현대적 애플리케이션: 마이크로서비스 아키텍처

비즈니스 민첩성을 위한 현대적 아키텍처

합리적인 가격대의 고속 컴퓨팅 기술이 확산됨에 따라 기업은 그 규모에 관계없이 내부적으로 효율성을 높이고 디지털 제품을 통해 더 많은 고객과 교류할 수 있게 되었습니다. 하지만 도구의 편재성, 다양한 출시 경로 그리고 변화하는 소비자의 선호도로 인해 경쟁력을 유지하려는 기업은 그 어느 때보다 빠른 속도로 혁신해야 합니다. 농업부터 은행 및 전기 통신업에 이르기까지 업계 전반에 걸쳐, 신속한 혁신 과정에서 디지털이 핵심적인 역할을 하고 있습니다. 기존에 디지털 제품을 제작하던 방식으로는 오늘날 시장에서 성공을 거두는 데 필요한 혁신 구현 속도를 더 이상 따라잡을 수가 없습니다.

새로운 아키텍처 패턴

마이크로서비스 같은 새로운 아키텍처 패턴을 활용하는 조직은 혁신 속도를 높일 수 있습니다. 마이크로서비스 아키텍처를 통해 구축된 현대적 애플리케이션은 시간의 경과에 따라 여러 소규모 팀에 혁신 노력과 투자를 분산하는 방식으로 혁신을 실현하는데, 이를 통해 변경 사항을 테스트하여 시장에 출시하는 속도가 빨라지게 됩니다. 또한 현대적 애플리케이션은 리소스를 세부적으로 최적화할 뿐 아니라, 팀이 제품 구축 방식과 실행 방식을 모두 신속하게 확장할 수 있게 해 줍니다.

마이크로서비스 아키텍처를 정의하는 요소

전문성

각각의 서비스는 일련의 기능을 대상으로 설계되어 있으며 특정 문제를 해결하는 데 중점을 둡니다. 시간이 지남에 따라 개발자가 서비스에 더 많은 코드를 제공하고 서비스가 복잡해지면 여러 개의 소규모 서비스로 분할할 수 있습니다.

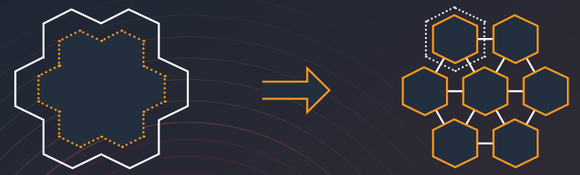
분산형

마이크로서비스 아키텍처는 함께 연동되어 가치를 제공하는 여러 구성 요소로 단일 프로세스의 애플리케이션을 분할합니다. 개별 구성 요소 간의 통신은 명확하게 정의된 약결합 API나 이벤트 및 메시징을 통해 이루어집니다.

자율성

마이크로서비스 아키텍처의 각 구성 요소 서비스는 다른 서비스의 기능에 영향을 주지 않으면서 개발, 배포, 운영하고 규모를 조정할 수 있습니다. 서비스는 다른 서비스와 코드나 구현을 공유할 필요가 없으며, 자체 포함형 블랙 박스처럼 작동합니다.

모놀리식 아키텍처의 경우 지금은 문제없이 작동할 수 있지만 비즈니스의 규모가 확장되면 문제가 발생하는 경우가 많습니다. 마이크로서비스를 활용하면 확장 및 새로운 기능의 신속한 배포 같은 일반적인 문제를 손쉽게 해결할 수 있습니다.



모놀리식 분할

모놀리식을 분할하는 일은 쉽게 엄두가 나지 않을 수 있습니다. 모놀리식을 마이크로서비스로 분할하는 프로세스를 안내하는 AWS 프로젝트를 완료하도록 초대하는 방식으로 팀을 훈련시키세요. 자습서 보기 >>

마이크로서비스로 이동

Mobvista가 자사 플랫폼의 확장성과 안정성을 개선하기 위해 마이크로서비스 아키텍처를 어떻게 도입했는지 그 방법을 알아보세요. 사례 연구 읽기 >>

마이크로서비스의 이점

마이크로서비스 아키텍처는 함께 연동되는 이산형의 모듈식 요소로 구축됩니다. 이러한 모듈성으로 인해 코드 '노출 영역'이 증가하는 문제가 따르지만 혁신 속도 개선, 독립적인 확장, 장애 영향 감소, 분산형 코드 개발 같은 핵심적인 이점도 누릴 수 있습니다.

마이크로서비스의 장점

1. 민첩성: 개별 서비스 구성 요소를 담당하는 소규모 팀은 다른 구성 요소의 제약 조건에서 벗어나므로 높은 민첩성을 바탕으로 문제나 기회에 더 빨리 대응할 수 있습니다.

2. 손쉬운 배포: 마이크로서비스는 변화의 규모가 작으므로 새로운 아이디어를 손쉽게 시도해 보고 제대로 작동하지 않으면 간단히 롤백할 수 있습니다.

3. 기술적 자유: 마이크로서비스 아키텍처를 활용하는 팀은 애플리케이션의 각 부분을 생성하는 데 가장 적합한 도구를 자유롭게 선택할 수 있습니다.

4. 확장성: 마이크로서비스를 사용하면 각 서비스에서 지원하는 애플리케이션 기능에 대한 수요를 충족하도록 서비스를 개별적으로 확장할 수 있습니다.

5. 복원력: 마이크로서비스는 장애로 인한 영향을 애플리케이션의 단일 부분으로 줄입니다. 따라서 개별 구성 요소에 장애가 발생하더라도 전체 애플리케이션의 작동이 중단되는 대신 기능만 저하됩니다.

6. 재사용 가능한 코드: 명확하게 정의된 소규모 서비스 구성 요소로 소프트웨어를 분할하면 팀이 이러한 서비스 구성 요소를 애플리케이션 내에서 여러 용도로 사용할 수 있게 됩니다. 덕분에 개발자는 코드를 처음부터 작성하거나 세부적인 구현을 다루지 않고도 API를 통해 기존 서비스를 활용하는 방식으로 새로운 기능을 생성할 수 있으므로 애플리케이션이 자체적으로 개선될 수 있습니다.

현대적 애플리케이션을 위한 마이크로서비스

마이크로서비스는 조직에서 애플리케이션 복원력을 개선하고 팀 생산성을 최적화하는 데 도움이 됩니다. 그에 따라 개발 팀은 실험과 혁신을 더 빠른 속도로 진행하는 것은 물론, 경쟁상의 이점을 제공하는 제품과 기능을 출시할 수 있습니다.

시작하는 방법

마이크로서비스 아키텍처 패턴을 도입하는 일을 '모 아니면 도'식의 극단적인 방식으로 진행할 필요는 없습니다. 서비스 중심 아키텍처를 도입하는 방식에는 크게 두 가지가 있는데, 하나는 기존 모놀리식을 API에 래핑하고 블랙 박스처럼 다루면서 새로운 기능을 마이크로서비스로 구축하는 것이고, 다른 하나는 스트랭글러 패턴을 사용하여 모놀리식을 마이크로서비스로 리팩터링하는 것입니다. 두 방법 모두 장단점이 있지만 어떤 것을 선택하든 먼저 적절한 개발 인프라를 설정해야 합니다. 이 과정에서 실행 가능한 서비스를 개별적으로 구축 및 테스트하고 배포하기 위한 자동화된 소프트웨어 전송 파이프라인뿐만 아니라, 분산 시스템을 보호, 모니터링, 운영하고 디버깅하기 위한 인프라도 구축해야 합니다.

핵심 기능을 업데이트할 필요가 없는 독립 실행형 시스템인 경우에는 모놀리식을 그대로 유지해도 무방할 수 있습니다. 이 경우 대부분의 새로운 개발 과정에서는 API를 통해 모놀리식에 연결되는 새로운 마이크로서비스를 구축하기만 하면 될 수 있습니다. 모놀리식을 유지하거나 버릴 수 없지만 일부를 다시 작성해야 하는 경우라면 스트랭글러 패턴을 사용하는 것이 가장 좋습니다.

개발 팀은 **스트랭글러 패턴**을 사용하는 것만으로 모놀리식과 이미 잘 분리된 기능을 만들게 됩니다. 이 기능은 마이크로서비스 구축 이후 손쉬운 교체 및 폐기를 위해 API에서 모놀리식과 분리되어 있습니다. 따라서 많은 고객 대면식 앱을 변경할 필요가 없고 자체 데이터 스토어가 불필요한 기능이 적절한 첫 번째 대상입니다. 예를 들어 전자 상거래 애플리케이션에서 고려할 수 있는 몇 가지 서비스로는 인증, 인보이스 발행 또는 고객 프로필이 있습니다. 대부분의 팀은 마이크로서비스를 처음 구축할 때 기능에 맞게 최적화하는 대신 소프트웨어 전송 파이프라인과 API 접근 방식을 테스트 및 최적화하고 팀원의 기술 수준을 향상하는 데 목표를 둡니다.

기업에서 활용할 수 있는 마이크로서비스에 대해 자세히 알아보려면 **이 사이트를 참고하세요.**