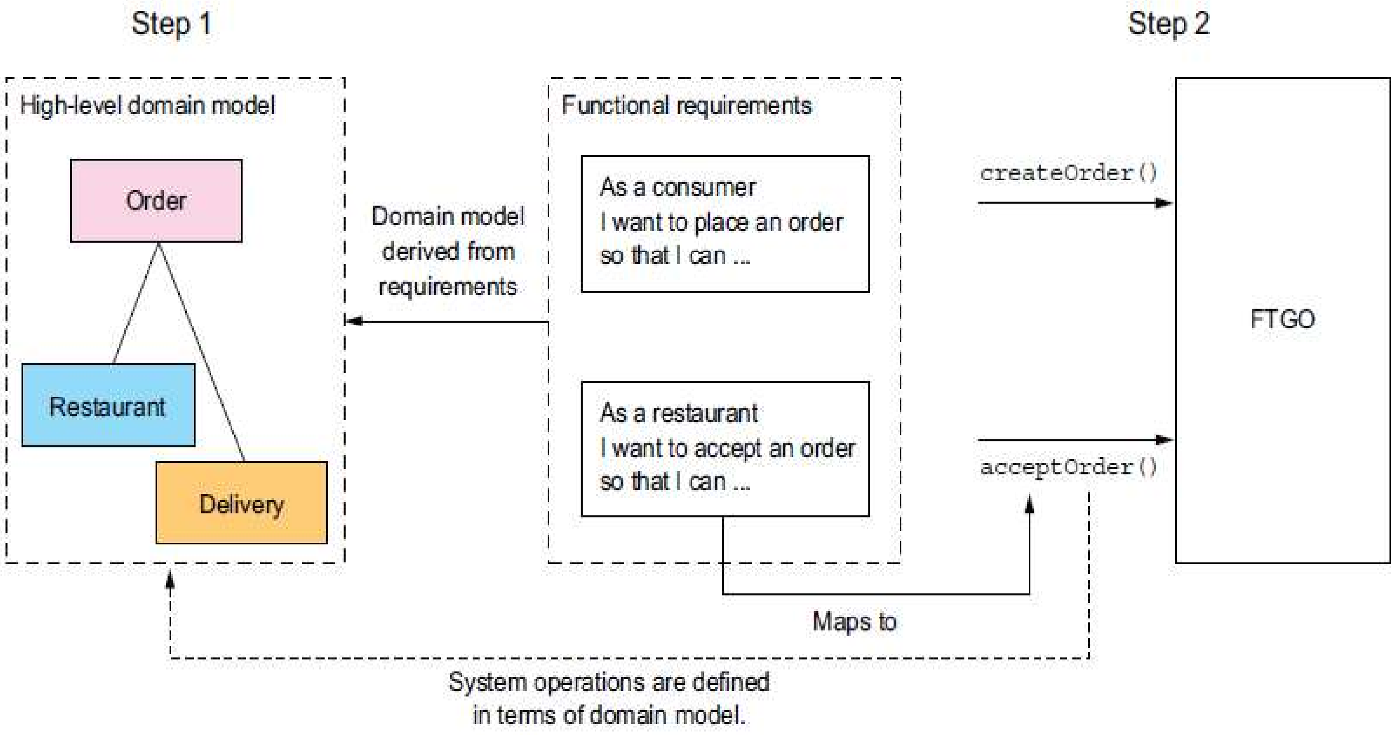
Step 1: Identify system operations

Step 2: Identify services

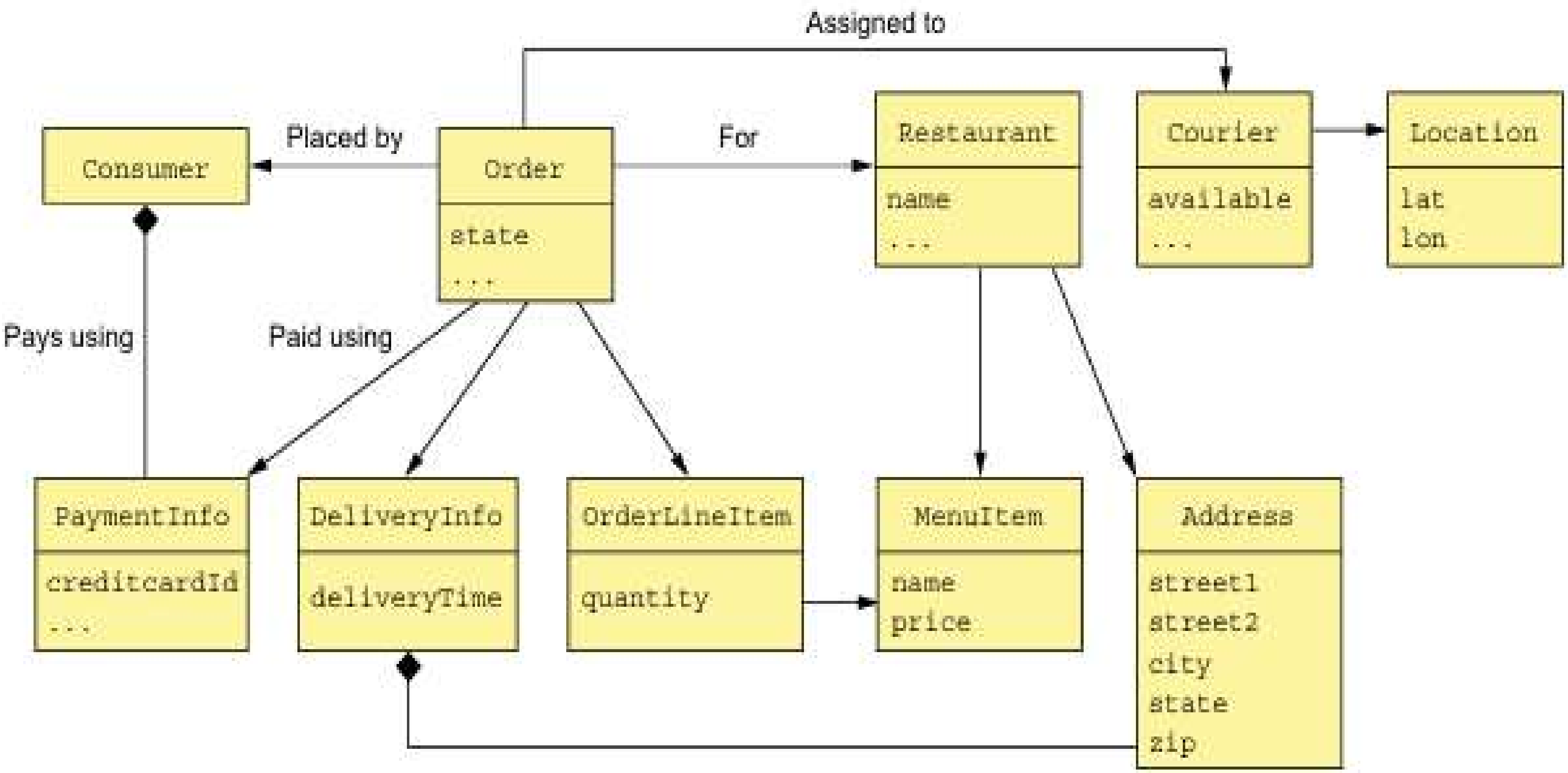
Step 3: Define service APIs and collaborations


The starting point are the requirements, such as the user stories.
A system operation represents an external request.
Step 1: Identify system operations
Functional requirements
As a consumer
I want to place an order
so that I can ...
As a restaurant
I want to accept an order
so that I can ...
createOrder()
acceptOrder()
FTGO
Step 2: Identify services
FTGO
createOrder()
acceptOrder()
Order Service
Restaurant Service
Kitchen Service
...
Step 3: Define service APIs and collaborations
FTGO
createOrder()
Iterate
createTicket()
acceptOrder()
Order Service
verifyOrder()
Restaurant Service
Kitchen Service

Step1: Identify system operations


Step 1
Step 2
High-level domain model
Order
Restaurant
Delivery
Domain model derived from requirements
Functional requirements
As a consumer I want to place an order so that I can ...
As a restaurant I want to accept an order so that I can ...
createOrder()
acceptOrder()
FTGO
Maps to
System operations are defined in terms of domain model.

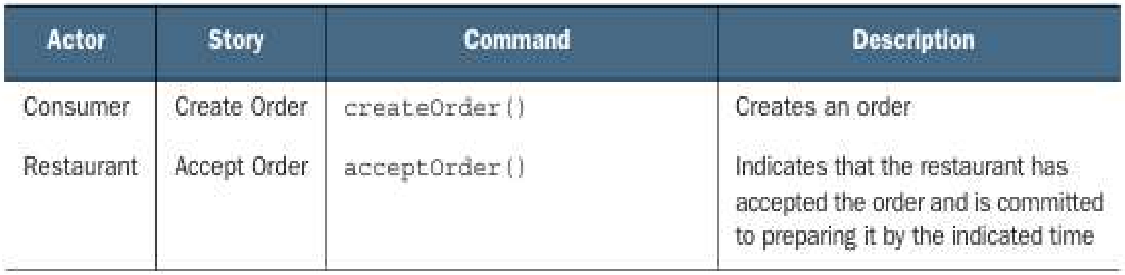
FTGO domain model



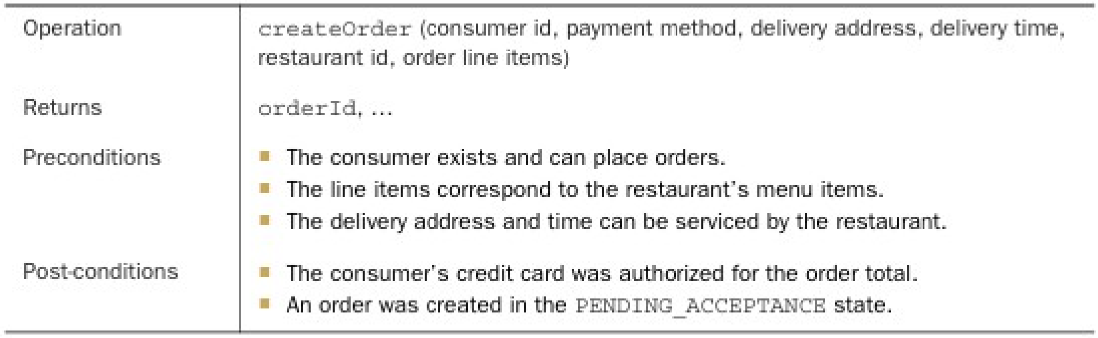
Defining system operations

There are two types of system operations:

- **Commands**—System operations that create, update, and delete data
- **Queries**—System operations that read (query) data



Actor	Story	Command	Description
Consumer	Create Order	`createOrder()`	Creates an order
Restaurant	Accept Order	`acceptOrder()`	Indicates that the restaurant has accepted the order and is committed to preparing it by the indicated time



Operation	`createOrder` (consumer id, payment method, delivery address, delivery time, restaurant id, order line items)
Returns	`orderId`, …
Preconditions	■ The consumer exists and can place orders. ■ The line items correspond to the restaurant's menu items. ■ The delivery address and time can be serviced by the restaurant.
Post-conditions	■ The consumer's credit card was authorized for the order total. ■ An order was created in the `PENDING_ACCEPTANCE` state.

Step 2 and 3: Defining services and service API

- One strategy is defining services by applying the Decompose by business capability pattern and another is defining services by applying the Decompose by sub-domain pattern
- The next step is to define each service’s API: its operations and events.
- A service API operation exists for one of two reasons: some operations correspond to system operations. They are invoked by external clients and perhaps by other services.
- The other operations exist to support collaboration between services
- The starting point for defining the service APIs is to map each system operation to a service.

innovate achieve lead

BITS Pilani
Pilani Campus

Decomposition based patterns to define services

Decompose by business capability pattern

- Business capability is something that a business does in order to generate value.
- **Example**: The capabilities of an online store include Order management, Inventory management, Shipping, and so on.

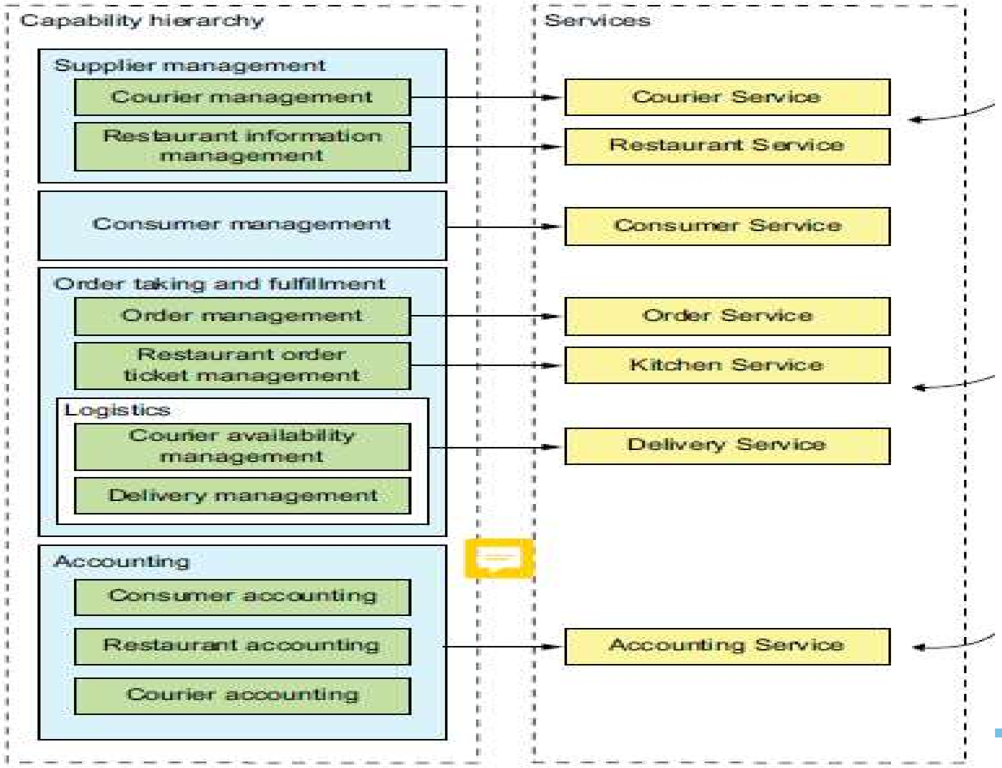
Identifying Business Capabilities

Business capabilities for FTGO include the following:

- Supplier management
- Consumer management
- Order taking and fulfilment
- Accounting

From business capabilities to services

- Once you've identified the business capabilities, you then define a service for each capability or group of related capabilities



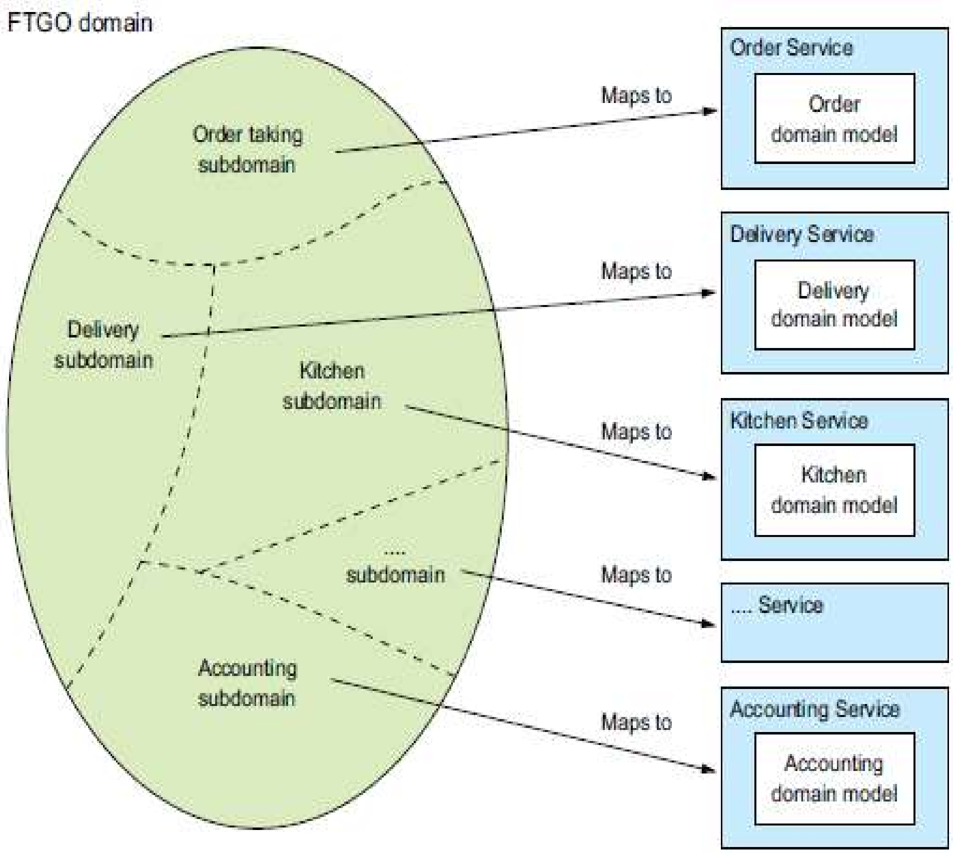
Decompose by sub-domain pattern

- DDD is an approach for building complex software applications centered on the development of an object-oriented, domain model.
- DDD has two concepts that are incredibly useful when applying the microservice architecture: **subdomains** and **bounded contexts**.

From Subdomains to Services

- DDD defines a separate domain model for each subdomain
- The examples of subdomains in FTGO include order taking, order management, restaurant order management, delivery, and financials.
- DDD calls the scope of a domain model a “bounded context.”
- When using the microservice architecture, each bounded context is a service or possibly a set of services.

Decompose by sub-domain pattern



Decomposition guidelines

- Single Responsibility Principle
- Common Closure Principle

Self Sudy

Article: https://kitrum.com/blog/is-microservice-architecture-still-a-trend/

Article: https://devops.com/microservices-amazon-monolithic-richixbw/

Article: https://www.linkedin.com/pulse/microservices-vs-monoliths-why-some-big-companies-moving-%C3%B6nden-btb4f/

Article: https://acquaintsoft.com/blog/microservice-scalability-success-vs-failure

References

- Chapter 2, Microservices Patterns by Chris Richardson
- Link: https://docs.microsoft.com/en-us/azure/architecture/microservices/design/
- Link: https://docs.microsoft.com/en-us/azure/architecture/microservices/design/patterns
- Amazon Prime Case Study: https://theodore.ie/the-monolith-returns-why-companies-are-moving-away-from-microservices-and-serverless-architecture/