마이크로서비스로 모놀리식 유형 분해

AWS 권장 가이드

AWS 권장 가이드: 마이크로서비스로 모놀리식 유형 분해

Table of Contents

모놀리스를 마이크로서비스로 분해

태비 워드와 드미트리 굴린, Amazon Web Services (AWS)

2023년 4월(문서 기록)

Amazon Web Services (AWS) 클라우드로 마이그레이션하면 기술 및 비즈니스 민첩성, 새로운 수익 기회, 비용 절감 등 많은 이점이 있습니다. 이러한 이점을 최대한 활용하려면 모놀리식 애플리케이션을 마이크로서비스로 리팩토링하여 조직의 소프트웨어를 지속적으로 현대화해야 합니다. 이 프로세스는 세 가지 주요 단계로 구성됩니다.

- 모놀리스를 마이크로서비스로 분해 – 이 가이드에서 제공하는 분해 패턴을 사용하여 모놀리식 애플리케이션을 마이크로서비스로 분류합니다.
- 마이크로서비스 통합 – AWS 서버리스 서비스를 사용하여 새로 만든 마이크로서비스를 마이크로서비스 아키텍처에 통합합니다.
- 마이크로서비스를 위한 데이터 지속성 활성화 – 데이터 저장소를 분산하여 마이크로 서비스 간의 다중 사용 지속성을 촉진합니다.

현대화에는 일반적으로 두 가지 유형의 프로젝트가 포함됩니다.

- 브라운필드 프로젝트에는 기존 또는 레거시 시스템의 맥락에서 새로운 소프트웨어 시스템을 개발하고 배포하는 작업이 포함됩니다.
- 그린필드 프로젝트에는 레거시 코드를 사용하지 않고 완전히 새로운 환경을 위한 시스템을 처음부터 만드는 작업이 포함됩니다.

브라운필드 프로젝트의 경우 애플리케이션 현대화 여정의 첫 단계 중 하나는 포트폴리오의 모놀리스를 마이크로서비스로 분해하는 것입니다.

대부분의 애플리케이션은 특정 비즈니스 사용 사례에 맞게 설계된 모놀리스로 시작됩니다. 모놀리스의 아키텍처에서 모듈식 설계를 적용하지 않는 경우에도 잘 정립된 도메인 지식의 범위 내에서 책임이 명확하게 정의되지 않은 애플리케이션에는 모놀리스가 여전히 유효한 선택이 될 수 있습니다. 단일 배치 단위라는 모놀리스의 중심적 특성은 또한 긴밀한 결합이나 내부 구조 부족과 같은 설계 결함을 완화하는 데 도움이 될 수 있습니다.

일부 사용 사례에서는 모놀리스가 유효한 옵션이 될 수 있지만 일반적으로 최신 애플리케이션에는 적합하지 않습니다. 모놀리스의 내부 구조가 잘못 정의되면 코드를 유지 관리하기가 어려워질 수 있으며,

이로 인해 신규 개발자의 학습 시간이 길어지고 추가 지원 비용이 발생할 수 있습니다. 결합성이 높고 응집력이 낮으면 새 기능을 추가하는 데 걸리는 시간이 크게 늘어날 수 있으며 트래픽 패턴에 따라 개별 구성 요소를 확장하지 못할 수도 있습니다. 또한 모놀리스는 하나의 대규모 릴리스를 위해 여러 팀이 협력해야 하므로 협업과 지식 이전 부담이 커집니다. 마지막으로, 비즈니스나 사용자 기반이 커지면 새로운 기능을 추가하거나 새로운 사용자 경험을 구축하는 것이 어려워진다는 것을 알 수 있습니다.

이를 방지하기 위해 분해 패턴을 사용하여 모놀리식 애플리케이션을 세분화하고 여러 마이크로서비스로 변환한 다음 마이크로서비스 아키텍처로 마이그레이션할 수 있습니다. 마이크로서비스 아키텍처는 애플리케이션을 일련의 느슨하게 결합된 서비스로 구성합니다. 마이크로서비스는 지속적인 배포 및 지속적인 배포 (CI/CD) 프로세스를 지원하여 소프트웨어 개발을 가속화하도록 설계되었습니다.

분해 프로세스를 시작하기 전에 분해할 모놀리스를 평가해야 합니다. 신뢰성이나 성능에 문제가 있는 모놀리스를 포함하거나 긴밀하게 결합된 아키텍처에 여러 구성 요소를 포함해야 합니다. 또한 모놀리스의 비즈니스 사용 사례, 기술, 다른 애플리케이션과의 상호 종속성을 완전히 이해하는 것이 좋습니다.

이 가이드는 애플리케이션 소유자, 비즈니스 소유자, 설계자, 기술 책임자 및 프로젝트 매니저를 위한 것입니다. 모놀리스를 분해하는 데 사용되는 다음 6가지 클라우드 네이티브 패턴을 설명하고 각 패턴의 장단점을 설명합니다.

- 비즈니스 역량별 분해
- 하위 도메인별 분해
- 트랜잭션별 분해
- 팀별 서비스 패턴
- Strangler Fig 패턴
- 추상화 패턴별 브랜치

이 가이드는에서 권장하는 애플리케이션 현대화 접근 방식을 다루는 콘텐츠 시리즈의 일부입니다 AWS. 이 시리즈에는 다음 내용도 포함됩니다.

- AWS 클라우드에서 애플리케이션을 현대화하기 위한 전략
- AWS 클라우드에서 애플리케이션을 현대화하기 위한 단계별 접근 방식
- AWS 클라우드의 애플리케이션에 대한 현대화 준비 상태 평가
- AWS 서버리스 서비스를 사용하여 마이크로서비스 통합

목표 비즈니스 성과

모놀리스를 마이크로서비스로 분해한 후에는 다음과 같은 결과를 기대할 수 있습니다.

- 모놀리식 애플리케이션을 마이크로서비스 아키텍처로 효율적으로 전환합니다.
- 높은 확장성, 향상된 복원력, 지속적 제공, 장애 격리와 같은 핵심 활동을 중단하지 않으면서 변동하는 비즈니스 수요를 신속하게 조정합니다.
- 각 마이크로서비스를 개별적으로 테스트하고 배포할 수 있으므로 혁신이 더 빨라집니다.

모놀리스 분해 패턴

애플리케이션 포트폴리오에서 모놀리스를 분해하기로 결정한 후에는 적절한 분해 패턴을 선택하여 조직에 도입해야 합니다.

> **Note**
>
> 모놀리스를 분해하는 패턴은 여러가지가 있습니다. 예를 들어 비즈니스 역량별로 모놀리스를 분해한 다음 하위 도메인 패턴을 사용하여 더욱 세분화할 수 있습니다.

주제

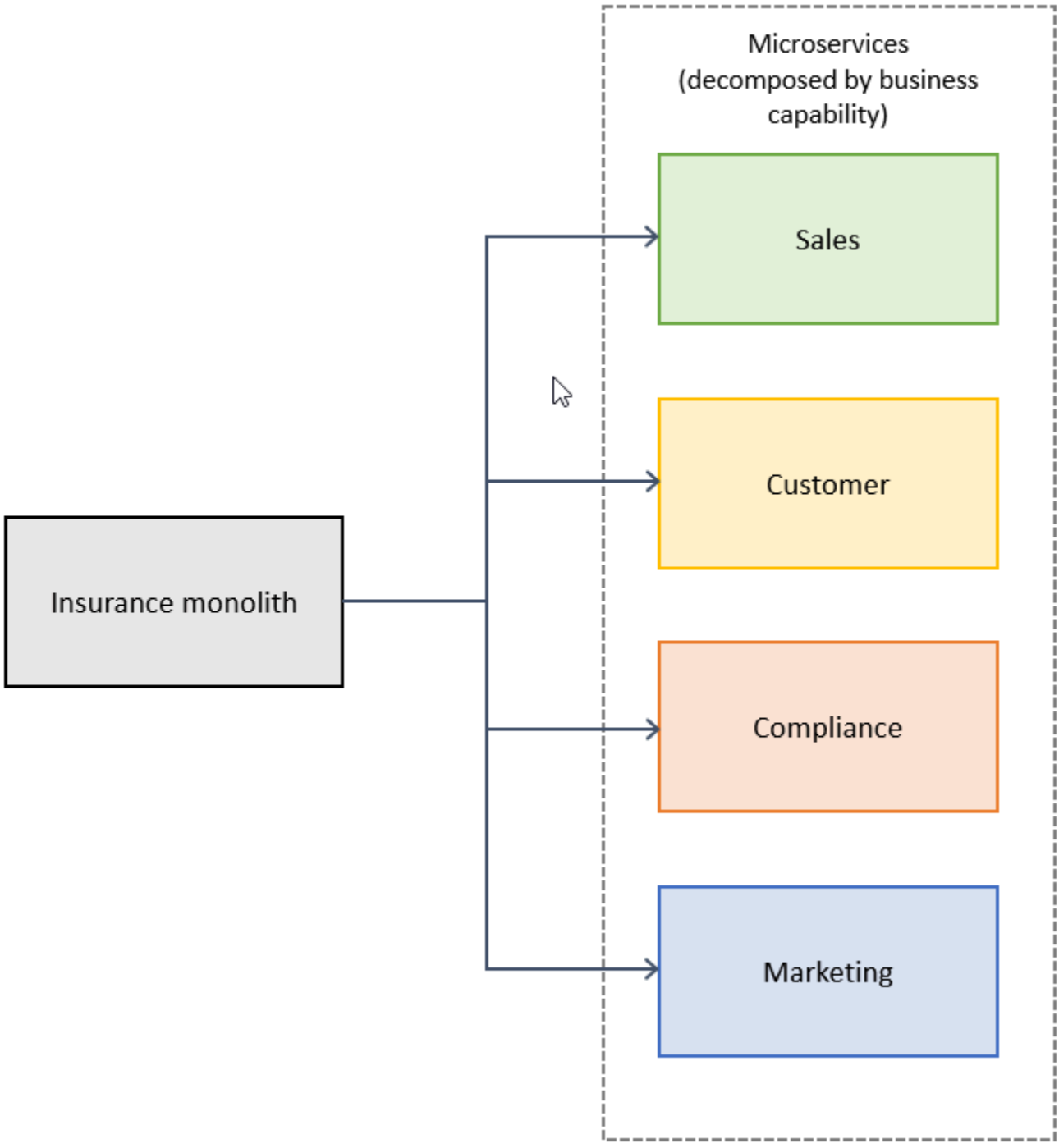
비즈니스 역량별 분해

조직의 비즈니스 프로세스나 역량을 사용하여 모놀리스를 분해할 수 있습니다. 비즈니스 역량은 기업이 가치를 창출하기 위해 하는 일(예: 영업, 고객 서비스 또는 마케팅)입니다. 일반적으로 한 조직에는 여러가지 비즈니스 역량이 있으며 이는 부문 또는 산업별로 달라집니다. 팀이 조직의 사업부를 충분히 파악하고 있고 각 사업부에 주제별 전문가(SME)가 있는 경우 이 패턴을 사용하세요. 다음 표는 이 패턴 사용의 장단점을 설명합니다.

장점	단점
• 비즈니스 역량이 비교적 안정적인 경우 안정적인 마이크로서비스 아키텍처를 생성합니다. • 개발팀은 여러 부서를 아우르며 기술적 기능 대신 비즈니스 가치를 제공하는 것을 중심으로 조직됩니다.	• 애플리케이션 디자인은 비즈니스 모델과 밀접하게 결합되어 있습니다. • 비즈니스 역량과 서비스를 파악하기 어려울 수 있으므로 전체 비즈니스에 대한 심층적인 이해가 필요합니다.

장점	단점
• 서비스는 느슨하게 결합되어 있습니다.	

다음 다이어그램은 보험 모놀리스를 비즈니스 역량에 따라 4개의 마이크로서비스로 분류합니다.

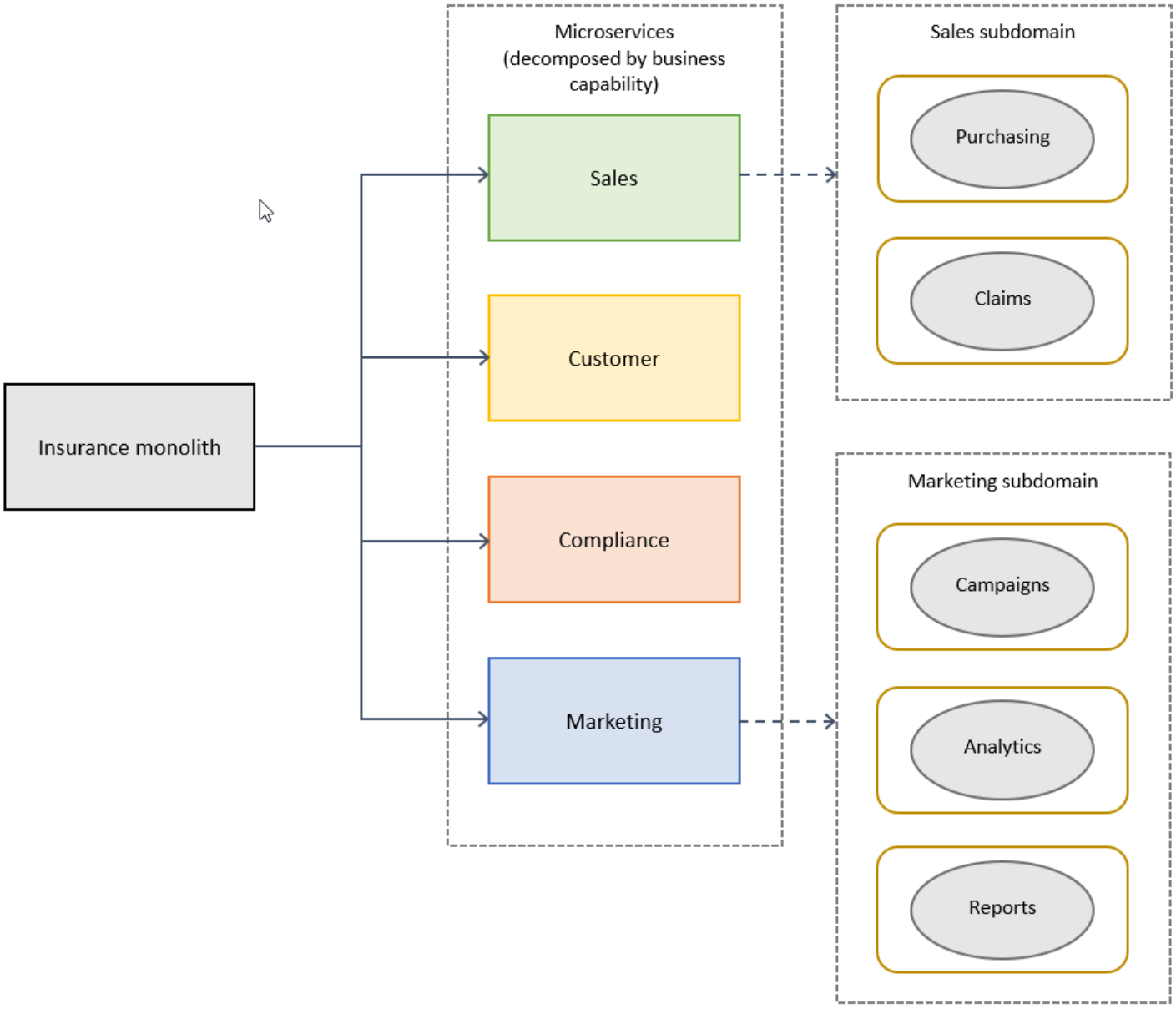


하위 도메인별 분해

이 패턴은 도메인 기반 설계(DDD) 하위 도메인을 사용하여 모놀리스를 분해합니다. 이 접근 방식은 조직의 도메인 모델을 핵심(비즈니스의 주요 차별화 요소), 지원(비즈니스와 관련이 있지만 차별화 요소는 아님) 또는 일반(모든 비즈니스에 공통적으로 적용됨)으로 분류되는 별도의 하위 도메인으로 분류합니다. 이 패턴은 하위 도메인 관련 모듈 간의 경계가 잘 정의된 기존의 모놀리식 시스템에 적합합니다. 즉, 기존 코드를 크게 다시 작성하지 않고도 기존 모듈을 마이크로서비스로 다시 패키징하여 모놀리스를 분해할 수 있습니다. 각 하위 도메인에는 모델이 있으며 해당 모델의 범위를 제한된 컨텍스트라고 합니다. 마이크로서비스는 이러한 제한된 컨텍스트를 중심으로 개발됩니다. 다음 표는 이 패턴 사용의 장단점을 설명합니다.

장점	단점
• 느슨하게 결합된 아키텍처는 확장성, 복원력, 유지 관리성, 확장성, 위치 투명성, 프로토콜 독립성 및 시간 독립성을 제공합니다. • 시스템의 확장성과 예측성이 향상됩니다.	• 마이크로서비스를 너무 많이 생성하여 서비스 검색 및 통합을 어렵게 만들 수 있습니다. • 전체 비즈니스에 대한 심층적인 이해가 필요하기 때문에 비즈니스 하위 도메인을 파악하기 어렵습니다.

다음 그림은 보험 모놀리스가 비즈니스 역량에 의해 분해된 후 다시 하위 도메인으로 분해되는 방법을 보여줍니다.



그림은 영업 및 마케팅 서비스가 더 작은 마이크로서비스로 분류되어 있음을 보여줍니다. 구매 및 청구 모델은 영업의 중요한 비즈니스 차별화 요소이며 두 개의 개별 마이크로서비스로 구분됩니다. 마케팅은 캠페인, 분석, 보고서와 같은 지원 비즈니스 기능을 사용하여 세분화됩니다.

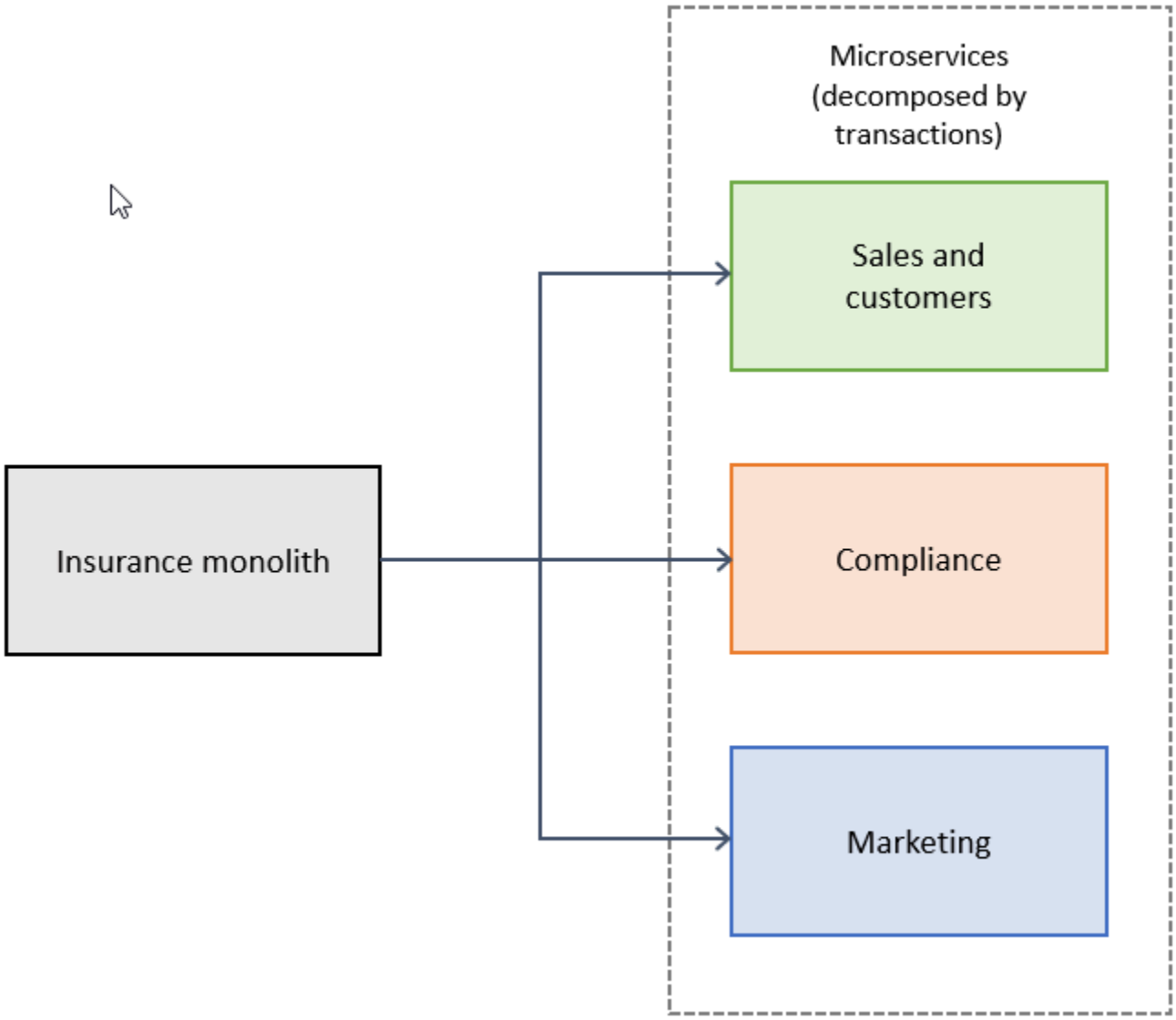
트랜잭션별 분해

분산 시스템에서 애플리케이션은 일반적으로 하나의 비즈니스 트랜잭션을 완료하기 위해 여러 마이크로서비스를 호출해야 합니다. 지연 문제나 2단계 커밋 문제를 방지하려면 트랜잭션을 기반으로 마이크로서비스를 그룹화할 수 있습니다. 이 패턴은 사용자가 응답 시간을 중요하게 생각하고, 모듈을 패키징

한 후 서로 다른 모듈이 모놀리스를 생성하지 않는 경우에 적합합니다. 다음 표는 이 패턴 사용의 장단점을 설명합니다.

장점	단점
• 응답 시간이 더 빠릅니다. • 데이터 일관성에 대해 걱정할 필요가 없습니다. • 가용성 개선.	• 여러 모듈을 함께 패키징할 수 있으며, 하나의 모놀리스를 생성할 수 있습니다. • 별도의 마이크로서비스가 아닌 단일 마이크로서비스에 여러 기능이 구현되므로 비용과 복잡성이 증가합니다. • 트랜잭션 지향 마이크로서비스는 비즈니스 도메인의 수와 이들 간의 종속성이 높을 경우 성장할 수 있습니다. • 동일한 비즈니스 도메인에 대해 일관되지 않은 버전이 동시에 배포될 수 있습니다.

다음 그림에서 보험 모놀리스는 트랜잭션을 기반으로 여러 마이크로서비스로 분류됩니다.



보험 시스템에서는 일반적으로 청구 요청이 제출된 후 고객에게 태그가 지정됩니다. 즉, 고객 마이크로서비스 없이는 청구 서비스가 존재할 수 없습니다. 영업과 고객은 하나의 마이크로서비스 패키지로 패키징되며, 비즈니스 트랜잭션에는 두 마이크로서비스 패키지와의 조정이 필요합니다.

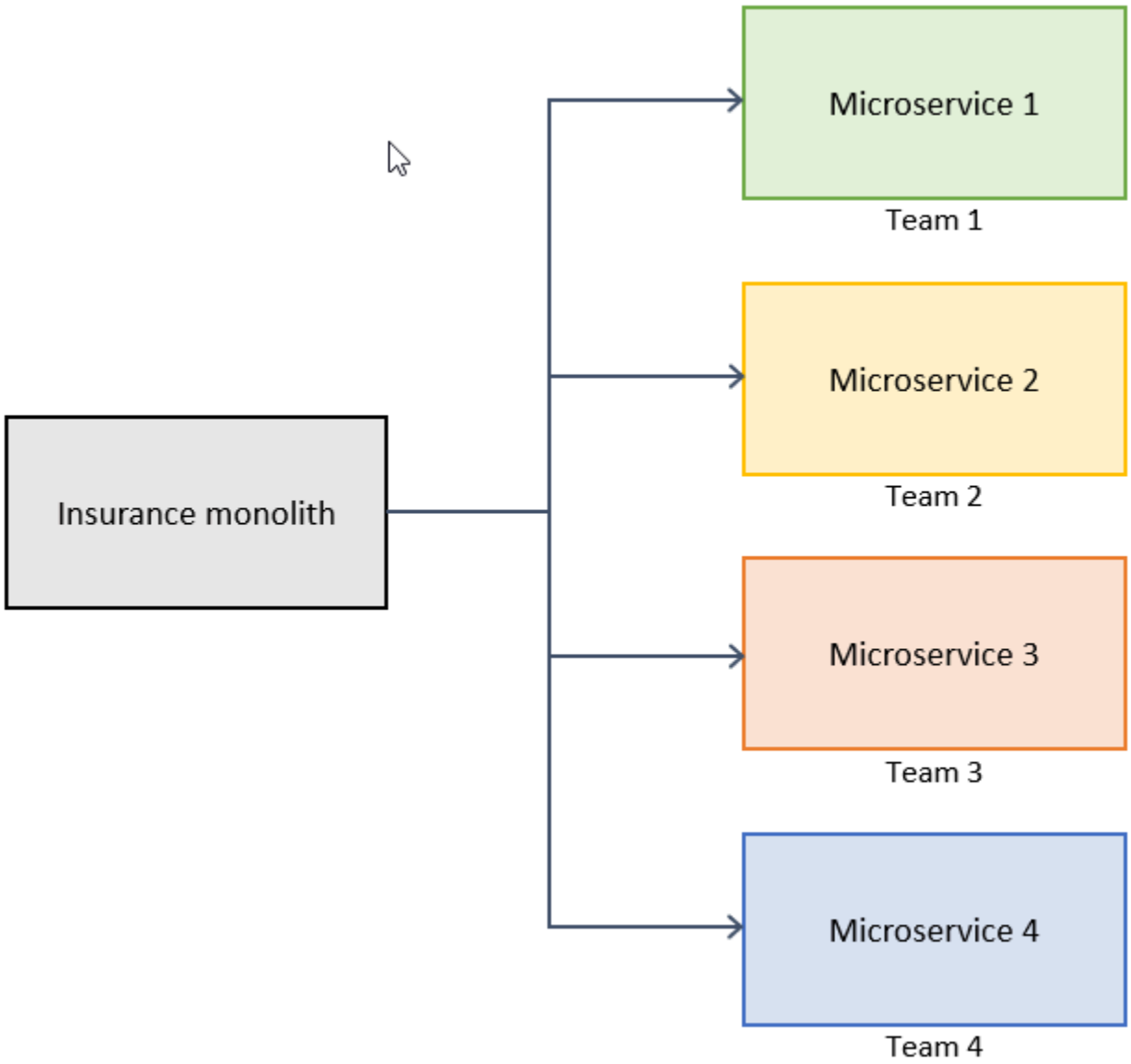
팀별 서비스 패턴

팀별 서비스 패턴은 비즈니스 역량이나 서비스별로 모놀리스를 분해하는 대신 개별 팀이 관리하는 마이크로서비스로 세분화합니다. 각 팀은 비즈니스 역량을 책임지고 해당 역량의 코드베이스를 소유합니다. 팀은 독립적으로 서비스를 개발, 테스트, 배포 또는 확장하며 주로 API를 협상하기 위해 다른 팀과 상호 작용합니다. 각 마이크로서비스를 단일 팀에 할당하는 것이 좋습니다. 그러나 팀 규모가 충분

히 크면 여러 하위 팀이 동일한 팀 구조 내에서 개별 마이크로서비스를 소유할 수 있습니다. 다음 표는 이 패턴 사용의 장단점을 설명합니다.

장점	단점
• 팀은 최소한의 조정만 필요로 하며 독립적으로 행동합니다. • 코드 베이스와 마이크로서비스를 여러 팀에서 공유하지 않습니다. • 팀은 제품 기능을 빠르게 혁신하고 반복할 수 있습니다. • 팀마다 다양한 기술, 프레임워크 또는 프로그래밍 언어를 사용할 수 있습니다. 중요: 이러한 정보는 잘 정의되고 안정적인 공개 API 뒤에 숨겨야 합니다.	• 최종 사용자 기능이나 비즈니스 역량에 맞게 팀을 조정하는 것은 어려울 수 있습니다. • 특히 팀 간에 순환적 종속성이 있는 경우, 더 크고 조율된 애플리케이션 증분을 제공하려면 추가적인 노력이 필요합니다.

다음 그림은 모놀리스를 개별 팀이 관리, 유지 관리 및 제공하는 마이크로서비스로 분할하는 방법을 보여줍니다.



Strangler fig 패턴

이 가이드에서 지금까지 설명한 설계 패턴은 미개발 프로젝트의 애플리케이션 분해에 적용됩니다. 대규모 모놀리식 애플리케이션을 포함하는 기존 프로젝트는 어떨까요? 이전 설계 패턴을 여기에 적용하기는 어려울 것입니다. 활발하게 사용되고 있는 패턴을 작은 조각으로 나누는 것은 부담이 큰 작업이기 때문입니다.

Strangler fig 패턴은 마틴 파울러(Martin Fowler)가 도입한 인기 있는 설계 패턴입니다. 마틴 파울러는 나무 위쪽 가지에 씨를 뿌리는 특정한 종류의 무화과에서 영감을 얻었죠. 기존 나무는 처음에는 새 무화과를 지탱하는 구조물이 됩니다. 그런 다음 무화과는 뿌리를 땅으로 내보내고, 점차 원래 나무를 감싸면서 자라 결국 스스로 선 새 무화과 나무만 남게 됩니다.

이 패턴은 일반적으로 특정 기능을 새 서비스로 대체하는 방식으로 모놀리식 애플리케이션을 마이크로서비스로 점진적으로 전환하는 데 사용됩니다. 목표는 기존 버전과 현대화된 새 버전이 공존하는 것입니다. 처음에는 기존 시스템이 새 시스템을 지원하며, 점차 새 시스템이 기존 시스템을 대체하게 됩니다. 이러한 지원을 통해 새 시스템을 확장하고 잠재적으로 기존 시스템을 완전히 교체할 수 있는 시간을 확보할 수 있습니다.

Strangler fig 패턴을 구현하여 모놀리식 애플리케이션에서 마이크로서비스로 전환하는 프로세스는 변환, 공존, 제거라는 세 단계로 구성됩니다.

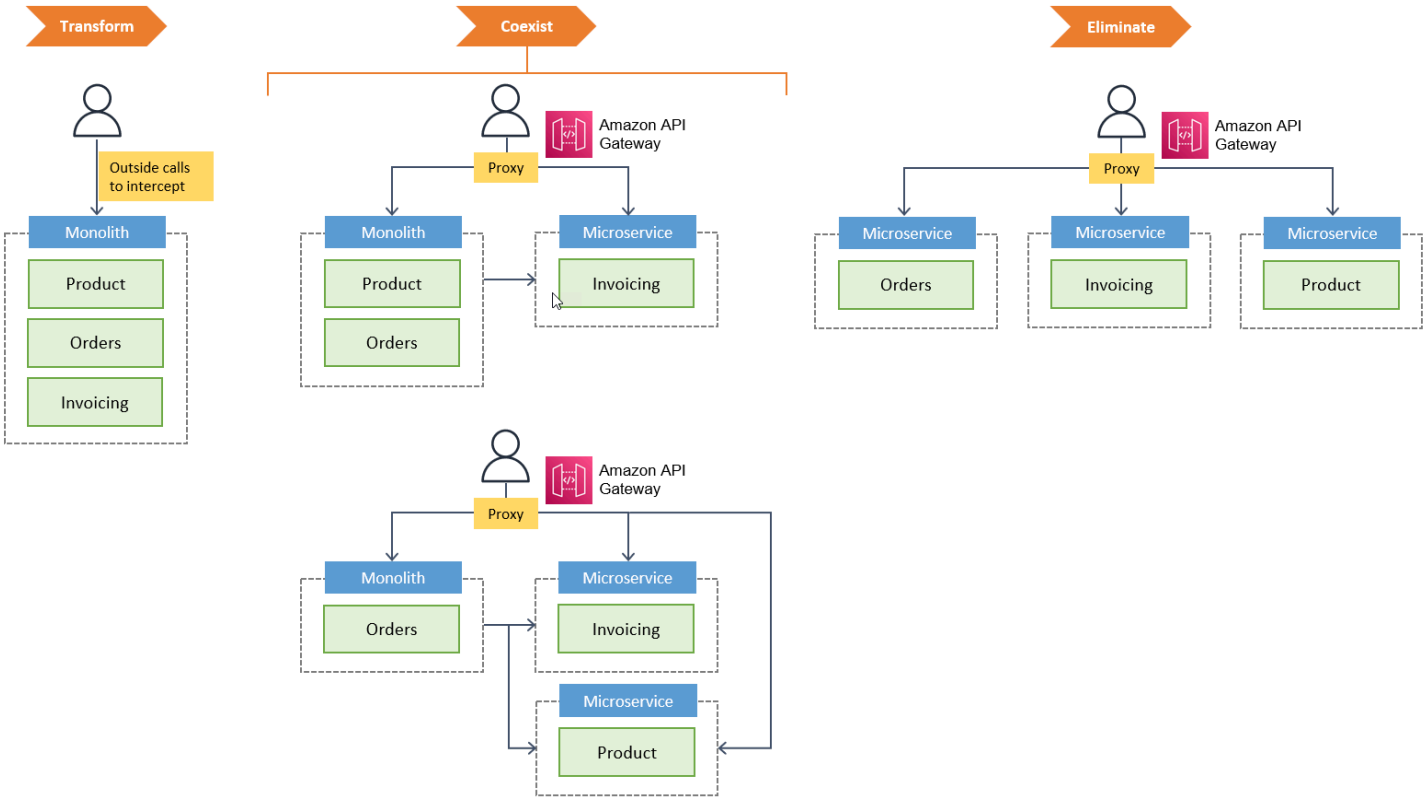
- 변환 - 기존 애플리케이션과 병렬로 포팅하거나 재작성하여 현대화된 구성 요소를 식별하고 생성합니다.
- 공존 - 모놀리스 애플리케이션을 롤백에 대비해 보관합니다. 모놀리스 경계에 HTTP 프록시(예: Amazon API Gateway)를 통합하여 외부 시스템 직접 호출을 가로채고 트래픽을 현대화된 버전으로 리디렉션합니다. 이를 통해 기능을 점진적으로 구현할 수 있습니다.
- 제거 - 트래픽이 기존 모놀리스에서 현대화된 서비스로 리디렉션되었으므로 모놀리스에서 기존 기능을 사용 중지합니다.

다음 표에는 Strangler fig 패턴 사용의 장점과 단점이 설명되어 있습니다.

장점	단점
• 서비스에서 하나 이상의 대체 서비스로 원활하게 마이그레이션할 수 있습니다. • 업데이트된 버전으로 리팩터링하는 동안 이전 서비스를 계속 사용할 수 있습니다. • 이전 서비스를 리팩터링하면서 새 서비스와 기능을 추가할 수 있습니다. • 패턴을 API 버전 관리에 사용할 수 있습니다. • 이 패턴을 업그레이드되지 않았거나 업그레이드되지 않을 솔루션의 기존 상호 작용에 사용할 수 있습니다.	• 복잡성이 낮고 크기가 작은 소규모 시스템에는 적합하지 않습니다. • 백엔드 시스템에 대한 요청을 가로채거나 라우팅할 수 없는 시스템에서는 사용할 수 없습니다. • 프록시 또는 파사드 레이어는 제대로 설계하지 않은 경우 단일 장애 지점이 되거나 성능 병목 현상이 발생할 수 있습니다. • 문제가 발생할 경우 작업을 수행하는 이전 방식으로 빠르고 안전하게 되돌리려면 리팩터링된 각 서비스에 대한 롤백 계획이 필요합니다.

다음 그림은 Strangler fig 패턴을 애플리케이션 아키텍처에 적용하여 모놀리스를 마이크로서비스로 분할하는 방법을 보여줍니다. 두 시스템 모두 병렬로 작동하지만 기능을 모놀리스 코드 베이스 외부로 이

동시키고 새로운 기능으로 개선해야 합니다. 이러한 새로운 기능을 통해 필요에 가장 적합한 방식으로 마이크로서비스를 설계할 수 있습니다. 마이크로서비스로 모두 대체될 때까지 모놀리스에서 계속 기능을 제거해야 합니다. 이 시점에서 모놀리스 애플리케이션을 제거할 수 있습니다. 여기서 주목해야 할 요점은 모놀리스와 마이크로서비스가 일정 기간 동안 함께 작동할 것이라는 점입니다.



추상화 패턴별 브랜치

Strangler fig 패턴은 모놀리스 주변에서 직접 호출을 가로챌 수 있을 때 원활하게 작동합니다. 하지만 기존 애플리케이션 스택에 더 깊이 존재하고 업스트림 종속성이 있는 구성 요소를 현대화하려면 추상화 패턴별 브랜치를 사용하는 것이 좋습니다. 이 패턴을 사용하면 기존 코드 베이스를 변경하여 중단 없이 현대화된 버전이 레거시 버전과 안전하게 공존할 수 있습니다.

추상화 패턴별 브랜치를 성공적으로 사용하려면 다음 프로세스를 따르세요.

1. 업스트림 종속성이 있는 모놀리스 구성 요소를 파악하세요.
2. 현대화할 코드와 해당 클라이언트 간의 상호 작용을 나타내는 추상화 계층을 만드세요.
3. 추상화가 적용되면 기존 클라이언트가 새 추상화를 사용하도록 변경하세요.
4. 모놀리스 외부에서 재작업된 기능을 사용하여 추상화의 새로운 구현을 생성하세요.
5. 준비가 되면 추상화를 새 구현으로 전환하세요.

6. 새 구현이 사용자에게 필요한 모든 기능을 제공하고 모놀리스를 더 이상 사용하지 않는 경우 이전 구현을 정리하세요.

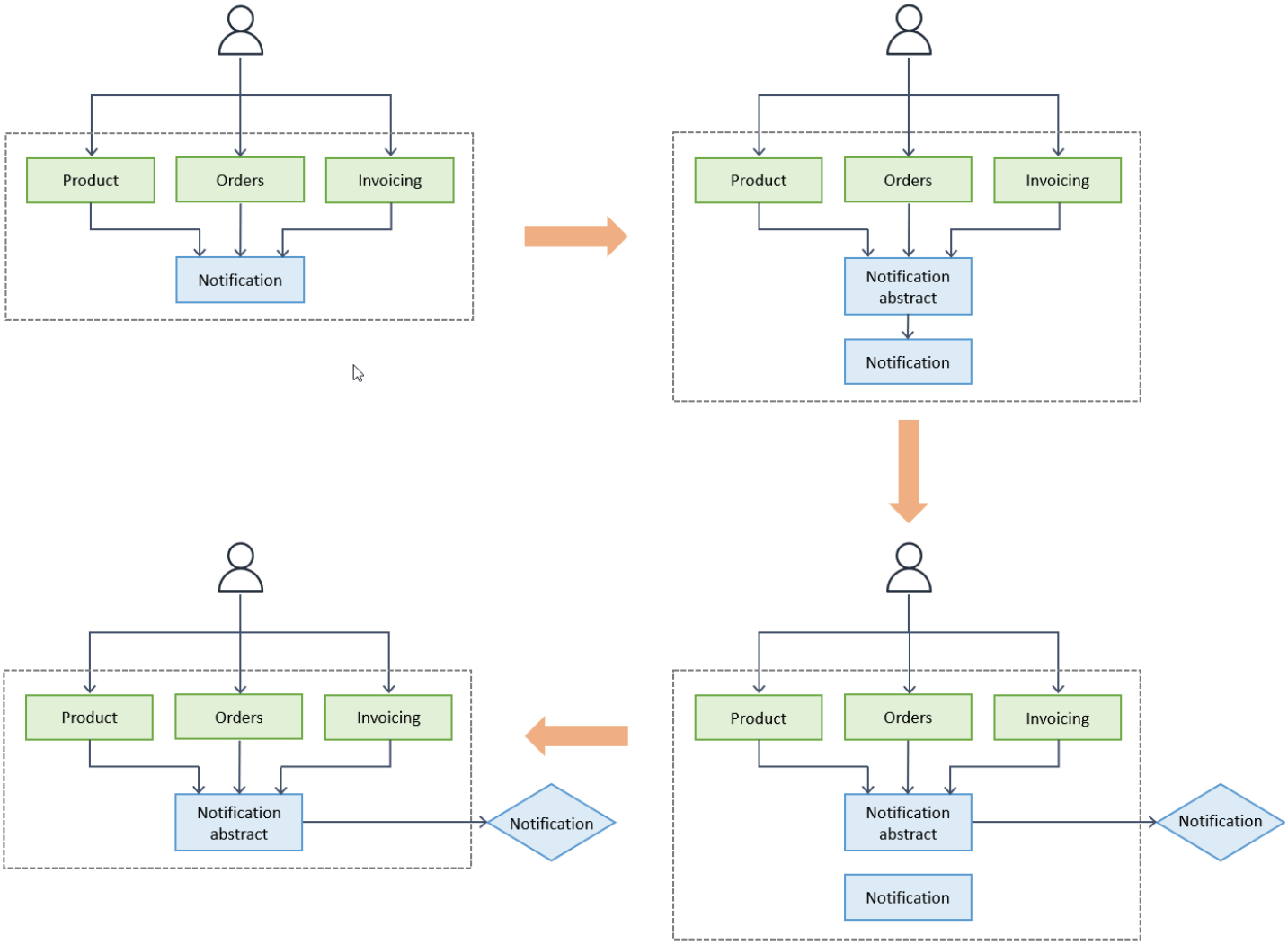
추상화 패턴에 의한 브랜치는 종종 시스템을 점진적으로 변경할 수 있게 하는 기능 토글과 혼동됩니다. 차이점은 기능 토글은 새 기능을 개발하고, 시스템이 실행 중일 때 사용자가 이 새 기능을 볼 수 없도록 하기 위한 것이라는 점입니다. 따라서 기능 토글은 배포 또는 런타임 시점에 특정 기능 또는 기능 세트를 애플리케이션에 표시할지 여부를 선택하는 데 사용됩니다. 추상화를 통한 브랜치는 하나의 개발 기법이며, 기능 토글과 결합하여 기존 구현과 새 구현 간에 전환할 수 있습니다.

다음 표는 추상화 패턴별 브랜치 사용의 장단점을 설명합니다.

장점	단점
• 문제가 발생할 경우 되돌릴 수 있는 점진적 변경이 가능합니다(이전 버전과 호환 가능). • 모놀리스 가장자리에서 모놀리스에 대한 호출을 가로챌 수 없을 때 모놀리스 내부 깊숙이 있는 기능을 추출할 수 있습니다. • 소프트웨어 시스템에 여러 구현이 공존할 수 있습니다. • 중간 검증 단계를 사용하여 새 기능과 기존 기능을 모두 호출함으로써 대체 메커니즘을 쉽게 구현할 수 있는 방법을 제공합니다. • 코드가 구조 조정 단계 내내 항상 작동하므로 지속적 전달이 지원됩니다.	• 데이터 일관성이 관련된 경우에는 적합하지 않습니다. • 기존 시스템을 변경해야 합니다. • 특히 코드베이스의 구조가 잘못된 경우 개발 프로세스에 더 많은 오버헤드가 추가될 수 있습니다. (많은 경우 추가로 노력을 기울일 만한 가치가 있다는 장점이 있으며, 구조 조정 규모가 클수록 추상화 패턴별 브랜치를 사용하는 것을 고려하는 것이 더 중요해집니다.)

다음 그림은 보험 모놀리스의 알림 구성 요소에 대한 추상화 패턴별 브랜치 패턴을 보여줍니다. 먼저 알림 기능을 위한 개요 또는 인터페이스를 생성합니다. 조금씩 기존 클라이언트가 새 추상화를 사용하도록 변경됩니다. 이 경우 알림 구성 요소와 관련된 API에 대한 호출을 코드베이스에서 검색해야 할 수 있습니다. 모놀리스 외부의 마이크로서비스로서 알림 기능의 새로운 구현을 생성하고 이를 현대화된 아키텍처에서 호스팅합니다. 모놀리스 내에서 새로 생성된 추상화 인터페이스는 브로커 역할을 하며 새 구현을 호출합니다. 완전히 테스트되고 준비가 완료될 때까지 비활성 상태로 유지될 새 구현에 알림 기능을 조금씩 이식합니다. 새 구현이 준비되면 추상화를 전환하여 사용할 수 있습니다. 문제가 발생할 경우 쉽게 이전 기능으로 다시 전환할 수 있도록 쉽게 전환할 수 있는 전환 메커니즘(예: 기능 토글)을 사용하는 것이 좋습니다. 새 구현으로 사용자에게 모든 알림 기능을 제공하기 시작하고 모놀리스를 더

이상 사용하지 않는 경우, 이전 구현을 정리하고 기존에 구현했던 모든 전환 기능 플래그를 제거할 수 있습니다.



모놀리스 분해 관련 FAQ

이 섹션에서는 모놀리스 분해와 관련하여 자주 제기되는 질문에 대한 답변을 제공합니다.

여러 패턴을 사용하여 하나의 모놀리스를 분해할 수 있습니까?

예, 여러 패턴을 사용하여 모놀리스를 분해할 수 있습니다. 가장 일반적인 방법은 비즈니스 역량별 분해 패턴을 사용하여 모놀리스를 분해한 다음 하위 도메인별 분해 패턴을 사용하여 더 세분화하는 것입니다.

모놀리스를 마이크로서비스로 분해하면 DevOps 프로세스에 어떤 영향을 미칩니까?

애플리케이션을 변경한 후 모든 것을 재배포할 필요가 없으므로 배포 프로세스에 추가되는 새로 생성된 마이크로서비스에 대한 지원과 소유권이 있어야 합니다. 이로 인해 DevOps 프로세스가 더 복잡해질 수 있습니다.

리소스

관련 가이드

- AWS 클라우드에서 애플리케이션을 현대화하기 위한 전략
- AWS 클라우드에서 애플리케이션을 현대화하기 위한 단계별 접근 방식
- AWS 클라우드의 애플리케이션에 대한 현대화 준비 상태 평가
- AWS 서버리스 서비스를 사용하여 마이크로서비스 통합

기타 리소스

- AWS Copilot, Amazon ECS, Docker 및를 사용하여 모놀리식 애플리케이션을 마이크로서비스로 나누기 AWS Fargate

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
새 패턴 추가됨	스트랭글러 피그 앤 브랜치에 대한 정보를 추상화 패턴별로 추가했습니다.	2023년 4월 6일
최초 게시	—	2021년 1월 11일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.