aws

컨테이너 및 Amazon API Gateway를 사용하여 레거시 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화

AWS 권장 가이드

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 권장 가이드: 컨테이너 및 Amazon API Gateway를 사용하여 레거시 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

소개 1
목표 비즈니스 성과 1
ASP.NET 웹 서비스의 유형 2
스트랭글러 무화과 패턴 3
REST 기반 ASP.NET 웹 서비스 4
SOAP 기반 ASP.NET 웹 서비스(ASMX) 6
아키텍처 예 8
기타 고려 사항 10
인증 및 권한 부여 10
도메인 기반 설계 10
중간 상태 및 대상 상태 10
FAQ 12
는 Windows 컨테이너를 AWS 지원합니까? 12
Linux에서 ASP.NET 서비스를 실행할 수 있나요? 12
에서 Windows 컨테이너를 실행할 수 있습니까 AWS Fargate? 12
Amazon API Gateway는 Windows 통합 인증을 지원하나요? OAuth 2.0을 지원하나요? 12
다음 단계 및 리소스 14
문서 기록 15
..... xvi

컨테이너 및 Amazon API Gateway를 사용하여 레거시 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화

Dan Broadway, Amazon Web Services(AWS)

2021년 11월(문서 기록)

이 가이드에서는 Amazon Web Services()에서 레거시 ASP.NET 웹 서비스를 현대화하기 위해 Strangler fig 패턴을 구현하기 위한 접근 방식과 모범 사례를 설명합니다AWS. Strangler fig 패턴은 새 시스템이 결국 레거시 시스템의 모든 이전 또는 주요 부분을 대체할 때까지 레거시 시스템의 기존 기능을 점진적으로 새 애플리케이션 및 서비스로 대체하는 현대화 접근 방식입니다. 기능이 교체되면 레거시(및 종종 모놀리식) 시스템의 비활성 부분을 제거하거나 사용 중지할 수 있습니다.

이 가이드에서는 소프트웨어 시스템을 점진적으로 변경하는 일반적인 기법인 추상화 접근 방식을 서비스 위임과 함께 사용하여 수정된 브랜치를 설명합니다. 서비스 위임에서는 서비스가 서비스 소비자의 주요 변경 사항을 관리하는 데 도움이 되도록 다른 최신 서비스에 대한 동작을 연기합니다.

목표 비즈니스 성과

이 가이드의 목적은 고객이 대표 상태 전송(REST) 또는 Simple Object Access Protocol(SOAP) 표준을 기반으로 하는 레거시 ASP.NET 서비스를 현대화할 수 있도록 지원하는 것입니다 AWS. 대규모 모놀리식 애플리케이션의 분리와 관련된 전송 위험을 관리하기 위해 점진적 현대화 접근 방식이 필요한 시나리오에 중점을 둡니다. 또한 주요 변경 사항을 도입하지 않고 여러 소비자가 있는 서비스를 동기식으로 업그레이드할 수 없는 시나리오에 대해서도 설명합니다.

이 가이드는 다음을 돕기 위한 것입니다.

- 에서 ASP.NET 애플리케이션 마이그레이션, 현대화 및 확장 AWS
- 레거시 애플리케이션 현대화와 관련된 전송 위험 관리
- 고가용성 소비자가 여러 명 있는 서비스 분리

ASP.NET 웹 서비스의 유형

레거시 ASP.NET 웹 서비스를 현대화할 계획이라면 일반적으로 이러한 서비스를 두 범주로 그룹화 AWS할 수 있습니다.

- REST 기반 ASP.NET 웹 서비스. 이는 ASP.NET Web API(RESTful) 또는 ASP.NET Model-View-Controller(MVC) 패턴을 사용하여 빌드됩니다.
- SOAP 기반 ASP.NET 웹 서비스. 이는 Windows Communication Foundation(WCF) 또는 ASP.NET Web Service(ASMX) 파일과 ASP.NET WebMethods를 사용하여 구현됩니다.

REST는 현재 웹 APIs. 따라서이 접근 방식을 사용하여 구현되는 서비스는 대부분의 최신 도구 및 프레임워크와 호환되므로 보다 간단한 현대화 경로를 갖게 됩니다. 이러한 서비스의 현대화는 API 게이트웨이 및 API 관리자를 사용하여 촉진되며, 주요 관심사에는 서비스가 나타내는 리소스 및 비즈니스 도메인을 모델링하는 방법과 서비스가 보호되는 방법이 포함됩니다. 그러나 SOAP를 사용하는 웹 서비스는 일반적으로 REST 기반 서비스보다 우선하므로 현재 지원이 제한적입니다. 이러한 서비스의 마이그레이션에는 REST 기반 서비스와 동일한 문제가 수반됩니다. 또한 도구 및 프레임워크 지원이 거의 없거나 전혀 없는 추가 문제에 직면합니다.

스트랭글러 무화과 패턴

스트랭글러 무화과 패턴은 대규모 모놀리식 시스템을 현대화하거나 다시 작성할 때 위험을 관리하는 방법으로 Martin Fowler에서 도입했습니다. 패턴은 오래된 기성 나무와 함께 자라면서 목숨을 시작하는 식물 유형과 유사합니다. 덩굴이 자라면서 덩굴이 완전히 소비되고 호스트 나무를 대체하여 새 스트랭글러 무화과 나무를 그 자리에 남겨 둡니다. ASP.NET 웹 서비스를 현대화하는 맥락에서이 패턴은 다른 시스템에 웹 서비스에 대한 종속성이 있는 프록시를 설정하여 시스템 기능을 점진적으로 대체합니다. 처음에는 이러한 프록시가 기존 모놀리식 애플리케이션 서비스에서 구현되기 때문에 패스스루 동작이 있는 것으로 간주할 수 있습니다. 자연 아날로그에서 이는 스트랭글러 무화과가 처음에 호스트 트리의 트렁크를 따라 덩굴을 보내는 경우입니다. 그런 다음 모놀리스에서 분리된 새 서비스가 생성되고 프록시의 구현이 해당 새 서비스로 연기됩니다. 자연 아날로그에서 이는 스트랭글러 무화과 덩굴이 나무의 브랜치 중 하나를 둘러싸서 추월하는 경우입니다. 이 프록시 패턴은 모든 레거시 시스템의 함수가 새 서비스로 마이그레이션될 때까지 프록시 구현을 새 서비스로 대체합니다. 이 시점에서 스트랭글러 무화과 덩굴은 나무를 완전히 소비하고 레거시 시스템을 폐기할 수 있습니다.

스트랭글러 무화과 패턴을 사용할 때 다음 모범 사례를 따르면 애플리케이션을 독립적으로 확장하고 보다 원활하게 배포할 수 있습니다.

- 테스트 적용 범위가 양호하고 관련 기술 부채가 적은 구성 요소를 선택합니다. 이 구성 요소부터 시작하면 현대화 프로세스 중에 팀에 많은 확신을 줄 수 있습니다.
- 확장성 요구 사항이 있는 구성 요소를 선택하고 이러한 구성 요소 중 하나로 시작합니다.
- 비즈니스 요구 사항이 자주 변경되고 배포가 자주 발생하는 구성 요소를 선택합니다.
- 대규모로이 패턴을 구현하려면 Amazon Elastic Container Service(Amazon ECS)에서 실행 중인 Windows 컨테이너에 리팩터링된 ASMX 서비스를 AWS배포하고 Amazon API Gateway를 사용하여 현대화된 REST API를 게시합니다.

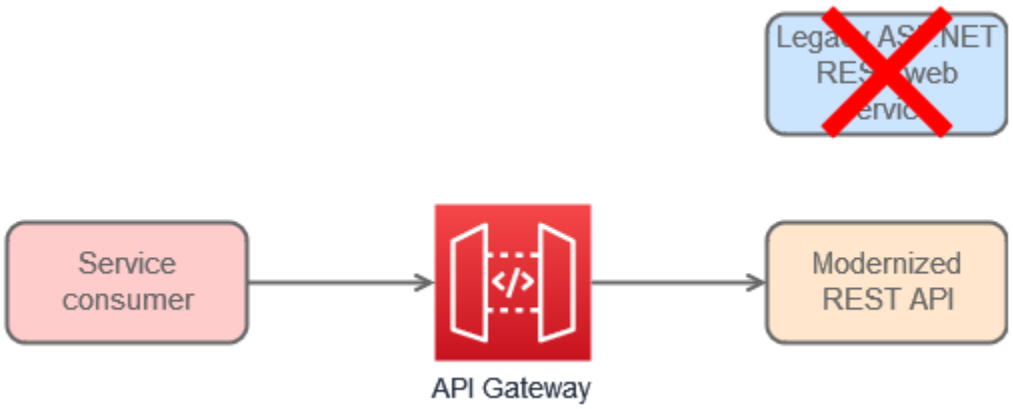
REST 기반 ASP.NET 웹 서비스

Strangler fig 패턴을 AWS 사용하여에서 REST 기반 ASP.NET 서비스를 현대화하는 경우 Amazon API Gateway를 사용하여 트래픽을 새 서비스로 전환하는 데 사용할 프록시를 설정하는 것이 좋습니다. API Gateway 엔드포인트를 서비스 소비자와 현대화 중인 레거시 서비스 간의 중개자로 도입할 수 있습니다. 레거시 서비스가 이미에 있는 경우 AWS API Gateway 엔드포인트는 요청을 레거시 REST 서비스로 라우팅하도록 구성됩니다. 서비스가 아직 켜져 있지 않으면 새 API Gateway 프록시를 설정하기 전에 그대로 마이그레이션할 AWS수 있습니다. 불가능한 경우와 같은 AWS 연결 서비스를 사용하여 API Gateway를 온프레미스 데이터 센터에 Direct Connect 연결하여 하이브리드 클라우드 접근 방식을 취할 수 있습니다. 다음 그림은 API Gateway를 도입하기 전과 후의 ASP.NET REST 서비스와 해당 소비자를 둘 사이의 프록시로 보여줍니다.

프록시를 도입하기 전에:

API Gateway를 프록시로 추가한 후:

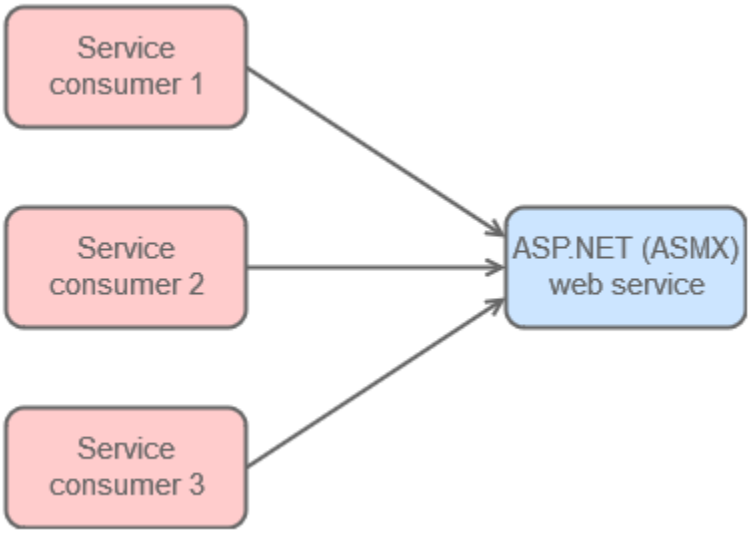
API Gateway 프록시가 마련되어 있으면 Amazon ECS를 사용하여 현대화된 서비스를 생성하고에 AWS 배포하여 확장성과 가용성이 뛰어난 서비스를 달성할 수 있습니다. 프록시와 새로 현대화된 서비스가 생성 및 테스트되면 구현을 위해 현대화된 REST API를 가리키도록 API Gateway 엔드포인트를 재구성할 수 있습니다.



새로 현대화된 서비스에 소비 시스템이 의존하는 레거시 프록시 계약과 다른 API 계약이 있는 경우 API Gateway의 데이터 변환 기능을 사용할 수 있습니다. 레거시 시스템의 스키마를 사용하여 구조화된 수신 API 요청을 매핑하고 새 서비스의 계약으로 변환할 수 있습니다.

SOAP 기반 ASP.NET 웹 서비스(ASMX)

레거시 ASP.NET 웹 서비스(ASMX)를의 REST API로 현대화하면 종속 소비자가 있을 수 있습니다. AWS종속 소비자는 관련 위험으로 인해 동시에 업그레이드할 수 없습니다. 다음 그림은 세 명의 종속 소비자가 있는 ASMX 웹 서비스를 보여줍니다.

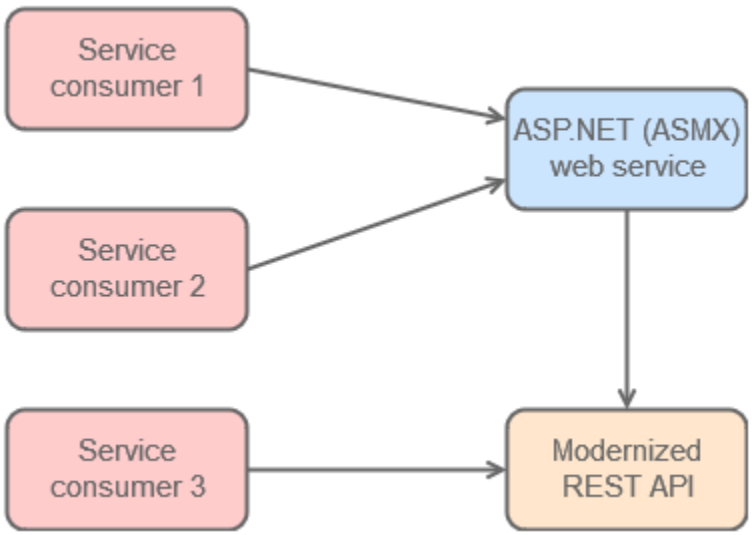


이러한 소비자는 SOAP 프로토콜을 사용하여 ASMX 서비스를 호출하기 때문에 ASMX 서비스를 REST API로 대체하면 이러한 소비자가 중단됩니다. 또한 시스템을 모놀리식 아키텍처에서 마이크로서비스로 변환하는 동안 ASMX 서비스를 REST API로 리팩터링하려면 시스템 동작 및 도메인 모델을 리모델링해야 할 가능성이 높습니다. 이로 인해 소비자에게 추가적인 주요 변경 사항이 발생할 수 있습니다.

시스템을 점진적으로 현대화할 수 있도록 이전 서비스와의 이전 버전과의 호환성을 유지하기 위해 Strangler fig 접근 방식의 지침을 구현할 수 있습니다. 스트랭글러 무화과 접근 방식에서 중요한 레거시 시스템 현대화와 관련된 위험은 부분적으로 레거시 시스템을 새 시스템으로 점진적으로 교체하여 관리됩니다.

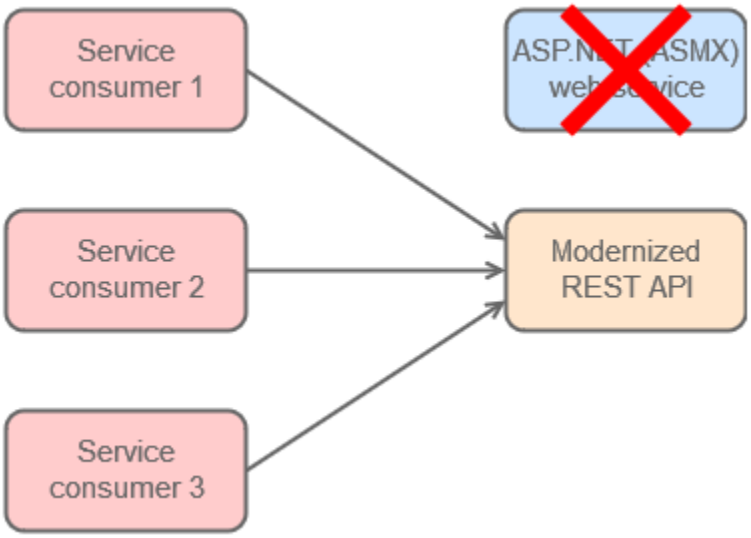
추상화를 통한 분기는 스트랭글러 무화과 패턴을 구현하는 효과적인 기법인 Martin Fowler가 도입한 또 다른 접근 방식입니다. 스트랭글러 무화과 접근 방식과 마찬가지로 추상화를 통한 브랜치는 주요 기능을 교체하거나 현대화해야 할 때 엔지니어가 시스템을 점진적으로 변경하는 데 도움이 됩니다. 이 접근 방식에서는 특정 기능을 구현하는 코드와 해당 기능을 사용하는 코드(즉, 클라이언트 코드) 사이에 추상화 계층이 도입됩니다. 이 추상화 계층은 구체적인 클래스에서 상속할 수 있는 추상적 유형일 필요는 없습니다. 대신 소비자와 공급자 간의 기능적 계약을 캡처하는 구현일 수 있습니다.

레거시 SOAP 기반 ASP.NET 웹 서비스의 경우 추상화 기법으로 브랜치에서 영감을 받은 접근 방식을 통해 호환성을 유지하여 서비스 소비자를 이전에서 새 로 점진적으로 마이그레이션할 수 있습니다. 그러나 SOAP 기반 ASP.NET 웹 서비스에 대한 접근 방식은 상속 대신 위임을 사용합니다. ASMX 소비자와의 호환성은 구현을 새 REST API에 위임하도록 레거시 ASMX 서비스를 리팩터링하여 달성됩니다. 다음 그림은 구현을 새 REST API에 위임하는 레거시 ASMX 서비스를 보여줍니다. 이 위임을 통해 서비스 소비자 3은 새 REST API로 업그레이드할 수 있고 서비스 소비자 1과 2는 독립적으로 업그레이드됩니다.



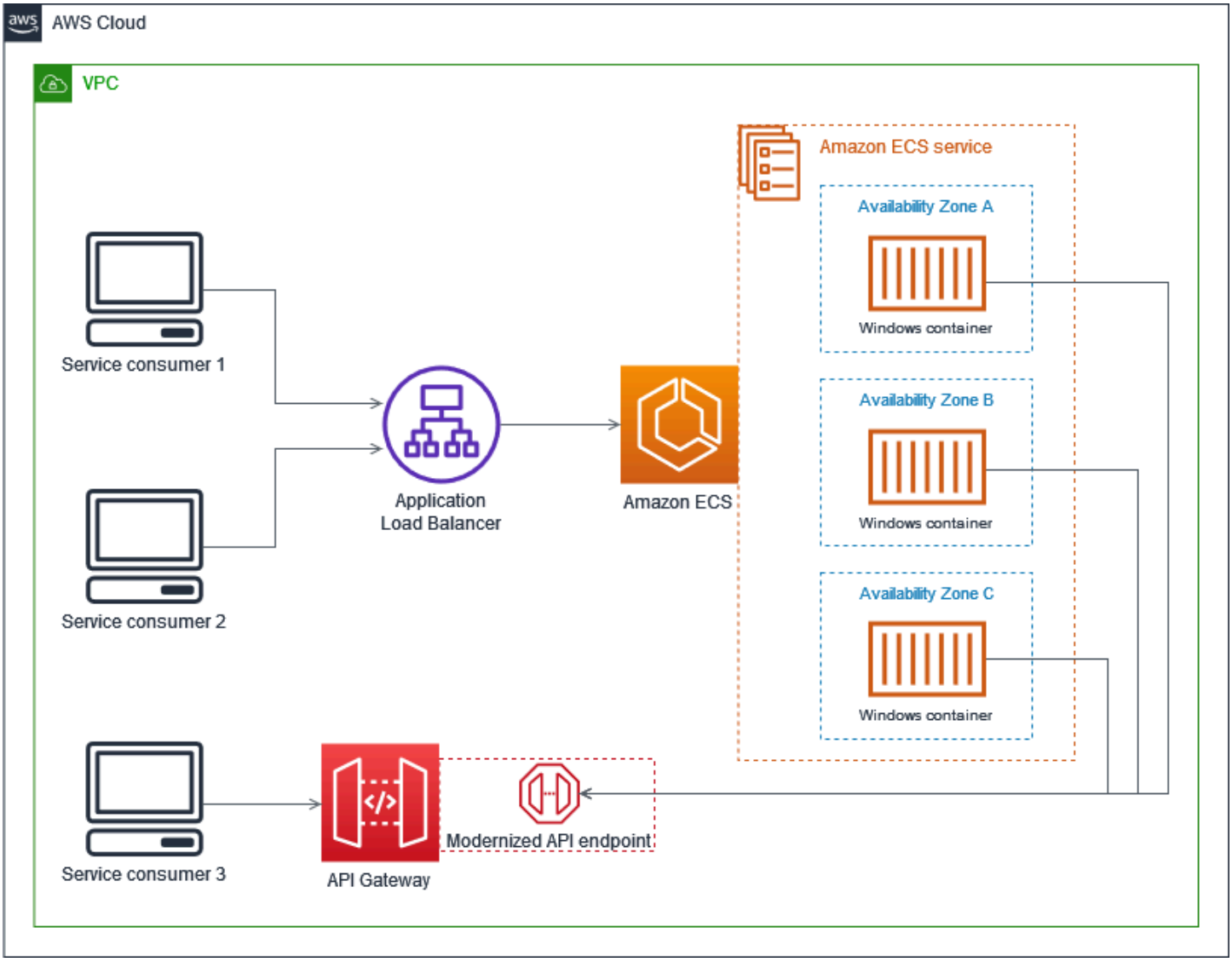
ASMX 서비스 리팩터링의 특성은 해당 서비스의 기능 및 비기능(교차 기능) 요구 사항에 따라 달라집니다. 경우에 따라 리팩터링에는 이전 인증 및 권한 부여 메커니즘에서 새 인증 및 권한 부여 메커니즘으로 변환하고, 서비스 ASMX 페이로드를 XML에서 JSON으로 변환한 다음, 변환된 페이로드를 사용하여 REST API를 호출하는 작업이 포함될 수 있습니다. 보다 복잡한 시나리오에서는 리팩터링된 ASMX 서비스가 여러 REST APIs하고 상태를 유지해야 할 수 있습니다.

서비스 소비자를 레거시 ASMX 서비스에서 현대화된 REST API로 마이그레이션하는 프로세스는 모든 서비스 소비자가 새 API를 참조하도록 업데이트될 때까지 점진적으로 계속됩니다. 다음 그림은 모든 서비스 소비자가 REST API를 참조하고 더 이상 필요하지 않아 ASMX 서비스가 폐기된 완전히 현대화된 상태를 보여줍니다.



아키텍처 예

SOAP 기반 ASMX 서비스의 최종 상태 아키텍처는 현대화된 REST API를 사용하는 모든 서비스 소비자를 표시하므로 레거시 ASMX 웹 서비스가 폐기됩니다. 그러나이 패턴의 초점은 최종 상태 아키텍처가 아니라 마이그레이션 중에 소비자가 업그레이드되는 동안 필요한 중간 상태 아키텍처입니다. 이 중간 상태는 다음 다이어그램에 나와 있습니다. 이 임시 아키텍처에서 서비스 소비자 1과 2는 구현을 현대화된 REST API에 위임하기 위해 리팩터링된 레거시 ASMX 서비스를 계속 사용하고 있습니다. 리팩터링된 Windows 서비스는 Amazon ECS에서 Windows 컨테이너로 배포되고, 고가용성을 위해 3개의 가용 영역에 구성되며, Application Load Balancer를 통해 액세스됩니다. 서비스 소비자 3은 API Gateway를 통해 직접 새 REST API를 사용하도록 업데이트되었습니다.


AWS Cloud
VPC
Amazon ECS service
Availability Zone A
Windows container
Availability Zone B
Windows container
Availability Zone C
Windows container
Service consumer 1
Service consumer 2
Service consumer 3
Application Load Balancer
Amazon ECS
API Gateway
Modernized API endpoint

기타 고려 사항

지금까지 API Gateway 및 Windows 컨테이너를 사용하여 레거시 ASP.NET 웹 서비스를 현대화하는 방법에 대해 설명했습니다 AWS. 고려해야 할 몇 가지 다른 고려 사항은 다음과 같습니다.

- 보안
- 서비스 도메인 리모델링
- 현대화할 서비스가 많은 경우 웹 서비스 업그레이드 순서 지정

다음 단원에서 각각에 대해 살펴 봅니다.

인증 및 권한 부여

최신 APIs 일반적으로 OAuth 2.0 및 OpenID Connect와 같은 토큰 기반(JSON 웹 토큰 또는 JWT) 인증 및 권한 부여 표준을 활용하는 반면, 레거시 ASP.NET 웹 서비스는 전통적으로 Windows Active Directory 도메인에 대한 기본 인증 또는 Windows 통합 인증에 의존했습니다. 모범 사례로, 이전 및 새로운 인증 및 권한 부여 접근 방식이 호환되지 않는 경우 웹 서비스를 현대화하기 전에 이러한 보안 메커니즘을 업그레이드하는 것이 좋습니다 AWS. 자격 증명을 매핑하거나 이전 시스템과 새 시스템 간에 보안 상태를 안전하게 전송하려고 시도하는 것은 큰 노력이 아닐 수 있지만 보안 취약성이 발생할 수 있습니다.

도메인 기반 설계

모놀리스를 개별 서비스로 나눌 때 DDD(도메인 기반 설계)는 시스템을 응집력 있는 비즈니스 도메인으로 모델링하는 데 자주 사용되는 모범 사례입니다. DDD는 구현을 진화하는 핵심 비즈니스 개념 모델에 연결하여 복잡한 요구 사항에 맞는 소프트웨어를 개발하는 접근 방식입니다. 그 전제는 프로젝트의 주요 초점을 핵심 도메인 및 도메인 로직에 두고, 비즈니스를 반영하는 모델을 기반으로 시스템 설계를 설계하는 것입니다. 기존 모놀리식 애플리케이션을 현대화할 때 DDD를 사용하는 경우 모놀리식의 기능 및 사용자 흐름에서 역방향으로 작업해야 합니다. DDD를 스트랭글러 패턴과 함께 사용하여 모놀리스를 깨고 스트랭글링할 서비스를 결정하는 프로세스를 안내할 수 있습니다. 따라서 도메인 기반 설계라는 용어가 사용됩니다.

중간 상태 및 대상 상태

몇 개 이상의 ASP.NET 웹 서비스를 현대화하는 경우 AWS먼저 모든 서비스가 현대화된 후 대상 상태 아키텍처를 정의하는 것이 좋습니다. 그러나 비즈니스 컨텍스트가 자주 변경되고 비즈니스 목표에 맞

게 유지하려면 필요에 따라 시스템 대상 상태를 업데이트해야 하므로 대상 상태 아키텍처가 반드시 최종 상태 또는 최종 상태일 필요는 없습니다. 대상 상태를 정의한 경우 거기에서 역방향으로 작업하여 대상 상태 비전을 점진적으로 이행하는 중간 상태 아키텍처를 정의할 수 있습니다. 이러한 중간 상태 아키텍처는 호스트 트리의 주요 부분을 접하고 묶을 때 스트랭글러 무화과가 나무를 따라 이동하는 것으로 생각할 수 있습니다. 중간 상태 아키텍처는 대상 상태로의 진화를 촉진하기 위해 최종 상태 아키텍처의 일부가 아닌 임시 또는 중첩 구문을 도입하는 경우가 많습니다. SOAP 기반 ASP.NET 웹 서비스의 현대화는 예제를 제공합니다. Windows 기반 컨테이너는 새 REST API로 업그레이드할 수 있을 때까지 레거시 서비스에 의존하는 호출 시스템 간의 격차를 메우기 위해 일시적으로 도입됩니다.

FAQ

이 섹션에서는 레거시 ASP.NET 웹 서비스 현대화에 대해 일반적으로 제기되는 질문에 대한 답변을 제공합니다.

는 Windows 컨테이너를 AWS 지원합니까?

예, Amazon Elastic Container Service(Amazon ECS)와 Amazon Elastic Kubernetes Service(Amazon EKS) 모두 Windows 컨테이너를 지원합니다. Windows 컨테이너를 사용하여 이러한 AWS 서비스를 시작하려면 다음을 참조하세요.

- Amazon EC2 Windows 컨테이너를 사용하여 Amazon ECS 콘솔 시작하기(Amazon ECS 설명서)
- Amazon EKS Windows 컨테이너 지원 현재 정식 출시(AWS 뉴스 블로그)

Linux에서 ASP.NET 서비스를 실행할 수 있나요?

예, .NET Core로 작성된 ASP.NET 서비스(APIs Linux에서 실행할 수 있습니다. 를 사용하여 Linux에서 .NET Core 애플리케이션을 실행하려면 Elastic Beanstalk 설명서의 Linux에서 .NET Core 작업을 AWS Elastic Beanstalk참조하세요.

에서 Windows 컨테이너를 실행할 수 있습니까 AWS Fargate?

이 안내서 작성부터 Windows 컨테이너는 Amazon EC2 시작 유형에 대해서만 지원됩니다. Fargate 시작 유형은 현재 Windows 컨테이너에서 지원되지 않습니다. 자세한 내용은 Amazon ECS 설명서의 Amazon EC2 Windows 컨테이너를 참조하세요.

Amazon API Gateway는 Windows 통합 인증을 지원하나요? OAuth 2.0을 지원하나요?

기본적으로 그렇지 않습니다. 그러나 API Gateway는 Amazon Cognito 사용자 풀을 사용하여 OAuth 2.0을 지원하고 사용자 지정 권한 부여자를 사용하여 다른 인증 및 권한 부여 공급자를 지원합니다. 이러한 옵션을 시작하려면 다음을 참조하세요.

- Amazon API Gateway의 사용자 지정 권한 부여자 소개(AWS 컴퓨팅 블로그)

- Amazon Cognito 사용자 풀을 권한 부여자로 사용하여 REST API에 대한 액세스 제어(API Gateway 설명서)

다음 단계 및 리소스

레거시 ASP.NET 애플리케이션의 마이그레이션 및 현대화 프로세스를 시작하려면:

1. 마이그레이션할 ASP.NET 웹 서비스의 유형을 결정합니다 AWS. 현대화 관점에서 ASP.NET 웹 서비스는 세 가지 범주로 분류할 수 있습니다. 이 안내서에서 설명하는 처음 두 범주는 ASMX 파일로 작성되거나 Windows Communication Foundation(WCF)을 사용하여 작성된 SOAP 기반 ASP.NET 웹 서비스와 RESTful ASP.NET 웹 APIs 또는 ASP.NET Model-View-Controller(MVC) 패턴을 사용하여 작성된 REST 기반 ASP.NET 서비스입니다. RESTful 세 번째 유형의 ASP.NET 웹 서비스는 ASP.NET MVC 및 .NET Core 또는 .NET 5를 사용하여 작성됩니다. 이는 ASP.NET REST APIs를 작성하는 현재 권장 방법이며 마이그레이션에 상당한 현대화가 필요하지 않습니다 AWS.
2. 인증 및 권한 부여를 위해 이러한 서비스를 보호하는 방법을 결정하고, 접근 방식을 설정하고, 현대화 작업의이 측면을 우선시합니다.
3. Windows 컨테이너, Amazon ECS 및 Amazon API Gateway에 대해 알아봅니다.
4. 도메인 기반 설계를 읽어보세요. 이렇게 하면 현대화된 서비스가 레거시 시스템의 설계 대신 일관된 기능 도메인을 중심으로 모델링되도록 하여 현대화 노력을 최대한 활용할 수 있습니다.
5. 대상 상태 아키텍처를 설정한 다음 역방향으로 작업하여 해당 결과를 달성하는 데 필요한 중간 상태 아키텍처를 결정합니다.

이 가이드에서 설명하는 패턴에 대한 자세한 내용은 Martin Fowler의 다음 문서를 참조하세요.

- Strangler 무화과 애플리케이션
- 추상화별 분기
- 도메인 기반 설계

이 안내서에서 설명하는 AWS 서비스에 대한 자세한 내용은 다음 참조를 참조하세요.

- Amazon Elastic Container Service(Amazon ECS)
- Amazon API Gateway

문서 기록

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

변경 사항	설명	날짜
최초 게시	—	2021년 11월 15일

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