에서 마이크로프런트엔드 이해 및 구현 AWS

AWS 권장 가이드

AWS 권장 가이드: 에서 마이크로프런트엔드 이해 및 구현 AWS

Table of Contents

에서 마이크로 프런트엔드 이해 및 구현 AWS

Amazon Web Services(기여자)

2024년 7월(문서 기록)

조직이 민첩성과 확장성을 위해 노력함에 따라 기존의 모놀리식 아키텍처는 종종 병목 현상이 발생하여 빠른 개발 및 배포를 방해합니다. 마이크로 프론트엔드는 복잡한 사용자 인터페이스를 자율적으로 개발, 테스트 및 배포할 수 있는 더 작고 독립적인 구성 요소로 분류하여 이를 완화합니다. 이 접근 방식은 개발 팀의 효율성을 높이고 백엔드와 프런트엔드 간의 협업을 촉진하여 분산 시스템의 end-to-end 조정을 촉진합니다.

이 규범적 지침은 다양한 전문 도메인의 IT 리더, 제품 소유자 및 아키텍트가 마이크로 프론트엔드 아키텍처를 이해하고 Amazon Web Services()에서 마이크로 프론트엔드 애플리케이션을 구축할 수 있도록 조정되었습니다AWS.

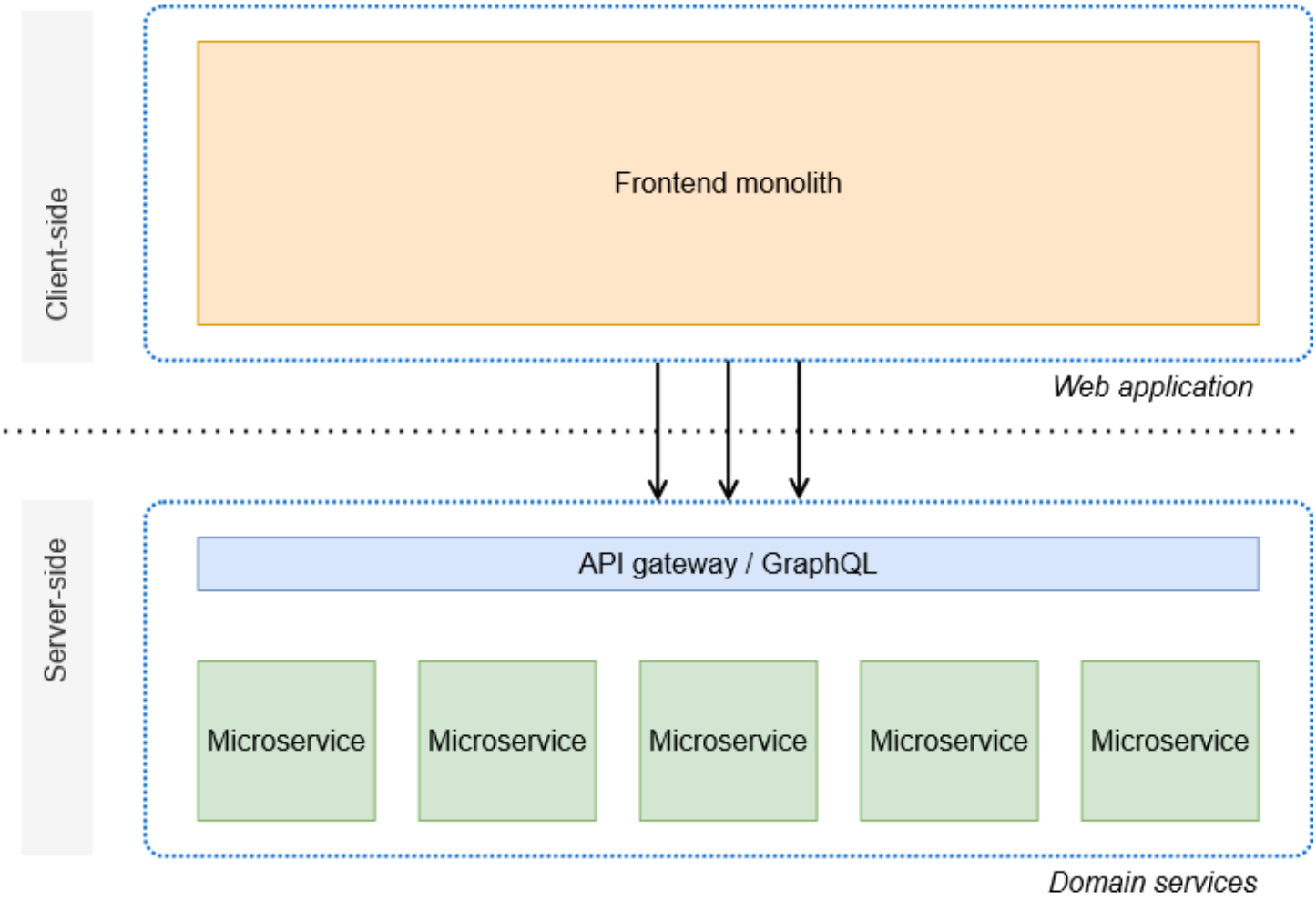
개요

마이크로 프런트엔드는 애플리케이션 프런트엔드를 독립적으로 개발 및 배포된 아티팩트로 분해하기 위해 구축된 아키텍처입니다. 대규모 프런트엔드를 자율 소프트웨어 아티팩트로 분할하면 비즈니스 로직을 캡슐화하고 종속성을 줄일 수 있습니다. 이를 통해 제품 증분을 더 빠르고 자주 제공할 수 있습니다.

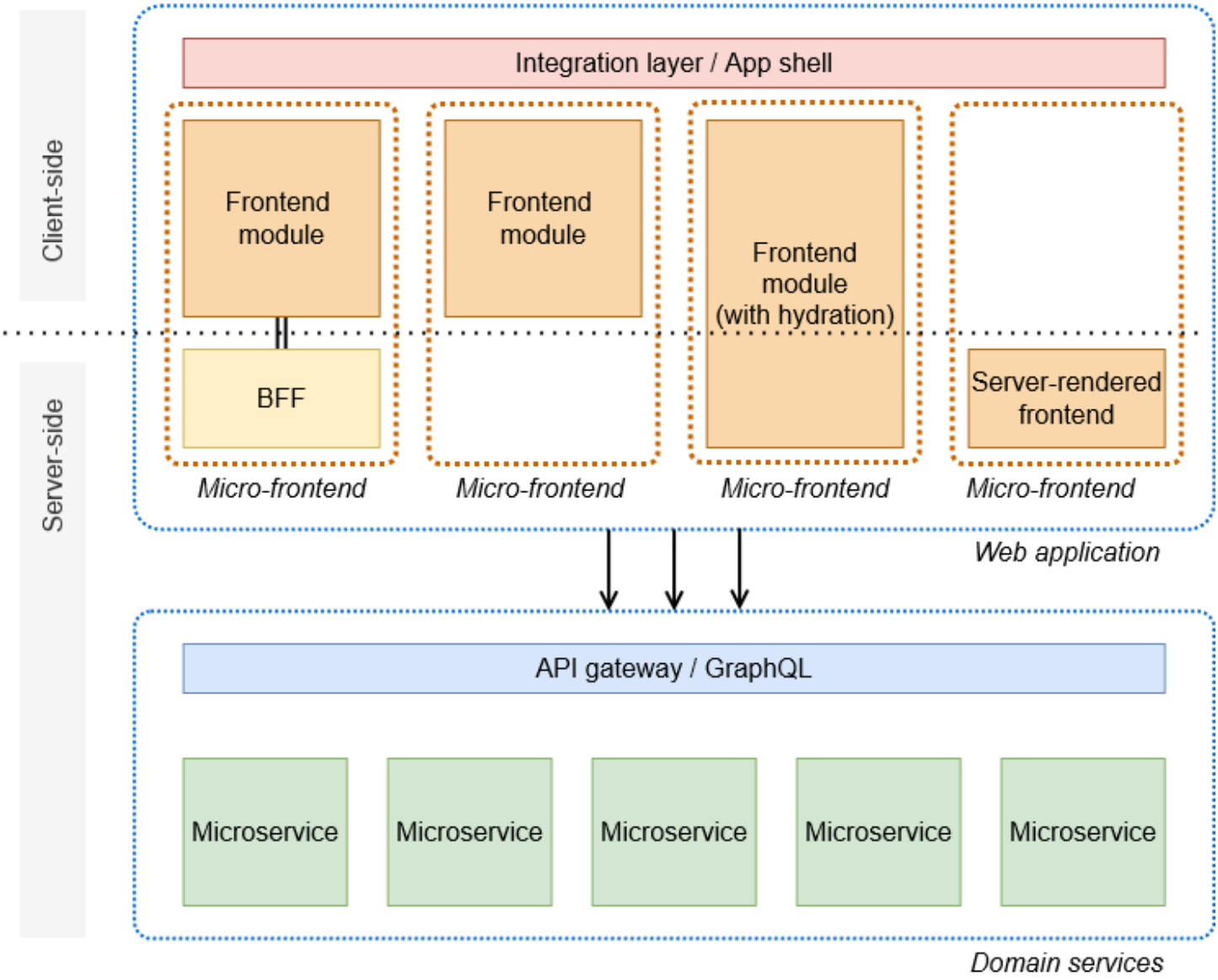
마이크로 프론트엔드는 마이크로서비스와 유사합니다. 실제로 마이크로 프론트엔드라는 용어는 마이크로서비스라는 용어에서 파생되며 마이크로서비스의 개념을 프런트엔드로 전달하는 것을 목표로 합니다. 마이크로서비스 아키텍처는 일반적으로 백엔드의 분산 시스템을 모놀리식 프런트엔드와 결합하지만 마이크로 프런트엔드는 독립형 분산 프런트엔드 서비스입니다. 이러한 서비스는 두 가지 방법으로 설정할 수 있습니다.

- 프런트엔드 전용, 마이크로서비스 아키텍처를 실행하는 공유 API 계층과 통합
- 풀 스택. 즉, 각 마이크로 프론트엔드에는 자체 백엔드 구현이 있습니다.

다음 다이어그램은 API 게이트웨이를 사용하여 백엔드 마이크로서비스에 연결하는 프런트엔드 모놀리스가 있는 기존 마이크로서비스 아키텍처를 보여줍니다.



다음 다이어그램은 마이크로서비스의 다양한 구현을 갖춘 마이크로 프론트엔드 아키텍처를 보여줍니다.



이전 다이어그램과 같이 클라이언트 측 렌더링 또는 서버 측 렌더링 아키텍처와 함께 마이크로 프런트엔드를 사용할 수 있습니다.

- 클라이언트 측 렌더링된 마이크로 프런트엔드는 중앙 집중식 APIs 노출되는 API를 직접 사용할 수 있습니다.
- 팀은 경계 컨텍스트 내에 backend-for-frontend(BFF)를 생성하여 API에 대한 프런트엔드의 진동을 줄일 수 APIs.
- 서버 측에서 마이크로 프런트엔드는 수화라는 기술을 사용하여 클라이언트 측에서 보강된 서버 측 접근 방식으로 표현할 수 있습니다. 브라우저에서 페이지를 렌더링하면 버튼 클릭과 같은 UI 요소와의 상호 작용을 허용하도록 연결된 JavaScript가 수화됩니다.

- 마이크로 프론트엔드는 백엔드에서 렌더링하고 하이퍼링크를 사용하여 웹 사이트의 새 부분으로 라우팅할 수 있습니다.

마이크로 프론트엔드는 다음을 수행하려는 조직에 적합합니다.

- 동일한 프로젝트에서 작업하는 여러 팀으로 확장합니다.
- 의사 결정의 분산을 수용하여 개발자가 식별된 시스템 경계 내에서 혁신할 수 있도록 지원합니다.

이 접근 방식은 팀의 인지 부하를 크게 줄입니다. 팀의 인지 부하가 시스템의 특정 부분을 담당하게 되기 때문입니다. 나머지를 중단하지 않고 시스템의 한 부분을 수정할 수 있으므로 비즈니스 민첩성이 향상됩니다.

마이크로 프론트엔드는 고유한 아키텍처 접근 방식입니다. 마이크로 프런트엔드를 구축하는 방법에는 여러 가지가 있지만 모두 공통적인 특성이 있습니다.

- 마이크로 프론트엔드 아키텍처는 여러 독립 요소로 구성됩니다. 구조는 백엔드의 마이크로서비스에서 발생하는 모듈화와 유사합니다.
- 마이크로 프런트엔드는 다음으로 구성된 경계 컨텍스트 내에서 프런트엔드 구현을 전적으로 담당합니다.
  - 사용자 인터페이스
  - 데이터
  - 상태 또는 세션
  - 비즈니스 로직
  - 플로우

경계가 지정된 컨텍스트는 내부적으로 일관된 시스템으로, 경계를 신중하게 설계하여 출입할 수 있는 항목을 중재합니다. 마이크로 프론트엔드는 가능한 한 적은 비즈니스 로직 및 데이터를 다른 마이크로 프론트엔드와 공유해야 합니다. 공유가 필요한 모든 곳에서 사용자 지정 이벤트 또는 사후 대응 스트림과 같이 명확하게 정의된 인터페이스를 통해 이루어집니다. 그러나 설계 시스템 또는 로깅 라이브러리와 같은 몇 가지 교차 커팅 문제에 대해서는 의도적인 공유를 환영합니다.

권장되는 패턴은 부서 간 팀을 사용하여 마이크로 프런트엔드를 구축하는 것입니다. 즉, 각 마이크로 프런트엔드는 백엔드에서 프런트엔드로 작업하는 동일한 팀이 개발합니다. 코딩부터 프로덕션 환경의 시스템 운영에 이르기까지 팀 소유권은 매우 중요합니다.

이 지침은 하나의 특정 접근 방식을 권장하지 않습니다. 대신 다양한 패턴, 모범 사례, 장단점, 아키텍처 및 조직 고려 사항에 대해 설명합니다.

기본 개념

마이크로 프론트엔드 아키텍처는 이전의 세 가지 아키텍처 개념에서 많은 영감을 받았습니다.

- 도메인 기반 설계는 복잡한 애플리케이션을 일관된 도메인으로 구조화하는 멘탈 모델입니다.
- 분산 시스템은 애플리케이션을 독립적으로 개발하여 자체 전용 인프라에서 실행되는 느슨하게 연결된 서브시스템으로 구축하는 접근 방식입니다.
- 클라우드 컴퓨팅은 모델을 사용하여 IT 인프라를 서비스로 실행하는 접근 방식입니다 pay-as-you-go .

도메인 기반 설계

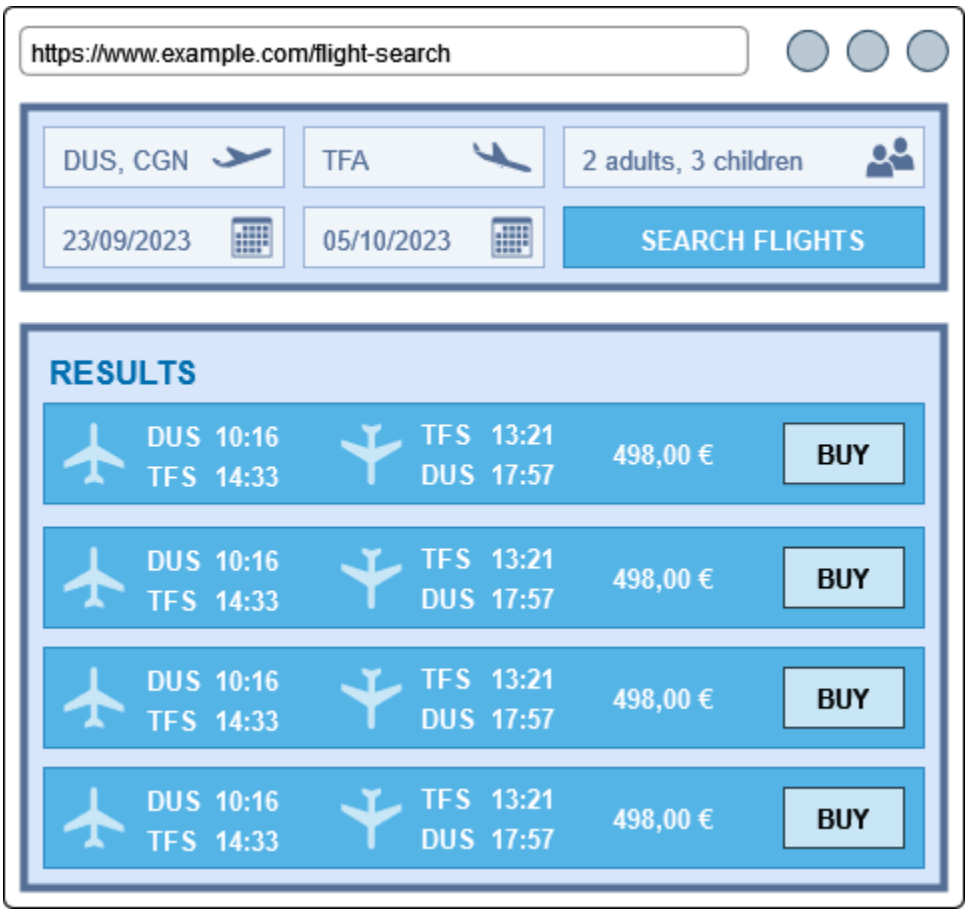
도메인 기반 디자인 (DDD) 은 에릭 에반스가 개발한 패러다임입니다. 에반스는 2003년 저서 '도메인 기반 디자인: 소프트웨어 중심부의 복잡성 해결하기'에서 소프트웨어 개발은 기술적인 문제보다는 비즈니스 문제에 의해 주도되어야 한다고 주장합니다. Evans는 IT 프로젝트에서 기술 및 분야 전문가가 공통된 이해를 찾는 데 도움이 되는 유비쿼터스 언어를 먼저 개발해야 한다고 제안합니다. 이러한 언어를 기반으로 이들은 상호 이해되는 비즈니스 현실 모델을 만들 수 있습니다.

이러한 접근 방식이 당연하겠지만, 많은 소프트웨어 프로젝트는 비즈니스와 IT 간의 단절로 어려움을 겪고 있습니다. 이러한 연결이 끊기면 심각한 오해가 발생하여 예산 초과, 품질 저하 또는 프로젝트 실패로 이어지는 경우가 많습니다.

Evans는 그 외에도 여러 가지 중요한 용어를 소개하는데, 그 중 하나가 경계 문맥입니다. 제한된 컨텍스트는 정확히 하나의 비즈니스 문제에 대한 솔루션 또는 구현을 포함하는 대규모 IT 응용 프로그램의 독립된 세그먼트입니다. 대규모 응용 프로그램은 통합 패턴을 통해 느슨하게 결합되는 여러 개의 제한된 컨텍스트로 구성됩니다. 이러한 제한된 컨텍스트에는 유비쿼터스 언어의 고유한 방언이 있을 수도 있습니다. 예를 들어, 결제 시에는 배송이라는 개념이 무의미하기 때문에 애플리케이션의 결제 컨텍스트에 있는 사용자는 전송 컨텍스트의 사용자와 다른 측면을 가질 수 있습니다.

Evans는 제한된 컨텍스트가 얼마나 작거나 커야 하는지를 정의하지 않습니다. 크기는 소프트웨어 프로젝트에 따라 결정되며 시간이 지남에 따라 달라질 수 있습니다. 컨텍스트의 경계를 나타내는 좋은 지표는 개체 (도메인 개체) 와 비즈니스 로직 간의 통합 정도입니다.

마이크로 프론트엔드의 경우 항공편 예약 페이지와 같은 복잡한 웹 페이지를 예로 들어 도메인 기반 설계를 설명할 수 있습니다.



이 페이지의 주요 구성 요소는 검색 양식, 필터 패널, 결과 목록입니다. 경계를 식별하려면 독립적인 기능 컨텍스트를 식별해야 합니다. 또한 재사용성, 성능 및 보안과 같은 비기능적 측면도 고려하십시오. “서로 연관되어 있는 것”을 나타내는 가장 중요한 지표는 커뮤니케이션 패턴입니다. 아키텍처의 일부 요소가 자주 통신하고 복잡한 정보를 교환해야 하는 경우 동일한 경계 컨텍스트를 공유할 가능성이 높습니다.

버튼과 같은 개별 UI 요소는 기능적으로 독립적이지 않기 때문에 제한된 컨텍스트가 아닙니다. 또한 전체 페이지는 더 작은 독립 컨텍스트로 나눌 수 있기 때문에 제한된 컨텍스트에는 적합하지 않습니다. 검색 양식을 하나의 제한된 컨텍스트로 취급하고 결과 목록을 두 번째 경계 컨텍스트로 처리하는 것이 합리적입니다. 이제 이러한 두 개의 제한된 컨텍스트를 각각 별도의 마이크로 프론트엔드로 구현할 수 있습니다.

분산 시스템

유지 관리를 용이하게 하고 발전 능력을 지원하기 위해 대다수의 사소하지 않은 IT 솔루션은 모듈식입니다. 이 경우 모듈식이란 IT 시스템이 식별 가능한 구성 요소로 구성되고 인터페이스를 통해 분리되어 문제를 분리하는 것을 의미합니다.

분산 시스템은 모듈식일 뿐만 아니라 그 자체로도 독립된 시스템이어야 합니다. 단순한 모듈식 시스템에서 각 모듈은 이상적으로 캡슐화되어 인터페이스를 통해 기능을 노출하지만 독립적으로 배포할 수는 없으며 자체적으로 작동할 수도 없습니다. 또한 모듈은 일반적으로 동일한 시스템에 속한 다른 모듈과 동일한 수명 주기를 따릅니다. 반면 분산 시스템의 구성 요소에는 각각 고유한 수명 주기가 있습니다. 도메인 기반 설계 패러다임을 적용하면 각 빌딩 블록은 하나의 비즈니스 도메인 또는 하위 도메인을 다루며 고유한 경계 내에서 작동합니다.

구축 중에 분산 시스템이 상호 작용하는 경우 일반적인 접근 방식은 문제를 신속하게 식별할 수 있는 메커니즘을 개발하는 것입니다. 예를 들어, 형식화된 언어를 채택하고 단위 테스트에 막대한 투자를 할 수 있습니다. 여러 팀이 모듈의 개발 및 유지 관리를 위해 협업할 수 있습니다. 모듈의 개발 및 유지 관리는 시스템이 npm, Apache Maven, NuGet pip와 같은 도구를 사용하여 사용할 수 있도록 라이브러리로 배포되는 경우가 많습니다.

런타임 동안 상호 작용하는 분산 시스템은 일반적으로 개별 팀이 소유합니다. 종속성이 사용되면 오류 처리, 성능 균형 조정 및 보안으로 인해 운영이 복잡해집니다. 통합 테스트 및 옵저버빌리티에 대한 투자는 위험을 줄이는 데 필수적입니다.

오늘날 가장 인기 있는 분산 시스템의 예는 마이크로서비스입니다. 마이크로서비스 아키텍처에서 백엔드 서비스는 UI 또는 인증과 같은 기술적 문제가 아닌 도메인 중심이며 자율 팀이 소유합니다. 마이크로 프런트엔드는 동일한 원칙을 공유하여 솔루션 범위를 프런트엔드까지 확장합니다.

클라우드 컴퓨팅

클라우드 컴퓨팅은 자체 데이터 센터를 구축하고 온프레미스에서 운영하기 위한 하드웨어를 구매하는 대신 pay-as-you-go 모델을 통해 IT 인프라를 서비스로 구매하는 방법입니다. 클라우드 컴퓨팅은 다음과 같은 몇 가지 이점을 제공합니다.

- 조직은 사전에 대규모 장기 재정 약정을 하지 않고도 새로운 기술을 실험할 수 있어 비즈니스 민첩성이 크게 향상됩니다.
- 와 같은 클라우드 공급자를 사용하면 조직은 API 게이트웨이 AWS, 데이터베이스, 컨테이너 오케스트레이션, 클라우드 기능 등 유지 관리가 거의 필요 없고 통합성이 뛰어난 서비스로 구성된 광범위한 포트폴리오를 이용할 수 있습니다. 이러한 서비스에 액세스하면 직원들이 경쟁 업체와 차별화되는 업무에 집중할 수 있습니다.
- 조직이 전 세계에 솔루션을 출시할 준비가 되면 전 세계 클라우드 인프라에 솔루션을 배포할 수 있습니다.

클라우드 컴퓨팅은 고도로 관리되는 인프라를 제공하여 마이크로 프론트엔드를 지원합니다. 따라서 부서 간 팀이 더 쉽게 end-to-end 소유권을 가질 수 있습니다. 팀은 강력한 운영 지식을 갖추고 있어야 하지만 인프라 프로비저닝, 운영 체제 업데이트, 네트워킹과 같은 수동 작업은 방해가 될 수 있습니다.

마이크로 프론트엔드는 제한된 상황에 있기 때문에 팀이 가장 적합한 서비스를 선택하여 실행할 수 있습니다. 예를 들어 팀은 클라우드 함수와 컴퓨팅용 컨테이너 중에서 선택하고 다양한 종류의 SQL 및 NoSQL 데이터베이스 또는 인메모리 캐시 중에서 선택할 수 있습니다. 팀은 서버리스 인프라를 위한 사전 구성된 빌딩 블록과 함께 제공되는 고도로 통합된 툴킷을 기반으로 마이크로 프론트엔드를 구축할 수도 있습니다. AWS Amplify

마이크로 프런트엔드와 대체 아키텍처 비교

모든 아키텍처 전략과 마찬가지로 마이크로 프론트엔드 채택 결정은 조직의 원칙에 따라 내려지는 평가 기준을 기반으로 해야 합니다. 마이크로 프론트엔드에는 장단점이 있습니다. 조직에서 마이크로 프런트엔드를 사용하기로 결정한 경우 분산 시스템의 문제를 해결하기 위한 전략이 있어야 합니다.

애플리케이션 아키텍처를 선택할 때 마이크로 프런트엔드에 대한 가장 널리 사용되는 대안은 모놀리스, n-티어 애플리케이션 및 단일 페이지 애플리케이션(SPA) 프런트엔드와 결합된 마이크로서비스입니다. 모두 유효한 접근 방식이며 각 접근 방식에는 장단점이 있습니다.

모놀리스

자주 변경할 필요가 없는 작은 애플리케이션은 모놀리스로 매우 빠르게 제공할 수 있습니다. 상당한 성장이 예상되는 상황에서도 모놀리스는 자연스러운 첫 번째 단계입니다. 나중에 모놀리스를 사용 중지하거나 보다 유연한 구조로 리팩터링할 수 있습니다. 모놀리스부터 시작하여 조직은 시장에 진출하고, 고객 피드백을 받고, 제품을 더 빠르게 개선할 수 있습니다.

그러나 모놀리식 애플리케이션은 신중하게 유지 관리하지 않거나 시간이 지남에 따라 코드베이스의 크기가 증가함에 따라 성능이 저하되는 경향이 있습니다. 여러 팀이 동일한 코드베이스에 크게 기여하면 유지 관리 및 운영에 모두 기여하는 경우는 거의 없습니다. 이로 인해 책임의 불균형이 발생하여 속도에 영향을 미치고 비효율성이 발생합니다. 동시에 모놀리스의 모듈 간에 실수로 결합하면 코드 베이스가 발전함에 따라 의도하지 않은 부작용이 발생합니다. 이러한 부작용으로 인해 오작동 및 중단이 발생할 수 있습니다.

N 티어 애플리케이션

비교적 정적인 진화 속도를 가진 보다 복잡한 애플리케이션은 프런트엔드와 백엔드 사이에 REST 또는 GraphQL 계층이 있는 3계층 아키텍처(프레젠테이션, 애플리케이션, 데이터)로 구축할 수 있습니다. 이는 훨씬 더 유연하며 여러 계층의 팀이 어느 정도 독립적으로 개발할 수 있습니다. n-티어 애플리케이션의 단점은 기능을 배포하기가 훨씬 더 어렵다는 것입니다. 프런트엔드와 백엔드는 API 계약을 통해 분리되므로 주요 변경 사항을 함께 배포하거나 API 버전을 지정해야 합니다.

다음 일반적인 시나리오를 고려하세요. 새 기능을 릴리스하려면 데이터 스키마를 변경해야 하는 경우 제품 소유자가 프런트엔드 팀과 기능 세트에 합의하는 데 며칠이 걸릴 수 있습니다. 그런 다음 프런트엔드 팀은 백엔드 팀에 기능을 개발하고 릴리스하도록 요청합니다. 백엔드 팀은 데이터 소유자와 협력하여 데이터베이스 스키마 업데이트를 릴리스합니다. 그런 다음 프런트엔드 팀이 변경 사항을 개발하고 릴리스할 수 있도록 백엔드 팀이 API의 새 버전을 릴리스합니다. 이 시나리오에서는 각 팀에 변경

사항 개발, 테스트 및 릴리스와 관련된 자체 백로그, 우선순위 및 메커니즘이 있기 때문에 모든 변경 사항을 프로덕션에 전파하는 데 몇 주 또는 몇 달이 걸릴 수 있습니다.

마이크로서비스

마이크로서비스 아키텍처에서 백엔드는 각각 제한된 컨텍스트 내에서 특정 비즈니스 문제를 해결하는 소규모 서비스로 분해됩니다. 또한 각 마이크로서비스는 명확하게 정의된 인터페이스 계약을 노출하여 다른 서비스와 강력하게 분리됩니다.

경계 컨텍스트 및 인터페이스 계약은 잘 만들어진 모놀리스 및 n계층 아키텍처에도 존재해야 합니다. 그러나 마이크로서비스 아키텍처에서는 일반적으로 HTTP 프로토콜과 같은 네트워크를 통해 통신이 이루어지며 서비스에는 전용 런타임 인프라가 있습니다. 이는 각 백엔드 서비스의 독립적인 개발, 제공 및 운영을 지원합니다.

요구 사항에 맞는 접근 방식 선택

모놀리스 및 n-티어 아키텍처는 여러 도메인 문제를 하나의 기술 아티팩트로 번들링합니다. 따라서 종속성 및 내부 데이터 흐름과 같은 측면을 쉽게 관리할 수 있지만 새로운 기능을 제공하기가 더 어렵습니다. 일관성 있는 코드 기반을 유지하기 위해 팀은 처리해야 하는 대규모 코드 기반 때문에 리팩터링 및 디커플링에 시간을 투자하는 경우가 많습니다.

일부 팀에서 개발한 애플리케이션은 마이크로 프런트엔드로 전환할 때 발생하는 추가 복잡성이 필요하지 않을 수 있습니다. 이는 팀이 높은 결합과 릴리스에 대한 긴 리드 타임으로 인한 벌금을 지불하지 않는 경우 특히 그렇습니다.

요약하면 더 복잡하고 분산된 아키텍처는 복잡하고 빠르게 움직이는 애플리케이션에 적합한 선택인 경우가 많습니다. 중소 규모의 애플리케이션의 경우 분산 아키텍처가 모놀리식 아키텍처보다 반드시 우수하지는 않습니다. 특히 애플리케이션이 단기간에 크게 발전하지 않을 경우에는 더욱 그렇습니다.

마이크로 프런트엔드의 아키텍처 결정

애플리케이션에 마이크로 프런트엔드 아키텍처 패턴을 적용하는 팀은 아키텍처에 대한 몇 가지 결정을 초기에 내려야 합니다.

- 마이크로 프런트엔드 식별 및 경계 정의
- 마이크로 프런트엔드로 페이지 및 뷰 구성
- 마이크로 프런트엔드 간 라우팅, 상태 관리 및 통신
- 교차 커팅 문제에 대한 종속성 관리

다음 섹션에서는 이러한 주제를 더 자세히 다룹니다.

아키텍처 결정을 내릴 때는 올바른 지표를 가지고 사용 패턴 애플리케이션 특성과 장단점을 이해하는 것이 중요합니다. 예를 들어 전자 상거래 사이트의 특성 및 사용 패턴은 비디오 편집 도구 또는 관찰성 대시보드와 다릅니다.

트래픽이 많고 세션 깊이가 짧은 퍼블릭 애플리케이션은 TTI(Time to Interactive) 및 FCP(First Contentful Paint)와 같은 초기 페이지 로드 지표에 최적화할 수 있습니다. 반면 사용자가 하루를 시작할 때 로그인하고 하루 종일 계속 상호 작용하는 애플리케이션은 인애플리케이션 환경에 최적화될 수 있습니다. 애플리케이션 팀은 초기 페이지 로드 대신 각 탐색 후 첫 번째 입력 지연(FID) 지표에 맞게 최적화할 수 있습니다.

퍼블릭 웹 사이트는 다양한 브라우저 환경을 충족해야 합니다. 클라이언트 환경에 알려진 제약 조건이 있는 엔터프라이즈 애플리케이션은 제약 조건에 따라 마이크로 프런트엔드 구성을 최적화할 수 있습니다.

아키텍처 결정에 적합한 선택은 없습니다. 장단점, 비즈니스가 운영되는 컨텍스트, 사용 패턴 및 지표를 이해하여 각 개별 애플리케이션에 적합한 결정을 안내합니다.

마이크로 프런트엔드 경계 식별

팀 자율성을 개선하기 위해 애플리케이션에서 제공하는 비즈니스 기능을 서로에 대한 종속성을 최소화하면서 여러 마이크로 프런트엔드로 분해할 수 있습니다.

앞서 설명한 DDD 방법론에 따라 팀은 애플리케이션 도메인을 비즈니스 하위 도메인과 제한된 컨텍스트로 나눌 수 있습니다. 그런 다음 자율 팀은 제한된 컨텍스트의 기능을 소유하고 이러한 컨텍스트를 마이크로 프런트엔드로 제공할 수 있습니다.

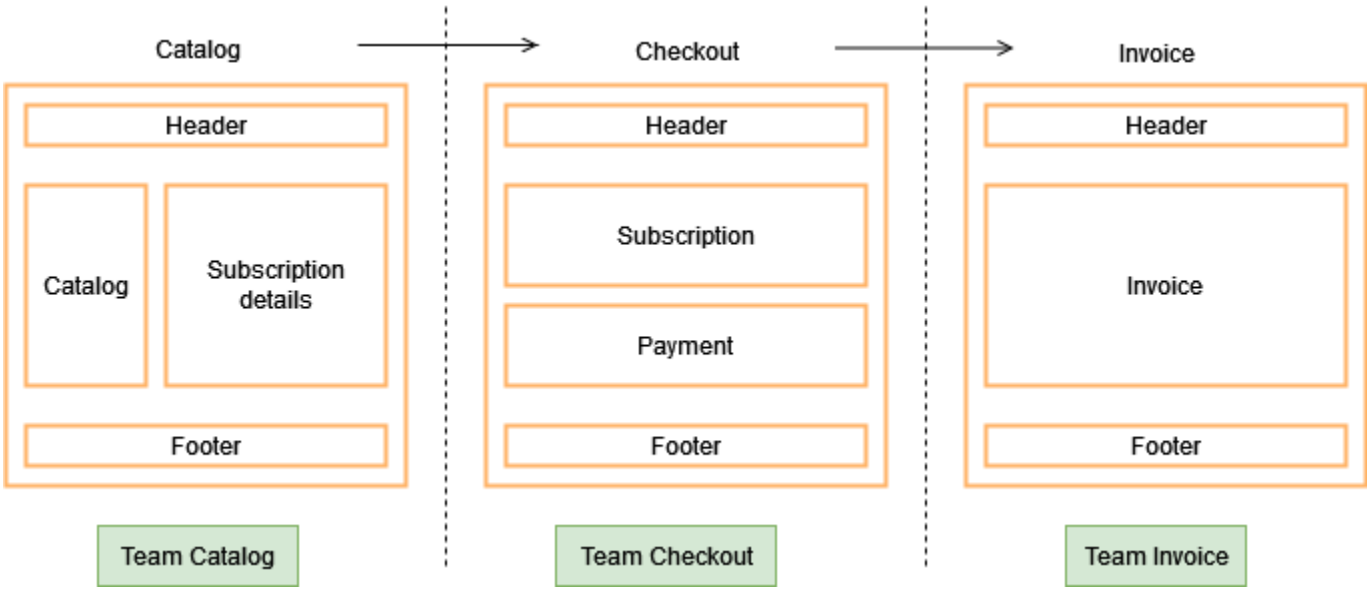
경계가 잘 정의된 컨텍스트는 기능적 중복과 컨텍스트 간 런타임 통신의 필요성을 최소화해야 합니다. 필요한 통신은 이벤트 기반 방법으로 구현할 수 있습니다. 이는 마이크로서비스 개발을 위한 이벤트 기반 아키텍처와 다르지 않습니다.

또한 잘 설계된 애플리케이션은 고객에게 일관된 경험을 제공하기 위해 새 팀의 향후 확장 제공을 지원해야 합니다.

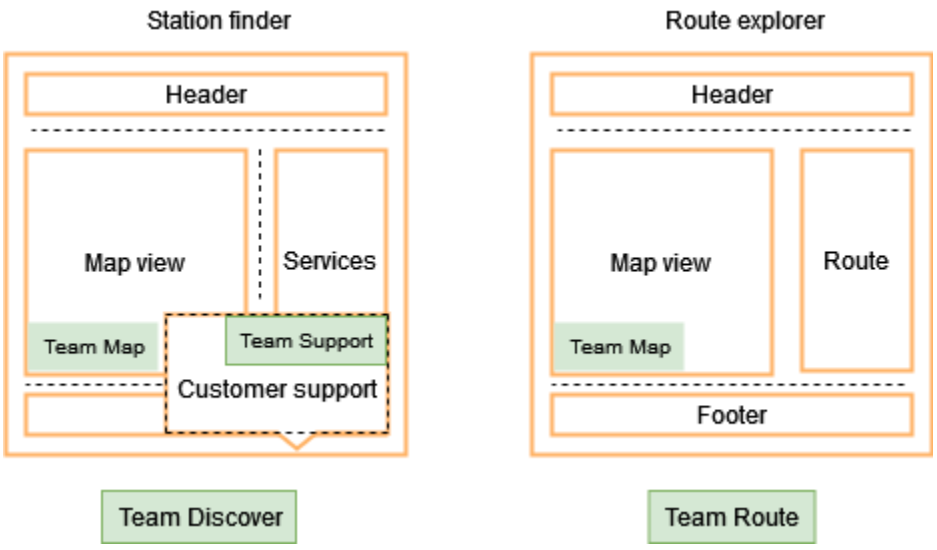
모놀리식 애플리케이션을 마이크로 프런트엔드로 분할하는 방법

개요 섹션에는 웹 페이지에서 독립적인 기능 컨텍스트를 식별하는 예제가 포함되어 있습니다. 사용자 인터페이스에서 기능을 분할하기 위한 몇 가지 패턴이 나타납니다.

예를 들어 비즈니스 도메인이 사용자 여정의 단계를 구성하는 경우 프런트엔드에 수직 분할을 적용할 수 있습니다. 여기서 사용자 여정의 뷰 모음은 마이크로 프런트엔드로 전달됩니다. 다음 다이어그램은 카탈로그, 체크아웃 및 인보이스 단계가 별도의 팀에서 별도의 마이크로 프런트엔드로 제공되는 세로 분할을 보여줍니다.



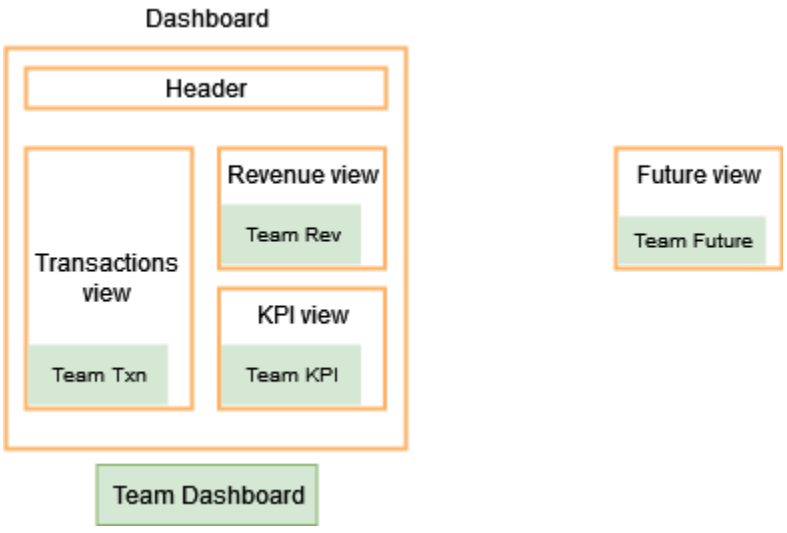
일부 애플리케이션의 경우 수직 분할만으로는 충분하지 않을 수 있습니다. 예를 들어 일부 기능은 여러 뷰에서 제공해야 할 수 있습니다. 이러한 애플리케이션의 경우 혼합 분할을 적용할 수 있습니다. 다음 다이어그램은 Station Finder 및 Route Explorer의 마이크로 프런트엔드가 모두 맵 보기 기능을 사용하는 혼합 분할 솔루션을 보여줍니다.



포털 유형 또는 대시보드 유형 애플리케이션은 일반적으로 프런트엔드 기능을 단일 보기로 통합합니다. 이러한 유형의 애플리케이션에서는 각 위젯을 마이크로 프론트엔드로 제공할 수 있으며, 호스팅 애플리케이션은 마이크로 프론트엔드가 구현해야 하는 제약 조건과 인터페이스를 정의합니다.

이 접근 방식은 마이크로 프런트엔드가 뷰포트 크기 조정, 인증 공급자, 구성 설정 및 메타데이터와 같은 문제를 처리할 수 있는 메커니즘을 제공합니다. 이러한 유형의 애플리케이션은 확장성을 최적화합니다. 새 팀은 대시보드 기능을 확장하기 위해 새로운 기능을 개발할 수 있습니다.

다음 다이어그램은 팀 대시보드의 일부인 세 개의 개별 팀에서 개발한 대시보드 애플리케이션을 보여줍니다.



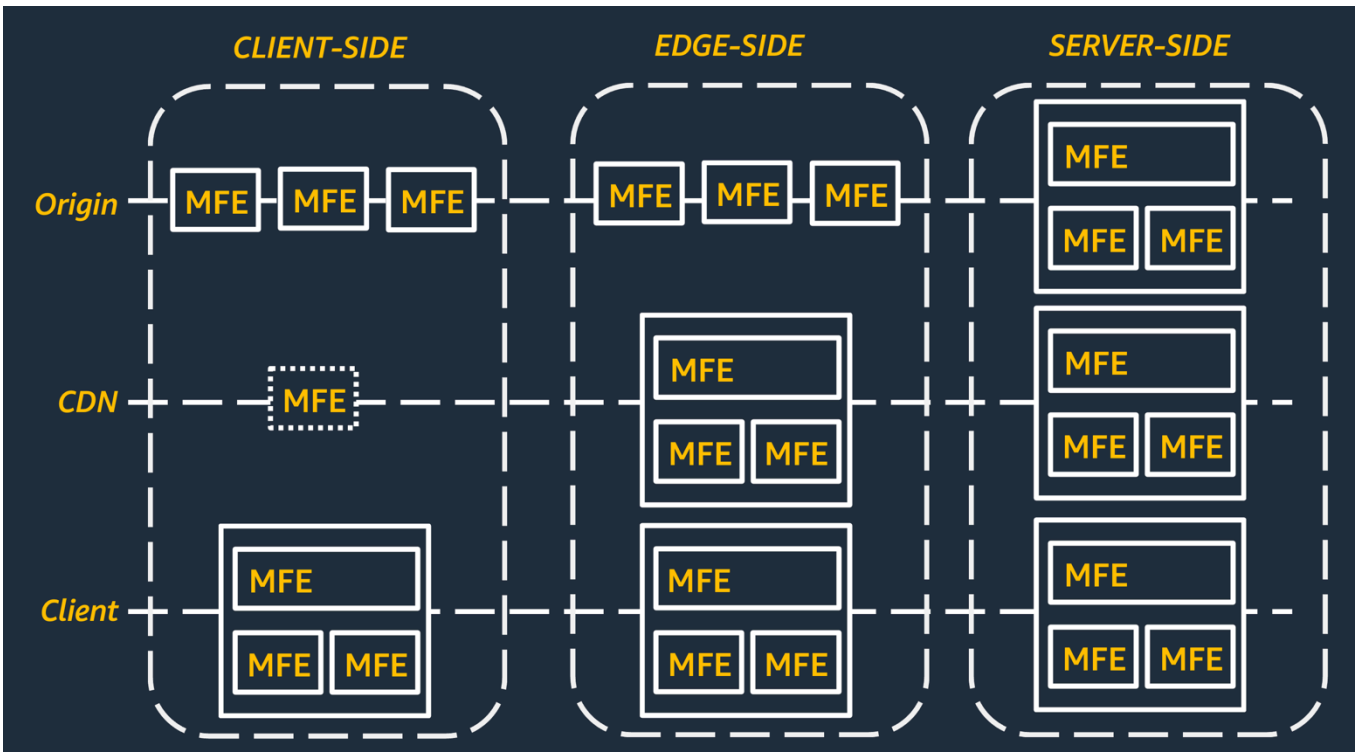
다이어그램에서 미래 보기는 팀 대시보드와 대시보드 기능을 확장하기 위해 새 팀이 개발한 새로운 기능을 나타냅니다.

포털 및 대시보드 애플리케이션은 일반적으로 UI에서 혼합 분할을 사용하여 기능을 구성합니다. 마이크로 프런트엔드는 위치 및 크기 제약 조건을 포함하여 잘 정의된 설정으로 구성할 수 있습니다.

마이크로 프런트엔드로 페이지 및 뷰 구성

클라이언트 측 구성, 엣지 측 구성 및 서버 측 구성을 사용하여 애플리케이션의 보기를 구성할 수 있습니다. 구성 패턴은 필요한 팀 기술, 내결함성, 성능 및 캐시 동작 측면에서 서로 다른 특성을 갖습니다.

다음 다이어그램은 마이크로 프런트엔드 아키텍처의 클라이언트 측, 엣지 측 및 서버 측 계층에서 구성이 어떻게 이루어지는지 보여줍니다.



클라이언트 측, 엣지 측 및 서버 측 계층은 다음 섹션에서 설명합니다.

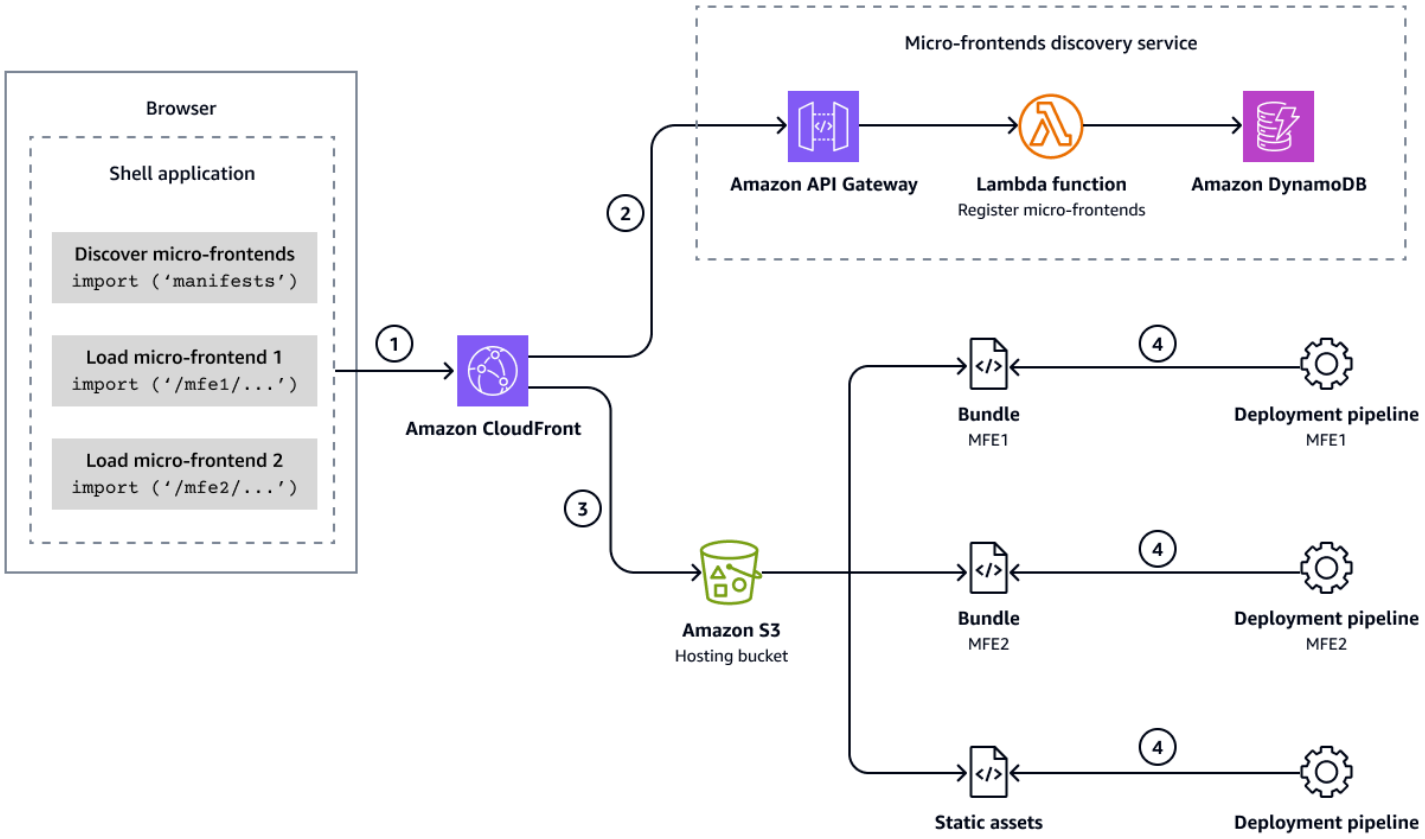
클라이언트 측 구성

클라이언트(브라우저 또는 모바일 웹 보기)에서 마이크로 프런트엔드를 문서 객체 모델(DOM) 조각으로 동적으로 로드하고 추가합니다. JavaScript 또는 CSS 파일과 같은 마이크로 프론트엔드 아티팩트는 지연 시간을 줄이기 위해 콘텐츠 전송 네트워크(CDNs)에서 로드할 수 있습니다. 클라이언트 측 구성에는 다음이 필요합니다.

- 브라우저에서 런타임에 마이크로 프론트엔드 구성 요소를 검색, 로드 및 렌더링할 수 있도록 셸 애플리케이션 또는 마이크로 프론트엔드 프레임워크를 소유하고 유지 관리하는 팀

- HTML, CSS, JavaScript와 같은 프런트엔드 기술의 높은 기술 수준과 브라우저 환경에 대한 심층적인 이해
- 페이지에 로드된 JavaScript 양 최적화 및 글로벌 네임스페이스 충돌을 방지하기 위한 원칙

다음 다이어그램은 서버리스 클라이언트 측 구성을 위한 예제 AWS 아키텍처를 보여줍니다.



클라이언트 측 구성은 쉘 애플리케이션을 통해 브라우저 환경에서 발생합니다. 다이어그램은 다음 세부 정보를 보여줍니다.

1. 쉘 애플리케이션을 로드한 후 Amazon CloudFront에 매니페스트 엔드포인트를 통해 로드할 마이크로 프런트엔드를 검색하도록 초기 요청합니다.
2. 매니페스트에는 각 마이크로 프론트엔드에 대한 정보(예: 이름, URL, 버전 및 대체 동작)가 포함됩니다. 매니페스트는 마이크로 프런트엔드 검색 서비스에서 제공됩니다. 다이어그램에서이 검색 서비스는 Amazon API Gateway, AWS Lambda 함수 및 Amazon DynamoDB로 표시됩니다. 쉘 애플리케이션은 매니페스트 정보를 사용하여 개별 마이크로 프런트엔드가 지정된 레이아웃 내에서 페이지를 작성하도록 요청합니다.

3. 각 마이크로 프론트엔드 번들은 정적 파일(예: JavaScript, CSS, HTML)로 구성됩니다. 파일은 Amazon Simple Storage Service(Amazon S3) 버킷에서 호스팅되며 CloudFront를 통해 제공됩니다.
4. 팀은 자신이 소유한 배포 파이프라인을 사용하여 마이크로 프런트엔드의 새 버전을 배포하고 매니페스트 정보를 업데이트할 수 있습니다.

엣지 측 구성

오리진 서버 앞의 일부 CDNs 및 프록시에서 지원하는 엣지 측 포함(ESI) 또는 서버 측 포함(SSI)과 같은 트랜스클루전 기술을 사용하여 클라이언트에 유선으로 전송하기 전에 페이지를 작성합니다. ESI에는 다음이 필요합니다.

- ESI 기능이 있는 CDN 또는 서버 측 마이크로 프런트엔드 앞에 프록시 배포. HAProxy, Varnish 및 NGINX와 같은 프록시 구현은 SSI를 지원합니다.
- ESI 및 SSI 구현의 사용 및 제한에 대한 이해.

새 애플리케이션을 시작하는 팀은 일반적으로 구성 패턴에 엣지 측 구성을 선택하지 않습니다. 그러나 이 패턴은 트랜스클루전에 의존하는 레거시 애플리케이션에 대한 경로를 제공할 수 있습니다.

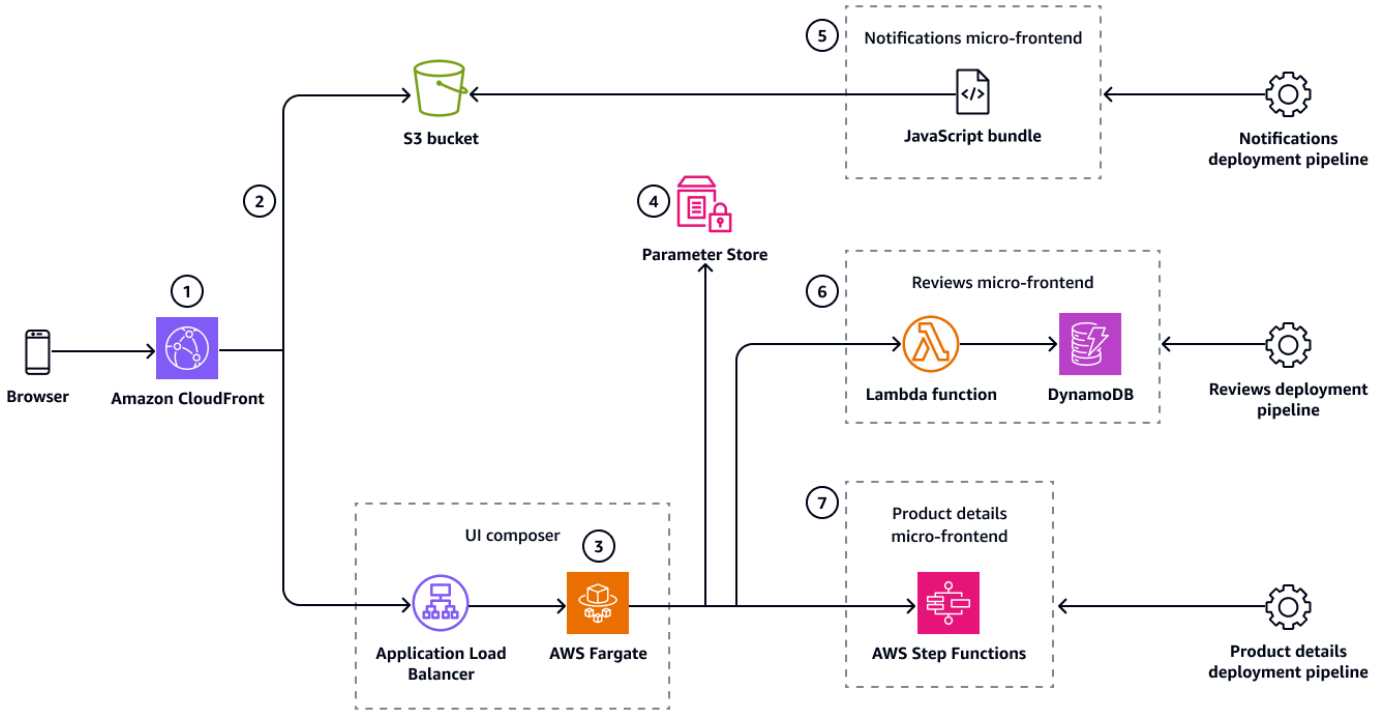
서버 측 구성

오리진 서버를 사용하여 엣지에 캐시되기 전에 페이지를 작성합니다. 이는 PHP, JSP(자카르타 서버 페이지) 또는 템플릿 라이브러리와 같은 기존 기술을 사용하여 마이크로 프런트엔드의 조각을 포함하여 페이지를 구성할 수 있습니다. 서버에서 실행되는 Next.js와 같은 JavaScript 프레임워크를 사용하여 서버의 페이지를 서버 측 렌더링(SSR)으로 구성할 수도 있습니다.

서버에서 페이지를 렌더링한 후 CDNs에 캐시하여 지연 시간을 줄일 수 있습니다. 마이크로 프론트엔드의 새 버전이 배포되면 페이지를 다시 렌더링하고 최신 버전을 고객에게 전달하도록 캐시를 업데이트해야 합니다.

서버 측 구성을 위해서는 서버 환경을 심층적으로 파악하여 배포, 서버 측 마이크로 프런트엔드 검색 및 캐시 관리를 위한 패턴을 설정해야 합니다.

다음 다이어그램은 서버 측 구성을 보여줍니다.



다이어그램에는 다음 구성 요소와 프로세스가 포함되어 있습니다.

1. Amazon CloudFront는 애플리케이션에 고유한 진입점을 제공합니다. 배포에는 두 개의 오리진이 있습니다. 하나는 정적 파일에 대한 오리진이고 다른 하나는 UI 컴포저에 대한 오리진입니다.
2. 정적 파일은 Amazon S3 버킷에서 호스팅됩니다. 브라우저와 HTML 템플릿용 UI 작성기에서 사용됩니다.
3. UI 작성기는의 컨테이너 클러스터에서 실행됩니다AWS Fargate. 컨테이너화된 솔루션을 사용하면 필요한 경우 스트리밍 기능과 멀티스레드 렌더링을 사용할 수 있습니다.
4. 의 기능인 Parameter Store AWS Systems Manager는 기본 마이크로 프런트엔드 검색 시스템으로 사용됩니다. 이 기능은 UI 작성기가 사용할 마이크로 프런트엔드 엔드포인트를 검색하는 데 사용하는 키-값 저장소를 제공합니다.
5. 알림 마이크로 프런트엔드는 최적화된 JavaScript 번들을 S3 버킷에 저장합니다. 이는 사용자 상호 작용에 반응해야 하므로 클라이언트에서 렌더링됩니다.
6. 리뷰 마이크로 프론트엔드는 Lambda 함수로 구성되며 사용자 리뷰는 DynamoDB에 저장됩니다. 리뷰 마이크로 프론트엔드는 서버 측에서 완전히 렌더링되고 HTML 조각을 출력합니다.
7. 제품 세부 정보 마이크로 프론트엔드는를 사용하는 로우 코드 마이크로 프론트엔드입니다AWS Step Functions. Express 워크플로는 동기식으로 호출할 수 있으며 HTML 조각과 캐싱 계층을 렌더링하기 위한 로직이 포함되어 있습니다.

서버 측 구성에 대한 자세한 내용은 블로그 게시물 서버 측 렌더링 마이크로 프론트엔드 - 아키텍처를 참조하세요.

마이크로 프런트엔드 간 라우팅 및 통신

라우팅 옵션은 구성 접근 방식에 따라 달라집니다. 프런트엔드 구성 요소 간의 결합을 줄여 통신을 최적화할 수 있습니다.

라우팅

수직 분할과 함께 클라이언트 측 구성을 사용하는 애플리케이션은 서버 측 라우팅(다중 페이지 애플리케이션) 또는 클라이언트 측 라우팅(단일 페이지 애플리케이션)을 사용할 수 있습니다. UI 구성에 혼합 분할을 사용하는 경우 페이지에서 마이크로 프론트엔드의 심층 라우팅 계층 구조를 지원하려면 클라이언트 측 라우팅이 필요합니다.

엣지 측 구성 및 서버 측 구성을 사용하는 애플리케이션은 서버 측 라우팅 또는 Amazon CloudFront를 사용하는 Lambda@Edge와 같은 엣지 컴퓨팅을 사용한 라우팅에 더 적합합니다.

마이크로 프런트엔드 간 통신

마이크로 프런트엔드 아키텍처의 경우 프런트엔드 구성 요소 간의 결합을 줄이는 것이 좋습니다. 결합을 줄이는 한 가지 방법은 동기식 함수 호출에서 비동기식 메시징으로 이동하는 것입니다.

브라우저 런타임 및 사용자 상호 작용은 비동기식입니다. 이벤트를 메시지를 통해 생산자와 소비자 간에 교환할 수 있습니다. 이벤트는 마이크로 프런트엔드 간 통신을 위한 잘 정의된 인터페이스를 제공합니다.

DDD 관행에 따라 마이크로 프론트엔드의 제한된 컨텍스트를 식별하는 경우 다음 단계는 경계를 넘어 통신해야 하는 이벤트를 식별하는 것입니다.

이벤트에 대한 메시징 메커니즘은 네이티브 DOM 이벤트(`CustomEvents`), JavaScript 이벤트 이미터 또는 플랫폼 팀이 제공하는 사후 대응 스트림 라이브러리일 수 있습니다. 마이크로 프론트엔드는 이벤트를 게시하고 제한된 컨텍스트와 관련된 이벤트를 구독합니다. 이 방법을 사용하면 게시자와 구독자가 서로를 인식할 필요가 없습니다. 계약은 이벤트 정의입니다. 이에 대한 시각적 표현은 이벤트 아키텍처를 사용한 경계 컨텍스트 다이어그램의 이벤트와 통신 섹션을 참조하세요.

교차 커팅 문제에 대한 종속성 관리

마이크로 프런트엔드와 같은 분산 아키텍처의 성공을 위해서는 신중한 종속성 관리가 중요합니다. 종속성 관리는 마이크로 프런트엔드 개발에서 가장 어려운 부분 중 하나입니다.

마이크로 프런트엔드 아키텍처에서 종속성 관리의 두 가지 중요한 측면은 대규모 코드 아티팩트를 클라이언트로 전송할 때 발생하는 성능 저하와 컴퓨팅 리소스의 오버헤드입니다. 조직은 분산 프런트엔드 아키텍처의 종속성을 유지하는 방법을 의무화하는 것이 가장 좋습니다.

종속성 유지 관리를 의무화하기 위한 세 가지 실행 가능한 전략은 가져오기 맵 및 모듈 페더레이션과 같은 웹 표준을 사용하는 것입니다. 다른 접근 방식은 분산 아키텍처의 기본 원칙을 위반하기 때문에 안티 패턴입니다.

가능하면 아무것도 공유하지 않습니다.

공유 없음 접근 방식은 독립 소프트웨어 아티팩트 간의 종속성을 전혀 공유해서는 안 되거나 최소한 통합 또는 런타임에 공유해서는 안 된다고 가정합니다. 즉, 두 개의 마이크로 프런트엔드가 동일한 라이브러리에 의존하는 경우 각 마이크로 프런트엔드는 빌드 시 라이브러리에서 베이크하여 별도로 배송해야 합니다. 또한 각 마이크로 프론트엔드는 라이브러리가 글로벌 네임스페이스 및 공유 리소스를 폴링하지 않는지 확인해야 합니다.

이로 인해 중복이 발생하지만 민첩성이 극대화된 의식적인 절충입니다. 런타임 종속성을 공유하지 않으면 팀은 솔루션 범위에 있고 인터페이스 계약을 위반하지 않는 한 유용하다고 생각되는 방식으로 소프트웨어를 발전시킬 수 있는 유연성을 극대화할 수 있습니다.

마이크로 프론트엔드가 공유 없음 원칙을 따르는 플랫폼에서는 마이크로 프론트엔드를 최대한 가볍게 유지하는 것이 중요합니다. 이를 위해서는 성능을 위해 마이크로 프런트엔드를 최적화하는 데 능숙하고 성실하며 개발자 경험을 위해 사용자 경험을 희생하지 않는 개발자가 필요합니다.

코드를 공유하는 경우

일부 코드를 공유하기로 결정할 때 라이브러리 또는 런타임 모듈로 공유할 수 있습니다. 예를 들어 프런트엔드 코어 팀은 CDNs을 통해 마이크로 프런트엔드 소비를 위한 라이브러리를 제공합니다. 비즈니스 가치 팀은 런타임에 라이브러리를 로드하거나 패키지 리포지토리를 사용하여 라이브러리를 게시할 수 있습니다. 마이크로 프론트엔드 팀은 하이브리드 프레임워크를 사용하는 모바일 애플리케이션과 마찬가지로 빌드 시 패키지 라이브러리의 특정 버전에 대해 개발할 수 있습니다.

세 번째 옵션은 프라이빗 패키지 레지스트리를 사용하여 공통 라이브러리의 빌드 타임 통합을 지원하는 것입니다. 이렇게 하면 라이브러리 계약의 주요 변경으로 인해 런타임에 오류가 발생할 위험이 줄어듭니다. 그러나이 보다 보수적인 접근 방식을 사용하려면 모든 마이크로 프런트엔드를 최신 라이브러리 버전과 동기화하기 위해 더 많은 거버넌스가 필요합니다.

페이지 로드 시간을 개선하기 위해 마이크로 프런트엔드는 Amazon CloudFront와 같은 CDN의 캐시된 청크에서 로드할 라이브러리 종속성을 외부화할 수 있습니다.

런타임 종속성을 관리하기 위해 마이크로 프론트엔드는 가져오기 맵(또는와 같은 라이브러리System.js)을 사용하여 런타임 시 각 모듈이에서 로드되는 위치를 지정할 수 있습니다. Webpack 모듈 페더레이션은 원격 모듈의 호스팅 버전을 가리키고 독립적인 마이크로 프론트엔드 간의 일반적인 종속성을 해결하는 또 다른 접근 방식입니다.

또 다른 접근 방식은 검색 엔드포인트에 대한 초기 요청으로 가져오기 맵의 동적 로드를 용이하게 하는 것입니다.

공유 상태

마이크로 프론트엔드의 결합을 줄이려면 모놀리식 아키텍처와 마찬가지로 동일한 뷰의 모든 마이크로 프론트엔드에서 액세스할 수 있는 글로벌 상태 관리를 피하는 것이 중요합니다. 예를 들어 모든 마이크로 프런트엔드에서 글로벌 Redux 스토어에 액세스할 수 있으면 결합이 증가합니다.

공유 상태를 제거하는 패턴은 마이크로 프런트엔드 내에 캡슐화하고 앞에서 설명한 비동기 메시지와 통신하는 것입니다.

절대적으로 필요한 경우 전역 상태에 잘 정의된 인터페이스를 도입하고 예기치 않은 동작을 방지하기 위해 읽기 전용 공유를 옵트인합니다.

- 세로 분할이 있는 경우 URL 구성 요소와 브라우저 스토리지를 사용하여 호스트 환경의 정보에 액세스할 수 있습니다.
- 혼합 분할이 있는 경우 DOM 표준 사용자 지정 이벤트 또는 이벤트 이미터 또는 양방향 스트림과 같은 JavaScript 라이브러리를 사용하여 마이크로 프런트엔드에 정보를 전달할 수도 있습니다.

마이크로 프론트엔드 간에 몇 가지 정보를 공유해야 하는 경우 마이크로 프론트엔드 경계를 다시 살펴보는 것이 좋습니다. 공유 필요성은 비즈니스 진화 또는 초기 설계에 따라 발생할 수 있습니다.

각 마이크로 프론트엔드가 세션 식별자를 사용하여 필요한 데이터를 가져오는 서버 측 세션을 사용할 수도 있습니다. 결합을 줄이려면 공유 상태를 제거하고 마이크로 프런트엔드별 세션 데이터를 별도로 유지하는 것이 중요합니다.

프레임워크 및 툴

Angular와 Next.js 같은 프론트엔드 프레임워크는 부족하지 않지만, 대부분은 마이크로 프론트엔드를 염두에 두고 만들어지지 않았습니다. 따라서 마이크로 프론트엔드 아키텍처의 문제를 해결할 수 있는 메커니즘이 없는 경우가 있습니다.

일반 프레임워크 고려 사항

이 가이드는 개별 프레임워크를 추천하거나 비교하는 것을 목표로 하지 않습니다. 동일한 웹 애플리케이션 페이지에서 여러 마이크로 프론트엔드가 실행되는 경우가 많기 때문에 로딩 및 런타임 성능이 주요 관심사입니다. 오버헤드가 최대한 적은 프레임워크를 선택하는 것이 중요합니다.

프레임워크는 렌더링 레이어에 따라 구분됩니다.

- 클라이언트 사이드 렌더링 (CSR)
- 서버 측 렌더링(SSR)

프런트엔드 아키텍처에는 정적 사이트 생성 (SSG) 과 같은 다른 기능이 포함됩니다. 하지만 SSG는 한 번만 수행됩니다. 마이크로 프론트엔드는 주로 런타임에 구성되므로 CSR과 SSR이 주요 옵션입니다.

클라이언트 사이드 렌더링

CSR의 경우 널리 사용되는 두 가지 옵션이 있습니다.

- 단일 SPA 프레임워크
- 모듈 페더레이션

단일 SPA는 마이크로 프론트엔드를 구성하기 위한 간단한 선택입니다. 동일한 페이지에 여러 마이크로 프론트엔드를 구성하고 종속성 충돌을 방지하는 등 마이크로 프론트엔드 아키텍처에서 가장 흔히 발생하는 문제를 해결합니다.

모듈 페더레이션은 webpack 5에서 제공하는 플러그인으로 시작되었으며, 다양한 아티팩트 간의 종속성 관리를 포함하여 마이크로 프론트엔드 아키텍처의 대부분의 문제를 해결합니다. 모듈 페더레이션 2.0은 기본적으로 Rspack, webpack, esbuild와 함께 작동하며 현재는 함께 작동합니다. JavaScript

프레임워크를 전혀 사용하지 않는 것을 고려해 보세요. caniuse.com에 따르면 전체 시장 점유율이 98% 에 달하는 최신 브라우저는 사용자 지정 요소와 같은 기능을 기본적으로 제공하며 마이크로 프론

트엔드 애플리케이션에 적합합니다. 필요한 경우 이벤트 전파, 국제화 또는 기타 특정 문제를 해결하기 위해 사용자 정의 요소를 간단한 라이브러리와 결합하세요.

서버 사이드 렌더링

SSR 측에서는 두 가지 주요 옵션이 더 복잡합니다.

- Next.js 같은 기존 프레임워크를 수용하고 모듈 페더레이션을 사용하는 마이크로 프론트엔드 원칙을 적용하세요.
- over-the-wire HTML-를 사용하여 마이크로 프론트엔드를 나타내는 HTML 프래그먼트를 교환하고 런타임에 템플릿 내에서 이러한 프래그먼트를 구성하십시오. 이러한 접근 방식의 예로는 Podium이 있습니다.

API 통합 – 프런트엔드용 백엔드

프런트엔드 백엔드(BFF) 패턴은 일반적으로 마이크로서비스 환경에서 사용됩니다. 마이크로 프론트엔드의 맥락에서 BFF는 마이크로 프론트엔드에 속하는 서버 측 서비스입니다. 모든 마이크로 프론트엔드에 BFF가 있어야 하는 것은 아닙니다. 그러나 BFF를 사용하는 경우 동일한 경계 컨텍스트 내에서 실행되어야 하며 다른 경계 컨텍스트 간에 공유되지 않아야 합니다.

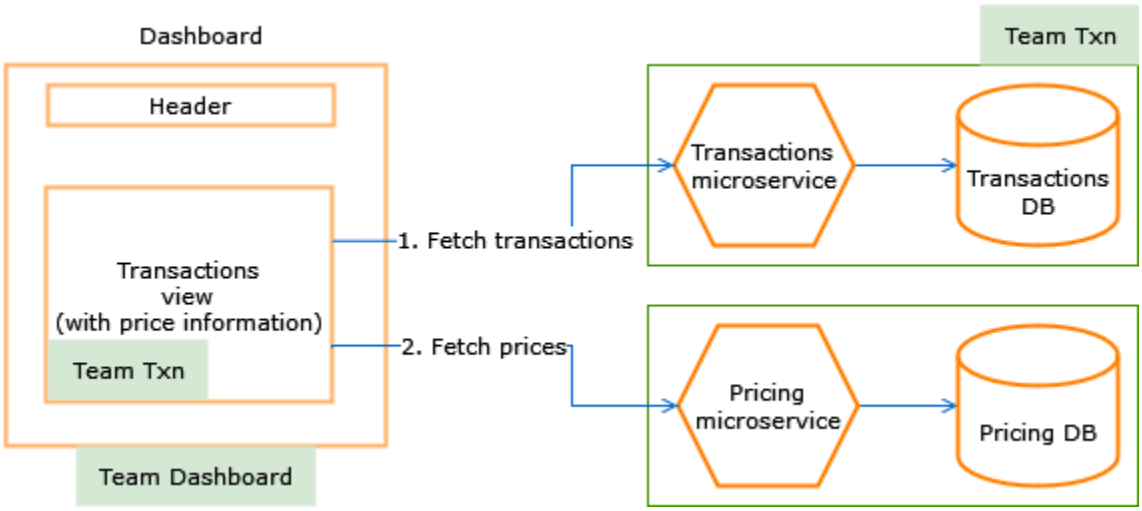
기존 서비스와 달리 BFF는 도메인 모델을 따르지 않습니다. 대신 마이크로 프런트엔드가 클라이언트에 도달하기 전에 데이터를 사전 처리하는 API 계층입니다. 이 기능이 유용한 영역은 다음과 같습니다.

- 프라이빗 APIs에 대한 권한 부여
- 다양한 소스의 데이터 집계
- 네트워크 로드를 줄이고 클라이언트의 데이터 소비를 용이하게 하기 위한 데이터 변환

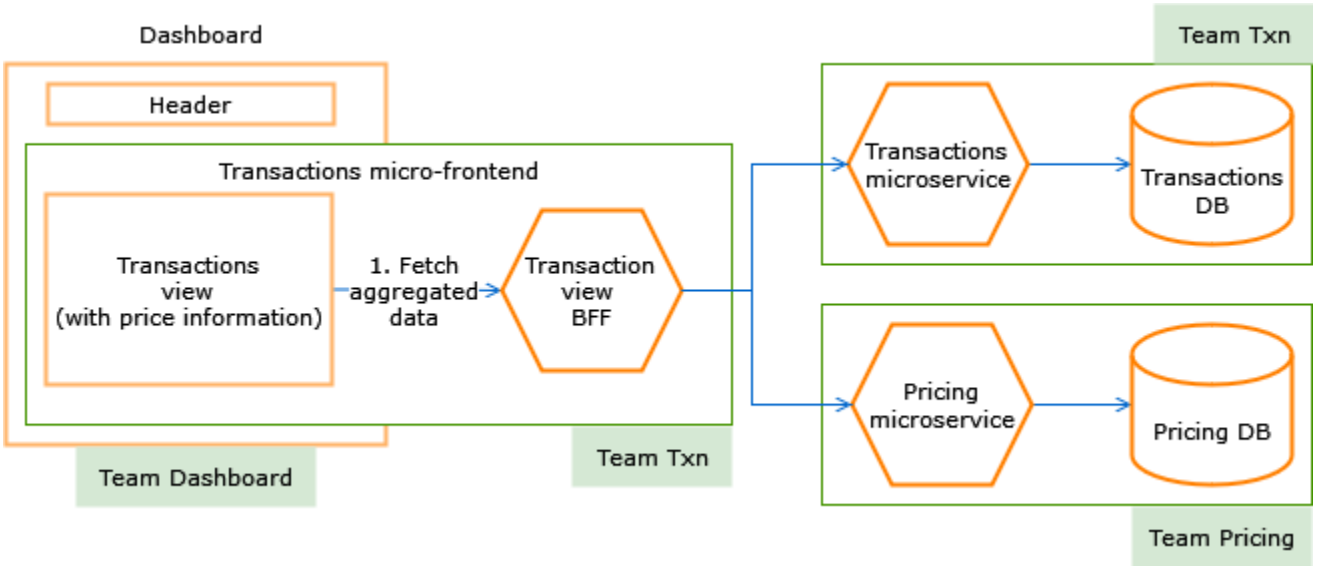
따라서 BFF는 도메인 서비스 계층이 아닌 마이크로 프런트엔드가 소유합니다. BFFs 다음을 사용하여 배포할 수 있습니다.

- AWS AppSync GraphQL APIs
- AWS Lambda 함수 세트
- Amazon ECS, Amazon EKS 또는 AWS AppRunner에서 실행되는 컨테이너

다음 다이어그램은 BFF 패턴이 없으면 마이크로 프론트엔드가 개별 마이크로서비스 API 엔드포인트에 연결하여 데이터를 가져오고 집계해야 함을 보여줍니다.



대신 다음 다이어그램의 BFF 패턴을 사용하면 마이크로 프론트엔드가 자체 백엔드와 통신하고 집계된 데이터를 가져올 수 있습니다.



팀은 채팅을 줄여 백엔드 상호 작용을 최적화하기 위한 요구 사항과 함께 모바일, 웹 또는 특정 뷰와 같은 다양한 채널에 대한 BFFs를 개발할 수 있습니다.

스타일 및 CSS

계단식 스타일 시트(CSS)는 텍스트 및 객체의 하드 코딩 형식 대신 문서 프레젠테이션을 중앙에서 결정하기 위한 언어입니다. 언어의 계단식 기능은 상속을 사용하여 스타일 간의 우선순위를 제어하도록 설계되었습니다. 마이크로 프런트엔드에서 작업하고 종속성을 관리하기 위한 전략을 생성하면 언어의 계단식 기능이 어려울 수 있습니다.

예를 들어 두 개의 마이크로 프론트엔드가 동일한 페이지에 공존하며, 각 마이크로 프론트엔드는 body HTML 요소에 대한 자체 스타일을 정의합니다. 각가 자체 CSS 파일을 가져와 `style` 태그를 사용하여 DOM에 연결하는 경우 CSS 파일은 공통 HTML 요소, 클래스 이름 또는 요소 IDs. 스타일 관리를 위해 선택하는 종속성 전략에 따라 이러한 문제를 해결할 수 있는 다양한 전략이 있습니다.

현재 성능, 일관성 및 개발자 경험의 균형을 맞추는 가장 인기 있는 접근 방식은 설계 시스템을 개발하고 유지하는 것입니다.

디자인 시스템 – 공유 접근 방식

이 접근 방식은 시스템을 사용하여 적절한 경우 스타일을 공유하는 동시에 간헐적인 분산을 지원하여 일관성, 성능 및 개발자 경험의 균형을 맞춥니다. 설계 시스템은 명확한 표준에 따라 재사용 가능한 구성 요소의 모음입니다. 설계 시스템 개발은 일반적으로 여러 팀의 의견과 기여가 있는 한 팀이 주도합니다. 실제로 설계 시스템은 JavaScript 라이브러리로 내보낼 수 있는 하위 수준 요소를 공유하는 방법입니다. 마이크로 프론트엔드 개발자는 라이브러리를 종속성으로 사용하여 미리 만들어진 사용 가능한 리소스를 구성하여 간단한 인터페이스를 구축하고 새 인터페이스를 만들기 위한 출발점으로 사용할 수 있습니다.

양식이 필요한 마이크로 프런트엔드의 예를 생각해 보세요. 일반적인 개발자 경험은 디자인 시스템에서 사용할 수 있는 미리 만들어진 구성 요소를 사용하여 텍스트 상자, 버튼, 드롭다운 목록 및 기타 UI 요소를 구성하는 것으로 구성됩니다. 개발자는 실제 구성 요소에 대한 스타일을 작성할 필요가 없으며 함께 보이는 방식에 대해서만 작성하면 됩니다. 빌드 및 릴리스할 시스템은 Webpack Module Federation 또는 유사한 접근 방식을 사용하여 설계 시스템을 외부 종속성으로 선언할 수 있으므로 설계 시스템을 포함하지 않고 양식의 로직이 패키징됩니다.

그런 다음 여러 마이크로 프런트엔드가 공유된 문제를 처리하기 위해 동일한 작업을 수행할 수 있습니다. 팀이 여러 마이크로 프런트엔드 간에 공유할 수 있는 새 구성 요소를 개발하면 성숙도에 도달한 후 해당 구성 요소가 설계 시스템에 추가됩니다.

설계 시스템 접근 방식의 주요 장점은 높은 수준의 일관성입니다. 마이크로 프론트엔드는 스타일을 쓰고 때때로 디자인 시스템에서 스타일을 재정의할 수 있지만, 거의 필요하지 않습니다. 기본 하위 수준

요소는 자주 변경되지 않으며 기본적으로 확장 가능한 기본 기능을 제공합니다. 또 다른 장점은 성능입니다. 빌드 및 릴리스를 위한 좋은 전략을 사용하면 애플리케이션 쉘에서 계측되는 최소한의 공유 번들을 생성할 수 있습니다. 네트워크 대역폭 측면에서 설치 공간을 최소화하면서 여러 마이크로 프런트엔드 특정 번들이 온디맨드로 비동기식으로 로드되는 경우 훨씬 더 개선할 수 있습니다. 마지막으로 개발자 경험은 휠을 재창조하지 않고도 풍부한 인터페이스를 구축하는 데 집중할 수 있기 때문에 이상적입니다(예: 버튼을 페이지에 추가해야 할 때마다 JavaScript 및 CSS 작성).

단점은 모든 종류의 설계 시스템이 종속성이므로 유지 관리 및 업데이트해야 한다는 것입니다. 여러 마이크로 프론트엔드에 새 버전의 공유 종속성이 필요한 경우 다음 중 하나를 사용할 수 있습니다.

- 충돌 없이 공유 종속성의 여러 버전을 가끔 가져올 수 있는 오케스트레이션 메커니즘
- 새 버전을 사용하도록 모든 종속 항목을 이동하는 공유 전략

예를 들어 모든 마이크로 프론트엔드가 설계 시스템의 버전 3.0에 의존하고 공유 방식으로 사용할 3.1이라는 새 버전이 있는 경우 모든 마이크로 프론트엔드가 위험을 최소화하면서 마이그레이션할 수 있도록 기능 플래그를 구현할 수 있습니다. 자세한 내용은 기능 플래그 섹션을 참조하세요. 또 다른 잠재적 단점은 설계 시스템이 일반적으로 스타일 그 이상을 해결한다는 것입니다. 여기에는 JavaScript 사례 및 도구도 포함됩니다. 이러한 측면은 토론과 협업을 통해 합의에 도달해야 합니다.

설계 시스템을 구현하는 것은 좋은 장기 투자입니다. 이는 널리 사용되는 접근 방식이며 복잡한 프런트엔드 아키텍처를 작업하는 모든 사용자가 고려해야 합니다. 일반적으로 프런트엔드 엔지니어와 제품 및 설계 팀이 협업하고 상호 작용 메커니즘을 정의해야 합니다. 원하는 상태에 도달할 시간을 예약하는 것이 중요합니다. 또한 사람들이 장기적으로 신뢰할 수 있고 잘 유지 관리되며 성과를 낼 수 있도록 경영진의 후원을 받는 것이 중요합니다.

완전히 캡슐화된 CSS – 아무것도 공유하지 않는 접근 방식

각 마이크로 프론트엔드는 규칙과 도구를 사용하여 CSS의 계단식 기능을 극복합니다. 예를 들어 각 요소의 스타일이 항상 요소의 ID 대신 클래스 이름과 연결되고 클래스 이름이 항상 고유하도록 하는 것입니다. 이렇게 하면 모든 것이 개별 마이크로 프론트엔드로 범위가 지정되고 원치 않는 충돌의 위험이 최소화됩니다. 애플리케이션 쉘은 일반적으로 마이크로 프런트엔드 스타일을 DOM에 로드한 후 로드하는 역할을 하지만 일부 도구는 JavaScript를 사용하여 스타일을 번들링합니다.

아무것도 공유하지 않으면 마이크로 프론트엔드 간에 충돌이 발생할 위험이 줄어듭니다. 또 다른 장점은 개발자의 경험입니다. 각 마이크로 프론트엔드는 다른 마이크로 프론트엔드와 아무것도 공유하지 않습니다. 독립 실행 및 테스트는 더 간단하고 빠릅니다.

공유 없음 접근 방식의 주요 단점은 일관성이 부족할 수 있다는 것입니다. 일관성을 평가하기 위한 시스템이 없습니다. 공유된 내용을 복제하는 것이 목표라고 해도 릴리스 및 협업 속도의 균형을 맞출 때는 어려워집니다. 일반적인 완화 방법은 일관성을 측정하는 도구를 만드는 것입니다. 예를 들어 헤드리스 브라우저가 있는 페이지에서 렌더링된 여러 마이크로 프런트엔드의 자동 스크린샷을 생성하는 시스템을 만들 수 있습니다. 그런 다음 릴리스 전에 스크린샷을 수동으로 검토할 수 있습니다. 하지만 이를 위해서는 원칙과 거버넌스가 필요합니다. 자세한 내용은 정렬을 통한 밸런싱 자율성 섹션을 참조하세요.

사용 사례에 따라 또 다른 잠재적 단점은 성능입니다. 모든 마이크로 프론트엔드에서 다량의 스타일을 사용하는 경우 고객은 많은 중복 코드를 다운로드해야 합니다. 이는 사용자 경험에 부정적인 영향을 미칩니다.

이 공유 없음 접근 방식은 소수의 팀만 관련된 마이크로 프런트엔드 아키텍처 또는 낮은 일관성을 허용할 수 있는 마이크로 프런트엔드에만 고려되어야 합니다. 조직이 설계 시스템에서 작업하는 동안 자연스러운 초기 단계일 수도 있습니다.

공유 글로벌 CSS – 전체 공유 접근 방식

이 접근 방식을 사용하면 스타일과 관련된 모든 코드가 중앙 리포지토리에 저장되어 기고자가 CSS 파일에서 작업하거나 Sass와 같은 프리프로세서를 사용하여 모든 마이크로 프론트엔드에 대해 CSS를 작성합니다. 변경이 이루어지면 빌드 시스템은 CDN에서 호스팅되고 애플리케이션 쉘에 의해 각 마이크로 프런트엔드에 포함될 수 있는 단일 CSS 번들을 생성합니다. 마이크로 프론트엔드 개발자는 로컬 호스팅 애플리케이션 셸을 통해 코드를 실행하여 애플리케이션을 설계하고 구축할 수 있습니다.

마이크로 프런트엔드 간의 충돌 위험을 줄이는 명백한 이점 외에도이 접근 방식의 장점은 일관성과 성능입니다. 그러나 스타일을 마크업 및 로직과 분리하면 개발자가 스타일 사용 방법, 스타일 진화 방법, 스타일 사용 중단 방법을 이해하기가 더 어려워집니다. 예를 들어 기존 클래스와 속성 편집의 결과에 대해 알아보는 것보다 새 클래스 이름을 도입하는 것이 더 빠를 수 있습니다. 새 클래스 이름을 생성할 때의 단점은 성능에 영향을 미치는 번들 크기 증가 및 사용자 경험에 불일치가 도입될 가능성입니다.

공유 글로벌 CSS는 monolith-to-micro-frontends 마이그레이션의 출발점이 될 수 있지만 둘 이상의 팀이 함께 협업하는 마이크로 프론트엔드 아키텍처에는 거의 유용하지 않습니다. 설계 시스템을 개발하는 동안 최대한 빨리 설계 시스템에 투자하고 공유 없는 접근 방식을 구현하는 것이 좋습니다.

조직 및 작업 방법

모든 아키텍처 전략과 마찬가지로 마이크로 프론트엔드는 조직이 구현하기로 선택한 기술을 훨씬 넘어서 영향을 미칩니다. 마이크로 프론트엔드 애플리케이션 구축 결정은 비즈니스, 제품, 조직, 운영 및 문화(예: 팀 권한 부여 및 의사 결정 분산)와 일치해야 합니다. 그 대가로 이러한 유형의 마이크로 프론트엔드 아키텍처는 독립 팀 간의 통신 오버헤드를 크게 줄이기 때문에 진정으로 민첩한 제품 기반 개발을 지원합니다.

애자일 개발

애자일 소프트웨어 개발에 대한 아이디어는 최근 몇 년 동안 매우 보편적이 되어 거의 모든 조직이 애자일이라고 주장합니다. 애자일의 결정적 정의는이 전략의 범위를 벗어나지만 마이크로 프런트엔드 개발과 관련된 주요 요소를 검토하는 것이 좋습니다.

애자일 패러다임의 기반은 애자일 매니페스토(2001)입니다. 애자일 매니페스토는 네 가지 주요 원칙(예: "프로세스 및 도구에 대한 개인 및 상호 작용")과 12가지 원칙을 게시했습니다. 스크럼 및 스케일링된 애자일 프레임워크(SAFe)와 같은 프로세스 프레임워크는 애자일 매니페스토를 중심으로 등장했으며 일상적인 관행으로 나아가고 있습니다. 그러나 그 이면의 철학은 대부분 잘못 이해되거나 무시됩니다.

마이크로 프런트엔드 아키텍처의 맥락에서 다음과 같은 애자일 원칙을 수용하는 것이 중요합니다.

- “더 짧은 기간을 선호하면서 몇 주에서 두 달로 작업 소프트웨어를 자주 제공하세요.”

  이 원칙은 증분 방식으로 작업하고 소프트웨어를 가능한 한 자주 정기적으로 프로덕션에 제공하는 것이 얼마나 중요한지 강조합니다. 기술적 관점에서 이는 지속적 통합 및 지속적 전달(CI/CD)을 의미합니다. CI/CD에서 구축, 테스트 및 배포를 위한 도구와 프로세스는 각 소프트웨어 프로젝트의 중요한 부분입니다. 또한 원칙은 런타임 인프라와 운영 책임을 팀이 소유해야 함을 의미합니다. 이러한 소유권은 독립 하위 시스템이 인프라 및 운영에 대한 요구 사항이 크게 다를 수 있는 분산 시스템에서 특히 중요합니다.
- “의미 있는 개인을 중심으로 프로젝트를 구축합니다. 필요한 환경과 지원을 제공하고 작업을 완료할 수 있도록 신뢰합니다.”

  “최고의 아키텍처, 요구 사항 및 설계는 자체 조직 팀에서 나옵니다.”

  이 두 원칙 모두 소유권, 독립성 및 end-to-end 책임의 이점을 강조합니다. 마이크로 프론트엔드 아키텍처는 팀이 실제로 마이크로 프론트엔드를 소유할 때(그리고 소유할 때만) 성공할 수 있습니다. 개념부터의 설계 및 구현, 제공 및 운영에 이르기까지 End-to-end 책임을 통해 팀은 실제로 소유권을

행사할 수 있습니다. 팀이 전략적 방향에 대한 자율성을 가지려면 기술적으로나 조직적으로나 이러한 독립성이 필요합니다. 폭포 개발 모델을 사용하는 중앙 집중식 조직에서는 마이크로 프런트엔드 플랫폼을 사용하지 않는 것이 좋습니다.

팀 구성 및 크기

소프트웨어 팀이 소유권을 행사하려면 조직이 부과하는 경계 내에서 팀이 제공하는 방식과 내용을 포함하여 자체적으로 관리해야 합니다.

효과적이 되려면 팀이 소프트웨어를 독립적으로 제공할 수 있어야 하며 소프트웨어를 제공하는 최선의 방법을 결정할 권한이 있어야 합니다. 외부 제품 관리자로부터 기능 요구 사항을 받거나 외부 디자이너로부터 UI 설계를 받는 팀은 이러한 항목의 계획에 관여하지 않고 자율적인 것으로 간주될 수 없습니다. 기능은 기존 계약 또는 기능을 위반할 수 있습니다. 이러한 위반에는 추가 논의 및 협상이 필요하며, 전달이 지연되고 팀 간에 불필요한 충돌이 발생할 위험이 있습니다.

동시에 팀이 너무 커져서는 안 됩니다. 대규모 팀에는 더 많은 리소스가 있고 개별 결근을 수용할 수 있지만 커뮤니케이션 복잡성은 각 신규 멤버에 따라 기하급수적으로 증가합니다. 보편적으로 유효한 최대 팀 크기를 설명할 수 없습니다. 프로젝트에 필요한 사람의 수는 팀 성숙도, 기술적 복잡성, 혁신 속도, 인프라와 같은 요인에 따라 달라집니다. 예를 들어 Amazon은 피자 2개 규칙을 따릅니다. 피자 2개에 공급하기에 너무 큰 팀은 더 작은 팀으로 분할해야 합니다. 이는 문제가 될 수 있습니다. 분할은 자연 경계를 따라 이루어져야 하며 각 팀에 작업에 대한 자율성과 소유권을 부여해야 합니다.

DevOps 문화

DevOps는 개발 수명 주기의 단계가 조직 및 기술 관점에서 긴밀하게 통합되는 소프트웨어 엔지니어링 관행을 말합니다. DevOps는 문화와 사고방식에 관한 것이지 역할과 도구에 관한 것이 아닙니다.

일반적으로 소프트웨어 조직에는 설계, 구현, 테스트, 배포 및 운영과 같은 전문가 팀이 있습니다. 팀이 작업을 마칠 때마다 프로젝트를 다음 팀에 인계합니다. 그러나 특수 팀의 사일로를 통해 소프트웨어를 제공하면 인계 중에 마찰이 발생합니다. 동시에 전문가가 좁은 초점으로 작업해야 하는 경우 이웃 도메인에 대한 지식이 부족하고 제품에 대한 체계적인 관점이 없습니다. 이러한 결함은 소프트웨어 제품의 낮은 일관성으로 이어질 수 있습니다.

예를 들어 소프트웨어 아키텍트가 다른 팀의 누군가가 구현할 솔루션을 설계하는 경우 구현의 고유한 측면(예: 종속성 불일치)을 간과할 수 있습니다. 그런 다음 개발자는 바로 가기(예: 몽키 패치)를 사용하거나 아키텍트와 개발 팀 간에 공식화된 back-and-forth 작업을 시작합니다. 이러한 프로세스 관리의 오버헤드로 인해 개발은 더 이상 민첩하지 않습니다(유연, 적응, 증분 및 비공식적).

DevOps라는 용어는 주로 문화와 관련이 있지만 실제로 DevOps를 가능하게 하는 기술과 프로세스를 의미합니다. DevOps는 CI/CD와 밀접한 관련이 있습니다. 개발자가 소프트웨어 증분 구현을 완료하면 Git과 같은 버전 제어 시스템에 이를 커밋합니다. 일반적으로 빌드 시스템은 소프트웨어를 빌드 및 통합하고 보다 통합되지 않은 중앙 집중식 프로세스에서 소프트웨어를 테스트 및 릴리스합니다. CI/CD를 사용하면 소프트웨어의 구축, 통합, 테스트 및 릴리스가 고유하고 자동화됩니다. 이 프로세스는 특정 프로젝트에 맞게 조정된 구성 파일을 통해 소프트웨어 프로젝트 자체의 일부인 것이 이상적입니다.

가능한 한 많은 단계가 자동화됩니다. 예를 들어, 거의 모든 유형의 테스트를 자동화할 수 있으므로 수동 테스트 관행을 줄여야 합니다. 프로젝트가 이러한 방식으로 설정되면 소프트웨어 제품에 대한 업데이트를 높은 신뢰도로 매일 여러 번 제공할 수 있습니다. DevOps를 지원하는 또 다른 기술은 코드형 인프라(IaC)입니다.

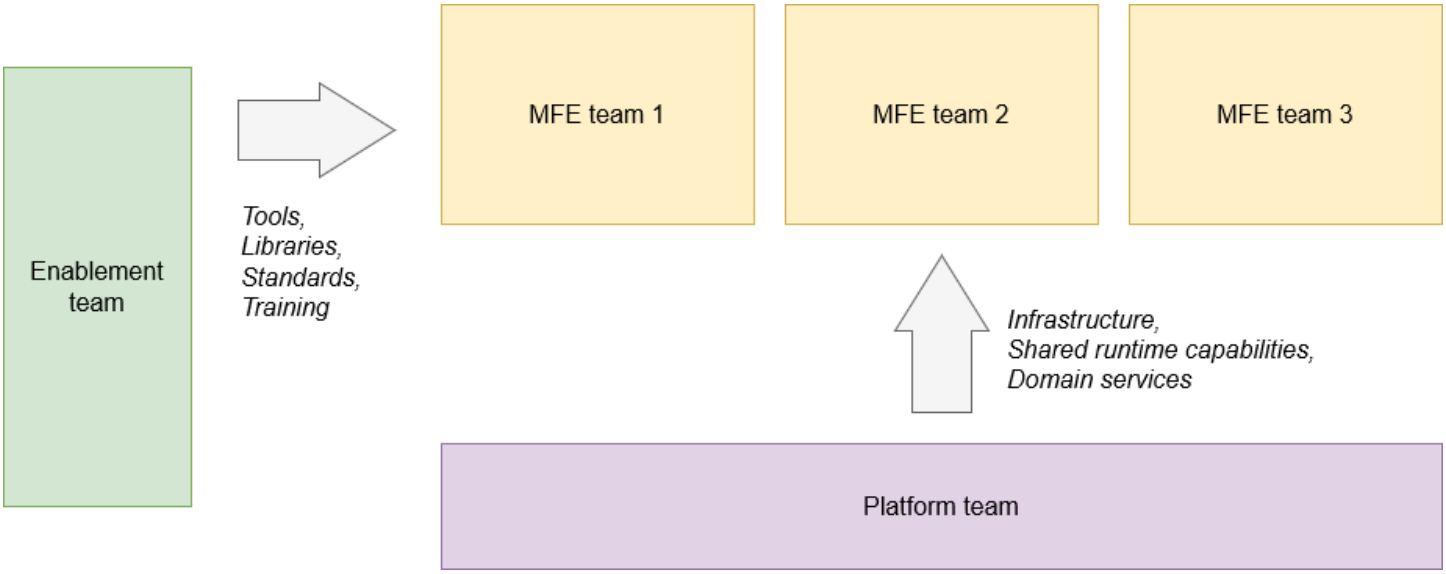
일반적으로 IT 인프라를 설정하고 유지 관리하려면 하드웨어(데이터 센터에서 케이블 및 서버 설정) 및 운영 소프트웨어를 설치하고 유지 관리하는 수동 작업이 필요합니다. 이는 필요했지만 단점이 많았습니다. 설정이 시간이 많이 걸리고 오류가 발생하기 쉽습니다. 하드웨어가 과다 프로비저닝되거나 과소 프로비저닝되어 과도한 지출 또는 성능 저하가 발생하는 경우가 많습니다. IaC를 사용하면 클라우드 서비스를 자동으로 배포하고 업데이트할 수 있는 구성 파일을 통해 IT 시스템의 인프라 요구 사항을 설명할 수 있습니다.

이 모든 것이 마이크로 프런트엔드와 어떤 관련이 있나요? DevOps, CI/CD 및 IaC는 마이크로 프런트엔드 아키텍처를 보완하는 데 이상적입니다. 마이크로 프론트엔드의 이점은 빠르고 원활한 전송 프로세스에 의존합니다. DevOps 문화는 팀이 end-to-end 책임을 가진 소프트웨어 프로젝트를 소유하는 환경에서만 영향을 미칠 수 있습니다.

여러 팀에서 마이크로 프런트엔드 개발 오케스트레이션

여러 부서 간 팀에서 마이크로 프론트엔드 개발을 확장할 때 두 가지 문제가 나타납니다. 첫째, 팀은 패러다임에 대한 자체 해석을 개발하고, 프레임워크 및 라이브러리를 선택하고, 자체 도구 및 헬퍼 라이브러리를 생성하기 시작합니다. 둘째, 완전 자율 팀은 하위 수준 인프라 관리와 같은 일반적인 기능에 대한 책임을 져야 합니다. 따라서 다중 팀 마이크로 프런트엔드 조직에 활성화 팀과 플랫폼 팀이라는 두 개의 추가 팀을 도입하는 것이 좋습니다. 이러한 개념은 분산 시스템이 있는 최신 IT 조직에서 널리 채택되며 팀 토폴로지에 잘 설명되어 있습니다.

다음 다이어그램은 세 개의 마이크로 프런트엔드 팀에 도구, 라이브러리, 표준 및 테스트를 제공하는 지원 팀을 보여줍니다. 플랫폼 팀은 동일한 세 개의 마이크로 프런트엔드 팀에 인프라, 공유 런타임 기능 및 도메인 서비스를 제공합니다.



플랫폼 팀은 마이크로 프런트엔드 팀이 차별화되지 않은 과중한 작업으로부터 벗어날 수 있도록 지원합니다. 이 지원에는 컨테이너 런타임, CI/CD 파이프라인, 공동 작업 도구 및 모니터링과 같은 인프라 서비스가 포함될 수 있습니다. 그러나 플랫폼 팀을 설정해도 개발이 운영에서 분리되는 조직으로 이어져서는 안 됩니다. 그 반대의 경우도 마찬가지입니다. 플랫폼 팀은 엔지니어링 제품을 제공하고 마이크로 프런트엔드 팀은 플랫폼에서 서비스에 대한 소유권과 런타임 책임을 집니다.

지원 팀은 거버넌스에 집중하고 마이크로 프런트엔드 팀 간의 일관성을 보장하여 지원을 제공합니다. (플랫폼 팀은 이에 관여해서는 안 됩니다.) 지원 팀은 UI 라이브러리와 같은 공유 리소스를 유지 관리하며 프레임워크, 성능 예산 및 상호 운용성 규칙 선택과 같은 표준을 생성합니다. 동시에 새로운 팀 또는 팀원에게 거버넌스에서 정의한 표준 및 도구를 적용하는 교육을 제공합니다.

배포

마이크로 프론트엔드 팀에서 자율성의 북극성은 다른 마이크로 프론트엔드 팀과 독립적인 프로덕션 경로가 있는 자동화된 파이프라인을 확보하는 것입니다. 공유 없음 원칙을 따르는 팀은 독립적인 파이프라인을 구현할 수 있습니다. 라이브러리를 공유하거나 플랫폼 팀에 의존하는 팀은 배포 파이프라인에서 종속성을 관리하는 방법을 결정해야 합니다.

일반적으로 각 파이프라인은 다음을 수행합니다.

- 프런트엔드 자산을 빌드합니다.
- 사용을 위해 호스팅에 자산을 배포합니다.
- 새 버전을 고객에게 제공할 수 있도록 레지스트리 및 캐시가 업데이트되었는지 확인합니다.

실제 파이프라인 단계는 기술 스택과 페이지 구성 접근 방식에 따라 달라집니다.

클라이언트 측 구성의 경우, 이는 호스팅 버킷에 애플리케이션 번들을 업로드하고 CDN에서 캐싱을 통해 소비에 릴리스하는 것을 의미합니다. 서비스 작업자와 함께 브라우저 캐싱을 사용하는 애플리케이션은 서비스 작업자 캐시를 업데이트하는 방법도 구현해야 합니다.

서버 측 구성의 경우 이는 일반적으로 서버 구성 요소의 새 버전을 배포하고 새 버전을 검색할 수 있도록 마이크로 프런트엔드 레지스트리를 업데이트하는 것을 의미합니다. 블루/그린 또는 카나리 배포 패턴을 사용하여 새 버전을 점진적으로 배포할 수 있습니다.

지배구조

여러 페르소나는 일반적으로 마이크로 프런트엔드에서 작업하며, 각 페르소나는 공통 비즈니스 목표에 대해 서로 다른 제약 조건에서 작동합니다. 사람 간의 커뮤니케이션과 협업이 성공의 핵심이지만 지나치게 복잡한 프로세스를 과도하게 커뮤니케이션하고 구현하면 개발 주기가 느려집니다. 이로 인해 사기가 저하되고 품질 기준이 낮아집니다.

여러 팀을 사용하여 마이크로 프런트엔드를 구현하는 가장 성공적인 기업은 자율성과 조정의 균형을 맞추는 메커니즘을 생성합니다. 의사 결정자는 필요한 경우에만 로컬에서 조치를 취하고 계층적으로 에스컬레이션할 수 있습니다. 메커니즘에는 다음이 포함됩니다.

- API 계약
- 이벤트를 사용한 교차 상호 작용
- 정렬을 통한 자율성 균형 조정
- 기능 플래그
- 서비스 검색

API 계약

각 마이크로 프론트엔드는 의견, 로직 및 복잡성을 캡슐화할 수 있는 시스템입니다. 교차 절단 문제에는 일반적으로 다음이 포함됩니다.

- 시스템 설계 – 라이브러리로 배포된 UIs를 개발하기 위한 도구
- 구성 – 마이크로 프런트엔드가 애플리케이션 셸과 상호 작용하여 컨텍스트를 렌더링하고 상속해야 하는 방식
- 로직 처리 – 영구 상태를 처리하기 위한 APIs와의 상호 작용
- 다른 마이크로 프론트엔드와의 상호 작용 - 이벤트를 게시 및 소비하거나 한 마이크로 프론트엔드에서 다른 마이크로 프론트엔드로 이동하는 등의 시나리오

소비 및 문제 해결을 가속화하기 위해 마이크로 프런트엔드 종속성을 포함하여 이러한 인터페이스가 선언되고 문서화되는 방식을 표준화하는 데 투자하는 것이 일반적입니다. 사람이 큐레이션한 Wiki는 좋은 출발점입니다. 보다 확장 가능한 접근 방식은이 정보를 코드에 구조화된 메타데이터로 저장하는 것입니다. 그런 다음 자동화를 사용하여 과거 변경 사항을 추적하고 전체 텍스트 검색을 제공하여 사용을 위해 중앙 집중화할 수 있습니다.

마이크로 프론트엔드에 많은 팀이 포함되는 경우 팀 간에 조정하기 위한 전략이 필요합니다. 통합 방식으로 API 계약을 공유하는 것은 통신 오버헤드를 줄이고 개발자 경험을 개선하므로 필수가 됩니다.

OpenAPI는 통합 방식으로 APIs 인터페이스 및 계약을 정의하는 것을 지원하는 HTTP API용 사양 언어입니다. Amazon APIs를 사용하여 REST API를 구현할 수 있습니다. OpenAPI Amazon API Gateway 컨테이너 또는 가상 머신에서 호스팅할 수 있는 다양한 오픈 소스 프레임워크를 사용할 수도 있습니다. 중요한 이점은 OpenAPI가 일관된 형식으로 설명서를 자동으로 생성할 수 있으므로 여러 팀이 최소한의 초기 투자로 지식을 공유할 수 있다는 것입니다.

여러 팀이 마이크로 프런트엔드에서 작업할 때 종종 그룹을 형성합니다. 이러한 그룹에서는 사람들이 서로 만나서 배우면서 더 큰 그림에 대해 생각하고 기여할 수 있습니다. 이러한 이니셔티브는 일반적으로 소유권 경계를 정의 및 문서화하고, 교차 해결 문제를 논의하며, 일반적인 문제를 해결하기 위한 노력의 중복을 조기에 식별합니다.

이벤트를 사용한 교차 상호 작용

일부 시나리오에서는 상태 변경 또는 사용자 작업에 대응하기 위해 여러 마이크로 프런트엔드가 서로 상호 작용해야 할 수 있습니다. 예를 들어 페이지의 여러 마이크로 프런트엔드에는 축소 가능한 메뉴가 포함될 수 있습니다. 사용자가 버튼을 선택하면 메뉴가 나타납니다. 사용자가 다른 마이크로 프런트엔드 내에서 렌더링되는 다른 메뉴를 포함하여 다른 곳을 클릭하면 메뉴가 숨겨집니다.

기술적으로 Redux와 같은 공유 상태 라이브러리는 여러 마이크로 프런트엔드에서 사용하고 쉘에서 조정할 수 있습니다. 그러나이 경우 애플리케이션 간에 상당한 결합이 생성되어 테스트하기 어려운 코드가 생성되고 렌더링 중에 성능이 느려질 수 있습니다.

한 가지 일반적인 효과적인 접근 방식은 라이브러리로 배포되고, 애플리케이션 쉘에서 오케스트레이션되며, 여러 마이크로 프런트엔드에서 사용하는 이벤트 버스를 개발하는 것입니다. 이렇게 하면 각 마이크로 프론트엔드는 특정 이벤트를 비동기적으로 게시하고 수신하여 자체 내부 상태만을 기반으로 동작을 수행합니다. 그런 다음 여러 팀이 이벤트를 설명하고 사용자 경험 디자이너가 동의한 동작을 문서화하는 공유 Wiki 페이지를 유지할 수 있습니다.

이벤트 버스 예제의 구현에서 드롭다운 구성 요소는 공유 버스를 사용하여 페이로드`drop-down-open-menu`가 인 라는 이벤트를 게시합니다`{"id": "homepage-aboutus-button"}`. 구성 요소는 `drop-down-open-menu` 이벤트에 리스너를 추가하여 새 ID에 대해 이벤트가 실행되는 경우 드롭다운 구성 요소가 렌더링되어 접을 수 있는 섹션을 숨깁니다. 이러한 방식으로 마이크로 프런트엔드는 성능 향상과 캡슐화 개선으로 변화에 비동기적으로 대응할 수 있으므로 여러 팀이 동작을 더 쉽게 설계하고 테스트할 수 있습니다.

단순성과 유지 관리를 개선하려면 최신 브라우저에서 기본적으로 구현한 표준 APIs를 사용하는 것이 좋습니다. MDN 이벤트 참조는 클라이언트 측 렌더링 애플리케이션에서 이벤트를 사용하는 방법에 대한 정보를 제공합니다.

정렬을 통한 자율성 균형 조정

마이크로 프런트엔드 아키텍처는 팀 자율성에 크게 편향되어 있습니다. 그러나 문제 해결을 위한 유연성과 다양한 접근 방식을 지원할 수 있는 영역과 표준화가 필요한 영역을 구분하여 조정하는 것이 중요합니다. 고위 리더와 아키텍트는 이러한 영역을 조기에 식별하고 투자를 우선시하여 마이크로 프런트엔드의 보안, 성능, 운영 우수성 및 신뢰성의 균형을 맞춰야 합니다. 이 균형을 찾으려면 마이크로 프런트엔드 생성, 테스트, 릴리스 및 로깅, 모니터링 및 알림이 필요합니다.

마이크로 프런트엔드 생성

모든 팀이 최종 사용자 성능 측면에서 이점을 극대화하도록 강력하게 조율하는 것이 가장 좋습니다. 실제로 이는 어려울 수 있으며 더 많은 노력이 필요할 수 있습니다. 여러 팀이 공개적이고 투명한 토론을 통해 기여할 수 있는 몇 가지 서면 지침부터 시작하는 것이 좋습니다. 그런 다음 팀은 프로젝트를 스캐폴드하는 통합 방법을 제공하는 도구 생성을 지원하는 Cookiecutter 소프트웨어 패턴을 점진적으로 채택할 수 있습니다.

이 접근 방식을 사용하면 의견과 제약 조건으로 베이크할 수 있습니다. 단점은 이러한 도구의 생성 및 유지 관리에 상당한 투자가 필요하고 개발자 생산성에 영향을 주지 않고 블로커를 신속하게 처리해야 한다는 것입니다.

마이크로 프런트엔드End-to-end 테스트

단위 테스트는 소유자에게 맡길 수 있습니다. 초기에 전략을 구현하여 고유한 셸에서 실행되는 마이크로 프런트엔드를 교차 테스트하는 것이 좋습니다. 전략에는 프로덕션 릴리스 전후에 애플리케이션을 테스트하는 기능이 포함되어 있습니다. 중요한 기능을 수동으로 테스트하기 위해 기술 및 비기술 인력을 위한 프로세스와 설명서를 개발하는 것이 좋습니다.

변경으로 인해 기능 또는 비기능 고객 경험이 저하되지 않도록 하는 것이 중요합니다. 이상적인 전략은 주요 기능과 보안 및 성능과 같은 아키텍처 특성에 대한 자동화된 테스트에 점진적으로 투자하는 것입니다.

마이크로 프런트엔드 해제

각 팀은 코드를 배포하고, 의견을 수렴하고, 인프라를 자체적으로 구축할 수 있습니다. 이러한 시스템을 유지 관리하는 데 드는 복잡성 비용은 일반적으로 방지 효과가 있습니다. 대신 공유 도구로 적용할 수 있는 공유 전략을 구현하기 위해에 조기에 투자하는 것이 좋습니다.

선택한 CI/CD 플랫폼을 사용하여 템플릿을 개발합니다. 그런 다음 팀은 사전 승인된 템플릿과 공유 인프라를 사용하여 프로덕션에 대한 변경 사항을 릴리스할 수 있습니다. 초기 테스트 및 통합 기간 후에는 이러한 시스템에 중요한 업데이트가 거의 필요하지 않으므로이 개발 작업에 조기에 투자할 수 있습니다.

로깅 및 모니터링

각 팀은 운영 또는 분석 목적으로 추적하려는 다양한 비즈니스 및 시스템 지표를 가질 수 있습니다. 여기에도 Cookiecutter 소프트웨어 패턴을 적용할 수 있습니다. 이벤트 전송을 추상화하여 여러 마이크로프런트엔드가 사용할 수 있는 라이브러리로 사용할 수 있습니다. 유연성의 균형을 맞추고 자율성을 제공하려면 사용자 지정 지표를 로깅하고 사용자 지정 대시보드 또는 보고서를 생성하기 위한 도구를 개발합니다. 보고는 제품 소유자와의 긴밀한 협업을 촉진하고 최종 고객 피드백 루프를 줄입니다.

제공을 표준화하면 여러 팀이 협력하여 지표를 추적할 수 있습니다. 예를 들어 전자 상거래 웹 사이트는 "제품 세부 정보" 마이크로 프론트엔드에서 "카트" 마이크로 프론트엔드, "구매" 마이크로 프론트엔드까지의 사용자 여정을 추적하여 참여, 이탈 및 문제를 측정할 수 있습니다. 각 마이크로 프런트엔드가 단일 라이브러리를 사용하여 이벤트를 기록하는 경우이 데이터를 전체적으로 사용하고, 전체적으로 탐색하고, 통찰력 있는 추세를 식별할 수 있습니다.

알림

로깅 및 모니터링과 마찬가지로 유연성이 확보된 표준화를 통해 알림을 받을 수 있습니다. 팀마다 기능적 알림과 비기능적 알림에 다르게 반응할 수 있습니다. 그러나 모든 팀이 공유 플랫폼에서 수집 및 분석되는 지표를 기반으로 알림을 시작할 수 있는 통합 방법이 있는 경우 비즈니스는 팀 간 문제를 식별할 수 있습니다. 이 기능은 인시던트 관리 이벤트에서 유용합니다. 예를 들어 다음과 같은 방법으로 알림을 시작할 수 있습니다.

- 특정 브라우저 버전에서 JavaScript 클라이언트 측 예외 수 증가
- 지정된 임계값을 초과하여 크게 저하된 렌더링 시간
- 특정 API를 사용할 때 승격된 5xx 상태 코드 수

시스템의 성숙도에 따라 다음 표와 같이 인프라의 여러 부분에 대한 노력의 균형을 맞출 수 있습니다.

채택	연구 및 개발	상승	성숙도
마이크로 프런트엔드를 생성합니다.	학습 내용을 실험, 문서화 및 공유합니다.	새로운 마이크로 프런트엔드를 스캐폴드하기 위한 도구에 투자합니다. 채택을 복음화합니다.	스캐폴딩을 위한 통합 도구입니다. 도입을 위해 푸시합니다.
마이크로 프런트엔드 엔드 투 엔드를 테스트합니다.	모든 관련 마이크로 프런트엔드를 수동으로 테스트하기 위한 메커니즘을 구현합니다.	자동화된 보안 및 성능 테스트를 위한 도구에 투자합니다. 기능 플래그 및 서비스 검색을 조사합니다.	서비스 검색, 프로덕션 테스트 및 자동화된 end-to-end 테스트를 위한 도구를 통합합니다.
마이크로 프런트엔드를 릴리스합니다.	공유 CI/CD 인프라와 자동화된 다중 환경 릴리스에 투자합니다. 채택을 복음화합니다.	CI/CD 인프라용 통합 도구 수동 롤백 메커니즘을 구현합니다. 도입을 위해 푸시합니다.	시스템 및 비즈니스 지표와 알림을 기반으로 자동 롤백을 시작하는 메커니즘을 생성합니다.
마이크로 프런트엔드 성능을 관찰합니다.	시스템 및 비즈니스 이벤트를 일관되게 로깅하기 위해 공유 모니터링 인프라 및 라이브러리에 투자합니다.	모니터링 및 알림을 위한 도구를 통합합니다. 팀 간 대시보드를 구현하여 일반적인 상태를 모니터링하고 인시던트 관리를 개선합니다.	로깅 스키마를 표준화합니다. 비용에 최적화합니다. 복잡한 비즈니스 지표를 기반으로 알림을 구현합니다.

기능 플래그

기능 플래그는 마이크로 프런트엔드에서 구현하여 여러 환경에서 기능을 테스트하고 릴리스하는 조정을 용이하게 할 수 있습니다. 기능 플래그 기법은 부울 기반 저장소에서 결정을 중앙 집중화하고 이를 기반으로 동작을 유도하는 것으로 구성됩니다. 특정 시점까지 숨길 수 있는 변경 사항을 자동으로 전파하는 동시에 차단될 수 있는 새 기능에 대한 새 릴리스를 잠금 해제하여 팀 속도를 줄이는 데 자주 사용됩니다.

특정 날짜에 시작될 마이크로 프런트엔드 기능을 작업하는 팀의 예를 생각해 보세요. 기능은 준비되었지만 독립적으로 릴리스된 다른 마이크로 프런트엔드의 변경 사항과 함께 릴리스해야 합니다. 두 마이크로 프론트엔드의 릴리스를 차단하면 안티 패턴으로 간주되며 배포 시 위험이 증가합니다.

대신 팀은 렌더링 시간(공유 특성 플래그 API에 대한 HTTP 호출을 통해) 동안 둘 다 사용하는 부울 특성 플래그를 데이터베이스에 생성할 수 있습니다. 팀은 프로덕션으로 시작하기 전에 프로젝트 간 기능 및 비기능 요구 사항을 확인하기 `True` 위해 부울 값이 로 설정된 테스트 환경에서 변경 사항을 릴리스할 수도 있습니다.

기능 플래그 사용의 또 다른 예는 `QueryString` 파라미터를 통해 특정 값을 설정하거나 쿠키에 특정 테스트 문자열을 저장하여 플래그 값을 재정의하는 메커니즘을 구현하는 것입니다. 제품 소유자는 시작 날짜까지 다른 기능의 릴리스 또는 버그 수정을 차단하지 않고 기능을 반복할 수 있습니다. 지정된 날짜에 데이터베이스의 플래그 값을 변경하면 팀 간 조정 릴리스 없이 프로덕션 환경에서 변경 사항을 즉시 볼 수 있습니다. 기능이 릴리스되면 개발 팀은 코드를 정리하여 이전 동작을 제거합니다.

다른 사용 사례에는 컨텍스트 기반 기능 플래그 시스템 릴리스가 포함됩니다. 예를 들어 단일 웹 사이트가 여러 언어로 고객에게 서비스를 제공하는 경우 특정 국가의 방문자만 기능을 사용할 수 있습니다. 기능 플래그 시스템은 국가 컨텍스트를 보내는 소비자에 따라 달라질 수 있으며(예: `Accept-Language` HTTP 헤더 사용) 해당 컨텍스트에 따라 다른 동작이 있을 수 있습니다.

기능 플래그는 개발자와 제품 소유자 간의 협업을 촉진하기 위한 강력한 도구이지만 코드 베이스가 크게 저하되지 않도록 사람들의 주의를 기울입니다. 여러 기능에서 플래그를 활성 상태로 유지하면 문제 해결 시 복잡성이 증가하고, JavaScript 번들 크기가 증가하며, 궁극적으로 기술 부채가 누적될 수 있습니다. 일반적인 완화 활동은 다음과 같습니다.

- 플래그 뒤의 각 기능을 유닛 테스트하여 버그 발생 가능성을 줄입니다. 그러면 테스트를 실행하는 자동화된 CI/CD 파이프라인에 더 긴 피드백 루프가 발생할 수 있습니다.
- 코드 변경 중에 번들 크기 증가를 측정하는 도구 생성, 코드 검토 중에 완화할 수 있음

AWS 는 Amazon CloudFront 함수 또는 Lambda@Edge를 사용하여 엣지에서 A/B 테스트를 최적화하기 위한 다양한 솔루션을 제공합니다. 이러한 접근 방식은 가정을 주장하는 데 사용하는 솔루션 또는

기존 SaaS 제품을 통합하는 복잡성을 줄이는 데 도움이 됩니다. 자세한 내용은 A/B 테스트를 참조하세요.

서비스 검색

프런트엔드 검색 패턴은 마이크로 프런트엔드를 개발, 테스트 및 제공할 때 개발 경험을 개선합니다. 패턴은 마이크로 프런트엔드의 진입점을 설명하는 공유 가능한 구성을 사용합니다. 공유 가능한 구성에는 canary 릴리스를 사용하여 각 환경에서 안전하게 배포하는 데 사용되는 추가 메타데이터도 포함됩니다.

최신 프런트엔드 개발에는 개발 중에 모듈화를 지원하기 위해 다양한 도구와 라이브러리가 필요합니다. 일반적으로이 프로세스는 초기 로드(브라우저에서 앱이 열릴 때) 및 사용량(고객이 버튼 선택 또는 정보 삽입과 같은 작업을 수행할 때)을 포함하여 런타임 중에 네트워크 호출을 최소한으로 유지하기 위해 CDN에서 호스팅할 수 있는 개별 파일로 코드를 번들링하는 것으로 구성되었습니다.

번들 분할

마이크로 프론트엔드 아키텍처는 대규모 기능 세트를 개별적으로 번들링하여 생성된 매우 큰 번들로 인한 성능 문제를 해결합니다. 예를 들어, 매우 큰 전자 상거래 웹 사이트를 6MB JavaScript 파일로 번들링할 수 있습니다. 압축에도 불구하고 해당 파일의 크기는 앱을 로드하고 엣지 최적화 CDN에서 파일을 다운로드할 때 사용자의 경험에 부정적인 영향을 미칠 수 있습니다.

앱을 홈 페이지, 제품 세부 정보 및 장바구니 마이크로 프론트엔드로 분할하는 경우 번들링 메커니즘을 사용하여 개별 2MB 번들 3개를 생성할 수 있습니다. 이 변경으로 인해 사용자가 홈 페이지를 사용할 때 첫 번째 로드의 성능이 300% 향상될 수 있습니다. 제품 또는 장바구니 마이크로 프론트엔드 번들은 사용자가 항목에 대한 제품 페이지를 방문하여 구매하기로 결정한 경우에만 비동기식으로 로드됩니다.

이 접근 방식을 기반으로 많은 프레임워크와 라이브러리를 사용할 수 있으며 고객과 개발자 모두에게 이점이 있습니다. 코드의 종속성을 분리할 수 있는 비즈니스 경계를 식별하기 위해 다양한 비즈니스 기능을 여러 팀에 매핑할 수 있습니다. 분산 소유권은 독립성과 민첩성을 도입합니다.

빌드 패키지를 분할할 때 구성을 사용하여 마이크로 프런트엔드를 매핑하고 초기 로드 및 사후 로드 탐색을 위한 오케스트레이션을 구동할 수 있습니다. 그러면 빌드 시간이 아닌 런타임 중에 구성을 사용할 수 있습니다. 예를 들어 클라이언트 측 프런트엔드 코드 또는 서버 측 백엔드 코드는 API에 대한 초기 네트워크 호출을 수행하여 마이크로 프런트엔드 목록을 동적으로 가져올 수 있습니다. 또한 구성 및 통합에 필요한 메타데이터를 가져옵니다. 신뢰성과 성능을 위해 장애 조치 전략과 캐싱을 구성할 수 있습니다. 마이크로 프론트엔드를 매핑하면 쉘 앱에서 오케스트레이션한 이전에 배포된 마이크로 프론트엔드에서 마이크로 프론트엔드를 개별적으로 배포할 수 있습니다.

카나리아 릴리스

카나리아 릴리스는 마이크로 서비스를 배포하기 위해 잘 정립되고 널리 사용되는 패턴입니다. Canary는 릴리스의 대상 사용자를 여러 그룹으로 버킷팅하고 즉각적인 대체(블루/그린 배포라고도 함)와 달리 변경 사항을 점진적으로 릴리스합니다. 카나리아 릴리스 전략의 예로는 대상 사용자의 10%에 새 변경 사항을 롤아웃하고 1분마다 10%를 추가하는 것이 있으며, 총 기간은 10분으로 100%에 도달합니다.

카나리아 릴리스의 목표는 변경 사항에 대한 초기 피드백을 받아 시스템을 모니터링하여 문제의 영향을 줄이는 것입니다. 자동화가 적용되면 배포를 중지하거나 롤백을 시작할 수 있는 내부 시스템에서 비즈니스 또는 시스템 지표를 모니터링할 수 있습니다.

예를 들어 변경으로 인해 릴리스 후 처음 몇 분 내에 수익 손실 또는 성능 저하를 초래하는 버그가 발생할 수 있습니다. 자동 모니터링은 경보를 시작할 수 있습니다. 서비스 검색 패턴을 사용하면이 경보가 배포를 중지하고 즉시 롤백하여 사용자의 100%가 아닌 20%만 영향을 받을 수 있습니다. 비즈니스는 문제의 범위가 축소되어 이점을 얻을 수 있습니다.

DynamoDB를 스토리지로 사용하여 REST 관리자 API를 구현하는 아키텍처 예제는 GitHub의 AWS 에서 프런트엔드 서비스 검색을 참조하세요. 템플릿을 사용하여 AWS CloudFormation 아키텍처를 자체 CI/CD 파이프라인에 통합합니다. 솔루션에는 솔루션을 프런트엔드 애플리케이션과 통합하기 위한 REST 소비자 API가 포함되어 있습니다.

플랫폼 팀이 필요하세요?

일부 회사에는 마이크로 프론트엔드에서 작업하기 위해 다른 팀에서 채택한 코드, 인프라 및 프로세스를 소유하고 유지 관리하는 팀이 있습니다. 일반적인 책임은 다음과 같습니다.

- 마이크로 프론트엔드가 포함된 리포지토리와 함께 사용할 수 있는 CI/CD 파이프라인을 만들고 유지 관리합니다. 코드 변경 사항을 빌드 및 테스트하고 여러 환경에서 릴리스하세요.
- 공유 대시보드, 경고 메커니즘, 문제에 대응하는 시스템 등 가시성 관련 도구를 만들고 유지 관리하세요.
- 이벤트 처리, 공유 서비스 사용 및 타사 종속성을 위한 공유 라이브러리를 만들고 유지 관리하세요.
- 시스템의 성능, 보안, 안정성과 같은 비기능적 특성을 지속적으로 모니터링하는 도구를 만들고 유지 관리하세요.
- 설계 시스템을 만들고 유지 관리합니다.
- 마이크로 프론트엔드 시스템용 애플리케이션 셸을 생성, 유지 관리 및 지원합니다.

프로젝트 규모에 따라 다음 접근 방식 중 하나를 사용하여 이러한 책임을 관리할 수 있습니다.

- 공유 도구에 대한 작업만 담당하는 전담 플랫폼 팀을 구성하세요.
- 여러 팀의 구성원으로 구성된 그룹을 만드세요. 그룹 구성원은 마이크로 프론트엔드 작업과 공유 도구 작업 사이에서 시간을 나눕니다. 이 팀을 타이거 팀이라고도 합니다.

타이거 팀 접근 방식은 고객 중심을 유지하는 효과적인 방법이지만, 프로젝트가 견인력과 책임을 얻으면 타이거 팀이 플랫폼 팀으로 발전하는 경우가 많습니다. 플랫폼 팀과 타이거 팀 모두 마이크로 프론트엔드에서 일하는 가장 성공적인 회사가 이러한 팀을 구성하므로 다양한 배경과 기술을 가진 여러 사람이 기여할 수 있습니다. 팀원에는 백엔드 엔지니어, 프론트엔드 엔지니어, 사용자 경험 (UX) 디자이너, 기술 제품 관리자가 포함될 수 있습니다. 이러한 다양성으로 인해 사람들은 계속해서 건전한 토론에 참여하고 단순함을 염두에 두고 디자인해야 합니다.

다음 단계

이 가이드에서는 아키텍처 및 조직 패턴, 주요 의사 결정의 장단점, 마이크로 프론트엔드와 관련된 거버넌스 문제를 다루었습니다. 표에는 이 문서에서 논의된 관행의 장단점이 다음과 같은 측면에서 요약되어 있습니다.

- 자율성 – 각 마이크로 프론트엔드 팀이 독립적으로 구현을 발전시켜 최종 사용자에게 배포할 수 있는 능력.
- 일관성 - 각 마이크로 프론트엔드가 예상대로 작동하는 애플리케이션의 전반적인 경험. 일관성이 높다는 것은 마이크로 프론트엔드가 나머지 애플리케이션과 일관성을 유지하고 전체 애플리케이션의 사용자 경험에 해를 끼치지 않는다는 것을 의미합니다.
- 복잡성 – 마이크로 프론트엔드, 전체 애플리케이션 및 거버넌스 제어를 구현하고 테스트하는 데 필요한 인프라, 코드 및 노력의 양

연습	자율성	일관성	복잡성
모놀리식 애플리케이션 대신 마이크로 프론트엔드로 구축	높음	중간	높음

코드 공유 관행	자율성	일관성	복잡성
아무것도 공	높음	낮음	낮음

코드 공유 관행	자율성	일관성	복잡성
유하지 마세요			
크로스 컷팅 우려 공유하기	중간	높음	중간
비즈니스 로직 공유	낮음	높음	중간
빌드 시 라이브러리를 통해 공유	중간	높음	낮음
런타임 시 공유	높음	높음	높음

마이크로 프론트엔드 디스커버리 사례	자율성	일관성	복잡성
애플리케이션 빌드 중 구성	낮음	높음	낮음
서버측 검색	높음	높음	중간
클라이언트측(런타임) 검색	높음	높음	중간

구성 사례 보기	자율성	일관성	복잡성
서버측 구성	높음	중간	높음
엣지 사이드 컴포지션	중간	중간	높음

구성 사례 보기	자율성	일관성	복잡성
클라이언트 측 구성	높음	중간	중간

이 지침에 소개된 개념에 대해 자세히 알아보려면 리소스 섹션을 참조하십시오.

리소스

- 상황에 맞는 마이크로 프론트엔드
- 도메인 기반 설계
- EDA 비주얼
- 프론트엔드 디스커버리
- 프런트엔드 서비스 검색 켜짐 AWS
- 애자일 매니페스토
- MDN 이벤트 레퍼런스
- OpenAPI

기여자

이 안내서에 기여한 사람은 다음과 같습니다.

- 마테오 피구스, 수석 솔루션 아키텍트, AWS
- 알렉산더 귄쉐, 선임 솔루션 아키텍트, AWS
- 하룬 해스달, 선임 솔루션 아키텍트, AWS
- 영국 서버리스 솔루션 아키텍트 수석 고투 마켓 스페셜리스트 루카 메잘리라, AWS

문서 이력

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

변경 사항	설명	날짜
최초 게시	—	2024년 7월 12일

AWS 권장 가이드 용어집

다음은 AWS 권장 가이드에서 제공하는 전략, 가이드 및 패턴에서 일반적으로 사용되는 용어입니다. 용어집 항목을 제안하려면 용어집 끝에 있는 피드백 제공 링크를 사용하십시오.

숫자

7가지 전략

애플리케이션을 클라우드로 이전하기 위한 7가지 일반적인 마이그레이션 전략 이러한 전략은 Gartner가 2011년에 파악한 5가지 전략을 기반으로 하며 다음으로 구성됩니다.

- 리팩터링/리아키텍트 - 클라우드 네이티브 기능을 최대한 활용하여 애플리케이션을 이동하고 해당 아키텍처를 수정함으로써 민첩성, 성능 및 확장성을 개선합니다. 여기에는 일반적으로 운영 체제와 데이터베이스 이식이 포함됩니다. 예: 온프레미스 Oracle 데이터베이스를 Amazon Aurora PostgreSQL 호환 에디션으로 마이그레이션합니다.
- 리플랫포밍(리프트 앤드 리셰이프) - 애플리케이션을 클라우드로 이동하고 일정 수준의 최적화를 도입하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 AWS 클라우드의 Amazon Relational Database Service(Amazon RDS) for Oracle로 마이그레이션합니다.
- 재구매(드롭 앤드 숍) - 일반적으로 기존 라이선스에서 SaaS 모델로 전환하여 다른 제품으로 전환합니다. 예: 고객 관계 관리(CRM) 시스템을 Salesforce.com으로 마이그레이션합니다.
- 리호스팅(리프트 앤드 시프트) - 애플리케이션을 변경하지 않고 클라우드로 이동하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 AWS 클라우드클라우드의 EC2 인스턴스에 있는 Oracle로 마이그레이션합니다.
- 재배치(하이퍼바이저 수준의 리프트 앤 시프트) - 새 하드웨어를 구매하거나, 애플리케이션을 다시 작성하거나, 기존 운영을 수정하지 않고도 인프라를 클라우드로 이동합니다. 온프레미스 플랫폼에서 동일한 플랫폼의 클라우드 서비스로 서버를 마이그레이션합니다. 예: Microsoft Hyper-V 애플리케이션을 로 마이그레이션합니다 AWS.
- 유지(보관) - 소스 환경에 애플리케이션을 유지합니다. 대규모 리팩터링이 필요하고 해당 작업을 나중으로 연기하려는 애플리케이션과 비즈니스 차원에서 마이그레이션할 이유가 없어 유지하려는 레거시 애플리케이션이 여기에 포함될 수 있습니다.
- 사용 중지 - 소스 환경에서 더 이상 필요하지 않은 애플리케이션을 폐기하거나 제거합니다.

A

A2A(Agent-to-Agent)

작업 위임 및 상태 전송agent-to-agent 공동 작업을 위한 상태 저장 프로토콜입니다.

ABAC

속성 기반 액세스 제어를 참조하세요.

추상화된 서비스

관리형 서비스를 참조하세요.

ACID

원자성, 일관성, 격리성, 내구성을 참조하세요.

능동-능동 마이그레이션

양방향 복제 도구 또는 이중 쓰기 작업을 사용하여 소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되고, 두 데이터베이스 모두 마이그레이션 중 연결 애플리케이션의 트랜잭션을 처리하는 데이터베이스 마이그레이션 방법입니다. 이 방법은 일회성 전환이 필요한 대신 소규모의 제어된 배치로 마이그레이션을 지원합니다. 더 유연하지만 액티브 패시브 마이그레이션보다 더 많은 작업이 필요합니다.

능동-수동 마이그레이션

소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되지만 소스 데이터베이스만 연결 애플리케이션의 트랜잭션을 처리하고 데이터는 대상 데이터베이스로 복제되는 데이터베이스 마이그레이션 방법입니다. 대상 데이터베이스는 마이그레이션 중 어떤 트랜잭션도 허용하지 않습니다.

에이전트

목표를 달성하기 위한 도구를 사용하여 자율적으로 추론, 계획 및 조치를 취할 수 있는 AI 시스템입니다.

에이전트 운영

대규모 프로덕션 환경에서 AI 에이전트를 구축, 테스트, 배포 및 실행하기 위한 운영 사례입니다.

집계 함수

행 그룹에서 작동하고 그룹에 대한 단일 반환 값을 계산하는 SQL 함수입니다. 집계 함수의 예로 `SUM` 및 `MAX`가 있습니다.

AI

인공 지능을 참조하세요.

AIOps

인공 지능 운영을 참조하세요.

익명화

데이터세트에서 개인 정보를 영구적으로 삭제하는 프로세스입니다. 익명화는 개인 정보 보호에 도움이 될 수 있습니다. 익명화된 데이터는 더 이상 개인 데이터로 간주되지 않습니다.

안티 패턴

솔루션이 다른 솔루션보다 비생산적이거나 비효율적이거나 덜 효과적이어서 반복되는 문제에 자주 사용되는 솔루션입니다.

애플리케이션 제어

맬웨어로부터 시스템을 보호하기 위해 승인된 애플리케이션만 사용하도록 허용하는 보안 접근 방식입니다.

애플리케이션 포트폴리오

애플리케이션 구축 및 유지 관리 비용과 애플리케이션의 비즈니스 가치를 비롯하여 조직에서 사용하는 각 애플리케이션에 대한 세부 정보 모음입니다. 이 정보는 포트폴리오 탐색 및 분석 프로세스의 핵심이며 마이그레이션, 현대화 및 최적화할 애플리케이션을 식별하고 우선순위를 정하는 데 도움이 됩니다.

인공 지능

컴퓨터 기술을 사용하여 학습, 문제 해결, 패턴 인식 등 일반적으로 인간과 관련된 인지 기능을 수행하는 것을 전문으로 하는 컴퓨터 과학 분야입니다. 자세한 내용은 What is Artificial Intelligence?를 참조하십시오.

인공 지능 운영(AIOps)

기계 학습 기법을 사용하여 운영 문제를 해결하고, 운영 인시던트 및 사용자 개입을 줄이고, 서비스 품질을 높이는 프로세스입니다. AWS 마이그레이션 전략에서 AIOps가 사용되는 방법에 대한 자세한 내용은 운영 통합 가이드를 참조하십시오.

비대칭 암호화

한 쌍의 키, 즉 암호화를 위한 퍼블릭 키와 복호화를 위한 프라이빗 키를 사용하는 암호화 알고리즘입니다. 퍼블릭 키는 복호화에 사용되지 않으므로 공유할 수 있지만 프라이빗 키에 대한 액세스는 엄격히 제한되어야 합니다.

원자성, 일관성, 격리성, 내구성(ACID)

오류, 정전 또는 기타 문제가 발생한 경우에도 데이터베이스의 데이터 유효성과 운영 신뢰성을 보장하는 소프트웨어 속성 세트입니다.

ABAC(속성 기반 액세스 제어)

부서, 직무, 팀 이름 등의 사용자 속성을 기반으로 세분화된 권한을 생성하는 방식입니다. 자세한 내용은 AWS Identity and Access Management (IAM) 설명서의 용 ABAC AWS를 참조하세요.

신뢰할 수 있는 데이터 소스

가장 신뢰할 수 있는 정보 소스로 간주되는 기본 버전의 데이터를 저장하는 위치입니다. 익명화, 편집 또는 가명화와 같은 데이터 처리 또는 수정의 목적으로 신뢰할 수 있는 데이터 소스의 데이터를 다른 위치로 복사할 수 있습니다.

가용 영역

다른 가용 영역의 장애로부터 격리 AWS 리전 되고 동일한 리전의 다른 가용 영역에 저렴하고 지연 시간이 짧은 네트워크 연결을 제공하는 내의 고유한 위치입니다.

AWS 클라우드 채택 프레임워크(AWS CAF)

조직이 클라우드로 성공적으로 전환하기 위한 효율적이고 효과적인 계획을 개발하는 AWS 데 도움이 되는의 지침 및 모범 사례 프레임워크입니다. AWS CAF는 지침을 비즈니스, 사람, 거버넌스, 플랫폼, 보안 및 운영이라는 6가지 중점 영역으로 구성합니다. 비즈니스, 사람 및 거버넌스 관점은 비즈니스 기술과 프로세스에 초점을 맞추고, 플랫폼, 보안 및 운영 관점은 전문 기술과 프로세스에 중점을 둡니다. 예를 들어, 사람 관점은 인사(HR), 직원 배치 기능 및 인력 관리를 담당하는 이해관계자를 대상으로 합니다. 이러한 관점에서 AWS CAF는 성공적인 클라우드 채택을 위해 조직을 준비하는 데 도움이 되는 인력 개발, 교육 및 커뮤니케이션에 대한 지침을 제공합니다. 자세한 내용은 AWS CAF 웹사이트와 AWS CAF 백서를 참조하세요.

AWS 워크로드 검증 프레임워크(AWS WQF)

데이터베이스 마이그레이션 워크로드를 평가하고, 마이그레이션 전략을 권장하고, 작업 견적을 제공하는 도구입니다. AWS WQF는 AWS Schema Conversion Tool (AWS SCT)에 포함되어 있습니다. 데이터베이스 스키마 및 코드 객체, 애플리케이션 코드, 종속성 및 성능 특성을 분석하고 평가 보고서를 제공합니다.

B

악성 봇

개인 또는 조직을 방해하거나 해를 입히기 위한 봇입니다.

BCP

비즈니스 연속성 계획을 참조하세요.

동작 그래프

리소스 동작과 시간 경과에 따른 상호 작용에 대한 통합된 대화형 뷰입니다. Amazon Detective에서 동작 그래프를 사용하여 실패한 로그온 시도, 의심스러운 API 직접 호출 및 유사한 작업을 검사할 수 있습니다. 자세한 내용은 Detective 설명서의 Data in a behavior graph를 참조하십시오.

빅 엔디안 시스템

가장 중요한 바이트를 먼저 저장하는 시스템입니다. 엔디안도 참조하세요.

바이너리 분류

바이너리 결과(가능한 두 클래스 중 하나)를 예측하는 프로세스입니다. 예를 들어, ML 모델이 “이 이메일이 스팸인가요, 스팸이 아닌가요?”, '이 제품은 책인가요, 자동차인가요?' 등의 문제를 예측해야 할 수 있습니다.

블룸 필터

요소가 세트의 멤버인지 여부를 테스트하는 데 사용되는 메모리 효율성이 높은 확률론적 데이터 구조입니다.

블루/그린(Blue/Green) 배포

동일하지만 별개의 두 환경을 생성하는 배포 전략입니다. 하나의 환경(파란색)에서 현재 애플리케이션 버전을 실행하고 새 애플리케이션 버전은 다른 환경(녹색)에서 실행합니다. 이 전략을 사용하면 영향을 최소화하면서 신속하게 롤백할 수 있습니다.

bot

인터넷을 통해 자동화된 태스크를 실행하고 인적 활동이나 상호 작용을 시뮬레이션하는 소프트웨어 애플리케이션입니다. 인터넷에서 정보를 인덱싱하는 웹 크롤러와 같이 유용하거나 이로운 봇도 있습니다. 악성 봇이라고 하는 다른 일부 봇은 개인 또는 조직을 방해하거나 해를 입히기 위한 봇입니다.

봇넷

맬웨어에 감염되고 봇 허더 또는 봇 운영자와 같은 단일 당사자가 제어하는 봇 네트워크입니다. 봇넷은 봇의 규모와 봇의 영향 범위를 확대하는 가장 잘 알려진 메커니즘입니다.

브랜치

코드 리포지토리의 포함된 영역입니다. 리포지토리에 생성되는 첫 번째 브랜치가 기본 브랜치입니다. 기존 브랜치에서 새 브랜치를 생성한 다음 새 브랜치에서 기능을 개발하거나 버그를 수정할 수 있습니다. 기능을 구축하기 위해 생성하는 브랜치를 일반적으로 기능 브랜치라고 합니다. 기능을 출시할 준비가 되면 기능 브랜치를 기본 브랜치에 다시 병합합니다. 자세한 내용은 About branches(GitHub 설명서)를 참조하십시오.

긴급 액세스 권한

예외적인 상황에서 승인된 프로세스를 통해 사용자가 일반적으로 액세스할 권한이 없는에 액세스할 수 AWS 계정 있는 빠른 방법입니다. 자세한 내용은 AWS Well-Architected 지침의 Implement break-glass procedures 지표를 참조하세요.

브라운필드 전략

사용자 환경의 기존 인프라 시스템 아키텍처에 브라운필드 전략을 채택할 때는 현재 시스템 및 인프라의 제약 조건을 중심으로 아키텍처를 설계합니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 그린필드 전략을 혼합할 수 있습니다.

버퍼 캐시

가장 자주 액세스하는 데이터가 저장되는 메모리 영역입니다.

사업 역량

기업이 가치를 창출하기 위해 하는 일(예: 영업, 고객 서비스 또는 마케팅)입니다. 마이크로서비스 아키텍처 및 개발 결정은 비즈니스 역량에 따라 이루어질 수 있습니다. 자세한 내용은 백서의 AWS에서 컨테이너화된 마이크로서비스 실행의 비즈니스 역량 중심의 구성화 섹션을 참조하십시오.

비즈니스 연속성 계획(BCP)

대규모 마이그레이션과 같은 중단 이벤트가 운영에 미치는 잠재적 영향을 해결하고 비즈니스가 신속하게 운영을 재개할 수 있도록 지원하는 계획입니다.

C

CAF

AWS Cloud Adoption Framework를 참조하세요.

카나리 배포

최종 사용자에게 제공하는 느린 증분 릴리스 버전입니다. 확신이 들면 새 버전을 배포하고 현재 버전을 완전히 교체합니다.

CCoE

클라우드 혁신 센터를 참조하세요.

CDC

데이터 캡처 변경을 참조하세요.

변경 데이터 캡처(CDC)

데이터베이스 테이블과 같은 데이터 소스의 변경 내용을 추적하고 변경 사항에 대한 메타데이터를 기록하는 프로세스입니다. 대상 시스템의 변경 내용을 감사하거나 복제하여 동기화를 유지하는 등의 다양한 용도로 CDC를 사용할 수 있습니다.

카오스 엔지니어링

시스템의 복원력을 테스트하기 위해 의도적으로 장애나 중단 이벤트를 도입합니다. AWS Fault Injection Service (AWS FIS)를 사용하여 AWS 워크로드에 스트레스를 주고 응답을 평가하는 실험을 수행할 수 있습니다.

CI/CD

지속적 통합 및 지속적 전송을 참조하세요.

분류

예측을 생성하는 데 도움이 되는 분류 프로세스입니다. 분류 문제에 대한 ML 모델은 이산 값을 예측합니다. 이산 값은 항상 서로 다릅니다. 예를 들어, 모델이 이미지에 자동차가 있는지 여부를 평가해야 할 수 있습니다.

시민 개발자

전문 기술 없이 노코드/로우코드 플랫폼을 사용하여 AI 애플리케이션을 생성하는 비즈니스 사용자입니다.

클라이언트측 암호화

대상이 데이터를 AWS 서비스 수신하기 전에 로컬에서 데이터를 암호화합니다.

클라우드 혁신 센터(CCoE)

클라우드 모범 사례 개발, 리소스 동원, 마이그레이션 타임라인 설정, 대규모 혁신을 통한 조직 선도 등 조직 전체에서 클라우드 채택 노력을 추진하는 다분야 팀입니다. 자세한 내용은 AWS 클라우드 엔터프라이즈 전략 블로그의 CCoE 게시물을 참조하세요.

클라우드 컴퓨팅

원격 데이터 스토리지와 IoT 디바이스 관리에 일반적으로 사용되는 클라우드 기술 클라우드 컴퓨팅은 일반적으로 엣지 컴퓨팅 기술에 연결되어 있습니다.

클라우드 운영 모델

IT 조직에서 하나 이상의 클라우드 환경을 구축, 성숙화 및 최적화하는 데 사용되는 운영 모델입니다. 자세한 내용은 클라우드 운영 모델 구축을 참조하십시오.

클라우드 채택 단계

조직이 AWS 클라우드로 마이그레이션할 때 일반적으로 거치는 4단계는 다음과 같습니다.

- 프로젝트 - 개념 증명 및 학습 목적으로 몇 가지 클라우드 관련 프로젝트 실행
- 기반 – 클라우드 채택 확장을 위한 기초 투자(예: 랜딩 존 생성, CCoE 정의, 운영 모델 구축)
- 마이그레이션 - 개별 애플리케이션 마이그레이션
- Re-invention - 제품 및 서비스 최적화와 클라우드 혁신

이러한 단계는 Stephen Orban이 블로그 게시물 The Journey Toward Cloud-First and the Stages of Adoption on the AWS 클라우드 Enterprise Strategy 블로그에서 정의했습니다. AWS 마이그레이션 전략과 어떤 관련이 있는지에 대한 자세한 내용은 마이그레이션 준비 가이드를 참조하세요.

CMDB

구성 관리 데이터베이스를 참조하세요.

코드 리포지토리

소스 코드와 설명서, 샘플, 스크립트 등의 기타 자산이 버전 관리 프로세스를 통해 저장되고 업데이트되는 위치입니다. 일반적인 클라우드 리포지토리로 GitHub 또는 Bitbucket Cloud가 포함됩니다. 코드의 각 버전을 브랜치라고 합니다. 마이크로서비스 구조에서 각 리포지토리는 단일 기능 전용입니다. 단일 CI/CD 파이프라인은 여러 리포지토리를 사용할 수 있습니다.

콜드 캐시

비어 있거나, 제대로 채워지지 않았거나, 오래되었거나 관련 없는 데이터를 포함하는 버퍼 캐시입니다. 주 메모리나 디스크에서 데이터베이스 인스턴스를 읽어야 하기 때문에 성능에 영향을 미치며, 이는 버퍼 캐시에서 읽는 것보다 느립니다.

콜드 데이터

거의 액세스되지 않고 일반적으로 과거 데이터인 데이터. 이런 종류의 데이터를 쿼리할 때는 일반적으로 느린 쿼리가 허용됩니다. 이 데이터를 성능이 낮고 비용이 저렴한 스토리지 계층 또는 클래스로 옮기면 비용을 절감할 수 있습니다.

컴퓨터 비전(CV)

기계 학습을 사용하여 디지털 이미지 및 비디오와 같은 시각적 형식에서 정보를 분석하고 추출하는 AI 필드입니다. 예를 들어 Amazon SageMaker AI는 CV에 대한 이미지 처리 알고리즘을 제공합니다.

구성 드리프트

워크로드의 경우 구성이 예상되는 상태에서 변경됩니다. 이로 인해 워크로드가 규정을 준수하지 않을 수 있으며, 이는 일반적으로 점진적이고 의도되지 않은 작업입니다.

구성 관리 데이터베이스(CMDB)

하드웨어 및 소프트웨어 구성 요소와 해당 구성을 포함하여 데이터베이스와 해당 IT 환경에 대한 정보를 저장하고 관리하는 리포지토리입니다. 일반적으로 마이그레이션의 포트폴리오 탐색 및 분석 단계에서 CMDB의 데이터를 사용합니다.

규정 준수 팩

규정 준수 및 보안 검사를 사용자 지정하기 위해 조합할 수 있는 AWS Config 규칙 및 문제 해결 작업의 모음입니다. YAML 템플릿을 사용하여 적합성 팩을 AWS 계정 및 리전 또는 조직 전체에 단일 엔터티로 배포할 수 있습니다. 자세한 내용은 AWS Config 설명서의 적합성 팩을 참조하세요.

지속적 통합 및 지속적 전달(CI/CD)

소프트웨어 릴리스 프로세스의 소스, 빌드, 테스트, 스테이징 및 프로덕션 단계를 자동화하는 프로세스입니다. CI/CD는 일반적으로 파이프라인으로 설명됩니다. CI/CD를 통해 프로세스를 자동화하고, 생산성을 높이고, 코드 품질을 개선하고, 더 빠르게 제공할 수 있습니다. 자세한 내용은 지속적 전달의 이점을 참조하십시오. CD는 지속적 배포를 의미하기도 합니다. 자세한 내용은 지속적 전달(Continuous Delivery)과 지속적인 개발을 참조하십시오.

CV

컴퓨터 비전을 참조하세요.

D

저장 데이터

스토리지에 있는 데이터와 같이 네트워크에 고정되어 있는 데이터입니다.

데이터 분류

중요도와 민감도를 기준으로 네트워크의 데이터를 식별하고 분류하는 프로세스입니다. 이 프로세스는 데이터에 대한 적절한 보호 및 보존 제어를 결정하는 데 도움이 되므로 사이버 보안 위험 관리 전략의 중요한 구성 요소입니다. 데이터 분류는 AWS Well-Architected Framework의 보안 원칙 구성 요소입니다. 자세한 내용은 데이터 분류를 참조하십시오.

데이터 드리프트

프로덕션 데이터와 ML 모델 학습에 사용된 데이터 간의 상당한 차이 또는 시간 경과에 따른 입력 데이터의 의미 있는 변화. 데이터 드리프트는 ML 모델 예측의 전반적인 품질, 정확성 및 공정성을 저하시킬 수 있습니다.

전송 중 데이터

네트워크를 통과하고 있는 데이터입니다. 네트워크 리소스 사이를 이동 중인 데이터를 예로 들 수 있습니다.

데이터 메시

중앙 집중식 관리 및 거버넌스를 통해 분산되고 탈중앙화된 데이터 소유권을 제공하는 아키텍처 프레임워크입니다.

데이터 최소화

꼭 필요한 데이터만 수집하고 처리하는 원칙입니다. 에서 데이터를 최소화하면 개인 정보 보호 위험, 비용 및 분석 탄소 발자국을 줄일 AWS 클라우드 수 있습니다.

데이터 경계

신뢰할 수 있는 자격 증명만 예상 네트워크에서 신뢰할 수 있는 리소스에 액세스하도록 하는 데 도움이 되는 AWS 환경의 예방 가드레일 세트입니다. 자세한 내용은 데이터 경계 구축을 참조하세요 AWS.

데이터 사전 처리

원시 데이터를 ML 모델이 쉽게 구문 분석할 수 있는 형식으로 변환하는 것입니다. 데이터를 사전 처리한다는 것은 특정 열이나 행을 제거하고 누락된 값, 일관성이 없는 값 또는 중복 값을 처리함을 의미할 수 있습니다.

데이터 출처

라이프사이클 전반에 걸쳐 데이터의 출처와 기록을 추적하는 프로세스(예: 데이터 생성, 전송, 저장 방법).

데이터 주체

데이터를 수집 및 처리하는 개인입니다.

데이터 웨어하우스

분석과 같은 비즈니스 인텔리전스를 지원하는 데이터 관리 시스템입니다. 데이터 웨어하우스에는 보통 많은 양의 기록 데이터가 포함되며 일반적으로 쿼리 및 분석에 사용됩니다.

데이터 정의 언어(DDL)

데이터베이스에서 테이블 및 객체의 구조를 만들거나 수정하기 위한 명령문 또는 명령입니다.

데이터베이스 조작 언어(DML)

데이터베이스에서 정보를 수정(삽입, 업데이트 및 삭제)하기 위한 명령문 또는 명령입니다.

DDL

데이터 정의 언어를 참조하세요.

딥 앙상블

예측을 위해 여러 딥 러닝 모델을 결합하는 것입니다. 딥 앙상블을 사용하여 더 정확한 예측을 얻거나 예측의 불확실성을 추정할 수 있습니다.

딥 러닝

여러 계층의 인공 신경망을 사용하여 입력 데이터와 관심 대상 변수 간의 매핑을 식별하는 ML 하위 분야입니다.

심층 방어

네트워크와 그 안의 데이터 기밀성, 무결성 및 가용성을 보호하기 위해 컴퓨터 네트워크 전체에 일련의 보안 메커니즘과 제어를 신중하게 계층화하는 정보 보안 접근 방식입니다. 이 전략을 채택하면 AWS Organizations 구조의 여러 계층에 여러 컨트롤을 AWS추가하여 리소스를 보호할 수 있습니다. 예를 들어, 심층 방어 접근 방식은 다단계 인증, 네트워크 세분화 및 암호화를 결합할 수 있습니다.

위임된 관리자

에서 AWS Organizations호환되는 서비스는 AWS 멤버 계정을 등록하여 조직의 계정을 관리하고 해당 서비스에 대한 권한을 관리할 수 있습니다. 이러한 계정을 해당 서비스의 위임된 관리자라고

합니다. 자세한 내용과 호환되는 서비스 목록은 AWS Organizations 설명서의 AWS Organizations와 함께 사용할 수 있는 AWS 서비스를 참조하십시오.

배포

대상 환경에서 애플리케이션, 새 기능 또는 코드 수정 사항을 사용할 수 있도록 하는 프로세스입니다. 배포에는 코드 베이스의 변경 사항을 구현한 다음 애플리케이션 환경에서 해당 코드베이스를 구축하고 실행하는 작업이 포함됩니다.

개발 환경

환경을 참조하세요.

탐지 제어

이벤트 발생 후 탐지, 기록 및 알림을 수행하도록 설계된 보안 제어입니다. 이러한 제어는 기존의 예방적 제어를 우회한 보안 이벤트를 알리는 2차 방어선입니다. 자세한 내용은 AWS에서 보안 제어 구현의 탐지 제어를 참조하세요.

개발 가치 흐름 매핑 (DVSM)

소프트웨어 개발 라이프사이클에서 속도와 품질에 부정적인 영향을 미치는 제약 조건을 식별하고 우선 순위를 지정하는 데 사용되는 프로세스입니다. DVSM은 원래 린 제조 방식을 위해 설계된 가치 흐름 매핑 프로세스를 확장합니다. 소프트웨어 개발 프로세스를 통해 가치를 창출하고 이동하는 데 필요한 단계와 팀에 중점을 둡니다.

디지털 트윈

건물, 공장, 산업 장비 또는 생산 라인과 같은 실제 시스템을 가상으로 표현한 것입니다. 디지털 트윈은 예측 유지 보수, 원격 모니터링, 생산 최적화를 지원합니다.

차원 테이블

스타 스키마에서 팩트 테이블의 정량적 데이터에 대한 데이터 속성을 포함하는 더 작은 테이블을 말합니다. 차원 테이블 속성은 일반적으로 텍스트 필드나 텍스트처럼 동작하는 개별 숫자입니다. 이러한 속성은 보통 쿼리 제약, 필터링 및 결과 세트 레이블 지정에 사용됩니다.

재해

워크로드 또는 시스템이 기본 배포 위치에서 비즈니스 목표를 달성하지 못하게 방해하는 이벤트입니다. 이러한 이벤트는 자연재해, 기술적 오류, 의도하지 않은 구성 오류 또는 멀웨어 공격과 같은 사람의 행동으로 인한 결과일 수 있습니다.

재해 복구(DR)

재해로 인한 가동 중지 시간 및 데이터 손실을 최소화하기 위해 사용하는 전략 및 프로세스입니다. 자세한 내용은 AWS Well-Architected Framework의 Disaster Recovery of Workloads on AWS: Recovery in the Cloud를 참조하세요.

DML

데이터베이스 조작 언어를 참조하세요.

도메인 기반 설계

구성 요소를 각 구성 요소가 제공하는 진화하는 도메인 또는 핵심 비즈니스 목표에 연결하여 복잡한 소프트웨어 시스템을 개발하는 접근 방식입니다. 이 개념은 에릭 에반스에 의해 그의 저서인 도메인 기반 디자인: 소프트웨어 중심의 복잡성 해결(Boston: Addison-Wesley Professional, 2003)에서 소개되었습니다. Strangler Fig 패턴과 함께 도메인 기반 설계를 사용하는 방법에 대한 자세한 내용은 컨테이너 및 Amazon API Gateway를 사용하여 기존의 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화하는 방법을 참조하십시오.

DR

재해 복구를 참조하세요.

드리프트 감지

기준이 되는 구성과의 편차 추적을 말합니다. 예를 들어 AWS CloudFormation 를 사용하여 시스템 리소스의 드리프트를 감지하거나를 사용하여 AWS Control Tower 거버넌스 요구 사항 준수에 영향을 미칠 수 있는 랜딩 존의 변경 사항을 감지할 수 있습니다.

DVSM

개발 가치 흐름 매핑을 참조하세요.

E

EDA

탐색 데이터 분석을 참조하세요.

EDI

전자 데이터 교환을 참조하세요.

엣지 컴퓨팅

IoT 네트워크의 엣지에서 스마트 디바이스의 컴퓨팅 성능을 개선하는 기술 엣지 컴퓨팅은 클라우드 컴퓨팅에 비해 보다 통신 지연 시간을 줄이고 응답 시간을 개선할 수 있습니다.

전자 데이터 교환(EDI)

조직 간 비즈니스 문서의 자동화된 교환을 나타냅니다. 자세한 내용은 전자 데이터 교환(EDI)이란 무엇인가요?를 참조하세요.

암호화

사람이 읽을 수 있는 일반 텍스트 데이터를 사이퍼텍스트로 변환하는 컴퓨팅 프로세스입니다.

암호화 키

암호화 알고리즘에 의해 생성되는 무작위 비트의 암호화 문자열입니다. 키의 길이는 다양할 수 있으며 각 키는 예측할 수 없고 고유하게 설계되었습니다.

엔디안

컴퓨터 메모리에 바이트가 저장되는 순서입니다. 빅 엔디안 시스템은 가장 중요한 바이트를 먼저 저장합니다. 리틀 엔디안 시스템은 가장 덜 중요한 바이트를 먼저 저장합니다.

엔드포인트

서비스 엔드포인트를 참조하세요.

엔드포인트 서비스

Virtual Private Cloud(VPC)에서 호스팅하여 다른 사용자와 공유할 수 있는 서비스입니다. 를 사용하여 엔드포인트 서비스를 생성하고 다른 AWS 계정 또는 AWS Identity and Access Management (IAM) 보안 주체에 권한을 AWS PrivateLink 부여할 수 있습니다. 이러한 계정 또는 보안 주체는 인터페이스 VPC 엔드포인트를 생성하여 엔드포인트 서비스에 비공개로 연결할 수 있습니다. 자세한 내용은 Amazon Virtual Private Cloud(VPC) 설명서의 엔드포인트 서비스 생성을 참조하십시오.

엔터프라이즈 리소스 계획(ERP)

엔터프라이즈의 주요 비즈니스 프로세스(예: 회계, MES, 프로젝트 관리)를 자동화하고 관리하는 시스템입니다.

봉투 암호화

암호화 키를 다른 암호화 키로 암호화하는 프로세스입니다. 자세한 내용은 AWS Key Management Service (AWS KMS) 설명서의 봉투 암호화를 참조하세요.

환경

실행 중인 애플리케이션의 인스턴스입니다. 다음은 클라우드 컴퓨팅의 일반적인 환경 유형입니다.

- 개발 환경 - 애플리케이션 유지 관리를 담당하는 핵심 팀만 사용할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. 개발 환경은 변경 사항을 상위 환경으로 승격하기 전에 테스트하는 데 사용됩니다. 이러한 유형의 환경을 테스트 환경이라고도 합니다.
- 하위 환경 - 초기 빌드 및 테스트에 사용되는 환경을 비롯한 애플리케이션의 모든 개발 환경입니다.
- 프로덕션 환경 - 최종 사용자가 액세스할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. CI/CD 파이프라인에서 프로덕션 환경이 마지막 배포 환경입니다.
- 상위 환경 - 핵심 개발 팀 이외의 사용자가 액세스할 수 있는 모든 환경입니다. 프로덕션 환경, 프로덕션 이전 환경 및 사용자 수용 테스트를 위한 환경이 여기에 포함될 수 있습니다.

에픽

애자일 방법론에서 작업을 구성하고 우선순위를 정하는 데 도움이 되는 기능적 범주입니다. 에픽은 요구 사항 및 구현 작업에 대한 개괄적인 설명을 제공합니다. 예를 들어, AWS CAF 보안 에픽에는 ID 및 액세스 관리, 탐지 제어, 인프라 보안, 데이터 보호 및 인시던트 대응이 포함됩니다. AWS 마이그레이션 전략의 에픽에 대한 자세한 내용은 프로그램 구현 가이드를 참조하십시오.

ERP

엔터프라이즈 리소스 계획을 참조하세요.

탐색 데이터 분석(EDA)

데이터 세트를 분석하여 주요 특성을 파악하는 프로세스입니다. 데이터를 수집 또는 집계한 다음 초기 조사를 수행하여 패턴을 찾고, 이상을 탐지하고, 가정을 확인합니다. EDA는 요약 통계를 계산하고 데이터 시각화를 생성하여 수행됩니다.

F

팩트 테이블

스타 스키마의 중앙 테이블입니다. 비즈니스 운영에 대한 정량적 데이터를 저장합니다. 일반적으로 팩트 테이블은 측정값이 있는 열 및 차원 테이블에 대한 외래 키가 있는 열과 같이 두 가지 열 유형을 포함합니다.

빠른 실패

개발 수명 주기를 줄이기 위해 빈번한 증분 테스트를 사용하는 철학입니다. 애자일 접근 방식의 핵심입니다.

장애 격리 경계

에서 장애의 영향을 제한하고 워크로드의 복원력을 개선하는 데 도움이 되는 가용 영역, AWS 리전 컨트롤 플레인 또는 데이터 플레인과 같은 AWS 클라우드경계입니다. 자세한 내용은 AWS 장애 격리 경계를 참조하세요.

기능 브랜치

브랜치를 참조하세요.

기능

예측에 사용하는 입력 데이터입니다. 예를 들어, 제조 환경에서 기능은 제조 라인에서 주기적으로 캡처되는 이미지일 수 있습니다.

기능 중요도

모델의 예측에 특성이 얼마나 중요한지를 나타냅니다. 이는 일반적으로 SHAP(Shapley Additive Descriptions) 및 통합 그레디언트와 같은 다양한 기법을 통해 계산할 수 있는 수치 점수로 표현됩니다. 자세한 내용은 기계 학습 모델 해석 가능성을 참조하세요 AWS.

기능 변환

추가 소스로 데이터를 보강하거나, 값을 조정하거나, 단일 데이터 필드에서 여러 정보 세트를 추출하는 등 ML 프로세스를 위해 데이터를 최적화하는 것입니다. 이를 통해 ML 모델이 데이터를 활용할 수 있습니다. 예를 들어, 날짜 '2021-05-27 00:15:37'을 '2021년', '5월', '목', '15일'로 분류하면 학습 알고리즘이 다양한 데이터 구성 요소와 관련된 미묘한 패턴을 학습하는 데 도움이 됩니다.

퓨샷 프롬프팅

유사한 태스크를 수행하도록 요청하기 전에 LLM에 태스크와 원하는 출력을 보여주는 몇 가지 예제를 제공합니다. 이 기법은 모델이 프롬프트에 포함된 예제(샷)에서 학습하는 컨텍스트 내 학습을 적용합니다. 퓨샷 프롬프팅은 특정 형식 지정, 추론 또는 분야별 지식이 필요한 태스크에 효과적일 수 있습니다. 제로샷 프롬프팅도 참조하세요.

FGAC

세분화된 액세스 제어를 참조하세요.

세분화된 액세스 제어(FGAC)

여러 조건을 사용하여 액세스 요청을 허용하거나 거부합니다.

플래시컷 마이그레이션

단계적 접근 방식을 사용하는 대신 변경 데이터 캡처를 통해 지속적 데이터 복제를 사용하여 최단 시간에 데이터를 마이그레이션하는 데이터베이스 마이그레이션 방법입니다. 목표는 가동 중지 시간을 최소화하는 것입니다.

FM

파운데이션 모델을 참조하세요.

파운데이션 모델(FM)

일반화되고 레이블이 지정되지 않은 데이터의 대규모 데이터세트에서 훈련된 대규모 딥 러닝 신경망입니다. FM은 언어 이해, 텍스트 및 이미지 생성, 자연어 대화와 같은 다양한 일반 태스크를 수행할 수 있습니다. 자세한 내용은 파운데이션 모델이란?을 참조하세요.

FM 게이트웨이

파운데이션 모델에 대한 액세스를 제어하고 정규화하는 중앙 집중식 중개자입니다. LLM 게이트웨이라고도 합니다.

G

생성형 AI

대량의 데이터에서 훈련되었으며 간단한 텍스트 프롬프트를 사용하여 이미지, 비디오, 텍스트, 오디오와 같은 새 콘텐츠와 아티팩트를 생성할 수 있는 AI 모델의 하위 세트입니다. 자세한 내용은 생성형 AI란 무엇인가요?를 참조하세요.

지리적 차단

지리적 제한을 참조하세요.

지리적 제한(지리적 차단)

Amazon CloudFront에서 특정 국가의 사용자가 콘텐츠 배포에 액세스하지 못하도록 하는 옵션입니다. 허용 목록 또는 차단 목록을 사용하여 승인된 국가와 차단된 국가를 지정할 수 있습니다. 자세한 내용은 CloudFront 설명서의 콘텐츠의 지리적 배포 제한을 참조하십시오.

Gitflow 워크플로

하위 환경과 상위 환경이 소스 코드 리포지토리의 서로 다른 브랜치를 사용하는 방식입니다. Gitflow 워크플로는 레거시로 간주되며 트렁크 기반 워크플로는 선호되는 현대적 접근 방식입니다.

골든 이미지

시스템 또는 소프트웨어의 새 인스턴스를 배포하기 위한 템플릿으로 사용되는 해당 시스템 또는 소프트웨어의 스냅샷입니다. 예를 들어 제조 분야에서는 골든 이미지를 사용하여 여러 디바이스에서 소프트웨어를 프로비저닝할 수 있으며 이를 통해 딥이스 제조 작업의 속도, 확장성 및 생산성을 개선할 수 있습니다.

브라운필드 전략

새로운 환경에서 기존 인프라의 부재 시스템 아키텍처에 대한 그린필드 전략을 채택할 때 브라운필드라고도 하는 기존 인프라와의 호환성 제한 없이 모든 새로운 기술을 선택할 수 있습니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 그린필드 전략을 혼합할 수 있습니다.

가드레일

조직 단위(OU) 전체에서 리소스, 정책 및 규정 준수를 관리하는 데 도움이 되는 중요 규칙입니다. 예방 가드레일은 규정 준수 표준에 부합하도록 정책을 시행하며, 서비스 제어 정책과 IAM 권한 경계를 사용하여 구현됩니다. 탐지 가드레일은 정책 위반 및 규정 준수 문제를 감지하고 해결을 위한 알림을 생성하며, 이는 AWS Config Amazon GuardDuty AWS Security Hub CSPM, , AWS Trusted Advisor Amazon Inspector 및 사용자 지정 AWS Lambda 검사를 사용하여 구현됩니다.

가드레일(AI)

책임감 있고 안전한 AI 동작을 보장하기 위해 에이전트 입력 및 출력을 필터링, 검증 및 제약하는 안전 메커니즘입니다.

H

HA

고가용성을 참조하세요.

이기종 데이터베이스 마이그레이션

다른 데이터베이스 엔진을 사용하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Oracle에서 Amazon Aurora로) 이기종 마이그레이션은 일반적으로 리아키텍트 작업의 일부이며 스키마를 변환하는 것은 복잡한 작업일 수 있습니다.AWS 는 스키마 변환에 도움이 되는 AWS SCT를 제공합니다.

높은 가용성(HA)

문제나 재해 발생 시 개입 없이 지속적으로 운영할 수 있는 워크로드의 능력. HA 시스템은 자동으로 장애 조치되고, 지속적으로 고품질 성능을 제공하고, 성능에 미치는 영향을 최소화하면서 다양한 부하와 장애를 처리하도록 설계되었습니다.

히스토리언 현대화

제조 산업의 요구 사항을 더 잘 충족하도록 운영 기술(OT) 시스템을 현대화하고 업그레이드하는 데 사용되는 접근 방식입니다. 히스토리언은 공장의 다양한 출처에서 데이터를 수집하고 저장하는 데 사용되는 일종의 데이터베이스입니다.

홀드아웃 데이터

기계 학습 모델을 훈련하는 데 사용되는 데이터세트에서 보류되는 레이블이 지정된 기록 데이터의 일부입니다. 홀드아웃 데이터를 사용하여 모델 예측을 홀드아웃 데이터와 비교해 모델 성능을 평가할 수 있습니다.

human-in-the-loop(HitL)

중요한 결정 시점에서 인적 검토 및 승인을 위해 에이전트 실행이 일시 중지되는 워크플로 패턴입니다.

동종 데이터베이스 마이그레이션

동일한 데이터베이스 엔진을 공유하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Microsoft SQL Server에서 Amazon RDS for SQL Server로) 동종 마이그레이션은 일반적으로 리호스팅 또는 리플랫포밍 작업의 일부입니다. 네이티브 데이터베이스 유틸리티를 사용하여 스키마를 마이그레이션할 수 있습니다.

핫 데이터

자주 액세스하는 데이터(예: 실시간 데이터 또는 최근 번역 데이터). 일반적으로 이 데이터에는 빠른 쿼리 응답을 제공하기 위한 고성능 스토리지 계층 또는 클래스가 필요합니다.

핫픽스

프로덕션 환경의 중요한 문제를 해결하기 위한 긴급 수정입니다. 핫픽스는 긴급하기 때문에 일반적인 DevOps 릴리스 워크플로 외부에서 실행됩니다.

하이퍼케어 기간

전환 직후 마이그레이션 팀이 문제를 해결하기 위해 클라우드에서 마이그레이션된 애플리케이션을 관리하고 모니터링하는 기간입니다. 일반적으로 이 기간은 1~4일입니다. 하이퍼케어 기간이 끝나면 마이그레이션 팀은 일반적으로 애플리케이션에 대한 책임을 클라우드 운영 팀에 넘깁니다.

I

IaC

코드형 인프라를 참조하세요.

자격 증명 기반 정책

AWS 클라우드 환경 내에서 권한을 정의하는 하나 이상의 IAM 보안 주체에 연결된 정책입니다.

유휴 애플리케이션

90일 동안 평균 CPU 및 메모리 사용량이 5~20%인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하거나 온프레미스에 유지하는 것이 일반적입니다.

IIoT

산업용 사물 인터넷을 참조하세요.

변경 불가능한 인프라

기존 인프라를 업데이트, 패치 또는 수정하는 대신 프로덕션 워크로드에 대한 새 인프라를 배포하는 모델입니다. 변경 불가능한 인프라는 변경 가능한 인프라보다 본질적으로 더 일관되고 안정적이며 예측 가능합니다. 자세한 내용은 AWS Well-Architected Framework의 변경 불가능한 인프라를 사용하여 배포 모범 사례를 참조하세요.

인바운드(수신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 외부에서 네트워크 연결을 수락, 검사 및 라우팅하는 VPC입니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

증분 마이그레이션

한 번에 전체 전환을 수행하는 대신 애플리케이션을 조금씩 마이그레이션하는 전환 전략입니다. 예를 들어, 처음에는 소수의 마이크로서비스나 사용자만 새 시스템으로 이동할 수 있습니다. 모든 것이 제대로 작동하는지 확인한 후에는 레거시 시스템을 폐기할 수 있을 때까지 추가 마이크로서비스 또는 사용자를 점진적으로 이동할 수 있습니다. 이 전략을 사용하면 대규모 마이그레이션과 관련된 위험을 줄일 수 있습니다.

Industry 4.0

연결성, 실시간 데이터, 자동화, 분석 및 AI/ML의 발전을 통해 제조 프로세스의 현대화를 나타내기 위해 2016년에 Klaus Schwab에서 도입한 용어입니다.

인프라

애플리케이션의 환경 내에 포함된 모든 리소스와 자산입니다.

코드형 인프라(IaC)

구성 파일 세트를 통해 애플리케이션의 인프라를 프로비저닝하고 관리하는 프로세스입니다. IaC는 새로운 환경의 반복 가능성, 신뢰성 및 일관성을 위해 인프라 관리를 중앙 집중화하고, 리소스를 표준화하고, 빠르게 확장할 수 있도록 설계되었습니다.

산업용 사물 인터넷(IIoT)

제조, 에너지, 자동차, 의료, 생명과학, 농업 등의 산업 부문에서 인터넷에 연결된 센서 및 디바이스의 사용 자세한 내용은 산업용 사물 인터넷(IoT) 디지털 트랜스포메이션 전략 구축을 참조하십시오.

검사 VPC

AWS 다중 계정 아키텍처에서는 VPC(동일하거나 다른 AWS 리전), 인터넷 및 온프레미스 네트워크 간의 네트워크 트래픽 검사를 관리하는 중앙 집중식 VPCs입니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

사물 인터넷(IoT)

인터넷이나 로컬 통신 네트워크를 통해 다른 디바이스 및 시스템과 통신하는 센서 또는 프로세서가 내장된 연결된 물리적 객체의 네트워크 자세한 내용은 IoT란?을 참조하십시오.

해석력

모델의 예측이 입력에 따라 어떻게 달라지는지를 사람이 이해할 수 있는 정도를 설명하는 기계 학습 모델의 특성입니다. 자세한 내용은 기계 학습 모델 해석 가능성을 참조하세요 AWS.

IoT

사물 인터넷을 참조하세요.

IT 정보 라이브러리(ITIL)

IT 서비스를 제공하고 이러한 서비스를 비즈니스 요구 사항에 맞게 조정하기 위한 일련의 모범 사례 ITIL은 ITSM의 기반을 제공합니다.

IT 서비스 관리(ITSM)

조직의 IT 서비스 설계, 구현, 관리 및 지원과 관련된 활동 클라우드 운영을 ITSM 도구와 통합하는 방법에 대한 자세한 내용은 운영 통합 가이드를 참조하십시오.

ITIL

IT 정보 라이브러리를 참조하세요.

ITSM

IT 서비스 관리를 참조하세요.

L

레이블 기반 액세스 제어(LBAC)

사용자 및 데이터 자체에 각각 보안 레이블 값을 명시적으로 할당하는 필수 액세스 제어(MAC)를 구현한 것입니다. 사용자 보안 레이블과 데이터 보안 레이블 간의 교차 부분에 따라 사용자가 볼 수 있는 행과 열이 결정됩니다.

랜딩 존

랜딩 존은 확장 가능하고 안전한 잘 설계된 다중 계정 AWS 환경입니다. 조직은 여기에서부터 보안 및 인프라 환경에 대한 확신을 가지고 워크로드와 애플리케이션을 신속하게 시작하고 배포할 수 있습니다. 랜딩 존에 대한 자세한 내용은 안전하고 확장 가능한 다중 계정 AWS 환경 설정을 참조하십시오.

대규모 언어 모델(LLM)

방대한 양의 데이터에서 사전 훈련된 딥 러닝 AI 모델입니다. LLM은 질문에 대한 답변, 문서 요약, 텍스트를 다른 언어로 번역, 문장 완성과 같은 여러 태스크를 수행할 수 있습니다. 자세한 내용은 대규모 언어 모델(LLM)이란 무엇인가요?를 참조하세요.

대규모 마이그레이션

300대 이상의 서버 마이그레이션입니다.

LBAC

레이블 기반 액세스 제어를 참조하세요.

최소 권한

작업을 수행하는 데 필요한 최소 권한을 부여하는 보안 모범 사례입니다. 자세한 내용은 IAM 설명서의 최소 권한 적용을 참조하십시오.

리프트 앤드 시프트

7R을 참조하세요.

리틀 엔디안 시스템

가장 덜 중요한 바이트를 먼저 저장하는 시스템입니다. 엔디안도 참조하세요.

LLM

대규모 언어 모델을 참조하세요.

하위 환경

환경을 참조하세요.

M

기계 학습(ML)

패턴 인식 및 학습에 알고리즘과 기법을 사용하는 인공 지능의 한 유형입니다. ML은 사물 인터넷(IoT) 데이터와 같은 기록된 데이터를 분석하고 학습하여 패턴을 기반으로 통계 모델을 생성합니다. 자세한 내용은 기계 학습을 참조하십시오.

기본 브랜치

브랜치를 참조하세요.

맬웨어

컴퓨터 보안 또는 프라이버시를 위협하도록 설계된 소프트웨어입니다. 맬웨어는 컴퓨터 시스템을 방해하거나 민감한 정보를 유출하거나 무단 액세스 권한을 확보할 수 있습니다. 맬웨어의 예로 바이러스, 웜, 랜섬웨어, 트로이 목마, 스파이웨어, 키로거 등이 있습니다.

관리형 서비스

AWS 서비스 는 인프라 계층, 운영 체제 및 플랫폼을 AWS 운영하며 사용자는 엔드포인트에 액세스하여 데이터를 저장하고 검색합니다. 관리형 서비스의 예로 Amazon Simple Storage Service(Amazon S3) 및 Amazon DynamoDB가 있습니다. 이를 추상화된 서비스라고도 합니다.

제조 실행 시스템(MES)

원자재를 생산 현장에서 완제품으로 변환하는 생산 프로세스를 추적, 모니터링, 문서화 및 제어하기 위한 소프트웨어 시스템입니다.

MAP

Migration Acceleration Program을 참조하세요.

MCP

모델 컨텍스트 프로토콜을 참조하세요.

Model Context Protocol(MCP)

에이전트 간??? 통신을 위한 상태 비저장 프로토콜입니다.

MCP 서버

모델 컨텍스트 프로토콜을 통해 하나 이상의 도구를 노출하는 서비스입니다.

메커니즘

도구를 생성하고 도구 채택을 유도한 다음 조정을 위해 결과를 검사하는 전체 프로세스입니다. 메커니즘은 작동 시 자체적으로 강화하고 개선하는 주기입니다. 자세한 내용은 AWS Well-Architected Framework의 메커니즘 구축을 참조하세요.

멤버 계정

조직의 일부인 관리 계정을 AWS 계정 제외한 모든 계정. AWS Organizations하나의 계정은 한 번에 하나의 조직 멤버만 될 수 있습니다.

MES

제조 실행 시스템을 참조하세요.

메시지 큐 원격 분석 전송(MQTT)

리소스 제약이 있는 IoT 디바이스에 대한 게시 및 구독 패턴을 기반으로 하는 경량 Machine-to-Machine(M2M) 통신 프로토콜입니다.

마이크로서비스

잘 정의된 API를 통해 통신하고 일반적으로 소규모 자체 팀이 소유하는 소규모 독립 서비스입니다. 예를 들어, 보험 시스템에는 영업, 마케팅 등의 비즈니스 역량이나 구매, 청구, 분석 등의 하위 영역에 매핑되는 마이크로 서비스가 포함될 수 있습니다. 마이크로서비스의 이점으로 민첩성, 유연한 확장, 손쉬운 배포, 재사용 가능한 코드, 복원력 등이 있습니다. 자세한 내용은 AWS 서버리스 서비스를 사용하여 마이크로서비스 통합을 참조하세요.

마이크로서비스 아키텍처

각 애플리케이션 프로세스를 마이크로서비스로 실행하는 독립 구성 요소를 사용하여 애플리케이션을 구축하는 접근 방식입니다. 이러한 마이크로서비스는 경량 API를 사용하여 잘 정의된 인터페이스를 통해 통신합니다. 애플리케이션의 특정 기능에 대한 수요에 맞게 이 아키텍처의 각 마이크로서비스를 업데이트, 배포 및 조정할 수 있습니다. 자세한 내용은 에서 마이크로서비스 구현을 참조하세요 AWS.

Migration Acceleration Program(MAP)

조직이 클라우드로 전환하기 위한 강력한 운영 기반을 구축하고 초기 마이그레이션 비용을 상쇄하는 데 도움이 되는 컨설팅 지원, 교육 및 서비스를 제공하는 AWS 프로그램입니다. MAP에는 레거시 마이그레이션을 체계적인 방식으로 실행하기 위한 마이그레이션 방법론과 일반적인 마이그레이션 시나리오를 자동화하고 가속화하는 도구 세트가 포함되어 있습니다.

대규모 마이그레이션

애플리케이션 포트폴리오의 대다수를 웨이브를 통해 클라우드로 이동하는 프로세스로, 각 웨이브에서 더 많은 애플리케이션이 더 빠른 속도로 이동합니다. 이 단계에서는 이전 단계에서 배운 모범 사례와 교훈을 사용하여 팀, 도구 및 프로세스의 마이그레이션 팩토리를 구현하여 자동화 및 민첩한 제공을 통해 워크로드 마이그레이션을 간소화합니다. 이것은 AWS 마이그레이션 전략의 세 번째 단계입니다.

마이그레이션 팩토리

자동화되고 민첩한 접근 방식을 통해 워크로드 마이그레이션을 간소화하는 다기능 팀입니다. 마이그레이션 팩토리 팀에는 일반적으로 스프린트에서 일하는 운영, 비즈니스 분석가 및 소유자, 마이그레이션 엔지니어, 개발자, DevOps 전문가가 포함됩니다. 엔터프라이즈 애플리케이션 포트폴리오의 20~50%는 공장 접근 방식으로 최적화할 수 있는 반복되는 패턴으로 구성되어 있습니다. 자세한 내용은 이 콘텐츠 세트의 클라우드 마이그레이션 팩토리 가이드와 마이그레이션 팩토리에 대한 설명을 참조하십시오.

마이그레이션 메타데이터

마이그레이션을 완료하는 데 필요한 애플리케이션 및 서버에 대한 정보 각 마이그레이션 패턴에는 서로 다른 마이그레이션 메타데이터 세트가 필요합니다. 마이그레이션 메타데이터의 예로는 대상 서브넷, 보안 그룹 및 AWS 계정이 있습니다.

마이그레이션 패턴

사용되는 마이그레이션 전략, 마이그레이션 대상, 마이그레이션 애플리케이션 또는 서비스를 자세히 설명하는 반복 가능한 마이그레이션 작업입니다. 예: AWS Application Migration Service를 사용하여 Amazon EC2로 마이그레이션을 리호스팅합니다.

Migration Portfolio Assessment(MPA)

AWS 클라우드로 마이그레이션하는 비즈니스 사례를 검증하기 위한 정보를 제공하는 온라인 도구입니다. MPA는 상세한 포트폴리오 평가(서버 적정 규모 조정, 가격 책정, TCO 비교, 마이그레이션 비용 분석)와 마이그레이션 계획(애플리케이션 데이터 분석 및 데이터 수집, 애플리케이션 그룹화, 마이그레이션 우선순위 지정, 웨이브 계획)을 제공합니다. MPA 도구(로그인 필요)는 모든 AWS 컨설턴트와 APN 파트너 컨설턴트가 무료로 사용할 수 있습니다.

마이그레이션 준비 상태 평가(MRA)

AWS CAF를 사용하여 조직의 클라우드 준비 상태에 대한 인사이트를 얻고, 강점과 약점을 식별하고, 식별된 격차를 해소하기 위한 행동 계획을 수립하는 프로세스입니다. 자세한 내용은 마이그레이션 준비 가이드를 참조하십시오. MRA는 AWS 마이그레이션 전략의 첫 번째 단계입니다.

마이그레이션 전략

워크로드를 AWS 클라우드로 마이그레이션하는 데 사용되는 접근 방식입니다. 자세한 내용은 이 용어집의 7R 항목과 조직을 동원하여 대규모 마이그레이션 가속화를 참조하세요.

ML

기계 학습을 참조하세요.

현대화

비용을 절감하고 효율성을 높이고 혁신을 활용하기 위해 구식(레거시 또는 모놀리식) 애플리케이션과 해당 인프라를 클라우드의 민첩하고 탄력적이고 가용성이 높은 시스템으로 전환하는 것입니다. 자세한 내용은 AWS 클라우드에서 애플리케이션을 현대화하기 위한 전략을 참조하세요.

현대화 준비 상태 평가

조직 애플리케이션의 현대화 준비 상태를 파악하고, 이점, 위험 및 종속성을 식별하고, 조직이 해당 애플리케이션의 향후 상태를 얼마나 잘 지원할 수 있는지를 확인하는 데 도움이 되는 평가입니다. 평가 결과는 대상 아키텍처의 청사진, 현대화 프로세스의 개발 단계와 마일스톤을 자세히 설명하는 로드맵 및 파악된 격차를 해소하기 위한 실행 계획입니다. 자세한 내용은 AWS 클라우드에서 애플리케이션의 현대화 준비 상태 평가를 참조하세요.

모놀리식 애플리케이션(모놀리식 유형)

긴밀하게 연결된 프로세스를 사용하여 단일 서비스로 실행되는 애플리케이션입니다. 모놀리식 애플리케이션에는 몇 가지 단점이 있습니다. 한 애플리케이션 기능에 대한 수요가 급증하면 전체 아키텍처 규모를 조정해야 합니다. 코드 베이스가 커지면 모놀리식 애플리케이션의 기능을 추가하거나 개선하는 것도 더 복잡해집니다. 이러한 문제를 해결하기 위해 마이크로서비스 아키텍처를 사용할 수 있습니다. 자세한 내용은 마이크로서비스로 모놀리식 유형 분해를 참조하십시오.

MPA

Migration Portfolio Assessment를 참조하세요.

MQTT

메시지 큐 원격 분석 전송을 참조하세요.

멀티클래스 분류

여러 클래스에 대한 예측(2개 이상의 결과 중 하나 예측)을 생성하는 데 도움이 되는 프로세스입니다. 예를 들어, ML 모델이 '이 제품은 책인가요, 자동차인가요, 휴대폰인가요?' 또는 '이 고객이 가장 관심을 갖는 제품 범주는 무엇인가요?'라고 물을 수 있습니다.

변경 가능한 인프라

프로덕션 워크로드에 대한 기존 인프라를 업데이트하고 수정하는 모델입니다. 일관성, 신뢰성 및 예측 가능성을 높이기 위해 AWS Well-Architected Framework는 변경 불가능한 인프라를 모범 사례로 사용할 것을 권장합니다.

O

OAC

오리진 액세스 제어를 참조하세요.

OAI

오리진 액세스 ID를 참조하세요.

OCM

조직 변경 관리를 참조하세요.

오프라인 마이그레이션

마이그레이션 프로세스 중 소스 워크로드가 중단되는 마이그레이션 방법입니다. 이 방법은 가동 중지 증가를 수반하며 일반적으로 작고 중요하지 않은 워크로드에 사용됩니다.

OI

운영 통합을 참조하세요.

OLA

운영 수준 계약을 참조하세요.

온라인 마이그레이션

소스 워크로드를 오프라인 상태로 전환하지 않고 대상 시스템에 복사하는 마이그레이션 방법입니다. 워크로드에 연결된 애플리케이션은 마이그레이션 중에도 계속 작동할 수 있습니다. 이 방법은 가동 중지 차단 또는 최소화를 수반하며 일반적으로 중요한 프로덕션 워크로드에 사용됩니다.

OPC-UA

Open Process Communications - Unified Architecture(OPC-UA)를 참조하세요.

Open Process Communications - Unified Architecture(OPC-UA)

산업 자동화를 위한 Machine-to-Machine(M2M) 통신 프로토콜입니다. OPC-UA는 데이터 암호화, 인증 및 권한 부여 체계에 관한 상호 운용성 표준을 제공합니다.

운영 수준 협약(OLA)

서비스 수준에 관한 계약(SLA)을 지원하기 위해 직무 IT 그룹이 서로에게 제공하기로 약속한 내용을 명확히 하는 계약입니다.

운영 준비 상태 검토(ORR)

인시던트 및 잠재적 장애의 범위를 이해, 평가 또는 예방하거나 줄이는 데 도움이 되는 질문 체크리스트 및 관련 모범 사례입니다. 자세한 내용은 AWS Well-Architected Framework의 운영 준비 상태 검토(ORR)를 참조하세요.

운영 기술(OT)

물리적 환경에서 작동하여 산업 운영, 장비 및 인프라를 제어하는 하드웨어 및 소프트웨어 시스템입니다. 제조 분야에서 OT 및 정보 기술(IT) 시스템의 통합은 Industry 4.0 트랜스포메이션의 주요 중점 사항입니다.

운영 통합(OI)

클라우드에서 운영을 현대화하는 프로세스로 준비 계획, 자동화 및 통합을 수반합니다. 자세한 내용은 운영 통합 가이드를 참조하십시오.

조직 트레일

조직 AWS 계정 내 모든에 대한 모든 이벤트를 로깅 AWS CloudTrail 하는에서 생성된 추적입니다 AWS Organizations. 이 트레일은 조직에 속한 각 AWS 계정 에 생성되고 각 계정의 활동을 추적합니다. 자세한 내용은 CloudTrail 설명서의 Creating a trail for an organization을 참조하십시오.

조직 변경 관리(OCM)

사람, 문화 및 리더십 관점에서 중대하고 파괴적인 비즈니스 혁신을 관리하기 위한 프레임워크입니다. OCM은 변화 채택을 가속화하고, 과도기적 문제를 해결하고, 문화 및 조직적 변화를 주도함으로써 조직이 새로운 시스템 및 전략을 준비하고 전환할 수 있도록 지원합니다. AWS 마이그레이션 전략에서는 클라우드 채택 프로젝트에 필요한 변경 속도 때문에이 프레임워크를 인력 가속화라고 합니다. 자세한 내용은 사용 가이드를 참조하십시오.

오리진 액세스 제어(OAC)

CloudFront에서 Amazon Simple Storage Service(S3) 콘텐츠를 보호하기 위해 액세스를 제한하는 고급 옵션입니다. OAC는 AWS KMS (SSE-KMS)를 사용한 모든 서버 측 암호화 AWS 리전와 S3 버킷에 대한 동적 PUT 및 DELETE 요청에서 모든 S3 버킷을 지원합니다.

오리진 액세스 ID(OAI)

CloudFront에서 Amazon S3 콘텐츠를 보호하기 위해 액세스를 제한하는 옵션입니다. OAI를 사용하면 CloudFront는 Amazon S3가 인증할 수 있는 보안 주체를 생성합니다. 인증된 보안 주체는 특정 CloudFront 배포를 통해서만 S3 버킷의 콘텐츠에 액세스할 수 있습니다. 더 세분화되고 향상된 액세스 제어를 제공하는 OAC도 참조하십시오.

ORR

운영 준비 상태 검토를 참조하세요.

OT

운영 기술을 참조하세요.

아웃바운드(송신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 내에서 시작된 네트워크 연결을 처리하는 VPC입니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

P

권한 경계

사용자나 역할이 가질 수 있는 최대 권한을 설정하기 위해 IAM 보안 주체에 연결되는 IAM 관리 정책입니다. 자세한 내용은 IAM 설명서의 권한 경계를 참조하십시오.

개인 식별 정보(PII)

직접 보거나 다른 관련 데이터와 함께 짝을 지을 때 개인의 신원을 합리적으로 추론하는 데 사용할 수 있는 정보입니다. PII의 예로는 이름, 주소, 연락처 정보 등이 있습니다.

PII

개인 식별 정보를 참조하세요.

플레이북

클라우드에서 핵심 운영 기능을 제공하는 등 마이그레이션과 관련된 작업을 캡처하는 일련의 사전 정의된 단계입니다. 플레이북은 스크립트, 자동화된 런북 또는 현대화된 환경을 운영하는 데 필요한 프로세스나 단계 요약의 형태를 취할 수 있습니다.

PLC

프로그래밍 가능 로직 컨트롤러를 참조하세요.

PLM

제품 수명 주기 관리를 참조하세요.

정책

권한 정의(ID 기반 정책 참조), 액세스 조건 지정(리소스 기반 정책 참조), AWS Organizations 내 조직의 모든 계정에 대한 최대 권한 정의(서비스 제어 정책 참조)와 같은 작업을 수행할 수 있는 객체입니다.

다국어 지속성

데이터 액세스 패턴 및 기타 요구 사항을 기반으로 독립적으로 마이크로서비스의 데이터 스토리지 기술 선택. 마이크로서비스가 동일한 데이터 스토리지 기술을 사용하는 경우 구현 문제가 발생하거나 성능이 저하될 수 있습니다. 요구 사항에 가장 적합한 데이터 저장소를 사용하면 마이크로서비스를 더 쉽게 구현하고 성능과 확장성을 높일 수 있습니다.

포트폴리오 평가

마이그레이션을 계획하기 위해 애플리케이션 포트폴리오를 검색 및 분석하고 우선순위를 정하는 프로세스입니다. 자세한 내용은 마이그레이션 준비 상태 평가를 참조하십시오.

조건자

보통 `WHERE` 절에 있는 `true` 또는 `false`를 반환하는 쿼리 조건입니다.

푸시다운 조건자

전송 전에 쿼리의 데이터를 필터링하는 데이터베이스 쿼리 최적화 기법입니다. 이렇게 하면 관계형 데이터베이스에서 검색하고 처리해야 하는 데이터의 양이 줄고 쿼리 성능이 향상됩니다.

예방적 제어

이벤트 발생을 방지하도록 설계된 보안 제어입니다. 이 제어는 네트워크에 대한 무단 액세스나 원치 않는 변경을 방지하는 데 도움이 되는 1차 방어선입니다. 자세한 내용은 Implementing security controls on AWS의 Preventative controls를 참조하십시오.

보안 주체

작업을 수행하고 리소스에 액세스할 수 AWS 있는의 엔터티입니다. 이 엔터티는 일반적으로 , AWS 계정 IAM 역할 또는 사용자의 루트 사용자입니다. 자세한 내용은 IAM 설명서의 역할 용어 및 개념의 보안 주체를 참조하십시오.

개인 정보 보호 중심 설계

전체 개발 프로세스에서 개인 정보를 고려하는 시스템 엔지니어링에서의 접근 방식입니다.

프라이빗 호스팅 영역

Amazon Route 53에서 하나 이상의 VPC 내 도메인과 하위 도메인에 대한 DNS 쿼리에 응답하는 방법에 대한 정보가 담긴 컨테이너입니다. 자세한 내용은 Route 53 설명서의 프라이빗 호스팅 영역 작업을 참조하십시오.

선제적 제어

규정 미준수 리소스의 배포를 방지하도록 설계된 보안 제어입니다. 이러한 제어는 리소스를 프로비저닝하기 전에 리소스를 스캔합니다. 리소스가 제어를 준수하지 않으면 프로비저닝되지 않습니다. 자세한 내용은 AWS Control Tower 설명서의 제어 참조 가이드를 참조하고 보안 제어 구현의 사전 예방적 제어를 참조하세요. AWS

제품 수명 주기 관리(PLM)

설계, 개발 및 출시부터 성장 및 성숙도를 거쳐 거부 및 제거에 이르기까지 전체 수명 주기 동안 제품의 데이터 및 프로세스 관리를 나타냅니다.

프로덕션 환경

환경을 참조하세요.

프로그래밍 가능 로직 컨트롤러(PLC)

제조 분야에서 기계를 모니터링하고 제조 프로세스를 자동화하는 매우 안정적이고 적응력이 뛰어난 컴퓨터입니다.

프롬프트 체이닝

한 LLM 프롬프트의 출력을 다음 프롬프트의 입력으로 사용하여 더 나은 응답을 생성합니다. 이 기법은 복잡한 태스크를 하위 태스크로 나누거나 예비 응답을 반복적으로 세부 조정하거나 확장하는 데 사용됩니다. 이를 통해 모델 응답의 정확성과 관련성을 개선하고 보다 세분화되고 개인화된 결과를 얻을 수 있습니다.

가명화

데이터세트의 개인 식별자를 자리 표시자 값으로 바꾸는 프로세스입니다. 가명화는 개인 정보를 보호하는 데 도움이 될 수 있습니다. 가명화된 데이터는 여전히 개인 데이터로 간주됩니다.

게시/구독(pub/sub)

여러 마이크로서비스에서 비동기 통신을 지원하여 확장성과 응답성을 개선하는 패턴입니다. 예를 들어 마이크로서비스 기반 MES에서 마이크로서비스는 다른 마이크로서비스가 구독할 수 있는 채널에 이벤트 메시지를 게시할 수 있습니다. 시스템은 게시 서비스를 변경하지 않고도 새 마이크로서비스를 추가할 수 있습니다.

Q

쿼리 계획

SQL 관계형 데이터베이스 시스템의 데이터에 액세스하는 데 사용되는 명령어와 같은 일련의 단계입니다.

쿼리 계획 회귀

데이터베이스 서비스 최적화 프로그램이 데이터베이스 환경을 변경하기 전보다 덜 최적의 계획을 선택하는 경우입니다. 통계, 제한 사항, 환경 설정, 쿼리 파라미터 바인딩 및 데이터베이스 엔진 업데이트의 변경으로 인해 발생할 수 있습니다.

R

RACI 매트릭스

Responsible, Accountable, Consulted, Informed(RACI)를 참조하세요.

RAG

검색 증강 생성을 참조하세요.

랜섬웨어

결제가 완료될 때까지 컴퓨터 시스템이나 데이터에 대한 액세스를 차단하도록 설계된 악성 소프트웨어입니다.

RASCI 매트릭스

Responsible, Accountable, Consulted, Informed(RACI)를 참조하세요.

RCAC

행 및 열 액세스 제어를 참조하세요.

읽기 전용 복제본

읽기 전용 용도로 사용되는 데이터베이스의 사본입니다. 쿼리를 읽기 전용 복제본으로 라우팅하여 기본 데이터베이스의 로드를 줄일 수 있습니다.

리아키텍팅

7R을 참조하세요.

Recovery Point Objective(RPO)

마지막 데이터 복구 시점 이후 허용되는 최대 시간입니다. 이에 따라 마지막 복구 시점과 서비스 중단 사이에 허용되는 데이터 손실로 간주되는 범위가 결정됩니다.

Recovery Time Objective(RTO)

서비스 중단과 서비스 복원 사이의 허용 가능한 지연 시간입니다.

리팩터링

7R을 참조하세요.

리전

지리적 영역의 AWS 리소스 모음입니다. 각 AWS 리전 는 내결함성, 안정성 및 복원력을 제공하기 위해 서로 격리되고 독립적입니다. 자세한 내용은 계정에서 사용할 수 있는 AWS 리전 지정을 참조하세요.

회귀

숫자 값을 예측하는 ML 기법입니다. 예를 들어, '이 집은 얼마에 팔릴까?'라는 문제를 풀기 위해 ML 모델은 선형 회귀 모델을 사용하여 주택에 대해 알려진 사실(예: 면적)을 기반으로 주택의 매매 가격을 예측할 수 있습니다.

리호스팅

7R을 참조하세요.

릴리스

배포 프로세스에서 변경 사항을 프로덕션 환경으로 승격시키는 행위입니다.

재배치

7R을 참조하세요.

리플랫포밍

7R을 참조하세요.

재구매

7R을 참조하세요.

복원력

중단에 저항하거나 중단을 복구할 수 있는 애플리케이션의 기능입니다. 고가용성 및 재해 복구는 AWS 클라우드에서 복원력을 계획할 때 일반적인 고려 사항입니다. 자세한 내용은 AWS 클라우드 복원력을 참조하세요.

리소스 기반 정책

Amazon S3 버킷, 엔드포인트, 암호화 키 등의 리소스에 연결된 정책입니다. 이 유형의 정책은 액세스가 허용된 보안 주체, 지원되는 작업 및 충족해야 하는 기타 조건을 지정합니다.

RACI(Responsible, Accountable, Consulted, Informed) 매트릭스

마이그레이션 활동 및 클라우드 운영에 참여하는 모든 당사자의 역할과 책임을 정의하는 매트릭스입니다. 매트릭스 이름은 매트릭스에 정의된 책임 유형에서 파생됩니다. 실무 담당자 (R), 의사 결정권자 (A), 업무 수행 조언자 (C), 결과 통보 대상자 (I). 지원자는 (S) 선택사항입니다. 지원자를 포함하면 매트릭스를 RASCI 매트릭스라고 하고, 지원자를 제외하면 RACI 매트릭스라고 합니다.

대응 제어

보안 기준에서 벗어나거나 부정적인 이벤트를 해결하도록 설계된 보안 제어입니다. 자세한 내용은 AWS에서 보안 제어 구현의 대응 제어를 참조하세요.

retain

7R을 참조하세요.

사용 중지

7R을 참조하세요.

검색 증강 세대(RAG)

응답을 생성하기 전에 LLM이 훈련 데이터 소스 외부에 있는 신뢰할 수 있는 데이터 소스를 참조하는 생성형 AI 기술입니다. 예를 들어 RAG 모델은 조직의 지식 기반 또는 사용자 지정 데이터에 대

한 시맨틱 검색을 수행할 수 있습니다. 자세한 내용은 검색 증강 생성(RAG)이란 무엇인가요?를 참조하세요.

교체

공격자가 자격 증명에 액세스하는 것을 더욱 어렵게 만들기 위해 보안 암호를 주기적으로 업데이트하는 프로세스입니다.

행 및 열 액세스 제어(RCAC)

액세스 규칙이 정의된 기본적이고 유연한 SQL 표현식을 사용합니다. RCAC는 행 권한과 열 마스크로 구성됩니다.

RPO

목표 복구 시점(RPO)을 참조하세요.

RTO

목표 복구 시간(RTO)을 참조하세요.

런북

특정 작업을 수행하는 데 필요한 일련의 수동 또는 자동 절차입니다. 일반적으로 오류율이 높은 반복 작업이나 절차를 간소화하기 위해 런북을 만듭니다.

S

SAML 2.0

많은 ID 제공업체(idP)에서 사용하는 개방형 표준입니다. 이 기능을 사용하면 연동 SSO(Single Sign-On)를 AWS Management Console 사용할 수 있으므로 사용자는 조직의 모든 사용자에 대해 IAM에서 사용자를 생성하지 않고도에 로그인하거나 AWS API 작업을 호출할 수 있습니다. SAML 2.0 기반 페더레이션에 대한 자세한 내용은 IAM 설명서의 SAML 2.0 기반 페더레이션 정보를 참조하십시오.

SCADA

감독 제어 및 데이터 획득을 참조하세요.

SCP

서비스 제어 정책을 참조하세요.

보안 암호

에는 암호 또는 사용자 자격 증명과 같이 암호화된 형식으로 저장하는 AWS Secrets Manager기밀 또는 제한된 정보가 있습니다. 보안 암호 값과 메타데이터로 구성됩니다. 보안 암호 값은 바이너리, 단일 문자열 또는 여러 문자열일 수 있습니다. 자세한 내용은 AWS Secrets Manager 설명서의 Secrets Manager 보안 암호란 무엇인가요?를 참조하세요.

보안 중심 설계

전체 개발 프로세스에서 보안을 고려하는 시스템 엔지니어링에서의 접근 방식입니다.

보안 제어

위협 행위자가 보안 취약성을 악용하는 능력을 방지, 탐지 또는 감소시키는 기술적 또는 관리적 가드레일입니다. 보안 제어는 예방, 감지, 대응, 선제적과 같은 기본적인 네 가지 보안 제어 유형으로 구분됩니다.

보안 강화

공격 표면을 줄여 공격에 대한 저항력을 높이는 프로세스입니다. 더 이상 필요하지 않은 리소스 제거, 최소 권한 부여의 보안 모범 사례 구현, 구성 파일의 불필요한 기능 비활성화 등의 작업이 여기에 포함될 수 있습니다.

보안 정보 및 이벤트 관리(SIEM) 시스템

보안 정보 관리(SIM)와 보안 이벤트 관리(SEM) 시스템을 결합하는 도구 및 서비스입니다. SIEM 시스템은 서버, 네트워크, 디바이스 및 기타 소스에서 데이터를 수집, 모니터링 및 분석하여 위협과 보안 침해를 탐지하고 알림을 생성합니다.

보안 응답 자동화

보안 이벤트에 자동으로 응답하거나 이를 해결하도록 설계된 사전 정의되고 프로그래밍된 작업입니다. 이러한 자동화는 보안 모범 사례를 구현하는 데 도움이 되는 탐지 또는 대응 AWS 보안 제어 역할을 합니다. 자동화된 응답 작업의 예로 VPC 보안 그룹 수정, Amazon EC2 인스턴스 패치 적용 또는 자격 증명 교체 등이 있습니다.

서버 측 암호화

데이터를 AWS 서비스 수신하는가 대상에서 데이터를 암호화합니다.

서비스 제어 정책(SCP)

AWS Organizations에 속한 조직의 모든 계정에 대한 권한을 중앙 집중식으로 제어하는 정책입니다. SCP는 관리자가 사용자 또는 역할에 위임할 수 있는 작업에 대해 제한을 설정하거나 가드레일을 정의합니다. SCP를 허용 목록 또는 거부 목록으로 사용하여 허용하거나 금지할 서비스 또는 작

업을 지정할 수 있습니다. 자세한 내용은 AWS Organizations 설명서의 서비스 제어 정책을 참조하세요.

서비스 엔드포인트

에 대한 진입점의 URL입니다 AWS 서비스. 엔드포인트를 사용하여 대상 서비스에 프로그래밍 방식으로 연결할 수 있습니다. 자세한 내용은 AWS 일반 참조의 AWS 서비스 엔드포인트를 참조하십시오.

서비스 수준에 관한 계약(SLA)

IT 팀이 고객에게 제공하기로 약속한 내용(예: 서비스 가동 시간 및 성능)을 명시한 계약입니다.

서비스 수준 지표(SLI)

오류 발생률, 가용성 또는 처리량과 같은 서비스의 성능 측면에 대한 측정값입니다.

서비스 수준 목표(SLO)

서비스 수준 지표로 측정되는 서비스의 상태를 나타내는 목표 지표입니다.

공동 책임 모델

클라우드 보안 및 규정 준수를 AWS 위해와 공유하는 책임을 설명하는 모델입니다. AWS 는 클라우드의 보안을 책임지고, 사용자는 클라우드의 보안을 책임집니다. 자세한 내용은 공동 책임 모델을 참조하십시오.

섀도우 AI

조직 내 관리형 채널 외부에서 구축되거나 사용되는 승인되지 않은 AI 애플리케이션입니다.

SIEM

보안 정보 및 이벤트 관리 시스템을 참조하세요.

단일 장애점(SPOF)

애플리케이션을 중단시킬 수 있는 애플리케이션의 중요한 단일 구성 요소에서 발생하는 장애입니다.

SLA

서비스 수준 계약을 참조하세요.

SLI

서비스 수준 지표를 참조하세요.

SLO

서비스 수준 목표를 참조하세요.

분할 앤 시드 모델

현대화 프로젝트를 확장하고 가속화하기 위한 패턴입니다. 새로운 기능과 제품 릴리스가 정의되면 핵심 팀이 분할되어 새로운 제품 팀이 만들어집니다. 이를 통해 조직의 역량과 서비스 규모를 조정하고, 개발자 생산성을 개선하고, 신속한 혁신을 지원할 수 있습니다. 자세한 내용은 AWS 클라우드에서 애플리케이션을 현대화하기 위한 단계별 접근 방식을 참조하세요.

SPOF

단일 장애점을 참조하세요.

스타 스키마

하나의 큰 팩트 테이블을 사용하여 트랜잭션 또는 측정된 데이터를 저장하고 하나 이상의 더 작은 차원 테이블을 사용하여 데이터 속성을 저장하는 데이터베이스 조직 구조입니다. 이 구조는 데이터 웨어하우스에서 또는 비즈니스 인텔리전스 목적으로 사용하도록 설계되었습니다.

Strangler Fig 패턴

레거시 시스템을 폐기할 수 있을 때까지 시스템 기능을 점진적으로 다시 작성하고 교체하여 모놀리식 시스템을 현대화하기 위한 접근 방식. 이 패턴은 무화과 덩굴이 나무로 자라 결국 숙주를 압도하고 대체하는 것과 비슷합니다. Martin Fowler가 모놀리식 시스템을 다시 작성할 때 위험을 관리하는 방법으로 이 패턴을 도입했습니다. 이 패턴을 적용하는 방법의 예는 컨테이너 및 Amazon API Gateway를 사용하여 기존의 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화하는 방법을 참조하십시오.

서브넷

VPC의 IP 주소 범위입니다. 서브넷은 단일 가용 영역에 상주해야 합니다.

감독 제어 및 데이터 획득(SCADA)

제조 분야에서 하드웨어와 소프트웨어를 사용하여 물리적 자산과 프로덕션 작업을 모니터링하는 시스템입니다.

대칭 암호화

동일한 키를 사용하여 데이터를 암호화하고 복호화하는 암호화 알고리즘입니다.

합성 테스트

사용자 상호 작용을 시뮬레이션하여 잠재적 문제를 감지하거나 성능을 모니터링하는 방식으로 진행되는 시스템 테스트입니다. Amazon CloudWatch Synthetics를 사용하여 이러한 테스트를 생성할 수 있습니다.

시스템 프롬프트

LLM에 컨텍스트, 명령 또는 지침을 제공하여 동작을 지시하는 기법입니다. 시스템 프롬프트는 컨텍스트를 설정하고 사용자와의 상호 작용을 위한 규칙을 설정하는 데 도움이 됩니다.

T

tags

AWS 리소스를 구성하기 위한 메타데이터 역할을 하는 키-값 페어입니다. 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색, 필터링할 수 있습니다. 자세한 내용은 AWS 리소스에 태그 지정을 참조하십시오.

대상 변수

지도 ML에서 예측하려는 값으로, 결과 변수라고도 합니다. 예를 들어, 제조 설정에서 대상 변수는 제품 결함일 수 있습니다.

작업 목록

런북을 통해 진행 상황을 추적하는 데 사용되는 도구입니다. 작업 목록에는 런북의 개요와 완료해야 할 일반 작업 목록이 포함되어 있습니다. 각 일반 작업에 대한 예상 소요 시간, 소유자 및 진행 상황이 작업 목록에 포함됩니다.

테스트 환경

환경을 참조하세요.

훈련

ML 모델이 학습할 수 있는 데이터를 제공하는 것입니다. 훈련 데이터에는 정답이 포함되어야 합니다. 학습 알고리즘은 훈련 데이터에서 대상(예측하려는 답)에 입력 데이터 속성을 매핑하는 패턴을 찾고, 이러한 패턴을 캡처하는 ML 모델을 출력합니다. 그런 다음 ML 모델을 사용하여 대상을 모르는 새 데이터에 대한 예측을 할 수 있습니다.

tool

에이전트가 외부 시스템에서 작업을 수행하기 위해 호출할 수 있는 함수 또는 API입니다.

Transit Gateway

VPC와 온프레미스 네트워크를 상호 연결하는 데 사용할 수 있는 네트워크 전송 허브입니다. 자세한 내용은 AWS Transit Gateway 설명서의 전송 게이트웨이란 무엇입니까?를 참조하세요.

트렁크 기반 워크플로

개발자가 기능 브랜치에서 로컬로 기능을 구축하고 테스트한 다음 해당 변경 사항을 기본 브랜치에 병합하는 접근 방식입니다. 이후 기본 브랜치는 개발, 프로덕션 이전 및 프로덕션 환경에 순차적으로 구축됩니다.

신뢰할 수 있는 액세스

사용자를 대신하여 AWS Organizations 및 해당 계정에서 조직에서 작업을 수행하도록 지정하는 서비스에 대한 권한 부여. 신뢰할 수 있는 서비스는 필요할 때 각 계정에 서비스 연결 역할을 생성하여 관리 작업을 수행합니다. 자세한 내용은 설명서의 다른 AWS 서비스와 AWS Organizations 함께 사용을 참조하세요 AWS Organizations .

튜닝

ML 모델의 정확도를 높이기 위해 훈련 프로세스의 측면을 여러 변경하는 것입니다. 예를 들어, 레이블링 세트를 생성하고 레이블을 추가한 다음 다양한 설정에서 이러한 단계를 여러 번 반복하여 모델을 최적화하는 방식으로 ML 모델을 훈련할 수 있습니다.

피자 두 판 팀

피자 두 판이면 충분한 소규모 DevOps 팀. 피자 두 판 팀 규모는 소프트웨어 개발에 있어 가능한 최상의 공동 작업 기회를 보장합니다.

U

불확실성

예측 ML 모델의 신뢰성을 저해할 수 있는 부정확하거나 불완전하거나 알려지지 않은 정보를 나타내는 개념입니다. 불확실성에는 두 가지 유형이 있습니다. 인식론적 불확실성은 제한적이고 불완전한 데이터에 의해 발생하는 반면, 우연한 불확실성은 데이터에 내재된 노이즈와 무작위성에 의해 발생합니다.

차별화되지 않은 작업

애플리케이션을 만들고 운영하는 데 필요하지만 최종 사용자에게 직접적인 가치를 제공하거나 경쟁 우위를 제공하지 못하는 작업을 헤비 리프팅이라고도 합니다. 차별화되지 않은 작업의 예로는 조달, 유지보수, 용량 계획 등이 있습니다.

상위 환경

환경을 참조하세요.

V

정리

스토리지를 회수하고 성능을 향상시키기 위해 증분 업데이트 후 정리 작업을 수반하는 데이터베이스 유지 관리 작업입니다.

버전 제어

리포지토리의 소스 코드 변경과 같은 변경 사항을 추적하는 프로세스 및 도구입니다.

VPC 피어링

프라이빗 IP 주소를 사용하여 트래픽을 라우팅할 수 있게 하는 두 VPC 간의 연결입니다. 자세한 내용은 Amazon VPC 설명서의 VPC 피어링이란?을 참조하십시오.

취약성

시스템 보안을 손상시키는 소프트웨어 또는 하드웨어 결함입니다.

W

웜 캐시

자주 액세스하는 최신 관련 데이터를 포함하는 버퍼 캐시입니다. 버퍼 캐시에서 데이터베이스 인스턴스를 읽을 수 있기 때문에 주 메모리나 디스크에서 읽는 것보다 빠릅니다.

웜 데이터

자주 액세스하지 않는 데이터입니다. 이런 종류의 데이터를 쿼리할 때는 일반적으로 적절히 느린 쿼리가 허용됩니다.

창 함수

현재 레코드와 어떤 식으로든 관련된 행 그룹에서 계산을 수행하는 SQL 함수입니다. 창 함수는 이동 평균을 계산하거나 현재 행의 상대적 위치를 기반으로 행 값에 액세스하는 등의 태스크를 처리하는 데 유용합니다.

워크로드

고객 대면 애플리케이션이나 백엔드 프로세스 같이 비즈니스 가치를 창출하는 리소스 및 코드 모음입니다.

워크스트림

마이그레이션 프로젝트에서 특정 작업 세트를 담당하는 직무 그룹입니다. 각 워크스트림은 독립적이지만 프로젝트의 다른 워크스트림을 지원합니다. 예를 들어, 포트폴리오 워크스트림은 애플리케이션 우선순위 지정, 웨이브 계획, 마이그레이션 메타데이터 수집을 담당합니다. 포트폴리오 워크스트림은 이러한 자산을 마이그레이션 워크스트림에 전달하고, 마이그레이션 워크스트림은 서버와 애플리케이션을 마이그레이션합니다.

WORM

Write Once, Read Many(WORM)를 참조하세요.

WQF

AWS Workload Qualification Framework를 참조하세요.

Write Once Read Many(WORM)

데이터를 한 번 쓰고 데이터가 삭제되거나 수정되지 않도록 하는 스토리지 모델입니다. 권한 있는 사용자는 필요한 만큼 여러 번 데이터를 읽을 수 있지만 데이터를 변경할 수는 없습니다. 이 데이터 스토리지 인프라는 변경 불가능한 항목으로 간주됩니다.

Z

제로데이 익스플로잇

제로데이 취약성을 악용하는 공격(일반적으로 맬웨어)입니다.

제로데이 취약성

프로덕션 시스템의 명백한 결함 또는 취약성입니다. 위협 행위자는 이러한 유형의 취약성을 사용하여 시스템을 공격할 수 있습니다. 개발자는 공격의 결과로 취약성을 인지하는 경우가 많습니다.

제로샷 프롬프팅

태스크를 수행하기 위해 LLM에 명령을 제공하지만 안내에 도움이 되는 예제(샷)는 제공하지 않습니다. LLM은 사전 훈련된 지식을 사용하여 태스크를 처리해야 합니다. 제로샷 프롬프팅의 효과는 태스크의 복잡성과 프롬프트의 품질에 따라 달라집니다. 퓨샷 프롬프팅도 참조하세요.

좀비 애플리케이션

평균 CPU 및 메모리 사용량이 5% 미만인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하는 것이 일반적입니다.

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