Microservices Contd.

BITS Pilani
Pilani Campus

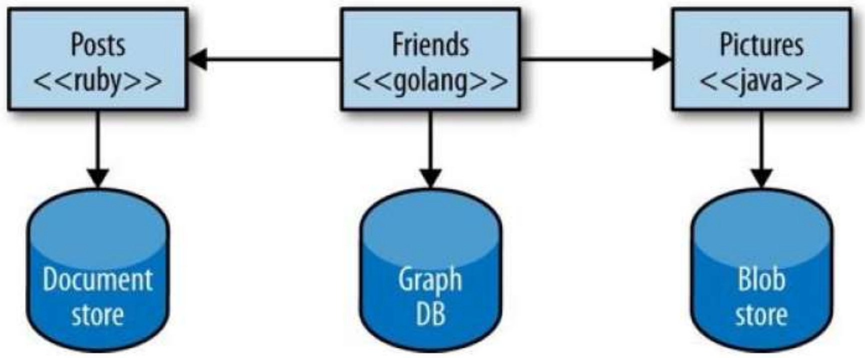
Akanksha Bharadwaj
Asst. Professor, CSIS DEpartment

BITS Pilani
Pilani Campus

SE ZG583, Scalable Services
Lecture No. 5

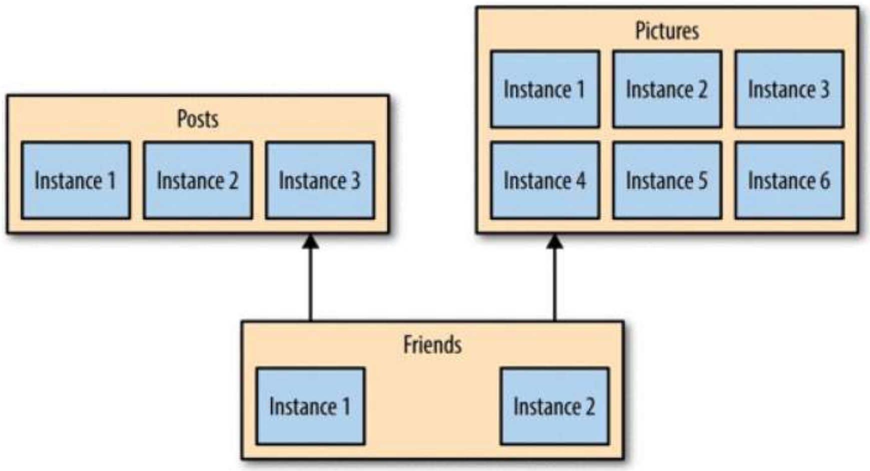
Advantages of Microservices

- Technology Heterogeneity



Advantages contd.

- Resilience
- Ease of Deployment
- Scaling



Advantages contd.

- Organizational Alignment
- Reusability
- Agility

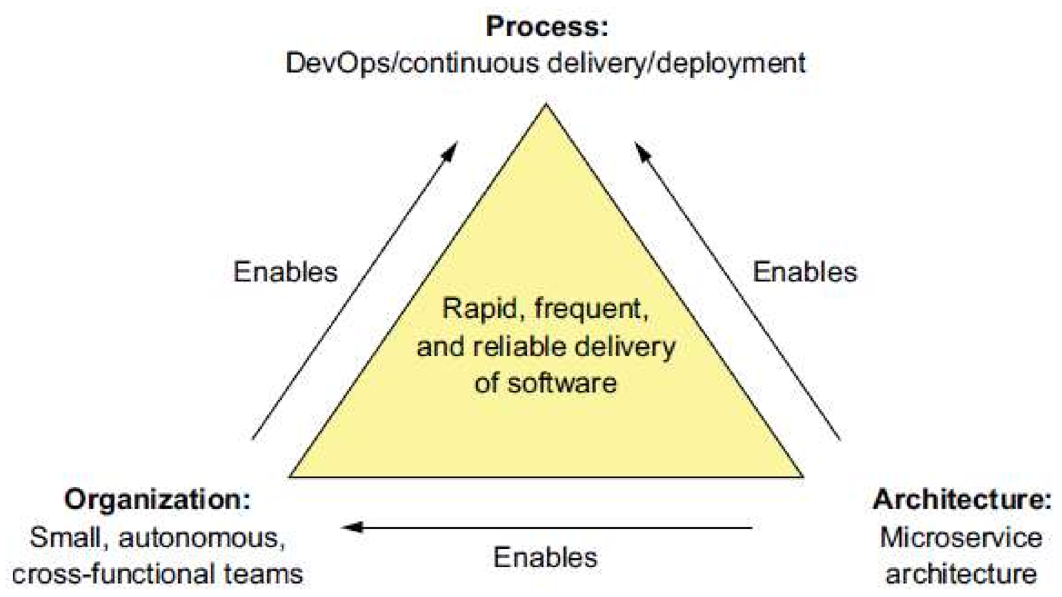
Microservices is not a silver bullet

- Complexity
- Finding the right services is challenging
- Development and testing
- Network congestion and latency
- Data integrity
- Careful Coordination
- Versioning

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

- 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

- Dependencies between services and their consumers are minimized with the application of the principle of loose coupling

Fault tolerance

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

Discoverability

- The aim of discoverability is to communicate a clear understanding of the business purpose and technical interface of the microservice.

Reusability

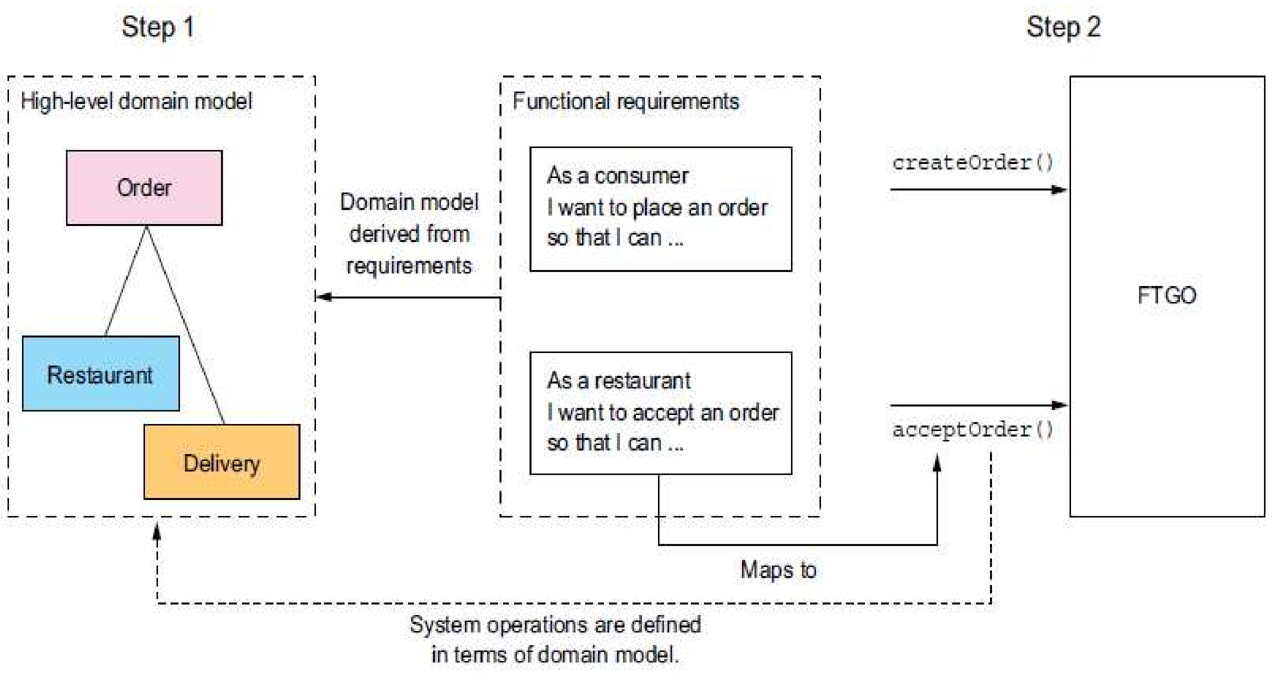
- Reuse continues to be a principle of microservice design. However, the scope of reuse has been reduced to specific domains within the business.

Steps for defining an application's microservice architecture

Step 1: Identify system operations

Step 2: Identify services

Step 3: Define service APIs and collaborations



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.

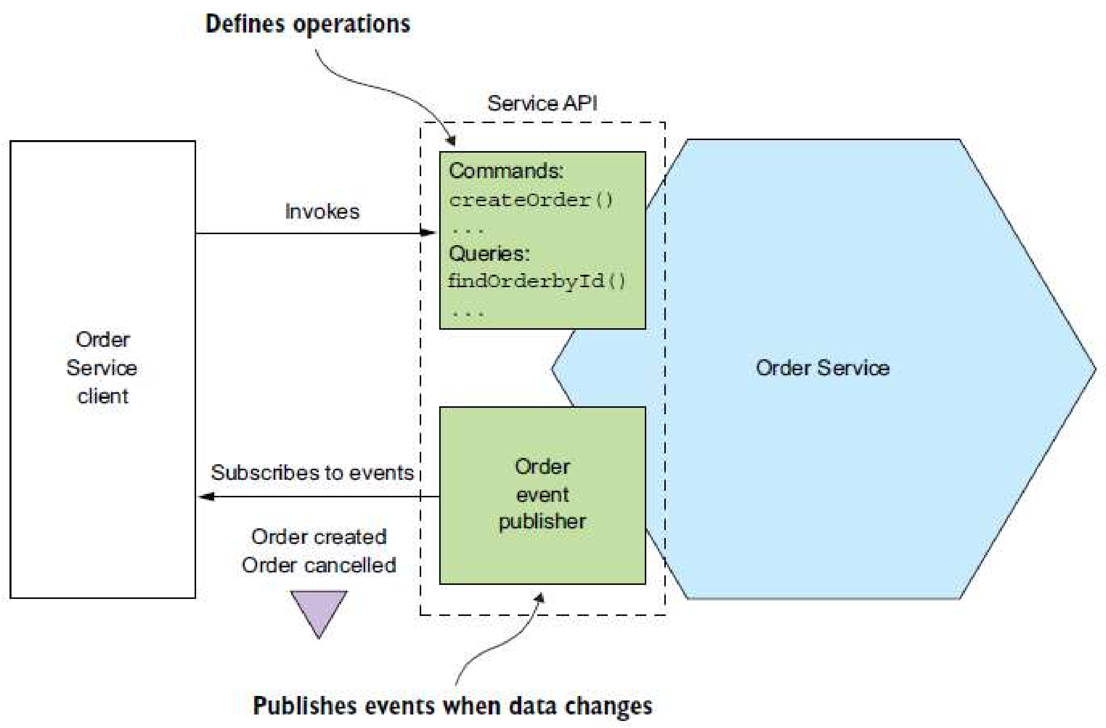
innovate achieve lead

BITS Pilani
Pilani Campus

Important concepts

What is a Service?

- The first and most important aspect of the architecture is, the definition of the 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.

Size of a Service

- size isn’t a useful metric.
- A much better goal is to define a well-designed service
- Build a set of small, loosely coupled services.

innovate achieve lead

BITS Pilani

Pilani Campus

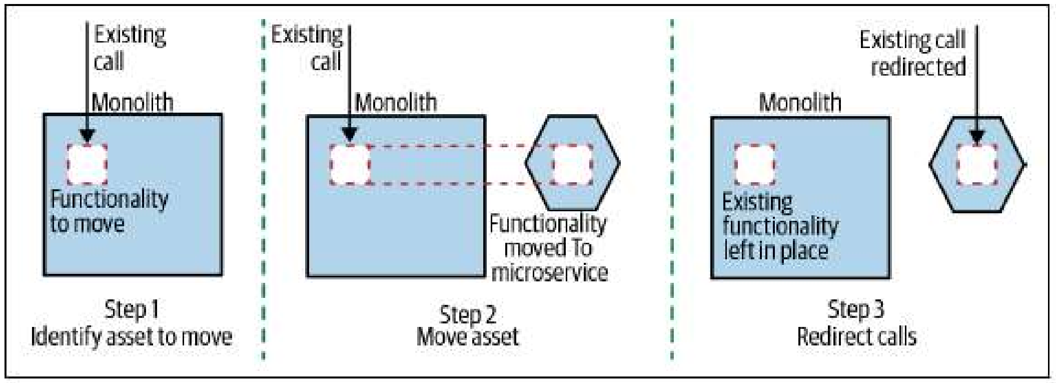
Monolith to Microservices

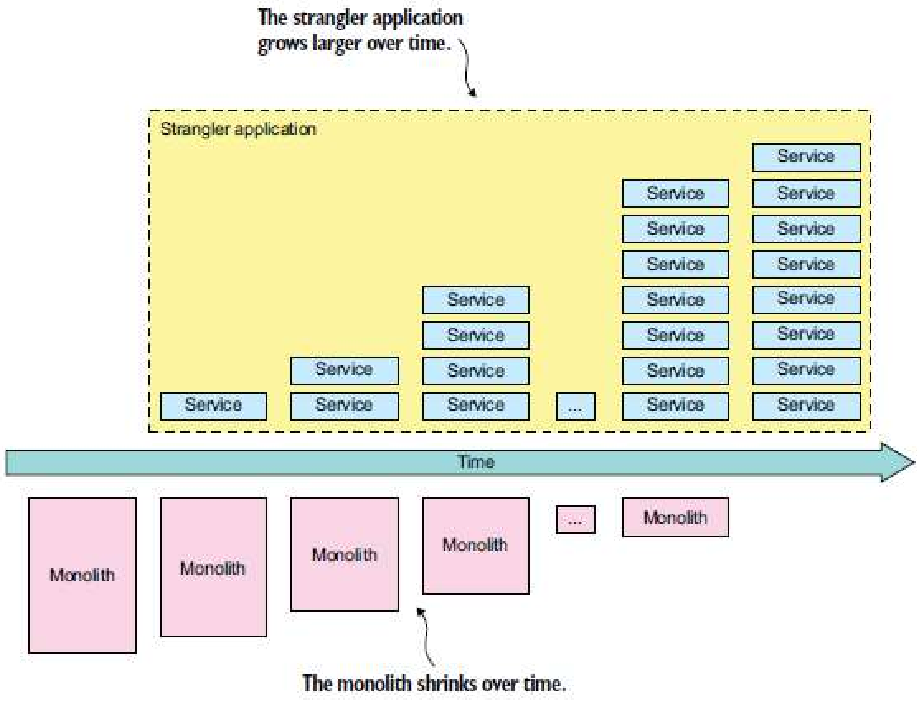
Rebuild From Scratch

- One of the biggest challenges for us was having a good understanding of the legacy system.
- Can not use the system until complete
- Longer duration required

Strangler Pattern

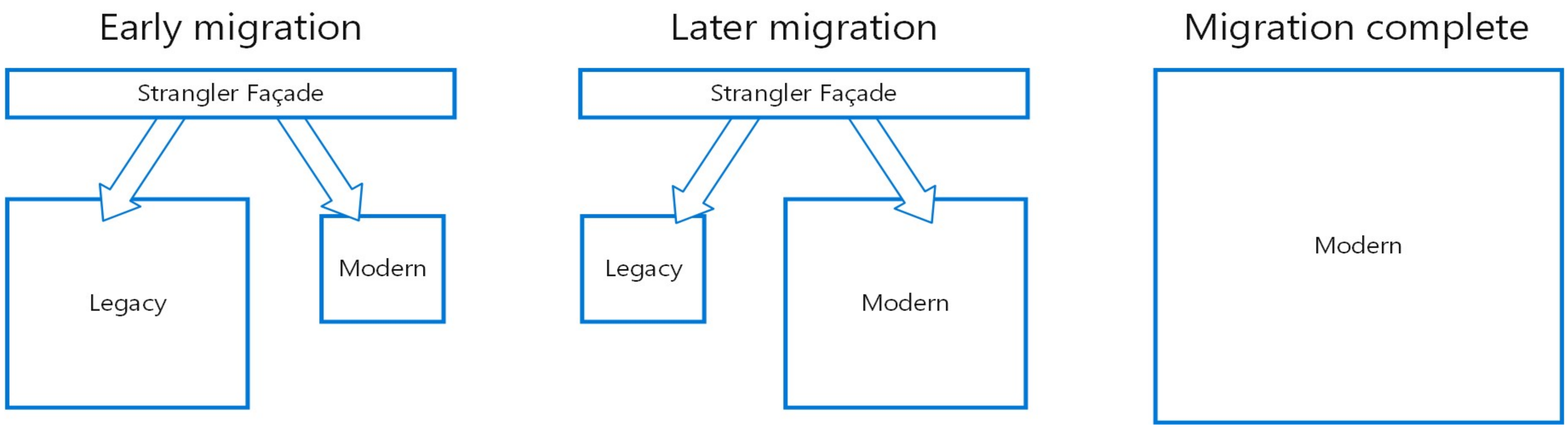
- The Strangler Pattern is a popular design pattern to incrementally transform your monolithic application into microservices by replacing a particular functionality with a new service.
- Any new feature to be added is done as part of the new service




The strangler application grows larger over time.
Strangler application
Service
...
Time
Monolith
The monolith shrinks over time.

Strangler Pattern

Steps involved in transition



Strangler Pattern: Issues

- What component to start with?
- How to handle services and data stores that are potentially used by both new and legacy systems?
- Migration

When not to use Strangler Pattern?

- When requests to the back-end system cannot be intercepted.
- For smaller systems where the complexity of wholesale replacement is low.

innovate achieve lead

BITS Pilani
Pilani Campus

Decomposition based patterns

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.

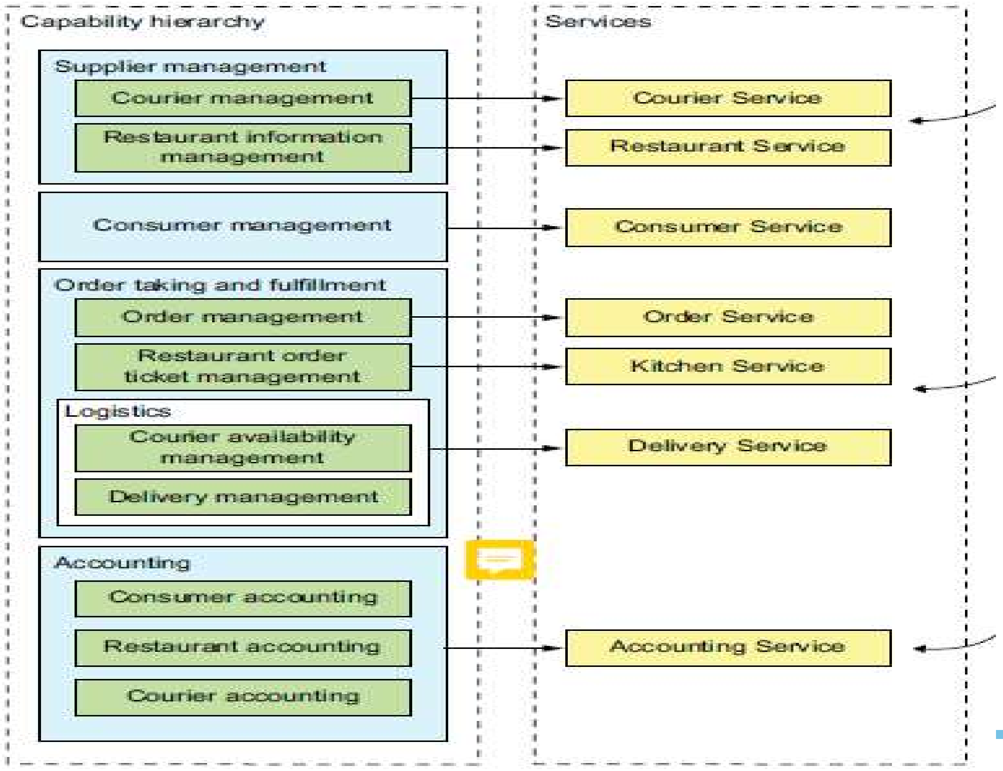
Identifying Business Capabilities

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



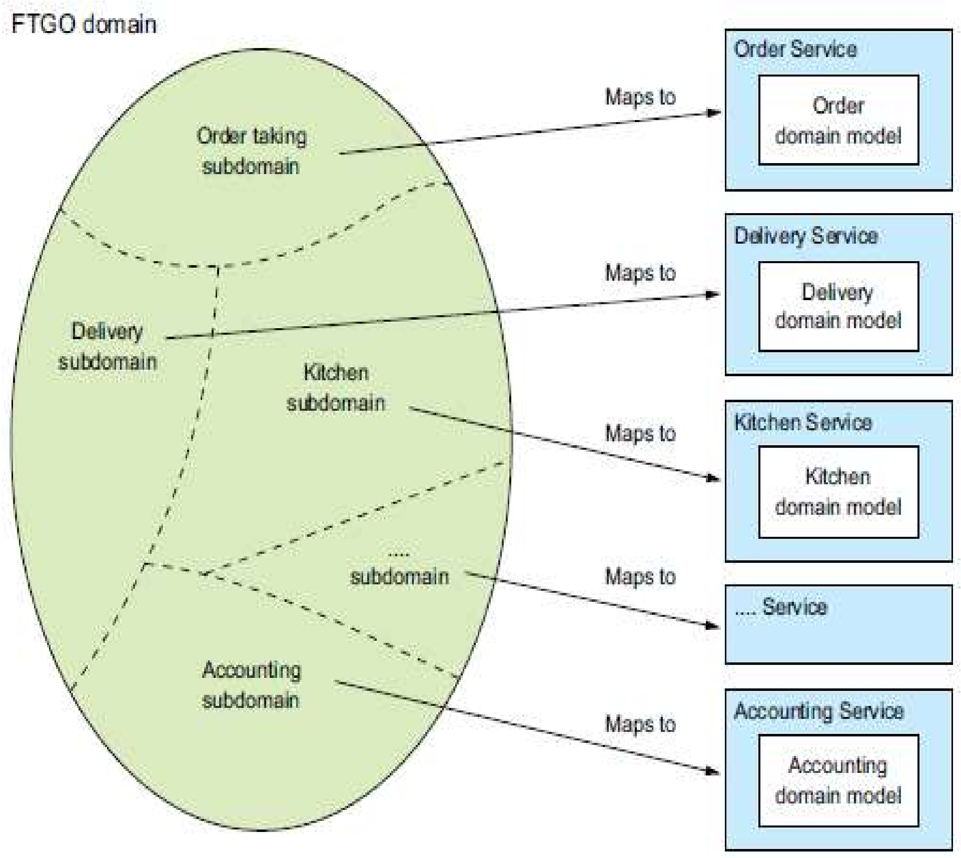
Decompose by sub-domain pattern

- 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

References

- Book: Microservices Patterns by Chris Richardson
- Book: Building Microservices by Sam Newman
- Book: Monolith to Microservices by Sam Newman
- Link: https://docs.microsoft.com/en-us/azure/architecture/microservices/design/
- Link: https://docs.microsoft.com/en-us/azure/architecture/microservices/design/patterns