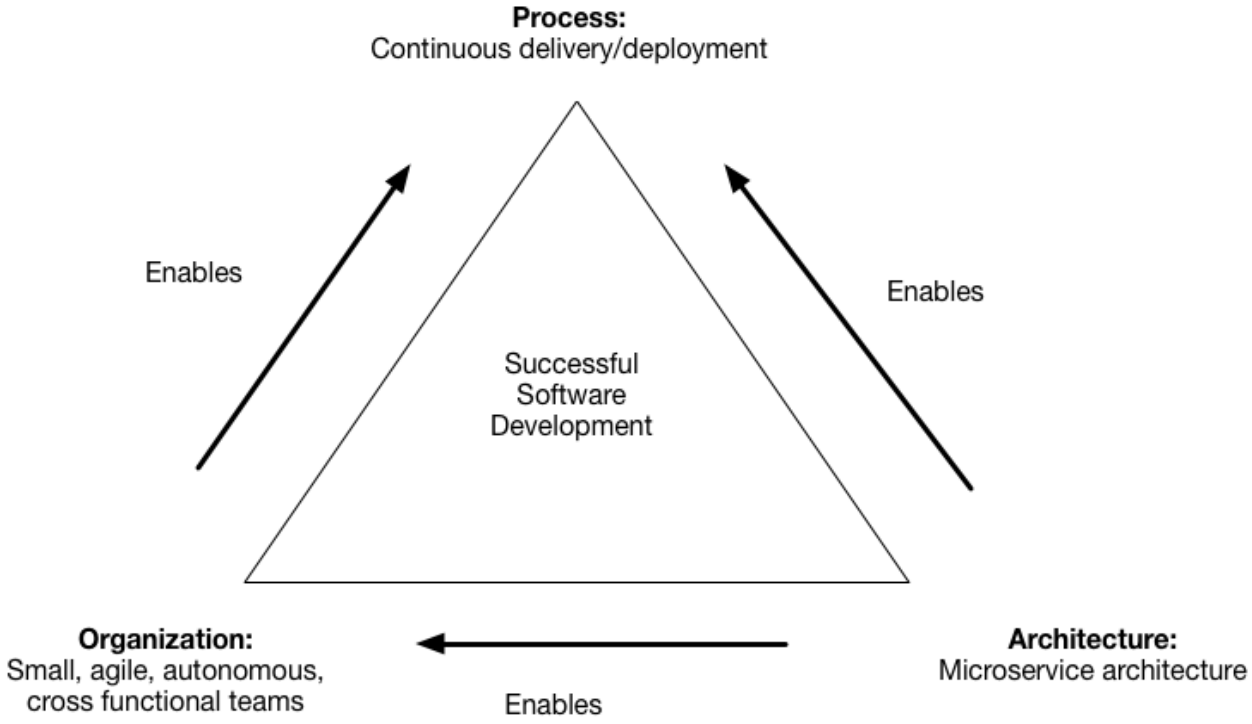
Pattern: Decompose by subdomain

version 2019.30

상황

당신은 크고 복잡한 애플리케이션을 개발하는 중이며 마이크로서비스 아키텍처를 적용하려고 합니다. 마이크로서비스 아키텍처는 애플리케이션을 느슨하게 결합한 서비스들의 집합으로 구조화합니다. 마이크로서비스 아키텍처의 목표는 CI/CD를 가능하게 함으로써 소프트웨어 개발을 가속화 하는 것입니다.



마이크로서비스 아키텍처는 다음 두 가지 방법으로 이를 수행합니다.

1. 테스트를 간소화하고 컴포넌트들을 독립적으로 배포할 수 있게 합니다.
2. 엔지니어링 조직을 소규모(6~10 명)으로 구성하고 자율 팀을 구성하며 각 팀은 하나 또는 그 이상의 서비스를 담당합니다.

이러한 혜택은 자연스레 얻어지는 것이 아닙니다. 대신에 응용 프로그램을 서비스 단위로 세분화하여 기능을 수행해야만 이러한 혜택을 취할 수 있습니다.

하나의 서비스는 작은 팀에서 개발하고 쉽게 테스트를 할 수 있을만큼 작아야합니다. OOD(Object-Oriented Design)의 유용한 지침 중 SRP(Single Responsibility Principle)이 있습니다. SRP는 클래스의 책임을 변경해야 할 이유로 정의하고 클래스에는 변경해야 할 한 가지 이유만 있어야 한다고 명시합니다. SRP를 서비스 디자인에 적용하고 일관성 있는 서비스를 설계하고 강력하게 관련된 작은 기능 세트를 구현하는 것이 이치에 맞습니다.

또한 애플리케이션은 보통 새롭거나 수정된 요구사항이 단일 서비스에만 영향을 미치도록 분해됩니다. 이는 여러 서비스에 영향을 미치는 변경 사항이 여러 팀간에 조정을 필요로 함으로써 개발 속도를 느려지게 만들기 때문입니다. OOD의 또 다른 유용한 지침 중 하나는 CCP(Common Closure Principle)입니다. 동일한 이유로 변경되는 클래스는 동일한 패키지에 있어야 합니다. 예를 들어, 두 클래스는 동일한 비즈니스 규칙의 다른 측면을 구현합니다. The goal is that when that business rule changes developers, only need to change code in a small number - ideally only one - of packages. 이러한 종류의 사고는 서비스를 설계할 때 각 변경 사항이 하나의 서비스에만 영향을 미치도록 보장하기 때문에 의미가 있습니다.

문제

애플리케이션을 어떻게 서비스들로 분해할 것인가?

Forces

- 아키텍처는 안정적이어야 합니다.
- 서비스는 응집력이 있어야 합니다. 서비스는 강하게 관련된 기능의 작은 집합을 구현해야 합니다.
- 서비스는 공통 변경 원칙(Common Closure Principle)을 준수해야 합니다. 변경 사항은 함께 포장되어야 하며 각 변경 사항은 하나의 서비스에만 영향을 줘야 합니다.
- 서비스는 느슨하게 결합하여야 합니다. 각 서비스는 구현을 캡슐화하는 API입니다. 클라이언트에 영향을 주지 않고 구현을 변경할 수 있습니다.
- 서비스는 테스트 가능해야 합니다.
- 각 서비스는 "two pizza"팀, 즉 6-10명으로 구성된 팀에 의해 개발 될 만큼 충분히 작아야 합니다.
- 하나 이상의 서비스를 소유하고 있는 각 팀은 자율적이어야 합니다. 팀은 다른 팀과 최소한의 협업을 통해 서비스를 개발하고 배포할 수 있어야 합니다.

해결책

DDD(Domain-Driven Design)에서의 하위 도메인에 일치하는 서비스를 정의합니다. DDD는 응용 프로그램의 비즈니스 관점의 problem space를 도메인이라고합니다. 하나의 도메인의 여러 하위 도메인으로 구성됩니다. 각 하위 도메인은 비즈니스의 각각 다른 부분을 담당합니다.

하위 도메인을 다음과 같이 분류할 수 있습니다.

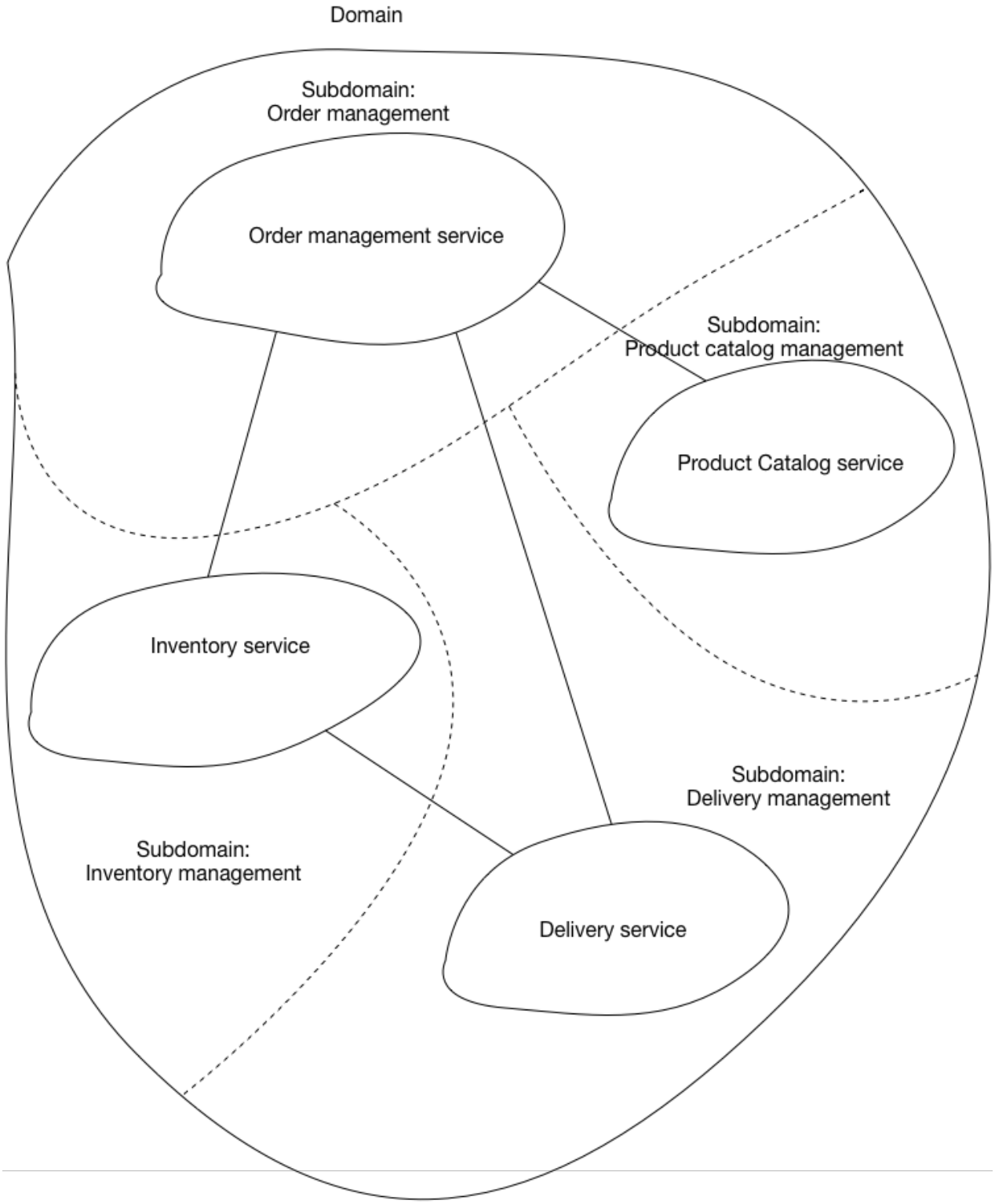
- Core - 비즈니스의 핵심 차별화 요소이자 애플리케이션의 가장 중요한 부분입니다.
- Supporting - 비즈니스가 하는 일과 관련이 있지만 차별화되지는 않습니다. 이들은 사내 또는 외주 처리 될 수 있습니다.
- Generic - 비즈니스와 관련이 없으며 self software를 사용하여 이상적으로 구현됩니다.

예제

온라인 쇼핑몰은 다음과 같은 하위 도메인을 가집니다.

- Product catalog
- Inventory management
- Order management
- Delivery management
- ...

해당 마이크로서비스 아키텍처는 각 하위 도메인에 해당하는 서비스를 가져야합니다.



예상되는 상황

이 패턴은 다음과 같은 이점이 있습니다.

- 하위 도메인이 비교적 안정적이기 때문에 아키텍처가 안정적입니다.
- 개발 팀은 기술적 기능보다는 비즈니스 가치를 제공하기 위해 cross-functional하고 자율적으로 구성됩니다.
- 서비스는 응집력이 있고 느슨하게 결합됩니다.

이슈

아래는 해결해야할 이슈입니다.

- **어떻게 하위 도메인을 식별할 것인가?** 하위 도메인과 서비스를 식별하려면 비즈니스에 대한 이해가 필요합니다. 비즈니스 기능 처럼 하위 도메인은 비즈니스 및 조직 구조를 분석하고 다양한 전문 분야를 구별함으로써 식별됩니다. 하위 도메인은 반복 프로세스를 사용하여 가장 잘 식별됩니다. 하위 도메인을 식별하기위한 좋은 출발점은 다음과 같습니다.
- organization structure - 조직 내의 여러 그룹이 하위 도메인에 해당 할 수 있습니다.
- high-level domain model - 하위 도메인에는 종종 핵심 도메인 객체를 가집니다.

관련 패턴

- Decompose by business capability 패턴을 대안으로 사용할 수 있습니다.

> **참고 문헌**
>
> 본 문서는 Microservices.io에서 작성한 A pattern language for microservices를 번역한 문서입니다.
>
> **Reference**
>
> This document is a translation of A pattern language for microservices written by Microservices.io.