



사용자 가이드

AWS App Mesh



AWS App Mesh: 사용자 가이드

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 브랜드 디자인은 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

AWS App Mesh란?	1
예제 애플리케이션에 App Mesh 추가	1
App Mesh 구성 요소	2
시작하는 방법	3
App Mesh 액세스	4
시작하기	5
App Mesh와 아마존 ECS	5
시나리오	5
사전 조건	6
1단계: 메시 및 가상 서비스 생성	6
2단계: 가상 노드 생성	7
3단계: 가상 라우터 생성 및 라우팅	9
4단계: 검토 및 생성	11
5단계: 추가 리소스 생성	12
6단계: 서비스 업데이트	17
고급 주제	33
App Mesh 및 Kubernetes	34
사전 조건	34
1단계: 통합 구성 요소 설치	35
2단계: App Mesh 리소스 배포	41
3단계: 서비스 생성 또는 업데이트	55
4단계: 정리	61
App Mesh와 아마존 EC2	62
시나리오	5
사전 조건	6
1단계: 메시 및 가상 서비스 생성	63
2단계: 가상 노드 생성	64
3단계: 가상 라우터 생성 및 라우팅	9
4단계: 검토 및 생성	11
5단계: 추가 리소스 생성	12
6단계: 서비스 업데이트	17
App Mesh 예제	84
개념	85
메시	85

서비스 메시 생성	85
메시 삭제	88
가상 서비스	89
가상 서비스 생성	89
가상 서비스 삭제	92
가상 게이트웨이	93
가상 게이트웨이 생성	94
가상 게이트웨이 배포	99
가상 게이트웨이 삭제	100
게이트웨이 경로	101
가상 노드	107
가상 노드 생성	108
가상 노드 삭제	117
가상 라우터	119
가상 라우터 생성	120
가상 라우터 삭제	122
경로	124
Envoy	135
Envoy 이미지 변형	135
Envoy 구성 변수	139
필수 변수	140
선택적 변수	140
App Mesh에서 설정한 Envoy 기본값	147
기본 경로 재시도 정책	148
기본 회로 차단기	149
Envoy 1.17로 업데이트/마이그레이션	149
시크릿 디스커버리 서비스 포함 SPIRE	149
정규식 변경 사항	149
역참조	152
Agent for Envoy	153
관찰성	155
로깅	155
Firelens 및 Cloudwatch	157
Envoy 지표	157
예제 애플리케이션 지표	160
지표 내보내기	164

추적	171
X-Ray	172
Jaeger	174
추적용 Datadog	144
도구	176
AWS CloudFormation	176
AWS CDK	176
Kubernetes용 App Mesh 컨트롤러	176
Terraform	177
공유 메시 작업	178
메시 공유 권한 부여	178
메시 공유 권한 부여	178
메시에 권한 부여	178
메시 공유를 위한 사전 조건	180
관련 서비스	180
메시 공유	181
공유 메시의 공유 해제	181
공유 메시 식별	182
결제 및 측정	182
인스턴스 할당량	183
다른 서비스와 함께 사용	184
AWS CloudFormation을 사용하여 App Mesh 리소스 생성	184
App Mesh 및 AWS CloudFormation 템플릿	184
AWS CloudFormation에 대해 자세히 알아보기	184
AWS Outpost의 App Mesh	185
필수 조건	185
제한 사항	185
네트워크 연결 고려 사항	185
Outpost에서 App Mesh Envoy 프록시 생성	186
모범 사례	187
재시도를 통해 모든 경로 계속	187
배포 속도 조정	188
스케일 인 전에 스케일 아웃	188
컨테이너 상태 검사 구현	189
DNS 해상도 최적화	189
애플리케이션 보호	191

전송 계층 보안 (TLS)	192
인증서 요구 사항	192
TLS인증 인증서	193
App Mesh가 협상을 위해 Envoy를 구성하는 방법 TLS	195
암호화 확인	196
인증서 갱신	197
TLS인증을 사용하도록 Amazon ECS 워크로드를 구성합니다. AWS App Mesh	197
인증을 사용하도록 Kubernetes 워크로드를 구성합니다. TLS AWS App Mesh	198
상호 TLS 인증	199
상호 TLS 인증 인증서	199
메시 엔드포인트 구성	200
상호 TLS 인증으로 서비스 마이그레이션	200
상호 TLS 인증 확인	201
App Mesh 상호 TLS 인증 살펴보기	202
자격 증명 및 액세스 관리	202
고객	202
ID를 통한 인증	203
정책을 사용한 액세스 관리	206
수정할 수 있다면 방법이 무엇입니까? AWS App Mesh 다음과 함께 작동합니다. IAM	208
자격 증명 기반 정책 예제	212
AWS 관리형 정책	216
서비스 연결 역할 사용	219
Envoy 프록시 권한 부여	222
문제 해결	227
CloudTrail 로그	228
의 App Mesh 관리 이벤트 CloudTrail	229
App Mesh 이벤트 예제	230
데이터 보호	231
데이터 암호화	231
규정 준수 확인	232
인프라 보안	233
인터페이스 VPC 엔드포인트(AWS PrivateLink)	234
복원성	235
AWS App Mesh의 재해 복구	236
구성 및 취약성 분석	236
문제 해결	237

모범 사례	237
Envoy 프록시 관리 인터페이스 활성화	237
지표 오프로드를 위한 Envoy DogStats D 통합 활성화	238
액세스 로그 활성화	238
사전 프로덕션 환경에서 Envoy 디버그 로깅 활성화	238
App Mesh 제어 영역을 사용하여 Envoy 프록시 연결 모니터링	239
설치	239
Envoy 컨테이너 이미지를 끌어올 수 없음	239
App Mesh Envoy Management Service에 연결할 수 없음	240
Envoy가 오류 텍스트를 나타내며 App Mesh Envoy Management Service에서 연결이 끊김 ..	241
Envoy 컨테이너 상태 확인, 준비 상태 프로브 또는 활성화 프로브 실패	243
로드 밸런서에서 메시 엔드포인트로의 상태 확인이 실패함	244
가상 게이트웨이는 포트 1024 이하에서 트래픽을 허용하지 않음	245
연결	246
가상 서비스의 DNS 이름을 확인할 수 없음	246
가상 서비스 백엔드에 연결할 수 없음	246
외부 서비스에 연결할 수 없음	248
MySQL 또는 SMTP 서버에 연결할 수 없음	248
App Mesh에서 TCP 가상 노드 또는 가상 라우터로 모델링된 서비스에 연결할 수 없음	249
가상 노드의 가상 서비스 백엔드로 나열되지 않은 서비스에 성공적으로 연결됨	250
가상 서비스에 가상 노드 공급자가 있는 경우 일부 요청은 HTTP 상태 코드 503을 나타내며 실패합니다.	251
Amazon EFS 파일 시스템에 연결할 수 없음	251
서비스에 성공적으로 연결되지만 수신 요청은 Envoy의 액세스 로그, 추적 또는 지표에 나타 나지 않음	252
컨테이너 수준에서 HTTP_PROXY/HTTPS_PROXY 환경 변수를 설정해도 예상대로 작동하지 않습니다.	252
경로 제한 시간을 설정한 후에도 업스트림 요청 제한 시간이 초과됩니다.	253
Envoy는 HTTP 잘못된 요청으로 응답합니다.	254
제한 시간을 제대로 구성할 수 없습니다.	254
스케일링	255
가상 노드/가상 게이트웨이의 복제본을 50개 이상으로 스케일링할 경우 연결이 실패하고 컨 테이너 상태 확인이 실패합니다.	255
가상 서비스 백엔드가 수평적으로 스케일 아웃되거나 스케일 인될 때 503을 나타내며 요청이 실패함	255
로드가 증가하면 Envoy 컨테이너가 segfault를 나타내며 충돌함	256

기본 리소스 증가는 서비스 제한에 반영되지 않습니다. 256

방대한 상태 확인 호출로 인해 애플리케이션이 충돌합니다. 256

관찰성 257

 내 애플리케이션에 대한 AWS X-Ray 흔적을 볼 수 없습니다. 257

 Amazon 메트릭에서 내 애플리케이션에 대한 Envoy 메트릭을 볼 수 없습니다.
 CloudWatch 258

 AWS X-Ray 추적에 대한 사용자 지정 샘플링 규칙을 구성할 수 없음 258

보안 260

 TLS 클라이언트 정책을 사용하여 백엔드 가상 서비스에 연결할 수 없음 260

 애플리케이션에서 TLS를 시작할 때 백엔드 가상 서비스에 연결할 수 없음 261

 Envoy 프록시 간의 연결이 TLS를 사용하고 있음을 어설션할 수 없음 261

 Elastic Load Balancing을 통한 TLS 문제 해결 263

Kubernetes 264

 Kubernetes에서 생성된 App Mesh 리소스를 App Mesh에서 찾을 수 없음 264

 Envoy 사이드카를 삽입한 후 포드의 준비 상태 및 활성화 상태 확인이 실패함 265

 포드가 AWS Cloud Map 인스턴스로 등록 또는 등록 취소되지 않음 265

 App Mesh 리소스에 대한 포드가 실행 중인 위치를 확인할 수 없음 266

 포드가 어떤 App Mesh 리소스로 실행되고 있는지 확인할 수 없음 266

 클라이언트 Envoy는 IMDSv1이 비활성화된 상태에서 App Mesh Envoy Management Service
 와 통신할 수 없습니다. 267

 App Mesh가 활성화되고 Envoy가 삽입된 경우 IRSA가 애플리케이션 컨테이너에서 작동하지
 않음 267

프리뷰 채널 269

서비스 할당량 273

사용 설명서 기록 274

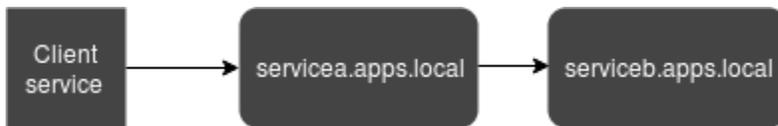
..... cclxxx

AWS App Mesh란?

AWS App Mesh는 서비스를 손쉽게 모니터링하고 제어할 수 있는 서비스 메시입니다. 서비스 메시는 일반적으로 애플리케이션 코드와 함께 배포된 간단한 네트워크 프록시 배열을 통해 서비스 간 통신을 처리하는 전용 인프라 계층입니다. App Mesh는 서비스가 통신하는 방법을 표준화하여 엔드 투 엔드 가시성을 제공하고 애플리케이션의 고가용성을 보장합니다. App Mesh는 애플리케이션의 모든 서비스에 대한 일관된 가시성 및 네트워크 트래픽 제어를 제공합니다.

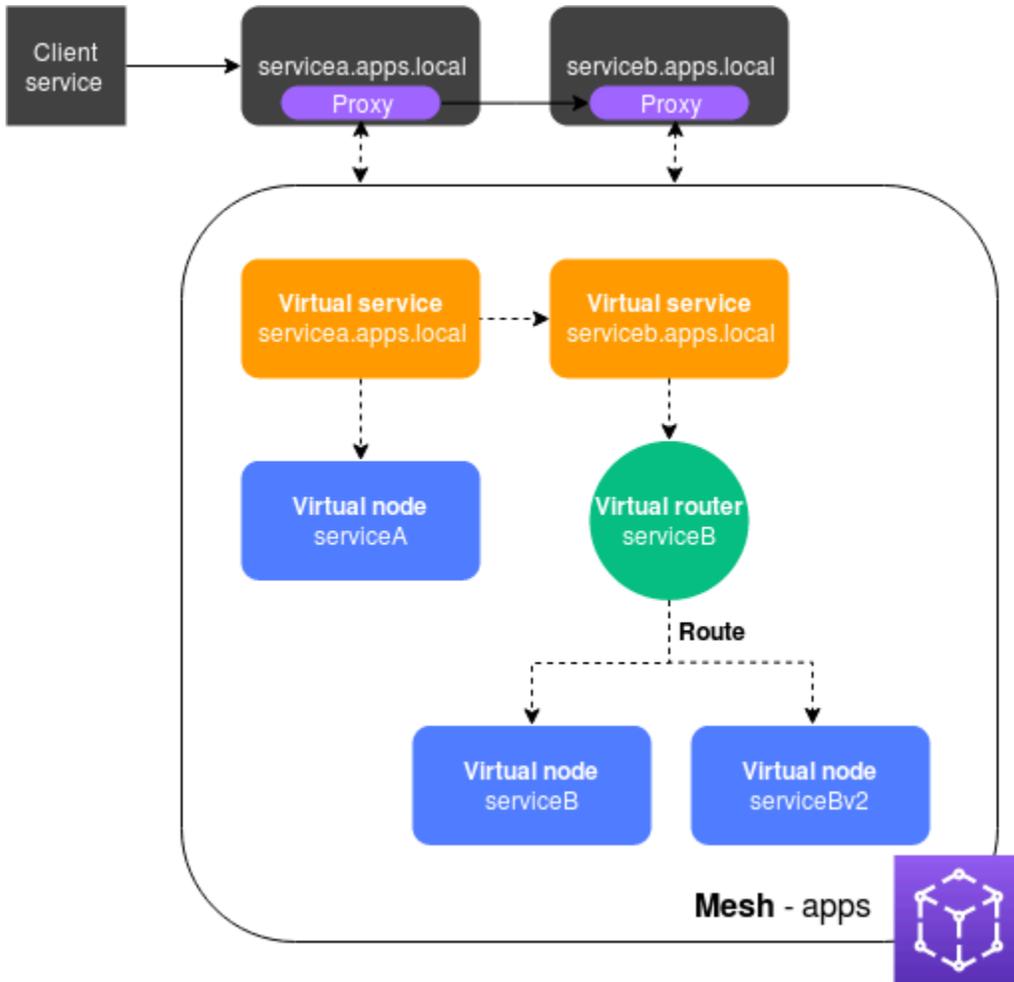
예제 애플리케이션에 App Mesh 추가

App Mesh를 사용하지 않는 다음과 같은 간단한 예제 애플리케이션을 고려해 보세요. 두 서비스는 AWS Fargate, Amazon Elastic Container Service(Amazon ECS), Amazon Elastic Kubernetes Service(Amazon EKS), Amazon Elastic Compute Cloud(Amazon EC2)의 Kubernetes 인스턴스 또는 Docker가 있는 Amazon EC2 인스턴스에서 실행할 수 있습니다.



이 그림에서는 `apps.local` 네임스페이스를 통해 `serviceA` 및 `serviceB`를 모두 검색할 수 있습니다. 예를 들어 이름이 `servicebv2.apps.local`인 새 버전의 `serviceb.apps.local`을 배포하기로 결정했다고 가정해 보겠습니다. 다음으로, `servicea.apps.local`의 트래픽 특정 비율을 `serviceb.apps.local`로, 특정 비율을 `servicebv2.apps.local`로 전송하려고 합니다. `servicebv2`의 성능이 좋다고 확신하는 경우 트래픽의 100%를 해당 위치로 보내는 것이 좋습니다.

App Mesh를 사용하면 애플리케이션 코드나 등록된 서비스 이름을 변경하지 않고도 이 작업을 수행할 수 있습니다. 이 예제 애플리케이션에서 App Mesh를 사용하는 경우 메시는 다음 그림과 같아 보일 수 있습니다.



이 구성에서는 서비스가 더 이상 서로 직접 통신하지 않습니다. 대신 프록시를 통해 서로 통신합니다. `servicea.apps.local` 서비스와 함께 배포된 프록시는 App Mesh 구성을 읽고 구성에 따라 `serviceb.apps.local` 또는 `servicebv2.apps.local`로 트래픽을 전송합니다.

App Mesh 구성 요소

App Mesh는 이전 예제와 같이 다음 구성 요소로 이루어집니다.

- 서비스 메시 – 서비스 메시는 내부에 있는 서비스 간의 네트워크 트래픽에 대한 논리적 경계입니다. 이 예제에서 메시는 이름이 `apps`이며 메시에 대한 다른 모든 리소스를 포함합니다. 자세한 내용은 [서비스 메시](#) 섹션을 참조하세요.
- 가상 서비스 – 가상 서비스는 가상 노드가 가상 라우터를 통해 직접 또는 간접적으로 제공하는 실제 서비스의 추상화입니다. 그림에서 두 개의 가상 서비스는 두 개의 실제 서비스를 나타냅니다. 가상 서비스 이름은 실제 서비스에서 검색 가능한 이름입니다. 가상 서비스와 실제 서비스의 이름이 같으

면 여러 서비스가 App Mesh가 구현되기 전에 사용한 것과 동일한 이름을 사용하여 서로 통신할 수 있습니다. 자세한 내용은 [가상 서비스](#) 섹션을 참조하세요.

- 가상 노드 — 가상 노드는 Amazon ECS 또는 Kubernetes 서비스와 같은 검색 가능한 서비스에 대한 논리 포인터 역할을 합니다. 각 가상 서비스에는 하나 이상의 가상 노드가 있습니다. 그림에서 `servicea.apps.local` 가상 서비스는 이름이 `serviceA`인 가상 노드에 대한 구성 정보를 가져옵니다. `serviceA` 가상 노드는 서비스 검색을 위한 `servicea.apps.local` 이름으로 구성됩니다. `serviceb.apps.local` 가상 서비스는 `serviceB`라는 가상 라우터를 통해 `serviceB` 및 `serviceBv2` 가상 노드로 트래픽을 라우팅하도록 구성됩니다. 자세한 내용은 [가상 노드](#) 섹션을 참조하세요.
- 가상 라우터 및 경로 - 가상 라우터는 메시 내에 있는 하나 이상의 가상 서비스에 대한 트래픽을 처리합니다. 경로는 가상 라우터와 연결됩니다. 이 경로는 가상 라우터에 대해 일치하는 요청을 찾고 관련 가상 노드에 트래픽을 분산하는 데 사용됩니다. 이전 그림에서 `serviceB` 가상 라우터에는 트래픽의 일정 비율을 `serviceB` 가상 노드로, 일정 비율을 `serviceBv2` 가상 노드로 전달하는 경로가 있습니다. 특정 가상 노드로 라우팅되는 트래픽의 비율을 설정하고 시간이 지남에 따라 변경할 수 있습니다. HTTP 헤더, URL 경로, gRPC 서비스 및 메서드 이름과 같은 기준에 따라 트래픽을 라우팅할 수 있습니다. 응답에 오류가 있는 경우 연결을 재시도하도록 재시도 정책을 구성할 수 있습니다. 예를 들어, 그림에서 경로에 대한 재시도 정책은 `serviceb.apps.local`에서 특정 유형의 오류를 반환하는 경우 `serviceb.apps.local`에 대한 연결을 5번 재시도하고 재시도 간격은 10초가 되도록 지정할 수 있습니다. 자세한 정보는 [가상 라우터 및 경로](#) 섹션을 참조하세요.
- 프록시 - 메시와 해당 리소스를 생성한 후 프록시를 사용하도록 서비스를 구성합니다. 프록시는 App Mesh 구성을 읽고 트래픽을 적절하게 전달합니다. 그림에서 `servicea.apps.local`에서 `serviceb.apps.local`로의 모든 통신은 각 서비스에 배포된 프록시를 통해 이루어집니다. 서비스는 App Mesh를 도입하기 전에 사용한 것과 동일한 서비스 검색 이름을 사용하여 서로 통신합니다. 프록시가 App Mesh 구성을 읽으므로 두 서비스가 서로 통신하는 방식을 제어할 수 있습니다. App Mesh 구성을 변경하려는 경우 서비스 자체 또는 프록시를 변경하거나 재배포할 필요가 없습니다. 자세한 내용은 [Envoy 이미지](#) 섹션을 참조하세요.

시작하는 방법

App Mesh를 사용하려면 AWS Fargate, Amazon ECS, Amazon EKS, Amazon EC2의 Kubernetes 또는 Docker가 있는 Amazon EC2에서 실행되는 기존 서비스가 있어야 합니다.

App Mesh를 시작하려면 다음 가이드 중 하나를 참조하세요.

- [App Mesh 및 Amazon ECS 시작하기](#)
- [App Mesh 및 Kubernetes 시작하기](#)

- [App Mesh 및 Amazon EC2 시작하기](#)

App Mesh 액세스

다음과 같은 방식으로 App Mesh 작업을 수행할 수 있습니다.

AWS Management Console

콘솔은 App Mesh 리소스를 관리하는 데 사용할 수 있는 브라우저 기반 인터페이스입니다. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 열 수 있습니다.

AWS CLI

다양한 AWS 제품에서 사용되는 명령어를 제공하며 Windows, Mac, Linux를 지원합니다. 시작하려면 [AWS Command Line Interface 사용 설명서](#)를 참조하세요. App Mesh 명령에 대한 자세한 내용은 [AWS CLI 명령 참조](#)에서 [appmesh](#)를 참조하세요.

AWS Tools for Windows PowerShell

PowerShell 환경에서 스크립트 작업을 선호하는 다양한 AWS 제품을 관리할 수 있도록, 명령줄 도구를 제공합니다. 시작하려면 [AWS Tools for Windows PowerShell 사용 설명서](#)를 참조하세요. App Mesh용 cmdlets에 대한 자세한 내용은 [PowerShell Cmdlet용 AWS 도구 참조](#)의 [App Mesh](#)를 참조하세요.

AWS CloudFormation

원하는 모든 AWS 리소스를 설명하는 템플릿을 생성할 수 있도록 합니다. AWS CloudFormation에서는 템플릿을 사용하여 리소스를 프로비저닝하고 구성합니다. 시작하려면 [AWS CloudFormation 사용 설명서](#)를 참조하세요. App Mesh 리소스 유형에 대한 자세한 내용은 [AWS CloudFormation 템플릿 참조](#)의 [App Mesh 리소스 유형 참조](#)를 참조하세요.

AWS SDK

SDK를 사용해 다양한 프로그래밍 언어로 App Mesh에 액세스할 수 있습니다. SDK는 다음 작업들을 자동으로 수행합니다.

- 서비스 요청에 대한 암호화 서명
- 요청 재시도
- 오류 응답 처리

사용 가능한 SDK에 대한 자세한 내용은 [Amazon Web Services용 도구](#) 단원을 참조하세요.

App Mesh API에 대한 자세한 내용은 [AWS App Mesh API 참조](#)를 참조하세요.

App Mesh 시작하기

AmazonECS, Kubernetes (자체 Amazon EC2 인스턴스에 배포하거나 Amazon에서 실행 중인 애플리케이션EKS) 및 Amazon에 배포하는 애플리케이션과 함께 App Mesh를 사용할 수 있습니다. EC2 App Mesh를 시작하려면 애플리케이션이 배포된 서비스 중에서 App Mesh에서 사용하려는 서비스를 하나 선택합니다. 시작하기 설명서 중 하나를 완료한 후에는 언제든지 다른 서비스의 애플리케이션이 App Mesh에서 작동하도록 설정할 수 있습니다.

주제

- [아마존 시작하기 AWS App Mesh ECS](#)
- [AWS App Mesh 쿠버네티스 시작하기](#)
- [아마존 시작하기 AWS App Mesh EC2](#)
- [App Mesh 예제](#)

아마존 시작하기 AWS App Mesh ECS

이 주제는 Amazon에서 실행 중인 실제 서비스와 AWS App Mesh 함께 사용할 수 있도록 도와줍니다 ECS. 이 자습서에서는 몇 가지 App Mesh 리소스 유형의 기본 기능을 다룹니다.

시나리오

App Mesh를 사용하는 방법을 설명하기 위해 다음과 같은 특징을 가진 애플리케이션이 있다고 가정합니다.

- serviceA 및 serviceB라는 두 개의 서비스로 구성됩니다.
- 두 서비스 모두 apps.local이라는 네임스페이스에 등록됩니다.
- ServiceA HTTP/2, 포트 serviceB 80 이상과 통신합니다.
- serviceB의 버전 2를 이미 배포했고 apps.local 네임스페이스에 serviceBv2 이름으로 등록했습니다.

다음과 같은 요구 사항이 있습니다.

- 트래픽의 75% 를 serviceA ~로 보내고 25% 를 serviceB 첫 번째로 보내려고 serviceBv2 합니다. 로 25% 만 전송하면 트래픽의 100% 를 전송하기 전에 버그가 없는지 확인할 수 serviceA 있습니다. serviceBv2

- 트래픽이 신뢰할 수 있는 것으로 입증되면 해당 트래픽의 100%가 serviceBv2로 이동하도록 트래픽 가중치를 쉽게 조정할 수 있기를 원합니다. 모든 트래픽이 serviceBv2로 전송되면 serviceB를 중단하려고 할 수 있습니다.
- 이전 요구 사항을 충족하기 위해 실제 서비스에 대한 기존 애플리케이션 코드 또는 서비스 검색 등록을 변경할 필요가 없도록 하고 싶습니다.

요구 사항을 충족하기 위해 가상 서비스, 가상 노드, 가상 라우터 및 루트가 포함된 App Mesh 서비스 메시지를 생성하기로 결정했습니다. 메시지를 구현한 후 서비스가 Envoy 프록시를 사용하도록 서비스를 업데이트합니다. 업데이트된 서비스는 서로 직접 통신하지 않고 Envoy 프록시를 통해 서로 통신합니다.

사전 조건

- App Mesh 개념에 대한 기존의 이해. 자세한 내용은 [AWS App Mesh란?](#) 단원을 참조하십시오.
- Amazon ECSs 개념에 대한 기존 이해. 자세한 내용은 [Amazon ECS Elastic 컨테이너 서비스 개발자 안내서의 Amazon이란 무엇입니까?](#) 를 참조하십시오.
- App Mesh는 DNS AWS Cloud Map, 또는 둘 다에 등록된 리눅스 서비스를 지원합니다. 이 시작 안내서를 사용하려면 기존 서비스 3개를 등록해 두는 것이 좋습니다. 이 주제의 절차에서는 기존 서비스의 이름이 serviceA, serviceB 및 serviceBv2이고 apps.local이라는 네임스페이스를 통해 모든 서비스를 검색할 수 있다고 가정합니다.

서비스가 존재하지 않더라도 서비스 메시지 및 해당 리소스를 생성할 수 있지만 실제 서비스를 배포할 때까지 메시지를 사용할 수 없습니다. ECS Amazon에서의 서비스 검색에 대한 자세한 내용은 [서비스 검색을 참조하십시오](#). ECS 서비스 검색을 사용하여 Amazon 서비스를 생성하려면 [자습서: 서비스 검색을 사용하여 서비스 생성을 참조하십시오](#). 서비스를 아직 실행하지 않은 경우 [서비스 검색을 사용하여 Amazon ECS 서비스를 생성할 수 있습니다](#).

1단계: 메시지 및 가상 서비스 생성

서비스 메시지는 내부에 있는 서비스 간의 네트워크 트래픽에 대한 논리적 경계입니다. 자세한 내용은 [서비스 메시지](#) 단원을 참조하십시오. 가상 서비스는 실제 서비스를 추상화한 것입니다. 자세한 내용은 [가상 서비스](#) 단원을 참조하십시오.

다음의 리소스를 생성합니다.

- 시나리오의 모든 서비스가 apps.local 네임스페이스에 등록되기 때문에 apps이라는 메시지를 생성합니다.

- 가상 서비스는 해당 이름으로 검색할 수 있는 서비스를 나타내며 다른 이름을 참조하도록 코드를 변경하고 싶지 않기 때문에 `serviceb.apps.local`이라는 이름의 가상 서비스를 생성합니다. `servicea.apps.local`이라는 이름의 가상 서비스는 이후 단계에서 추가됩니다.

또는 AWS CLI 버전 1.18.116 이상 AWS Management Console 또는 2.0.38 이상을 사용하여 다음 단계를 완료할 수 있습니다. 를 AWS CLI 사용하는 경우 `aws --version` 명령을 사용하여 설치된 버전을 확인하십시오. AWS CLI 버전 1.18.116 이상 또는 2.0.38 이상이 설치되어 있지 않으면 [AWS CLI를 설치하거나 업데이트](#)해야 합니다. 사용할 도구의 탭을 선택합니다.

AWS Management Console

- 시작할 때 App Mesh 콘솔 최초 실행 마법사를 엽니다 <https://console.aws.amazon.com/appmesh/>.
- Mesh name(메시 이름)에 **apps**를 입력합니다.
- Virtual service name(가상 서비스 이름)에 **serviceb.apps.local**를 입력합니다.
- 계속하려면 다음을 선택합니다.

AWS CLI

- [create-mesh](#) 명령을 사용하여 메시를 생성합니다.

```
aws appmesh create-mesh --mesh-name apps
```

- [create-virtual-service](#) 명령을 사용하여 가상 서비스를 생성합니다.

```
aws appmesh create-virtual-service --mesh-name apps --virtual-service-name serviceb.apps.local --spec {}
```

2단계: 가상 노드 생성

가상 노드는 실제 서비스에 대한 논리적 포인터 역할을 합니다. 자세한 내용은 [가상 노드](#) 단원을 참조하십시오.

가상 노드 중 하나가 `serviceB`라는 실제 서비스를 나타내므로 `serviceB`이라는 가상 노드를 생성합니다. 가상 노드가 나타내는 실제 서비스는 호스트 이름 `serviceb.apps.local`을 사용하여 DNS를 통해 검색할 수 있습니다. 또는 를 사용하여 실제 서비스를 찾아볼 수도 있습니다. AWS Cloud Map가상 노드는 포트 80에서 HTTP /2 프로토콜을 사용하여 트래픽을 수신합니다. 상태 확인과 마찬가지로

다른 프로토콜도 지원됩니다. 이후 단계에서 `serviceA` 및 `serviceBv2`에 대한 가상 노드를 생성합니다.

AWS Management Console

1. Virtual node name(가상 노드 이름)에 **serviceB**를 입력합니다.
2. 서비스 검색 방법에서 DNS 호스트 이름을 DNS선택하고 **serviceb.apps.local** 입력합니다.
3. 리스너 구성에서 프로토콜로 `http2`를 선택하고 포트로 **80**을 입력합니다.
4. 계속하려면 다음을 선택합니다.

AWS CLI

1. 다음 콘텐츠를 가진 `create-virtual-node-serviceb.json`이라는 파일을 생성합니다:

```
{
  "meshName": "apps",
  "spec": {
    "listeners": [
      {
        "portMapping": {
          "port": 80,
          "protocol": "http2"
        }
      }
    ],
    "serviceDiscovery": {
      "dns": {
        "hostname": "serviceB.apps.local"
      }
    }
  },
  "virtualNodeName": "serviceB"
}
```

2. JSON파일을 입력으로 사용하여 [create-virtual-node](#) 명령을 사용하여 가상 노드를 생성합니다.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-serviceb.json
```

3단계: 가상 라우터 생성 및 라우팅

가상 라우터는 메시 내에 있는 하나 이상의 가상 서비스에 대한 트래픽을 라우팅합니다. 자세한 내용은 [가상 라우터](#) 및 [경로](#) 단원을 참조하세요.

다음의 리소스를 생성합니다.

- `serviceB`라는 이름의 가상 라우터. `serviceB.apps.local`이라는 가상 서비스는 다른 서비스와의 아웃바운드 통신을 시작하지 않기 때문입니다. 이전에 만든 가상 서비스는 실제 `serviceb.apps.local` 서비스를 추상화한 것입니다. 가상 서비스는 가상 라우터로 트래픽을 보냅니다. 가상 라우터는 포트 80에서 HTTP /2 프로토콜을 사용하여 트래픽을 수신합니다. 다른 프로토콜도 지원됩니다.
- `serviceB`라는 이름의 라우팅. 트래픽을 100% `serviceB` 가상 노드로 라우팅합니다. `serviceBv2` 가상 노드를 추가한 후 이후 단계에서 가중치가 변경됩니다. 이 안내서에서는 다루지 않지만 해당 라우팅에 대한 필터 조건을 추가하고 재시도 정책을 추가하여 통신 문제가 발생할 경우 Envoy 프록시가 가상 노드에 트래픽을 여러 번 보내도록 할 수 있습니다.

AWS Management Console

1. Virtual router name(가상 라우터 이름)에 **serviceB**를 입력합니다.
2. 리스너 구성에서 프로토콜로 `http2`를 선택하고 포트로 **80**을 지정합니다.
3. Route Name(라우팅 이름)에 **serviceB**를 입력합니다.
4. 루트 유형에 대해 `http2`를 선택합니다.
5. 대상 구성의 가상 노드 이름에 대해 `serviceB`를 선택하고 가중치로 **100**을 입력합니다.
6. 일치 구성에서 방법을 선택합니다.
7. 계속하려면 다음을 선택합니다.

AWS CLI

1. 가상 라우터를 생성합니다.
 - a. 다음 콘텐츠를 가진 `create-virtual-router.json`이라는 파일을 생성합니다:

```
{
  "meshName": "apps",
  "spec": {
    "listeners": [
```

```

        {
            "portMapping": {
                "port": 80,
                "protocol": "http2"
            }
        }
    ],
    "virtualRouterName": "serviceB"
}

```

- b. JSON파일을 입력으로 사용하여 [create-virtual-router](#) 명령을 사용하여 가상 라우터를 생성합니다.

```
aws appmesh create-virtual-router --cli-input-json file://create-virtual-router.json
```

2. 라우팅을 생성합니다.

- a. 다음 콘텐츠를 가진 `create-route.json`이라는 파일을 생성합니다:

```

{
  "meshName" : "apps",
  "routeName" : "serviceB",
  "spec" : {
    "httpRoute" : {
      "action" : {
        "weightedTargets" : [
          {
            "virtualNode" : "serviceB",
            "weight" : 100
          }
        ]
      },
      "match" : {
        "prefix" : "/"
      }
    }
  },
  "virtualRouterName" : "serviceB"
}

```

- b. JSON파일을 입력으로 사용하여 [create-route](#) 명령으로 경로를 생성합니다.

```
aws appmesh create-route --cli-input-json file://create-route.json
```

4단계: 검토 및 생성

이전 지침에 대한 설정을 검토합니다.

AWS Management Console

섹션에서 변경해야 할 경우 편집을 선택합니다. 설정에 만족하면 메시 생성을 선택합니다.

상태 화면에는 생성된 모든 메시 리소스가 표시됩니다. 메시 보기를 선택하여 콘솔에서 생성된 리소스를 볼 수 있습니다.

AWS CLI

[describe-mesh](#) 명령을 사용하여 생성한 메시의 설정을 검토합니다.

```
aws appmesh describe-mesh --mesh-name apps
```

명령으로 만든 가상 서비스의 설정을 검토하십시오. [describe-virtual-service](#)

```
aws appmesh describe-virtual-service --mesh-name apps --virtual-service-name serviceb.apps.local
```

[describe-virtual-node](#) 명령으로 만든 가상 노드의 설정을 검토하십시오.

```
aws appmesh describe-virtual-node --mesh-name apps --virtual-node-name serviceB
```

[describe-virtual-router](#) 명령으로 만든 가상 라우터의 설정을 검토하십시오.

```
aws appmesh describe-virtual-router --mesh-name apps --virtual-router-name serviceB
```

[describe-route](#) 명령으로 생성한 라우팅의 설정을 검토합니다.

```
aws appmesh describe-route --mesh-name apps \  
--virtual-router-name serviceB --route-name serviceB
```

5단계: 추가 리소스 생성

시나리오를 완료하려면 다음을 수행해야 합니다.

- `serviceBv2`라는 가상 노드 하나와 `serviceA`라는 가상 노드 하나를 생성합니다. 두 가상 노드 모두 HTTP /2 포트 80을 통한 요청을 수신합니다. `serviceA` 가상 노드의 경우 `serviceb.apps.local`의 백엔드를 구성합니다. `serviceA` 가상 노드의 모든 아웃바운드 트래픽은 이름이 `serviceb.apps.local`인 가상 서비스로 전송됩니다. 이 안내서에서는 다루지 않지만 가상 노드에 대한 액세스 로그를 쓸 파일 경로를 지정할 수도 있습니다.
- `servicea.apps.local`이라는 가상 서비스를 추가로 생성하면 모든 트래픽이 `serviceA` 가상 노드로 직접 전송됩니다.
- 트래픽의 75%를 `serviceB` 가상 노드로 보내고 트래픽의 25%를 `serviceBv2` 가상 노드로 보내도록 이전 단계에서 생성한 `serviceB` 라우팅을 업데이트합니다. 시간이 지남에 따라 `serviceBv2`가 트래픽의 100%를 수신할 때까지 가중치를 계속 수정할 수 있습니다. 모든 트래픽이 `serviceBv2`로 전송되면 `serviceB` 가상 노드와 실제 서비스를 종료하고 중단할 수 있습니다. `serviceb.apps.local` 가상 및 실제 서비스 이름이 변경되지 않으므로 가중치를 변경할 때 코드를 수정할 필요가 없습니다. `serviceb.apps.local` 가상 서비스는 트래픽을 가상 라우터로 전송하여 트래픽을 가상 노드로 라우팅한다는 점을 기억하십시오. 가상 노드의 서비스 검색 이름은 언제든지 변경할 수 있습니다.

AWS Management Console

1. 왼쪽 탐색 창에서 Meshes(메시)를 선택합니다.
2. 이전 단계에서 생성한 apps 메시를 선택합니다.
3. 왼쪽 탐색 창에서 Virtual node(가상 노드)를 선택합니다.
4. Create virtual node(가상 노드 생성)를 선택합니다.
5. 가상 노드 이름에는 을 입력하고 **serviceBv2**, 서비스 검색 방법에는 을 DNS, DNS호스트 이름에는 를 입력합니다. **servicebv2.apps.local**
6. 리스너 구성에서 프로토콜로 http2를 선택하고 포트로 **80**을 입력합니다.
7. Create virtual node(가상 노드 생성)를 선택합니다.
8. 가상 노드 생성을 다시 선택합니다. 가상 노드 이름에 **serviceA**를 입력합니다. 서비스 검색 방법에서는 을 선택하고 DNSDNS호스트 이름에는 를 입력합니다. **servicea.apps.local**
9. 새 백엔드에서 가상 서비스 이름 입력에 **serviceb.apps.local**을 입력합니다.
10. 리스너 구성에서 프로토콜로 http2를 선택하고 포트에 **80**을 입력한 다음, 가상 노드 생성을 선택합니다.

11. 왼쪽 탐색 창에서 Virtual routers(가상 라우터)를 선택한 다음 목록에서 serviceB 가상 라우터를 선택합니다.
12. 루트에서 이전 단계에서 생성한 ServiceB라는 루트를 선택하고 편집을 선택합니다.
13. 대상, 가상 노드 이름에서 serviceB의 가중치 값을 **75**로 변경합니다.
14. 대상 추가를 선택하고 드롭다운 목록에서 serviceBv2를 선택하고 가중치의 값을 **25**로 설정합니다.
15. 저장(Save)을 선택합니다.
16. 왼쪽 탐색 창에서 가상 서비스를 선택한 다음 가상 서비스 생성을 선택합니다.
17. 가상 서비스 이름으로 **servicea.apps.local**을 입력하고 공급자로 가상 노드를 선택하고 가상 노드로 serviceA를 선택한 다음, 가상 서비스 생성을 선택합니다.

AWS CLI

1. serviceBv2 가상 노드를 생성합니다.
 - a. 다음 콘텐츠를 가진 create-virtual-node-servicebv2.json이라는 파일을 생성합니다:

```
{
  "meshName": "apps",
  "spec": {
    "listeners": [
      {
        "portMapping": {
          "port": 80,
          "protocol": "http2"
        }
      }
    ],
    "serviceDiscovery": {
      "dns": {
        "hostname": "serviceBv2.apps.local"
      }
    }
  },
  "virtualNodeName": "serviceBv2"
}
```

- b. 가상 노드를 생성합니다.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-servicebv2.json
```

2. serviceA 가상 노드를 생성합니다.

- a. 다음 콘텐츠를 가진 create-virtual-node-servicea.json이라는 파일을 생성합니다:

```
{
  "meshName" : "apps",
  "spec" : {
    "backends" : [
      {
        "virtualService" : {
          "virtualServiceName" : "serviceb.apps.local"
        }
      }
    ],
    "listeners" : [
      {
        "portMapping" : {
          "port" : 80,
          "protocol" : "http2"
        }
      }
    ],
    "serviceDiscovery" : {
      "dns" : {
        "hostname" : "servicea.apps.local"
      }
    }
  },
  "virtualNodeName" : "serviceA"
}
```

- b. 가상 노드를 생성합니다.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-servicea.json
```

3. 이전 단계에서 생성한 `serviceb.apps.local` 가상 서비스를 업데이트하여 트래픽을 `serviceB` 가상 라우터로 보냅니다. 가상 서비스가 원래 생성되었을 때는 `serviceB` 가상 라우터가 아직 생성되지 않았기 때문에 트래픽을 어디에도 보내지 않았습니다.
 - a. 다음 콘텐츠를 가진 `update-virtual-service.json`이라는 파일을 생성합니다:

```
{
  "meshName" : "apps",
  "spec" : {
    "provider" : {
      "virtualRouter" : {
        "virtualRouterName" : "serviceB"
      }
    }
  },
  "virtualServiceName" : "serviceb.apps.local"
}
```

- b. [update-virtual-service](#) 명령을 사용하여 가상 서비스를 업데이트합니다.

```
aws appmesh update-virtual-service --cli-input-json file://update-virtual-service.json
```

4. 이전 단계에서 생성한 `serviceB` 라우팅을 업데이트합니다.
 - a. 다음 콘텐츠를 가진 `update-route.json`이라는 파일을 생성합니다:

```
{
  "meshName" : "apps",
  "routeName" : "serviceB",
  "spec" : {
    "http2Route" : {
      "action" : {
        "weightedTargets" : [
          {
            "virtualNode" : "serviceB",
            "weight" : 75
          },
          {
            "virtualNode" : "serviceBv2",
            "weight" : 25
          }
        ]
      }
    }
  }
}
```

```

    },
    "match" : {
      "prefix" : "/"
    }
  }
},
"virtualRouterName" : "serviceB"
}

```

- b. [update-route](#) 명령을 사용하여 라우팅을 업데이트합니다.

```
aws appmesh update-route --cli-input-json file://update-route.json
```

5. serviceA 가상 서비스를 생성합니다.

- a. 다음 콘텐츠를 가진 create-virtual-servicea.json이라는 파일을 생성합니다:

```

{
  "meshName" : "apps",
  "spec" : {
    "provider" : {
      "virtualNode" : {
        "virtualNodeName" : "serviceA"
      }
    }
  },
  "virtualServiceName" : "servicea.apps.local"
}

```

- b. 가상 서비스를 생성합니다.

```
aws appmesh create-virtual-service --cli-input-json file://create-virtual-servicea.json
```

메시 요약

서비스 메시지를 생성하기 전에는 servicea.apps.local, serviceb.apps.local 및 servicebv2.apps.local이라는 세 가지 실제 서비스가 있었습니다. 실제 서비스 외에도 이제 실제 서비스를 나타내는 다음 리소스가 포함된 서비스 메시가 있습니다.

- 두 개의 가상 서비스. 프록시는 가상 라우터를 통해 `servicea.apps.local` 가상 서비스의 모든 트래픽을 `serviceb.apps.local` 가상 서비스로 보냅니다.
- `serviceA`, `serviceB` 및 `serviceBv2`라는 세 개의 가상 노드. Envoy 프록시는 가상 노드에 대해 구성된 서비스 검색 정보를 사용하여 실제 서비스의 IP 주소를 조회합니다.
- Envoy 프록시가 인바운드 트래픽의 75%를 `serviceB` 가상 노드로 라우팅하고 트래픽의 25%를 `serviceBv2` 가상 노드로 라우팅하도록 지시하는 하나의 라우팅이 있는 가상 라우터 하나.

6단계: 서비스 업데이트

메시를 생성한 후에는 다음 작업을 완료해야 합니다.

- 각 Amazon ECS 작업과 함께 배포하는 Envoy 프록시가 하나 이상의 가상 노드 구성을 읽을 수 있도록 승인합니다. 프록시에 권한을 부여하는 방법에 대한 자세한 내용은 [프록시 권한 부여](#)를 참조하십시오.
- Envoy 프록시를 사용하도록 기존 Amazon ECS 작업 정의를 각각 업데이트하십시오.

보안 인증 정보

Envoy 컨테이너에는 App Mesh 서비스로 전송되는 서명 요청에 대한 AWS Identity and Access Management 자격 증명이 필요합니다. Amazon EC2 시작 유형으로 배포된 Amazon ECS 작업의 경우 자격 증명은 [인스턴스 역할 또는 작업 IAM 역할에서](#) 가져올 수 있습니다. Linux 컨테이너에서 Fargate를 사용하여 배포한 Amazon ECS 작업은 인스턴스 IAM 프로필 자격 증명을 제공하는 Amazon EC2 메타데이터 서버에 액세스할 수 없습니다. 자격 증명을 제공하려면 Fargate on Linux 컨테이너 유형으로 배포된 모든 작업에 작업 역할을 연결해야 합니다. IAM

작업을 Amazon EC2 Launch 유형으로 배포하고 Amazon EC2 메타데이터 서버에 대한 액세스가 차단된 경우 ([작업을 위한 IAM 역할의](#) 중요 주석에 설명되어 있음) 작업 IAM 역할도 작업에 연결해야 합니다. 인스턴스 또는 작업에 할당하는 역할에는 [프록시 권한 부여에](#) 설명된 대로 IAM 정책이 연결되어 있어야 합니다.

를 사용하여 작업 정의를 업데이트하려면 AWS CLI

Amazon ECS AWS CLI 명령을 사용합니다 [register-task-definition](#). 아래 예제 작업 정의는 서비스에 맞게 App Mesh를 구성하는 방법을 보여줍니다.

Note

콘솔을 ECS 통해 Amazon용 App Mesh를 구성하는 것은 불가능합니다.

태스크 정의 ARN**프록시 구성**

App Mesh를 사용하도록 Amazon ECS 서비스를 구성하려면 서비스의 작업 정의에 다음과 같은 프록시 구성 섹션이 있어야 합니다. 프록시 구성 type을 APPMESH로 설정하고 containerName을 envoy로 설정합니다. 다음 속성 값을 적절히 설정합니다.

IgnoredUID

Envoy 프록시는 이 사용자 ID를 사용하는 프로세스의 트래픽을 라우팅하지 않습니다. 이 속성 값에 대해 원하는 사용자 ID를 선택할 수 있지만, 이 ID는 작업 정의의 Envoy 컨테이너에 대한 user ID와 동일해야 합니다. 이렇게 일치할 경우 Envoy는 프록시를 사용하지 않고 자체 트래픽을 무시할 수 있습니다. 예제에서는 기록을 위해 **1337**을 사용합니다.

ProxyIngressPort

Envoy 프록시 컨테이너의 인바운드 포트입니다. 이 값을 15000로 설정합니다.

ProxyEgressPort

Envoy 프록시 컨테이너의 아웃바운드 포트입니다. 이 값을 15001로 설정합니다.

AppPorts

애플리케이션 컨테이너가 수신 대기하는 모든 인바운드 포트를 지정합니다. 이 예제에서 애플리케이션 컨테이너는 **9080** 포트를 통해 수신합니다. 지정하는 포트는 가상 노드 리스너에 구성된 포트와 일치해야 합니다.

EgressIgnoredIPs

Envoy는 이러한 IP 주소에 대한 트래픽을 프록시 처리하지 않습니다. 이 값을 로 169.254.170.2, 169.254.169.254 설정하면 Amazon EC2 메타데이터 서버와 Amazon ECS 작업 메타데이터 엔드포인트는 무시됩니다. 메타데이터 엔드포인트는 작업 자격 증명에 대한 IAM 역할을 제공합니다. 주소를 추가할 수 있습니다.

EgressIgnoredPorts

범포로 구분된 포트 목록을 추가할 수 있습니다. Envoy는 이러한 포트에 대한 트래픽을 프록시 처리하지 않습니다. 포트를 나열하지 않은 경우에도 포트 22는 무시됩니다.

Note

무시할 수 있는 최대 아웃바운드 포트 수는 15개입니다.

```
"proxyConfiguration": {
  "type": "APPMESH",
  "containerName": "envoy",
  "properties": [{
    "name": "IgnoredUID",
    "value": "1337"
  },
  {
    "name": "ProxyIngressPort",
    "value": "15000"
  },
  {
    "name": "ProxyEgressPort",
    "value": "15001"
  },
  {
    "name": "AppPorts",
    "value": "9080"
  },
  {
    "name": "EgressIgnoredIPs",
    "value": "169.254.170.2,169.254.169.254"
  },
  {
    "name": "EgressIgnoredPorts",
    "value": "22"
  }
  ]
}
```

애플리케이션 컨테이너 Envoy 종속성

작업 정의의 애플리케이션 컨테이너를 시작하려면 Envoy 프록시가 부트스트랩되고 시작될 때까지 기다려야 합니다. 이를 위해서는 각 애플리케이션 컨테이너 정의에 `dependsOn` 섹션을 설정하여 Envoy 컨테이너가 HEALTHY로 보고될 때까지 기다립니다. 다음 코드는 이 종속성이 있는 애플리케이션 컨테

이너 정의 예제를 보여줍니다. 다음 예제의 모든 속성이 필요합니다. 일부 속성 값도 필요하지만 일부는 필수입니다. *replaceable*.

```
{
  "name": "appName",
  "image": "appImage",
  "portMappings": [{
    "containerPort": 9080,
    "hostPort": 9080,
    "protocol": "tcp"
  }],
  "essential": true,
  "dependsOn": [{
    "containerName": "envoy",
    "condition": "HEALTHY"
  }]
}
```

Envoy 컨테이너 정의

Amazon ECS 작업 정의에는 App Mesh Envoy 컨테이너 이미지가 포함되어야 합니다.

me-south-1, ap-east-1, ap-southeast-3, eu-south-1, il-central-1 및 af-south-1을 제외한 모든 [지원](#) 리전 다음을 대체할 수 있습니다. *Region-code* me-south-1,, ap-east-1 ap-southeast-3 eu-south-1 il-central-1, 및 af-south-1 를 제외한 모든 지역으로

표준

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

FIPS-규정 준수

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

me-south-1

표준

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

FIPS-준수

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

ap-east-1

표준

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

FIPS-준수

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

ap-southeast-3

표준

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

FIPS-준수

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

eu-south-1

표준

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

FIPS-준수

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

il-central-1

표준

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

FIPS-준수

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

af-south-1

표준

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

FIPS-준수

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

Public repository

표준

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.29.6.1-prod
```

FIPS-준수

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

 Important

v1.9.0.0-prod 이상 버전만 App Mesh에서 사용할 수 있습니다.

Envoy 프로젝트 팀이 App Mesh를 지원하는 변경 사항을 병합할 때까지 App Mesh Envoy 컨테이너 이미지를 사용해야 합니다. 자세한 내용은 [GitHub 로드맵](#) 문제를 참조하십시오.

다음 예제의 모든 속성이 필요합니다. 일부 속성 값도 필요하지만 일부는 필수입니다. *replaceable*.

 Note

- Envoy 컨테이너 정의는 `essential`로 표시되어야 합니다.

- Envoy 컨테이너에 512 CPU 유닛과 최소 64 MiB 메모리를 할당하는 것이 좋습니다. Fargate에서 설정할 수 있는 최소 용량은 1024MiB 메모리입니다.
- Amazon ECS 서비스의 가상 노드 이름은 APPMESH_RESOURCE_ARN 속성 값으로 설정되어야 합니다. 이 속성에는 1.15.0 이상의 Envoy 이미지 버전이 필요합니다. 자세한 내용은 [Envoy](#) 단원을 참조하십시오.
- user 설정 값은 작업 정의 프록시 구성의 IgnoredUID 값과 일치해야 합니다. 이 예제에서는 **1337**을 사용합니다.
- 여기에 표시된 상태 점검은 Envoy 컨테이너가 정상 상태이고 애플리케이션 컨테이너를 시작할 준비가 되었다고 ECS Amazon에 보고하기 전에 Envoy 컨테이너가 제대로 부트스트랩 될 때까지 기다립니다.
- 기본적으로 App Mesh는 Envoy가 지표 및 트레이스에서 자신을 참조할 때 APPMESH_RESOURCE_ARN에서 지정한 리소스의 이름을 사용합니다. APPMESH_RESOURCE_CLUSTER 환경 변수를 사용자 고유의 이름으로 설정하여 이 동작을 재정의할 수 있습니다. 이 속성에는 1.15.0 이상의 Envoy 이미지 버전이 필요합니다. 자세한 내용은 [Envoy](#) 단원을 참조하십시오.

다음 코드는 Envoy 컨테이너 정의 예제를 보여줍니다.

```
{
  "name": "envoy",
  "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod",
  "essential": true,
  "environment": [{
    "name": "APPMESH_RESOURCE_ARN",
    "value": "arn:aws:appmesh:us-west-2:111122223333:mesh/apps/virtualNode/serviceB"
  }],
  "healthCheck": {
    "command": [
      "CMD-SHELL",
      "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
    ],
    "startPeriod": 10,
    "interval": 5,
    "timeout": 2,
    "retries": 3
  },
  "user": "1337"
```

```
}

```

태스크 정의 예제

다음 예제 Amazon ECS 작업 정의는 위의 예제를 작업 정의에 병합하는 방법을 보여줍니다 taskB. 사용 여부에 관계없이 Amazon ECS 시작 유형 모두에 대한 작업을 생성하는 예제가 제공됩니다 AWS X-Ray. 변경: *replaceable* 값을 적절히 지정하여 taskA 시나리오에서 이름이 taskBv2 지정되고 생성된 작업에 대한 작업 정의를 생성합니다. APPMESH_RESOURCE_ARN 값에 대한 메시 이름 및 가상 노드 이름과, 애플리케이션이 프록시 구성 AppPorts 값을 수신하는 포트 목록을 대체합니다. 기본적으로 App Mesh는 Envoy가 지표 및 트레이스에서 자신을 참조할 때 APPMESH_RESOURCE_ARN에서 지정한 리소스의 이름을 사용합니다. APPMESH_RESOURCE_CLUSTER 환경 변수를 사용자 고유의 이름으로 설정하여 이 동작을 재정의할 수 있습니다. 다음 예제의 모든 속성이 필요합니다. 일부 속성 값도 필요하지만 일부는 필수입니다. *replaceable*.

자격 증명 섹션에 설명된 대로 Amazon ECS 작업을 실행하는 경우 [예제에 기존 작업 IAM 역할을](#) 추가해야 합니다.

Important

Fargate는 1024보다 큰 포트 값을 사용해야 합니다.

Example JSON아마존 ECS 작업 정의용 - 리눅스 컨테이너의 Fargate

```
{
  "family" : "taskB",
  "memory" : "1024",
  "cpu" : "0.5 vCPU",
  "proxyConfiguration" : {
    "containerName" : "envoy",
    "properties" : [
      {
        "name" : "ProxyIngressPort",
        "value" : "15000"
      },
      {
        "name" : "AppPorts",
        "value" : "9080"
      },
      {

```

```

        "name" : "EgressIgnoredIPs",
        "value" : "169.254.170.2,169.254.169.254"
    },
    {
        "name": "EgressIgnoredPorts",
        "value": "22"
    },
    {
        "name" : "IgnoredUID",
        "value" : "1337"
    },
    {
        "name" : "ProxyEgressPort",
        "value" : "15001"
    }
],
"type" : "APPMESH"
},
"containerDefinitions" : [
    {
        "name" : "appName",
        "image" : "appImage",
        "portMappings" : [
            {
                "containerPort" : 9080,
                "protocol" : "tcp"
            }
        ],
        "essential" : true,
        "dependsOn" : [
            {
                "containerName" : "envoy",
                "condition" : "HEALTHY"
            }
        ]
    },
    {
        "name" : "envoy",
        "image" : "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.29.6.1-prod",
        "essential" : true,
        "environment" : [
            {
                "name" : "APPMESH_VIRTUAL_NODE_NAME",

```

```

        "value" : "mesh/apps/virtualNode/serviceB"
      }
    ],
    "healthCheck" : {
      "command" : [
        "CMD-SHELL",
        "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
      ],
      "interval" : 5,
      "retries" : 3,
      "startPeriod" : 10,
      "timeout" : 2
    },
    "memory" : 500,
    "user" : "1337"
  }
],
"requiresCompatibilities" : [ "FARGATE" ],
"taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
"executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
"networkMode" : "awsipc"
}

```

Example JSON AWS X-Ray - 리눅스 컨테이너의 Fargate를 사용하는 Amazon ECS 작업 정의의 경우

X-Ray에서는 사용자가 트래픽 흐름을 시각화하는 데 사용할 수 있는 도구를 애플리케이션이 다루고 제공해야 한다는 요청에 대한 데이터를 수집할 수 있습니다. Envoy에 대한 X-Ray 드라이버를 사용하면 Envoy가 추적 정보를 X-Ray에 보고할 수 있습니다. [Envoy 구성](#)을 사용하여 X-Ray 추적 기능을 활성화할 수 있습니다. 구성에 따라 Envoy는 [사이드카](#) 컨테이너로 실행되는 X-Ray 대몬(daemon)에 추적 데이터를 보내고 대몬(daemon)은 추적을 X-Ray 서비스에 전달합니다. 추적이 X-Ray에 게시되면 X-Ray 콘솔을 사용하여 서비스 호출 그래프를 시각화하고 추적 세부 정보를 요청할 수 있습니다. 다음은 X-Ray 통합을 가능하게 하는 작업 정의를 JSON 나타냅니다.

```

{

  "family" : "taskB",
  "memory" : "1024",
  "cpu" : "512",
  "proxyConfiguration" : {
    "containerName" : "envoy",
    "properties" : [

```

```
{
  "name" : "ProxyIngressPort",
  "value" : "15000"
},
{
  "name" : "AppPorts",
  "value" : "9080"
},
{
  "name" : "EgressIgnoredIPs",
  "value" : "169.254.170.2,169.254.169.254"
},
{
  "name": "EgressIgnoredPorts",
  "value": "22"
},
{
  "name" : "IgnoredUID",
  "value" : "1337"
},
{
  "name" : "ProxyEgressPort",
  "value" : "15001"
}
],
"type" : "APPMESH"
},
"containerDefinitions" : [
  {
    "name" : "appName",
    "image" : "appImage",
    "portMappings" : [
      {
        "containerPort" : 9080,
        "protocol" : "tcp"
      }
    ],
    "essential" : true,
    "dependsOn" : [
      {
        "containerName" : "envoy",
        "condition" : "HEALTHY"
      }
    ]
  }
]
```

```
  },
  {
    "name" : "envoy",
    "image" : "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.29.6.1-prod",
    "essential" : true,
    "environment" : [
      {
        "name" : "APPMESH_VIRTUAL_NODE_NAME",
        "value" : "mesh/apps/virtualNode/serviceB"
      },
      {
        "name": "ENABLE_ENVOY_XRAY_TRACING",
        "value": "1"
      }
    ],
    "healthCheck" : {
      "command" : [
        "CMD-SHELL",
        "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
      ],
      "interval" : 5,
      "retries" : 3,
      "startPeriod" : 10,
      "timeout" : 2
    },
    "memory" : 500,
    "user" : "1337"
  },
  {
    "name" : "xray-daemon",
    "image" : "amazon/aws-xray-daemon",
    "user" : "1337",
    "essential" : true,
    "cpu" : "32",
    "memoryReservation" : "256",
    "portMappings" : [
      {
        "containerPort" : 2000,
        "protocol" : "udp"
      }
    ]
  }
]
```

```

],
"requiresCompatibilities" : [ "FARGATE" ],
"taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
"executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
"networkMode" : "awsvpc"
}

```

Example JSON Amazon ECS 작업 정의용 - EC2 시작 유형

```

{
  "family": "taskB",
  "memory": "256",
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "envoy",
    "properties": [
      {
        "name": "IgnoredUID",
        "value": "1337"
      },
      {
        "name": "ProxyIngressPort",
        "value": "15000"
      },
      {
        "name": "ProxyEgressPort",
        "value": "15001"
      },
      {
        "name": "AppPorts",
        "value": "9080"
      },
      {
        "name": "EgressIgnoredIPs",
        "value": "169.254.170.2,169.254.169.254"
      },
      {
        "name": "EgressIgnoredPorts",
        "value": "22"
      }
    ]
  },
  "containerDefinitions": [

```

```

{
  "name": "appName",
  "image": "appImage",
  "portMappings": [
    {
      "containerPort": 9080,
      "hostPort": 9080,
      "protocol": "tcp"
    }
  ],
  "essential": true,
  "dependsOn": [
    {
      "containerName": "envoy",
      "condition": "HEALTHY"
    }
  ]
},
{
  "name": "envoy",
  "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.29.6.1-prod",
  "essential": true,
  "environment": [
    {
      "name": "APPMESH_VIRTUAL_NODE_NAME",
      "value": "mesh/apps/virtualNode/serviceB"
    }
  ],
  "healthCheck": {
    "command": [
      "CMD-SHELL",
      "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
    ],
    "startPeriod": 10,
    "interval": 5,
    "timeout": 2,
    "retries": 3
  },
  "user": "1337"
}
],
"requiresCompatibilities" : [ "EC2" ],
"taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",

```

```

"executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
"networkMode": "awsvpc"
}

```

Example JSON Amazon ECS 작업 정의의 경우 AWS X-Ray - EC2 시작 유형

```

{
  "family": "taskB",
  "memory": "256",
  "cpu" : "1024",
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "envoy",
    "properties": [
      {
        "name": "IgnoredUID",
        "value": "1337"
      },
      {
        "name": "ProxyIngressPort",
        "value": "15000"
      },
      {
        "name": "ProxyEgressPort",
        "value": "15001"
      },
      {
        "name": "AppPorts",
        "value": "9080"
      },
      {
        "name": "EgressIgnoredIPs",
        "value": "169.254.170.2,169.254.169.254"
      },
      {
        "name": "EgressIgnoredPorts",
        "value": "22"
      }
    ]
  },
  "containerDefinitions": [
    {
      "name": "appName",

```

```
"image": "appImage",
"portMappings": [
  {
    "containerPort": 9080,
    "hostPort": 9080,
    "protocol": "tcp"
  }
],
"essential": true,
"dependsOn": [
  {
    "containerName": "envoy",
    "condition": "HEALTHY"
  }
]
},
{
  "name": "envoy",
  "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.29.6.1-prod",
  "essential": true,
  "environment": [
    {
      "name": "APPMESH_VIRTUAL_NODE_NAME",
      "value": "mesh/apps/virtualNode/serviceB"
    },
    {
      "name": "ENABLE_ENVOY_XRAY_TRACING",
      "value": "1"
    }
  ],
  "healthCheck": {
    "command": [
      "CMD-SHELL",
      "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
    ],
    "startPeriod": 10,
    "interval": 5,
    "timeout": 2,
    "retries": 3
  },
  "user": "1337"
},
{
```

```

    "name": "xray-daemon",
    "image": "amazon/aws-xray-daemon",
    "user": "1337",
    "essential": true,
    "cpu": 32,
    "memoryReservation": 256,
    "portMappings": [
      {
        "containerPort": 2000,
        "protocol": "udp"
      }
    ]
  },
  "requiresCompatibilities" : [ "EC2" ],
  "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
  "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}

```

고급 주제

App Mesh를 사용한 Canary 배포

Canary 배포 및 릴리스를 통해 애플리케이션의 이전 버전과 새로 배포된 버전 간에 트래픽을 전환할 수 있습니다. 또한 새로 배포된 버전의 상태도 모니터링됩니다. 새 버전에 문제가 있는 경우 Canary 배포는 자동으로 트래픽을 이전 버전으로 다시 전환할 수 있습니다. Canary 배포를 통해 보다 강력하게 애플리케이션 버전 간에 트래픽을 전환할 수 있습니다.

App ECS Mesh를 사용하여 Amazon에 카나리아 배포를 구현하는 방법에 대한 자세한 내용은 App Mesh를 사용하여 Amazon용 [카나리아 배포가 포함된 파이프라인 생성을 참조하십시오. ECS](#)

Note

App Mesh에 대한 더 많은 예제와 연습 내용을 보려면 [App Mesh 예제 리포지토리](#)를 참조하세요.

AWS App Mesh 쿠버네티스 시작하기

쿠버네티스용 App Mesh 컨트롤러를 사용하여 AWS App Mesh 쿠버네티스와 통합하면 메시, 가상 서비스, 가상 노드, 가상 라우터, 쿠버네티스를 통한 경로 등과 같은 App Mesh 리소스를 관리할 수 있습니다. 또한 App Mesh 사이드카 컨테이너 이미지를 Kubernetes 포드 사양에 자동으로 추가합니다. 이 자습서에서는 Kubernetes용 App Mesh 컨트롤러를 설치하여 이러한 통합을 활성화하는 방법을 안내합니다.

컨트롤러에는 Kubernetes 사용자 지정 리소스 정의(meshes, virtual services, virtual nodes 및 virtual routers)의 배포가 포함됩니다. 컨트롤러는 사용자 지정 리소스의 생성, 수정 및 삭제를 감시하고 App Mesh를 통해 해당 App Mesh [the section called “메시”](#) [the section called “가상 서비스”](#) [the section called “가상 노드”](#) [the section called “가상 게이트웨이”](#) [the section called “게이트웨이 경로”](#),,,, [the section called “가상 라우터”](#) (포함 [the section called “경로”](#)) 리소스를 변경합니다API. 컨트롤러에 대해 자세히 알아보거나 컨트롤러에 기여하려면 [GitHub프로젝트](#)를 참조하십시오.

또한 컨트롤러는 사용자가 지정한 이름으로 레이블이 지정된 Kubernetes 포드에 다음 컨테이너를 삽입하는 웹훅을 설치합니다.

- App Mesh Envoy 프록시 - Envoy는 App Mesh 컨트롤 영역에 정의된 구성을 사용하여 애플리케이션 트래픽을 전송할 위치를 결정합니다.
- App Mesh 프록시 경로 관리자 - 포드의 네트워크 네임스페이스에서 Envoy를 통한 인바운드 및 아웃바운드 트래픽을 라우팅하는 iptables 규칙을 업데이트합니다. 이 컨테이너는 포드 내부의 Kubernetes 초기화 컨테이너로 실행됩니다.

사전 조건

- App Mesh 개념에 대한 기존의 이해. 자세한 내용은 [AWS App Mesh란?](#) 단원을 참조하십시오.
- Kubernetes 개념에 대한 기존의 이해. 자세한 내용은 Kubernetes 설명서의 [Kubernetes란?](#)을 참조하세요.
- 기존 Kubernetes 클러스터. 기존 클러스터가 없는 경우 Amazon EKS사용 설명서의 [Amazon EKS 시작하기](#)를 참조하십시오. Amazon에서 자체 Kubernetes 클러스터를 실행하는 경우 Envoy 이미지가 있는 ECR Amazon EC2 리포지토리에 Docker가 인증되었는지 확인하십시오. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 [Envoy 이미지](#), [레지스트리 인증](#) 및 Kubernetes 설명서의 [프라이빗 레지스트리에서 이미지 가져오기](#)를 참조하세요.
- App Mesh는 DNS AWS Cloud Map, 또는 둘 다에 등록된 리눅스 서비스를 지원합니다. 이 시작 안내서를 사용하려면 기존 서비스 3개를 등록해 두는 것이 좋습니다DNS. 이 주제의 절차에서는 기존

서비스의 이름이 serviceA, serviceB 및 serviceBv2이고 apps.local이라는 네임스페이스를 통해 모든 서비스를 검색할 수 있다고 가정합니다.

서비스가 존재하지 않더라도 서비스 메시 및 해당 리소스를 생성할 수 있지만 실제 서비스를 배포할 때까지 메시지를 사용할 수 없습니다.

- AWS CLI 버전 1.18.116 이상 또는 2.0.38 이상이 설치되어 있습니다. [클 설치 또는 업그레이드하려면 설치를 참조하십시오. AWS CLI](#)
- Kubernetes 클러스터와 통신하도록 구성된 kubectl 클라이언트. Amazon Elastic Kubernetes Service를 사용하는 경우 [kubectl](#) 설치 및 [kubeconfig](#) 파일 구성에 대한 지침을 사용할 수 있습니다.
- Helm 버전 3.0 이상이 설치되어 있어야 합니다. Helm이 설치되어 있지 않은 경우 [Amazon 사용 EKS 설명서의 EKS Amazon에서 Helm 사용하기](#) 섹션을 참조하십시오.
- Amazon은 EKS 현재 IP 기본 IPv6_ONLY 설정만 지원합니다. IPv4_ONLY. Amazon은 EKS 현재 트래픽만 처리하거나 IPv4 트래픽만 처리할 수 있는 포드만 지원하기 때문입니다. IPv6

나머지 단계에서는 실제 서비스의 이름이 serviceA, serviceB 및 serviceBv2이고 apps.local이라는 네임스페이스를 통해 모든 서비스를 검색할 수 있다고 가정합니다.

1단계: 통합 구성 요소 설치

App Mesh에 사용할 포드를 호스팅하는 각 클러스터에 통합 구성 요소를 한 번 설치합니다.

통합 구성 요소를 설치하려면

1. 이 절차의 나머지 단계에서는 시험판 버전의 컨트롤러가 설치되지 않은 클러스터가 필요합니다. 시험판 버전을 설치했거나 보유하고 있는지 확실하지 않은 경우 클러스터에 시험판 버전이 설치되어 있는지 확인하는 스크립트를 다운로드하여 실행할 수 있습니다.

```
curl -o pre_upgrade_check.sh https://raw.githubusercontent.com/aws/eks-charts/master/stable/appmesh-controller/upgrade/pre_upgrade_check.sh
sh ./pre_upgrade_check.sh
```

스크립트가 Your cluster is ready for upgrade. Please proceed to the installation instructions를 반환하면 다음 단계로 진행할 수 있습니다. 다른 메시지가 반환되면 계속하기 전에 업그레이드 단계를 완료해야 합니다. 시험판 버전 업그레이드에 대한 자세한 내용은 [Upgrade on](#)을 참조하십시오. GitHub

2. Helm에 eks-charts 리포지토리를 추가합니다.

```
helm repo add eks https://aws.github.io/eks-charts
```

3. App Mesh 쿠버네티스 사용자 지정 리소스 정의 () 를 설치합니다. CRD

```
kubectl apply -k "https://github.com/aws/eks-charts/stable/appmesh-controller/crds?ref=master"
```

4. 컨트롤러에 대한 Kubernetes 네임스페이스를 만듭니다.

```
kubectl create ns appmesh-system
```

5. 이후 단계에서 사용할 수 있도록 다음 변수를 설정합니다. *cluster-name* 및 *Region-code*를 기존 클러스터의 값으로 바꿉니다.

```
export CLUSTER_NAME=cluster-name
export AWS_REGION=Region-code
```

6. (선택 사항) Fargate에서 컨트롤러를 실행하려면 Fargate 프로필을 생성해야 합니다. 아직 eksctl 설치하지 않은 경우 Amazon EKS 사용 설명서의 [설치 또는 업그레이드를 eksctl](#) 참조하십시오. 콘솔을 사용하여 프로필을 생성하려면 Amazon 사용 EKS 설명서의 [Fargate 프로필 생성을](#) 참조하십시오.

```
eksctl create fargateprofile --cluster $CLUSTER_NAME --name appmesh-system --namespace appmesh-system
```

7. 클러스터의 OpenID Connect (OIDC) 아이덴티티 공급자를 생성합니다. 아직 eksctl 설치하지 않은 경우 Amazon 사용 EKS 설명서의 [설치 또는 업그레이드 eksctl 지침에 따라 설치할 수 있습니다](#). 콘솔을 사용하여 공급자를 생성하려는 경우 Amazon EKS User Guide의 [클러스터의 서비스 계정에 대한 IAM 역할 활성화](#)를 참조하십시오.

```
eksctl utils associate-iam-oidc-provider \
  --region=$AWS_REGION \
  --cluster $CLUSTER_NAME \
  --approve
```

8. IAM 역할을 생성하고, 여기에 [AWSCloudMapFullAccess](#) AWS 관리형 정책을 연결하고, appmesh-controller Kubernetes 서비스 계정에 바인딩하십시오. [AWSAppMeshFullAccess](#) 이 역할을 사용하면 컨트롤러가 App Mesh 리소스를 추가, 제거 및 변경할 수 있습니다.

Note

이 명령은 자동 생성된 이름을 사용하여 AWS IAM 역할을 생성합니다. 생성된 IAM 역할 이름은 지정할 수 없습니다.

```
eksctl create iamserviceaccount \
  --cluster $CLUSTER_NAME \
  --namespace appmesh-system \
  --name appmesh-controller \
  --attach-policy-arn arn:aws:iam::aws:policy/
AWSCloudMapFullAccess,arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
  --override-existing-serviceaccounts \
  --approve
```

AWS Management Console AWS CLI OR를 사용하여 서비스 계정을 생성하려는 경우 Amazon EKS User Guide의 [서비스 계정에 대한 IAM 역할 및 정책 생성](#)을 참조하십시오. OR를 사용하여 계정을 생성하는 경우 Kubernetes 서비스 계정에도 역할을 매핑해야 합니다. AWS Management Console AWS CLI 자세한 내용은 Amazon EKS 사용 설명서의 [서비스 계정 IAM 역할 지정](#)을 참조하십시오.

9. App Mesh 컨트롤러를 배포합니다. 모든 구성 옵션 목록은 구성 [설정을](#) 참조하십시오 GitHub.
 1. 프라이빗 클러스터용 App Mesh 컨트롤러를 배포하려면 먼저 연결된 프라이빗 서브넷에 대한 App Mesh 및 서비스 검색 Amazon VPC 엔드포인트를 활성화해야 합니다. 또한 accountId를 설정해야 합니다.

```
--set accountId=$AWS_ACCOUNT_ID
```

프라이빗 클러스터에서 X-Ray 추적을 활성화하려면 X-Ray 및 Amazon ECR Amazon VPC 엔드포인트를 활성화하십시오. 컨트롤러는 public.ecr.aws/xray/aws-xray-daemon:latest 기본적으로 를 사용하므로 이 이미지를 로컬로 가져와서 [개인 ECR 저장소로 푸시하십시오](#).

Note

[Amazon VPC 엔드포인트](#)는 현재 Amazon ECR 퍼블릭 리포지토리를 지원하지 않습니다.

다음 예에서는 X-Ray용 구성을 사용하여 컨트롤러를 배포하는 방법을 보여줍니다.

```
helm upgrade -i appmesh-controller eks/appmesh-controller \
  --namespace appmesh-system \
  --set region=$AWS_REGION \
  --set serviceAccount.create=false \
  --set serviceAccount.name=appmesh-controller \
  --set accountId=$AWS_ACCOUNT_ID \
  --set log.level=debug \
  --set tracing.enabled=true \
  --set tracing.provider=x-ray \
  --set xray.image.repository=your-account-id.dkr.ecr.your-
region.amazonaws.com/your-repository \
  --set xray.image.tag=your-xray-daemon-image-tag
```

애플리케이션 배포를 가상 노드 또는 게이트웨이에 바인딩할 때 X-Ray 데몬(daemon)이 성공적으로 삽입되었는지 확인합니다.

자세한 내용은 Amazon EKS 사용 설명서의 [프라이빗 클러스터를](#) 참조하십시오.

2. 다른 클러스터용 App Mesh 컨트롤러를 배포합니다. 모든 구성 옵션 목록은 [구성](#) 쉼표를 참조하십시오 GitHub.

```
helm upgrade -i appmesh-controller eks/appmesh-controller \
  --namespace appmesh-system \
  --set region=$AWS_REGION \
  --set serviceAccount.create=false \
  --set serviceAccount.name=appmesh-controller
```

Note

Amazon EKS 클러스터 패밀리인 경우 IPv6, App Mesh 컨트롤러를 배포할 때 이전 명령에 `--set clusterName=$CLUSTER_NAME` 다음 옵션을 추가하여 클러스터 이름을 설정하십시오.

Important

클러스터가 `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1` 또는 `af-south-1` 리전에 있는 경우, 이전 명령에 다음 옵션을 추가해야 합니다.

Replace `account-id` 그리고 `Region-code` 적절한 값 집합 중 하나를 사용하여

- 사이트카 이미지의 경우:

- ```
--set image.repository=account-id.dkr.ecr.Region-code.amazonaws.com/amazon/appmesh-controller
```
- `772975370895.dkr.ecr.me-south-1.amazonaws.com /:v1.29.6.1-prod aws-appmesh-envoy`
- `856666278305.dkr.ecr.ap-east-1.amazonaws.com /:v1.29.6.1-prod aws-appmesh-envoy`
- `909464085924.dkr.ecr.ap-southeast-3.amazonaws.com /:v1.29.6.1-prod aws-appmesh-envoy`
- `422531588944.dkr.ecr.eu-south-1.amazonaws.com /:v1.29.6.1-prod aws-appmesh-envoy`
- `564877687649.dkr.ecr.il-central-1.amazonaws.com /:v1.29.6.1-prod aws-appmesh-envoy`
- `924023996002.dkr.ecr.af-south-1.amazonaws.com /:v1.29.6.1-prod aws-appmesh-envoy`
- [이전 이미지는 변경 로그온에서 URIs 찾을 수 있습니다.](#) GitHub 이미지가 있는 AWS 계정의 버전이 변경되었습니다 `v1.5.0`. 이전 버전의 이미지는 Amazon Elastic Kubernetes Service [Amazon 컨테이너 이미지 레지스트리](#)에 있는 AWS 계정에 호스팅됩니다.

- 컨트롤러 이미지의 경우:

- ```
--set sidecar.image.repository=account-id.dkr.ecr.Region-code.amazonaws.com/aws-appmesh-envoy
```

- 772975370895.dkr.ecr.me-south-1.amazonaws.com /amazon/appmesh-controller:v1.13.1
- 856666278305.dkr.ecr.ap-east-1.amazonaws.com /amazon/appmesh-controller:v1.13.1
- 909464085924.dkr.ecr.ap-southeast-3.amazonaws.com /amazon/appmesh-controller:v1.13.1
- 422531588944.dkr.ecr.eu-south-1.amazonaws.com /아마존/앱메시-컨트롤러:v1.13.1
- 564877687649.dkr.ecr.il-central-1.amazonaws.com /amazon/appmesh-controller:v1.13.1
- 924023996002.dkr.ecr.af-south-1.amazonaws.com /아마존/앱메시-컨트롤러:v1.13.1

- 사이드카 초기화 이미지의 경우:

- ```
--set sidecar.image.repository=account-id.dkr.ecr.Region-code.amazonaws.com/aws-appmesh-envoy
```

- 772975370895.dkr.ecr.me-south-1.amazonaws.com/-매니저:v7-prod aws-appmesh-proxy-route
- 856666278305.dkr.ecr.ap-east-1.amazonaws.com/-매니저:v7-prod aws-appmesh-proxy-route
- 909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/-manager:v7-prod aws-appmesh-proxy-route
- 422531588944.dkr.ecr.eu-south-1.amazonaws.com/-manager:v7-prod aws-appmesh-proxy-route
- 564877687649.dkr.ecr.il-central-1.amazonaws.com/-매니저: v7-prod aws-appmesh-proxy-route
- 924023996002.dkr.ecr.af-south-1.amazonaws.com/-매니저: v7-prod aws-appmesh-proxy-route

**⚠ Important**

v1.9.0.0-prod 이상 버전만 App Mesh에서 사용할 수 있습니다.

10. 컨트롤러 버전이 v1.4.0 이상인지 확인합니다. [변경 로그온을 검토할 수 있습니다](#). GitHub

```
kubectl get deployment appmesh-controller \
 -n appmesh-system \
 -o json | jq -r ".spec.template.spec.containers[].image" | cut -f2 -d ':'
```

**i Note**

실행 중인 컨테이너에 대한 로그를 보면 무시해도 되는 다음 텍스트가 포함된 줄이 표시될 수 있습니다.

```
Neither -kubeconfig nor -master was specified. Using the inClusterConfig.
This might not work.
```

## 2단계: App Mesh 리소스 배포

Kubernetes에 애플리케이션을 배포할 때 Kubernetes 사용자 지정 리소스도 생성되므로 컨트롤러가 해당 App Mesh 리소스를 생성할 수 있습니다. 다음 절차는 일부 기능을 사용하여 App Mesh 리소스를 배포하는 데 도움이 됩니다. [다른 App Mesh 리소스 기능을 배포하기 위한 예제 매니페스트는 App Mesh 안내 페이지에 나열된 여러 기능 v1beta2 폴더의 하위 폴더에서 찾을 수 있습니다](#). GitHub

**⚠ Important**

컨트롤러가 App Mesh 리소스를 생성한 후에는 컨트롤러를 사용하여 해당 App Mesh 리소스만 변경하거나 삭제하는 것이 좋습니다. App Mesh를 사용하여 리소스를 변경하거나 삭제하면 컨트롤러는 기본적으로 10시간 동안 변경되거나 삭제된 App Mesh 리소스를 변경하거나 다시 생성하지 않습니다. 이 기간을 더 짧게 구성할 수 있습니다. [자세한 내용은 구성 설정을 참조하십시오](#). GitHub

## 앱 메시 리소스를 배포하려면

1. App Mesh 리소스를 배포할 Kubernetes 네임스페이스를 생성합니다.

a. 다음 콘텐츠를 컴퓨터에 namespace.yaml이라는 파일에 저장합니다.

```
apiVersion: v1
kind: Namespace
metadata:
 name: my-apps
 labels:
 mesh: my-mesh
 appmesh.k8s.aws/sidecarInjectorWebhook: enabled
```

b. 네임스페이스를 생성합니다.

```
kubectl apply -f namespace.yaml
```

2. App Mesh 서비스 메시지를 생성합니다.

a. 다음 콘텐츠를 컴퓨터에 mesh.yaml이라는 파일에 저장합니다. 이 파일은 *my-mesh*라는 메시 리소스를 생성하는 데 사용됩니다. 서비스 메시는 내부에 있는 서비스 간의 네트워크 트래픽에 대한 논리적 경계입니다.

```
apiVersion: appmesh.k8s.aws/v1beta2
kind: Mesh
metadata:
 name: my-mesh
spec:
 namespaceSelector:
 matchLabels:
 mesh: my-mesh
```

b. 메시지를 생성합니다.

```
kubectl apply -f mesh.yaml
```

c. 생성된 Kubernetes 메시 리소스의 세부 정보를 봅니다.

```
kubectl describe mesh my-mesh
```

### 출력

```

Name: my-mesh
Namespace:
Labels: <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
 {"apiVersion":"appmesh.k8s.aws/v1beta2","kind":"Mesh","metadata":{"annotations":{},"name":"my-mesh"},"spec":
 {"namespaceSelector":{"matchLa...
API Version: appmesh.k8s.aws/v1beta2
Kind: Mesh
Metadata:
 Creation Timestamp: 2020-06-17T14:51:37Z
 Finalizers:
 finalizers.appmesh.k8s.aws/mesh-members
 finalizers.appmesh.k8s.aws/aws-appmesh-resources
 Generation: 1
 Resource Version: 6295
 Self Link: /apis/appmesh.k8s.aws/v1beta2/meshes/my-mesh
 UID: 111a11b1-c11d-1e1f-gh1i-j11k11111m711
Spec:
 Aws Name: my-mesh
 Namespace Selector:
 Match Labels:
 Mesh: my-mesh
Status:
 Conditions:
 Last Transition Time: 2020-06-17T14:51:37Z
 Status: True
 Type: MeshActive
 Mesh ARN: arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh
 Observed Generation: 1
Events: <none>

```

- d. 컨트롤러가 생성한 App Mesh 서비스 메시에 대한 세부 정보를 봅니다.

```
aws appmesh describe-mesh --mesh-name my-mesh
```

#### 출력

```
{
 "mesh": {
 "meshName": "my-mesh",
 "metadata": {
```

```

 "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh",
 "createdAt": "2020-06-17T09:51:37.920000-05:00",
 "lastUpdatedAt": "2020-06-17T09:51:37.920000-05:00",
 "meshOwner": "111122223333",
 "resourceOwner": "111122223333",
 "uid": "111a11b1-c11d-1e1f-gh1i-j11k11111m711",
 "version": 1
 },
 "spec": {},
 "status": {
 "status": "ACTIVE"
 }
}
}

```

3. 앱 메시 가상 노드를 만듭니다. 가상 노드는 Kubernetes 배포에 대한 논리적 포인터 역할을 합니다.
  - a. 다음 콘텐츠를 컴퓨터에 `virtual-node.yaml`이라는 파일에 저장합니다. 이 파일은 `my-service-a`에서 이름이 지정된 App Mesh 가상 노드를 만드는 데 사용됩니다. `##-#` 네임스페이스. 가상 노드는 이후 단계에서 만드는 Kubernetes 서비스를 나타냅니다. 의 값은 이 가상 노드가 나타내는 실제 서비스의 정규화된 DNS 호스트 이름입니다. `hostname`

```

apiVersion: appmesh.k8s.aws/v1beta2
kind: VirtualNode
metadata:
 name: my-service-a
 namespace: my-apps
spec:
 podSelector:
 matchLabels:
 app: my-app-1
 listeners:
 - portMapping:
 port: 80
 protocol: http
 serviceDiscovery:
 dns:
 hostname: my-service-a.my-apps.svc.cluster.local

```

가상 노드에는 이 자습서에서 다루지 않는 end-to-end 암호화 및 상태 확인과 같은 기능이 있습니다. 자세한 내용은 [the section called “가상 노드”](#) 단원을 참조하십시오. 앞의 사양에서 설정할 수 있는 가상 노드에 사용 가능한 모든 설정을 보려면 다음 명령을 실행합니다.

```
aws appmesh create-virtual-node --generate-cli-skeleton yml-input
```

- b. 가상 노드를 배포합니다.

```
kubectl apply -f virtual-node.yml
```

- c. 생성된 Kubernetes 가상 노드 리소스의 세부 정보를 봅니다.

```
kubectl describe virtualnode my-service-a -n my-apps
```

## 출력

```
Name: my-service-a
Namespace: my-apps
Labels: <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
 {"apiVersion":"appmesh.k8s.aws/v1beta2", "kind":"VirtualNode", "metadata":{"annotations":{},"name":"my-service-a", "namespace":"my-apps"},"s...
API Version: appmesh.k8s.aws/v1beta2
Kind: VirtualNode
Metadata:
 Creation Timestamp: 2020-06-17T14:57:29Z
 Finalizers:
 finalizers.appmesh.k8s.aws/aws-appmesh-resources
 Generation: 2
 Resource Version: 22545
 Self Link: /apis/appmesh.k8s.aws/v1beta2/namespaces/my-apps/virtualnodes/my-service-a
 UID: 111a11b1-c11d-1e1f-gh1i-j11k11111m711
Spec:
 Aws Name: my-service-a_my-apps
 Listeners:
 Port Mapping:
 Port: 80
 Protocol: http
 Mesh Ref:
```

```

Name: my-mesh
UID: 111a11b1-c11d-1e1f-gh1i-j11k11111m711
Pod Selector:
Match Labels:
 App: nginx
Service Discovery:
 Dns:
 Hostname: my-service-a.my-apps.svc.cluster.local
Status:
Conditions:
 Last Transition Time: 2020-06-17T14:57:29Z
 Status: True
 Type: VirtualNodeActive
Observed Generation: 2
Virtual Node ARN: arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualNode/my-service-a_my-apps
Events: <none>

```

- d. 컨트롤러가 App Mesh에 생성한 가상 노드의 세부 정보를 봅니다.

#### Note

Kubernetes에서 생성된 가상 노드의 이름은 *my-service-a*이지만, App Mesh에서 생성한 가상 노드의 이름은 *my-service-a\_my-apps*입니다. 컨트롤러는 App Mesh 리소스를 생성할 때 App Mesh 가상 노드 이름에 Kubernetes 네임스페이스 이름을 추가합니다. Kubernetes에서는 서로 다른 네임스페이스에 같은 이름의 가상 노드를 만들 수 있지만, App Mesh에서 가상 노드 이름은 메시 내에서 고유해야 하기 때문에 네임스페이스 이름이 추가됩니다.

```
aws appmesh describe-virtual-node --mesh-name my-mesh --virtual-node-name my-service-a_my-apps
```

#### 출력

```

{
 "virtualNode": {
 "meshName": "my-mesh",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualNode/my-service-a_my-apps",

```

```

 "createdAt": "2020-06-17T09:57:29.840000-05:00",
 "lastUpdatedAt": "2020-06-17T09:57:29.840000-05:00",
 "meshOwner": "111122223333",
 "resourceOwner": "111122223333",
 "uid": "111a11b1-c11d-1e1f-gh1i-j11k11111m711",
 "version": 1
 },
 "spec": {
 "backends": [],
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http"
 }
 }
],
 "serviceDiscovery": {
 "dns": {
 "hostname": "my-service-a.my-apps.svc.cluster.local"
 }
 }
 },
 "status": {
 "status": "ACTIVE"
 },
 "virtualNodeName": "my-service-a_my-apps"
}
}

```

4. App Mesh 가상 라우터를 생성합니다. 가상 라우터는 메시 내에 있는 하나 이상의 가상 서비스에 대한 트래픽을 처리합니다.
  - a. 다음 콘텐츠를 컴퓨터에 `virtual-router.yaml`이라는 파일에 저장합니다. 이 파일은 이전 단계에서 생성된 `my-service-a`라는 가상 노드로 트래픽을 라우팅하는 가상 라우터를 생성하는 데 사용됩니다. 컨트롤러는 App Mesh 가상 라우터 및 경로 리소스를 생성합니다. 경로에 대해 더 많은 기능을 지정하고 http 이외의 프로토콜을 사용할 수 있습니다. 자세한 내용은 [the section called “가상 라우터”](#) 및 [the section called “경로”](#) 단원을 참조하세요. 참조된 가상 노드 이름은 컨트롤러에서 App Mesh에 생성한 App Mesh 가상 노드 이름이 아니라 Kubernetes 가상 노드 이름입니다.

```
apiVersion: appmesh.k8s.aws/v1beta2
```

```

kind: VirtualRouter
metadata:
 namespace: my-apps
 name: my-service-a-virtual-router
spec:
 listeners:
 - portMapping:
 port: 80
 protocol: http
 routes:
 - name: my-service-a-route
 httpRoute:
 match:
 prefix: /
 action:
 weightedTargets:
 - virtualNodeRef:
 name: my-service-a
 weight: 1

```

(선택 사항) 앞의 사양에서 설정할 수 있는 가상 라우터에 사용 가능한 모든 설정을 보려면 다음 명령 중 하나를 실행합니다.

```
aws appmesh create-virtual-router --generate-cli-skeleton yaml-input
```

앞의 사양에서 설정할 수 있는 경로에 사용 가능한 모든 설정을 보려면 다음 명령 중 하나를 실행합니다.

```
aws appmesh create-route --generate-cli-skeleton yaml-input
```

- b. 가상 라우터를 배포합니다.

```
kubectl apply -f virtual-router.yaml
```

- c. 생성된 Kubernetes 가상 라우터 리소스를 봅니다.

```
kubectl describe virtualrouter my-service-a-virtual-router -n my-apps
```

간략한 출력

```
Name: my-service-a-virtual-router
```

```
Namespace: my-apps
Labels: <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
 {"apiVersion":"appmesh.k8s.aws/v1beta2", "kind":"VirtualRouter", "metadata":{"annotations":{}}, "name":"my-
service-a-virtual-router", "namespac...
API Version: appmesh.k8s.aws/v1beta2
Kind: VirtualRouter
...
Spec:
 Aws Name: my-service-a-virtual-router_my-apps
 Listeners:
 Port Mapping:
 Port: 80
 Protocol: http
 Mesh Ref:
 Name: my-mesh
 UID: 111a11b1-c11d-1e1f-gh1i-j11k11111m711
 Routes:
 Http Route:
 Action:
 Weighted Targets:
 Virtual Node Ref:
 Name: my-service-a
 Weight: 1
 Match:
 Prefix: /
 Name: my-service-a-route
Status:
 Conditions:
 Last Transition Time: 2020-06-17T15:14:01Z
 Status: True
 Type: VirtualRouterActive
 Observed Generation: 1
 Route AR Ns:
 My - Service - A - Route: arn:aws:appmesh:us-west-2:111122223333:mesh/my-
mesh/virtualRouter/my-service-a-virtual-router_my-apps/route/my-service-a-route
 Virtual Router ARN: arn:aws:appmesh:us-west-2:111122223333:mesh/my-
mesh/virtualRouter/my-service-a-virtual-router_my-apps
 Events: <none>
```

- d. 컨트롤러가 App Mesh에 생성한 가상 라우터 리소스를 봅니다. 컨트롤러가 App Mesh에서 가상 라우터를 생성할 때 Kubernetes 네임스페이스 이름을 가상 라우터의 이름에 추가했기 때문에 name에 대해 `my-service-a-virtual-router_my-apps`를 지정합니다.

```
aws appmesh describe-virtual-router --virtual-router-name my-service-a-virtual-router_my-apps --mesh-name my-mesh
```

### 출력

```
{
 "virtualRouter": {
 "meshName": "my-mesh",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/virtualRouter/my-service-a-virtual-router_my-apps",
 "createdAt": "2020-06-17T10:14:01.547000-05:00",
 "lastUpdatedAt": "2020-06-17T10:14:01.547000-05:00",
 "meshOwner": "111122223333",
 "resourceOwner": "111122223333",
 "uid": "111a11b1-c11d-1e1f-gh1i-j11k11111m711",
 "version": 1
 },
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http"
 }
 }
]
 },
 "status": {
 "status": "ACTIVE"
 },
 "virtualRouterName": "my-service-a-virtual-router_my-apps"
 }
}
```

- e. 컨트롤러가 App Mesh에 생성한 경로 리소스를 봅니다. 경로가 Kubernetes에서 가상 라우터 구성의 일부이기 때문에 경로 리소스가 Kubernetes에서 생성되지 않았습니다. 경로 정보는 하위 단계 c에서 Kubernetes 리소스 세부 정보에 표시되었습니다. 경로 이름은 가상 라

우터에 고유하기 때문에 컨트롤러는 App Mesh에서 경로를 생성할 App Mesh 경로 이름에 Kubernetes 네임스페이스 이름을 추가하지 않았습니다.

```
aws appmesh describe-route \
 --route-name my-service-a-route \
 --virtual-router-name my-service-a-virtual-router_my-apps \
 --mesh-name my-mesh
```

## 출력

```
{
 "route": {
 "meshName": "my-mesh",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/virtualRouter/my-service-a-virtual-router_my-apps/route/my-service-a-route",
 "createdAt": "2020-06-17T10:14:01.577000-05:00",
 "lastUpdatedAt": "2020-06-17T10:14:01.577000-05:00",
 "meshOwner": "111122223333",
 "resourceOwner": "111122223333",
 "uid": "111a11b1-c11d-1e1f-gh1i-j11k11111m711",
 "version": 1
 },
 "routeName": "my-service-a-route",
 "spec": {
 "httpRoute": {
 "action": {
 "weightedTargets": [
 {
 "virtualNode": "my-service-a_my-apps",
 "weight": 1
 }
]
 },
 "match": {
 "prefix": "/"
 }
 }
 },
 "status": {
 "status": "ACTIVE"
 }
 },
}
```

```

 "virtualRouterName": "my-service-a-virtual-router_my-apps"
 }
}

```

5. App Mesh 가상 서비스를 생성합니다. 가상 서비스는 가상 노드가 가상 라우터를 통해 직접 또는 간접적으로 제공하는 실제 서비스의 추상화입니다. 종속 서비스는 가상 서비스를 이름으로 호출합니다. 이름은 App Mesh에 중요하지 않지만 가상 서비스의 이름을 가상 서비스가 나타내는 실제 서비스의 정규화된 도메인 이름으로 지정하는 것이 좋습니다. 이러한 방식으로 가상 서비스의 이름을 지정하면 다른 이름을 참조하도록 애플리케이션 코드를 변경할 필요가 없습니다. 이 요청은 가상 서비스의 공급자로 지정된 가상 노드 또는 가상 라우터로 라우팅됩니다.
  - a. 다음 콘텐츠를 컴퓨터에 `virtual-service.yaml`이라는 파일에 저장합니다. 이 파일은 가상 라우터 공급자를 사용하여 이전 단계에서 생성된 `my-service-a`라는 가상 노드로 트래픽을 라우팅하는 가상 서비스를 생성하는 데 사용됩니다. 의 값은 이 가상 서비스가 추상화하는 실제 Kubernetes 서비스의 정규화된 도메인 이름 (FQDN) spec 입니다. `awsName` Kubernetes 서비스는 [the section called “3단계: 서비스 생성 또는 업데이트”](#)에서 생성됩니다. 자세한 내용은 [the section called “가상 서비스”](#) 단원을 참조하십시오.

```

apiVersion: appmesh.k8s.aws/v1beta2
kind: VirtualService
metadata:
 name: my-service-a
 namespace: my-apps
spec:
 awsName: my-service-a.my-apps.svc.cluster.local
 provider:
 virtualRouter:
 virtualRouterRef:
 name: my-service-a-virtual-router

```

앞의 사양에서 설정할 수 있는 가상 서비스에 사용 가능한 모든 설정을 보려면 다음 명령을 실행합니다.

```
aws appmesh create-virtual-service --generate-cli-skeleton yaml-input
```

- b. 가상 서비스를 생성합니다.

```
kubectl apply -f virtual-service.yaml
```

- c. 생성된 Kubernetes 가상 서비스 리소스의 세부 정보를 봅니다.

```
kubectl describe virtualservice my-service-a -n my-apps
```

## 출력

```
Name: my-service-a
Namespace: my-apps
Labels: <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
 {"apiVersion":"appmesh.k8s.aws/v1beta2","kind":"VirtualService","metadata":{"annotations":{},"name":"my-
 service-a","namespace":"my-apps"}}...
API Version: appmesh.k8s.aws/v1beta2
Kind: VirtualService
Metadata:
 Creation Timestamp: 2020-06-17T15:48:40Z
 Finalizers:
 finalizers.appmesh.k8s.aws/aws-appmesh-resources
 Generation: 1
 Resource Version: 13598
 Self Link: /apis/appmesh.k8s.aws/v1beta2/namespaces/my-apps/
 virtualservices/my-service-a
 UID: 111a11b1-c11d-1e1f-gh1i-j11k11111m711
Spec:
 Aws Name: my-service-a.my-apps.svc.cluster.local
 Mesh Ref:
 Name: my-mesh
 UID: 111a11b1-c11d-1e1f-gh1i-j11k11111m711
 Provider:
 Virtual Router:
 Virtual Router Ref:
 Name: my-service-a-virtual-router
Status:
 Conditions:
 Last Transition Time: 2020-06-17T15:48:40Z
 Status: True
 Type: VirtualServiceActive
 Observed Generation: 1
 Virtual Service ARN: arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
 virtualservice/my-service-a.my-apps.svc.cluster.local
Events: <none>
```

- d. 컨트롤러가 App Mesh에 생성한 가상 서비스 리소스의 세부 정보를 봅니다. Kubernetes 컨트롤러는 App Mesh에서 가상 서비스를 생성할 때 가상 서비스 이름이 고유하기 때문에 App Mesh 가상 서비스 이름에 Kubernetes 네임스페이스 이름을 추가하지 않았습니다. FQDN

```
aws appmesh describe-virtual-service --virtual-service-name my-service-a.my-apps.svc.cluster.local --mesh-name my-mesh
```

## 출력

```
{
 "virtualService": {
 "meshName": "my-mesh",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/virtualService/my-service-a.my-apps.svc.cluster.local",
 "createdAt": "2020-06-17T10:48:40.182000-05:00",
 "lastUpdatedAt": "2020-06-17T10:48:40.182000-05:00",
 "meshOwner": "111122223333",
 "resourceOwner": "111122223333",
 "uid": "111a11b1-c11d-1e1f-gh1i-j11k11111m711",
 "version": 1
 },
 "spec": {
 "provider": {
 "virtualRouter": {
 "virtualRouterName": "my-service-a-virtual-router_my-apps"
 }
 }
 },
 "status": {
 "status": "ACTIVE"
 },
 "virtualServiceName": "my-service-a.my-apps.svc.cluster.local"
 }
}
```

이 자습서에서는 다루지 않지만 컨트롤러는 App Mesh [the section called “가상 게이트웨이”](#) 및 [the section called “게이트웨이 경로”](#)를 배포할 수도 있습니다. [컨트롤러를 사용하여 이러한 리소스를 배포하는 방법에 대한 자세한 내용은 인바운드 게이트웨이 구성 또는 리소스가 포함된 샘플 매니페스트를 참조하십시오.](#) [GitHub](#)

### 3단계: 서비스 생성 또는 업데이트

App Mesh에서 사용할 모든 포드에는 App Mesh 사이드카 컨테이너가 추가되어 있어야 합니다. 인젝터는 사용자가 지정한 레이블에 배포된 모든 포드에 사이드카 컨테이너를 자동으로 추가합니다.

1. 프록시 권한 부여를 활성화합니다. 각 Kubernetes 배포를 활성화하여 자체 App Mesh 가상 노드에 대한 구성을 스트리밍하는 것이 좋습니다.
  - a. 다음 콘텐츠를 컴퓨터에 proxy-auth.json이라는 파일에 저장합니다. 반드시 교체하십시오. *alternate-colored values* 직접 구매하세요.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "appmesh:StreamAggregatedResources",
 "Resource": [
 "arn:aws:appmesh:Region-code:111122223333:mesh/my-mesh/virtualNode/my-service-a_my-apps"
]
 }
]
}
```

- b. 정책을 생성합니다.

```
aws iam create-policy --policy-name my-policy --policy-document file://proxy-auth.json
```

- c. IAM역할을 생성하고, 이전 단계에서 만든 정책을 여기에 연결하고, Kubernetes 서비스 계정을 만들고, 정책을 Kubernetes 서비스 계정에 바인딩합니다. 이 역할을 사용하면 컨트롤러가 App Mesh 리소스를 추가, 제거 및 변경할 수 있습니다.

```
eksctl create iamserviceaccount \
 --cluster $CLUSTER_NAME \
 --namespace my-apps \
 --name my-service-a \
 --attach-policy-arn arn:aws:iam::111122223333:policy/my-policy \
 --override-existing-serviceaccounts \
 --approve
```

AWS Management Console AWS CLI OR를 사용하여 서비스 계정을 생성하려는 경우 Amazon 사용 EKS 설명서의 [서비스 계정에 대한 IAM 역할 및 정책 생성](#)을 참조하십시오. OR를 사용하여 계정을 생성하는 경우 Kubernetes 서비스 계정에도 역할을 매핑해야 합니다. AWS Management Console AWS CLI 자세한 내용은 Amazon EKS 사용 설명서의 [서비스 계정 IAM 역할 지정](#)을 참조하십시오.

2. (선택 사항) 배포를 Fargate 포드에 배포하려면 Fargate 프로필을 생성해야 합니다. 아직 eksctl 설치하지 않은 경우 Amazon 사용 EKS 설명서의 [설치 또는 업그레이드 eksctl 지침에 따라 설치](#)할 수 있습니다. 콘솔을 사용하여 프로필을 생성하려면 Amazon 사용 EKS 설명서의 [Fargate 프로필 생성](#)을 참조하십시오.

```
eksctl create fargateprofile --cluster my-cluster --region Region-code --name my-service-a --namespace my-apps
```

3. Kubernetes 서비스 및 배포를 만듭니다. App Mesh와 함께 사용하려는 기존 배포가 있는 경우 [the section called “2단계: App Mesh 리소스 배포”](#)의 하위 단계 3에서 수행한 것처럼 가상 노드를 배포해야 합니다. 배포의 레이블이 가상 노드에 설정한 레이블과 일치하도록 배포를 업데이트하여 사이드카 컨테이너가 자동으로 포드에 추가되고 포드가 재배포되도록 합니다.
  - a. 다음 콘텐츠를 컴퓨터에 example-service.yaml이라는 파일에 저장합니다. 네임스페이스 이름을 변경하고 Fargate 포드를 사용하는 경우 네임스페이스 이름이 Fargate 프로필에 정의한 네임스페이스 이름과 일치하는지 확인합니다.

```
apiVersion: v1
kind: Service
metadata:
 name: my-service-a
 namespace: my-apps
 labels:
 app: my-app-1
spec:
 selector:
 app: my-app-1
 ports:
 - protocol: TCP
 port: 80
 targetPort: 80

apiVersion: apps/v1
kind: Deployment
metadata:
```

```

name: my-service-a
namespace: my-apps
labels:
 app: my-app-1
spec:
 replicas: 3
 selector:
 matchLabels:
 app: my-app-1
 template:
 metadata:
 labels:
 app: my-app-1
 spec:
 serviceAccountName: my-service-a
 containers:
 - name: nginx
 image: nginx:1.19.0
 ports:
 - containerPort: 80

```

### ⚠ Important

사양에서 app matchLabels selector의 값은 [the section called “2단계: App Mesh 리소스 배포”](#)의 3 하위 단계에서 가상 노드를 생성할 때 지정한 값과 일치해야 합니다. 그렇지 않으면 사이드카 컨테이너가 포트에 삽입되지 않습니다. 앞의 예에서 레이블의 값은 `my-app-1`입니다. 가상 노드가 아닌 가상 게이트웨이를 배포하는 경우 Deployment 매니페스트에는 Envoy 컨테이너만 포함되어야 합니다. 사용할 이미지에 대한 자세한 내용은 [Envoy](#) 섹션을 참조하세요. 샘플 매니페스트는 [의 배포 예를](#) 참조하십시오. GitHub

- b. 서비스를 배포합니다.

```
kubectl apply -f example-service.yaml
```

- c. 서비스 및 배포를 봅니다.

```
kubectl -n my-apps get pods
```

출력

| NAME                          | READY | STATUS  | RESTARTS | AGE |
|-------------------------------|-------|---------|----------|-----|
| my-service-a-54776556f6-2cxd9 | 2/2   | Running | 0        | 10s |
| my-service-a-54776556f6-w26kf | 2/2   | Running | 0        | 18s |
| my-service-a-54776556f6-zw5kt | 2/2   | Running | 0        | 26s |

- d. 배포된 포드 중 하나에 대한 세부 정보를 봅니다.

```
kubectl -n my-apps describe pod my-service-a-54776556f6-2cxd9
```

### 간략한 출력

```
Name: my-service-a-54776556f6-2cxd9
Namespace: my-app-1
Priority: 0
Node: ip-192-168-44-157.us-west-2.compute.internal/192.168.44.157
Start Time: Wed, 17 Jun 2020 11:08:59 -0500
Labels: app=nginx
 pod-template-hash=54776556f6
Annotations: kubernetes.io/psp: eks.privileged
Status: Running
IP: 192.168.57.134
IPs:
 IP: 192.168.57.134
Controlled By: ReplicaSet/my-service-a-54776556f6
Init Containers:
 proxyinit:
 Container ID: docker://
e0c4810d584c21ae0cb6e40f6119d2508f029094d0e01c9411c6cf2a32d77a59
 Image: 111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
proxy-route-manager:v2
 Image ID: docker-pullable://111345817488.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-proxy-route-manager
 Port: <none>
 Host Port: <none>
 State: Terminated
 Reason: Completed
 Exit Code: 0
 Started: Fri, 26 Jun 2020 08:36:22 -0500
 Finished: Fri, 26 Jun 2020 08:36:22 -0500
 Ready: True
 Restart Count: 0
 Requests:
```

```

 cpu: 10m
 memory: 32Mi
 Environment:
 APPMESH_START_ENABLED: 1
 APPMESH_IGNORE_UID: 1337
 APPMESH_ENVOY_INGRESS_PORT: 15000
 APPMESH_ENVOY_EGRESS_PORT: 15001
 APPMESH_APP_PORTS: 80
 APPMESH_EGRESS_IGNORED_IP: 169.254.169.254
 APPMESH_EGRESS_IGNORED_PORTS: 22
 AWS_ROLE_ARN: arn:aws:iam::111122223333:role/eksctl-app-
mesh-addon-iamserviceaccount-my-a-Role1-NMNCVWB6PL0N
 AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
 ...
 Containers:
 nginx:
 Container ID: docker://
be6359dc6ecd3f18a1c87df7b57c2093e1f9db17d5b3a77f22585ce3bcab137a
 Image: nginx:1.19.0
 Image ID: docker-pullable://nginx
 Port: 80/TCP
 Host Port: 0/TCP
 State: Running
 Started: Fri, 26 Jun 2020 08:36:28 -0500
 Ready: True
 Restart Count: 0
 Environment:
 AWS_ROLE_ARN: arn:aws:iam::111122223333:role/eksctl-app-
mesh-addon-iamserviceaccount-my-a-Role1-NMNCVWB6PL0N
 AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
 ...
 envoy:
 Container ID:
docker://905b55cbf33ef3b3debc51cb448401d24e2e7c2dbfc6a9754a2c49dd55a216b6
 Image: 840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.12.4.0-prod
 Image ID: docker-pullable://840364872350.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-envoy
 Port: 9901/TCP
 Host Port: 0/TCP
 State: Running
 Started: Fri, 26 Jun 2020 08:36:36 -0500

```

```

Ready: True
Restart Count: 0
Requests:
 cpu: 10m
 memory: 32Mi
Environment:
 APPMESH_RESOURCE_ARN: arn:aws:iam::111122223333:mesh/my-mesh/
virtualNode/my-service-a_my-apps
 APPMESH_PREVIEW: 0
 ENVOY_LOG_LEVEL: info
 AWS_REGION: us-west-2
 AWS_ROLE_ARN: arn:aws:iam::111122223333:role/eksctl-app-
mesh-addon-iamserviceaccount-my-a-Role1-NMNCVWB6PL0N
 AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
...
Events:
 Type Reason Age From
 Message
 ---- -
 Normal Pulling 30s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Pulling image "111345817488.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-proxy-route-manager:v2"
 Normal Pulled 23s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Successfully pulled image "111345817488.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-proxy-route-manager:v2"
 Normal Created 21s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Created container proxyinit
 Normal Started 21s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Started container proxyinit
 Normal Pulling 20s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Pulling image "nginx:1.19.0"
 Normal Pulled 16s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Successfully pulled image "nginx:1.19.0"
 Normal Created 15s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Created container nginx
 Normal Started 15s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Started container nginx
 Normal Pulling 15s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Pulling image "840364872350.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-envoy:v1.12.4.0-prod"

```

```

Normal Pulled 8s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Successfully pulled image "840364872350.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-envoy:v1.12.4.0-prod"
Normal Created 7s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Created container envoy
Normal Started 7s kubelet, ip-192-168-44-157.us-
west-2.compute.internal Started container envoy

```

앞의 출력에서 proxyinit 및 envoy 컨테이너가 컨트롤러에 의해 포드에 추가되었는지 확인할 수 있습니다. 예제 서비스를 Fargate에 배포한 경우 컨트롤러에서 envoy 컨테이너를 포드에 추가했지만 proxyinit 컨테이너는 추가하지 않았습니다.

4. (선택 사항) 프로메테우스, 그라파나, 예거, 데이터독과 같은 애드온을 설치합니다. AWS X-Ray자세한 내용은 App [Mesh 애드온 GitHub](#) 및 App Mesh 사용 설명서의 [Observability](#) 섹션을 참조하십시오.

#### Note

App Mesh에 대한 더 많은 예제와 연습 내용을 보려면 [App Mesh 예제 리포지토리](#)를 참조하십시오.

## 4단계: 정리

이 자습서에서 만든 예제 리소스를 모두 제거합니다. 컨트롤러는 my-mesh App Mesh 서비스 메시에서 생성된 리소스도 제거합니다.

```
kubectl delete namespace my-apps
```

예제 서비스용 Fargate 프로필을 생성한 경우 해당 프로필을 제거하세요.

```
eksctl delete fargateprofile --name my-service-a --cluster my-cluster --region Region-code
```

메시를 삭제합니다.

```
kubectl delete mesh my-mesh
```

(선택 사항) Kubernetes 통합 구성 요소를 제거할 수 있습니다.

```
helm delete appmesh-controller -n appmesh-system
```

(선택 사항) Kubernetes 통합 구성 요소를 Fargate에 배포한 경우 Fargate 프로필을 삭제하세요.

```
eksctl delete fargateprofile --name appmesh-system --cluster my-cluster --
region Region-code
```

## 아마존 시작하기 AWS App Mesh EC2

이 주제는 Amazon에서 실행 중인 실제 서비스와 AWS App Mesh 함께 사용할 수 있도록 도와줍니다 EC2. 이 자습서에서는 몇 가지 App Mesh 리소스 유형의 기본 기능을 다룹니다.

### 시나리오

App Mesh를 사용하는 방법을 설명하기 위해 다음과 같은 특징을 가진 애플리케이션이 있다고 가정합니다.

- serviceA 및 serviceB라는 두 개의 서비스로 구성됩니다.
- 두 서비스 모두 apps.local이라는 네임스페이스에 등록됩니다.
- ServiceA HTTP/2, 포트 serviceB 80 이상과 통신합니다.
- serviceB의 버전 2를 이미 배포했고 apps.local 네임스페이스에 serviceBv2 이름으로 등록했습니다.

다음과 같은 요구 사항이 있습니다.

- 트래픽의 75% 를 serviceA ~로 보내고 25% 를 serviceB 첫 번째로 보내려고 serviceBv2 합니다. 25% 만 전송하면 트래픽의 100% 를 전송하기 전에 버그가 없는지 확인할 수 serviceA 있습니다. serviceBv2
- 트래픽이 신뢰할 수 있는 것으로 입증되면 해당 트래픽의 100%가 serviceBv2로 이동하도록 트래픽 가중치를 쉽게 조정할 수 있기를 원합니다. 모든 트래픽이 serviceBv2로 전송되면 serviceB를 중단하려고 할 수 있습니다.
- 이전 요구 사항을 충족하기 위해 실제 서비스에 대한 기존 애플리케이션 코드 또는 서비스 검색 등록을 변경할 필요가 없도록 하고 싶습니다.

요구 사항을 충족하기 위해 가상 서비스, 가상 노드, 가상 라우터 및 루트가 포함된 App Mesh 서비스 메시를 생성하기로 결정했습니다. 메시를 구현한 후 서비스가 Envoy 프록시를 사용하도록 서비스를

업데이트합니다. 업데이트된 서비스는 서로 직접 통신하지 않고 Envoy 프록시를 통해 서로 통신합니다.

## 사전 조건

App Mesh는 DNS AWS Cloud Map, 또는 둘 다에 등록된 리눅스 서비스를 지원합니다. 이 시작 안내서를 사용하려면 기존 서비스 3개를 등록해 두는 것이 좋습니다. DNS 서비스가 존재하지 않더라도 서비스 메시 및 해당 리소스를 생성할 수 있지만 실제 서비스를 배포할 때까지 메시지를 사용할 수 없습니다.

서비스를 아직 실행하지 않은 경우 Amazon 인스턴스를 시작하고 애플리케이션을 EC2 해당 인스턴스에 배포할 수 있습니다. 자세한 내용은 Amazon EC2 사용 [설명서의 자습서: Amazon EC2 Linux 인스턴스 시작하기](#)를 참조하십시오. 나머지 단계에서는 실제 서비스의 이름이 serviceA, serviceB 및 serviceBv2이고 apps.local이라는 네임스페이스를 통해 모든 서비스를 검색할 수 있다고 가정합니다.

## 1단계: 메시 및 가상 서비스 생성

서비스 메시는 내부에 있는 서비스 간의 네트워크 트래픽에 대한 논리적 경계입니다. 자세한 내용은 [서비스 메시](#) 단원을 참조하십시오. 가상 서비스는 실제 서비스를 추상화한 것입니다. 자세한 내용은 [가상 서비스](#) 단원을 참조하십시오.

다음의 리소스를 생성합니다.

- 시나리오의 모든 서비스가 apps.local 네임스페이스에 등록되기 때문에 apps이라는 메시지를 생성합니다.
- 가상 서비스는 해당 이름으로 검색할 수 있는 서비스를 나타내며 다른 이름을 참조하도록 코드를 변경하고 싶지 않기 때문에 serviceb.apps.local이라는 이름의 가상 서비스를 생성합니다. servicea.apps.local이라는 이름의 가상 서비스는 이후 단계에서 추가됩니다.

또는 AWS CLI 버전 1.18.116 이상 AWS Management Console 또는 2.0.38 이상을 사용하여 다음 단계를 완료할 수 있습니다. 를 AWS CLI 사용하는 경우 aws --version 명령을 사용하여 설치된 버전을 확인하십시오. AWS CLI 버전 1.18.116 이상 또는 2.0.38 이상이 설치되어 있지 않으면 [AWS CLI를 설치하거나 업데이트](#)해야 합니다. 사용할 도구의 탭을 선택합니다.

### AWS Management Console

1. 시작할 때 App Mesh 콘솔 최초 실행 마법사를 엽니다 <https://console.aws.amazon.com/appmesh/>.
2. Mesh name(메시 이름)에 **apps**를 입력합니다.

3. Virtual service name(가상 서비스 이름)에 **serviceb.apps.local**를 입력합니다.
4. 계속하려면 다음을 선택합니다.

## AWS CLI

1. [create-mesh](#) 명령을 사용하여 메시를 생성합니다.

```
aws appmesh create-mesh --mesh-name apps
```

2. [create-virtual-service](#) 명령을 사용하여 가상 서비스를 생성합니다.

```
aws appmesh create-virtual-service --mesh-name apps --virtual-service-name
serviceb.apps.local --spec {}
```

## 2단계: 가상 노드 생성

가상 노드는 실제 서비스에 대한 논리적 포인터 역할을 합니다. 자세한 내용은 [가상 노드](#) 단원을 참조하십시오.

가상 노드 중 하나가 serviceB라는 실제 서비스를 나타내므로 serviceB이라는 가상 노드를 생성합니다. 가상 노드가 나타내는 실제 서비스는 호스트 이름 serviceb.apps.local을 사용하여 DNS를 통해 검색할 수 있습니다. 또는 AWS Cloud Map을 사용하여 실제 서비스를 검색할 수 있습니다. 가상 노드는 포트 80에서 HTTP /2 프로토콜을 사용하여 트래픽을 수신합니다. 상태 확인과 마찬가지로 다른 프로토콜도 지원됩니다. 이후 단계에서 serviceA 및 serviceBv2에 대한 가상 노드를 생성합니다.

## AWS Management Console

1. Virtual node name(가상 노드 이름)에 **serviceB**를 입력합니다.
2. 서비스 검색 방법에서 호스트 이름을 DNS선택하고 **serviceb.apps.local** 입력합니다.  
DNS
3. 리스너 구성에서 프로토콜로 http2를 선택하고 포트로 **80**을 입력합니다.
4. 계속하려면 다음을 선택합니다.

## AWS CLI

1. 다음 콘텐츠를 가진 create-virtual-node-serviceb.json이라는 파일을 생성합니다:

```
{
 "meshName": "apps",
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http2"
 }
 }
],
 "serviceDiscovery": {
 "dns": {
 "hostname": "serviceB.apps.local"
 }
 }
 },
 "virtualNodeName": "serviceB"
}
```

2. JSON파일을 입력으로 사용하여 [create-virtual-node](#) 명령을 사용하여 가상 노드를 생성합니다.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-
serviceb.json
```

### 3단계: 가상 라우터 생성 및 라우팅

가상 라우터는 메시 내에 있는 하나 이상의 가상 서비스에 대한 트래픽을 라우팅합니다. 자세한 내용은 [가상 라우터](#) 및 [경로](#) 단원을 참조하세요.

다음의 리소스를 생성합니다.

- `serviceB`라는 이름의 가상 라우터. `serviceB.apps.local`이라는 가상 서비스는 다른 서비스와의 아웃바운드 통신을 시작하지 않기 때문입니다. 이전에 만든 가상 서비스는 실제 `serviceb.apps.local` 서비스를 추상화한 것입니다. 가상 서비스는 가상 라우터로 트래픽을 보냅니다. 가상 라우터는 포트 80에서 HTTP /2 프로토콜을 사용하여 트래픽을 수신합니다. 다른 프로토콜도 지원됩니다.
- `serviceB`라는 이름의 라우팅. 트래픽을 100% `serviceB` 가상 노드로 라우팅합니다. `serviceBv2` 가상 노드를 추가한 후 이후 단계에서 가중치가 변경됩니다. 이 안내서에서는 다루지

않지만 해당 라우팅에 대한 필터 조건을 추가하고 재시도 정책을 추가하여 통신 문제가 발생할 경우 Envoy 프록시가 가상 노드에 트래픽을 여러 번 보내도록 할 수 있습니다.

## AWS Management Console

1. Virtual router name(가상 라우터 이름)에 **serviceB**를 입력합니다.
2. 리스너 구성에서 프로토콜로 http2를 선택하고 포트로 **80**을 지정합니다.
3. Route Name(라우팅 이름)에 **serviceB**를 입력합니다.
4. 루트 유형에 대해 http2를 선택합니다.
5. 대상 구성의 가상 노드 이름에 대해 serviceB를 선택하고 가중치로 **100**을 입력합니다.
6. 일치 구성에서 방법을 선택합니다.
7. 계속하려면 다음을 선택합니다.

## AWS CLI

1. 가상 라우터를 생성합니다.
  - a. 다음 콘텐츠를 가진 `create-virtual-router.json`이라는 파일을 생성합니다:

```
{
 "meshName": "apps",
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http2"
 }
 }
]
 },
 "virtualRouterName": "serviceB"
}
```

- b. JSON파일을 입력으로 사용하여 [create-virtual-router](#) 명령을 사용하여 가상 라우터를 생성합니다.

```
aws appmesh create-virtual-router --cli-input-json file://create-virtual-router.json
```

2. 라우팅을 생성합니다.

a. 다음 콘텐츠를 가진 `create-route.json`이라는 파일을 생성합니다:

```
{
 "meshName" : "apps",
 "routeName" : "serviceB",
 "spec" : {
 "httpRoute" : {
 "action" : {
 "weightedTargets" : [
 {
 "virtualNode" : "serviceB",
 "weight" : 100
 }
]
 },
 "match" : {
 "prefix" : "/"
 }
 }
 },
 "virtualRouterName" : "serviceB"
}
```

b. JSON파일을 입력으로 사용하여 [create-route](#) 명령으로 경로를 생성합니다.

```
aws appmesh create-route --cli-input-json file://create-route.json
```

## 4단계: 검토 및 생성

이전 지침에 대한 설정을 검토합니다.

AWS Management Console

섹션에서 변경해야 할 경우 편집을 선택합니다. 설정에 만족하면 메시 생성을 선택합니다.

상태 화면에는 생성된 모든 메시 리소스가 표시됩니다. 메시 보기를 선택하여 콘솔에서 생성된 리소스를 볼 수 있습니다.

## AWS CLI

[describe-mesh](#) 명령을 사용하여 생성한 메시의 설정을 검토합니다.

```
aws appmesh describe-mesh --mesh-name apps
```

명령으로 만든 가상 서비스의 설정을 검토하십시오. [describe-virtual-service](#)

```
aws appmesh describe-virtual-service --mesh-name apps --virtual-service-name
serviceb.apps.local
```

[describe-virtual-node](#) 명령으로 만든 가상 노드의 설정을 검토하십시오.

```
aws appmesh describe-virtual-node --mesh-name apps --virtual-node-name serviceB
```

[describe-virtual-router](#) 명령으로 만든 가상 라우터의 설정을 검토하십시오.

```
aws appmesh describe-virtual-router --mesh-name apps --virtual-router-name serviceB
```

[describe-route](#) 명령으로 생성한 라우팅의 설정을 검토합니다.

```
aws appmesh describe-route --mesh-name apps \
--virtual-router-name serviceB --route-name serviceB
```

## 5단계: 추가 리소스 생성

시나리오를 완료하려면 다음을 수행해야 합니다.

- `serviceBv2`라는 가상 노드 하나와 `serviceA`라는 가상 노드 하나를 생성합니다. 두 가상 노드 모두 HTTP /2 포트 80을 통한 요청을 수신합니다. `serviceA` 가상 노드의 경우 `serviceb.apps.local`의 백엔드를 구성합니다. `serviceA` 가상 노드의 모든 아웃바운드 트래픽은 이름이 `serviceb.apps.local`인 가상 서비스로 전송됩니다. 이 안내서에서는 다루지 않지만 가상 노드에 대한 액세스 로그를 쓸 파일 경로를 지정할 수도 있습니다.
- `servicea.apps.local`이라는 가상 서비스를 추가로 생성하면 모든 트래픽이 `serviceA` 가상 노드로 직접 전송됩니다.

- 트래픽의 75%를 `serviceB` 가상 노드로 보내고 트래픽의 25%를 `serviceBv2` 가상 노드로 보내도록 이전 단계에서 생성한 `serviceB` 라우팅을 업데이트합니다. 시간이 지남에 따라 `serviceBv2`가 트래픽의 100%를 수신할 때까지 가중치를 계속 수정할 수 있습니다. 모든 트래픽이 `serviceBv2`로 전송되면 `serviceB` 가상 노드와 실제 서비스를 종료하고 중단할 수 있습니다. `serviceb.apps.local` 가상 및 실제 서비스 이름이 변경되지 않으므로 가중치를 변경할 때 코드를 수정할 필요가 없습니다. `serviceb.apps.local` 가상 서비스는 트래픽을 가상 라우터로 전송하여 트래픽을 가상 노드로 라우팅한다는 점을 기억하십시오. 가상 노드의 서비스 검색 이름은 언제든지 변경할 수 있습니다.

## AWS Management Console

1. 왼쪽 탐색 창에서 Meshes(메시)를 선택합니다.
2. 이전 단계에서 생성한 apps 메시를 선택합니다.
3. 왼쪽 탐색 창에서 Virtual node(가상 노드)를 선택합니다.
4. Create virtual node(가상 노드 생성)를 선택합니다.
5. 가상 노드 이름에는 을 입력하고 `serviceBv2`, 서비스 검색 방법에는 을 DNS, DNS호스트 이름에는 를 입력합니다. `servicebv2.apps.local`
6. 리스너 구성에서 프로토콜로 http2를 선택하고 포트로 **80**을 입력합니다.
7. Create virtual node(가상 노드 생성)를 선택합니다.
8. 가상 노드 생성을 다시 선택합니다. 가상 노드 이름에 `serviceA`를 입력합니다. 서비스 검색 방법에서는 을 선택하고 DNSDNS호스트 이름에는 를 입력합니다. `servicea.apps.local`
9. 새 백엔드에서 가상 서비스 이름 입력에 `serviceb.apps.local`을 입력합니다.
10. 리스너 구성에서 프로토콜로 http2를 선택하고 포트에 **80**을 입력한 다음, 가상 노드 생성을 선택합니다.
11. 왼쪽 탐색 창에서 Virtual routers(가상 라우터)를 선택한 다음 목록에서 `serviceB` 가상 라우터를 선택합니다.
12. 루트에서 이전 단계에서 생성한 `ServiceB`라는 루트를 선택하고 편집을 선택합니다.
13. 대상, 가상 노드 이름에서 `serviceB`의 가중치 값을 **75**로 변경합니다.
14. 대상 추가를 선택하고 드롭다운 목록에서 `serviceBv2`를 선택하고 가중치의 값을 **25**로 설정합니다.
15. 저장(Save)을 선택합니다.
16. 왼쪽 탐색 창에서 가상 서비스를 선택한 다음 가상 서비스 생성을 선택합니다.

17. 가상 서비스 이름으로 **servicea.apps.local**을 입력하고 공급자로 가상 노드를 선택하고 가상 노드로 **serviceA**를 선택한 다음, 가상 서비스 생성을 선택합니다.

## AWS CLI

1. **serviceBv2** 가상 노드를 생성합니다.
  - a. 다음 콘텐츠를 가진 `create-virtual-node-servicebv2.json`이라는 파일을 생성합니다:

```
{
 "meshName": "apps",
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http2"
 }
 }
],
 "serviceDiscovery": {
 "dns": {
 "hostname": "serviceBv2.apps.local"
 }
 }
 },
 "virtualNodeName": "serviceBv2"
}
```

- b. 가상 노드를 생성합니다.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-servicebv2.json
```

2. **serviceA** 가상 노드를 생성합니다.
  - a. 다음 콘텐츠를 가진 `create-virtual-node-servicea.json`이라는 파일을 생성합니다:

```
{
 "meshName" : "apps",
```

```

"spec" : {
 "backends" : [
 {
 "virtualService" : {
 "virtualServiceName" : "serviceb.apps.local"
 }
 }
],
 "listeners" : [
 {
 "portMapping" : {
 "port" : 80,
 "protocol" : "http2"
 }
 }
],
 "serviceDiscovery" : {
 "dns" : {
 "hostname" : "servicea.apps.local"
 }
 }
},
"virtualNodeName" : "serviceA"
}

```

- b. 가상 노드를 생성합니다.

```

aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-
servicea.json

```

3. 이전 단계에서 생성한 `serviceb.apps.local` 가상 서비스를 업데이트하여 트래픽을 `serviceB` 가상 라우터로 보냅니다. 가상 서비스가 원래 생성되었을 때는 `serviceB` 가상 라우터가 아직 생성되지 않았기 때문에 트래픽을 어디에도 보내지 않았습니다.

- a. 다음 콘텐츠를 가진 `update-virtual-service.json`이라는 파일을 생성합니다:

```

{
 "meshName" : "apps",
 "spec" : {
 "provider" : {
 "virtualRouter" : {
 "virtualRouterName" : "serviceB"
 }
 }
 }
}

```

```

 }
 },
 "virtualServiceName" : "serviceb.apps.local"
}

```

- b. [update-virtual-service](#) 명령을 사용하여 가상 서비스를 업데이트합니다.

```
aws appmesh update-virtual-service --cli-input-json file://update-virtual-service.json
```

4. 이전 단계에서 생성한 serviceB 라우팅을 업데이트합니다.

- a. 다음 콘텐츠를 가진 update-route.json이라는 파일을 생성합니다:

```

{
 "meshName" : "apps",
 "routeName" : "serviceB",
 "spec" : {
 "http2Route" : {
 "action" : {
 "weightedTargets" : [
 {
 "virtualNode" : "serviceB",
 "weight" : 75
 },
 {
 "virtualNode" : "serviceBv2",
 "weight" : 25
 }
]
 },
 "match" : {
 "prefix" : "/"
 }
 }
 },
 "virtualRouterName" : "serviceB"
}

```

- b. [update-route](#) 명령을 사용하여 라우팅을 업데이트합니다.

```
aws appmesh update-route --cli-input-json file://update-route.json
```

## 5. serviceA 가상 서비스를 생성합니다.

- a. 다음 콘텐츠를 가진 `create-virtual-servicea.json`이라는 파일을 생성합니다:

```
{
 "meshName" : "apps",
 "spec" : {
 "provider" : {
 "virtualNode" : {
 "virtualNodeName" : "serviceA"
 }
 }
 },
 "virtualServiceName" : "servicea.apps.local"
}
```

- b. 가상 서비스를 생성합니다.

```
aws appmesh create-virtual-service --cli-input-json file://create-virtual-servicea.json
```

## 메시 요약

서비스 메시지를 생성하기 전에는 `servicea.apps.local`, `serviceb.apps.local` 및 `servicebv2.apps.local`이라는 세 가지 실제 서비스가 있었습니다. 실제 서비스 외에도 이제 실제 서비스를 나타내는 다음 리소스가 포함된 서비스 메시가 있습니다.

- 두 개의 가상 서비스. 프록시는 가상 라우터를 통해 `servicea.apps.local` 가상 서비스의 모든 트래픽을 `serviceb.apps.local` 가상 서비스로 보냅니다.
- `serviceA`, `serviceB` 및 `serviceBv2`라는 세 개의 가상 노드. Envoy 프록시는 가상 노드에 대해 구성된 서비스 검색 정보를 사용하여 실제 서비스의 IP 주소를 조회합니다.
- Envoy 프록시가 인바운드 트래픽의 75%를 `serviceB` 가상 노드로 라우팅하고 트래픽의 25%를 `serviceBv2` 가상 노드로 라우팅하도록 지시하는 하나의 라우팅이 있는 가상 라우터 하나.

## 6단계: 서비스 업데이트

메시를 생성한 후에는 다음 작업을 완료해야 합니다.

- 각 서비스와 함께 배포하는 Envoy 프록시에 하나 이상의 가상 노드의 구성을 읽을 수 있는 권한을 부여합니다. 프록시에 권한을 부여하는 방법에 대한 자세한 내용은 [Envoy 프록시 권한 부여](#) 섹션을 참조하세요.
- 기존 서비스를 업데이트하려면 다음 단계를 완료하세요.

Amazon EC2 인스턴스를 가상 노드 구성원으로 구성하려면

## 1. IAM 역할을 생성합니다.

- 다음 콘텐츠를 가진 `ec2-trust-relationship.json`이라는 파일을 생성합니다:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "ec2.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

- 다음 명령으로 IAM 역할을 생성합니다.

```
aws iam create-role --role-name mesh-virtual-node-service-b --assume-role-policy-document file://ec2-trust-relationship.json
```

## 2. Amazon에서 읽을 수 있도록 허용하는 ECR 역할과 특정 App Mesh 가상 노드의 구성만 허용하는 IAM 정책을 역할에 연결합니다.

- 다음 콘텐츠를 통해 `virtual-node-policy.json`이라는 파일을 생성합니다. `apps`는 [the section called “1단계: 메시 및 가상 서비스 생성”](#)에서 생성한 메시의 이름이고, `serviceB`는 [the section called “2단계: 가상 노드 생성”](#)에서 생성한 가상 노드의 이름입니다. Replace **111122223333** 계정 ID와 함께 **us-west-2** 메시지를 생성한 지역과 함께

```
{
 "Version": "2012-10-17",
 "Statement": [
 {

```

```

 "Effect": "Allow",
 "Action": "appmesh:StreamAggregatedResources",
 "Resource": [
 "arn:aws:appmesh:us-west-2:111122223333:mesh/apps/
virtualNode/serviceB"
]
 }
]
}

```

- b. 다음 명령을 사용하여 정책을 생성합니다.

```

aws iam create-policy --policy-name virtual-node-policy --policy-document
file://virtual-node-policy.json

```

- c. 이전 단계에서 생성한 정책을 역할에 연결하여 역할이 App Mesh에서 *serviceB* 가상 노드에 대한 구성만 읽을 수 있도록 합니다.

```

aws iam attach-role-policy --policy-arn arn:aws:iam::111122223333:policy/
virtual-node-policy --role-name mesh-virtual-node-service-b

```

- d. Amazon에서 Envoy 컨테이너 이미지를 가져올 수 있도록 역할에 AmazonEC2ContainerRegistryReadOnly 관리형 정책을 연결합니다. ECR

```

aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly --role-name mesh-virtual-node-service-b

```

3. 생성한 [IAM 역할을 사용하여 Amazon EC2 인스턴스를 시작합니다.](#)
4. 를 통해 인스턴스에 연결합니다 SSH.
5. 운영 체제 설명서에 AWS CLI 따라 인스턴스에 Docker와 를 설치합니다.
6. Docker 클라이언트가 이미지를 가져오기를 원하는 지역의 Envoy ECR Amazon 리포지토리에 인증합니다.
  - me-south-1, ap-east-1, ap-southeast-3, eu-south-1, il-central-1 및 af-south-1을 제외한 모든 리전입니다. 다음을 대체할 수 있습니다. *us-west-2* me-south-1,, ap-east-1 ap-southeast-3 eu-south-1 il-central-1, 및 af-south-1 를 제외한 [지원되는 모든 지역으로](#)

```

$aws ecr get-login-password \
--region us-west-2 \

```

```
| docker login \
 --username AWS \
 --password-stdin 840364872350.dkr.ecr.us-west-2.amazonaws.com
```

- me-south-1 리전

```
$aws ecr get-login-password \
 --region me-south-1 \
| docker login \
 --username AWS \
 --password-stdin 772975370895.dkr.ecr.me-south-1.amazonaws.com
```

- ap-east-1 리전

```
$aws ecr get-login-password \
 --region ap-east-1 \
| docker login \
 --username AWS \
 --password-stdin 856666278305.dkr.ecr.ap-east-1.amazonaws.com
```

7. 이미지를 가져올 리전에 따라 다음 명령 중 하나를 실행하여 인스턴스에서 App Mesh Envoy 컨테이너를 시작합니다. The *apps* 그리고 *serviceB* 값은 시나리오에 정의된 메시 및 가상 노드 이름입니다. 이 정보는 프록시에 App Mesh에서 읽을 가상 노드 구성을 알려줍니다. 시나리오를 완료하려면 *serviceBv2* 및 *serviceA* 가상 노드로 표시되는 서비스를 호스팅하는 Amazon EC2 인스턴스에 대한 다음 단계도 완료해야 합니다. 자체 애플리케이션의 경우 이 값을 자체 값으로 바꿉니다.

- me-south-1, ap-east-1, ap-southeast-3, eu-south-1, il-central-1 및 af-south-1을 제외한 모든 리전입니다. 다음과 같이 교체할 수 있습니다. *Region-code* me-south-1,,, ap-east-1 ap-southeast-3 eu-south-1il-central-1, 및 [지역을 제외한 지원되는](#) 모든 af-south-1 지역으로 *1337*을 0과 2147483647 사이의 값으로 바꿀 수 있습니다.

```
sudo docker run --detach --env APPMESH_RESOURCE_ARN=mesh/apps/
virtualNode/serviceB \
-u 1337 --network host 840364872350.dkr.ecr.region-code.amazonaws.com/aws-
appmesh-envoy:v1.29.6.1-prod
```

- me-south-1 리전. *1337*을 0과 2147483647 사이의 값으로 바꿀 수 있습니다.

```
sudo docker run --detach --env APPMESH_RESOURCE_ARN=mesh/apps/
virtualNode/serviceB \
-u 1337 --network host 772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-
envoy:v1.29.6.1-prod
```

- ap-east-1 리전. 1337을 0과 2147483647 사이의 값으로 바꿀 수 있습니다.

```
sudo docker run --detach --env APPMESH_RESOURCE_ARN=mesh/apps/
virtualNode/serviceB \
-u 1337 --network host 856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-
envoy:v1.29.6.1-prod
```

### Note

APPMESH\_RESOURCE\_ARN 속성에는 1.15.0 이상의 Envoy 이미지 버전이 필요합니다. 자세한 내용은 [Envoy](#) 단원을 참조하십시오.

### Important

v1.9.0.0-prod 이상 버전만 App Mesh에서 사용할 수 있습니다.

8. 아래의 Show more를 선택합니다. 다음 콘텐츠를 통해 인스턴스에 envoy-networking.sh라는 파일을 생성합니다. Replace 8000 애플리케이션 코드가 들어오는 트래픽에 사용하는 포트를 사용합니다. APPMESH\_IGNORE\_UID의 값을 변경할 수 있지만 이 값은 이전 단계에서 지정한 값과 같아야 합니다(예: 1337). 필요한 경우 APPMESH\_EGRESS\_IGNORED\_IP에 주소를 추가할 수 있습니다. 어떠한 행동도 수정하지 마십시오.

```
#!/bin/bash -e

#
Start of configurable options
#

#APPMESH_START_ENABLED="0"
APPMESH_IGNORE_UID="1337"
APPMESH_APP_PORTS="8000"
```

```
APPMESH_ENVOY_EGRESS_PORT="15001"
APPMESH_ENVOY_INGRESS_PORT="15000"
APPMESH_EGRESS_IGNORED_IP="169.254.169.254,169.254.170.2"

Enable routing on the application start.
[-z "$APPMESH_START_ENABLED"] && APPMESH_START_ENABLED="0"

Enable IPv6.
[-z "$APPMESH_ENABLE_IPV6"] && APPMESH_ENABLE_IPV6="0"

Egress traffic from the processes owned by the following UID/GID will be
ignored.
if [-z "$APPMESH_IGNORE_UID"] && [-z "$APPMESH_IGNORE_GID"]; then
 echo "Variables APPMESH_IGNORE_UID and/or APPMESH_IGNORE_GID must be set."
 echo "Envoy must run under those IDs to be able to properly route it's egress
traffic."
 exit 1
fi

Port numbers Application and Envoy are listening on.
if [-z "$APPMESH_ENVOY_EGRESS_PORT"]; then
 echo "APPMESH_ENVOY_EGRESS_PORT must be defined to forward traffic from the
application to the proxy."
 exit 1
fi

If an app port was specified, then we also need to enforce the proxies ingress
port so we know where to forward traffic.
if [! -z "$APPMESH_APP_PORTS"] && [-z "$APPMESH_ENVOY_INGRESS_PORT"]; then
 echo "APPMESH_ENVOY_INGRESS_PORT must be defined to forward traffic from the
APPMESH_APP_PORTS to the proxy."
 exit 1
fi

Comma separated list of ports for which egress traffic will be ignored, we always
refuse to route SSH traffic.
if [-z "$APPMESH_EGRESS_IGNORED_PORTS"]; then
 APPMESH_EGRESS_IGNORED_PORTS="22"
else
 APPMESH_EGRESS_IGNORED_PORTS="$APPMESH_EGRESS_IGNORED_PORTS,22"
fi

#
End of configurable options
```

```
#

function initialize() {
 echo "=== Initializing ==="
 if [! -z "$APPMESH_APP_PORTS"]; then
 iptables -t nat -N APPMESH_INGRESS
 if ["$APPMESH_ENABLE_IPV6" == "1"]; then
 ip6tables -t nat -N APPMESH_INGRESS
 fi
 fi
 iptables -t nat -N APPMESH_EGRESS
 if ["$APPMESH_ENABLE_IPV6" == "1"]; then
 ip6tables -t nat -N APPMESH_EGRESS
 fi
}

function enable_egress_routing() {
 # Stuff to ignore
 [! -z "$APPMESH_IGNORE_UID"] && \
 iptables -t nat -A APPMESH_EGRESS \
 -m owner --uid-owner $APPMESH_IGNORE_UID \
 -j RETURN

 [! -z "$APPMESH_IGNORE_GID"] && \
 iptables -t nat -A APPMESH_EGRESS \
 -m owner --gid-owner $APPMESH_IGNORE_GID \
 -j RETURN

 [! -z "$APPMESH_EGRESS_IGNORED_PORTS"] && \
 for IGNORED_PORT in $(echo "$APPMESH_EGRESS_IGNORED_PORTS" | tr "," "\n");
do
 iptables -t nat -A APPMESH_EGRESS \
 -p tcp \
 -m multiport --dports "$IGNORED_PORT" \
 -j RETURN
 done

 if ["$APPMESH_ENABLE_IPV6" == "1"]; then
 # Stuff to ignore ipv6
 [! -z "$APPMESH_IGNORE_UID"] && \
 ip6tables -t nat -A APPMESH_EGRESS \
 -m owner --uid-owner $APPMESH_IGNORE_UID \
 -j RETURN
 fi
}
```

```

[! -z "$APPMESH_IGNORE_GID"] && \
 iptables -t nat -A APPMESH_EGRESS \
 -m owner --gid-owner $APPMESH_IGNORE_GID \
 -j RETURN

[! -z "$APPMESH_EGRESS_IGNORED_PORTS"] && \
 for IGNORED_PORT in $(echo "$APPMESH_EGRESS_IGNORED_PORTS" | tr "," "\n");
do
 iptables -t nat -A APPMESH_EGRESS \
 -p tcp \
 -m multiport --dports "$IGNORED_PORT" \
 -j RETURN
done
fi

The list can contain both IPv4 and IPv6 addresses. We will loop over this
list
to add every IPv4 address into `iptables` and every IPv6 address into
`ip6tables`.
[! -z "$APPMESH_EGRESS_IGNORED_IP"] && \
 for IP_ADDR in $(echo "$APPMESH_EGRESS_IGNORED_IP" | tr "," "\n"); do
 if [[$IP_ADDR =~ .*:.*]]
 then
 ["$APPMESH_ENABLE_IPV6" == "1"] && \
 ip6tables -t nat -A APPMESH_EGRESS \
 -p tcp \
 -d "$IP_ADDR" \
 -j RETURN
 else
 iptables -t nat -A APPMESH_EGRESS \
 -p tcp \
 -d "$IP_ADDR" \
 -j RETURN
 fi
 done

Redirect everything that is not ignored
iptables -t nat -A APPMESH_EGRESS \
 -p tcp \
 -j REDIRECT --to $APPMESH_ENVYOY_EGRESS_PORT

Apply APPMESH_EGRESS chain to non local traffic
iptables -t nat -A OUTPUT \
 -p tcp \

```

```

-m addrtype ! --dst-type LOCAL \
-j APPMESH_EGRESS

if ["$APPMESH_ENABLE_IPV6" == "1"]; then
Redirect everything that is not ignored ipv6
ip6tables -t nat -A APPMESH_EGRESS \
-p tcp \
-j REDIRECT --to $APPMESH_ENVOY_EGRESS_PORT
Apply APPMESH_EGRESS chain to non local traffic ipv6
ip6tables -t nat -A OUTPUT \
-p tcp \
-m addrtype ! --dst-type LOCAL \
-j APPMESH_EGRESS
fi
}

function enable_ingress_redirect_routing() {
Route everything arriving at the application port to Envoy
iptables -t nat -A APPMESH_INGRESS \
-p tcp \
-m multiport --dports "$APPMESH_APP_PORTS" \
-j REDIRECT --to-port "$APPMESH_ENVOY_INGRESS_PORT"

Apply AppMesh ingress chain to everything non-local
iptables -t nat -A PREROUTING \
-p tcp \
-m addrtype ! --src-type LOCAL \
-j APPMESH_INGRESS

if ["$APPMESH_ENABLE_IPV6" == "1"]; then
Route everything arriving at the application port to Envoy ipv6
ip6tables -t nat -A APPMESH_INGRESS \
-p tcp \
-m multiport --dports "$APPMESH_APP_PORTS" \
-j REDIRECT --to-port "$APPMESH_ENVOY_INGRESS_PORT"

Apply AppMesh ingress chain to everything non-local ipv6
ip6tables -t nat -A PREROUTING \
-p tcp \
-m addrtype ! --src-type LOCAL \
-j APPMESH_INGRESS
fi
}

```

```
function enable_routing() {
 echo "=== Enabling routing ==="
 enable_egress_routing
 if [! -z "$APPMESH_APP_PORTS"]; then
 enable_ingress_redirect_routing
 fi
}

function disable_routing() {
 echo "=== Disabling routing ==="
 iptables -t nat -F APPMESH_INGRESS
 iptables -t nat -F APPMESH_EGRESS

 if ["$APPMESH_ENABLE_IPV6" == "1"]; then
 ip6tables -t nat -F APPMESH_INGRESS
 ip6tables -t nat -F APPMESH_EGRESS
 fi
}

function dump_status() {
 echo "=== iptables FORWARD table ==="
 iptables -L -v -n
 echo "=== iptables NAT table ==="
 iptables -t nat -L -v -n

 if ["$APPMESH_ENABLE_IPV6" == "1"]; then
 echo "=== ip6tables FORWARD table ==="
 ip6tables -L -v -n
 echo "=== ip6tables NAT table ==="
 ip6tables -t nat -L -v -n
 fi
}

function clean_up() {
 disable_routing
 ruleNum=$(iptables -L PREROUTING -t nat --line-numbers | grep APPMESH_INGRESS |
 cut -d " " -f 1)
 iptables -t nat -D PREROUTING $ruleNum

 ruleNum=$(iptables -L OUTPUT -t nat --line-numbers | grep APPMESH_EGRESS | cut
 -d " " -f 1)
 iptables -t nat -D OUTPUT $ruleNum
}
```

```
iptables -t nat -X APPMESH_INGRESS
iptables -t nat -X APPMESH_EGRESS

if ["$APPMESH_ENABLE_IPV6" == "1"]; then
 ruleNum=$(ip6tables -L PREROUTING -t nat --line-numbers | grep
APPMESH_INGRESS | cut -d " " -f 1)
 ip6tables -t nat -D PREROUTING $ruleNum

 ruleNum=$(ip6tables -L OUTPUT -t nat --line-numbers | grep APPMESH_EGRESS |
cut -d " " -f 1)
 ip6tables -t nat -D OUTPUT $ruleNum

 ip6tables -t nat -X APPMESH_INGRESS
 ip6tables -t nat -X APPMESH_EGRESS
fi
}

function main_loop() {
 echo "=== Entering main loop ==="
 while read -p '> ' cmd; do
 case "$cmd" in
 "quit")
 clean_up
 break
 ;;
 "status")
 dump_status
 ;;
 "enable")
 enable_routing
 ;;
 "disable")
 disable_routing
 ;;
 *)
 echo "Available commands: quit, status, enable, disable"
 ;;
 esac
 done
}

function print_config() {
 echo "=== Input configuration ==="
 env | grep APPMESH_ || true
}
```

```
}

print_config

initialize

if ["$APPMESH_START_ENABLED" == "1"]; then
 enable_routing
fi

main_loop
```

9. 애플리케이션 트래픽을 Envoy 프록시로 라우팅하는 iptables 규칙을 구성하려면 이전 단계에서 생성한 스크립트를 실행합니다.

```
sudo ./envoy-networking.sh
```

10. 가상 노드 애플리케이션 코드를 시작합니다.

#### Note

App Mesh에 대한 더 많은 예제와 연습 내용을 보려면 [App Mesh 예제 리포지토리](#)를 참조하세요.

## App Mesh 예제

다음 리포지토리에서 다양한 AWS 서비스와 통합하기 위한 작업 중인 AWS App Mesh 및 코드 예제를 보여 주는 엔드투엔드 연습을 확인할 수 있습니다.

[App Mesh 예제](#)

# App Mesh 개념

App Mesh는 다음 개념으로 구성됩니다.

- [서비스 메시](#)
- [가상 서비스](#)
- [가상 게이트웨이](#)
- [가상 노드](#)
- [가상 라우터](#)

## 서비스 메시

서비스 메시는 내부에 있는 서비스 간의 네트워크 트래픽에 대한 논리적 경계입니다. 서비스 메시지를 생성한 후에는 메시의 애플리케이션 간에 트래픽을 분산하는 가상 서비스, 가상 노드, 가상 라우터 및 라우팅을 생성할 수 있습니다.

## 서비스 메시 생성

### Note

메시를 생성할 때 네임스페이스 선택기를 추가해야 합니다. 네임스페이스 선택기가 비어 있는 경우 모든 네임스페이스를 선택합니다. 네임스페이스를 제한하려면 레이블을 사용하여 App Mesh 리소스를 생성된 메시에 연결합니다.

## AWS Management Console

AWS Management Console을 사용하여 서비스 메시지를 생성하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 메시 생성을 선택합니다.
3. 메시 이름에서 서비스 메시의 이름을 지정합니다.
4. (선택 사항) 외부 트래픽 허용을 선택합니다. 기본적으로 메시의 프록시는 서로 간의 트래픽만 전달합니다. 외부 트래픽을 허용하는 경우 메시의 프록시는 메시에 정의된 프록시와 함께 배포되지 않은 서비스에 TCP 트래픽을 직접 전달하기도 합니다.

**Note**

ALLOW\_ALL을 사용할 때 가상 노드에 백엔드를 지정하는 경우 해당 가상 노드에 대한 모든 송신을 백엔드로 지정해야 합니다. 그렇지 않으면 ALLOW\_ALL이 해당 가상 노드에서는 더 이상 작동하지 않습니다.

**5. IP 버전 기본 설정**

기본 IP 버전 동작 재정의를 켜서 메시 내 트래픽에 사용해야 하는 IP 버전을 제어할 수 있습니다. 기본적으로 App Mesh는 다양한 IP 버전을 사용합니다.

**Note**

메시는 메시 내의 모든 가상 노드와 가상 게이트웨이에 IP 기본 설정을 적용합니다. 노드를 생성하거나 편집할 때 IP 기본 설정을 지정하여 개별 가상 노드에서 이 동작을 재정의할 수 있습니다. IPv4 및 IPv6 트래픽을 모두 수신할 수 있도록 하는 가상 게이트웨이의 구성은 메시에 설정된 기본 설정에 관계없이 동일하기 때문에 가상 게이트웨이에서는 IP 기본 설정을 재정의할 수 없습니다.

- 기본값
  - Envoy의 DNS 확인자는 IPv6를 선호하고 IPv4로 대체합니다.
  - 가능한 경우 AWS Cloud Map에서 반환한 IPv4 주소를 사용하고 대신 IPv6 주소를 사용합니다.
  - 로컬 애플용으로 생성된 엔드포인트는 IPv4 주소를 사용합니다.
  - Envoy 리스너는 모든 IPv4 주소에 바인딩됩니다.
- IPv6 선호
  - Envoy의 DNS 확인자는 IPv6를 선호하고 IPv4로 대체합니다.
  - 가능한 경우 AWS Cloud Map에서 반환한 IPv6 주소가 사용되고 IPv4 주소를 사용하도록 대체합니다.
  - 로컬 애플용으로 생성된 엔드포인트는 IPv6 주소를 사용합니다.
  - Envoy 리스너는 모든 IPv4 및 IPv6 주소에 바인딩됩니다.
- IPv4 선호
  - Envoy의 DNS 확인자는 IPv4를 선호하고 IPv6로 대체합니다.

- 가능한 경우 AWS Cloud Map에서 반환한 IPv4 주소를 사용하고 대신 IPv6 주소를 사용합니다.
  - 로컬 애플용으로 생성된 엔드포인트는 IPv4 주소를 사용합니다.
  - Envoy 리스너는 모든 IPv4 및 IPv6 주소에 바인딩됩니다.
- IPv6 전용
    - Envoy의 DNS 확인자는 IPv6만 사용합니다.
    - AWS Cloud Map에서 반환한 IPv6 주소만 사용됩니다. AWS Cloud Map가 IPv4주소를 반환하는 경우 IP 주소는 사용되지 않으며 빈 결과가 Envoy에 반환됩니다.
    - 로컬 애플용으로 생성된 엔드포인트는 IPv6 주소를 사용합니다.
    - Envoy 리스너는 모든 IPv4 및 IPv6 주소에 바인딩됩니다.
  - IPv4 전용
    - Envoy의 DNS 확인자는 IPv4만 사용합니다.
    - AWS Cloud Map에서 반환한 IPv4 주소만 사용됩니다. AWS Cloud Map가 IPv6주소를 반환하는 경우 IP 주소는 사용되지 않으며 빈 결과가 Envoy에 반환됩니다.
    - 로컬 애플용으로 생성된 엔드포인트는 IPv4 주소를 사용합니다.
    - Envoy 리스너는 모든 IPv4 및 IPv6 주소에 바인딩됩니다.
6. 메시 생성을 선택하여 완료합니다.
  7. (선택 사항) 메시지를 다른 계정과 공유합니다. 공유 메시지를 사용하면 다른 계정에서 생성한 리소스가 동일한 메시지에서 서로 통신할 수 있습니다. 자세한 내용은 [공유 메시지 작업](#) 섹션을 참조하세요.

## AWS CLI

AWS CLI를 사용하여 메시지를 생성하려면

다음 명령을 사용하여 서비스 메시지를 생성합니다(### 값을 원하는 값으로 대체).

1. 

```
aws appmesh create-mesh --mesh-name meshName
```

2. 출력 예시:

```
{
 "mesh": {
 "meshName": "meshName",
 "metadata": {
```

```

 "arn": "arn:aws:appmesh:us-west-2:123456789012:mesh/meshName",
 "createdAt": "2022-04-06T08:45:50.072000-05:00",
 "lastUpdatedAt": "2022-04-06T08:45:50.072000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "123456789012",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 1
 },
 "spec": {},
 "status": {
 "status": "ACTIVE"
 }
}
}

```

App Mesh용 AWS CLI를 사용하여 메시를 생성하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [create-mesh](#) 명령을 참조하세요.

## 메시 삭제

### AWS Management Console

AWS Management Console을 사용하여 가상 게이트웨이를 삭제하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 삭제하려는 메시를 선택합니다. 소유하고 있는 모든 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 확인 상자에 **delete**를 입력한 다음, 삭제를 클릭합니다.

### AWS CLI

AWS CLI를 사용하여 메시를 삭제하려면

1. 다음 명령을 사용하여 메시를 삭제합니다(### 값을 원하는 값으로 대체).

```
aws appmesh delete-mesh \
 --mesh-name meshName
```

2. 출력 예시:

```
{
 "mesh": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:123456789012:mesh/meshName",
 "createdAt": "2022-04-06T08:45:50.072000-05:00",
 "lastUpdatedAt": "2022-04-07T11:06:32.795000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "123456789012",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 1
 },
 "spec": {},
 "status": {
 "status": "DELETED"
 }
 }
}
```

App Mesh용 AWS CLI를 사용하여 메시를 삭제하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [delete-mesh](#) 명령을 참조하세요.

## 가상 서비스

가상 서비스는 가상 노드가 가상 라우터를 통해 직접 또는 간접적으로 제공하는 실제 서비스의 추상화입니다. 종속 서비스는 `virtualServiceName`으로 가상 서비스를 호출하고, 이러한 요청은 가상 서비스의 공급자로 지정된 가상 노드 또는 가상 라우터로 라우팅됩니다.

## 가상 서비스 생성

### AWS Management Console

AWS Management Console을 사용하여 가상 서비스를 생성하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 가상 서비스를 생성하려는 메시를 선택합니다. 소유하고 있는 모든 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 Virtual services(가상 서비스)를 선택합니다.

4. Create virtual service(가상 서비스 생성)를 선택합니다.
5. Virtual service name(가상 서비스 이름)에서 가상 서비스의 이름을 선택합니다. 원하는 이름을 선택할 수 있지만 가상 서비스를 실제 서비스와 쉽게 연관시킬 수 있도록 대상으로 하는 실제 서비스의 서비스 검색 이름(예: my-service.default.svc.cluster.local)을 사용하는 것이 좋습니다. 이렇게 하면 코드에서 현재 참조하는 이름과 다른 이름을 참조하도록 코드를 변경할 필요가 없습니다. 요청이 Envoy 프록시로 전송되기 전에 앱 컨테이너가 이름을 성공적으로 확인할 수 있어야 하므로 지정하는 이름은 루프백이 아닌 IP 주소로 확인되어야 합니다. 앱이나 프록시 컨테이너 모두 이 IP 주소와 통신하지 않으므로 루프백이 아닌 모든 IP 주소를 사용할 수 있습니다. 프록시는 이름이 확인되는 IP 주소를 통하지 않고 App Mesh에서 구성한 이름을 통해 다른 가상 서비스와 통신합니다.
6. Provider(공급자)에서 가상 서비스의 공급자 유형을 선택합니다.
  - 가상 서비스에서 여러 가상 노드로 트래픽을 분산시키려면 Virtual router(가상 라우터)를 선택한 다음, 드롭다운 메뉴에서 사용할 가상 라우터를 선택합니다.
  - 가상 라우터 없이 가상 서비스를 직접 가상 노드에 연결하려면 가상 노드를 선택한 다음, 드롭다운 메뉴에서 사용할 가상 노드를 선택합니다.

 Note

App Mesh는 2020년 7월 29일 또는 그 이후에 정의한 각 가상 노드 공급자에 대해 기본 Envoy 경로 재시도 정책을 자동으로 생성할 수 있습니다. App Mesh API를 통해 이러한 정책을 정의할 수 없더라도 마찬가지입니다. 자세한 내용은 [기본 경로 재시도 정책](#) 섹션을 참조하세요.

- 가상 서비스가 지금 트래픽을 라우팅하지 않게 하려면(예: 가상 노드 또는 가상 라우터가 아직 없는 경우) None(없음)을 선택합니다. 나중에 이 가상 서비스의 공급자를 업데이트할 수 있습니다.

7. Create virtual service(가상 서비스 생성)를 선택하여 완료합니다.

## AWS CLI

AWS CLI를 사용하여 가상 서비스를 생성하려면

다음 명령과 입력 JSON 파일을 사용하여 가상 노드 공급자로 가상 서비스를 생성합니다(### 값을 원하는 값으로 대체).

1. 

```
aws appmesh create-virtual-service \
```

```
--cli-input-json file://create-virtual-service-virtual-node.json
```

2. 예제 create-virtual-service-virtual-node.json의 내용은 다음과 같습니다.

```
{
 "meshName": "meshName",
 "spec": {
 "provider": {
 "virtualNode": {
 "virtualNodeName": "nodeName"
 }
 }
 },
 "virtualServiceName": "serviceA.svc.cluster.local"
}
```

3. 출력 예시:

```
{
 "virtualService": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/virtualService/serviceA.svc.cluster.local",
 "createdAt": "2022-04-06T09:45:35.890000-05:00",
 "lastUpdatedAt": "2022-04-06T09:45:35.890000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 1
 },
 "spec": {
 "provider": {
 "virtualNode": {
 "virtualNodeName": "nodeName"
 }
 }
 },
 "status": {
 "status": "ACTIVE"
 }
 },
 "virtualServiceName": "serviceA.svc.cluster.local"
}
```

}

App Mesh용 AWS CLI를 사용하여 가상 서비스를 생성하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [create-virtual-service](#) 명령을 참조하세요.

## 가상 서비스 삭제

### Note

게이트웨이 경로에서 참조하는 가상 서비스는 삭제할 수 없습니다. 먼저 게이트웨이 경로를 삭제해야 합니다.

### AWS Management Console

AWS Management Console를 사용하여 가상 서비스를 삭제하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 가상 서비스를 삭제하려는 메시지를 선택합니다. 소유하고 있는 모든 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 Virtual services(가상 서비스)를 선택합니다.
4. 삭제하려는 가상 서비스를 선택하고 오른쪽 상단 구석에서 삭제를 클릭합니다. 계정이 리소스 소유자로 나열된 경우에만 가상 게이트웨이를 삭제할 수 있습니다.
5. 확인 상자에 **delete**를 입력한 다음, 삭제를 클릭합니다.

### AWS CLI

AWS CLI를 사용하여 가상 서비스를 삭제하려면

1. 다음 명령을 사용하여 가상 서비스를 삭제합니다(**###** 값을 원하는 값으로 대체).

```
aws appmesh delete-virtual-service \
 --mesh-name meshName \
 --virtual-service-name serviceA.svc.cluster.local
```

2. 출력 예시:

```
{
 "virtualService": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/virtualService/serviceA.svc.cluster.local",
 "createdAt": "2022-04-06T09:45:35.890000-05:00",
 "lastUpdatedAt": "2022-04-07T10:39:42.772000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 2
 },
 "spec": {
 "provider": {
 "virtualNode": {
 "virtualNodeName": "nodeName"
 }
 }
 },
 "status": {
 "status": "DELETED"
 },
 "virtualServiceName": "serviceA.svc.cluster.local"
 }
}
```

App Mesh용 AWS CLI를 사용하여 가상 서비스를 삭제하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [delete-virtual-service](#) 명령을 참조하세요.

## 가상 게이트웨이

가상 게이트웨이를 사용하면 메시 외부의 리소스가 메시 내부에 있는 리소스와 통신할 수 있습니다. 가상 게이트웨이는 Amazon ECS 서비스, Kubernetes 서비스 또는 Amazon EC2 인스턴스에서 실행되는 Envoy 프록시를 나타냅니다. 애플리케이션과 함께 실행되는 Envoy를 나타내는 가상 노드와 달리 가상 게이트웨이는 자체적으로 배포된 Envoy를 나타냅니다.

외부 리소스는 DNS 이름을 Envoy를 실행하는 서비스 또는 인스턴스에 할당된 IP 주소로 확인할 수 있어야 합니다. 그러면 Envoy는 메시 내에 있는 리소스의 모든 App Mesh 구성에 액세스할 수 있습니다.

가상 게이트웨이에서 들어오는 요청을 처리하기 위한 구성은 [게이트웨이 경로](#)를 사용하여 지정됩니다.

### ⚠ Important

HTTP 또는 HTTP2 리스너가 있는 가상 게이트웨이는 들어오는 요청의 호스트 이름을 게이트웨이 경로 대상 가상 서비스의 이름에 다시 쓰고, 게이트웨이 경로의 일치하는 접두사가 기본적으로 /로 다시 작성됩니다. 예를 들어 게이트웨이 경로 일치 접두사를 /chapter로 구성한 경우, 수신 요청이 /chapter/1인 경우 요청이 /1로 다시 작성됩니다. 재작성을 구성하려면 게이트웨이 경로의 [게이트웨이 경로 생성](#) 섹션을 참조하세요.

가상 게이트웨이를 생성할 때는 proxyConfiguration 및 user를 구성하지 않아야 합니다.

전체적인 과정 연습을 완료하려면 [인바운드 게이트웨이 구성](#)을 참조하세요.

## 가상 게이트웨이 생성

### ℹ Note

가상 게이트웨이를 생성할 때는 게이트웨이 경로를 생성된 가상 게이트웨이에 연결할 네임스페이스 목록을 식별할 수 있도록 레이블이 있는 네임스페이스 선택기를 추가해야 합니다.

## AWS Management Console

AWS Management Console을 사용하여 가상 게이트웨이를 생성하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 가상 게이트웨이를 생성하려는 메시지를 선택합니다. 소유하고 있는 모든 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 가상 게이트웨이를 선택합니다.
4. 가상 게이트웨이 생성을 선택합니다.
5. 가상 게이트웨이 이름에 가상 게이트웨이의 이름을 입력합니다.
6. (선택 사항이지만 권장됨) 클라이언트 정책 기본값을 구성합니다.
  - a. (선택 사항) 게이트웨이가 전송 계층 보안(TLS)을 사용하여 가상 서비스와만 통신하도록 하려면 TLS 적용을 선택합니다.

- b. (선택 사항) 포트에는 가상 서비스와의 TLS 통신을 적용하려는 하나 이상의 포트를 지정합니다.
  - c. 검증 방법에서 다음 옵션 중 하나를 선택합니다. 지정하는 인증서는 이미 존재하고 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [인증서 요구 사항](#) 섹션을 참조하세요.
    - AWS Private Certificate Authority 호스팅 - 기존 인증서를 하나 이상 선택합니다.
    - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
    - 로컬 파일 호스팅 - Envoy가 배포되는 파일 시스템의 인증서 체인 파일 경로를 지정합니다.
  - d. (선택 사항) 주체 대체 이름을 입력합니다. SAN을 더 추가하려면 SAN 추가를 선택합니다. SAN은 FQDN 또는 URI 형식이어야 합니다.
  - e. (선택 사항) 클라이언트 인증서 제공과 아래 옵션 중 하나를 선택하여 서버가 요청할 때 클라이언트 인증서를 제공하고 상호 TLS 인증을 활성화합니다. 상호 TLS에 대해 자세히 알아보려면 App Mesh [상호 TLS 인증](#) 설명서를 참조하세요.
    - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
    - 로컬 파일 호스팅 - Envoy가 배포되는 파일 시스템의 프라이빗 키 뿐만 아니라 인증서 체인 파일의 경로를 지정합니다. 로컬 파일에 암호화를 사용하여 샘플 애플리케이션으로 메시지를 배포하는 과정을 전체적으로 자세히 알아보려면 GitHub에서 [파일 제공 TLS 인증서를 사용하여 TLS 구성](#)을 참조하세요.
7. (선택 사항) 로깅을 구성하려면 로깅을 선택합니다. Envoy에서 사용할 HTTP 액세스 로그 경로를 입력합니다. Docker 로그 드라이버를 사용하여 Envoy 로그를 Amazon CloudWatch Logs와 같은 서비스로 내보낼 수 있도록 `/dev/stdout` 경로를 사용하는 것이 좋습니다.

 Note

로그는 계속 애플리케이션의 에이전트에 의해 수집되어 대상으로 전송되어야 합니다. 이 파일 경로는 로그를 보낼 위치만 Envoy에 지시합니다.

8. 리스너를 구성합니다.
- a. 프로토콜을 선택하고 Envoy가 트래픽을 수신하는 포트를 지정합니다. http 리스너는 웹 소켓으로의 연결 전환을 허용합니다. 리스너 추가를 클릭하여 리스너를 여러 개 추가할 수 있습니다. 제거 버튼을 누르면 해당 리스너가 제거됩니다.

## b. (선택 사항) 연결 풀 활성화

연결 풀링은 가상 게이트웨이 Envoy가 동시에 설정할 수 있는 연결 수를 제한합니다. 이 작업은 Envoy 인스턴스의 연결 과부하를 방지하고 애플리케이션 요구 사항에 맞게 트래픽 셰이핑을 조정할 수 있도록 하기 위한 것입니다.

가상 게이트웨이 리스너의 대상 측 연결 풀 설정을 구성할 수 있습니다. App Mesh는 기본적으로 클라이언트 측 연결 풀 설정을 무제한으로 설정하여 메시 구성을 단순화합니다.

### Note

`connectionPool` 및 `connectionPool portMapping` 프로토콜은 동일해야 합니다. 리스너 프로토콜이 `grpc` 또는 `http2`인 경우 `maxRequests`만 지정합니다. 리스너 프로토콜이 `http`인 경우 `maxConnections` 및 `maxPendingRequests`를 모두 지정할 수 있습니다.

- 최대 연결 수에는 최대 아웃바운드 연결 수를 지정합니다.
- 최대 요청 수에는 가상 게이트웨이 Envoy에 대해 설정할 수 있는 최대 병렬 요청 수를 지정합니다.
- (선택 사항) 대기 중인 최대 요청 수에는 Envoy가 대기하는 최대 연결 수 이후에 오버플로되는 요청 수를 지정합니다. 기본값은 2147483647입니다.

## c. (선택 사항) 리스너에 대한 상태 확인을 구성하려면 상태 확인 사용을 선택합니다.

상태 확인 정책은 선택 사항이지만 상태 정책에 대한 값을 지정한 경우 정상 임계 값, 상태 확인 간격, 상태 확인 프로토콜, 제한 시간 및 비정상 임계값에 대해 값을 지정해야 합니다.

- 상태 확인 프로토콜에 대해 프로토콜을 선택합니다. `grpc`를 선택한 경우 서비스는 [GRPC 상태 확인 프로토콜](#)을 준수해야 합니다.
- Health check port(상태 확인 포트)에서 상태 확인을 실행해야 할 포트를 지정합니다.
- Healthy threshold(정상 임계 값)에서 리스너를 정상으로 선언하기 전까지 발생해야 하는 연속적인 상태 확인 성공 횟수를 지정합니다.
- Health check interval(상태 확인 간격)에서 각 상태 확인 실행 사이의 시간 간격을 밀리초 단위로 지정합니다.

- Path(경로)에서 상태 확인 요청에 대한 대상 경로를 지정합니다. 이 값은 상태 확인 프로토콜이 http 또는 http2인 경우에만 사용됩니다. 다른 프로토콜에서는 이 값이 무시됩니다.
  - 제한 시간에서 상태 확인으로부터 응답을 받을 때까지 대기할 시간을 밀리초 단위로 지정합니다.
  - 비정상 임계값에서 리스너를 비정상적으로 선언하기 전까지 발생해야 하는 연속적인 상태 확인 실패 횟수를 지정합니다.
- d. (선택 사항) 클라이언트가 TLS를 사용하여 이 가상 게이트웨이와 통신할지 여부를 지정하려면 TLS 종료 활성화를 선택합니다.
- 모드의 경우 리스너에서 TLS를 구성할 모드를 선택합니다.
  - 인증서 방법에서 다음 옵션 중 하나를 수행합니다. 인증서는 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [인증서 요구 사항](#) 섹션을 참조하세요.
    - AWS Certificate Manager 호스팅 - 기존 인증서를 선택합니다.
    - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
    - 로컬 파일 호스팅 - Envoy가 배포된 파일 시스템의 인증서 체인 및 프라이빗 키 파일 경로를 지정합니다.
  - (선택 사항) 클라이언트가 인증서를 제공하는 경우 클라이언트 인증서 필요 및 아래 옵션 중 하나를 선택하여 상호 TLS 인증을 활성화합니다. 상호 TLS에 대해 자세히 알아보려면 App Mesh [상호 TLS 인증](#) 설명서를 참조하세요.
    - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
    - 로컬 파일 호스팅 - Envoy가 배포되는 파일 시스템의 인증서 체인 파일 경로를 지정합니다.
  - (선택 사항) 주체 대체 이름을 입력합니다. SAN을 더 추가하려면 SAN 추가를 선택합니다. SAN은 FQDN 또는 URI 형식이어야 합니다.
9. 가상 게이트웨이 생성을 선택하여 완료합니다.

## AWS CLI

AWS CLI를 사용하여 가상 게이트웨이를 생성하려면

다음 명령을 사용하여 가상 게이트웨이를 생성하고 JSON을 입력합니다(### 값을 원하는 값으로 대체).

1. 

```
aws appmesh create-virtual-gateway \
--mesh-name meshName \
--virtual-gateway-name virtualGatewayName \
--cli-input-json file://create-virtual-gateway.json
```
2. 예제 create-virtual-gateway.json의 내용은 다음과 같습니다.

```
{
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 9080,
 "protocol": "http"
 }
 }
]
 }
}
```

3. 출력 예시:

```
{
 "virtualGateway": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:123456789012:mesh/meshName/virtualGateway/virtualGatewayName",
 "createdAt": "2022-04-06T10:42:42.015000-05:00",
 "lastUpdatedAt": "2022-04-06T10:42:42.015000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "123456789012",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 1
 },
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 9080,
 "protocol": "http"
 }
 }
]
 }
 }
}
```

```

]
 },
 "status": {
 "status": "ACTIVE"
 },
 "virtualGatewayName": "virtualGatewayName"
}
}

```

App Mesh용 AWS CLI를 사용하여 가상 게이트웨이를 생성하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [create-virtual-gateway](#) 명령을 참조하세요.

## 가상 게이트웨이 배포

[Envoy 컨테이너](#)만 포함하는 Amazon ECS 또는 Kubernetes 서비스를 배포합니다. 또한 Amazon EC2 인스턴스의 Envoy 컨테이너를 배포할 수도 있습니다. 자세한 내용은 [App Mesh 및 Amazon EC2 시작하기](#)를 참조하세요. [App Mesh 및 Amazon ECS 시작하기](#) 또는 [AWS App Mesh 및 Kubernetes 시작하기](#)에서 Amazon ECS에 배포하는 방법에 대한 자세한 내용을 참조하여 Kubernetes에 배포하세요. 프록시의 트래픽이 자체적으로 리디렉션되지 않도록 APPMESH\_RESOURCE\_ARN 환경 변수를 `mesh/mesh-name/virtualGateway/virtual-gateway-name`로 설정하고 프록시 구성을 지정하지 않아야 합니다. 기본적으로 App Mesh는 Envoy가 지표 및 트레이스에서 자신을 참조할 때 APPMESH\_RESOURCE\_ARN에서 지정한 리소스의 이름을 사용합니다. APPMESH\_RESOURCE\_CLUSTER 환경 변수를 사용자 고유의 이름으로 설정하여 이 동작을 재정의할 수 있습니다.

컨테이너의 여러 인스턴스를 배포하고 인스턴스에 대한 트래픽을 로드 밸런싱하도록 Network Load Balancer를 설정하는 것이 좋습니다. 로드 밸런서의 서비스 검색 이름은 외부 서비스가 메시에 있는 리소스에 액세스하는 데 사용하려는 이름(예: `myapp.example.com`)입니다. 자세한 내용은 [Network Load Balancer 생성](#)(Amazon ECS), [외부 로드 밸런서 생성](#)(Kubernetes) 또는 [자습서: Amazon EC2에서 애플리케이션의 가용성 향상](#)을 참조하세요. 또한 [App Mesh 예제](#)에서 더 많은 예제와 연습을 찾을 수 있습니다.

Envoy에 대한 프록시 권한 부여를 활성화합니다. 자세한 내용은 [Envoy 프록시 권한 부여](#) 섹션을 참조하세요.

# 가상 게이트웨이 삭제

## AWS Management Console

AWS Management Console을 사용하여 가상 게이트웨이를 삭제하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 가상 게이트웨이를 삭제하려는 메시를 선택합니다. 소유하고 있는 모든 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 가상 게이트웨이를 선택합니다.
4. 삭제하려는 가상 게이트웨이를 선택하고 삭제를 선택합니다. 연결된 게이트웨이 경로가 있는 가상 게이트웨이는 삭제할 수 없습니다. 연결된 게이트웨이 경로를 먼저 삭제해야 합니다. 계정이 리소스 소유자로 나열된 경우에만 가상 게이트웨이를 삭제할 수 있습니다.
5. 확인 상자에 **delete**를 입력한 다음, 삭제를 선택합니다.

## AWS CLI

AWS CLI을 사용하여 가상 게이트웨이를 삭제하려면

1. 다음 명령을 사용하여 가상 게이트웨이를 삭제합니다(**###** 값을 원하는 값으로 대체).

```
aws appmesh delete-virtual-gateway \
 --mesh-name meshName \
 --virtual-gateway-name virtualGatewayName
```

2. 출력 예시:

```
{
 "virtualGateway": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:123456789012:mesh/meshName/virtualGateway/virtualGatewayName",
 "createdAt": "2022-04-06T10:42:42.015000-05:00",
 "lastUpdatedAt": "2022-04-07T10:57:22.638000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "123456789012",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 2
 }
 },
```

```

 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 9080,
 "protocol": "http"
 }
 }
]
 },
 "status": {
 "status": "DELETED"
 },
 "virtualGatewayName": "virtualGatewayName"
 }
}

```

App Mesh용 AWS CLI를 사용하여 가상 게이트웨이를 삭제하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [delete-virtual-gateway](#) 명령을 참조하세요.

## 게이트웨이 경로

게이트웨이 라우팅은 가상 게이트웨이에 연결되며 기존 가상 서비스로 트래픽을 라우팅합니다. 라우팅이 요청과 일치하는 경우 대상 가상 서비스에 트래픽을 분산할 수 있습니다. 이 주제는 서비스 메시의 게이트웨이 경로를 사용하는 데 도움이 됩니다.

## 게이트웨이 경로 생성

### AWS Management Console

AWS Management Console을 사용하여 게이트웨이 경로를 생성하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 게이트웨이 경로를 생성하려는 메시지를 선택합니다. 소유하고 있는 메시와 **공유된** 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 가상 게이트웨이를 선택합니다.
4. 새 게이트웨이 경로를 연결할 가상 게이트웨이를 선택합니다. 목록에 항목이 없는 경우 먼저 [가상 게이트웨이를 생성](#)해야 합니다. 사용자 계정이 리소스 소유자로 나열된 가상 게이트웨이에 대해서만 게이트웨이 경로를 생성할 수 있습니다.

5. 게이트웨이 경로 테이블에서 게이트웨이 경로 생성을 선택합니다.
6. 게이트웨이 경로 이름에서 게이트웨이 경로에 사용할 이름을 지정합니다.
7. 게이트웨이 경로 유형으로는 http, http2 또는 grpc를 선택합니다.
8. 기존 가상 서비스 이름을 선택합니다. 목록에 항목이 없는 경우 먼저 [가상 서비스](#)를 생성해야 합니다.
9. 가상 서비스 공급자 포트의 대상에 해당하는 포트를 선택합니다. 선택한 가상 서비스의 공급자 (라우터 또는 노드)에 리스너가 여러 개 있는 경우 가상 서비스 공급자 포트가 필수입니다.
10. (선택 사항) 우선순위에서 이 게이트웨이 경로의 우선순위를 지정합니다.
11. 일치 구성의 경우 다음을 지정합니다.
  - http/http2가 선택된 유형인 경우:
    - (선택 사항) 메서드 - 들어오는 http/http2 요청에서 일치시킬 메서드 헤더를 지정합니다.
    - (선택 사항) 포트 일치 - 들어오는 트래픽에 맞는 포트를 찾습니다. 이 가상 게이트웨이에 여러 리스너가 있는 경우 포트 일치가 필수입니다.
    - (선택 사항) 정확한/접미사 호스트 이름 - 수신 요청에서 대상 가상 서비스로 라우팅하기 위해 일치시켜야 하는 호스트 이름을 지정합니다.
    - (선택 사항) 접두사/정확한/정규식 경로 - URL 경로를 일치시키는 방법입니다.
    - 접두사 일치 - 기본적으로 게이트웨이 경로의 일치하는 요청이 대상 가상 서비스 이름에 다시 쓰여지고 일치하는 접두사가 다시 작성됩니다(기본값 /). 가상 서비스를 구성하는 방법에 따라 가상 라우터를 사용하여 특정 접두사 또는 헤더를 기준으로 요청을 다른 가상 노드로 라우팅할 수 있습니다.

#### Important

- 접두사 일치에 대해 /aws-appmesh\* 및 /aws-app-mesh\* 둘 다 지정할 수 없습니다. 이러한 접두사는 향후 App Mesh 내부 사용을 위해 예약되어 있습니다.
- 게이트웨이 경로를 여러 개 정의하면 요청은 접두사가 가장 긴 경로와 일치됩니다. 예를 들어 두 개의 게이트웨이 경로가 존재하는데 하나는 접두사가 /chapter이고 다른 하나는 접두사가 /인 경우 www.example.com/chapter/에 대한 요청은 접두사 /chapter인 게이트웨이 경로와 일치합니다.

**Note**

경로/접두사 기반 일치를 활성화하면 App Mesh는 경로 정규화([normalize\\_path](#) 및 [merge\\_slashes](#))를 활성화하여 경로 혼동 취약성이 발생할 가능성을 최소화합니다.

요청에 참여하는 당사자가 서로 다른 경로 표현을 사용하는 경우 경로 혼동 취약성이 발생합니다.

- 정확한 일치 - `exact` 파라미터는 경로의 부분 일치를 비활성화하고 경로가 현재 URL에 대해 EXACT 일치인 경우에만 경로를 반환하도록 합니다.
- 정규식 일치 - 여러 URL이 실제로 웹 사이트의 단일 페이지를 식별할 수 있는 패턴을 설명하는 데 사용됩니다.
- (선택 사항) 쿼리 파라미터 - 이 필드를 사용하면 쿼리 파라미터를 일치시킬 수 있습니다.
- (선택 사항) 헤더 - `http` 및 `http2`에 대한 헤더를 지정합니다. `요청` 가상 서비스로 라우팅하려면 들어오는 요청과 일치해야 합니다.
- `grpc`가 선택된 유형인 경우:
  - 호스트 이름 일치 유형 및 (선택 사항) 정확한/접미사 일치 - 수신 요청에서 대상 가상 서비스로 라우팅하기 위해 일치시켜야 하는 호스트 이름을 지정합니다.
  - `grpc` 서비스 이름 - `grpc` 서비스는 애플리케이션의 API 역할을 하며 `ProtoBuf`로 정의됩니다.

**Important**

서비스 이름에는 `/aws.app-mesh*` 또는 `aws.appmesh`를 지정할 수 없습니다. 이러한 서비스 이름은 향후 App Mesh 내부 사용을 위해 예약되어 있습니다.

- (선택 사항) 메타데이터 - `grpc`의 메타데이터를 지정합니다. 대상 가상 서비스로 라우팅하려면 들어오는 요청과 일치해야 합니다.

## 12. (선택 사항) 다시 작성 구성의 경우:

- `http/http2`가 선택된 유형인 경우:
  - 접두사가 선택된 일치 유형인 경우:
    - 호스트 이름 자동 다시 작성 재정의 - 기본적으로 호스트 이름은 대상 가상 서비스의 이름으로 다시 작성됩니다.

- 접두사 자동 다시 작성 재정의 - 켜져 있는 경우 접두사 다시 작성이 다시 작성된 접두사 값을 지정합니다.
- 정확한 경로가 선택된 일치 유형인 경우:
  - 호스트 이름 자동 다시 작성 재정의 - 기본적으로 호스트 이름은 대상 가상 서비스의 이름으로 다시 작성됩니다.
  - 경로 다시 작성 - 다시 작성된 경로의 값을 지정합니다. 기본 경로는 없습니다.
- 접두사 경로가 선택된 일치 유형인 경우:
  - 호스트 이름 자동 다시 작성 재정의 - 기본적으로 호스트 이름은 대상 가상 서비스의 이름으로 다시 작성됩니다.
  - 경로 다시 작성 - 다시 작성된 경로의 값을 지정합니다. 기본 경로는 없습니다.
- grpc가 선택된 유형인 경우:
  - 호스트 이름 자동 다시 작성 재정의 - 기본적으로 호스트 이름은 대상 가상 서비스의 이름으로 다시 작성됩니다.

13. 게이트웨이 경로 생성을 선택하여 완료합니다.

## AWS CLI

AWS CLI를 사용하여 게이트웨이 경로를 생성하려면

다음 명령을 사용하여 게이트웨이 경로를 생성하고 JSON을 입력합니다(### 값을 원하는 값으로 대체).

1.

```
aws appmesh create-virtual-gateway \
--mesh-name meshName \
--virtual-gateway-name virtualGatewayName \
--gateway-route-name gatewayRouteName \
--cli-input-json file://create-gateway-route.json
```

2. 예제 create-gateway-route.json의 내용은 다음과 같습니다.

```
{
 "spec": {
 "httpRoute" : {
 "match" : {
 "prefix" : "/"
 },
 "action" : {
```

```

 "target" : {
 "virtualService": {
 "virtualServiceName": "serviceA.svc.cluster.local"
 }
 }
 }
}

```

### 3. 출력 예시:

```

{
 "gatewayRoute": {
 "gatewayRouteName": "gatewayRouteName",
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/virtualGateway/virtualGatewayName/gatewayRoute/gatewayRouteName",
 "createdAt": "2022-04-06T11:05:32.100000-05:00",
 "lastUpdatedAt": "2022-04-06T11:05:32.100000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 1
 },
 "spec": {
 "httpRoute": {
 "action": {
 "target": {
 "virtualService": {
 "virtualServiceName": "serviceA.svc.cluster.local"
 }
 }
 },
 "match": {
 "prefix": "/"
 }
 }
 },
 "status": {
 "status": "ACTIVE"
 }
 },
 "virtualGatewayName": "gatewayName"
}

```

```
}
}
```

App Mesh용 AWS CLI를 사용하여 게이트웨이 경로를 생성하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [create-gateway-route](#) 명령을 참조하세요.

## 게이트웨이 경로 삭제

### AWS Management Console

AWS Management Console를 사용하여 게이트웨이 경로를 삭제하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 게이트웨이 경로를 삭제하려는 메시지를 선택합니다. 소유하고 있는 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 가상 게이트웨이를 선택합니다.
4. 게이트웨이 경로를 삭제하려는 가상 게이트웨이를 선택합니다.
5. 게이트웨이 경로 테이블에서 삭제하려는 게이트웨이 경로를 선택하고 삭제를 선택합니다. 계정이 리소스 소유자로 나열된 경우에만 게이트웨이 경로를 삭제할 수 있습니다.
6. 확인 상자에 **delete**를 입력한 다음, 삭제를 클릭합니다.

### AWS CLI

AWS CLI를 사용하여 게이트웨이 경로를 삭제하려면

1. 다음 명령을 사용하여 게이트웨이 경로를 삭제합니다(**###** 값을 원하는 값으로 대체).

```
aws appmesh delete-gateway-route \
 --mesh-name meshName \
 --virtual-gateway-name virtualGatewayName \
 --gateway-route-name gatewayRouteName
```

2. 출력 예시:

```
{
 "gatewayRoute": {
 "gatewayRouteName": "gatewayRouteName",
```

```

 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualGateway/virtualGatewayName/gatewayRoute/gatewayRouteName",
 "createdAt": "2022-04-06T11:05:32.100000-05:00",
 "lastUpdatedAt": "2022-04-07T10:36:33.191000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 2
 },
 "spec": {
 "httpRoute": {
 "action": {
 "target": {
 "virtualService": {
 "virtualServiceName": "serviceA.svc.cluster.local"
 }
 }
 },
 "match": {
 "prefix": "/"
 }
 }
 },
 "status": {
 "status": "DELETED"
 },
 "virtualGatewayName": "virtualGatewayName"
 }
}

```

App Mesh용 AWS CLI를 사용하여 게이트웨이 경로를 삭제하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [delete-gateway-route](#) 명령을 참조하세요.

## 가상 노드

가상 노드는 Amazon ECS 서비스 또는 Kubernetes 배포와 같은 특정 작업 그룹에 대한 논리 포인터 역할을 합니다. 가상 노드를 생성할 때 작업 그룹에 대한 서비스 검색 방법을 지정해야 합니다. 가상 노드

에서 예상되는 인바운드 트래픽은 리스너로 지정해야 합니다. 가상 노드에서 아웃바운드 트래픽을 보내는 모든 가상 서비스는 백엔드로 지정됩니다.

새로운 가상 노드에 대한 응답 메타 데이터에는 가상 노드와 연결된 Amazon 리소스 이름(ARN)이 들어 있습니다. Amazon ECS 태스크 정의 또는 Kubernetes 포드 사양에서 이 값을 태스크 그룹의 Envoy 프록시 컨테이너에 대한 APPMESH\_RESOURCE\_ARN 환경 변수로 설정합니다. 예를 들어 값이 `arn:aws:appmesh:us-west-2:111122223333:mesh/myMesh/virtualNode/myVirtualNode`일 수 있습니다. 그러면 `node.id` 및 `node.cluster` Envoy 파라미터로 매핑됩니다. 이 변수를 설정할 때는 Envoy 이미지 1.15.0 이상을 사용하고 있어야 합니다. App Mesh Envoy 변수에 대한 자세한 내용은 [Envoy](#) 섹션을 참조하세요.

### Note

기본적으로 App Mesh는 Envoy가 지표 및 트레이스에서 자신을 참조할 때 APPMESH\_RESOURCE\_ARN에서 지정한 리소스의 이름을 사용합니다. APPMESH\_RESOURCE\_CLUSTER 환경 변수를 사용자 고유의 이름으로 설정하여 이 동작을 재정의할 수 있습니다.

## 가상 노드 생성

### AWS Management Console

AWS Management Console을 사용하여 가상 노드를 생성하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 가상 노드를 생성하려는 메시지를 선택합니다. 소유하고 있는 모든 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 가상 노드를 선택합니다.
4. 가상 노드 생성을 선택한 다음, 가상 노드에 대한 설정을 지정합니다.
5. 가상 노드 이름에서 가상 노드의 이름을 입력합니다.
6. 서비스 검색 방법에서 다음 옵션 중 하나를 선택합니다.
  - DNS – 가상 노드가 나타내는 실제 서비스의 DNS 호스트 이름을 지정합니다. Envoy 프록시는 Amazon VPC에 배포됩니다. 프록시는 VPC용으로 구성된 DNS 서버에 이름 확인 요청을 보냅니다. 호스트 이름이 확인되면 DNS 서버는 하나 이상의 IP 주소를 반환합니다. VPC DNS 설정에 대한 자세한 내용은 [VPC와 함께 DNS 사용](#)을 참조하세요. DNS 응답 유형(선택

사항)의 경우 DNS 확인자가 반환하는 엔드포인트 유형을 지정합니다. 로드 밸런서는 DNS 확인자가 부하 분산 엔드포인트 세트를 반환하는 것을 의미합니다. 엔드포인트는 DNS 확인자가 모든 엔드포인트를 반환한다는 것을 의미합니다. 기본적으로 응답 유형은 로드 밸런서로 간주됩니다.

 Note

Route53을 사용하는 경우 로드 밸런서를 사용해야 합니다.

- AWS Cloud Map— 기존 서비스 이름과 HTTP 네임스페이스를 지정합니다. 선택적으로 행 추가를 선택하고 키와 값을 지정하여 App Mesh가 AWS Cloud Map를 쿼리할 수 있는 속성을 지정할 수도 있습니다. 지정된 모든 키/값 페어와 일치하는 인스턴스만 반환됩니다. AWS Cloud Map을 사용하려면 AWSServiceRoleForAppMesh [서비스 연결 역할](#)이 필요합니다. AWS Cloud Map에 대한 자세한 내용은 [AWS Cloud Map개발자 안내서](#) 단원을 참조하세요.
- 없음 – 가상 노드에서 수신 트래픽을 예상하지 않을 경우에 선택합니다.

## 7. IP 버전 기본 설정

기본 IP 버전 동작 재정의의 커서 메시 내 트래픽에 사용해야 하는 IP 버전을 제어할 수 있습니다. 기본적으로 App Mesh는 다양한 IP 버전을 사용합니다.

 Note

가상 노드에서 IP 기본 설정을 지정하면 이 특정 노드의 메시에 대해 설정된 IP 기본 설정만 재정의됩니다.

- 기본값
  - Envoy의 DNS 확인자는 IPv6를 선호하고 IPv4로 대체합니다.
  - 가능한 경우 AWS Cloud Map에서 반환한 IPv4 주소를 사용하고 대신 IPv6 주소를 사용합니다.
  - 로컬 애플용으로 생성된 엔드포인트는 IPv4 주소를 사용합니다.
  - Envoy 리스너는 모든 IPv4 주소에 바인딩됩니다.
- IPv6 선호
  - Envoy의 DNS 확인자는 IPv6를 선호하고 IPv4로 대체합니다.
  - 가능한 경우 AWS Cloud Map에서 반환한 IPv6 주소가 사용되고 IPv4 주소를 사용하도록 대체합니다.

- 로컬 애플리케이션으로 생성된 엔드포인트는 IPv6 주소를 사용합니다.
  - Envoy 리스너는 모든 IPv4 및 IPv6 주소에 바인딩됩니다.
- IPv4 선호
    - Envoy의 DNS 확인자는 IPv4를 선호하고 IPv6로 대체합니다.
    - 가능한 경우 AWS Cloud Map에서 반환한 IPv4 주소를 사용하고 대신 IPv6 주소를 사용합니다.
    - 로컬 애플리케이션으로 생성된 엔드포인트는 IPv4 주소를 사용합니다.
    - Envoy 리스너는 모든 IPv4 및 IPv6 주소에 바인딩됩니다.
  - IPv6 전용
    - Envoy의 DNS 확인자는 IPv6만 사용합니다.
    - AWS Cloud Map에서 반환한 IPv6 주소만 사용됩니다. AWS Cloud Map가 IPv4 주소를 반환하는 경우 IP 주소는 사용되지 않으며 빈 결과가 Envoy에 반환됩니다.
    - 로컬 애플리케이션으로 생성된 엔드포인트는 IPv6 주소를 사용합니다.
    - Envoy 리스너는 모든 IPv4 및 IPv6 주소에 바인딩됩니다.
  - IPv4 전용
    - Envoy의 DNS 확인자는 IPv4만 사용합니다.
    - AWS Cloud Map에서 반환한 IPv4 주소만 사용됩니다. AWS Cloud Map가 IPv6 주소를 반환하는 경우 IP 주소는 사용되지 않으며 빈 결과가 Envoy에 반환됩니다.
    - 로컬 애플리케이션으로 생성된 엔드포인트는 IPv4 주소를 사용합니다.
    - Envoy 리스너는 모든 IPv4 및 IPv6 주소에 바인딩됩니다.
8. (선택 사항) 클라이언트 정책 기본값 - 백엔드 가상 서비스와 통신할 때 기본 요구 사항을 구성합니다.

#### Note

- 기존 가상 노드에 대해 전송 계층 보안(TLS)을 활성화하려면 기존 가상 노드와 동일한 서비스를 나타내며 TLS를 활성화할 새 가상 노드를 생성하는 것이 좋습니다. 그런 다음, 가상 라우터와 경로를 사용하여 트래픽을 새 가상 노드로 점진적으로 이동합니다. 경로 생성 및 전환 가중치 조정에 대한 자세한 내용은 [경로](#) 섹션을 참조하세요. 트래픽을 처리하는 기존 가상 노드를 TLS로 업데이트하면 업데이트한 가상 노드의 Envoy 프록시가 인증서를 받기 전에 다운스트림 클라이언트 Envoy 프록시가 TLS 검증 컨텍스트를 수신할 수 있습니다. 이로 인해 다운스트림 Envoy 프록시에서 TLS 협상 오류가 발생할 수 있습니다.

- 백엔드 서비스의 가상 노드로 표시되는 애플리케이션과 함께 배포된 Envoy 프록시에 대해 [프록시 인증](#)을 활성화해야 합니다. 프록시 인증을 활성화할 때는 이 가상 노드가 통신하는 가상 노드로만 액세스를 제한하는 것이 좋습니다.

- (선택 사항) 가상 노드가 게이트웨이가 전송 계층 보안(TLS)을 사용하여 모든 백엔드와 통신하도록 하려면 TLS 적용을 선택합니다.
  - (선택 사항) 하나 이상의 특정 포트에만 TLS를 사용하도록 하려면 포트에 번호를 입력합니다. 포트를 추가하려면 포트 추가를 선택합니다. 포트를 지정하지 않으면 모든 포트에 TLS가 적용됩니다.
  - 검증 방법에서 다음 옵션 중 하나를 선택합니다. 지정하는 인증서는 이미 존재하고 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [인증서 요구 사항](#) 섹션을 참조하세요.
    - AWS Private Certificate Authority 호스팅 - 기존 인증서를 하나 이상 선택합니다. ACM 인증서에 암호화를 사용하여 샘플 애플리케이션으로 메시지를 배포하는 과정을 전체적으로 자세히 알아보려면 GitHub에서 [AWS Certificate Manager를 사용하여 TLS 구성](#)을 참조하세요.
    - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
    - 로컬 파일 호스팅 - Envoy가 배포되는 파일 시스템의 인증서 체인 파일 경로를 지정합니다. 로컬 파일에 암호화를 사용하여 샘플 애플리케이션으로 메시지를 배포하는 과정을 전체적으로 자세히 알아보려면 GitHub에서 [파일 제공 TLS 인증서를 사용하여 TLS 구성](#)을 참조하세요.
  - (선택 사항) 주체 대체 이름을 입력합니다. SAN을 더 추가하려면 SAN 추가를 선택합니다. SAN은 FQDN 또는 URI 형식이어야 합니다.
  - (선택 사항) 클라이언트 인증서 제공과 아래 옵션 중 하나를 선택하여 서버가 요청할 때 클라이언트 인증서를 제공하고 상호 TLS 인증을 활성화합니다. 상호 TLS에 대해 자세히 알아보려면 App Mesh [상호 TLS 인증](#) 설명서를 참조하세요.
    - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
    - 로컬 파일 호스팅 - Envoy가 배포되는 파일 시스템의 프라이빗 키 뿐만 아니라 인증서 체인 파일의 경로를 지정합니다.
9. (선택 사항) 서비스 백엔드 - 가상 노드가 통신할 App Mesh 가상 서비스를 지정합니다.
- 가상 노드가 통신하는 가상 서비스의 가상 서비스에 대해 App Mesh 가상 서비스 이름 또는 전체 Amazon 리소스 이름(ARN)을 입력합니다.

- (선택 사항) 백엔드에 대해 고유한 TLS 설정을 지정하려면 TLS 설정을 선택한 다음, 기본값 재정의를 선택합니다.
- (선택 사항) 가상 노드가 게이트웨이가 TLS를 사용하여 모든 백엔드와 통신하도록 하려면 TLS 적용을 선택합니다.
- (선택 사항) 하나 이상의 특정 포트에만 TLS를 사용하도록 하려면 포트에 번호를 입력합니다. 포트를 추가하려면 포트 추가를 선택합니다. 포트를 지정하지 않으면 모든 포트에 TLS가 적용됩니다.
- 검증 방법에서 다음 옵션 중 하나를 선택합니다. 지정하는 인증서는 이미 존재하고 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [인증서 요구 사항](#) 섹션을 참조하세요.
  - AWS Private Certificate Authority 호스팅 - 기존 인증서를 하나 이상 선택합니다.
  - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
  - 로컬 파일 호스팅 - Envoy가 배포되는 파일 시스템의 인증서 체인 파일 경로를 지정합니다.
- (선택 사항) 주체 대체 이름을 입력합니다. SAN을 더 추가하려면 SAN 추가를 선택합니다. SAN은 FQDN 또는 URI 형식이어야 합니다.
- (선택 사항) 클라이언트 인증서 제공과 아래 옵션 중 하나를 선택하여 서버가 요청할 때 클라이언트 인증서를 제공하고 상호 TLS 인증을 활성화합니다. 상호 TLS에 대해 자세히 알아보려면 App Mesh [상호 TLS 인증](#) 설명서를 참조하세요.
  - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
  - 로컬 파일 호스팅 - Envoy가 배포되는 파일 시스템의 프라이빗 키 뿐만 아니라 인증서 체인 파일의 경로를 지정합니다.
- 백엔드를 더 추가하려면 백엔드 추가를 선택합니다.

## 10. (선택 사항) 로깅

로깅을 구성하려면 Envoy에서 사용할 HTTP 액세스 로그 경로를 입력합니다. Docker 로그 드라이버를 사용하여 Envoy 로그를 Amazon CloudWatch Logs와 같은 서비스로 내보낼 수 있도록 `/dev/stdout` 경로를 사용하는 것이 좋습니다.

### Note

로그는 계속 애플리케이션의 에이전트에 의해 수집되어 대상으로 전송되어야 합니다. 이 파일 경로는 로그를 보낼 위치만 Envoy에 지시합니다.

## 11. 리스너 구성

리스너는 HTTP, HTTP/2, GRPC 및 TCP 프로토콜을 지원합니다. HTTPS는 지원되지 않습니다.

- a. 가상 노드에서 인바운드 트래픽을 예상하는 경우 리스너에 대해 포트 및 프로토콜을 지정합니다. http 리스너는 웹 소켓으로의 연결 전환을 허용합니다. 리스너 추가를 클릭하여 리스너를 여러 개 추가할 수 있습니다. 제거 버튼을 누르면 해당 리스너가 제거됩니다.
- b. (선택 사항) 연결 풀 활성화

연결 풀링은 Envoy가 로컬 애플리케이션 클러스터에 대해 동시에 설정할 수 있는 연결 수를 제한합니다. 이 작업은 로컬 애플리케이션의 연결 과부하를 방지하고 애플리케이션 요구 사항에 맞게 트래픽 셰이핑을 조정할 수 있도록 하기 위한 것입니다.

가상 노드 리스너의 대상 측 연결 풀 설정을 구성할 수 있습니다. App Mesh는 기본적으로 클라이언트 측 연결 풀 설정을 무제한으로 설정하여 메시 구성을 단순화합니다.

### Note

connectionPool 및 portMapping 프로토콜은 동일해야 합니다. 리스너 프로토콜이 tcp인 경우 maxConnections만 지정합니다. 리스너 프로토콜이 grpc 또는 http2인 경우 maxRequests만 지정합니다. 리스너 프로토콜이 http인 경우 maxConnections 및 maxPendingRequests를 모두 지정할 수 있습니다.

- 최대 연결 수에는 최대 아웃바운드 연결 수를 지정합니다.
  - (선택 사항) 대기 중인 최대 요청 수에는 Envoy가 대기하는 최대 연결 수 이후에 오버플로되는 요청 수를 지정합니다. 기본값은 2147483647입니다.
- c. (선택 사항) 이상치 탐지 활성화

클라이언트 Envoy에 적용된 이상치 탐지를 통해 클라이언트는 알려진 오류가 관찰된 연결에 대해 거의 즉각적인 조치를 취할 수 있습니다. 업스트림 서비스에 있는 개별 호스트의 상태를 추적하는 회로 차단기 구현의 한 형태입니다.

이상치 탐지는 업스트림 클러스터의 엔드포인트가 다른 엔드포인트와 다르게 작동하는지 여부를 동적으로 판단하여 정상 로드 밸런싱 세트에서 제거합니다.

**Note**

서버 가상 노드에 대한 이상치 탐지를 효과적으로 구성하기 위해 해당 가상 노드의 서비스 검색 방법은 AWS Cloud Map 또는 응답 유형 필드가 ENDPOINTS로 설정된 DNS일 수 있습니다. 응답 유형이 LOADBALANCER인 DNS 서비스 검색 방법을 사용하는 경우 Envoy 프록시는 업스트림 서비스로 라우팅하기 위해 단일 IP 주소만 선택합니다. 이렇게 하면 호스트 세트에서 비정상 호스트를 배출하는 이상치 탐지 동작이 무효화됩니다. 서비스 검색 유형과 관련된 Envoy 프록시의 동작에 대한 자세한 내용은 서비스 검색 방법 섹션을 참조하세요.

- 서버 오류에서 배출에 필요한 연속된 5xx 오류 수를 지정합니다.
  - 이상치 탐지 간격에서 배출 스윙 분석 간의 시간 간격과 단위를 지정합니다.
  - 기본 배출 기간에서 호스트가 배출되는 기본 시간 및 단위를 지정합니다.
  - 배출 비율에 배출될 수 있는 로드 밸런싱 풀의 최대 호스트 비율을 지정합니다.
- d. (선택 사항) 상태 확인 사용 - 상태 확인 정책에 대한 설정을 구성합니다.

상태 확인 정책은 선택 사항이지만 상태 정책에 대한 값을 지정한 경우 정상 임계 값, 상태 확인 간격, 상태 확인 프로토콜, 제한 시간 및 비정상 임계값에 대해 값을 지정해야 합니다.

- 상태 확인 프로토콜에 대해 프로토콜을 선택합니다. grpc를 선택한 경우 서비스는 [GRPC 상태 확인 프로토콜](#)을 준수해야 합니다.
- Health check port(상태 확인 포트)에서 상태 확인을 실행해야 할 포트를 지정합니다.
- Healthy threshold(정상 임계 값)에서 리스너를 정상으로 선언하기 전까지 발생해야 하는 연속적인 상태 확인 성공 횟수를 지정합니다.
- Health check interval(상태 확인 간격)에서 각 상태 확인 실행 사이의 시간 간격을 밀리초 단위로 지정합니다.
- Path(경로)에서 상태 확인 요청에 대한 대상 경로를 지정합니다. 이 값은 상태 확인 프로토콜이 http 또는 http2인 경우에만 사용됩니다. 다른 프로토콜에서는 이 값이 무시됩니다.
- Timeout period(제한 시간)에서 상태 확인으로부터 응답을 받을 때까지 대기할 시간을 밀리초 단위로 지정합니다.
- Unhealthy threshold(비정상 임계값)에서 리스너를 비정상적으로 선언하기 전까지 발생해야 하는 연속적인 상태 확인 실패 횟수를 지정합니다.

- e. (선택 사항) TLS 종료 활성화 - 다른 가상 노드가 TLS를 사용하여 이 가상 노드와 통신하는 방법을 구성합니다.
- 모드의 경우 리스너에서 TLS를 구성할 모드를 선택합니다.
  - 인증서 방법에서 다음 옵션 중 하나를 수행합니다. 인증서는 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [인증서 요구 사항](#) 섹션을 참조하세요.
  - AWS Certificate Manager 호스팅 - 기존 인증서를 선택합니다.
  - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
  - 로컬 파일 호스팅 - Envoy 프록시가 배포되는 파일 시스템의 프라이빗 키 뿐만 아니라 인증서 체인 파일의 경로를 지정합니다.
  - (선택 사항) 클라이언트가 인증서를 제공하는 경우 클라이언트 인증서 필요 및 아래 옵션 중 하나를 선택하여 상호 TLS 인증을 활성화합니다. 상호 TLS에 대해 자세히 알아보려면 App Mesh [상호 TLS 인증](#) 설명서를 참조하세요.
  - Envoy 보안 암호 검색 서비스(SDS) 호스팅 - Envoy가 보안 암호 검색 서비스를 사용하여 가져오는 보안 암호의 이름을 입력합니다.
  - 로컬 파일 호스팅 - Envoy가 배포되는 파일 시스템의 인증서 체인 파일 경로를 지정합니다.
  - (선택 사항) 주체 대체 이름을 입력합니다. SAN을 더 추가하려면 SAN 추가를 선택합니다. SAN은 FQDN 또는 URI 형식이어야 합니다.
- f. (선택 사항) 제한 시간

 Note

제한 시간을 기본값보다 크게 지정하는 경우 기본값보다 큰 제한 시간으로 가상 라우터와 경로를 설정해야 합니다. 그러나 제한 시간을 기본값보다 낮은 값으로 줄이는 경우 경로에서 제한 시간을 업데이트할 수도 있습니다. 자세한 내용은 [경로](#)를 참조하세요.

- 요청 제한 시간 — 리스너 프로토콜로 grpc, http 또는 http2를 선택한 경우 요청 제한 시간을 지정할 수 있습니다. 기본값은 15초입니다. 0 값은 제한 시간을 비활성화합니다.
- 유휴 기간 - 모든 리스너 프로토콜의 유휴 기간을 지정할 수 있습니다. 기본값은 300초입니다.

## 12. 가상 노드 생성을 선택하여 완료합니다.

## AWS CLI

AWS CLI를 사용하여 가상 노드를 생성하려면

다음 명령과 입력 JSON 파일을 사용하여 서비스 검색에 DNS를 사용하는 가상 노드를 생성합니다 (### 값을 원하는 값으로 대체).

1. 

```
aws appmesh create-virtual-node \
 --cli-input-json file://create-virtual-node-dns.json
```
2. 예제 create-virtual-node-dns.json의 내용은 다음과 같습니다.

```
{
 "meshName": "meshName",
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http"
 }
 }
],
 "serviceDiscovery": {
 "dns": {
 "hostname": "serviceBv1.svc.cluster.local"
 }
 }
 },
 "virtualNodeName": "nodeName"
}
```

3. 출력 예시:

```
{
 "virtualNode": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/virtualNode/nodeName",
 "createdAt": "2022-04-06T09:12:24.348000-05:00",
 "lastUpdatedAt": "2022-04-06T09:12:24.348000-05:00",
 "meshOwner": "123456789012",

```

```
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 1
 },
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http"
 }
 }
],
 "serviceDiscovery": {
 "dns": {
 "hostname": "serviceBv1.svc.cluster.local"
 }
 }
 },
 "status": {
 "status": "ACTIVE"
 },
 "virtualNodeName": "nodeName"
}
}
```

App Mesh용 AWS CLI를 사용하여 가상 노드를 생성하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [create-virtual-node](#) 명령을 참조하세요.

## 가상 노드 삭제

### Note

가상 노드가 [경로](#)의 대상으로 지정되거나 [가상 서비스](#)의 공급자로 지정된 경우 가상 노드를 삭제할 수 없습니다.

## AWS Management Console

AWS Management Console을 사용하여 가상 노드를 삭제하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 가상 노드를 삭제하려는 메시지를 선택합니다. 소유하고 있는 모든 메시와 **공유된** 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 가상 노드를 선택합니다.
4. 가상 노드 테이블에서 삭제하려는 가상 노드를 선택하고 삭제를 선택합니다. 가상 노드를 삭제하려면 가상 노드의 메시 소유자 또는 리소스 소유자 옆에 계정 ID가 나열되어야 합니다.
5. 확인 상자에 **delete**를 입력한 다음, 삭제를 선택합니다.

## AWS CLI

AWS CLI을 사용하여 가상 노드를 삭제하려면

1. 다음 명령을 사용하여 가상 노드를 삭제합니다(### 값을 원하는 값으로 대체).

```
aws appmesh delete-virtual-node \
 --mesh-name meshName \
 --virtual-node-name nodeName
```

2. 출력 예시:

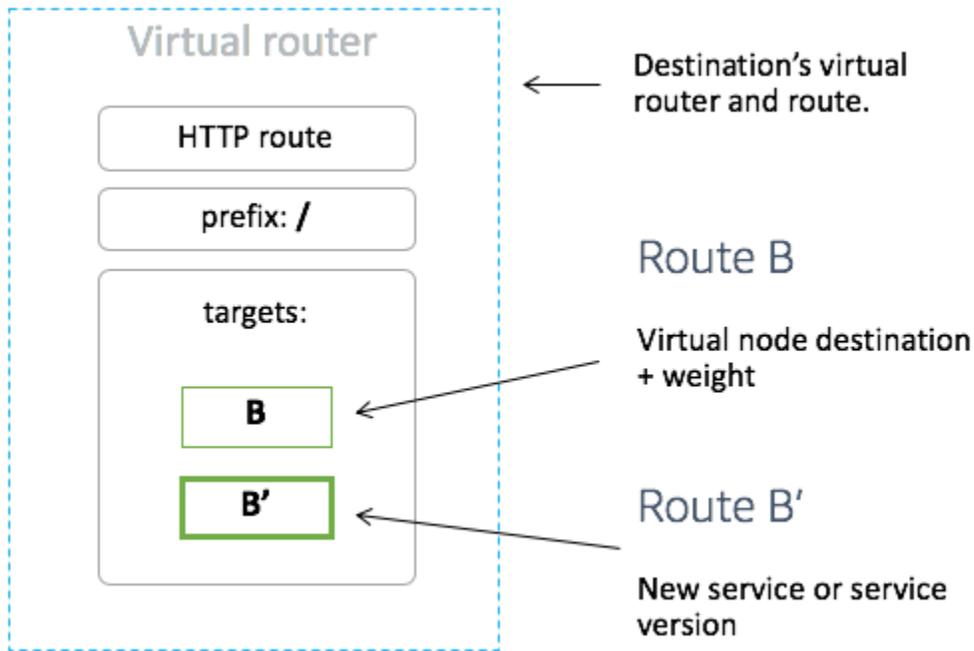
```
{
 "virtualNode": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualNode/nodeName",
 "createdAt": "2022-04-06T09:12:24.348000-05:00",
 "lastUpdatedAt": "2022-04-07T11:03:48.120000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 2
 },
 "spec": {
 "backends": [],
 "listeners": [
```

```
 {
 "portMapping": {
 "port": 80,
 "protocol": "http"
 }
 }
],
 "serviceDiscovery": {
 "dns": {
 "hostname": "serviceBv1.svc.cluster.local"
 }
 }
},
"status": {
 "status": "DELETED"
},
"virtualNodeName": "nodeName"
}
}
```

App Mesh용 AWS CLI를 사용하여 가상 노드를 삭제하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [delete-virtual-node](#) 명령을 참조하세요.

## 가상 라우터

가상 라우터는 메시 내에 있는 하나 이상의 가상 서비스에 대한 트래픽을 처리합니다. 가상 라우터를 생성한 후에는 해당 가상 라우터에 대해 수신 요청을 다른 가상 노드로 보내는 라우팅을 만들어 연결할 수 있습니다.



가상 라우터가 수신할 인바운드 트래픽을 리스너로 지정해야 합니다.

## 가상 라우터 생성

### AWS Management Console

AWS Management Console을 사용하여 가상 라우터를 생성하려면

#### Note

가상 라우터를 생성할 때는 경로를 생성된 가상 라우터에 연결할 네임스페이스 목록을 식별할 수 있도록 레이블이 있는 네임스페이스 선택기를 추가해야 합니다.

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 가상 라우터를 생성하려는 메시지를 선택합니다. 소유하고 있는 모든 메시와 **공유된** 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 Virtual routers(가상 라우터)를 선택합니다.
4. Create virtual router(가상 라우터 생성)를 선택합니다.
5. Virtual router name(가상 라우터 이름)에서 가상 라우터의 이름을 지정합니다. 최대 255개의 문자, 숫자, 하이픈 및 밑줄이 허용됩니다.

6. (선택 사항) 리스너 구성에서 가상 라우터에 대한 포트 및 프로토콜을 지정합니다. http 리스너는 웹 소켓으로의 연결 전환을 허용합니다. 리스너 추가를 클릭하여 리스너를 여러 개 추가할 수 있습니다. 제거 버튼을 누르면 해당 리스너가 제거됩니다.
7. Create virtual router(가상 라우터 생성)를 선택하여 완료합니다.

## AWS CLI

AWS CLI를 사용하여 가상 라우터를 생성하려면

다음 명령을 사용하여 가상 라우터를 생성하고 JSON을 입력합니다(### 값을 원하는 값으로 대체).

1. 

```
aws appmesh create-virtual-router \
 --cli-input-json file://create-virtual-router.json
```

2. 예제 create-virtual-router.json의 내용은 다음과 같습니다.

3. 

```
{
 "meshName": "meshName",
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http"
 }
 }
]
 },
 "virtualRouterName": "routerName"
}
```

4. 출력 예시:

```
{
 "virtualRouter": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/virtualRouter/routerName",
 "createdAt": "2022-04-06T11:49:47.216000-05:00",
 "lastUpdatedAt": "2022-04-06T11:49:47.216000-05:00",
 "meshOwner": "123456789012",

```

```

 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 1
 },
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http"
 }
 }
]
 },
 "status": {
 "status": "ACTIVE"
 },
 "virtualRouterName": "routerName"
}
}

```

App Mesh용 AWS CLI를 사용하여 가상 라우터를 생성하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [create-virtual-router](#) 명령을 참조하세요.

## 가상 라우터 삭제

### Note

[경로](#)가 있거나 [가상 서비스](#)의 공급자로 지정된 가상 라우터는 삭제할 수 없습니다.

### AWS Management Console

AWS Management Console을 사용하여 가상 라우터를 삭제하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 가상 라우터를 삭제하려는 메시지를 선택합니다. 소유하고 있는 모든 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 Virtual routers(가상 라우터)를 선택합니다.

- 가상 라우터 테이블에서 삭제하려는 가상 라우터를 선택하고 오른쪽 상단 구석에서 삭제를 선택합니다. 가상 라우터를 삭제하려면 가상 라우터의 메시 소유자 또는 리소스 소유자 옆에 계정 ID가 나열되어야 합니다.
- 확인 상자에 **delete**를 입력한 다음, 삭제를 클릭합니다.

## AWS CLI

AWS CLI를 사용하여 가상 라우터를 삭제하려면

- 다음 명령을 사용하여 가상 라우터를 삭제합니다(### 값을 원하는 값으로 대체).

```
aws appmesh delete-virtual-router \
 --mesh-name meshName \
 --virtual-router-name routerName
```

- 출력 예시:

```
{
 "virtualRouter": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/virtualRouter/routerName",
 "createdAt": "2022-04-06T11:49:47.216000-05:00",
 "lastUpdatedAt": "2022-04-07T10:49:53.402000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 2
 },
 "spec": {
 "listeners": [
 {
 "portMapping": {
 "port": 80,
 "protocol": "http"
 }
 }
]
 },
 "status": {
 "status": "DELETED"
 }
 }
}
```

```

 },
 "virtualRouterName": "routerName"
 }
}

```

App Mesh용 AWS CLI를 사용하여 가상 라우터를 삭제하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [delete-virtual-router](#) 명령을 참조하세요.

## 경로

경로가 가상 라우터와 연결됩니다. 이 경로는 가상 라우터에 대해 일치하는 요청을 찾고 관련 가상 노드에 트래픽을 분산하는 데 사용됩니다. 라우팅이 요청과 일치하는 경우 하나 이상의 대상 가상 노드에 트래픽을 분산할 수 있습니다. 각 가상 노드에 상대적 가중치를 지정할 수 있습니다. 이 주제는 서비스 메시의 경로를 사용하는 데 도움이 됩니다.

## 경로 생성

### AWS Management Console

AWS Management Console을 사용하여 경로를 생성하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 경로를 생성하려는 메시지를 선택합니다. 소유하고 있는 모든 메시와 **공유된** 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 Virtual routers(가상 라우터)를 선택합니다.
4. 새로운 라우팅을 연결할 가상 라우터를 선택합니다. 목록에 가상 라우터가 없는 경우 먼저 [가상 라우터를 생성](#)해야 합니다.
5. Routes(라우팅) 테이블에서 Create route(라우팅 생성)를 선택합니다. 경로를 생성하려면 계정 ID가 경로의 리소스 소유자로 등록되어야 합니다.
6. Route name(라우팅 이름)에서 라우팅에 사용할 이름을 지정합니다.
7. Route type(라우팅 유형)에 대해 라우팅할 프로토콜을 선택합니다. 선택하는 프로토콜은 가상 라우터에 대해 선택한 리스너 프로토콜 및 트래픽을 라우팅하는 가상 노드와 일치해야 합니다.
8. (선택 사항) Route priority(라우팅 우선 순위)에 경로에 사용할 우선 순위를 0~1000에서 선택합니다. 경로는 지정된 값을 기준으로 일치되며, 0이 가장 높은 우선 순위입니다.
9. (선택 사항) 추가 구성을 선택합니다. 아래 프로토콜 중에서 경로 유형으로 선택한 프로토콜을 선택하고 콘솔에서 원하는 대로 설정을 지정합니다.

10. 대상 구성의 경우 트래픽을 라우팅할 기존 App Mesh 가상 노드를 선택하고 가중치를 지정합니다. 대상 추가를 선택하여 대상을 더 추가할 수 있습니다. 모든 대상의 비율을 합하면 최대 100이 됩니다. 목록에 가상 노드가 없는 경우 먼저 가상 노드를 하나 [생성](#)해야 합니다. 선택한 가상 노드에 리스너가 여러 개 있는 경우 대상 포트가 필수입니다.
11. 일치 구성의 경우 다음을 지정합니다.

tcp에 대해 일치 구성을 사용할 수 없습니다.

- http/http2가 선택된 유형인 경우:

- (선택 사항) 메서드 - 들어오는 http/http2 요청에서 일치시킬 메서드 헤더를 지정합니다.
- (선택 사항) 포트 일치 - 들어오는 트래픽에 맞는 포트를 찾습니다. 이 가상 라우터에 여러 리스너가 있는 경우 포트 일치가 필수입니다.
- (선택 사항) 접두사/정확한/정규식 경로 - URL 경로를 일치시키는 방법입니다.
  - 접두사 일치 - 기본적으로 게이트웨이 경로의 일치하는 요청이 대상 가상 서비스 이름에 다시 쓰여지고 일치하는 접두사가 다시 작성됩니다(기본값 /). 가상 서비스를 구성하는 방법에 따라 가상 라우터를 사용하여 특정 접두사 또는 헤더를 기준으로 요청을 다른 가상 노드로 라우팅할 수 있습니다.

**Note**

경로/접두사 기반 일치를 활성화하면 App Mesh는 경로 정규화([normalize\\_path](#) 및 [merge\\_slashes](#))를 활성화하여 경로 혼동 취약성이 발생할 가능성을 최소화합니다.

요청에 참여하는 당사자가 서로 다른 경로 표현을 사용하는 경우 경로 혼동 취약성이 발생합니다.

- 정확한 일치 - 정확한 파라미터는 경로의 부분 일치를 비활성화하고 경로가 현재 URL과 정확히 일치하는 경우에만 경로를 반환하도록 합니다.
- 정규식 일치 - 여러 URL이 실제로 웹 사이트의 단일 페이지를 식별할 수 있는 패턴을 설명하는 데 사용됩니다.
- (선택 사항) 쿼리 파라미터 - 이 필드를 사용하면 쿼리 파라미터를 일치시킬 수 있습니다.
- (선택 사항) 헤더 - http 및 http2에 대한 헤더를 지정합니다.  $\mu$ 대상 가상 서비스로 라우팅하려면 들어오는 요청과 일치해야 합니다.
- grpc가 선택된 유형인 경우:
  - 서비스 이름 - 요청과 일치시킬 대상 서비스입니다.

- 메서드 이름 - 요청과 일치시킬 대상 메서드입니다.
- (선택 사항) 메타데이터 - 메타데이터의 존재 여부에 따라 Match를 지정합니다. 요청을 처리하려면 모두 일치해야 합니다.

12. 라우팅 생성을 선택합니다.

## AWS CLI

AWS CLI를 사용하여 경로를 생성하려면

다음 명령을 사용하여 gRPC 경로를 생성하고 JSON을 입력합니다(### 값을 원하는 값으로 대체).

1.

```
aws appmesh create-route \
 --cli-input-json file://create-route-grpc.json
```

2. 예제 create-route-grpc.json의 내용은 다음과 같습니다.

```
{
 "meshName" : "meshName",
 "routeName" : "routeName",
 "spec" : {
 "grpcRoute" : {
 "action" : {
 "weightedTargets" : [
 {
 "virtualNode" : "nodeName",
 "weight" : 100
 }
]
 },
 "match" : {
 "metadata" : [
 {
 "invert" : false,
 "match" : {
 "prefix" : "123"
 },
 "name" : "myMetadata"
 }
],
 "methodName" : "nameOfmethod",
 "serviceName" : "serviceA.svc.cluster.local"
 }
 }
 }
}
```

```

 },
 "retryPolicy" : {
 "grpcRetryEvents" : ["deadline-exceeded"],
 "httpRetryEvents" : ["server-error", "gateway-error"],
 "maxRetries" : 3,
 "perRetryTimeout" : {
 "unit" : "s",
 "value" : 15
 },
 },
 "tcpRetryEvents" : ["connection-error"]
 }
},
"priority" : 100
},
"virtualRouterName" : "routerName"
}

```

### 3. 출력 예시:

```

{
 "route": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/virtualRouter/routerName/route/routeName",
 "createdAt": "2022-04-06T13:48:20.749000-05:00",
 "lastUpdatedAt": "2022-04-06T13:48:20.749000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 1
 },
 "routeName": "routeName",
 "spec": {
 "grpcRoute": {
 "action": {
 "weightedTargets": [
 {
 "virtualNode": "nodeName",
 "weight": 100
 }
]
 }
 },
 "match": {

```

```

 "metadata": [
 {
 "invert": false,
 "match": {
 "prefix": "123"
 },
 "name": "myMetadata"
 }
],
 "methodName": "nameOfMehod",
 "serviceName": "serviceA.svc.cluster.local"
 },
 "retryPolicy": {
"grpcRetryEvents": [
 "deadline-exceeded"
],
 "httpRetryEvents": [
 "server-error",
 "gateway-error"
],
 "maxRetries": 3,
 "perRetryTimeout": {
 "unit": "s",
 "value": 15
 },
 "tcpRetryEvents": [
 "connection-error"
]
 },
 "priority": 100
},
"status": {
 "status": "ACTIVE"
},
"virtualRouterName": "routerName"
}
}

```

App Mesh용 AWS CLI를 사용하여 경로를 생성하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [create-route](#) 명령을 참조하세요.

## gRPC

### (선택 사항) 일치

- (선택 사항) 요청과 일치시킬 대상 서비스의 서비스 이름을 입력합니다. 이름을 지정하지 않으면 모든 서비스에 대한 요청이 일치됩니다.
- (선택 사항) 요청과 일치시킬 대상 메서드의 메서드 이름을 입력합니다. 이름을 지정하지 않으면 모든 메서드에 대한 요청이 일치됩니다. 메서드 이름을 지정하는 경우 서비스 이름을 지정해야 합니다.

### (선택 사항) 메타데이터

[메타데이터 추가(Add metadata)]를 선택합니다.

- (선택 사항) 경로의 기준이 될 메타데이터 이름을 입력하고 일치 유형을 선택한 다음, 일치 값을 입력합니다. Invert(반전)를 선택하면 반대 값을 일치시킵니다. 예를 들어 메타데이터 이름으로 myMetadata를 지정하고 일치 유형으로 일치로 지정하고 일치 값으로 123을 지정한 후 반전을 선택하면 메타데이터 이름이 123이 아닌 다른 이름으로 시작하는 모든 요청에 대해 경로가 일치됩니다.
- (선택 사항) 메타데이터 추가를 선택하여 메타데이터 항목을 10개까지 추가합니다.

### (선택 사항) 재시도 정책

클라이언트는 재시도 정책을 통해 간헐적 네트워크 장애 또는 간헐적 서버 측 오류로부터 스스로를 보호할 수 있습니다. 재시도 정책은 선택 사항이며, 권장 사항은 아닙니다. 재시도 제한 시간 값은 재시도 (초기 시도 포함)당 제한 시간을 정의합니다. 재시도 정책을 정의하지 않으면 App Mesh가 각 경로에 대한 기본 정책을 자동으로 생성할 수 있습니다. 자세한 내용은 [기본 경로 재시도 정책](#) 섹션을 참조하세요.

- 재시도 제한 시간에 제한 시간 동안의 단위 수를 입력합니다. 프로토콜 재시도 이벤트를 선택하는 경우 값이 필요합니다.
- 재시도 제한 시간 단위에 대해 단위를 선택합니다. 프로토콜 재시도 이벤트를 선택하는 경우 값이 필요합니다.
- 최대 재시도 수에는 요청 실패 시 최대 재시도 횟수를 입력합니다. 프로토콜 재시도 이벤트를 선택하는 경우 값이 필요합니다. 2개 이상의 값을 권장합니다.
- HTTP 재시도 이벤트를 하나 이상 선택합니다. 최소한 stream-error 및 gateway-error를 선택하는 것이 좋습니다.
- TCP 재시도 이벤트를 선택합니다.

- gRPC 재시도 이벤트를 하나 이상 선택합니다. 최소한 취소됨 및 사용할 수 없음을 선택하는 것이 좋습니다.

#### (선택 사항) 제한 시간

- 기본값은 15초입니다. 재시도 정책을 지정한 경우 여기에서 지정하는 기간은 재시도 정책에 정의된 최대 재시도 횟수를 재시도 기간과 곱한 값보다 크거나 같아야 재시도 정책이 완료될 수 있습니다. 기간을 15초보다 크게 지정하는 경우 모든 가상 노드 대상의 리스너에 대해 지정된 제한 시간도 15초 이상이어야 합니다. 자세한 내용은 [가상 노드](#)를 참조하십시오.
- 0 값은 제한 시간을 비활성화합니다.
- 경로가 유휴 상태일 수 있는 최대 기간입니다.

#### HTTP 및 HTTP/2

##### (선택 사항) 일치

- 경로가 일치해야 하는 접두사를 지정합니다. 예를 들어, 가상 서비스 이름이 `service-b.local`이고 라우팅을 `service-b.local/metrics`에 대한 요청과 일치시키려면 접두사가 `/metrics`여야 합니다. `/`를 지정하면 모든 트래픽이 라우팅됩니다.
- (선택 사항) 방법을 선택합니다.
- (선택 사항) 체계를 선택합니다. HTTP2 라우팅에만 적용됩니다.

##### (선택 사항) 헤더

- (선택 사항) Add header(추가 헤더)를 선택합니다. 경로가 기반할 Header name(헤더 이름)을 입력하고 일치 유형을 선택한 다음 일치 값을 입력합니다. Invert(반전)를 선택하면 반대 값을 일치시킵니다. 예를 들어 접두사가 `123`이고 이름이 `clientRequestId`인 헤더를 지정하고 반전을 선택하면 `123` 이외의 다른 헤더로 시작하는 헤더가 있는 요청과 일치하는 경로가 검색됩니다.
- (선택 사항) Add header(추가 헤더)를 선택합니다. 최대 10개의 헤더를 추가할 수 있습니다.

##### (선택 사항) 재시도 정책

클라이언트는 재시도 정책을 통해 간헐적 네트워크 장애 또는 간헐적 서버 측 오류로부터 스스로를 보호할 수 있습니다. 재시도 정책은 선택 사항이며, 권장 사항은 아닙니다. 재시도 제한 시간 값은 재시도 (초기 시도 포함)당 제한 시간을 정의합니다. 재시도 정책을 정의하지 않으면 App Mesh가 각 경로에

대한 기본 정책을 자동으로 생성할 수 있습니다. 자세한 내용은 [기본 경로 재시도 정책](#) 섹션을 참조하세요.

- 재시도 제한 시간에 제한 시간 동안의 단위 수를 입력합니다. 프로토콜 재시도 이벤트를 선택하는 경우 값이 필요합니다.
- 재시도 제한 시간 단위에 대해 단위를 선택합니다. 프로토콜 재시도 이벤트를 선택하는 경우 값이 필요합니다.
- 최대 재시도 수에는 요청 실패 시 최대 재시도 횟수를 입력합니다. 프로토콜 재시도 이벤트를 선택하는 경우 값이 필요합니다. 2개 이상의 값을 권장합니다.
- HTTP 재시도 이벤트를 하나 이상 선택합니다. 최소한 stream-error 및 gateway-error를 선택하는 것이 좋습니다.
- TCP 재시도 이벤트를 선택합니다.

#### (선택 사항) 제한 시간

- 요청 제한 시간 - 기본값은 15초입니다. 재시도 정책을 지정한 경우 여기에서 지정하는 기간은 재시도 정책에 정의된 최대 재시도 횟수를 재시도 기간과 곱한 값보다 크거나 같아야 재시도 정책이 완료될 수 있습니다.
- 유휴 기간 - 기본값은 300초입니다.
- 0 값은 제한 시간을 비활성화합니다.

#### Note

제한 시간을 기본값보다 크게 지정하는 경우 모든 가상 노드 참여자의 리스너에 지정된 제한 시간도 기본값보다 커야 합니다. 그러나 제한 시간을 기본값보다 낮은 값으로 줄이는 경우 가상 노드에서 제한 시간을 업데이트할 수도 있습니다. 자세한 내용은 [가상 노드](#)를 참조하십시오.

## TCP

#### (선택 사항) 제한 시간

- 유휴 기간 - 기본값은 300초입니다.
- 0 값은 제한 시간을 비활성화합니다.

## 경로 삭제

### AWS Management Console

AWS Management Console을 사용하여 경로를 삭제하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 경로를 삭제하려는 메시지를 선택합니다. 소유하고 있는 모든 메시와 [공유된](#) 모든 메시가 나열됩니다.
3. 왼쪽 탐색 창에서 Virtual routers(가상 라우터)를 선택합니다.
4. 경로를 삭제하려는 라우터를 선택합니다.
5. 경로 테이블에서 삭제하려는 경로를 선택하고 오른쪽 상단 구석에서 삭제를 선택합니다.
6. 확인 상자에 **delete**를 입력한 다음, 삭제를 클릭합니다.

### AWS CLI

AWS CLI을 사용하여 경로를 삭제하려면

1. 다음 명령을 사용하여 경로를 삭제합니다(### 값을 원하는 값으로 대체).

```
aws appmesh delete-route \
 --mesh-name meshName \
 --virtual-router-name routerName \
 --route-name routeName
```

2. 출력 예시:

```
{
 "route": {
 "meshName": "meshName",
 "metadata": {
 "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualRouter/routerName/route/routeName",
 "createdAt": "2022-04-06T13:46:54.750000-05:00",
 "lastUpdatedAt": "2022-04-07T10:43:57.152000-05:00",
 "meshOwner": "123456789012",
 "resourceOwner": "210987654321",
 "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
 "version": 2
 }
 },
}
```

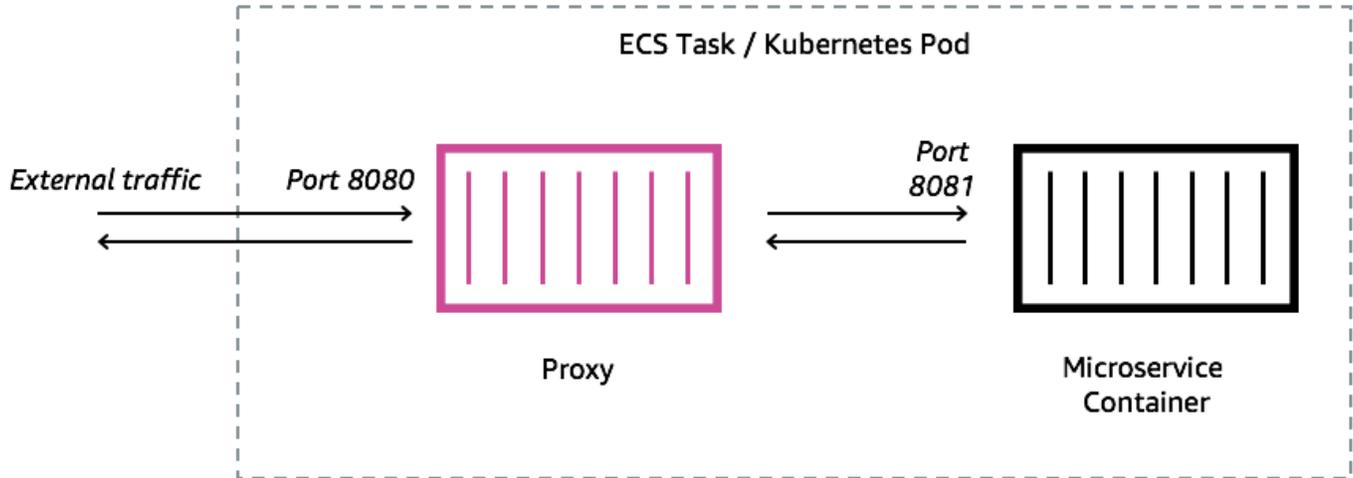
```
"routeName": "routeName",
"spec": {
 "grpcRoute": {
 "action": {
 "weightedTargets": [
 {
 "virtualNode": "nodeName",
 "weight": 100
 }
]
 },
 "match": {
 "metadata": [
 {
 "invert": false,
 "match": {
 "prefix": "123"
 },
 "name": "myMetadata"
 }
],
 "methodName": "methodName",
 "serviceName": "serviceA.svc.cluster.local"
 },
 "retryPolicy": {
 "grpcRetryEvents": [
 "deadline-exceeded"
],
 "httpRetryEvents": [
 "server-error",
 "gateway-error"
],
 "maxRetries": 3,
 "perRetryTimeout": {
 "unit": "s",
 "value": 15
 },
 "tcpRetryEvents": [
 "connection-error"
]
 }
 },
 "priority": 100
},
```

```
"status": {
 "status": "DELETED"
},
"virtualRouterName": "routerName"
}
}
```

App Mesh용 AWS CLI를 사용하여 경로를 삭제하는 방법에 대한 자세한 내용은 AWS CLI 참조의 [delete-route](#) 명령을 참조하세요.

# Envoy 이미지

AWS App Mesh [Envoy](#) 프록시를 기반으로 하는 서비스 메쉬입니다.



Envoy 프록시를 App Mesh 엔드포인트로 표시되는 Amazon ECS 태스크, Kubernetes 포드 또는 Amazon EC2 인스턴스 (예: 가상 노드 또는 가상 게이트웨이)에 추가해야 합니다. App Mesh는 최신 취약성 및 성능 업데이트로 패치된 Envoy 프록시 컨테이너 이미지를 판매합니다. App Mesh는 새 이미지를 제공하기 전에 각각의 새로운 Envoy 프록시 릴리스를 App Mesh 기능 세트에 대해 테스트합니다.

## Envoy 이미지 변형

App Mesh는 Envoy 프록시 컨테이너 이미지의 두 가지 변형을 제공합니다. 이 둘의 차이점은 Envoy 프록시가 App Mesh 데이터 플레인과 통신하는 방식과 Envoy 프록시가 서로 통신하는 방식입니다. 하나는 표준 App Mesh 서비스 엔드포인트와 통신하는 표준 이미지입니다. 또 다른 변형은 FIPS-compliant로, App Mesh FIPS 서비스 엔드포인트와 통신하고 App Mesh 서비스 간 FIPS TLS 통신에서 암호화를 적용합니다.

아래 목록에서 리전별 이미지를 선택하거나 이름이 `aws-appmesh-envoy`인 [퍼블릭 리포지토리](#)에서 이미지를 선택할 수 있습니다.

**⚠ Important**

- 2023년 6월 30일부터 Envoy 이미지 v1.17.2.0-prod 이상의 버전만 App Mesh와 함께 사용할 수 있습니다. 이전에 Envoy 이미지를 사용하고 있는 현재 고객의 v1.17.2.0 경우 기존 Envoy는 계속 호환되지만 최신 버전으로 마이그레이션하는 것이 좋습니다.
- Envoy 버전을 정기적으로 최신 버전으로 업그레이드하는 것이 가장 좋습니다. 최신 Envoy 버전만 최신 보안 패치, 기능 릴리스 및 성능 개선으로 검증됩니다.
- 버전 1.17은 Envoy의 중요한 업데이트였습니다. 자세한 내용은 [Envoy 1.17로 업데이트/마이그레이션](#)을 참조하세요.
- 버전 1.20.0.1 이상은 ARM64와 호환됩니다.
- IPv6 지원을 받으려면 Envoy 버전 1.20 이상이 필요합니다.

me-south-1, ap-east-1, ap-southeast-3, eu-south-1, il-central-1 및 af-south-1을 제외한 모든 [지원](#) 리전 교체할 수 있습니다. *Region-code* me-south-1,, ap-southeast-3 eu-south-1 il-central-1, 및 af-south-1 를 제외한 모든 지역으로

**표준**

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

**FIPS-규정 준수**

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

**me-south-1****표준**

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

**FIPS-규정 준수**

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

**ap-east-1****표준**

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

### FIPS-규정 준수

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

## ap-southeast-3

### 표준

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

### FIPS-규정 준수

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

## eu-south-1

### 표준

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

### FIPS-규정 준수

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

## il-central-1

### 표준

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

### FIPS-규정 준수

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

## af-south-1

## 표준

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

## FIPS-규정 준수

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

## Public repository

## 표준

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.29.6.1-prod
```

## FIPS-규정 준수

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.29.6.1-prod-fips
```

**Note**

Envoy 컨테이너에 512개의 CPU 유닛과 최소 64MiB의 메모리를 할당하는 것이 좋습니다. Fargate에서 설정할 수 있는 최소 메모리 양은 1024MiB입니다. 부하 증가로 인해 컨테이너 인 사이트 또는 기타 지표에 리소스가 충분하지 않은 것으로 나타나는 경우 Envoy 컨테이너에 대한 리소스 할당을 늘릴 수 있습니다.

**Note**

v1.22.0.0부터 모든 aws-appmesh-envoy 이미지 릴리스 버전은 Distroless Docker 이미지로 빌드됩니다. 이미지 크기를 줄이고 이미지에 있는 미사용 패키지의 취약성 노출을 줄일 수 있도록 변경했습니다. aws-appmesh-envoy 이미지를 기반으로 빌드하고 일부 AL2 기본 패키지 (예: yum) 와 기능을 사용하는 경우 이미지 내부의 바이너리를 복사하여 base를 사용하여 새 Docker aws-appmesh-envoy 이미지를 빌드하는 것이 좋습니다. AL2 이 스크립트를 실행하여 aws-appmesh-envoy:v1.22.0.0-prod-a12: 태그가 포함된 사용자 지정 docker 이미지를 생성합니다.

```

cat << EOF > Dockerfile
FROM public.ecr.aws/appmesh/aws-appmesh-envoy:v1.22.0.0-prod as envoy

FROM public.ecr.aws/amazonlinux/amazonlinux:2
RUN yum -y update && \
 yum clean all && \
 rm -rf /var/cache/yum

COPY --from=envoy /usr/bin/envoy /usr/bin/envoy
COPY --from=envoy /usr/bin/agent /usr/bin/agent
COPY --from=envoy /aws_appmesh_aggregate_stats.wasm /
aws_appmesh_aggregate_stats.wasm

CMD ["/usr/bin/agent"]
EOF

docker build -f Dockerfile -t aws-appmesh-envoy:v1.22.0.0-prod-al2 .

```

Amazon에서 이 컨테이너 이미지에 대한 ECR 액세스는 AWS Identity and Access Management (IAM) 에 의해 제어됩니다. 따라서 Amazon에 대한 읽기 액세스 권한이 있는지 확인하는 IAM 데 사용해야 ECR 합니다. 예를 ECS 들어 Amazon을 사용할 때 Amazon 작업에 적절한 작업 실행 역할을 할당할 수 있습니다. ECS 특정 Amazon ECR 리소스에 대한 액세스를 제한하는 IAM 정책을 사용하는 경우 aws-appmesh-envoy 리포지토리를 식별하는 지역별 Amazon 리소스 이름 (ARN) 에 대한 액세스를 허용하는지 확인하십시오. 예를 들어, us-west-2 리전에서는 arn:aws:ecr:us-west-2:840364872350:repository/aws-appmesh-envoy 리소스에 대한 액세스를 허용합니다. 자세한 내용은 [Amazon ECR 관리형 정책을](#) 참조하십시오. Amazon EC2 인스턴스에서 Docker를 사용하는 경우 리포지토리에서 Docker를 인증하십시오. 자세한 내용은 [레지스트리 인증](#)을 참조하십시오.

아직 업스트림 Envoy 이미지에 병합되지 않은 Envoy 변경 사항을 기준으로 하는 새로운 App Mesh 기능이 출시되는 경우가 있습니다. Envoy 변경 사항이 업스트림에 병합되기 전에 이러한 새로운 App Mesh 기능을 사용하려면 App Mesh에서 제공하는 Envoy 컨테이너 이미지를 사용해야 합니다. 변경 사항 목록은 Envoy Upstream 라벨과 관련된 [App Mesh GitHub 로드맵 문제를](#) 참조하십시오. App Mesh Envoy 컨테이너 이미지를 최상의 지원 옵션으로 사용하는 것이 좋습니다.

## Envoy 구성 변수

다음 환경 변수를 사용하여 App Mesh 가상 노드 태스크 그룹의 Envoy 컨테이너를 구성합니다.

**Note**

App Mesh Envoy 1.17은 Envoy의 v2 XD를 지원하지 않습니다. API Envoy 구성 파일을 허용하는 [Envoy 구성 변수를 사용하는 경우 해당 변수를](#) 최신 v3 XD로 업데이트해야 합니다. API

## 필수 변수

모든 App Mesh Envoy 컨테이너에는 다음과 같은 환경 변수가 필요합니다. 이 변수는 Envoy 이미지 1.15.0 이상 버전에서만 사용할 수 있습니다. 이전 버전의 이미지를 사용하는 경우 APPMESH\_VIRTUAL\_NODE\_NAME 변수를 대신 설정해야 합니다.

### APPMESH\_RESOURCE\_ARN

Envoy 컨테이너를 작업 그룹에 추가할 때는 이 환경 변수를 작업 그룹이 나타내는 가상 노드 또는 가상 ARN 게이트웨이의 환경 변수로 설정하십시오. 다음 목록에는 예제가 포함되어 있습니다.

#### ARNs

- 가상 노드 — `arn:aws:appmesh:Region-code:111122223333:메시/meshName/virtualNode/virtualNodeName`
- 가상 게이트웨이 — `arn:aws:앱메시:Region-code:111122223333:메시/meshName/virtualGateway/virtualGatewayName`

[App Mesh 미리보기 채널](#)을 사용할 때는 ARNs 다음을 사용해야 합니다. `us-west-2` 지역 및 용도 `appmesh-preview` 대신 사용하세요 `appmesh`. 예를 들어 App Mesh 프리뷰 ARN 채널의 가상 노드는 다음과 같습니다 `arn:aws:appmesh-preview:us-west-2:111122223333:mesh/meshName/virtualNode/virtualNodeName`.

## 선택적 변수

App Mesh Envoy 컨테이너의 경우 다음 환경 변수는 선택 사항입니다.

### ENVOY\_LOG\_LEVEL

Envoy 컨테이너의 로그 수준을 지정합니다.

유효한 값: `trace, debug, info, warn, error, critical, off`

기본값: `info`

## ENVOY\_INITIAL\_FETCH\_TIMEOUT

초기화 프로세스 중에 Envoy가 관리 서버의 첫 번째 구성 응답을 기다리는 시간을 지정합니다.

자세한 내용은 Envoy 설명서의 [구성 소스](#)를 읽어보세요. 0으로 설정하면 제한 시간이 없습니다.

기본값: 0

## ENVOY\_CONCURRENCY

Envoy를 시작하는 동안 `--concurrency` 명령줄 옵션을 설정합니다. 기본적으로 설정되어 있지 않습니다. 이 옵션은 Envoy 버전 v1.24.0.0-prod 이상에서 사용할 수 있습니다.

자세한 내용은 Envoy 설명서의 [명령줄 옵션](#)을 참조하세요.

## 관리 변수

이러한 환경 변수를 사용하여 Envoy의 관리 인터페이스를 구성합니다.

### ENVOY\_ADMIN\_ACCESS\_PORT

Envoy가 수신 대기할 사용자 지정 관리 포트를 지정합니다. 기본값: 9901.

#### Note

Envoy 관리 포트는 가상 게이트웨이 또는 가상 노드의 리스너 포트와 달라야 합니다.

### ENVOY\_ADMIN\_ACCESS\_LOG\_FILE

Envoy 액세스 로그를 쓸 사용자 지정 경로를 지정합니다. 기본값: `/tmp/envoy_admin_access.log`.

### ENVOY\_ADMIN\_ACCESS\_ENABLE\_IPV6

Envoy의 관리 인터페이스가 IPv6 트래픽을 허용하도록 전환합니다. 그러면 이 인터페이스가 IPv4 및 IPv6 트래픽을 모두 수락할 수 있습니다. 기본적으로 이 플래그는 `false`로 설정되며 Envoy는 IPv4 트래픽만 수신합니다. 이 변수는 Envoy 이미지 버전 1.22.0 이상에서만 사용할 수 있습니다.

## Agent 변수

이러한 환경 변수를 사용하여 AWS App Mesh Envoy용 에이전트를 구성하십시오. 자세한 내용은 App Mesh [Agent for Envoy](#)를 참조하세요.

### APPNET\_ENVOY\_RESTART\_COUNT

Envoy 프록시 프로세스가 종료된 경우 이 Agent가 실행 중인 태스크 또는 포드 내에서 이 프로세스를 다시 시작하는 횟수를 지정합니다. 또한 이 Agent는 Envoy가 종료될 때마다 종료 상태를 기록하여 문제 해결을 용이하게 합니다. 이 변수의 기본값은 0입니다. 기본값이 설정된 경우 이 Agent는 프로세스를 다시 시작하려고 시도하지 않습니다.

기본값: 0

최대: 10

### PID\_POLL\_INTERVAL\_MS

Agent가 Envoy 프록시의 프로세스 상태를 확인하는 간격을 밀리초 단위로 지정합니다. 기본값은 100입니다.

기본값: 100

최소: 100

최대: 1000

### LISTENER\_DRAIN\_WAIT\_TIME\_S

Envoy 프록시가 프로세스 종료 전에 활성 연결이 닫힐 때까지 기다리는 시간을 초 단위로 지정합니다.

기본값: 20

최소: 5

최대: 110

### APPNET\_AGENT\_ADMIN\_MODE

Agent의 관리 인터페이스 서버를 시작하고 tcp 주소 또는 Unix 소켓에 바인딩합니다.

유효값: tcp, uds

## APPNET\_AGENT\_HTTP\_PORT

Agent의 관리 인터페이스를 tcp 모드로 바인딩하는 데 사용할 포트를 지정합니다. `uid != 0`인 경우 포트 값이 1024보다 큰지 확인합니다. 포트가 65535보다 작은지 확인합니다.

기본값: 9902

## APPNET\_AGENT\_ADMIN\_UDS\_PATH

uds 모드에서 Agent 관리 인터페이스의 Unix 도메인 소켓 경로를 지정합니다.

기본값: `/var/run/ecs/appnet_admin.sock`

## 추적 변수

다음 추적 드라이버를 전혀 구성하지 않거나 하나를 구성할 수 있습니다.

### AWS X-Ray 변수

다음 환경 변수를 사용하여 AWS X-Ray으로 App Mesh를 구성합니다. 자세한 내용은 [개발자 안내서 AWS X-Ray](#)를 참조하세요.

## ENABLE\_ENVOY\_XRAY\_TRACING

127.0.0.1:2000을 기본 대몬(daemon) 엔드포인트로 사용하여 X-Ray 추적을 활성화합니다. 활성화하려면 값을 1로 설정합니다. 기본값은 0입니다.

## XRAY\_DAEMON\_PORT

포트 값을 지정하여 기본 X-Ray 에이전트 포트 2000을 재정의합니다.

## XRAY\_SAMPLING\_RATE

샘플링 속도를 지정하여 X-Ray 추적 프로그램의 기본 샘플링 속도인 0.05(5%)를 재정의합니다. 값을 0과 1.00(100%) 사이의 십진수로 지정합니다. `XRAY_SAMPLING_RULE_MANIFEST`를 지정하는 경우 이 값이 재정의됩니다. 이 변수는 Envoy 이미지 버전 v1.19.1.1-prod 이상에서 지원됩니다.

## XRAY\_SAMPLING\_RULE\_MANIFEST

Envoy 컨테이너 파일 시스템에서 파일 경로를 지정하여 X-Ray 추적 프로그램에 대한 현지화된 사용자 지정 샘플링 규칙을 구성합니다. 자세한 내용은 AWS X-Ray 개발자 안내서의 [샘플링 규칙](#)을 참조하세요. 이 변수는 Envoy 이미지 버전 v1.19.1.0-prod 이상에서 지원됩니다.

## XRAY\_SEGMENT\_NAME

추적 프로그램의 세그먼트 이름을 지정하여 기본 X-Ray 세그먼트 이름을 재정의합니다. 기본적으로 이 값은 mesh/resourceName으로 설정됩니다. 이 변수는 Envoy 이미지 버전 v1.23.1.0-prod 이상에서 지원됩니다.

## Datadog 추적 변수

다음 환경 변수는 Datadog 에이전트 추적 프로그램을 사용하여 App Mesh를 구성하는 데 도움이 됩니다. 자세한 내용을 알아보려면 Datadog 설명서의 [Agent 구성](#)을 참조하세요.

## ENABLE\_ENVOY\_DATADOG\_TRACING

127.0.0.1:8126을 기본 Datadog 에이전트 엔드포인트로 사용하여 Datadog 추적 수집을 활성화합니다. 활성화하려면 값을 1(기본값: 0)로 설정합니다.

## DATADOG\_TRACER\_PORT

포트 값을 지정하여 기본 Datadog 에이전트 포트 8126을 재정의합니다.

## DATADOG\_TRACER\_ADDRESS

IP 주소를 지정하여 기본 Datadog 에이전트 주소 127.0.0.1을 재정의합니다.

## DD\_SERVICE

추적의 서비스 이름을 지정하여 기본 DataDog 서비스 이름 envoy-meshName/virtualNodeName을 재정의합니다. 이 변수는 Envoy 이미지 버전 v1.18.3.0-prod 이상에서 지원됩니다.

## Jaeger 추적 변수

다음 환경 변수를 사용하여 Jaeger 추적으로 App Mesh를 구성합니다. 자세한 내용은 Jaeger 설명서의 [시작하기](#)를 참조하세요. 이러한 변수는 Envoy 이미지 버전 1.16.1.0-prod 이상에서 지원됩니다.

## ENABLE\_ENVOY\_JAEGER\_TRACING

127.0.0.1:9411을 기본 Jaeger 엔드포인트로 사용하여 Jaeger 추적 수집을 활성화합니다. 활성화하려면 값을 1(기본값: 0)로 설정합니다.

## JAEGER\_TRACER\_PORT

포트 값을 지정하여 기본 Jaeger 포트 9411을 재정의합니다.

## JAEGER\_TRACER\_ADDRESS

IP 주소를 지정하여 기본 Jaeger 주소 127.0.0.1을 재정의합니다.

## JAEGER\_TRACER\_VERSION

수집기에 JSON 또는 PROTO 인코딩 형식의 추적이 필요한지 여부를 지정합니다. 기본적으로 이 값은 PROTO로 설정됩니다. 이 변수는 Envoy 이미지 버전 v1.23.1.0-prod 이상에서 지원됩니다.

## Envoy 추적 변수

자체 추적 구성을 사용하도록 다음 환경 변수를 설정합니다.

## ENVOY\_TRACING\_CFG\_FILE

Envoy 컨테이너 파일 시스템에서 파일 경로를 지정합니다. 자세한 내용은 Envoy 설명서의 [config.trace.v3.Tracing](#)을 참조하세요.

### Note

추적 구성에 추적 클러스터를 지정해야 하는 경우 동일한 추적 구성 파일의 `static_resources`에서 관련 클러스터 구성을 구성해야 합니다. 예를 들어 Zipkin에는 추적 수집기를 호스팅하는 클러스터 이름에 대한 `collector_cluster` 필드가 있으며 해당 클러스터를 정적으로 정의해야 합니다.

## DogStatsD 변수

다음 환경 변수를 사용하여 DogStats D로 App Mesh를 구성합니다. 자세한 내용은 [DogStatsD](#) 설명서를 참조하십시오.

## ENABLE\_ENVOY\_DOG\_STATSD

기본 데몬 127.0.0.1:8125 엔드포인트로 사용하여 DogStats D 통계를 활성화합니다. 활성화하려면 값을 1로 설정합니다.

## STATSD\_PORT

기본 DogStats D 데몬 포트를 재정의하려면 포트 값을 지정하십시오.

## STATSD\_ADDRESS

기본 DogStats D 데몬 IP 주소를 재정의하려면 IP 주소 값을 지정합니다. 기본값: 127.0.0.1. 이 변수는 Envoy 이미지 1.15.0 이상 버전에서만 사용할 수 있습니다.

## STATSD\_SOCKET\_PATH

D 데몬의 유닉스 도메인 소켓을 지정하십시오. DogStats 이 변수를 지정하지 않고 D가 활성화된 경우 이 값은 DogStats DogStats D 데몬 IP 주소 포트의 기본값입니다. 127.0.0.1:8125 stats 싱크 구성을 포함하는 ENVOY\_STATS\_SINKS\_CFG\_FILE 변수를 지정하면 모든 D 변수를 무시합니다. DogStats 이 변수는 Envoy 이미지 버전 v1.19.1.0-prod 이상에서 지원됩니다.

## App Mesh 변수

다음 변수는 App Mesh를 구성하는 데 도움이 됩니다.

### APPMESH\_PREVIEW

값을 1으로 설정하여 App Mesh 프리뷰 채널 엔드포인트에 연결합니다. App Mesh 프리뷰 채널 사용에 대한 자세한 내용은 [App Mesh 프리뷰 채널](#) 섹션을 참조하세요.

### APPMESH\_RESOURCE\_CLUSTER

기본적으로 App Mesh는 Envoy가 지표 및 트레이스에서 자신을 참조할 때 APPMESH\_RESOURCE\_ARN에서 지정한 리소스의 이름을 사용합니다. APPMESH\_RESOURCE\_CLUSTER 환경 변수를 사용자 고유의 이름으로 설정하여 이 동작을 재정의할 수 있습니다. 이 변수는 Envoy 이미지 1.15.0 이상 버전에서만 사용할 수 있습니다.

### APPMESH\_METRIC\_EXTENSION\_VERSION

값을 1으로 설정하여 App Mesh 지표 확장을 활성화합니다. App Mesh 지표 확장 사용에 대한 자세한 내용은 [App Mesh의 지표 확장](#) 섹션을 참조하세요.

### APPMESH\_DUALSTACK\_ENDPOINT

값을 1으로 설정하여 App Mesh Dual Stack 엔드포인트에 연결합니다. 이 플래그가 설정되면 Envoy는 이중 스택 지원 도메인을 사용합니다. 기본적으로 이 플래그는 false로 설정되며 IPv4 도메인에만 연결됩니다. 이 변수는 Envoy 이미지 버전 1.22.0 이상에서만 사용할 수 있습니다.

## Envoy 통계 변수

다음 환경 변수를 사용하여 Envoy 통계로 App Mesh를 구성할 수 있습니다. 자세한 내용은 [Envoy 통계](#) 설명서를 참조하세요.

## ENABLE\_ENVOY\_STATS\_TAGS

App Mesh에서 정의한 태그 `appmesh.mesh` 및 `appmesh.virtual_node`를 사용할 수 있도록 합니다. [자세한 내용은 `config.metrics.v3`을 참조하십시오.](#) `TagSpecifier` Envoy 설명서에서 확인할 수 있습니다. 활성화하려면 값을 1로 설정합니다.

## ENVOY\_STATS\_CONFIG\_FILE

Envoy 컨테이너 파일 시스템에서 파일 경로를 지정하여 기본 통계 태그 구성 파일을 사용자 고유의 구성 파일로 재정의합니다. [자세한 내용은 `config.metrics.v3`을 참조하십시오.](#) `StatsConfig`.

### Note

통계 필터가 포함된 사용자 지정 통계 구성을 설정하면 Envoy가 더 이상 사용자 환경의 App Mesh 상태와 제대로 동기화되지 않는 상태가 될 수 있습니다. 이것이 Envoy의 [버그](#)입니다. Envoy에서 통계 필터링을 수행하지 않는 것이 좋습니다. 필터링이 꼭 필요한 경우 로드맵에 이 [문제](#)의 몇 가지 해결 방법을 나열했습니다.

## ENVOY\_STATS\_SINKS\_CFG\_FILE

Envoy 컨테이너 파일 시스템에서 파일 경로를 지정하여 기본 구성을 사용자 고유의 구성 파일로 재정의합니다. 자세한 내용은 [config.metrics.v3을 참조하십시오.](#) `StatsSink` Envoy 설명서에서 확인할 수 있습니다.

## 더 이상 사용되지 않는 변수

환경 변수는 `APPMESH_VIRTUAL_NODE_NAME` 및 `APPMESH_RESOURCE_NAME`은 Envoy 버전 1.15.0 이상에서 더 이상 지원되지 않습니다. 하지만 기존 메시에서는 여전히 지원됩니다. Envoy 버전 1.15.0 이상에서 이러한 변수를 사용하는 대신, 모든 App Mesh 엔드포인트에 `APPMESH_RESOURCE_ARN`을 사용하세요.

## App Mesh에서 설정한 Envoy 기본값

다음 섹션에서는 App Mesh에서 설정한 경로 재시도 정책 및 회로 차단기의 Envoy 기본값에 대한 정보를 제공합니다.

## 기본 경로 재시도 정책

2020년 7월 29일 이전에 계정에 메시가 없었던 경우, App Mesh는 2020년 7월 29일 또는 그 이후에 계정 내 모든 메시의 모든 HTTP, HTTP /2, gRPC 요청에 대해 기본 Envoy 경로 재시도 정책을 자동으로 생성합니다. 2020년 7월 29일 이전에 계정에 메시가 있었으면 2020년 7월 29일 이전, 당일 또는 이후에 존재했던 Envoy 경로에 대한 기본 정책이 생성되지 않은 것입니다. 지원팀에 티켓을 [개설하지](#) 않는 말입니다. AWS 지원 팀이 티켓을 처리한 후에는 티켓 처리 날짜 당일 또는 그 이후에 App Mesh가 생성하는 모든 향후 Envoy 경로에 대해 기본 정책이 생성됩니다. [Envoy 경로 재시도 정책에 대한 자세한 내용은 config.route.v3을 참조하십시오. RetryPolicy](#) Envoy 설명서에서 확인할 수 있습니다.

App Mesh는 App Mesh [경로](#)를 생성하거나 App Mesh [가상 서비스](#)를 위한 가상 노드 공급자를 정의할 때 Envoy 경로를 생성합니다. App Mesh 경로 재시도 정책을 생성할 수는 있지만 가상 노드 공급자에 대한 App Mesh 재시도 정책을 생성할 수는 없습니다.

기본 정책은 App Mesh를 통해 볼 수 없습니다. 기본 정책은 Envoy를 통해서만 볼 수 있습니다. 구성을 보려면 [관리 인터페이스를 활성화하고](#) Envoy에 config\_dump에 대한 요청을 보내세요. 기본 정책에는 다음 설정이 포함되어 있습니다.

- 최대 재시도 - 2
- gRPC 재시도 이벤트 — UNAVAILABLE
- HTTP재시도 이벤트 — 503

### Note

특정 HTTP 오류 코드를 찾는 App Mesh 라우트 재시도 정책을 생성할 수 없습니다. 하지만 App Mesh 경로 재시도 정책은 server-error 또는 gateway-error를 찾을 수 있습니다. 두 가지 모두 503 오류를 포함합니다. 자세한 내용은 [경로](#) 단원을 참조하십시오.

- TCP재시도 이벤트 - 및 connect-failure refused-stream

### Note

이러한 이벤트 중 하나를 찾는 App Mesh 경로 재시도 정책을 생성할 수는 없습니다. 하지만 App Mesh 경로 재시도 정책에서 connect-failure와 동급의 connection-error를 찾을 수 있습니다. 자세한 내용은 [경로](#) 단원을 참조하십시오.

- 재설정 - 업스트림 서버가 전혀 응답하지 않는 경우(연결 끊기/재설정/읽기 제한 시간 초과) Envoy가 다시 시도하려고 합니다.

## 기본 회로 차단기

App Mesh에 Envoy를 배포하면 일부 회로 차단기 설정의 Envoy 기본값이 설정됩니다. 자세한 내용은 [클러스터를 참조하십시오. CircuitBreakers.Envoy 설명서의 임계값](#). 이러한 설정은 App Mesh를 통해 볼 수 없습니다. API Envoy를 통해서만 설정을 볼 수 있습니다. 구성을 보려면 [관리 인터페이스를 활성화하고](#) Envoy에 config\_dump에 대한 요청을 보내세요.

2020년 7월 29일 이전에는 계정에 메시가 없었으면 2020년 7월 29일 또는 그 이후에 생성된 메시에 배포하는 각 Envoy에 대해 App Mesh는 다음 설정의 Envoy 기본값을 변경하여 회로 차단기를 효과적으로 비활성화합니다. 2020년 7월 29일 이전에 계정에 메시가 있는 경우, 지원팀에 티켓을 [개설하지](#) 않는 한 App Mesh에서 배포하는 모든 Envoy에 대해 Envoy 기본값이 2020년 7월 29일 또는 7월 29일 이후에 배포하는 모든 Envoy에 설정됩니다. AWS 지원 부서에서 티켓을 처리하면 티켓 처리 날짜 이후에 배포하는 모든 Envoy에서 다음 Envoy 설정에 대한 App Mesh의 기본값이 App Mesh에서 설정됩니다.

- **max\_requests** – 2147483647
- **max\_pending\_requests** – 2147483647
- **max\_connections** – 2147483647
- **max\_retries** – 2147483647

### Note

Envoy에 Envoy 또는 App Mesh 기본 회로 차단기 값이 있더라도 값을 수정할 수 없습니다.

## Envoy 1.17로 업데이트/마이그레이션

### 시크릿 디스커버리 서비스 포함 SPIRE

App Mesh와 함께 SPIRE (SPIFFERuntime Environment) 을 사용하여 서비스에 신뢰 인증서를 배포하는 경우 최소한 [SPIRE에이전트](#) 버전 0.12.0 (2020년 12월 출시) 을 사용하고 있는지 확인하세요. 이 버전은 1.16 이후 Envoy 버전을 지원할 수 있는 첫 번째 버전입니다.

### 정규식 변경 사항

1.17 Envoy부터 App Mesh는 Envoy가 기본적으로 정규 표현식 엔진을 사용하도록 구성합니다. [RE2](#) 대부분의 사용자에게는 이러한 변경이 명확하게 보이지만 Routes 또는 Gateway Routes의 일치 항목이 더 이상 정규 표현식에서 미리 보기 또는 역참조를 허용하지 않습니다.

## 긍정 및 부정 미리 보기

긍정 - 긍정 미리 보기는 `?=`로 시작하는 괄호로 묶인 표현식입니다.

```
(?=example)
```

이러한 방법은 문자를 일치의 일부로 사용하지 않고도 문자열을 일치시킬 수 있기 때문에 문자열 대체 시 가장 유용합니다. App Mesh는 정규식 문자열 대체를 지원하지 않으므로 정규식 문자열 대체를 일반 일치로 바꾸는 것이 좋습니다.

```
(example)
```

부정 - 부정 미리 보기는 `?!`로 시작하는 괄호로 묶인 표현식입니다.

```
ex(?!amp)le
```

괄호로 묶인 표현식은 표현식의 일부가 주어진 입력과 일치하지 않음을 어설선하는 데 사용됩니다. 대부분의 경우에 이를 0 한정사로 바꿀 수 있습니다.

```
ex(amp){0}le
```

표현식 자체가 문자 클래스인 경우 전체 클래스를 부정하고 `?`를 사용하여 선택 사항으로 표시할 수 있습니다.

```
prefix(?![0-9])suffix => prefix[^0-9]?suffix
```

사용 사례에 따라 경로를 변경하여 이를 처리할 수도 있습니다.

```
{
 "routeSpec": {
 "priority": 0,
 "httpRoute": {
 "match": {
 "headers": [
 {
 "name": "x-my-example-header",
 "match": {
 "regex": "^prefix(?!suffix)"
 }
 }
]
 }
 }
 }
}
```

```

]
 }
}
{
 "routeSpec": {
 "priority": 1,
 "httpRoute": {
 "match": {
 "headers": [
 {
 "name": "x-my-example-header",
 "match": {
 "regex": "^prefix"
 }
 }
]
 }
 }
 }
}
}

```

첫 번째 경로 일치에서는 “prefix”로 시작하지만 뒤에 “suffix”가 없는 헤더를 찾습니다. 두 번째 경로는 “suffix”로 끝나는 헤더를 포함하여 “prefix”로 시작하는 다른 모든 헤더를 일치시키는 역할을 합니다. 대신 부정 미리 보기를 없애기 위한 방법으로 이러한 방식을 거꾸로 진행할 수도 있습니다.

```

{
 "routeSpec": {
 "priority": 0,
 "httpRoute": {
 "match": {
 "headers": [
 {
 "name": "x-my-example-header",
 "match": {
 "regex": "^prefix.*?suffix"
 }
 }
]
 }
 }
 }
}

```

```

 }
 }
 {
 "routeSpec": {
 "priority": 1,
 "httpRoute": {
 "match": {
 "headers": [
 {
 "name": "x-my-example-header",
 "match": {
 "regex": "^prefix"
 }
 }
]
 }
 }
 }
 }
}

```

이 예제는 경로를 반대로 진행하여 “suffix”로 끝나는 헤더에 더 높은 우선 순위를 부여하고, “prefix”로 시작하는 다른 모든 헤더는 우선 순위가 낮은 경로에서 일치시킵니다.

## 역참조

역참조는 괄호로 묶인 이전 그룹을 반복하여 더 짧은 식을 작성하는 방법입니다. 형식은 다음과 같습니다.

```
(group1)(group2)\1
```

백슬래시 \ 뒤에 숫자가 오는 것은 표현식에서 괄호로 묶인 n번째 그룹의 자리 표시자 역할을 합니다. 이 예제에서는 \1이 (group1)을 한 번 더 쓰기 위한 대체 방법으로 사용됩니다.

```
(group1)(group2)(group1)
```

예제와 같이 역참조를 참조 대상 그룹으로 바꾸면 이러한 문제를 간단히 제거할 수 있습니다.

## Agent for Envoy

이 Agent는 App Mesh용으로 제공되는 Envoy 이미지 내의 프로세스 관리자입니다. 이 Agent는 Envoy가 계속 실행되도록 하고 정상 상태를 유지하며 가동 중지 시간을 줄입니다. Envoy 통계 및 보조 데이터를 필터링하여 App Mesh에서의 Envoy 프록시 작업을 간략하게 보여 줍니다. 이렇게 하면 관련 오류를 더 빨리 해결할 수 있습니다.

이 Agent를 사용하여 프록시가 비정상일 때 Envoy 프록시를 다시 시작할 횟수를 구성할 수 있습니다. 오류가 발생하는 경우 이 Agent는 Envoy가 종료될 때 최종 종료 상태를 기록합니다. 이 정보는 오류를 문제를 해결할 때 사용할 수 있습니다. 또한 이 Agent는 Envoy 연결 드레이닝을 촉진하여 장애에 대한 애플리케이션 복원력을 높일 수 있도록 합니다.

다음 변수를 사용하여 Agent for Envoy를 구성하세요.

- `APPNET_ENVOY_RESTART_COUNT` — 이 변수를 0이 아닌 값으로 설정하면 이 Agent는 폴링 시 프록시 프로세스 상태가 비정상일 때 사용자가 설정한 수만큼 Envoy 프록시 프로세스를 다시 시작하려고 시도합니다. 이 경우 프록시 상태 검사 실패 시 컨테이너 오케스트레이터가 태스크나 포드를 교체하는 것보다 더 빠르게 재시작할 수 있어 가동 중지 시간을 줄일 수 있습니다.
- `PID_POLL_INTERVAL_MS` — 이 변수를 구성할 때 기본값은 100으로 유지됩니다. 이 값으로 설정하면 컨테이너 오케스트레이터 상태 검사를 통해 태스크나 포드를 교체하는 것보다 종료 시 Envoy 프로세스를 더 빠르게 감지하고 재시작할 수 있습니다.
- `LISTENER_DRAIN_WAIT_TIME_S` — 이 변수를 구성할 때는 태스크 또는 포드 종지를 위해 설정된 컨테이너 오케스트레이터 제한 시간을 고려하세요. 예를 들어 이 값이 오케스트레이터 제한 시간보다 크면 Envoy 프록시는 오케스트레이터가 태스크 또는 포드를 강제로 중지할 때까지만 드레이닝할 수 있습니다.
- `APPNET_AGENT_ADMIN_MODE` — 이 변수를 `tcp` 또는 `uds`로 설정하면 이 Agent는 로컬 관리 인터페이스를 제공합니다. 이 관리 인터페이스는 Envoy 프록시와 상호 작용하기 위한 안전한 엔드포인트 역할을 하며 상태 확인, 원격 분석 데이터에 대한 다음 API를 제공하고 프록시의 작동 상태를 요약합니다.
  - `GET /status` — Envoy 통계를 쿼리하고 서버 정보를 반환합니다.
  - `POST /drain_listeners` — 모든 인바운드 리스너를 드레이닝합니다.
  - `POST /enableLogging?level=<desired_level>` — 모든 로거의 Envoy 로깅 수준을 변경합니다.
  - `GET /stats/prometheus` — Prometheus 형식으로 Envoy 통계를 표시합니다.
  - `GET /stats/prometheus?usedonly` — Envoy가 업데이트한 통계만 표시합니다.

Agent 구성 변수에 대한 자세한 내용은 [Envoy 구성 변수](#)를 참조하세요.

새 AWS App Mesh Agent는 버전 1.21.0.0부터 App Mesh에 최적화된 Envoy 이미지에 포함되며 고객 태스크나 포드에 추가 리소스를 할당할 필요가 없습니다.

# App Mesh 관찰성

App Mesh를 사용하여 얻을 수 있는 이점 중 하나는 마이크로서비스 애플리케이션에 대한 가시성이 향상된다는 것입니다. App Mesh는 다양한 로깅, 지표 및 추적 솔루션과 함께 사용할 수 있습니다.

Envoy 프록시 및 App Mesh는 애플리케이션과 프록시를 더 명확하게 볼 수 있도록 다음과 같은 유형의 도구를 제공합니다.

- [로깅](#)
- [지표](#)
- [추적](#)

## 로깅

가상 노드와 가상 게이트웨이를 생성할 때 Envoy 액세스 로그를 구성할 수 있습니다. 이 로그는 콘솔에 서는 가상 노드의 로깅 섹션에 있으며 가상 게이트웨이가 워크플로를 생성하거나 편집합니다.

### Logging

HTTP access logs path - *optional*

The path used to send logging information for the virtual node. App Mesh recommends using the standard out I/O stream.

```
/dev/stdout
```

 Logs must still be ingested by an agent in your application and sent to a destination. This file path only instructs Envoy where to send the logs.

위 이미지는 Envoy 액세스 로그에 대한 `/dev/stdout`의 로깅 경로를 보여 줍니다.

`format`의 경우 두 가지 가능한 형식 `json` 또는 `text` 중 하나와 패턴을 지정합니다. `json`에서는 키 페어를 가져와 JSON 구조체로 변환한 후 Envoy에 전달합니다.

다음 코드 블록에서는 AWS CLI에서 사용할 수 있는 JSON 표현을 보여 줍니다.

```
"logging": {
 "accessLog": {
 "file": {
 "path": "/dev/stdout",
 "format" : {
```

```

// Exactly one of json or text should be specified
"json": [// json will be implemented with key pairs
 {
 "key": "string",
 "value": "string"
 }
]
"text": "string" //e.g. "%LOCAL_REPLY_BODY%:%RESPONSE_CODE%:path=
%REQ(:path)%\n"
}
}
}
}

```

### ⚠ Important

입력 패턴이 Envoy에 유효한지 확인하십시오. 유효하지 않으면 Envoy가 업데이트를 거부하고 최신 변경 내용을 error state에 저장합니다.

/dev/stdout에 Envoy 액세스 로그를 보내면 해당 로그가 Envoy 컨테이너 로그와 혼합됩니다. awslogs와 같은 표준 Docker 로그 드라이버를 사용하여 CloudWatch Logs와 같은 로그 스토리지 및 처리 서비스로 내보낼 수 있습니다. 자세한 내용은 Amazon ECS 개발자 안내서의 [awslogs 로그 드라이버 사용](#)을 참조하세요. Envoy 액세스 로그만 내보내고 다른 Envoy 컨테이너 로그는 무시하려면 ENVOY\_LOG\_LEVEL을 off로 설정하면 됩니다. 형식 문자열 %REQ\_WITHOUT\_QUERY(X?Y):Z%를 포함하여 쿼리 문자열 없이 요청을 기록할 수 있습니다. 예제는 [ReqWithoutQuery 포맷터](#)를 참조하세요. 자세한 내용은 Envoy 설명서의 [액세스 로깅](#)을 참조하세요.

Kubernetes에서 액세스 로그 활성화

[Kubernetes용 App Mesh 컨트롤러](#)를 사용하는 경우 다음 예제와 같이 가상 노드 사양에 로깅 구성을 추가하여 액세스 로깅으로 가상 노드를 구성할 수 있습니다.

```

apiVersion: appmesh.k8s.aws/v1beta2
kind: VirtualNode
metadata:
 name: virtual-node-name
 namespace: namespace
spec:
 listeners:

```

```

- portMapping:
 port: 9080
 protocol: http
serviceDiscovery:
 dns:
 hostName: hostname
logging:
 accessLog:
 file:
 path: "/dev/stdout"

```

클러스터는 이러한 로그를 수집하려면 Fluentd와 같은 로그 전달자가 있어야 합니다. 자세한 내용은 [Fluentd를 DaemonSet으로 설정하여 CloudWatch Logs에 로그 전송](#)을 참조하세요.

또한 Envoy는 필터의 다양한 디버깅 로그를 stdout에 기록합니다. 이러한 로그는 Envoy와 App Mesh의 통신 및 서비스 간 트래픽 모두에 대한 인사이트를 얻는 데 유용합니다. ENVOY\_LOG\_LEVEL 환경 변수를 사용하여 특정 로깅 수준을 구성할 수 있습니다. 예를 들어, 다음 텍스트는 Envoy가 특정 HTTP 요청과 일치시킨 클러스터를 보여 주는 예제 디버그 로그에서 가져온 것입니다.

```

[debug][router] [source/common/router/router.cc:434] [C4][S17419808847192030829]
cluster 'cds_ingress_howto-http2-mesh_color_client_http_8080' match for URL '/ping'

```

## Firelens 및 Cloudwatch

[Firelens](#)는 Amazon ECS 및 AWS Fargate에 대한 로그를 수집하는 데 사용할 수 있는 컨테이너 로그 라우터입니다. [AWS 샘플 리포지토리](#)에서 Firelens를 사용하는 예제를 찾을 수 있습니다.

CloudWatch를 사용하여 지표 및 로깅 정보를 수집할 수 있습니다. App Mesh 설명서의 [지표 내보내기](#) 섹션에서 CloudWatch에 대한 자세한 내용을 확인할 수 있습니다.

## Envoy 지표를 사용하여 애플리케이션 모니터링

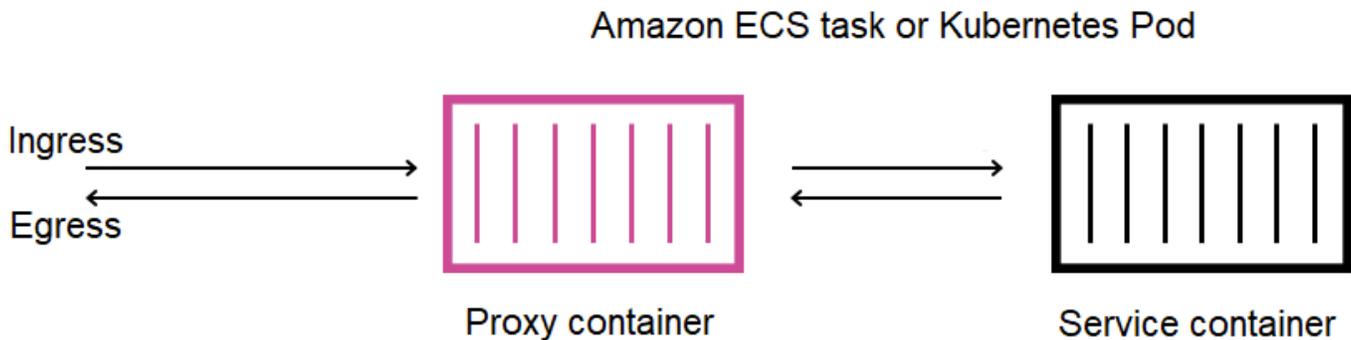
Envoy는 지표를 다음과 같은 주요 범주로 분류합니다.

- 다운스트림—프록시로 들어오는 연결 및 요청과 관련된 지표입니다.
- 업스트림—프록시에 의한 발신 연결 및 요청과 관련된 지표입니다.
- 서버—Envoy의 내부 상태를 설명하는 지표입니다. 여기에는 가동 시간 또는 할당된 메모리와 같은 지표가 포함됩니다.

App Mesh에서 프록시는 업스트림 및 다운스트림 트래픽을 가로챍니다. 예를 들어 Envoy는 클라이언트에서 받은 요청과 서비스 컨테이너의 요청을 다운스트림 트래픽으로 분류합니다. 이러한 다양한 유형의 업스트림 트래픽과 다운스트림 트래픽을 구분하기 위해 App Mesh는 서비스와 관련된 트래픽 방향에 따라 Envoy 지표를 추가적으로 분류합니다.

- 수신—서비스 컨테이너로 흐르는 연결 및 요청과 관련된 지표 및 리소스입니다.
- 송신—서비스 컨테이너에서, 궁극적으로는 Amazon ECS 태스크 또는 Kubernetes 포드에서 나가는 연결 및 요청과 관련된 지표 및 리소스입니다.

다음 그림은 프록시와 서비스 컨테이너 간의 통신을 보여 줍니다.



### 리소스 이름 지정 규칙

Envoy가 메시지를 보는 방식과 해당 리소스가 App Mesh에서 정의한 리소스에 다시 매핑되는 방식을 이해하는 데 유용합니다. App Mesh가 구성하는 기본 Envoy 리소스는 다음과 같습니다.

- 리스너—프록시가 다운스트림 연결을 수신하는 주소 및 포트입니다. 이전 그림에서 App Mesh는 Amazon ECS 태스크 또는 Kubernetes 포드로 들어오는 트래픽에 대한 수신 리스너와 서비스 컨테이너에서 나가는 트래픽에 대한 송신 리스너를 생성합니다.
- 클러스터—프록시가 연결하고 트래픽을 라우팅하는 업스트림 엔드포인트의 명명된 그룹입니다. App Mesh에서 서비스 컨테이너는 서비스가 연결할 수 있는 다른 모든 가상 노드뿐만 아니라 클러스터로 표시됩니다.
- 경로—메시에서 정의한 경로에 해당합니다. 여기에는 프록시가 요청과 일치시키는 조건과 요청이 전송되는 대상 클러스터가 포함됩니다.
- 엔드포인트 및 클러스터 로드 할당—업스트림 클러스터의 IP 주소입니다. 가상 노드의 서비스 검색 메커니즘으로 AWS Cloud Map를 사용하는 경우 App Mesh는 검색된 서비스 인스턴스를 엔드포인트 리소스로서 프록시에 전송합니다.

- 보안 암호—여기에는 암호화 키와 TLS 인증서가 포함되지만 이에 국한되지는 않습니다. 클라이언트 및 서버 인증서의 소스로 AWS Certificate Manager을 사용하는 경우 App Mesh는 퍼블릭 및 프라이빗 인증서를 보안 암호 리소스로서 프록시에 전송합니다.

App Mesh는 일관된 Envoy 리소스 이름 지정 체계를 사용하므로 메시와 다시 연관시키는 데 사용할 수 있습니다.

리스너 및 클러스터의 이름 지정 체계를 이해하는 것은 App Mesh의 Envoy 지표를 이해하는 데 중요합니다.

### 리스너 이름

리스너 이름은 다음 형식을 사용하여 지정됩니다.

```
lds_<traffic direction>_<listener IP address>_<listening port>
```

일반적으로 Envoy에 구성된 다음과 같은 리스너를 볼 수 있습니다.

- lds\_ingress\_0.0.0.0\_15000
- lds\_egress\_0.0.0.0\_15001

Kubernetes CNI 플러그인 또는 IP 테이블 규칙을 사용하면 Amazon ECS 태스크 또는 Kubernetes 포드의 트래픽이 포트 15000 및 15001로 전달됩니다. App Mesh는 수신 및 송신(발신) 트래픽을 수락하도록 이 두 리스너로 Envoy를 구성합니다. 가상 노드에 리스너가 구성되어 있지 않으면 수신 리스너가 보이지 않을 것입니다.

### 클러스터 이름

대부분의 클러스터는 다음 형식을 사용합니다.

```
cds_<traffic direction>_<mesh name>_<virtual node name>_<protocol>_<port>
```

서비스가 통신하는 가상 노드에는 자체 클러스터가 있습니다. 앞서 언급했듯이 App Mesh는 Envoy 옆에서 실행되는 서비스를 위한 클러스터를 생성하여 프록시가 수신 트래픽을 해당 서비스로 전송할 수 있도록 합니다.

예를 들어 포트 8080에서 http 트래픽을 수신하는 my-virtual-node라는 가상 노드가 있고 해당 가상 노드가 이름이 my-mesh인 메시에 있는 경우 App Mesh는 이름이 cds\_ingress\_my-mesh\_my-

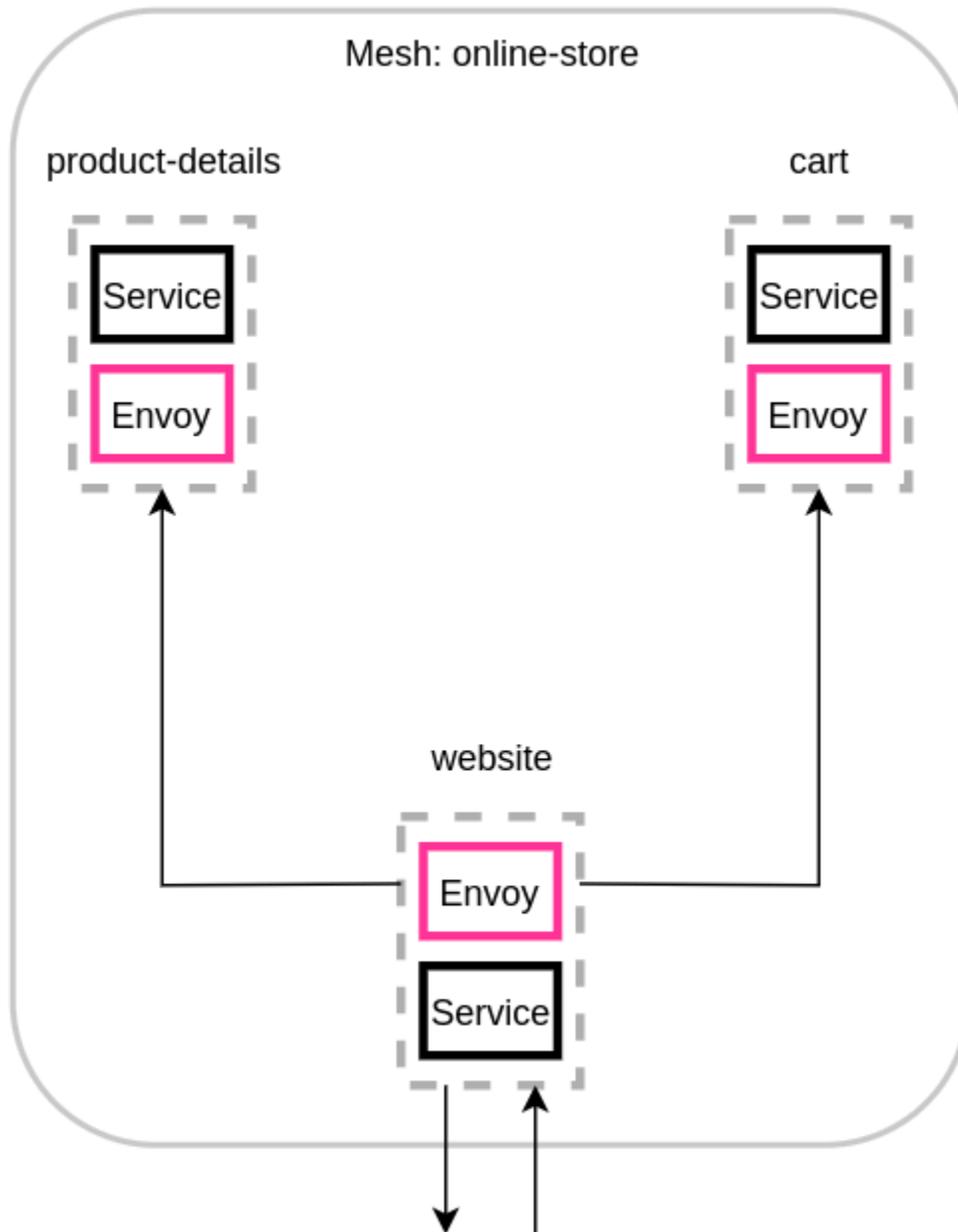
`virtual-node_http_8080`인 클러스터를 생성합니다. 이 클러스터는 `my-virtual-node`의 서비스 컨테이너로 들어오는 트래픽의 대상 역할을 합니다.

App Mesh는 다음과 같은 유형의 특수 클러스터를 추가로 생성할 수도 있습니다. 이러한 다른 클러스터가 메시에서 명시적으로 정의한 리소스와 반드시 일치하는 것은 아닙니다.

- 다른 AWS 서비스에 연결하는 데 사용되는 클러스터. 이 유형을 사용하면 기본적으로 메시가 대부분의 AWS 서비스에 도달할 수 있습니다(기본값: `cds_egress_<mesh name>_amazonaws`).
- 가상 게이트웨이의 라우팅을 수행하는 데 사용되는 클러스터. 다음은 무시해도 됩니다.
  - 단일 리스너의 경우: `cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<protocol>_<port>`
  - 여러 리스너의 경우: `cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<ingress_listener_port>_<protocol>_<port>`
- Envoy의 보안 암호 검색 서비스를 사용하여 보안 암호를 검색할 때 정의할 수 있는 엔드포인트가 되는 클러스터(예: TLS): `static_cluster_sds_unix_socket`.

## 예제 애플리케이션 지표

Envoy에서 사용할 수 있는 메트릭을 설명하기 위해 다음 샘플 애플리케이션에는 세 개의 가상 노드가 있습니다. 메시의 가상 서비스, 가상 라우터 및 경로는 Envoy의 지표에 반영되지 않으므로 무시해도 됩니다. 이 예제에서는 모든 서비스가 포트 8080에서 http 트래픽을 수신합니다.



메시에서 실행되는 Envoy 프록시 컨테이너에 환경 변수 `ENABLE_ENVOY_STATS_TAGS=1`을 추가하는 것이 좋습니다. 그러면 프록시에서 내보내는 모든 지표에 다음과 같은 지표 측정기준이 추가됩니다.

- `appmesh.mesh`
- `appmesh.virtual_node`
- `appmesh.virtual_gateway`

이러한 태그는 메시, 가상 노드 또는 가상 게이트웨이의 이름으로 설정되므로 메시의 리소스 이름을 사용하여 지표를 필터링할 수 있습니다.

## 리소스 이름

웹 사이트 가상 노드의 프록시에는 다음과 같은 리소스가 있습니다.

- 수신 및 송신 트래픽용 리스너 2개:
  - `lds_ingress_0.0.0.0_15000`
  - `lds_egress_0.0.0.0_15001`
- 두 개의 가상 노드 백엔드를 나타내는 송신 클러스터 2개:
  - `cds_egress_online-store-product-details_http_8080`
  - `cds_egress_online-store-cart_http_8080`
- 웹 사이트 서비스 컨테이너의 수신 클러스터:
  - `cds_ingress_online-store-website_http_8080`

## 예제 리스너 지표

- `listener.0.0.0.0_15000.downstream_cx_active`—Envoy에 대한 활성 수신 네트워크 연결 수
- `listener.0.0.0.0_15001.downstream_cx_active`—Envoy에 대한 활성 송신 네트워크 연결 수 애플리케이션에서 외부 서비스에 연결하는 경우도 이 수에 포함됩니다.
- `listener.0.0.0.0_15000.downstream_cx_total`—Envoy에 대한 총 수신 네트워크 연결 수
- `listener.0.0.0.0_15001.downstream_cx_total`—Envoy에 대한 총 송신 네트워크 연결 수

전체 리스너 지표 세트는 Envoy 설명서의 [통계](#)를 참조하세요.

## 예제 클러스터 지표

- `cluster_manager.active_clusters`—Envoy가 하나 이상의 연결을 설정한 클러스터의 총 수
- `cluster_manager.warming_clusters`—Envoy가 아직 연결하지 않은 클러스터의 총 수

다음 클러스터 지표는 `cluster.<cluster name>.<metric name>` 형식을 사용합니다. 이러한 지표 이름은 애플리케이션 예제에 고유하며 웹 사이트 Envoy 컨테이너에서 내보내집니다.

- `cluster.cds_egress_online-store_product-details_http_8080.upstream_cx_total`—웹 사이트와 제품 세부 정보 간의 총 연결 수
- `cluster.cds_egress_online-store_product-details_http_8080.upstream_cx_connect_fail`—웹 사이트와 제품 세부 정보 간의 실패한 총 연결 수
- `cluster.cds_egress_online-store_product-details_http_8080.health_check.failure`—웹 사이트와 제품 세부 정보 간의 실패한 총 상태 확인 수
- `cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_total`—웹 사이트와 제품 세부 정보 간에 수행된 총 요청 수
- `cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_time`—웹 사이트와 제품 세부 정보 간에 수행된 요청별 소요 시간
- `cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_2xx`—웹 사이트와 제품 세부 정보에서 받은 HTTP 2xx 응답 수.

전체 HTTP 지표 세트는 Envoy 설명서의 [통계](#)를 참조하세요.

## 관리 서버 지표

또한 Envoy는 Envoy의 관리 서버 역할을 하는 App Mesh 제어 영역에 대한 연결과 관련된 지표를 내보냅니다. 프록시가 제어 영역에서 장기간 비동기화될 때 알림을 받으려면 이러한 지표 중 일부를 모니터링하는 것이 좋습니다. 제어 영역에 대한 연결이 끊어지거나 업데이트가 실패하면 프록시가 App Mesh에서 App Mesh API를 통한 메시 변경을 비롯한 새 구성을 수신하지 못합니다.

- `control_plane.connected_state`—프록시가 App Mesh에 연결된 경우 이 지표는 1로 설정되고, 그렇지 않으면 0으로 설정됩니다.
- `*.update_rejected`—Envoy에서 거부한 구성 업데이트의 총 수. 이것은 일반적으로 사용자 구성 오류로 인한 것입니다. 예를 들어 Envoy에서 읽을 수 없는 파일에서 TLS 인증서를 읽도록 App Mesh를 구성하면 해당 인증서의 경로가 포함된 업데이트가 거부됩니다.
  - 리스너 업데이트가 거부된 경우 통계는 `listener_manager.lds.update_rejected`가 됩니다.
  - 클러스터 업데이트가 거부된 경우 통계는 `cluster_manager.cds.update_rejected`가 됩니다.

- \*.update\_success—App Mesh에서 프록시에 대해 성공적으로 수행한 구성 업데이트 수. 여기에는 새 Envoy 컨테이너가 시작될 때 전송되는 초기 구성 페이로드가 포함됩니다.
- 리스너 업데이트가 성공한 경우 통계는 listener\_manager.lds.update\_success가 됩니다.
- 클러스터 업데이트가 성공한 경우 통계는 cluster\_manager.cds.update\_success가 됩니다.

관리 서버 메트릭 세트는 Envoy 설명서의 [관리 서버](#)를 참조하세요.

## 지표 내보내기

Envoy는 자체 운영과 인바운드 및 아웃바운드 트래픽의 다양한 측정기준에 대한 많은 통계를 내보냅니다. Envoy 통계에 대한 자세한 내용은 Envoy 설명서의 [통계](#)를 참조하세요. 이러한 지표는 일반적으로 9901인 프록시의 관리 포트에 있는 /stats 엔드포인트를 통해 사용할 수 있습니다.

단일 리스너를 사용하는지 또는 여러 리스너를 사용하는지에 따라 stat 접두사가 달라집니다. 다음은 차이점을 보여 주는 몇 가지 예입니다.

### Warning

단일 리스너를 다중 리스너 기능으로 업데이트하면 다음 표에 나와 있는 업데이트된 통계 접두사로 인해 주요 변경 사항이 발생할 수 있습니다.

Envoy 이미지 1.22.2.1-prod 이상을 사용하는 것이 좋습니다. 이렇게 하면 Prometheus 엔드포인트에서 유사한 메트릭 이름을 볼 수 있습니다.

| 단일 리스너(SL)/"ingress" 리스너 접두사가 붙은 기존 통계                | 다중 리스너(ML)/"ingress.<protocol>.<port>" 리스너 접두사가 붙은 새 통계         |
|-------------------------------------------------------|-----------------------------------------------------------------|
| http.*ingress*.rds.rds_ingress_http_5555.version_text | http.*ingress.http.5555*.rds.rds_ingress_http_5555.version_text |
|                                                       | http.*ingress.http.6666*.rds.rds_ingress_http_6666.version_text |

|                                                         |                                                                                                                                            |  |
|---------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|--|
| 단일 리스너(SL)/"ingress" 리스너 접두사가 붙은 기존 통계                  | 다중 리스너(ML)/"ingress.<protocol>.<port>" 리스너 접두사가 붙은 새 통계                                                                                    |  |
|                                                         | ress_http_6666.version_text                                                                                                                |  |
| listener.0.0.0.0_15000.http.*ingress*.downstream_rq_2xx | listener.0.0.0.0_15000.http.*ingress.http.5555*.downstream_rq_2xx<br><br>listener.0.0.0.0_15000.http.*ingress.http.6666*.downstream_rq_2xx |  |
| http.*ingress*.downstream_cx_length_ms                  | http.*ingress.http.5555*.downstream_cx_length_ms<br><br>http.*ingress.http.6666*.downstream_cx_length_ms                                   |  |

통계 엔드포인트에 대한 자세한 내용은 Envoy 설명서의 [통계 엔드포인트](#)를 참조하세요. 관리 인터페이스에 대한 자세한 내용은 [Envoy 프록시 관리 인터페이스 활성화](#) 섹션을 참조하세요.

## Amazon EKS를 포함하는 App Mesh용 Prometheus

Prometheus는 오픈 소스 모니터링 및 알림 도구 키트입니다. 해당 기능 중 하나는 다른 시스템에서 사용할 수 있는 지표를 내보내는 형식을 지정하는 것입니다. Prometheus에 대한 자세한 내용은 Prometheus 설명서의 [개요](#)를 참조하세요. Envoy는 파라미터 /stats?format=prometheus를 전달하여 통계 엔드포인트를 통해 지표를 내보낼 수 있습니다.

Envoy 이미지 빌드 v1.22.2.1-prod를 사용하는 고객의 경우 수신 리스너별 통계를 나타내는 두 가지 추가 측정기준이 있습니다.

- appmesh.listener\_protocol

- `appmesh.listener_port`

아래는 Prometheus의 기존 통계와 새 통계를 비교한 것입니다.

- “수신” 리스너 접두사가 붙은 기존 통계

```
envoy_http_downstream_rq_xx{appmesh_mesh="multiple-listeners-mesh",appmesh_virtual_node="foodteller-vn",envoy_response_code_class="2",envoy_http_conn_manager_prefix="ingress"} 931433
```

- "ingress.<protocol>.<port>" + Appmesh Envoy Image v1.22.2.1-prod 이상이 있는 새 통계

```
envoy_http_downstream_rq_xx{appmesh_mesh="multiple-listeners-mesh",appmesh_virtual_node="foodteller-vn",envoy_response_code_class="2",appmesh_listener_protocol="http",appmesh_listener_port="55520"}
```

- "ingress.<protocol>.<port>" + 사용자 지정 Envoy Imagebuild가 있는 새 통계

```
envoy_http_http_5555_downstream_rq_xx{appmesh_mesh="multiple-listeners-mesh",appmesh_virtual_node="foodteller-vn",envoy_response_code_class="2",envoy_http_conn_manager_prefix="ingress"} 15983
```

여러 리스너의 경우 `cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<ingress_listener_port>_<protocol>_<port>` 특수 클러스터는 리스너별로 다릅니다.

- “수신” 리스너 접두사가 붙은 기존 통계

```
envoy_cluster_assignment_stale{appmesh_mesh="multiple-listeners-mesh",appmesh_virtual_gateway="telligateway-vg",Mesh="multiple-listeners-mesh",VirtualGateway="telligateway-vg",envoy_cluster_name="cds_ingress_multiple-listeners-mesh_telligateway-vg_self_redirect_http_15001"} 0
```

- "ingress.<protocol>.<port>"가 있는 새 통계

```
envoy_cluster_assignment_stale{appmesh_mesh="multiple-listeners-mesh",appmesh_virtual_gateway="telligateway-vg",envoy_cluster_name="cds_ingress_multiple-listeners-mesh_telligateway-vg_self_redirect_1111_http_15001"} 0
```

```
envoy_cluster_assignment_stale{appmesh_mesh="multiple-
listeners-mesh",appmesh_virtual_gateway="telligateway-
vg",envoy_cluster_name="cds_ingress_multiple-listeners-mesh_telligateway-
vg_self_redirect_2222_http_15001"} 0
```

## Prometheus 설치

1. Helm에 EKS 리포지토리를 추가합니다.

```
helm repo add eks https://aws.github.io/eks-charts
```

2. App Mesh Prometheus 설치

```
helm upgrade -i appmesh-prometheus eks/appmesh-prometheus \
--namespace appmesh-system
```

## Prometheus 예제

다음은 Prometheus용 PersistentVolumeClaim 영구 스토리지를 생성하는 예제입니다.

```
helm upgrade -i appmesh-prometheus eks/appmesh-prometheus \
--namespace appmesh-system \
--set retention=12h \
--set persistentVolumeClaim.claimName=prometheus
```

## Prometheus 사용 방법 살펴보기

- [EKS를 사용한 App Mesh — 관찰성: Prometheus](#)

Prometheus 및 Amazon EKS를 포함하는 Prometheus에 대해 자세히 알아보려면 다음을 참조하세요.

- [Prometheus 설명서](#)
- EKS - [Prometheus의 제어 영역 지표](#)

## App Mesh용 CloudWatch

### Envoy 통계를 Amazon EKS에서 CloudWatch로 내보내기

클러스터에 CloudWatch Agent를 설치하고 프록시에서 지표의 하위 세트를 수집하도록 구성할 수 있습니다. Amazon EKS 클러스터가 아직 없는 경우 GitHub에서 [살펴보기: Amazon EKS를 포함하는 App Mesh](#)의 단계에 따라 클러스터를 생성할 수 있습니다. 동일한 안내를 따라 클러스터에 샘플 애플리케이션을 설치할 수 있습니다.

클러스터에 대해 적절한 IAM 권한을 설정하고 에이전트를 설치하려면 [Prometheus 지표 수집과 함께 CloudWatch Agent 설치](#)의 단계를 따르세요. 기본 설치에는 Envoy 통계의 유용한 하위 세트를 가져오는 Prometheus 수집 구성이 포함되어 있습니다. 자세한 내용은 [App Mesh용 Prometheus 지표](#)를 참조하세요.

에이전트가 수집하는 지표를 표시하도록 구성된 App Mesh 사용자 지정 CloudWatch 대시보드를 생성하려면 [Prometheus 지표 보기](#) 자습서의 단계를 따르세요. 트래픽이 App Mesh 애플리케이션으로 유입되면 그래프가 해당 지표로 채워지기 시작합니다.

### CloudWatch의 지표 필터링

App Mesh [지표 확장](#)은 메시에서 정의한 리소스의 동작에 대한 인사이트를 제공하는 유용한 지표의 하위 세트를 제공합니다. CloudWatch 에이전트는 Prometheus 지표 수집을 지원하므로, Envoy에서 가져와서 CloudWatch로 전송할 지표를 선택하는 수집 구성을 제공할 수 있습니다.

Prometheus를 사용하여 지표를 스크랩하는 예제는 [지표 확장](#) 안내에서 확인할 수 있습니다.

### CloudWatch 예제

[AWS 샘플 리포지토리](#)에서 CloudWatch의 샘플 구성을 찾을 수 있습니다.

### CloudWatch 사용 방법 살펴보기

- [App Mesh 워크샵의 모니터링 및 로깅 기능 추가](#)
- [EKS를 사용한 App Mesh — 관찰성: CloudWatch](#)
- [ECS의 App Mesh 지표 확장 사용](#)

## App Mesh의 지표 확장

Envoy는 수백 개의 지표를 몇 가지 다른 측정기준으로 세분화하여 생성합니다. 지표가 App Mesh와 연관되는 방식은 간단하지 않습니다. 가상 서비스의 경우 어떤 가상 서비스가 특정 가상 노드나 가상 게이트웨이와 통신하고 있는지 확인할 수 있는 메커니즘이 없습니다.

App Mesh 지표 확장은 메시에서 실행되는 Envoy 프록시를 개선합니다. 이러한 개선을 통해 프록시는 사용자가 정의한 리소스를 인식하는 추가 지표를 내보낼 수 있습니다. 이 소규모 추가 지표 세트는 App Mesh에서 정의한 리소스의 동작에 대한 더 나은 인사이트를 얻는 데 도움이 됩니다.

App Mesh 지표 확장을 활성화하려면 환경 변수 `APPMESH_METRIC_EXTENSION_VERSION`을 1로 설정합니다.

```
APPMESH_METRIC_EXTENSION_VERSION=1
```

Envoy 구성 변수에 대한 자세한 내용은 [Envoy 구성 변수](#) 섹션을 참조하세요.

인바운드 트래픽 관련 지표

- **ActiveConnectionCount**

- `envoy.appmesh.ActiveConnectionCount` — 활성 TCP 연결의 수.
- 측정기준 — Mesh, VirtualNode, VirtualGateway

- **NewConnectionCount**

- `envoy.appmesh.NewConnectionCount` — 총 TCP 연결 수.
- 측정기준 — Mesh, VirtualNode, VirtualGateway

- **ProcessedBytes**

- `envoy.appmesh.ProcessedBytes` — 다운스트림 클라이언트에서 전송 및 수신한 총 TCP 바이트 수.
- 측정기준 — Mesh, VirtualNode, VirtualGateway

- **RequestCount**

- `envoy.appmesh.RequestCount` - 처리된 HTTP 요청 수.
- 측정기준 — Mesh, VirtualNode, VirtualGateway

- **GrpcRequestCount**

- `envoy.appmesh.GrpcRequestCount` - 처리된 gPRC 요청 수.
- 측정기준 — Mesh, VirtualNode, VirtualGateway

아웃바운드 트래픽과 관련된 지표

가상 노드 또는 가상 게이트웨이 중에서 아웃바운드 지표를 가져온 소스에 따라 다양한 측정기준이 표시됩니다.

- **TargetProcessedBytes**

- `envoy.appmesh.TargetProcessedBytes` — Envoy의 대상 업스트림에서 전송 및 수신된 총 TCP 바이트 수입입니다.
- 측정기준:
  - 가상 노드 측정기준 — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
  - 가상 게이트웨이 측정기준 — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode

- **HTTPCode\_Target\_2XX\_Count**

- `envoy.appmesh.HTTPCode_Target_2XX_Count` — Envoy의 대상 업스트림에 대한 HTTP 요청 중에서 2xx HTTP 응답을 받은 요청 수입입니다.
- 측정기준:
  - 가상 노드 측정기준 — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
  - 가상 게이트웨이 측정기준 — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode

- **HTTPCode\_Target\_3XX\_Count**

- `envoy.appmesh.HTTPCode_Target_3XX_Count` — Envoy의 대상 업스트림에 대한 HTTP 요청 중에서 3xx HTTP 응답을 받은 요청 수입입니다.
- 측정기준:
  - 가상 노드 측정기준 — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
  - 가상 게이트웨이 측정기준 — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode

- **HTTPCode\_Target\_4XX\_Count**

- `envoy.appmesh.HTTPCode_Target_4XX_Count` — Envoy의 대상 업스트림에 대한 HTTP 요청 중에서 4xx HTTP 응답을 받은 요청 수입입니다.
- 측정기준:
  - 가상 노드 측정기준 — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
  - 가상 게이트웨이 측정기준 — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode

- **HTTPCode\_Target\_5XX\_Count**

- `envoy.appmesh.HTTPCode_Target_5XX_Count` — Envoy의 대상 업스트림에 대한 HTTP 요청 중에서 5xx HTTP 응답을 받은 요청 수입입니다.
- 측정기준:
  - 가상 노드 측정기준 — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
  - 가상 게이트웨이 측정기준 — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode

- `envoy.appmesh.RequestCountPerTarget` — Envoy의 대상 업스트림으로 전송된 요청 수입니다.
- 측정기준:
  - 가상 노드 측정기준 — `Mesh`, `VirtualNode`, `TargetVirtualService`, `TargetVirtualNode`
  - 가상 게이트웨이 측정기준 — `Mesh`, `VirtualGateway`, `TargetVirtualService`, `TargetVirtualNode`
- **TargetResponseTime**
  - `envoy.appmesh.TargetResponseTime` — Envoy의 대상 업스트림에 요청을 보낸 시점부터 전체 응답을 받을 때까지 소요된 시간입니다.
  - 측정기준:
    - 가상 노드 측정기준 — `Mesh`, `VirtualNode`, `TargetVirtualService`, `TargetVirtualNode`
    - 가상 게이트웨이 측정기준 — `Mesh`, `VirtualGateway`, `TargetVirtualService`, `TargetVirtualNode`

## App Mesh용 Datadog

Datadog은 클라우드 애플리케이션의 엔드투엔드 모니터링, 지표 및 로깅을 위한 모니터링 및 보안 애플리케이션입니다. Datadog을 사용하면 인프라, 애플리케이션 및 타사 애플리케이션을 완벽하게 관찰할 수 있습니다.

### Datadog 설치

- EKS - EKS로 Datadog을 설정하려면 [Datadog 설명서](#)의 다음 단계를 따르세요.
- ECS EC2 - ECS EC2로 Datadog을 설정하려면 [Datadog 설명서](#)의 다음 단계를 따르세요.

Datadog에 대해 자세히 알아보려면 다음을 참조하세요.

- [Datadog 설명서](#)

## 추적

### Important

추적을 완전히 구현하려면 애플리케이션을 업데이트해야 합니다.

선택한 서비스에서 사용 가능한 모든 데이터를 보려면 해당 라이브러리를 사용하여 애플리케이션을 계속해야 합니다.

## AWS X-Ray로 App Mesh 모니터링

AWS X-Ray는 애플리케이션이 처리하는 요청에서 수집된 데이터를 보고, 필터링하고, 인사이트를 얻을 수 있는 도구를 제공하는 서비스입니다. 이러한 인사이트는 앱을 최적화할 수 있는 문제와 기회를 식별하는 데 도움이 됩니다. 요청과 응답, 애플리케이션이 다른 AWS 서비스에 대해 수행하는 다운스트림 호출에 대한 자세한 정보를 볼 수 있습니다.

X-Ray는 App Mesh와 통합되어 Envoy 마이크로서비스를 관리합니다. Envoy의 추적 데이터는 컨테이너에서 실행되는 X-Ray 대몬(daemon)으로 전송됩니다.

해당 언어에 맞는 [SDK](#) 안내서를 사용하여 애플리케이션 코드에 X-Ray를 구현합니다.

### App Mesh를 통해 X-Ray 추적 활성화

- 서비스 유형에 따라 다음을 수행합니다.
  - ECS - Envoy 프록시 컨테이너 정의에서 `ENABLE_ENVOY_XRAY_TRACING` 환경 변수를 1으로 설정하고 `XRAY_DAEMON_PORT` 환경 변수를 2000으로 설정합니다.
  - EKS - App Mesh 컨트롤러 구성에서 `--set tracing.enabled=true` 및 `--set tracing.provider=x-ray`를 포함합니다.
- X-Ray 컨테이너에서 포트 2000을 노출하고 사용자 1337 권한으로 실행합니다.

### X-Ray 예제

#### Amazon ECS용 Envoy 컨테이너 정의

```
{
 "name": "envoy",
 "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.15.1.0-prod",
 "essential": true,
 "environment": [
 {
 "name": "APPMESH_VIRTUAL_NODE_NAME",
 "value": "mesh/myMesh/virtualNode/myNode"
 },
 {
 "name": "ENABLE_ENVOY_XRAY_TRACING",
 "value": "1"
 }
]
}
```

```

 }
],
 "healthCheck": {
 "command": [
 "CMD-SHELL",
 "curl -s http://localhost:9901/server_info | cut -d' ' -f3 | grep -q live"
],
 "startPeriod": 10,
 "interval": 5,
 "timeout": 2,
 "retries": 3
 }
}

```

## Amazon EKS용 App Mesh 컨트롤러 업데이트

```

helm upgrade -i appmesh-controller eks/appmesh-controller \
--namespace appmesh-system \
--set region=${AWS_REGION} \
--set serviceAccount.create=false \
--set serviceAccount.name=appmesh-controller \
--set tracing.enabled=true \
--set tracing.provider=x-ray

```

## X-Ray 사용 방법 살펴보기

- [AWS X-Ray를 사용한 모니터링](#)
- [Amazon EKS의 App Mesh - 관찰성: X-Ray](#)
- AWS App Mesh [워크샵](#)의 [X-Ray를 사용한 분산 추적](#)

AWS X-Ray에 대해 자세히 알아보려면 다음을 참조하세요.

- [AWS X-Ray 설명서](#)

## App Mesh를 사용한 AWS X-Ray 문제 해결

- [내 애플리케이션의 AWS X-Ray 추적을 볼 수 없습니다.](#)

## Amazon EKS를 사용하는 App Mesh용 Jaeger

Jaeger는 오픈 소스의 엔드투엔드 분산 추적 시스템입니다. 네트워크를 프로파일링하고 모니터링하는 데 사용할 수 있습니다. Jaeger는 복잡한 클라우드 네이티브 애플리케이션 문제를 해결하는 데도 도움을 줄 수 있습니다.

Jaeger를 애플리케이션 코드로 구현하려면 Jaeger 설명서 [추적 라이브러리](#)에서 해당 언어에 맞는 가이드를 찾을 수 있습니다.

### Helm을 사용하여 Jaeger 설치

1. Helm에 EKS 리포지토리를 추가합니다.

```
helm repo add eks https://aws.github.io/eks-charts
```

2. App Mesh Jaeger 설치

```
helm upgrade -i appmesh-jaeger eks/appmesh-jaeger \
--namespace appmesh-system
```

### Jaeger 예제

다음은 Jaeger용 PersistentVolumeClaim 영구 스토리지를 생성하는 예제입니다.

```
helm upgrade -i appmesh-controller eks/appmesh-controller \
--namespace appmesh-system \
--set tracing.enabled=true \
--set tracing.provider=jaeger \
--set tracing.address=appmesh-jaeger.appmesh-system \
--set tracing.port=9411
```

### Jaeger 사용 방법 살펴보기

- [EKS를 사용한 App Mesh — 관찰성: Jaeger](#)

Jaeger에 대한 자세한 내용은 다음을 참조하세요.

- [Jaeger 설명서](#)

## 추적용 Datadog

Datadog는 지표뿐만 아니라 추적에도 사용할 수 있습니다. 자세한 내용 및 설치 지침은 [Datadog 설명서](#)에서 애플리케이션 언어별 안내서를 참조하세요.

# App Mesh 도구

App Mesh는 고객이 다음과 같은 도구를 사용하여 API와 간접적으로 상호 작용할 수 있는 기능을 제공합니다.

- AWS CloudFormation
- AWS Cloud Development Kit (AWS CDK)
- Kubernetes용 App Mesh 컨트롤러
- Terraform

## 앱 메시 및 AWS CloudFormation

AWS CloudFormation 애플리케이션에 필요한 모든 리소스가 포함된 템플릿을 만든 다음 리소스를 구성하고 AWS CloudFormation 프로비저닝할 수 있는 서비스입니다. 또한 모든 종속성을 구성하므로 리소스 관리보다는 애플리케이션에 더 집중할 수 있습니다.

App Mesh와 AWS CloudFormation 함께 사용하는 방법에 대한 자세한 내용 및 예제는 [AWS CloudFormation 설명서를](#) 참조하십시오.

## 앱 메시 및 AWS CDK

AWS CDK 코드를 사용하여 클라우드 인프라를 정의하고 이를 프로비저닝하는 AWS CloudFormation 데 사용하는 개발 프레임워크입니다. AWS CDK Python, Java TypeScript JavaScript, C#/.Net 등 여러 프로그래밍 언어를 지원합니다.

App AWS CDK Mesh와 함께 사용하는 방법에 대한 자세한 내용은 [AWS CDK 설명서를](#) 참조하십시오.

## Kubernetes용 App Mesh 컨트롤러

Kubernetes용 App Mesh 컨트롤러를 사용하면 Kubernetes 클러스터의 App Mesh 리소스를 관리하고 사이드카를 파드에 삽입할 수 있습니다. 이 컨트롤러는 특히 Amazon EKS와 함께 사용하기 위한 것으로, Kubernetes 고유의 방식으로 리소스를 관리할 수 있게 해줍니다.

App Mesh 컨트롤러에 대한 자세한 내용은 [App Mesh 컨트롤러 설명서](#)를 참조하세요.

## App Mesh 및 Terraform

[Terraform](#)은 코드 소프트웨어 도구로서의 오픈 소스 인프라입니다. Terraform은 CLI를 사용하여 클라우드 서비스를 관리하고 선언적 구성 파일을 사용하여 API와 상호 작용할 수 있습니다.

Terraform에서 App Mesh를 사용하는 방법에 대한 자세한 내용은 [Terraform 설명서](#)를 확인하세요.

## 공유 메시 작업

AWS Resource Access Manager 서비스를 사용하여 여러 AWS 계정에서 App Mesh 메시지를 공유할 수 있습니다. 공유 메시지를 사용하면 서로 다른 AWS 계정에서 만든 리소스가 동일한 메시지에서 서로 통신할 수 있습니다.

AWS 계정은 메시 리소스 소유자, 메시 소비자 또는 둘 다일 수 있습니다. 소비자는 자신의 계정과 공유되는 메시지에서 리소스를 생성할 수 있습니다. 소유자는 계정이 소유한 모든 메시지에서 리소스를 생성할 수 있습니다. 메시 소유자는 다음 유형의 메시 소비자와 메시지를 공유할 수 있습니다.

- 조직 내부 또는 외부의 특정 AWS 계정 AWS Organizations
- 조직 내 조직 단위: AWS Organizations
- 조직 전체가 AWS Organizations

메시 공유 방법에 GitHub 대한 자세한 내용은 [계정 간 메시 안내](#)를 참조하세요. end-to-end

## 메시 공유 권한 부여

계정 간에 메시지를 공유하는 경우 메시지를 공유하는 IAM 보안 주체에 필요한 권한과 메시 자체에 필요한 리소스 수준 권한이 있습니다.

### 메시 공유 권한 부여

IAM 보안 주체가 메시지를 공유하려면 최소 권한 세트가 필요합니다. IAM 보안 `AWSAppMeshFullAccess` 주체가 공유 메시지를 공유하고 사용하는 데 필요한 권한을 갖도록 하려면 `AWSResourceAccessManagerFullAccess` 관리형 IAM 정책을 사용하는 것이 좋습니다.

사용자 지정 IAM 정책을 사용하는 경우,, 및 조치가 필요합니다. `appmesh:PutMeshPolicy` `appmesh:GetMeshPolicy` `appmesh>DeleteMeshPolicy` 이는 권한 전용 IAM 작업입니다. IAM 보안 주체에 이러한 권한이 부여되지 않은 경우 서비스를 사용하여 메시지를 공유하려고 하면 오류가 발생합니다. AWS RAM

AWS Resource Access Manager 서비스가 IAM을 사용하는 방식에 대한 자세한 내용은 사용 설명서의 IAM [AWS RAM 사용 방법](#)을 참조하십시오. AWS Resource Access Manager

### 메시에 권한 부여

공유 메시에는 다음과 같은 권한이 있습니다.

- 소비자는 계정과 공유되는 메시의 모든 리소스를 나열하고 설명할 수 있습니다.
- 소유자는 계정이 소유한 모든 메시의 모든 리소스를 나열하고 설명할 수 있습니다.
- 소유자와 소비자는 해당 계정이 생성한 메시의 리소스를 수정할 수 있지만 다른 계정이 생성한 리소스는 수정할 수 없습니다.
- 소비자는 해당 계정이 생성한 메시의 모든 리소스를 삭제할 수 있습니다.
- 소유자는 임의의 계정이 생성한 메시의 모든 리소스를 삭제할 수 있습니다.
- 소유자의 리소스는 동일한 계정의 다른 리소스만 참조할 수 있습니다. 예를 들어, 가상 노드는 가상 노드의 소유자와 동일한 계정에 있는 AWS Certificate Manager 인증서 AWS Cloud Map 또는 참조만 할 수 있습니다.
- 소유자와 소비자는 Envoy 프록시를 계정이 소유한 가상 노드로서 App Mesh에 연결할 수 있습니다.
- 소유자는 가상 게이트웨이와 가상 게이트웨이 경로를 생성할 수 있습니다.
- 소유자와 소비자는 계정이 생성한 메시의 리소스에 대해 태그를 나열하고 태그를 지정하거나 태그를 해제할 수 있습니다. 계정이 생성하지 않은 메시의 리소스의 경우에는 태그를 나열하고 리소스에 태그를 지정하거나 태그를 해제할 수 없습니다.

공유 메시는 정책 기반 인증을 사용합니다. 메시는 고정된 권한 집합으로 공유됩니다. 이러한 권한은 리소스 정책에 추가하도록 선택되며 IAM 사용자/역할을 기준으로 선택적 IAM 정책을 선택할 수도 있습니다. 이러한 정책에서 허용된 권한과 거부된 덜 명시적인 권한의 교집합에 따라 메시에 대한 보안 주체의 액세스 권한이 결정됩니다.

메시를 공유할 때 AWS Resource Access Manager 서비스는 다음과 같은 권한을 제공하는 App Mesh 라는 관리형 정책을 생성하고 이를 App AWSRAMDefaultPermissionAppMesh Mesh와 연결합니다.

- `appmesh:CreateVirtualNode`
- `appmesh:CreateVirtualRouter`
- `appmesh:CreateRoute`
- `appmesh:CreateVirtualService`
- `appmesh:UpdateVirtualNode`
- `appmesh:UpdateVirtualRouter`
- `appmesh:UpdateRoute`
- `appmesh:UpdateVirtualService`
- `appmesh:ListVirtualNodes`
- `appmesh:ListVirtualRouters`

- `appmesh:ListRoutes`
- `appmesh:ListVirtualServices`
- `appmesh:DescribeMesh`
- `appmesh:DescribeVirtualNode`
- `appmesh:DescribeVirtualRouter`
- `appmesh:DescribeRoute`
- `appmesh:DescribeVirtualService`
- `appmesh>DeleteVirtualNode`
- `appmesh>DeleteVirtualRouter`
- `appmesh>DeleteRoute`
- `appmesh>DeleteVirtualService`
- `appmesh:TagResource`
- `appmesh:UntagResource`

## 메시 공유를 위한 사전 조건

메시를 공유하려면 다음 사전 조건을 충족해야 합니다.

- AWS 계정에서 메시를 소유해야 합니다. 나와 공유된 메시는 공유할 수 없습니다.
- AWS Organizations의 조직 또는 조직 단위와 메시를 공유하려면, AWS Organizations와의 공유를 활성화해야 합니다. 자세한 내용은 AWS RAM 사용 설명서의 [AWS Organizations를 사용하여 공유 사용](#)을 참조하세요.
- 서비스는 서로 통신하려는 메시 리소스가 포함된 계정 간에 연결이 공유되는 Amazon VPC에 배포되어야 합니다. 네트워크 연결을 공유하는 한 가지 방법은 메시에서 사용하려는 모든 서비스를 공유 서브넷에 배포하는 것입니다. 자세한 내용 및 제한 사항은 [서브넷 공유](#)를 참조하세요.
- 서비스는 DNS 또는 를 통해 검색할 수 있어야 합니다. AWS Cloud Map서비스 검색에 대한 자세한 내용은 [가상 노드](#)를 참조하세요.

## 관련 서비스

메시 공유는 () 와 통합됩니다. AWS Resource Access Manager AWS RAM AWS RAM 모든 AWS 계정과 또는 계정을 통해 AWS 리소스를 공유할 수 있는 서비스입니다. AWS Organizations를 사용하면 리소스 공유를 생성하여 소유한 리소스를 공유할 수 있습니다. AWS RAM리소스 공유는 공유할 리소

스와 공유 대상 소비자를 지정합니다. 소비자는 개별 AWS 계정, 조직 단위 또는 전체 조직일 수 AWS Organizations 있습니다.

에 대한 AWS RAM 자세한 내용은 [AWS RAM 사용 설명서](#)를 참조하십시오.

## 메시 공유

메시를 공유하면 다른 계정에서 생성한 메시 리소스가 동일한 메시에서 서로 통신할 수 있습니다. 소유한 메시만 공유할 수 있습니다. 메시지를 공유하려면 리소스 공유에 추가해야 합니다. 리소스 공유는 AWS 계정 간에 AWS RAM 리소스를 공유할 수 있는 리소스입니다. 리소스 공유는 공유할 리소스와 공유 대상 소비자를 지정합니다. Amazon Linux 콘솔을 사용하여 메시지를 공유하면 기존 리소스 공유에 추가합니다. 새 리소스 공유에 메시지를 추가하려면 [AWS RAM 콘솔](#)을 사용하여 리소스 공유를 생성합니다.

조직의 일원이고 조직 내 공유가 활성화된 경우 조직의 소비자에게 공유 메시에 대한 액세스 권한을 자동으로 부여할 수 있습니다. AWS Organizations 그렇지 않으면 소비자는 리소스 공유에 가입하라는 초대장을 받고 초대를 수락한 후 공유된 메시에 대한 액세스 권한을 받습니다.

AWS RAM 콘솔이나 를 사용하여 소유한 메시지를 공유할 수 AWS CLI 있습니다.

AWS RAM 콘솔을 사용하여 소유한 메시지를 공유하려면

자세한 지침은 AWS RAM 사용 설명서의 [리소스 공유 생성](#)을 참조하세요. 리소스 유형을 선택할 때 메시지를 선택한 다음, 공유하려는 메시지를 선택합니다. 목록에 메시가 없는 경우 먼저 메시지를 생성합니다. 자세한 정보는 [서비스 메시 생성](#)을 참조하세요.

를 사용하여 소유한 메시지를 공유하려면 AWS CLI

[create-resource-share](#) 명령을 사용합니다. --resource-arns 옵션에서 공유하려는 메시의 ARN을 지정합니다.

## 공유 메시의 공유 해제

메시의 공유를 해제하면 App Mesh는 이전 메시 사용자가 메시에 더 이상 액세스할 수 없도록 합니다. 하지만 App Mesh는 소비자가 생성한 리소스를 삭제하지 않습니다. 메시가 공유 해제된 후에는 메시 소유자만 리소스를 액세스하고 삭제할 수 있습니다. App Mesh는 메시가 공유 해제된 후 메시의 리소스를 소유한 계정이 구성 정보를 받지 못하도록 합니다. 또한 App Mesh는 메시에 리소스가 있는 다른 계정이 공유 해제된 메시로부터 구성 정보를 받지 못하도록 합니다. 메시 소유자만 공유된 메시의 공유를 해제할 수 있습니다.

소유하고 있는 공유된 메시지를 공유 해제하려면 리소스 공유에서 제거해야 합니다. AWS RAM 콘솔이나 [AWS CLI](#)를 사용하여 이 작업을 수행할 수 있습니다.

콘솔을 사용하여 소유한 공유 메시지의 AWS RAM 공유를 해제하려면

자세한 지침에 대해서는 [AWS RAM 사용 설명서](#)에서 [리소스 공유 업데이트](#)를 참조하세요.

를 사용하여 소유한 공유 메시지의 공유를 해제하려면 AWS CLI

[disassociate-resource-share](#) 명령을 사용합니다.

## 공유 메시지 식별

소유자와 소비자는 Amazon Linux 콘솔을 사용하여 공유 메시지와 메시지 리소스를 식별할 수 있습니다. AWS CLI

Amazon Linux 콘솔을 사용하여 공유 메시지를 식별하려면

1. <https://console.aws.amazon.com/appmesh/>에서 App Mesh 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 메시지를 선택합니다. 각 메시지의 메시지 소유자 계정 ID는 메시지 소유자 열에 나열되어 있습니다.
3. 왼쪽 탐색 창에서 가상 서비스, 가상 라우터 또는 가상 노드를 선택합니다. 각 리소스의 메시지 소유자 및 리소스 소유자의 계정 ID가 표시됩니다.

다음을 사용하여 공유 메시지를 식별하려면 AWS CLI

`aws appmesh list resource` 명령(예: `aws appmesh list-meshes`)을 사용합니다. 이 명령은 소유하고 있는 메시지 및 자신과 공유된 메시지를 반환합니다. `meshOwner` 속성에는 메시지의 AWS 계정 ID가 표시되고 `resourceOwner` 속성에는 리소스 소유자의 AWS 계정 ID가 표시됩니다. 메시지 리소스에 대해 실행된 모든 명령은 이러한 속성을 반환합니다.

공유 메시지에 연결하는 사용자 정의 태그는 AWS 계정만 사용할 수 있습니다. 메시지를 공유하는 다른 계정은 사용할 수 없습니다. 다른 계정의 메시지에 대해 `aws appmesh list-tags-for-resource` 명령을 실행하려고 하면 액세스가 거부됩니다.

## 결제 및 측정

메시지 공유에 대한 추가 비용은 없습니다.

## 인스턴스 할당량

메시에 대한 모든 할당량은 메시에 리소스를 생성한 사용자에게 관계없이 공유 메시에도 적용됩니다. 메시 소유자만 할당량 증가를 요청할 수 있습니다. 자세한 정보는 [App Mesh 서비스 할당량](#)을 참조하세요. AWS Resource Access Manager 서비스에도 할당량이 있습니다. 자세한 내용은 [서비스 할당량](#)을 참조하세요.

# App Mesh와 통합된 AWS 서비스

App Mesh를 다른 AWS 서비스와 함께 사용하여 비즈니스 문제에 유용한 추가 솔루션을 제공할 수 있습니다. 이 주제에서는 App Mesh를 사용하여 기능을 추가하는 서비스 또는 App Mesh에서 작업을 수행하는 데 사용하는 서비스에 대해 살펴봅니다.

## 목차

- [AWS CloudFormation을 사용하여 App Mesh 리소스 생성](#)
- [AWS Outpost의 App Mesh](#)

## AWS CloudFormation을 사용하여 App Mesh 리소스 생성

App Mesh는 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 AWS 리소스를 모델링하고 설정하는 데 도움이 되는 서비스인 AWS CloudFormation과 통합됩니다. 필요한 모든 AWS 리소스(예: App Mesh 메시)를 설명하는 템플릿을 생성하면 AWS CloudFormation이 해당 리소스의 프로비저닝과 구성을 담당합니다.

AWS CloudFormation을 사용할 때 템플릿을 재사용하여 App Mesh 리소스를 일관되고 반복적으로 설정할 수 있습니다. 리소스를 한 번만 설명하고 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝할 수 있습니다.

## App Mesh 및 AWS CloudFormation 템플릿

App Mesh 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 [AWS CloudFormation 템플릿](#)을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이 템플릿은 AWS CloudFormation 스택에서 프로비저닝할 리소스에 대해 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 AWS CloudFormation Designer를 사용하면 AWS CloudFormation 템플릿을 시작하는 데 도움이 됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 [AWS CloudFormation Designer이란 무엇입니까?](#)를 참조하세요.

App Mesh는 AWS CloudFormation의 메시, 경로, 가상 노드, 가상 라우터, 가상 서비스 생성을 지원합니다. App Mesh 리소스에 대한 JSON 및 YAML 템플릿의 예를 비롯한 자세한 내용은 AWS CloudFormation 사용 설명서의 [App Mesh 리소스 유형 참조](#)를 참조하세요.

## AWS CloudFormation에 대해 자세히 알아보기

AWS CloudFormation에 대한 자세한 내용은 다음 리소스를 참조하세요.

- [AWS CloudFormation](#)
- [AWS CloudFormation 사용 설명서](#)
- [AWS CloudFormation 명령줄 인터페이스 사용 설명서](#)

## AWS Outpost의 App Mesh

AWS Outpost는 온프레미스 시설의 기본 AWS 서비스, 인프라 및 운영 모델을 지원합니다. AWS Outpost 환경에서는 AWS 클라우드에서 사용하는 것과 동일한 AWS API, 도구 및 인프라를 사용할 수 있습니다. AWS Outpost의 App Mesh는 온프레미스 데이터 및 애플리케이션과 매우 가까운 거리에서 실행해야 하는 대기 시간이 짧은 워크로드에 이상적입니다. AWS Outpost에 대한 자세한 내용은 [AWS Outpost 사용 설명서](#)를 참조하세요.

### 필수 조건

다음은 AWS Outpost에서 App Mesh를 사용하기 위한 필수 조건입니다.

- 온프레미스 데이터 센터에 Outpost가 설치 및 구성되어 있어야 합니다.
- Outpost와 AWS 리전 간에 안정적인 네트워크 연결이 있어야 합니다.
- Outpost를 위한 AWS 리전은 AWS App Mesh를 지원해야 합니다. 지원되는 리전 목록은 AWS 일반 참조의 [AWS App Mesh 엔드포인트 및 할당량](#)을 참조하세요.

### 제한 사항

AWS Outpost에서 App Mesh를 사용할 때의 제한 사항은 다음과 같습니다.

- AWS Identity and Access Management, Application Load Balancer, Network Load Balancer, Classic Load Balancer 및 Amazon Route 53는 Outpost가 아닌 AWS 리전에서 실행됩니다. 이렇게 하면 이러한 서비스와 컨테이너 간의 지연 시간이 증가합니다.

### 네트워크 연결 고려 사항

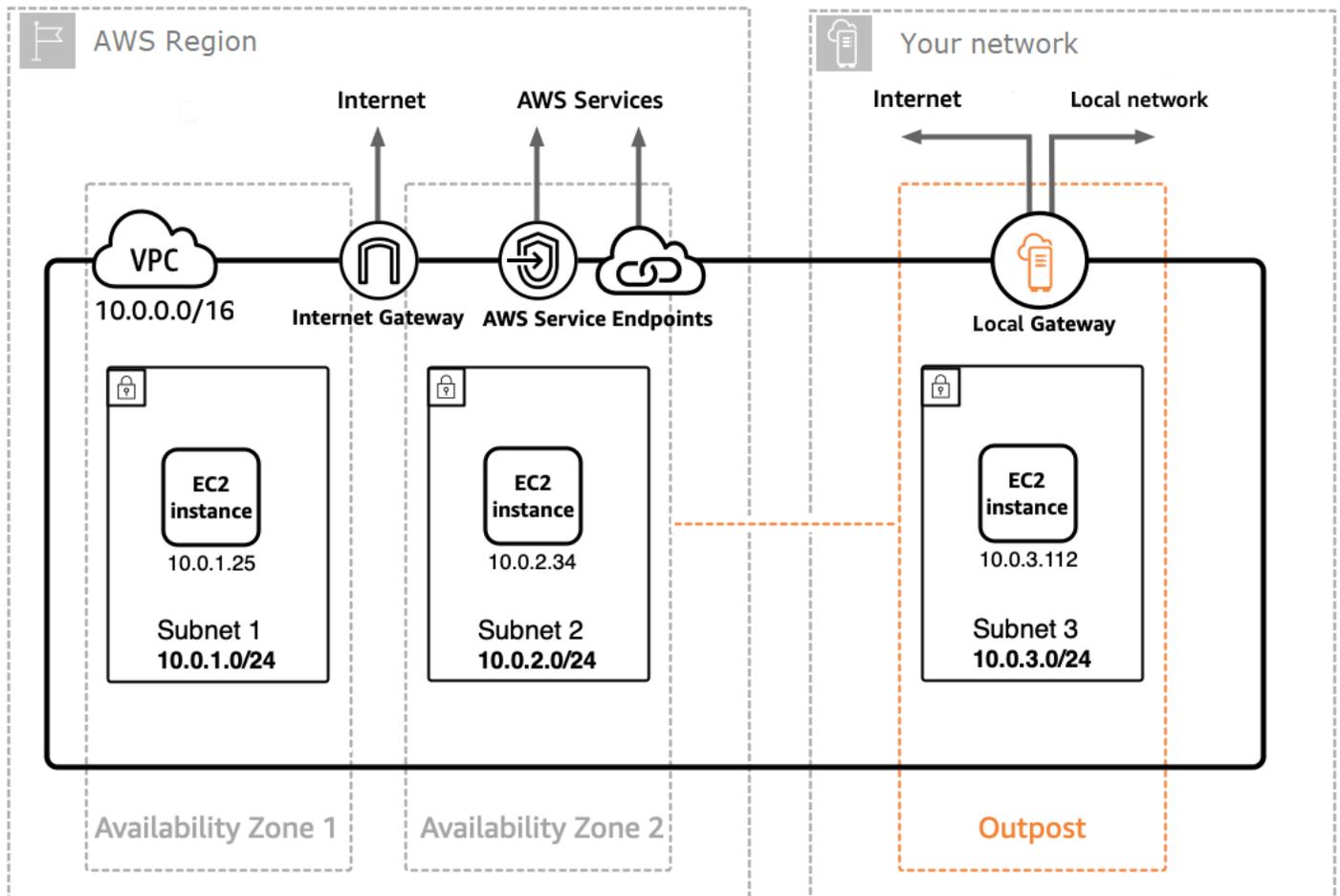
다음은 Amazon EKS AWS Outpost에 대한 네트워크 연결 고려 사항입니다.

- Outpost와 AWS 리전 간 네트워크 연결이 끊어져도 App Mesh Envoy 프록시는 계속 실행됩니다. 하지만 연결이 복원되기 전까지는 서비스 메시지를 수정할 수 없습니다.

- Outpost와 AWS 리전 간에 안정적이고 가용성이 높으며 지연 시간이 짧은 연결을 제공하는 것이 좋습니다.

## Outpost에서 App Mesh Envoy 프록시 생성

Outpost는 AWS 리전의 확장이며, 계정의 Amazon VPC를 여러 가용 영역 및 연결된 Outpost 위치에 걸쳐 확장할 수 있습니다. Outpost를 구성할 때 리전 환경을 온프레미스 시설로 확장하려면 서브넷과 연결합니다. Outpost의 인스턴스는 연결된 서브넷이 있는 가용 영역과 유사하게 리전 VPC의 일부로 나타납니다.



Outpost에 App Mesh Envoy 프록시를 생성하려면 Outpost에서 실행되는 Amazon ECS 태스크 또는 Amazon EKS 포드에 App Mesh Envoy 컨테이너 이미지를 추가합니다. 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 [AWS Outpost의 Amazon Elastic Container Service](#) 및 Amazon EKS 사용 설명서의 [AWS Outpost의 Amazon Elastic Kubernetes Service](#)를 참조하세요.

## App Mesh 모범 사례

계획된 배포 기간과 일부 호스트의 예상치 못한 손실 발생 시 요청 실패가 전혀 발생하지 않도록 한다는 목표를 달성하기 위해 이 주제의 모범 사례는 다음 전략을 구현합니다.

- 안전한 기본 재시도 전략을 사용하면 애플리케이션 관점에서 요청이 성공할 가능성을 높일 수 있습니다. 자세한 정보는 [재시도를 통해 모든 경로 계층](#)을 참조하세요.
- 재시도된 요청이 실제 대상으로 전송될 가능성을 최대화하여 재시도된 요청의 성공 가능성을 높입니다. 자세한 내용은 [배포 속도 조정](#), [스케일 인 전에 스케일 아웃](#), [컨테이너 상태 검사 구현](#) 단원을 참조하세요.

실패를 크게 줄이거나 없애려면 다음과 같은 모든 방법으로 권장 사항을 구현하는 것이 좋습니다.

### 재시도를 통해 모든 경로 계층

모든 가상 서비스가 가상 라우터를 사용하도록 구성하고 모든 경로에 대한 기본 재시도 정책을 설정합니다. 이렇게 하면 호스트가 다시 선택되고 새 요청이 전송되어 요청 실패를 줄일 수 있습니다. 재시도 정책의 경우 `maxRetries` 값을 2 이상으로 설정하고 재시도 이벤트 유형을 지원하는 각 경로 유형의 재시도 이벤트 유형에 대해 다음 옵션을 지정하는 것이 좋습니다.

- TCP – `connection-error`
- HTTP 및 HTTP/2 – `stream-error` 및 `gateway-error`
- gRPC – `cancelled` 및 `unavailable`

요청이 불능이 아닌 경우와 같이 안전하지 않을 수 있는 다른 재시도 이벤트는 `case-by-case` 근거를 두고 고려해야 합니다. `maxRetries` 및 `perRetryTimeout`에 대해 요청의 최대 지연 시간 (`maxRetries * perRetryTimeout`)과 추가 재시도의 성공률 증가 사이에서 적절한 균형을 이루는 값을 고려하고 테스트해야 합니다. 또한 Envoy가 더 이상 존재하지 않는 엔드포인트에 연결하려고 시도할 경우 해당 요청이 전체 `perRetryTimeout`을 소비할 것으로 예상할 수 있습니다. 재시도 정책을 구성하려면 [경로 생성](#) 섹션을 참조한 후 라우팅할 프로토콜을 선택하세요.

#### Note

2020년 7월 29일 또는 그 이후의 경로를 구현하고 재시도 정책을 지정하지 않은 경우, App Mesh는 2020년 7월 29일 또는 그 이후에 생성한 각 경로에 대해 이전 정책과 유사한 기본 재시

도 정책을 자동으로 생성했을 수 있습니다. 자세한 정보는 [기본 경로 재시도 정책](#)을 참조하세요.

## 배포 속도 조정

롤링 배포를 사용하는 경우 전체 배포 속도를 줄이세요. 기본적으로 Amazon ECS는 최소 100% 정상 태스크와 200%의 총 태스크로 이루어진 배포 전략을 구성합니다. 배포 시 이로 인해 높은 드리프트가 나타나는 두 지점이 발생합니다.

- 요청을 완료할 준비가 되기 전에 Envoy에서 새 태스크의 100% 플릿 크기를 확인할 수 있습니다(완화 방법은 [컨테이너 상태 검사 구현](#) 참조).
- 태스크가 종료되는 동안 Envoy에서 이전 태스크의 100% 플릿 크기를 확인할 수 있습니다.

이러한 배포 제약으로 구성된 경우 컨테이너 오케스트레이터는 모든 이전 대상을 동시에 숨기고 모든 새 대상을 표시하는 상태가 될 수 있습니다. Envoy 데이터 영역은 결과적으로 일관된 상태를 유지하므로 데이터 영역에 표시되는 대상 집합이 오케스트레이터의 관점과는 다르게 표시되는 기간이 발생할 수 있습니다. 이를 완화하려면 정상 태스크는 최소 100%로 유지하고 총 태스크는 125%로 줄이는 것이 좋습니다. 이렇게 하면 발산이 줄어들고 재시도의 안정성이 향상됩니다. 다양한 컨테이너 런타임의 경우 다음 제안 사항을 따르는 것이 좋습니다.

### Amazon ECS

원하는 서비스 개수가 2~3개인 경우 maximumPercent를 150퍼센트로 설정하세요. 그렇지 않으면 maximumPercent를 125%로 설정하세요.

### Kubernetes

maxUnavailable을 0%로, maxSurge를 25%로 설정하여 배포의 update strategy을 구성합니다. 배포에 대한 자세한 내용은 Kubernetes [배포](#) 설명서를 참조하세요.

## 스케일 인 전에 스케일 아웃

스케일 아웃과 스케일 인 모두 재시도에서 요청이 실패할 가능성이 있습니다. 스케일 아웃을 완화하는 태스크 권장 사항도 있지만 스케일 인에 대한 유일한 권장 사항은 한 번에 스케일 인되는 태스크의 비율을 최소화하는 것입니다. 기존 태스크나 배포를 스케일 인하기 전에 새 Amazon ECS 태스크 또는 Kubernetes 배포를 스케일 아웃하는 배포 전략을 사용하는 것이 좋습니다. 이 스케일링 전략을 사용

하면 속도는 동일하게 유지하면서 태스크 또는 배포의 스케일 인 비율을 낮출 수 있습니다. 이 방법은 Amazon ECS 태스크와 Kubernetes 배포 모두에 적용됩니다.

## 컨테이너 상태 검사 구현

스케일 업 시나리오에서 Amazon ECS 태스크의 컨테이너는 오류로 인해 초기에 응답하지 않을 수 있습니다. 다양한 컨테이너 런타임의 경우 다음 제안 사항을 따르는 것이 좋습니다.

### Amazon ECS

이를 완화하려면 아웃바운드 네트워크 연결이 필요한 컨테이너를 시작하기 전에 컨테이너 상태 검사 및 컨테이너 종속성 순서 지정을 사용하여 Envoy가 정상적으로 실행되고 있는지 확인하는 것이 좋습니다. 태스크 정의에서 애플리케이션 컨테이너와 Envoy 컨테이너를 올바르게 구성하려면 [컨테이너 종속성](#)을 참조하세요.

### Kubernetes

[없음. Kubernetes용 App Mesh 컨트롤러에서 인스턴스를 등록 및 등록 취소할 때 Kubernetes 활성화 및 준비 상태AWS Cloud Map 프로브가 고려되지 않기 때문입니다.](#) 자세한 내용은 [문제 #132](#) 를 참조하십시오. [시오. GitHub](#)

## DNS 해상도 최적화

서비스 검색에 DNS를 사용하는 경우 메시지를 구성할 때 DNS 해상도를 최적화하기 위해 적절한 IP 프로토콜을 선택하는 것이 중요합니다. App Mesh는 IPv4 및 을 모두 지원하며 IPv6, 선택에 따라 서비스 성능 및 호환성에 영향을 미칠 수 있습니다. 인프라가 지원하지 IPv6 않는 경우 기본 IPv6\_PREFERRED 동작에 의존하기보다는 인프라에 맞는 IP 설정을 지정하는 것이 좋습니다. 기본 IPv6\_PREFERRED 동작은 서비스 성능을 저하시킬 수 있습니다.

- IPv6\_Preferred — 기본 설정입니다. Envoy는 IPv6 주소에 대한 DNS 검색을 먼저 수행하고 주소가 발견되지 않으면 다시 검색합니다. IPv4 IPv6 이는 인프라가 주로 지원하지만 호환성이 필요한 경우에 유용합니다. IPv6 IPv4
- IPv4\_Preferred — Envoy는 먼저 IPv4 주소를 조회한 후 사용할 수 있는 주소가 없는 경우 다시 검색합니다 IPv6. IPv4 인프라가 주로 지원하지만 호환성이 어느 정도 있는 경우 이 설정을 사용하세요. IPv4 IPv6
- IPv6\_only — 서비스가 트래픽만 지원하는 경우 이 옵션을 선택합니다. IPv6 Envoy는 IPv6 주소에 대한 DNS 조회만 수행하여 모든 트래픽이 라우팅되도록 합니다. IPv6

- `IPv4_only` — 서비스가 트래픽만 지원하는 경우 이 설정을 선택합니다. IPv4 Envoy는 IPv4 주소에 대한 DNS 조회만 수행하여 모든 트래픽이 라우팅되도록 합니다. IPv4

메시 수준과 가상 노드 수준 모두에서 IP 버전 기본 설정을 지정할 수 있으며, 가상 노드 설정은 메시 수준의 설정보다 우선합니다.

자세한 내용은 [서비스 메시](#) 및 [가상](#) 노드를 참조하십시오.

# 보안 내부 AWS App Mesh

클라우드 AWS 보안이 최우선 과제입니다. AWS 고객은 가장 보안에 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 기업과 기업 간의 AWS 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 - AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호하는 역할을 합니다. AWS 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사원은 정기적으로 [AWS 규정 준수 프로그램](#)의 일환으로 보안 효과를 테스트하고 검증합니다. AWS App Mesh에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 [규정 준수 프로그램 제공 범위 내의 AWS 서비스](#)를 참조하십시오. App Mesh는 TLS 인증서 개인 키와 같은 비밀을 포함하여 로컬 프록시에 구성을 안전하게 전달하는 역할을 합니다.
- 클라우드에서의 보안 — 사용하는 AWS 서비스에 따라 책임이 결정됩니다. 다음과 같은 기타 요인에 대한 책임도 귀하에게 있습니다.
  - 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정
  - App Mesh 데이터 플레인의 보안 구성 (사용자 내 서비스 간에 트래픽이 전달되도록 허용하는 보안 그룹 구성 포함) VPC
  - App Mesh와 관련된 컴퓨팅 리소스의 구성
  - 컴퓨팅 리소스와 관련된 IAM 정책 및 App Mesh 컨트롤 플레인에서 검색할 수 있는 구성

이 설명서는 App Mesh 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 App Mesh를 구성하는 방법을 보여 줍니다. 또한 App Mesh 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법도 알아봅니다.

## App Mesh 보안 원칙

고객은 필요한 범위까지 보안을 조정할 수 있어야 합니다. 플랫폼이 보안을 강화하는 데 방해가 되어서는 안 됩니다. 플랫폼 기능은 기본적으로 안전합니다.

## 주제

- [전송 계층 보안 \(TLS\)](#)
- [상호 TLS 인증](#)

- [수정할 수 있다면 방법이 무엇입니까? AWS App Mesh 와 함께 작동합니다 IAM](#)
- [를 사용하여 AWS App Mesh API 호출 로깅 AWS CloudTrail](#)
- [의 데이터 보호 AWS App Mesh](#)
- [규정 준수 검증 AWS App Mesh](#)
- [의 인프라 보안 AWS App Mesh](#)
- [AWS App Mesh의 복원성](#)
- [의 구성 및 취약성 분석 AWS App Mesh](#)

## 전송 계층 보안 (TLS)

App Mesh에서 전송 계층 보안 (TLS) 은 App Mesh에서 메시 엔드포인트로 표시되는 컴퓨팅 리소스에 배포된 Envoy 프록시 간의 통신을 암호화합니다 (예: 및). [가상 노드 가상 게이트웨이](#) 프록시는 협상하고 종료합니다. TLS 프록시가 애플리케이션과 함께 배포되는 경우 애플리케이션 코드는 세션 협상을 담당하지 않습니다. TLS 프록시는 애플리케이션을 TLS 대신하여 협상합니다.

App Mesh를 사용하면 다음과 같은 방법으로 프록시에 TLS 인증서를 제공할 수 있습니다.

- AWS Certificate Manager (ACM) 에서 발급한 AWS Private Certificate Authority 사설 인증서.AWS Private CA
- 자체 인증 기관(CA)에서 발급한 가상 노드의 로컬 파일 시스템에 저장된 인증서
- Secrets Discovery Service (SDS) 엔드포인트가 로컬 Unix 도메인 소켓을 통해 제공하는 인증서.

메시 엔드포인트로 표시되는 배포된 Envoy 프록시에 대해 [Envoy 프록시 권한 부여](#)를 활성화해야 합니다. 프록시 권한 부여를 활성화할 때는 암호화를 활성화하려는 메시 엔드포인트로만 액세스를 제한합니다.

## 인증서 요구 사항

메시 엔드포인트가 나타내는 실제 서비스가 검색되는 방식에 따라 인증서의 주체 대체 이름 (SANs) 중 하나가 특정 기준과 일치해야 합니다.

- DNS— 인증서 중 하나가 DNS 서비스 검색 설정에 제공된 값과 SANs 일치해야 합니다. 서비스 검색 이름이 *mesh-endpoint.apps.local*인 애플리케이션의 경우 해당 이름과 일치하는 인증서 또는 와일드카드 *\*.apps.local*을 포함하는 인증서를 생성할 수 있습니다.
- AWS Cloud Map- 인증서 중 하나는 형식을 사용하여 AWS Cloud Map 서비스 검색 설정에 제공된 값과 SANs 일치해야 *service-name.namespace-name* 합니다. AWS Cloud Map 서

비스 검색 설정이 `serviceName mesh-endpoint` 및 인 응용 프로그램의 경우 이름과 `mesh-endpoint.apps.local` 일치하는 인증서를 만들거나 와일드카드가 있는 인증서를 만들 수 있습니다. `namespaceName apps.local *.apps.local`.

두 검색 메커니즘 모두에서 DNS 서비스 검색 설정과 SANs 일치하는 인증서가 없는 경우 Envoy 간 연결이 실패하고 클라이언트 Envoy에서 표시되는 다음 오류 메시지가 표시됩니다.

```
TLS error: 268435581:SSL routines:OPENSSL_internal:CERTIFICATE_VERIFY_FAILED
```

## TLS인증 인증서

App Mesh는 TLS 인증을 사용할 때 인증서에 대한 여러 소스를 지원합니다.

### AWS Private CA

인증서를 사용할 메시 엔드포인트와 동일한 지역 및 AWS 계정에 인증서를 저장해야 합니다. ACM CA의 인증서가 동일한 AWS 계정에 있을 필요는 없지만 메시 엔드포인트와 동일한 리전에 있어야 합니다. 인증서가 없는 AWS Private CA 경우 인증서를 요청하려면 먼저 [생성해야](#) 합니다. 기존 AWS Private CA 사용자에게 인증서를 요청하는 방법에 대한 자세한 내용은 [사설 인증서 요청을](#) 참조하십시오. ACM 인증서는 퍼블릭 인증서일 수 없습니다.

TLS클라이언트 정책에 CAs 사용하는 사설은 루트 사용자여야 합니다CAs.

인증서와 CAs 출처를 AWS Private CA 사용하여 가상 노드를 구성하려면 App Mesh를 호출하는 데 사용하는 보안 주체 (예: 사용자 또는 역할)에게 다음 IAM 권한이 있어야 합니다.

- 리스너 TLS 구성에 추가하는 모든 인증서의 경우 보안 주체에게 `acm:DescribeCertificate` 권한이 있어야 합니다.
- TLS클라이언트 정책에 CAs 구성된 모든 항목의 경우 보안 주체에게 `acm-pca:DescribeCertificateAuthority` 권한이 있어야 합니다.

#### Important

다른 CAs 계정과 공유하면 해당 계정에 의도하지 않은 권한이 CA에 부여될 수 있습니다. 리소스 기반 정책을 사용하여 CA에서 인증서를 발급할 필요가 없는 계정의 `acm-pca:DescribeCertificateAuthority` 및 `acm-pca:GetCertificateAuthorityCertificate`로만 액세스를 제한하는 것이 좋습니다.

보안 주체에 연결된 기존 IAM 정책에 이러한 권한을 추가하거나 새 보안 주체 및 정책을 만들고 정책을 보안 주체에 연결할 수 있습니다. 자세한 내용은 [IAM정책 편집](#), [IAM 정책 생성](#) 및 [IAMID 권한 추가](#)를 참조하십시오.

### Note

삭제할 AWS Private CA 때까지 각 작업에 대한 월별 요금을 지불합니다. 또한 매월 발급하는 프라이빗 인증서와 내보내는 프라이빗 인증서에 대한 비용도 지불하면 됩니다. 자세한 내용은 [AWS Certificate Manager 요금](#)을 참조하세요.

메시 엔드포인트가 나타내는 Envoy Proxy에 대한 [프록시 인증](#)을 활성화하는 경우 사용하는 IAM 역할에 다음 IAM 권한을 할당해야 합니다.

- 가상 노드의 리스너에 구성된 모든 인증서의 경우 역할에 `acm:ExportCertificate` 권한이 있어야 합니다.
- TLS클라이언트 정책에 CAs 구성된 모든 역할의 경우 역할에 권한이 있어야 합니다. `acm-pca:GetCertificateAuthorityCertificate`

### 파일 시스템

파일 시스템을 사용하여 Envoy에 인증서를 배포할 수 있습니다. 인증서 체인과 해당 프라이빗 키를 파일 경로에서 사용할 수 있도록 설정하여 이 작업을 수행할 수 있습니다. 이렇게 하면 Envoy 사이드카 프록시에서 이러한 리소스에 연결할 수 있습니다.

### Envoy의 비밀 검색 서비스 () SDS

Envoy는 비밀 검색 프로토콜을 통해 특정 엔드포인트에서 TLS 인증서와 같은 비밀을 가져옵니다. [이 프로토콜에 대한 자세한 내용은 Envoy 설명서를 참조하십시오.](#) [SDS](#)

App Mesh는 Envoy 프록시가 인증서 및 인증서 체인의 소스 역할을 할 때 Secret Discovery Service (SDS) 엔드포인트 SDS 역할을 할 때 프록시 로컬인 Unix 도메인 소켓을 사용하도록 구성합니다. `APPMESH_SDS_SOCKET_PATH` 환경 변수를 사용하여 이 엔드포인트에 대한 경로를 구성할 수 있습니다.

### Important

Unix 도메인 소켓을 사용하는 로컬 보안 암호 검색 서비스는 App Mesh Envoy 프록시 버전 1.15.1.0 이상에서 지원됩니다.

App Mesh는 g를 사용하는 V2 SDS 프로토콜을 지원합니다RPC.

## SPIFFE 런타임 환경과 통합 () SPIRE

[SPIFFERuntime Environment](#) () 와 같은 기존 도구 체인을 포함하여 의 모든 사이드카 구현을 사용할 수 있습니다. SDS API SPIRE SPIRE 분산 시스템의 여러 워크로드 간에 상호 TLS 인증을 배포할 수 있도록 설계되었습니다. 런타임 시 워크로드의 자격 증명을 증명합니다. SPIRE 또한 워크로드별로 수명이 짧고 자동으로 교체되는 키와 인증서를 워크로드에 직접 제공합니다.

SPIRE 에이전트를 Envoy용 공급자로 구성해야 합니다. SDS 상호 인증을 제공하는 데 필요한 주요 자료를 Envoy에 직접 제공할 수 있도록 허용하십시오. TLS Envoy 프록시 옆의 사이드카에서 SPIRE 에이전트를 실행하세요. 이 Agent는 필요에 따라 수명이 짧은 키와 인증서를 다시 생성합니다. 에이전트는 Envoy를 증명하고 Envoy가 에이전트가 노출한 서버에 연결할 때 Envoy가 사용할 수 있도록 제공해야 하는 서비스 ID 및 CA 인증서를 결정합니다. SDS SPIRE

이 프로세스 중에 서비스 자격 증명과 CA 인증서가 교체되고 업데이트가 Envoy로 다시 스트리밍됩니다. Envoy는 중단이나 가중 중지 없이, 그리고 파일 시스템에 프라이빗 키를 제공할 필요 없이 새 연결에 이러한 항목을 즉시 적용합니다.

## App Mesh가 협상을 위해 Envoy를 구성하는 방법 TLS

App Mesh는 메시에서 Envoy 간의 통신을 구성하는 방법을 결정할 때 클라이언트와 서버의 메시 엔드포인트 구성을 사용합니다.

### 클라이언트 정책 사용

클라이언트 정책이 사용을 강제하고 클라이언트 정책의 포트 중 하나가 서버 정책의 포트와 일치하면 클라이언트 정책을 사용하여 클라이언트의 유효성 검사 컨텍스트를 구성합니다. TLS TLS 예를 들어, 가상 게이트웨이의 클라이언트 정책이 가상 노드의 서버 정책과 일치하면 가상 게이트웨이의 클라이언트 정책에 정의된 설정을 사용하여 프록시 간에 TLS 협상이 시도됩니다. 클라이언트 정책이 서버 정책의 포트와 일치하지 않는 경우 서버 정책 설정에 따라 프록시 TLS 간 협상이 이루어질 수도 있고 협상되지 않을 수도 있습니다. TLS

### 클라이언트 정책을 사용하지 않는 경우

클라이언트가 클라이언트 정책을 구성하지 않았거나 클라이언트 정책이 서버의 포트와 일치하지 않는 경우 App Mesh는 서버를 사용하여 클라이언트와의 협상 TLS 여부 및 방법을 결정합니다. 예를 들어 가상 게이트웨이가 클라이언트 정책을 지정하지 않고 가상 노드가 TLS 종료를 구성하지 않은 경우 프록시 간에 협상되지 않습니다. TLS 클라이언트가 일치하는 클라이언트 정책을 지정하지 않고 서버가 TLS 모드 STRICT 또는 PERMISSIVE 로 구성된 경우 프록시는 협상하도록 구성됩니다. TLS TLS 종료를 위해 인증서가 제공된 방식에 따라 다음과 같은 추가 동작이 적용됩니다.

- ACM-관리형 TLS 인증서 - 서버에서 -managed 인증서를 사용하여 TLS 종료를 구성한 ACM 경우 App Mesh는 인증서가 체인으로 연결된 루트 사용자 CA에 대해 인증서를 TLS 협상하고 검증하도록 클라이언트를 자동으로 구성합니다.
- 파일 기반 TLS 인증서 - 서버가 프록시 로컬 파일 시스템의 인증서를 사용하여 TLS 종료를 구성한 경우 App Mesh는 TLS 협상할 클라이언트를 자동으로 구성하지만 서버의 인증서는 검증되지 않습니다.

## 주체 대체 이름

선택적으로 신뢰할 수 있는 주체 대체 이름 () 목록을 지정할 수 있습니다. SANs SANsFQDNor URI 형식이어야 합니다. 제공된 경우 SANs Envoy는 제출된 인증서의 주체 대체 이름이 이 목록에 있는 이름 중 하나와 일치하는지 확인합니다.

종료 메시 엔드포인트를 SANs 지정하지 않으면 해당 노드의 Envoy 프록시는 피어 클라이언트 인증서를 확인하지 않습니다. SAN 발신 메시 엔드포인트를 SANs 지정하지 않는 경우 종료 엔드포인트에서 제공하는 인증서의 내용이 메시 엔드포인트 서비스 검색 구성과 일치해야 합니다. SAN

자세한 내용은 App Mesh [TLS: 인증서 요구 사항을](#) 참조하십시오.

### Important

클라이언트 정책이 `not enforced` 설정된 경우에만 와일드카드를 SANs 사용할 수 있습니다. TLS 클라이언트 가상 노드 또는 가상 게이트웨이에 대한 클라이언트 정책이 적용되도록 구성된 경우 TLS 와일드카드를 받아들일 수 없습니다. SAN

## 암호화 확인

TLS활성화한 후에는 Envoy 프록시를 쿼리하여 통신이 암호화되었는지 확인할 수 있습니다. Envoy 프록시는 리소스에 대한 통계를 내보내 통신이 제대로 작동하는지 이해하는 데 도움이 될 수 있습니다. TLS. 예를 들어 Envoy 프록시는 지정된 메시 엔드포인트에 대해 협상한 성공적인 TLS 핸드셰이크 수에 대한 통계를 기록합니다. 다음 명령으로 이름을 지정한 메시 TLS 엔드포인트의 핸드셰이크 성공 횟수를 확인하십시오. `my-mesh-endpoint`

```
curl -s 'http://my-mesh-endpoint.apps.local:9901/stats' | grep ssl.handshake
```

다음 예제의 반환된 출력에서는 메시 엔드포인트에 대한 핸드셰이크가 세 번 있었으므로 통신이 암호화됩니다.

```
listener.0.0.0.0_15000.ssl.handshake: 3
```

Envoy 프록시는 협상이 실패할 때도 통계를 내보냅니다. TLS 메시 엔드포인트에 TLS 오류가 있었는지 확인하십시오.

```
curl -s 'http://my-mesh-endpoint.apps.local:9901/stats' | grep -e "ssl.*\((fail\|error\)"
```

반환된 예제의 출력에서는 여러 통계에 대해 오류가 0이었으므로 TLS 협상이 성공했습니다.

```
listener.0.0.0.0_15000.ssl.connection_error: 0
listener.0.0.0.0_15000.ssl.fail_verify_cert_hash: 0
listener.0.0.0.0_15000.ssl.fail_verify_error: 0
listener.0.0.0.0_15000.ssl.fail_verify_no_cert: 0
listener.0.0.0.0_15000.ssl.fail_verify_san: 0
```

[Envoy 통계에 대한 자세한 내용은 Envoy 리스너 TLS 통계를 참조하십시오.](#)

## 인증서 갱신

### AWS Private CA

를 사용하여 ACM 인증서를 갱신하면 갱신 완료 후 35분 이내에 갱신된 인증서가 연결된 프록시에 자동으로 배포됩니다. 유효 기간이 거의 끝나갈 즈음에 인증서를 자동으로 갱신하려면 관리형 갱신을 사용하는 것이 좋습니다. 자세한 내용은 [사용 ACM설명서의 Amazon 발급 인증서에 AWS Certificate Manager 대한 관리형 갱신](#)을 참조하십시오.

### 자체 인증서

로컬 파일 시스템의 인증서를 사용하는 경우 Envoy는 인증서가 변경되어도 인증서를 자동으로 다시 로드하지 않습니다. Envoy 프로세스를 다시 시작하거나 재배포하여 새 인증서를 로드할 수 있습니다. 새 인증서를 다른 파일 경로에 배치하고 해당 파일 경로로 가상 노드 또는 게이트웨이 구성을 업데이트할 수도 있습니다.

## TLS인증을 사용하도록 Amazon ECS 워크로드를 구성합니다. AWS App Mesh

TLS인증을 사용하도록 메시를 구성할 수 있습니다. 워크로드에 추가하는 Envoy 프록시 사이드카에서 인증서를 사용할 수 있는지 확인합니다. EBS또는 EFS 볼륨을 Envoy 사이드카에 연결하거나 Secrets AWS Manager에서 인증서를 저장하고 검색할 수 있습니다.

- 파일 기반 인증서 배포를 사용하는 경우 Envoy 사이드카에 EBS 또는 EFS 볼륨을 연결하십시오. 인증서 및 개인 키의 경로가 에 구성된 경로와 일치하는지 확인하십시오. AWS App Mesh
- SDS기반 배포를 사용하는 경우 인증서에 액세스할 수 있는 Envoy를 구현하는 사이드카를 SDS API 추가하세요.

### Note

SPIREAmazon에서는 지원되지 않습니다ECS.

## 인증을 사용하도록 Kubernetes 워크로드를 구성합니다. TLS AWS App Mesh

가상 노드와 가상 게이트웨이 서비스 백엔드 및 리스너에 대한 TLS 인증을 활성화하도록 Kubernetes 용 AWS App Mesh 컨트롤러를 구성할 수 있습니다. 워크로드에 추가하는 Envoy 프록시 사이드카에서 인증서를 사용할 수 있는지 확인합니다. 상호 인증의 [안내](#) 섹션에서 각 배포 유형의 예를 볼 수 있습니다. TLS

- 파일 기반 인증서 배포를 사용하는 경우 Envoy 사이드카에 EBS 또는 EFS 볼륨을 연결하세요. 인증서 및 프라이빗 키의 경로가 컨트롤러에 구성된 경로와 일치하는지 확인합니다. 또는 파일 시스템에 탑재된 Kubernetes Secret을 사용할 수도 있습니다.
- SDS기반 배포를 사용하는 경우 Envoy를 구현하는 노드 로컬 SDS 공급자를 설정해야 합니다. SDS API Envoy가 문제를 해결해 드릴 것입니다. UDS EKS AppMesh 컨트롤러에서 m SDS 기반 TLS 지원을 활성화하려면 플래그를 로 true 설정하고 enable-sds 플래그를 통해 컨트롤러에 대한 로컬 SDS 제공자의 UDS 경로를 제공하십시오. sds-uds-path helm을 사용하는 경우 컨트롤러 설치의 일부로 다음을 설정합니다.

```
--set sds.enabled=true
```

### Note

Fargate 모드에서 Amazon Elastic Kubernetes Service (Amazon) 를 사용하는 경우 인증서를 배포하는 SPIRE 데 사용할 수 없습니다. EKS

## 상호 TLS 인증

상호 TLS(전송 계층 보안) 인증은 양방향 피어 인증을 제공하는 TLS의 선택적 구성 요소입니다. 상호 TLS 인증은 TLS 위에 보안 계층을 추가하고 서비스가 연결을 설정하는 클라이언트를 확인할 수 있도록 합니다.

클라이언트-서버 관계에 있는 클라이언트도 세션 협상 프로세스 중에 X.509 인증서를 제공합니다. 서버는 이 인증서를 사용하여 클라이언트를 식별하고 인증합니다. 이 프로세스를 통해 신뢰할 수 있는 인증 기관(CA)에서 발급한 인증서인지와 유효한 인증서인지 확인할 수 있습니다. 또한 인증서의 주체 대체 이름(SAN)을 사용하여 클라이언트를 식별합니다.

에서 지원하는 AWS App Mesh 모든 프로토콜에 대해 상호 TLS 인증을 활성화할 수 있습니다. 이러한 프로토콜은 TCP, HTTP/1.1, HTTP/2, gRPC입니다.

### Note

App Mesh를 사용하면 서비스의 Envoy 프록시 간 통신을 위해 상호 TLS 인증을 구성할 수 있습니다. 하지만 애플리케이션과 Envoy 프록시 간의 통신은 암호화되지 않습니다.

## 상호 TLS 인증 인증서

AWS App Mesh 상호 TLS 인증을 위한 두 가지 가능한 인증서 소스를 지원합니다. TLS 클라이언트 정책의 클라이언트 인증서와 리스너 TLS 구성의 서버 검증은 다음에서 가져올 수 있습니다.

- 파일 시스템 - 실행 중인 Envoy 프록시의 로컬 파일 시스템에서 가져온 인증서. Envoy에 인증서를 배포하려면 인증서 체인의 파일 경로와 App Mesh API에 대한 프라이빗 키를 제공해야 합니다.
- Envoy의 비밀 검색 서비스 (SDS) — SDS를 구현하고 Envoy에 인증서를 보낼 수 있도록 하는 B ring-your-own 사이드카입니다. 여기에는 SPIFFE Runtime Environment(SPIRE)가 포함됩니다.

### Important

App Mesh는 상호 TLS 인증에 사용되는 인증서나 프라이빗 키를 저장하지 않습니다. 대신 Envoy는 이러한 항목을 메모리에 저장합니다.

## 메시 엔드포인트 구성

가상 노드 또는 게이트웨이와 같은 메시 엔드포인트에 대한 상호 TLS 인증을 구성합니다. 이러한 엔드포인트는 인증서를 제공하고 신뢰할 수 있는 기관을 지정합니다.

이렇게 하려면 클라이언트와 서버 모두에 대해 X.509 인증서를 프로비저닝하고 TLS 종료 및 TLS 시작 모두의 검증 컨텍스트에서 신뢰할 수 있는 기관 인증서를 명시적으로 정의해야 합니다.

### 메시 내부의 신뢰

서버 측 인증서는 가상 노드 리스너(TLS 종료)에서 구성되고, 클라이언트 측 인증서는 가상 노드 서비스 백엔드(TLS 시작)에서 구성됩니다. 이 구성 대신, 가상 노드의 모든 서비스 백엔드에 대한 기본 클라이언트 정책을 정의한 다음, 필요에 따라 특정 백엔드에 대해 이 정책을 재정의할 수 있습니다. 가상 게이트웨이는 모든 백엔드에 적용되는 기본 클라이언트 정책으로만 구성할 수 있습니다.

두 메시의 가상 게이트웨이에서 인바운드 트래픽에 대해 상호 TLS 인증을 활성화하여 서로 다른 메시 간에 신뢰를 구성할 수 있습니다.

### 메시 외부의 신뢰

가상 게이트웨이 리스너에서 TLS 종료를 위한 서버 측 인증서를 지정합니다. 가상 게이트웨이와 통신하여 클라이언트 측 인증서를 제공하는 외부 서비스를 구성합니다. 인증서는 서버 측 인증서가 TLS 시작을 위해 가상 게이트웨이 리스너에서 사용하는 것과 동일한 인증 기관(CA) 중 하나에서 파생되어야 합니다.

## 상호 TLS 인증으로 서비스 마이그레이션

App Mesh 내의 기존 서비스를 상호 TLS 인증으로 마이그레이션할 때 연결을 유지하려면 다음 지침을 따르세요.

### 일반 텍스트를 통해 통신하는 서비스 마이그레이션

1. 서버 엔드포인트의 TLS 구성에 대해 PERMISSIVE 모드를 활성화합니다. 이 모드에서는 일반 텍스트 트래픽이 엔드포인트에 연결하도록 허용합니다.
2. 서버 인증서, 신뢰 체인 및 선택적으로 신뢰할 수 있는 SAN을 지정하여 서버에서 상호 TLS 인증을 구성합니다.
3. TLS 연결을 통해 통신이 이루어지고 있는지 확인합니다.
4. 클라이언트 인증서, 신뢰 체인 및 선택적으로 신뢰할 수 있는 SAN을 지정하여 클라이언트에서 상호 TLS 인증을 구성합니다.

## 5. 서버의 TLS 구성에 대해 STRICT 모드를 활성화합니다.

### TLS를 통해 통신하는 서비스 마이그레이션

1. 클라이언트 인증서 및 선택적으로 신뢰할 수 있는 SAN을 지정하여 클라이언트에서 상호 TLS 설정을 구성합니다. 클라이언트 인증서는 백엔드 서버에서 요청하기 전까지는 백엔드로 전송되지 않습니다.
2. 신뢰 체인과 선택적으로 신뢰할 수 있는 SAN을 지정하여 서버의 상호 TLS 설정을 구성합니다. 이를 위해 서버는 클라이언트 인증서를 요청합니다.

## 상호 TLS 인증 확인

[전송 계층 보안: 암호화 확인](#) 설명서를 참조하여 Envoy가 TLS 관련 통계를 정확히 어떻게 내보내는지 확인할 수 있습니다. 상호 TLS 인증의 경우 다음 통계를 검사해야 합니다.

- `ssl.handshake`
- `ssl.no_certificate`
- `ssl.fail_verify_no_cert`
- `ssl.fail_verify_san`

다음 두 가지 통계 예제는 가상 노드로 종료되는 성공적인 TLS 연결이 모두 인증서를 제공한 클라이언트에서 시작되었음을 보여 줍니다.

```
listener.0.0.0.0_15000.ssl.handshake: 3
```

```
listener.0.0.0.0_15000.ssl.no_certificate: 0
```

다음 통계 예제는 가상 클라이언트 노드(또는 게이트웨이)에서 백엔드 가상 노드로의 연결이 실패했음을 보여 줍니다. 서버 인증서에 표시된 주체 대체 이름(SAN)이 클라이언트가 신뢰하는 SAN과 일치하지 않습니다.

```
cluster.cds_egress_my-mesh_my-backend-node_http_9080.ssl.fail_verify_san: 5
```

## App Mesh 상호 TLS 인증 살펴보기

- [상호 TLS 인증 살펴보기](#): 이 살펴보기에서는 App Mesh CLI를 사용하여 상호 TLS 인증으로 컬러 앱을 구축하는 방법을 설명합니다.
- [Amazon EKS 상호 TLS SDS 기반 살펴보기](#): 이 살펴보기에서는 Amazon EKS 및 SPIFFE Runtime Environment(SPIRE)에서 상호 TLS SDS 기반 인증을 사용하는 방법을 보여 줍니다.
- [Amazon EKS 상호 TLS 파일 기반 살펴보기](#): 이 살펴보기에서는 Amazon EKS 및 SPIFFE Runtime Environment(SPIRE)에서 상호 TLS 파일 기반 인증을 사용하는 방법을 보여 줍니다.

## 수정할 수 있다면 방법이 무엇입니까? AWS App Mesh 와 함께 작동합니다 IAM

AWS Identity and Access Management (IAM) 는 AWS 서비스 이를 통해 관리자는 다음 항목에 대한 액세스를 안전하게 제어할 수 있습니다. AWS 있습니다. IAM관리자는 App Mesh 리소스를 사용할 수 있는 인증 (로그인) 및 권한 부여 (권한 보유) 를 받을 수 있는 사용자를 제어합니다. IAM는 AWS 서비스 추가 비용 없이 사용할 수 있습니다.

### 주제

- [고객](#)
- [ID를 통한 인증](#)
- [정책을 사용한 액세스 관리](#)
- [수정할 수 있다면 방법이 무엇입니까? AWS App Mesh 다음과 함께 작동합니다. IAM](#)
- [AWS App Mesh ID 기반 정책 예제](#)
- [AWS App Mesh에 대한 관리형 정책](#)
- [App Mesh에 서비스 연결 역할 사용](#)
- [Envoy 프록시 권한 부여](#)
- [문제 해결 AWS App Mesh ID 및 액세스](#)

### 고객

사용 방법 AWS Identity and Access Management (IAM) 는 App Mesh에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 - App Mesh 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 App Mesh 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. App Mesh의 기능에 액세스할 수 없는 경우 [문제 해결 AWS App Mesh ID 및 액세스](#) 섹션을 참조하세요.

서비스 관리자 - 회사에서 App Mesh 리소스를 책임지고 있는 경우 App Mesh에 대한 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 App Mesh 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 서비스 사용자의 권한 변경 요청을 제출해야 합니다. 이 페이지의 정보를 검토하여 의 기본 개념을 IAM 이해하십시오. 회사에서 App Mesh를 사용하는 방법에 대해 자세히 알아보려면 IAM 을 참조하십시오 [수정할 수 있다면 방법이 무엇입니까? AWS App Mesh 다음과 함께 작동합니다. IAM](#).

IAM 관리자 - IAM 관리자인 경우 App Mesh에 대한 액세스를 관리하기 위한 정책을 작성하는 방법에 대해 자세히 알아보는 것이 좋습니다. 에서 IAM 사용할 수 있는 App Mesh ID 기반 정책의 예를 보려면 을 참조하십시오. [AWS App Mesh ID 기반 정책 예제](#)

## ID를 통한 인증

인증은 로그인하는 방법입니다. AWS ID 자격 증명 사용 인증 (로그인) 을 받아야 합니다. AWS(다음과 같이) AWS 계정 루트 사용자 IAM사용자로서, 또는 IAM 역할을 맡아서

에 로그인할 수 있습니다. AWS ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 사용할 수 있습니다. AWS IAM Identity Center 페더레이션 ID의 예로는 (IAMID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명입니다. 페더레이션 ID로 로그인하는 경우 관리자는 이전에 역할을 사용하여 ID 페더레이션을 설정했습니다. IAM 액세스하는 경우 AWS 페더레이션을 사용하면 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 로그인할 수 있습니다. AWS Management Console 또는 AWS 액세스 포털. 로그인에 대한 자세한 내용은 AWS로그인하는 [방법을 참조하십시오. AWS 계정](#)의 AWS 로그인 사용자 가이드.

액세스하는 경우 AWS 프로그래밍 방식으로 AWS 자격 증명을 사용하여 요청에 암호로 서명할 수 있는 소프트웨어 개발 키트 (SDKCLI) 와 명령줄 인터페이스 () 를 제공합니다. 사용하지 않는 경우 AWS 도구를 사용하려면 직접 요청에 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 [서명을 참조하십시오. AWS APIIAM사용 설명서](#)의 요청.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예: AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 [다단계 인증을](#) 참조하십시오. AWS IAM Identity Center 사용 설명서 및 다단계 인증 [사용 \(\) MFA 의 AWS](#)(출처: IAM 사용 설명서).

## AWS 계정 루트 사용자

생성할 때 AWS 계정 모든 계정에 완전히 액세스할 수 있는 하나의 로그인 ID로 시작합니다. AWS 서비스 및 계정 내 리소스. 이 ID를 다음과 같이 부릅니다. AWS 계정 루트 사용자는 계정을 만들 때 사용한 이메일 주소와 암호로 로그인하여 액세스할 수 있습니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 작업의 전체 목록은 사용 설명서의 [루트 사용자 자격 증명](#)이 필요한 작업을 참조하십시오. IAM

## IAM 사용자 및 그룹

[IAM 사용자](#)는 내 ID 내에 있는 ID입니다. AWS 계정 이는 한 사람이나 애플리케이션에 대한 특정 권한을 가지고 있습니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명을 가진 IAM 사용자를 만드는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 특정 사용 사례에서 IAM 사용자의 장기 자격 증명が必要な 경우에는 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 사용 설명서의 [장기 자격 증명](#)이 필요한 사용 사례에 대한 정기적인 액세스 키 IAM 교체를 참조하십시오.

[IAM 그룹](#)은 IAM 사용자 컬렉션을 지정하는 ID입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어 이름을 지정한 IAMAdmins 그룹을 만들고 해당 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수입할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세히 알아보려면 사용 [설명서의 역할 대신 IAM 사용자](#)를 만드는 시기를 참조하십시오. IAM

## IAM 역할

[IAM 역할](#)은 내 안의 정체성입니다. AWS 계정 여기에는 특정 권한이 있습니다. 사용자와 비슷하지만 특정 IAM 사용자와는 관련이 없습니다. 에서 일시적으로 IAM 역할을 맡을 수 있습니다. AWS Management Console [역할을 바꿔서 말](#)이죠. 를 호출하여 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 오버레이션을 사용하거나 사용자 지정을 사용합니다 URL. 역할 사용 방법에 대한 자세한 내용은 사용 IAM 설명서의 [IAM 역할 사용](#)을 참조하십시오.

IAM 임시 자격 증명에 있는 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페

더레이션을 위한 역할에 대한 자세한 내용은 IAM사용 설명서의 [타사 ID 제공자를 위한 역할 생성을](#) 참조하십시오. IAMIdentity Center를 사용하는 경우 권한 집합을 구성합니다. ID가 인증된 후 액세스할 수 있는 대상을 제어하기 위해 IAM Identity Center는 권한 집합을 역할의 상관 관계와 연결합니다. IAM 권한 집합에 대한 자세한 내용은 권한 집합의 사용 [권한](#) 집합을 참조하십시오. AWS IAM Identity Center 사용 설명서.

- **임시 IAM 사용자 권한** — IAM 사용자 또는 역할은 역할을 맡아 특정 작업에 대해 일시적으로 다른 권한을 부여받을 수 있습니다. IAM
- **계정 간 액세스** - IAM 역할을 사용하여 다른 계정의 사용자 (신뢰할 수 있는 사용자) 가 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 하지만 일부 경우에는 AWS 서비스역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 계정 간 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 사용 설명서의 [IAM 계정 간 리소스 액세스를](#) 참조하십시오. IAM
- **서비스 간 액세스** — 일부 AWS 서비스 다른 기능 사용 AWS 서비스. 예를 들어, 서비스를 호출하면 해당 서비스가 Amazon에서 애플리케이션을 EC2 실행하거나 Amazon S3에 객체를 저장하는 것이 일반적입니다. 서비스는 직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
- **전달 액세스 세션 (FAS)** — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 AWS, 귀하는 주도자로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS전화를 건 주체의 권한을 사용합니다. AWS 서비스, 요청과 결합 AWS 서비스 다운스트림 서비스에 요청할 수 있습니다. FAS요청은 서비스가 다른 서비스와의 상호 작용이 필요한 요청을 수신할 때만 이루어집니다. AWS 서비스 또는 완료해야 할 리소스. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS요청 시 적용되는 정책 세부 정보는 [전달 액세스 세션을](#) 참조하십시오.
- **서비스 역할** - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 간주하는 [IAM 역할입니다](#). IAM관리자는 내부에서 IAM 서비스 역할을 만들고, 수정하고, 삭제할 수 있습니다. 자세한 내용은 권한을 위임하기 위한 역할 [만들기를 참조하십시오](#). [AWS 서비스](#)(출처: IAM 사용 설명서).
- **서비스 연결 역할** - 서비스 연결 역할은 다음과 연결된 서비스 역할 유형입니다. AWS 서비스. 서비스가 사용자를 대신하여 작업을 수행하는 역할을 맡을 수 있습니다. 서비스 연결 역할은 다음과 같습니다. AWS 계정 서비스가 소유합니다. IAM관리자는 서비스 연결 역할에 대한 권한을 볼 수 있지만 편집할 수는 없습니다.
- **Amazon에서 실행되는 애플리케이션 EC2** — IAM 역할을 사용하여 EC2 인스턴스에서 실행 중이고 다음을 생성하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS CLI 또는 AWS API요청. EC2인스턴스 내에 액세스 키를 저장하는 것보다 이 방법이 더 좋습니다. 할당하려면 AWS EC2 인스턴스에 역할을 부여하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인

스턴스 프로필을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며, 이를 통해 EC2 인스턴스에서 실행 중인 프로그램이 임시 자격 증명을 얻을 수 있습니다. 자세한 내용은 [사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 IAM 참조하십시오](#).

IAM 역할을 사용할지 IAM 사용자를 사용할지 알아보려면 [사용 설명서의 IAM 역할 생성 시기 \(사용자 대신\)](#) 를 IAM 참조하십시오.

## 정책을 사용한 액세스 관리

에서 액세스를 제어할 수 있습니다. AWS 정책을 생성하여 정책에 연결함으로써 AWS ID 또는 리소스 정책은 다음의 객체입니다. AWS ID 또는 리소스와 연결될 경우 해당 권한을 정의합니다. AWS 주체 (사용자, 루트 사용자 또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는지를 결정합니다. 대부분의 정책은 다음 위치에 저장됩니다. AWS JSON 문서로. JSON 정책 문서의 구조 및 내용에 대한 자세한 내용은 IAM 사용 [설명서의 JSON 정책 개요](#) 를 참조하십시오.

관리자는 다음을 사용할 수 있습니다. AWS JSON 정책을 통해 누가 무엇에 액세스할 수 있는지 지정합니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하여 필요한 리소스에서 작업을 수행할 수 있는 권한을 사용자에게 부여할 수 있습니다. 그러면 관리자가 역할에 IAM 정책을 추가할 수 있으며, 사용자는 역할을 수입할 수 있습니다.

IAM 정책은 작업을 수행하는 데 사용하는 방법에 관계없이 작업에 대한 권한을 정의합니다. 예를 들어, iam:GetRole 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 에서 역할 정보를 가져올 수 있습니다. AWS Management Console, AWS CLI, 또는 AWS API.

## 보안 인증 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. ID 기반 정책을 만드는 방법을 알아보려면 [사용 설명서의 IAM 정책 생성](#) 을 참조하십시오. IAM

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 조직의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정. 관리형 정책에는 다음이 포함됩니다. AWS 관리형 정책

및 고객 관리형 정책. 관리형 정책과 인라인 정책 중에서 선택하는 방법을 알아보려면 IAM사용 설명서의 [관리형 정책과 인라인 정책 중 선택](#)을 참조하십시오.

## 리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 문서입니다. 리소스 기반 정책의 예로는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 또는 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 사용할 수 없습니다. AWS 리소스 기반 정책의 관리형 정책 IAM

## 액세스 제어 목록 (ACLs)

액세스 제어 목록 (ACLs)은 리소스에 액세스할 수 있는 권한을 가진 주체 (계정 구성원, 사용자 또는 역할)를 제어합니다. ACLs정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 JSON 비슷합니다.

아마존 S3, AWS WAF, VPC Amazon은 지원하는 서비스의 예입니다ACLs. 자세한 내용은 Amazon 심플 스토리지 서비스 개발자 안내서의 [액세스 제어 목록 \(ACL\) 개요](#)를 참조하십시오. ACLs

## 기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 - 권한 경계는 ID 기반 정책이 IAM 엔티티 (IAM사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 자격 증명 기반 정책과 그 권한 경계의 교집합입니다. Principal 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부 허용을 재정의합니다. 권한 경계에 대한 자세한 내용은 사용 IAM설명서의 [IAM 엔티티에 대한 권한 경계를](#) 참조하십시오.
- 서비스 제어 정책 (SCPs) — SCPs 조직 또는 OU (조직 구성 단위)에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations. AWS Organizations 여러 개를 그룹화하고 중앙에서 관리하는 서비스입니다. AWS 계정 귀사가 소유한 것입니다. 조직의 모든 기능을 사용하도록 설정하면 일부 또는 모든 계정에 서비스 제어 정책 (SCPs)을 적용할 수 있습니다. 각 항목을 포함하여 구성원 계정의 엔티티에 대한 권한을 SCP 제한합니다. AWS 계정 루트 사용자. Organizations 및 SCPs에 대한 자세한 내용은 의 [서비스 제어 정책을](#) 참조하십시오. AWS Organizations 사용 설명서.

- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 보안 인증 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 내용은 IAM사용 설명서의 [세션 정책을](#) 참조하십시오.

## 여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 방법을 알아보려면 AWS 여러 정책 유형이 관련된 경우 요청을 허용할지 여부를 결정하려면 IAM사용 설명서의 [정책 평가 로직](#)을 참조하십시오.

## 수정할 수 있다면 방법이 무엇입니까? AWS App Mesh 다음과 함께 작동합니다. IAM

를 사용하여 App Mesh에 대한 액세스를 IAM 관리하려면 먼저 App Mesh에서 사용할 수 있는 IAM 기능을 이해해야 합니다. App Mesh와 기타 방법을 개괄적으로 살펴보려면 AWS 서비스가 다음과 IAM 같이 작동합니다. [AWSIAM사용 설명서에 IAM](#) 나와 있는 서비스.

### 주제

- [App Mesh 자격 증명 기반 정책](#)
- [App Mesh 리소스 기반 정책](#)
- [App Mesh 태그 기반 권한 부여](#)
- [App Mesh의 IAM 역할](#)

## App Mesh 자격 증명 기반 정책

IAMID 기반 정책을 사용하면 허용 또는 거부된 작업 및 리소스는 물론 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. App Mesh는 특정 작업, 리소스 및 조건 키를 지원합니다. JSON정책에서 사용하는 모든 요소에 대해 알아보려면 사용 IAM설명서의 IAM JSON [정책 요소 참조](#)를 참조하십시오.

### 작업

관리자는 다음을 사용할 수 있습니다. AWS JSON정책을 통해 누가 무엇에 액세스할 수 있는지 지정합니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

정책 Action 요소는 JSON 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 일반적으로 정책 조치의 이름은 관련 조치와 동일합니다. AWS API 오퍼레이션. 일치하는 작업이 없는 권한 전용 작업과 같은 몇 가지 예외가 있습니다. API 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함합니다.

App Mesh의 정책 작업은 작업 앞에 접두사 `appmesh:`를 사용합니다. 예를 들어, 작업과 함께 계정의 메시지를 나열할 수 있는 권한을 다른 사람에게 부여하려면 해당 `appmesh:ListMeshes` API 작업을 해당 사용자의 정책에 포함해야 합니다 `appmesh:ListMeshes`. 정책 문에는 Action 또는 NotAction 요소가 포함되어야 합니다.

명령문 하나에 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
 "appmesh:ListMeshes",
 "appmesh:ListVirtualNodes"
]
```

와일드카드(\*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, Describe라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "appmesh:Describe*"
```

App Mesh 작업 목록을 보려면 다음에 정의된 [작업을 참조하십시오. AWS App Mesh](#)(출처: IAM 사용 설명서).

## 리소스

관리자는 다음을 사용할 수 있습니다. AWS JSON 정책을 통해 누가 무엇에 액세스할 수 있는지 지정합니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

ResourceJSON 정책 요소는 작업이 적용되는 하나 또는 여러 개의 객체를 지정합니다. 문장에는 Resource 또는 NotResource 요소가 반드시 추가되어야 합니다. [Amazon 리소스 이름 \(ARN\)](#) 을 사용하여 리소스를 지정하는 것이 가장 좋습니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이 태스크를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(\*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

App Mesh mesh 리소스에는 다음이 포함됩니다ARN.

```
arn:${Partition}:appmesh:${Region}:${Account}:mesh/${MeshName}
```

형식에 대한 자세한 내용은 [Amazon 리소스 이름 \(ARNs\) 을 참조하십시오. ARNs AWS 서비스 네임스페이스.](#)

예를 들어, 이름이 지정된 메시를 지정하려면 *apps* 에서 *Region-code* 명세서의 지역에는 다음을 사용하십시오ARN.

```
arn:aws:appmesh:Region-code:111122223333:mesh/apps
```

특정 계정에 속하는 모든 인스턴스를 지정하려면 와일드카드(\*)를 사용합니다.

```
"Resource": "arn:aws:appmesh:Region-code:111122223333:mesh/*"
```

리소스를 생성하기 위한 작업과 같은 일부 App Mesh 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(\*)를 사용해야 합니다.

```
"Resource": "*"
```

대부분의 App Mesh API 작업에는 여러 리소스가 포함됩니다. 예를 들어, 가상 노드 대상으로 경로를 CreateRoute 생성하므로 IAM 사용자에게 경로와 가상 노드를 사용할 수 있는 권한이 있어야 합니다. 명령문 하나에 여러 리소스를 지정하려면 ARNs 쉼표로 구분하십시오.

```
"Resource": [
 "arn:aws:appmesh:Region-code:111122223333:mesh/apps/virtualRouter/serviceB/route/*",
 "arn:aws:appmesh:Region-code:111122223333:mesh/apps/virtualNode/serviceB"
]
```

App Mesh 리소스 유형 및 해당 유형의 목록을 ARNs 보려면 [리소스 정의 기준을 참조하십시오. AWS App Mesh](#)(출처: IAM 사용 설명서). 각 리소스의 어떤 작업을 지정할 수 있는지 알아보려면 정의된 [작업을 참조하십시오. ARN AWS App Mesh](#).

## 조건 키

App Mesh는 일부 전역 조건 키를 사용하도록 지원합니다. 모두 보려면 AWS 글로벌 조건 키는 다음을 참조하십시오. [AWS IAM 사용 설명서의 글로벌 조건 컨텍스트 키](#). App Mesh가 지원하는 글로벌 조건 키 목록을 보려면 다음 [조건 키를 참조하십시오. AWS App Mesh](#)(출처: IAM 사용 설명서). 조건 키와 함께 사용할 수 있는 작업 및 리소스를 알아보려면 정의된 [작업을 참조하십시오. AWS App Mesh](#).

## 예시

App Mesh 자격 증명 기반 정책의 예를 보려면 [AWS App Mesh ID 기반 정책 예제](#) 섹션을 참조하세요.

## App Mesh 리소스 기반 정책

App Mesh는 리소스 기반 정책을 지원하지 않습니다. 하지만 사용하는 경우 AWS Resource Access Manager (AWS RAM) 서비스를 통해 메시지를 공유할 수 있습니다. AWS 서비스, 리소스 기반 정책은 다음을 통해 메시지에 적용됩니다. AWS RAM 서비스. 자세한 내용은 [메시에 권한 부여](#) 단원을 참조하십시오.

## App Mesh 태그 기반 권한 부여

태그를 App Mesh 리소스에 연결하거나 요청을 통해 태그를 App Mesh에 전달할 수 있습니다. 태그를 기준으로 액세스를 제어하려면 `appmesh:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 [조건 요소](#)에 태그 정보를 제공합니다. App Mesh 리소스에 태그를 지정하는 방법에 대한 자세한 내용은 [태깅을 참조하십시오. AWS 리소스](#).

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 [제한된 태그를 사용하여 App Mesh 메시 생성](#)에서 확인할 수 있습니다.

## App Mesh의 IAM 역할

[IAM 역할](#)은 사용자 내부의 독립체입니다. AWS 특정 권한이 있는 계정.

### App Mesh에서 임시 자격 증명 사용

임시 자격 증명을 사용하여 페더레이션으로 로그인하거나, 역할을 수입하거나, 계정 간 IAM 역할을 수입할 수 있습니다. 를 호출하여 임시 보안 자격 증명을 얻을 수 있습니다. AWS STS API [AssumeRole](#) 또는 와 같은 작업 [GetFederationToken](#).

App Mesh는 임시 자격 증명 사용을 지원합니다.

## 서비스 연결 역할

[서비스 연결 역할은 다음을 허용합니다](#). AWS 서비스는 다른 서비스의 리소스에 액세스하여 사용자를 대신하여 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 표시되며 서비스에서 소유합니다. IAM관리자는 서비스 연결 역할에 대한 권한을 볼 수 있지만 편집할 수는 없습니다.

App Mesh에서는 서비스 연결 역할을 지원합니다. App Mesh 서비스 연결 역할을 생성 또는 관리하는 방법에 대한 자세한 내용은 [App Mesh에 서비스 연결 역할 사용](#) 섹션을 참조하세요.

## 서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 [서비스 역할](#)을 수입할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 계정에 표시되며 해당 IAM 계정에서 소유합니다. 즉, IAM 관리자는 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

App Mesh는 서비스 역할을 지원하지 않습니다.

## AWS App Mesh ID 기반 정책 예제

기본적으로 IAM 사용자와 역할에는 App Mesh 리소스를 만들거나 수정할 권한이 없습니다. 또한 다음을 사용하여 작업을 수행할 수 없습니다. AWS Management Console, AWS CLI, 또는 AWS API. IAM 관리자는 필요한 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자 및 역할에 부여하는 IAM 정책을 만들어야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 해당 정책을 연결해야 합니다.

이 예제 JSON 정책 문서를 사용하여 IAM ID 기반 [정책을 만드는 방법을 알아보려면 사용 IAM설명서의 JSON 탭에서 정책 생성을](#) 참조하십시오.

### 주제

- [정책 모범 사례](#)
- [App Mesh 콘솔 사용](#)
- [사용자가 자신의 고유한 권한을 볼 수 있도록 허용](#)
- [메시 생성](#)
- [모든 메시 나열 및 설명](#)
- [제한된 태그를 사용하여 App Mesh 메시 생성](#)

## 정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 App Mesh 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이러한 조치로 인해 비용이 발생할 수 있습니다. AWS 계정. ID 기반 정책을 만들거나 편집할 때는 다음 지침 및 권장 사항을 따르십시오.

- 시작해 보세요. AWS 관리형 정책 및 최소 권한 권한으로의 이동 — 사용자와 워크로드에 권한 부여를 시작하려면 다음을 사용하십시오. AWS 여러 일반 사용 사례에 대한 권한을 부여하는 관리형 정책. 다음 사이트에서 사용할 수 있습니다. AWS 계정. 를 정의하여 권한을 더 줄이는 것이 좋습니다. AWS 사용 사례에 맞는 고객 관리형 정책. 자세한 내용은 [단원을 참조하세요.AWS 관리형 정책](#) 또는 [AWSIAM사용자 가이드의 작업 기능에](#) 대한 관리형 정책.
- 최소 권한 적용 — IAM 정책으로 권한을 설정하는 경우 작업 수행에 필요한 권한만 부여하십시오. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. 를 사용하여 권한을 IAM 적용하는 방법에 대한 자세한 내용은 사용 [설명서의 정책 및 권한을](#) 참조하십시오. IAM IAM
- IAM정책의 조건을 사용하여 액세스를 더욱 제한할 수 있습니다. - 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, 를 사용하여 모든 요청을 전송하도록 지정하는 정책 조건을 작성할 수 SSL 있습니다. 특정 작업을 통해 서비스 작업을 사용하는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. AWS 서비스예: AWS CloudFormation. 자세한 내용은 IAM사용 설명서의 [IAMJSON정책 요소: 조건을](#) 참조하십시오.
- IAMAccess Analyzer를 사용하여 IAM 정책을 검증하여 안전하고 기능적인 권한을 보장합니다. IAM Access Analyzer는 새 정책과 기존 정책을 검증하여 정책이 IAM 정책 언어 (JSON) 및 IAM 모범 사례를 준수하는지 확인합니다. IAMAccess Analyzer는 안전하고 기능적인 정책을 작성하는 데 도움이 되는 100개 이상의 정책 검사와 실행 가능한 권장 사항을 제공합니다. 자세한 내용은 사용 설명서의 [IAMAccess Analyzer 정책 검증을](#) 참조하십시오. IAM
- 다단계 인증 필요 (MFA) - 사용자 또는 루트 IAM 사용자가 필요한 시나리오가 있는 경우 AWS 계정 보안을 강화하려면 MFA 켜십시오. API작업 호출 MFA 시기를 요구하려면 정책에 MFA 조건을 추가하십시오. 자세한 내용은 IAM사용 설명서의 MFA [-보호된 API 액세스 구성을](#) 참조하십시오.

의 모범 사례에 IAM 대한 자세한 내용은 IAM사용 설명서의 [보안 모범 사례를](#) 참조하십시오. IAM

## App Mesh 콘솔 사용

액세스하려면 AWS App Mesh 콘솔에는 최소 권한 집합이 있어야 합니다. 이러한 권한을 통해 내 App Mesh 리소스의 세부 정보를 나열하고 볼 수 있어야 합니다. AWS 계정. 필요한 최소 권한보다 더 제한적인 ID 기반 정책을 만들면 해당 정책을 사용하는 엔티티 (IAM사용자 또는 역할) 에 대해 콘솔이 의도

한 대로 작동하지 않습니다. [AWSAppMeshReadOnly](#) 관리형 정책을 사용자에게 연결할 수 있습니다. 자세한 내용은 [사용 설명서의 사용자에게 권한 추가를 참조하십시오](#). IAM

전화만 거는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. AWS CLI 또는 AWS API. 대신 수행하려는 작업과 일치하는 API 작업에만 액세스를 허용하세요.

## 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제에서는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 하는 정책을 만드는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 다음을 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다. AWS CLI 또는 AWS API.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "ViewOwnUserInfo",
 "Effect": "Allow",
 "Action": [
 "iam:GetUserPolicy",
 "iam:ListGroupForUser",
 "iam:ListAttachedUserPolicies",
 "iam:ListUserPolicies",
 "iam:GetUser"
],
 "Resource": ["arn:aws:iam::*:user/${aws:username}"]
 },
 {
 "Sid": "NavigateInConsole",
 "Effect": "Allow",
 "Action": [
 "iam:GetGroupPolicy",
 "iam:GetPolicyVersion",
 "iam:GetPolicy",
 "iam:ListAttachedGroupPolicies",
 "iam:ListGroupPolicies",
 "iam:ListPolicyVersions",
 "iam:ListPolicies",
 "iam:ListUsers"
],
 "Resource": "*"
 }
]
}
```

```
}

```

## 메시 생성

이 예제는 모든 리전에서 사용자가 계정의 메시 생성을 허용하는 정책을 생성하는 방법을 보여 줍니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "appmesh:CreateMesh",
 "Resource": "arn:aws:appmesh:*:123456789012:CreateMesh"
 }
]
}
```

## 모든 메시 나열 및 설명

이 예제는 사용자에게 모든 메시지를 나열하고 설명하기 위한 읽기 전용 액세스를 허용하는 정책을 생성하는 방법을 보여 줍니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "appmesh:DescribeMesh",
 "appmesh:ListMeshes"
],
 "Resource": "*"
 }
]
}
```

## 제한된 태그를 사용하여 App Mesh 메시 생성

IAM정책에서 태그를 사용하여 IAM 요청에 전달할 수 있는 태그를 제어할 수 있습니다. IAM사용자 또는 역할에 추가, 변경 또는 제거할 수 있는 태그 키-값 쌍을 지정할 수 있습니다. 이 예제에서는 이

름이 지정된 태그로 메시지를 생성한 경우에만 메시지 생성을 허용하는 정책을 만드는 방법을 보여줍니다. `teamName` 그리고 값은 `booksTeam`.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "appmesh:CreateMesh",
 "Resource": "*",
 "Condition": {
 "StringEquals": {
 "aws:RequestTag/teamName": "booksTeam"
 }
 }
 }
]
}
```

계정의 IAM 사용자에게 이 정책을 첨부할 수 있습니다. 사용자가 메시지를 생성하려는 경우 메시지는 이름이 `teamName`이고 값이 `booksTeam`인 태그를 포함해야 합니다. 메시지에 이 태그와 값이 포함되어 있지 않으면 메시지 생성 시도가 실패합니다. 자세한 내용은 IAM사용 설명서의 [IAMJSON정책 요소: 조건](#)을 참조하십시오.

## AWS App Mesh에 대한 관리형 정책

AWS 관리형 정책은 에서 생성하고 관리하는 독립형 정책입니다. AWS AWS 관리형 정책은 많은 일반 사용 사례에 대한 권한을 제공하도록 설계되었으므로 사용자, 그룹 및 역할에 권한을 할당하기 시작할 수 있습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수도 있다는 점에 유의하세요. 사용 사례에 고유한 [고객 관리형 정책](#)을 정의하여 권한을 줄이는 것이 좋습니다.

관리형 정책에 정의된 권한은 변경할 수 없습니다. AWS AWS 관리형 정책에 정의된 권한을 업데이트 하는 경우 AWS 해당 업데이트는 정책이 연결된 모든 주체 ID (사용자, 그룹, 역할) 에 영향을 미칩니다. AWS 새 API 작업이 시작되거나 기존 서비스에 새 AWS 서비스 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 가장 높습니다.

자세한 내용은 IAM 사용자 설명서의 [AWS 관리형 정책](#)을 참조하세요.

## AWS 관리형 정책: AWSAppMeshServiceRolePolicy

AWSAppMeshServiceRolePolicy를 IAM 엔터티에 연결할 수 있습니다. 에서 사용하거나 관리하는 AWS 서비스 및 리소스에 액세스할 수 AWS App Mesh있습니다.

이 정책에 대한 권한을 보려면 AWS 관리형 정책 [AWSAppMeshServiceRolePolicy](#)참조의 내용을 참조하십시오.

AWSAppMeshServiceRolePolicy의 권한 세부 정보에 대한 자세한 내용은 [App Mesh의 서비스 연결 역할 권한](#)을 참조하세요.

## AWS 관리형 정책: AWSAppMeshEnvoyAccess

AWSAppMeshEnvoyAccess를 IAM 엔터티에 연결할 수 있습니다. 가상 노드 구성에 액세스하기 위한 App Mesh Envoy 정책입니다.

이 정책에 대한 권한을 보려면 AWS 관리형 정책 [AWSAppMeshEnvoyAccess](#)참조의 내용을 참조하십시오.

## AWS 관리형 정책: AWSAppMeshFullAccess

AWSAppMeshFullAccess를 IAM 엔터티에 연결할 수 있습니다. AWS App Mesh API 및 AWS Management Console에 대한 전체 액세스 권한을 제공합니다.

이 정책에 대한 권한을 보려면 AWS 관리형 정책 참조를 참조하십시오 [AWSAppMeshFullAccess](#).

## AWS 관리형 정책: AWSAppMeshPreviewEnvoyAccess

AWSAppMeshPreviewEnvoyAccess를 IAM 엔터티에 연결할 수 있습니다. 가상 노드 구성에 액세스하기 위한 App Mesh Preview Envoy 정책입니다.

이 정책에 대한 권한을 보려면 AWS 관리형 정책 [AWSAppMeshPreviewEnvoyAccess](#)참조의 내용을 참조하십시오.

## AWS 관리형 정책: AWSAppMeshPreviewServiceRolePolicy

AWSAppMeshPreviewServiceRolePolicy를 IAM 엔터티에 연결할 수 있습니다. 에서 사용하거나 관리하는 AWS 서비스 및 리소스에 액세스할 수 AWS App Mesh있습니다.

이 정책에 대한 권한을 보려면 AWS 관리형 정책 [AWSAppMeshPreviewServiceRolePolicy](#)참조의 내용을 참조하십시오.

## AWS 관리형 정책: AWSAppMeshReadOnly

AWSAppMeshReadOnly를 IAM 엔터티에 연결할 수 있습니다. AWS App Mesh API 및 에 대한 읽기 전용 액세스를 제공합니다. AWS Management Console

이 정책에 대한 권한을 보려면 AWS 관리형 정책 참조를 참조하십시오 [AWSAppMeshReadOnly](#).

## AWS App MeshAWS 관리형 정책 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 AWS App Mesh 이후의 AWS 관리형 정책 업데이트에 대한 세부 정보를 볼 수 있습니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 AWS App Mesh 문서 기록 페이지에서 RSS 피드를 구독합니다.

| 변경 사항                                                                                                   | 설명                                                                                                                         | 날짜            |
|---------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|---------------|
| <a href="#">AWSAppMeshFullAccess</a> — 정책이 업데이트되었습니다.                                                   | TagResource 및 UntagResource API에 대한 액세스를 AWSAppMeshFullAccess 허용하도록 업데이트되었습니다.                                             | 2024년 4월 24일  |
| <a href="#">AWSAppMeshServiceRolePolicy</a> , <a href="#">AWSServiceRoleForAppMesh</a> — 정책이 업데이트되었습니다. | AWSServiceRoleForAppMesh AWS Cloud Map DiscoverInstancesRevision API에 대한 액세스를 AWSAppMeshServiceRolePolicy 허용하도록 업데이트되었습니다. | 2023년 10월 12일 |

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- AWS IAM Identity Center다음 지역의 사용자 및 그룹:

권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 [권한 세트 생성](#)의 지침을 따르세요.

- ID 제공자를 통해 IAM에서 관리되는 사용자:

ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 [서드 파티 자격 증명 공급자의 역할 만들기\(연합\)](#)의 지침을 따르세요.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 [IAM 사용자의 역할 생성](#)의 지침을 따르세요.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 [사용자\(콘솔\)에 권한 추가](#)의 지침을 따르세요.

## App Mesh에 서비스 연결 역할 사용

AWS App Mesh은 AWS Identity and Access Management(IAM) [서비스 연결 역할](#)을 사용합니다. 서비스 연결 역할은 App Mesh에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 App Mesh에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 App Mesh를 더 쉽게 설정할 수 있습니다. App Mesh에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, App Mesh만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며, 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제해야만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 App Mesh 리소스에 대한 액세스 권한을 부주의로 삭제할 수 없기 때문에 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조해 서비스 연결 역할(Service-Linked Role) 열이 예(Yes)인 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

### App Mesh에 대한 서비스 연결 역할 권한

App Mesh는 AWSServiceRoleForAppMesh라는 서비스 연결 역할을 사용합니다. 이 서비스 연결 역할을 사용하여 사용자를 대신해서 AWS 서비스를 호출할 수 있습니다.

AWSServiceRoleForAppMesh 서비스 연결 역할은 appmesh.amazonaws.com 서비스를 신뢰하여 역할을 맡습니다.

### 권한 세부 정보

- `servicediscovery:DiscoverInstances` - App Mesh가 모든 AWS 리소스에 대한 작업을 완료할 수 있도록 합니다.
- `servicediscovery:DiscoverInstancesRevision` - App Mesh가 모든 AWS 리소스에 대한 작업을 완료할 수 있도록 합니다.

## AWSServiceRoleForAppMesh

이 정책에는 다음 권한이 포함되어 있습니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "CloudMapServiceDiscovery",
 "Effect": "Allow",
 "Action": [
 "servicediscovery:DiscoverInstances",
 "servicediscovery:DiscoverInstancesRevision"
],
 "Resource": "*"
 },
 {
 "Sid": "ACMCertificateVerification",
 "Effect": "Allow",
 "Action": [
 "acm:DescribeCertificate"
],
 "Resource": "*"
 }
]
}
```

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 연결 역할을 작성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#) 섹션을 참조하세요.

## App Mesh에 대한 서비스 연결 역할 생성

2019년 6월 5일 이후에 AWS Management Console, AWS CLI 또는 AWS API에서 메시를 생성한 경우, App Mesh가 서비스 연결 역할을 자동으로 생성합니다. 서비스 연결 역할을 자동으로 생성하려면 메시를 생성하는 데 사용한 IAM 계정에 [AWSAppMeshFullAccess](#) IAM 정책이 연결되어 있거나 `iam:CreateServiceLinkedRole` 권한이 포함된 역할에 정책이 연결되어 있어야 합니다. 이 서비스

연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 메시지를 생성할 때 App Mesh에서는 서비스 연결 역할을 다시 생성합니다. 계정에 2019년 6월 5일 이전에 생성된 메시만 포함되어 있고 해당 메시에 서비스 연결 역할을 사용하려는 경우 IAM 콘솔을 사용하여 역할을 생성할 수 있습니다.

IAM 콘솔을 사용하여 App Mesh 사용 사례에서 서비스 연결 역할을 생성할 수 있습니다. AWS CLI 또는 AWS API에서 `appmesh.amazonaws.com` 서비스 이름의 서비스 연결 역할을 생성합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 생성](#) 섹션을 참조하세요. 이 서비스 연결 역할을 삭제한 후에는 동일한 프로세스를 사용하여 역할을 다시 생성할 수 있습니다.

## App Mesh에 대한 서비스 연결 역할 편집

App Mesh는 `AWSServiceRoleForAppMesh` 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 연결 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

## App Mesh에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제할 것을 권장합니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 개체가 없도록 합니다. 단, 서비스 연결 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

### Note

리소스를 삭제하려 할 때 App Mesh 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

`AWSServiceRoleForAppMesh`에서 사용하는 App Mesh 리소스를 삭제하려면

1. 메시의 모든 라우터에 대해 정의된 모든 [경로](#)를 삭제합니다.
2. 메시의 모든 [가상 라우터](#)를 삭제합니다.
3. 메시의 모든 [가상 서비스](#)를 삭제합니다.
4. 메시의 모든 [가상 노드](#)를 삭제합니다.
5. [메시](#)를 삭제합니다.

계정의 모든 메시에 대해 이전 단계를 완료하세요.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 AWSServiceRoleForAppMesh 서비스 연결 역할을 삭제합니다. 자세한 내용은 [IAM 사용 설명서](#)의 서비스 연결 역할 삭제 섹션을 참조하세요.

## App Mesh 서비스 연결 역할에 대해 지원되는 리전

App Mesh에서는 서비스를 사용할 수 있는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은 [App Mesh 엔드포인트 및 할당량](#)을 참조하세요.

## Envoy 프록시 권한 부여

프록시 권한 부여는 Amazon ECS 태스크 내에서 실행되거나, Amazon EKS에서 실행되는 Kubernetes 포드에서 실행되거나, Amazon EC2 인스턴스에서 실행되는 [Envoy](#) 프록시가 App Mesh Envoy Management Service에서 하나 이상의 메시 엔드포인트의 구성을 읽을 수 있는 권한을 부여합니다. 2021년 4월 26일 이전에 이미 Envoy가 App Mesh 엔드포인트에 연결된 고객 계정의 경우 [전송 계층 보안](#)(TLS)을 사용하는 가상 노드와 가상 게이트웨이(TLS 사용 여부는 관계없음)에 프록시 권한 부여가 필요합니다. 2021년 4월 26일 이후에 Envoy를 App Mesh 엔드포인트에 연결하려는 고객 계정의 경우 모든 App Mesh 기능에 프록시 권한 부여가 필요합니다. 모든 고객 계정에서 TLS를 사용하지 않더라도 모든 가상 노드에 대해 프록시 인증을 활성화하여 특정 리소스에 대한 권한 부여를 위해 IAM을 사용하여 안전하고 일관된 환경을 구현하는 것이 좋습니다. 프록시 인증을 위해서는 IAM 정책에 `appmesh:StreamAggregatedResources` 권한이 지정되어야 합니다. 이 정책은 IAM 역할에 연결되어야 하고, IAM 역할은 프록시를 호스팅하는 컴퓨팅 리소스에 연결되어야 합니다.

## IAM 정책 생성

서비스 메시의 모든 메시 엔드포인트에서 모든 메시 엔드포인트의 구성을 읽을 수 있게 하려면 [IAM 역할 생성](#)으로 건너뛵니다. 개별 메시 엔드포인트에서 구성을 읽을 수 있는 메시 엔드포인트를 제한하려면 하나 이상의 IAM 정책을 생성해야 합니다. 구성을 읽을 수 있는 메시 엔드포인트를 특정 컴퓨팅 리소스에서 실행되는 Envoy 프록시로만 제한하는 것이 좋습니다. IAM 정책을 생성하고 정책에 `appmesh:StreamAggregatedResources` 권한을 추가합니다. 다음 예제 정책은 이름이 `serviceBv1` 및 `serviceBv2`이고 서비스 메시에서 읽을 수 있는 가상 노드의 구성을 허용합니다. 서비스 메시에 정의된 다른 가상 노드의 구성은 읽을 수 없습니다. IAM 정책 생성 및 편집에 대한 자세한 내용은 [IAM 정책 생성](#) 및 [IAM 정책 편집](#)을 참조하세요.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
```

```

 "Effect": "Allow",
 "Action": "appmesh:StreamAggregatedResources",
 "Resource": [
 "arn:aws:appmesh:us-east-1:123456789012:mesh/app1/virtualNode/
serviceBv1",
 "arn:aws:appmesh:us-east-1:123456789012:mesh/app1/virtualNode/
serviceBv2"
]
 }
]
}

```

각각이 서로 다른 메시 엔드포인트에 대한 액세스를 제한하는 정책을 여러 개 만들 수 있습니다.

## IAM 역할 생성

서비스 메시의 모든 메시 엔드포인트에서 모든 메시 엔드포인트의 구성을 읽을 수 있게 하려면 IAM 역할을 하나만 생성하면 됩니다. 개별 메시 엔드포인트에서 구성을 읽을 수 있는 메시 엔드포인트를 제한하려면 이전 단계에서 생성한 각 정책에 대한 역할을 생성해야 합니다. 프록시가 실행되는 컴퓨팅 리소스에 대한 지침을 완료하세요.

- Amazon EKS — 단일 역할을 사용하려는 경우 클러스터를 생성할 때 생성되어 작업자 노드에 할당된 기존 역할을 사용할 수 있습니다. 여러 역할을 사용하려면 클러스터가 [클러스터의 서비스 계정에 대한 IAM 역할 활성화](#)에 정의된 요구 사항을 충족해야 합니다. IAM 역할을 생성하고 역할을 Kubernetes 서비스 계정과 연결합니다. 자세한 내용은 [서비스 계정에 대한 IAM 역할 및 정책 생성 및 서비스 계정의 IAM 역할 지정](#)을 참조하세요.
- Amazon ECS — AWS 서비스를 선택하고, Elastic Container Service를 선택한 다음, IAM 역할을 생성할 때 Elastic Container Service 태스크 사용 사례를 선택합니다.
- Amazon EC2 – AWS 서비스를 선택하고 EC2를 선택한 다음, IAM 역할을 생성할 때 EC2 사용 사례를 선택합니다. 이 방법은 Amazon EC2 인스턴스에서 직접 프록시를 호스팅하든 인스턴스에서 실행되는 Kubernetes에서 호스팅하든 관계없이 적용됩니다.

IAM 역할 생성 방법에 대한 자세한 내용은 [AWS 서비스에 대한 역할 생성](#)을 참조하세요.

## IAM 정책 연결

서비스 메시의 모든 메시 엔드포인트에서 모든 메시 엔드포인트의 구성을 읽을 수 있게 하려면 [AWSAppMeshEnvoyAccess](#) 관리형 IAM 정책을 이전 단계에서 생성한 IAM 역할에 연결합니다. 개별 메시 엔드포인트에서 구성을 읽을 수 있는 메시 엔드포인트를 제한하려면 생성한 각 정책을 생성한 각

역할에 연결합니다. 사용자 지정 또는 관리형 IAM 정책을 IAM 역할에 연결하는 방법에 대한 자세한 내용은 [IAM ID 권한 추가](#)를 참조하세요.

## IAM 역할 연결

각 IAM 역할을 적절한 컴퓨팅 리소스에 연결합니다.

- Amazon EKS — 작업자 노드에 연결된 역할에 정책을 연결한 경우 이 단계를 건너뛸 수 있습니다. 별도의 역할을 생성한 경우 각 역할을 별도의 Kubernetes 서비스 계정에 할당하고 각 서비스 계정을 Envoy 프록시가 포함된 개별 Kubernetes 포드 배포 사양에 할당합니다. 자세한 내용은 Amazon EKS 사용 설명서의 [서비스 계정에 대한 IAM 역할 지정](#) 및 Kubernetes 설명서의 [포드에 대한 서비스 계정 구성](#)을 참조하세요.
- Amazon ECS — Envoy 프록시가 포함된 태스크 정의에 Amazon ECS 태스크 역할을 연결합니다. 태스크는 EC2 또는 Fargate 시작 유형으로만 배포할 수 있습니다. Amazon ECS 태스크 역할을 생성하여 태스크에 연결하는 방법에 대한 자세한 내용은 [태스크에 대한 IAM 역할 지정](#)을 참조하세요.
- Amazon EC2 — IAM 역할은 Envoy 프록시를 호스팅하는 Amazon EC2 인스턴스에 연결되어야 합니다. Amazon EC2 인스턴스에 역할을 연결하는 방법에 대한 자세한 내용은 [IAM 역할을 생성했으며 이제 해당 역할을 EC2 인스턴스에 할당하려고 함](#)을 참조하세요.

## 권한 확인

컴퓨팅 서비스 이름 중 하나를 선택하여 프록시를 호스팅하는 컴퓨팅 리소스에 `appmesh:StreamAggregatedResources` 권한이 할당되었는지 확인합니다.

### Amazon EKS

사용자 지정 정책은 작업자 노드나 개별 포드 또는 둘 다에 할당된 역할에 할당될 수 있습니다. 하지만 개별 포드에 대한 액세스를 개별 메시 엔드포인트로 제한할 수 있도록 개별 포드에만 정책을 할당하는 것이 좋습니다. 정책이 작업자 노드에 할당된 역할에 연결된 경우 Amazon EC2 탭을 선택하고 작업자 노드 인스턴스에 대한 단계를 완료하세요. Kubernetes 포드에 할당되는 IAM 역할을 확인하려면 다음 단계를 완료하세요.

1. Kubernetes 서비스 계정이 할당되었는지 확인하려는 포드가 포함된 Kubernetes 배포의 세부 정보를 확인하세요. 다음 명령은 이름이 `my-deployment`인 배포에 대한 세부 정보를 확인합니다.

```
kubectl describe deployment my-deployment
```

반환된 출력에서 `Service Account`: 오른쪽에 있는 값을 기록해 둡니다. `Service Account`:로 시작하는 줄이 없는 경우 사용자 지정 Kubernetes 서비스 계정이 현재 배포에 할당되지 않은 것입니다. 이 계정을 하나 할당해야 합니다. 자세한 내용은 Kubernetes 문서의 [포드용 서비스 계정 구성](#)을 참조하세요.

- 이전 단계에서 반환한 서비스 계정의 세부 정보를 확인하세요. 다음 명령을 실행하면 `my-service-account`라는 서비스 계정의 세부 정보가 표시됩니다.

```
kubectl describe serviceaccount my-service-account
```

Kubernetes 서비스 계정이 AWS Identity and Access Management 역할에 연결되어 있는 경우 반환되는 줄 중 하나는 다음 예와 비슷합니다.

```
Annotations: eks.amazonaws.com/role-arn=arn:aws:iam::123456789012:role/my-deployment
```

이전 예제 `my-deployment`는 서비스 계정과 연결된 IAM 역할의 이름입니다. 서비스 계정 출력에 위 예제와 비슷한 줄이 없으면 Kubernetes 서비스 계정이 AWS Identity and Access Management 계정에 연결되어 있지 않으므로 계정에 연결해야 합니다. 자세한 내용은 [서비스 계정을 위한 IAM 역할 지정](#)을 참조하세요.

- AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- 왼쪽 탐색 창에서 역할을 선택합니다. 이전 단계에서 적어 놓은 IAM 역할의 이름을 선택합니다.
- 이전에 생성한 사용자 지정 정책 또는 [AWSAppMeshEnvoyAccess](#) 관리형 정책이 나열되어 있는지 확인합니다. 두 정책이 모두 연결되지 않은 경우 [IAM 정책을 IAM 역할에 연결](#)합니다. 사용자 지정 IAM 정책을 연결하려고 하는데 이 정책이 아직 없는 경우에는 필요한 권한이 있는 [사용자 지정 IAM 정책을 생성](#)해야 합니다. 사용자 지정 IAM 정책이 연결된 경우 해당 정책을 선택하고 정책에 "Action": "appmesh:StreamAggregatedResources"가 포함되어 있는지 확인합니다. 그렇지 않은 경우 사용자 지정 IAM 정책에 해당 권한을 추가해야 합니다. 또한 특정 메시 엔드포인트에 대한 적절한 Amazon 리소스 이름(ARN)이 나열되어 있는지 확인할 수 있습니다. 나열된 ARN이 없는 경우 정책을 편집하여 나열된 ARN을 추가, 제거 또는 변경할 수 있습니다. 자세한 내용은 [IAM 정책 편집](#) 및 [IAM 정책 생성](#) 섹션을 참조하세요.
- Envoy 프록시가 포함된 각 Kubernetes 포드에 대해 이전 단계를 반복합니다.

## Amazon ECS

1. Amazon ECS 콘솔에서 태스크 정의를 선택합니다.
2. Amazon ECS 태스크를 선택합니다.
3. 태스크 정의 이름 페이지에서 태스크 정의를 선택합니다.
4. 태스크 정의 페이지에서 태스크 역할 오른쪽에 있는 IAM 역할 이름의 링크를 선택합니다. IAM 역할이 목록에 없는 경우 [IAM 역할을 생성](#)하고 [태스크 정의를 업데이트](#)하여 태스크에 연결해야 합니다.
5. 요약 페이지의 권한 탭에서 이전에 생성한 사용자 지정 정책 또는 [AWSAppMeshEnvoyAccess](#) 관리형 정책이 나열되어 있는지 확인합니다. 두 정책이 모두 연결되지 않은 경우 [IAM 정책을 IAM 역할에 연결](#)합니다. 사용자 지정 IAM 정책을 연결하려고 하는데 이 정책이 아직 없는 경우에는 [사용자 지정 IAM 정책을 생성](#)해야 합니다. 사용자 지정 IAM 정책이 연결된 경우 해당 정책을 선택하고 정책에 "Action": "appmesh:StreamAggregatedResources"가 포함되어 있는지 확인합니다. 그렇지 않은 경우 사용자 지정 IAM 정책에 해당 권한을 추가해야 합니다. 또한 특정 메시 엔드포인트에 대한 적절한 Amazon 리소스 이름(ARN)이 나열되어 있는지 확인할 수 있습니다. 나열된 ARN이 없는 경우 정책을 편집하여 나열된 ARN을 추가, 제거 또는 변경할 수 있습니다. 자세한 내용은 [IAM 정책 편집](#) 및 [IAM 정책 생성](#) 섹션을 참조하세요.
6. Envoy 프록시가 포함된 각 태스크 정의에 대해 이전 단계를 반복합니다.

## Amazon EC2

1. Amazon EC2 콘솔의 왼쪽 탐색 창에서 인스턴스를 선택합니다.
2. Envoy 프록시를 호스팅하는 인스턴스 중 하나를 선택합니다.
3. 설명 탭에서 IAM 역할 오른쪽에 있는 IAM 역할 이름 링크를 선택합니다. IAM 역할이 목록에 없는 경우 [IAM 역할을 생성](#)해야 합니다.
4. 요약 페이지의 권한 탭에서 이전에 생성한 사용자 지정 정책 또는 [AWSAppMeshEnvoyAccess](#) 관리형 정책이 나열되어 있는지 확인합니다. 두 정책이 모두 연결되지 않은 경우 [IAM 정책을 IAM 역할에 연결](#)합니다. 사용자 지정 IAM 정책을 연결하려고 하는데 이 정책이 아직 없는 경우에는 [사용자 지정 IAM 정책을 생성](#)해야 합니다. 사용자 지정 IAM 정책이 연결된 경우 해당 정책을 선택하고 정책에 "Action": "appmesh:StreamAggregatedResources"가 포함되어 있는지 확인합니다. 그렇지 않은 경우 사용자 지정 IAM 정책에 해당 권한을 추가해야 합니다. 또한 특정 메시 엔드포인트에 대한 적절한 Amazon 리소스 이름(ARN)이 나열되어 있는지 확인할 수 있습니다. 나열된 ARN이 없는 경우 정책을 편집하여 나열된 ARN을 추가, 제거 또는 변경할 수 있습니다. 자세한 내용은 [IAM 정책 편집](#) 및 [IAM 정책 생성](#) 섹션을 참조하세요.
5. Envoy 프록시를 호스팅하는 각 인스턴스에 대해 이전 단계를 반복합니다.

## 문제 해결 AWS App Mesh ID 및 액세스

다음 정보를 사용하면 App Mesh 및 작업 시 발생할 수 있는 일반적인 문제를 진단하고 해결하는 데 도움이 IAM 됩니다.

### 주제

- [App Mesh에서 작업을 수행할 권한이 없음](#)
- [제 집 밖에 있는 사람들을 허용하고 싶어요 AWS 내 App Mesh 리소스에 액세스하기 위한 계정](#)

### App Mesh에서 작업을 수행할 권한이 없음

만약 AWS Management Console 작업을 수행할 권한이 없다는 메시지가 표시되면 관리자에게 도움을 요청해야 합니다. 관리자는 로그인 보안 인증 정보를 제공한 사람입니다.

mateojacksonIAM사용자가 콘솔을 사용하여 이름이 지정된 가상 노드를 만들려고 하면 다음 오류가 발생합니다. *my-virtual-node* 라는 이름의 메시지에서 *my-mesh* 하지만 `appmesh:CreateVirtualNode` 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
perform: appmesh:CreateVirtualNode on resource: arn:aws:appmesh:us-
east-1:123456789012:mesh/my-mesh/virtualNode/my-virtual-node
```

이 경우 Mateo는 `appmesh:CreateVirtualNode` 작업을 사용하여 가상 노드 생성을 허용하도록 정책을 업데이트하라고 관리자에게 요청합니다.

#### Note

메시 내에 가상 노드가 생성되므로 Mateo 계정에는 콘솔에서 가상 노드를 생성하기 위한 `appmesh:DescribeMesh` 및 `appmesh:ListMeshes` 작업도 필요합니다.

### 제 집 밖에 있는 사람들을 허용하고 싶어요 AWS 내 App Mesh 리소스에 액세스하기 위한 계정

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록 (ACLs) 을 지원하는 서비스의 경우 해당 정책을 사용하여 사용자에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- App Mesh에서 이러한 기능을 지원하는지 여부를 알아보려면 [수정할 수 있다면 방법이 무엇입니까? AWS App Mesh 다음과 함께 작동합니다. IAM](#) 섹션을 참조하세요.
- 전 세계의 리소스에 대한 액세스를 제공하는 방법을 알아보려면 AWS 계정 소유한 사용자는 다른 IAM 사용자에게 액세스 권한 [제공을 참조하십시오. AWS 계정IAM사용 설명서에 있는 소유권.](#)
- 리소스에 대한 액세스 권한을 제3자에게 제공하는 방법 알아보기 AWS 계정액세스 [제공을 참조하십시오. AWS 계정IAM사용 설명서의](#) 제3자가 소유합니다.
- ID 페더레이션을 통해 액세스를 [제공하는 방법을 알아보려면 사용 설명서의 외부 인증된 사용자에 게 액세스 제공 \(ID 페더레이션\)](#) 을 IAM 참조하십시오.
- 계정 간 액세스에 대한 역할 사용과 리소스 기반 정책 사용의 차이점을 알아보려면 사용 설명서의 계정 간 [리소스 액세스](#)를 참조하십시오. IAM IAM

## 를 사용하여 AWS App Mesh API 호출 로깅 AWS CloudTrail

AWS App Mesh 사용자 [AWS CloudTrail](#), 역할 또는 담당자가 수행한 작업의 기록을 제공하는 서비스와 통합되어 AWS 서비스 있습니다. CloudTrail App Mesh에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 App Mesh 콘솔로부터의 호출과 App Mesh API 작업에 대한 코드 호출이 포함됩니다. 에서 수집한 CloudTrail 정보를 사용하여 App Mesh에 이루어진 요청, 요청이 이루어진 IP 주소, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트 사용자로 했는지 사용자 보안 인증으로 했는지 여부.
- IAM Identity Center 사용자를 대신하여 요청이 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자에게 대한 임시 보안 인증 정보를 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

CloudTrail 계정을 만들 AWS 계정 때 활성화되며 자동으로 CloudTrail 이벤트 기록에 액세스할 수 있습니다. CloudTrail 이벤트 기록은 지난 90일간의 기록된 관리 이벤트를 보고, 검색하고, 다운로드할 수 있고, 변경할 수 없는 기록을 제공합니다. AWS 리전자세한 내용은 사용 설명서의 [CloudTrail 이벤트 기록 사용](#)을 참조하십시오.AWS CloudTrail 이벤트 기록 조회에는 CloudTrail 요금이 부과되지 않습니다.

AWS 계정 지난 90일 동안 진행 중인 이벤트 기록을 보려면 트레일 또는 [CloudTrail호수](#) 이벤트 데이터 저장소를 생성하세요.

## CloudTrail 트레일

트레일을 사용하면 CloudTrail Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 를 사용하여 생성된 모든 트레일은 멀티 AWS Management Console 리전입니다. AWS CLI를 사용하여 단일 리전 또는 다중 리전 추적을 생성할 수 있습니다. 계정의 모든 활동을 기록할 수 있으므로 멀티 리전 트레일을 생성하는 것이 좋습니다 AWS 리전 . 단일 리전 추적을 생성하는 경우 추적의 AWS 리전에 로깅된 이벤트만 볼 수 있습니다. 추적에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 [Creating a trail for your AWS 계정](#) 및 [Creating a trail for an organization](#)을 참조하세요.

트레일을 CloudTrail 생성하여 진행 중인 관리 이벤트의 사본 하나를 Amazon S3 버킷으로 무료로 전송할 수 있지만 Amazon S3 스토리지 요금이 부과됩니다. CloudTrail 요금에 대한 자세한 내용은 [AWS CloudTrail 요금](#)을 참조하십시오. Amazon S3 요금에 대한 자세한 내용은 [Amazon S3 요금](#)을 참조하세요.

## CloudTrail 레이크 이벤트 데이터 스토어

CloudTrail Lake를 사용하면 이벤트에 대한 SQL 기반 쿼리를 실행할 수 있습니다. CloudTrail [Lake](#)는 [행 기반 JSON 형식의 기존 이벤트를 Apache ORC 형식으로 변환합니다](#). ORC는 빠른 데이터 검색에 최적화된 열 기반 스토리지 형식입니다. 이벤트는 이벤트 데이터 스토어로 집계되며, 이벤트 데이터 스토어는 [고급 이벤트 선택기](#)를 적용하여 선택한 기준을 기반으로 하는 변경 불가능한 이벤트 컬렉션입니다. 이벤트 데이터 스토어에 적용하는 선택기는 어떤 이벤트가 지속되고 쿼리할 수 있는지 제어합니다. CloudTrail Lake에 대한 자세한 내용은 사용 설명서의 Lake [사용](#)을 참조하십시오. AWS CloudTrail AWS CloudTrail

CloudTrail Lake 이벤트 데이터 저장 및 쿼리로 인해 비용이 발생합니다. 이벤트 데이터 스토어를 생성할 때 이벤트 데이터 스토어에 사용할 [요금 옵션](#)을 선택합니다. 요금 옵션에 따라 이벤트 모으기 및 저장 비용과 이벤트 데이터 스토어의 기본 및 최대 보존 기간이 결정됩니다. CloudTrail 요금에 대한 자세한 내용은 [AWS CloudTrail 요금](#)을 참조하십시오.

## 의 App Mesh 관리 이벤트 CloudTrail

[관리 이벤트](#)는 내 리소스에서 수행되는 관리 작업에 대한 정보를 제공합니다 AWS 계정. 이를 제어 영역 작업이라고도 합니다. 기본적으로 관리 이벤트를 CloudTrail 기록합니다.

AWS App Mesh 모든 App Mesh 컨트롤 플레인 작업을 관리 이벤트로 기록합니다. App Mesh가 기록하는 AWS App Mesh CloudTrail 컨트롤 플레인 작업의 목록은 [AWS App Mesh API 레퍼런스를](#) 참조하십시오.

## App Mesh 이벤트 예제

이벤트는 모든 소스의 단일 요청을 나타내며 요청된 API 작업, 작업 날짜 및 시간, 요청 파라미터 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 트레이스가 아니므로 이벤트가 특정 순서로 표시되지 않습니다.

다음 예제는 StreamAggregatedResources 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

```
{
 "eventVersion": "1.08",
 "userIdentity": {
 "type": "AssumedRole",
 "principalId": "AKIAIOSFODNN7EXAMPLE:d060be4ac3244e05aca4e067bfe241f8",
 "arn": "arn:aws:sts::123456789012:assumed-role/Application-TaskIamRole-C20GBLBRLBXE/d060be4ac3244e05aca4e067bfe241f8",
 "accountId": "123456789012",
 "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "invokedBy": "appmesh.amazonaws.com"
 },
 "eventTime": "2021-06-09T23:09:46Z",
 "eventSource": "appmesh.amazonaws.com",
 "eventName": "StreamAggregatedResources",
 "awsRegion": "us-west-2",
 "sourceIPAddress": "appmesh.amazonaws.com",
 "userAgent": "appmesh.amazonaws.com",
 "eventID": "e3c6f4ce-EXAMPLE",
 "readOnly": false,
 "eventType": "AwsServiceEvent",
 "managementEvent": true,
 "recipientAccountId": "123456789012",
 "serviceEventDetails": {
 "connectionId": "e3c6f4ce-EXAMPLE",
 "nodeArn": "arn:aws:appmesh:us-west-2:123456789012:mesh/CloudTrail-Test/virtualNode/cloudtrail-test-vn",
 "eventStatus": "ConnectionEstablished",
 "failureReason": ""
 },
 "eventCategory": "Management"
}
```

CloudTrail 레코드 내용에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail [레코드 콘텐츠를 참조하십시오](#).

## 의 데이터 보호 AWS App Mesh

The AWS [공동 책임 모델](#): : 다음과 같은 데이터 보호에 적용됩니다. AWS App Mesh. 이 모델에 설명된 바와 같이 AWS 모든 시스템을 운영하는 글로벌 인프라를 보호하는 책임이 있습니다. AWS 클라우드. 이 인프라에서 호스팅되는 콘텐츠에 대한 통제권을 유지할 책임은 귀하에게 있습니다. 또한 귀하에 대한 보안 구성 및 관리 작업을 담당합니다. AWS 서비스 사용하는 것. 데이터 프라이버시에 대한 자세한 내용은 [데이터 프라이버시를 참조하십시오](#)FAQ. 유럽의 데이터 보호에 대한 자세한 내용은 다음을 참조하십시오. [AWS 공동 책임 모델 및 관련 GDPR](#) 블로그 게시물 AWS 보안 블로그.

데이터 보호를 위해 다음을 보호하는 것이 좋습니다. AWS 계정 자격 증명 및 개별 사용자 설정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM). 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정마다 다단계 인증 (MFA) 을 사용하십시오.
- SSL/를 사용하여 다음과 TLS 통신할 수 있습니다. AWS 있습니다. TLS1.2가 필요하고 TLS 1.3을 권장합니다.
- 다음을 사용하여 사용자 활동 API 로깅을 설정하고 사용자 활동을 기록합니다. AWS CloudTrail. CloudTrail 트레일을 사용하여 캡처하는 방법에 대한 자세한 내용은 AWS 활동에 대한 자세한 내용은 [CloudTrail 트레일 사용을 참조하십시오](#). AWS CloudTrail 사용 설명서.
- 사용 AWS 암호화 솔루션 및 포함된 모든 기본 보안 제어 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 액세스 시 FIPS 140-3개의 검증된 암호화 모듈이 필요한 경우 AWS 명령줄 인터페이스 또는 API an 을 통해 엔드포인트를 사용하십시오. FIPS 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 [연방 정보 처리 표준 \(FIPS\) 140-3](#)을 참조하십시오.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 App Mesh 또는 다른 것으로 작업하는 경우가 포함됩니다. AWS 서비스 콘솔을 사용하여API, AWS CLI, 또는 AWS SDKs. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 제공하는 경우 해당 서버에 대한 요청을 URL 검증하기 위해 자격 증명 정보를 에 포함하지 않는 것이 좋습니다. URL

### 데이터 암호화

App Mesh를 사용하면 데이터가 암호화됩니다.

## 저장 중 암호화

기본적으로 생성한 App Mesh 구성은 유휴 상태에서 암호화됩니다.

## 전송 중 암호화

App Mesh 서비스 엔드포인트는 HTTPS 프로토콜을 사용합니다. Envoy 프록시와 App Mesh Envoy Management Service 간의 모든 통신은 암호화됩니다. Envoy 프록시와 App Mesh Envoy Management Service 간의 통신을 위해 FIPS 호환되는 암호화가 필요한 경우 사용할 수 있는 FIPS 다양한 Envoy 프록시 컨테이너 이미지가 있습니다. 자세한 내용은 [Envoy 이미지](#) 단원을 참조하십시오.

가상 노드 내 컨테이너 간 통신은 암호화되지 않고 이 트래픽은 네트워크 네임스페이스를 벗어나지 않습니다.

## 규정 준수 검증 AWS App Mesh

특정 규정 준수 프로그램의 범위 내에 AWS 서비스 있는지 알아보려면 AWS 서비스 규정 준수 [프로그램의 AWS 서비스 범위별, 규정](#) 참조하여 관심 있는 규정 준수 프로그램을 선택하십시오. 일반 정보는 [AWS 규정 준수 프로그램 AWS 보증 프로그램 규정 AWS](#) 참조하십시오.

를 사용하여 AWS Artifact 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 <https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html> 참조하십시오 AWS Artifact.

사용 시 규정 준수 AWS 서비스 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- [보안 및 규정 준수 킷 스타트 가이드](#) - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 AWS 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- [Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계](#) — 이 백서에서는 기업이 적합한 애플리케이션을 만드는 AWS HIPAA 데 사용할 수 있는 방법을 설명합니다.

### Note

모든 AWS 서비스 사람이 자격이 있는 것은 아닙니다. HIPAA 자세한 내용은 [HIPAA 적격 서비스 참조](#)를 참조하십시오.

- [AWS 규정 AWS 준수 리소스](#) — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.

- [AWS 고객 규정 준수 가이드](#) — 규정 준수의 관점에서 공동 책임 모델을 이해하십시오. 이 가이드에서는 보안을 유지하기 위한 모범 사례를 AWS 서비스 요약하고 여러 프레임워크 (국립 표준 기술 연구소 (NIST), 결제 카드 산업 보안 표준 위원회 (), 국제 표준화 기구 ()) 를 포함한 PCI) 전반의 보안 제어에 대한 지침을 매핑합니다. ISO
- AWS Config 개발자 안내서의 [규칙을 사용하여 리소스 평가](#) — 이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- [AWS Security Hub](#)— 이를 AWS 서비스 통해 내부 AWS보안 상태를 포괄적으로 파악할 수 있습니다. Security Hub는 보안 제어를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 [Security Hub 제어 참조](#)를 참조하세요.
- [Amazon GuardDuty](#) — 환경에 의심스럽고 악의적인 활동이 있는지 AWS 계정모니터링하여 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하는 PCI DSS 등 다양한 규정 준수 요구 사항을 해결하는 데 도움이 될 수 있습니다.
- [AWS Audit Manager](#)— 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위협을 관리하고 규정 및 업계 표준을 준수하는 방법을 단순화할 수 있습니다.

## 의 인프라 보안 AWS App Mesh

관리형 서비스로서 AWS 글로벌 네트워크 보안으로 AWS App Mesh 보호됩니다. AWS 보안 서비스 및 인프라 AWS 보호 방법에 대한 자세한 내용은 [AWS 클라우드 보안을](#) 참조하십시오. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 Security Pillar AWS Well-Architected Framework의 [인프라 보호](#)를 참조하십시오.

AWS 게시된 API 호출을 사용하여 네트워크를 통해 App Mesh에 액세스합니다. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안 (TLS). TLS1.2가 필요하고 TLS 1.3을 권장합니다.
- (임시 디피-헬만) 또는 (타원 곡선 임시 디피-헬만PFS) 와 같이 완벽한 순방향 기밀성 DHE () 을 갖춘 암호 제품군. ECDHE Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 액세스 키 ID와 보안 주체와 연결된 비밀 액세스 키를 사용하여 요청에 서명해야 합니다. IAM 또는 [AWS Security Token Service](#)(AWS STS)를 사용하여 임시 보안 인증을 생성하여 요청에 서명할 수 있습니다.

인터페이스 VPC 엔드포인트를 사용하도록 App Mesh를 VPC 구성하여 보안 상태를 개선할 수 있습니다. 자세한 내용은 [App Mesh 인터페이스 VPC 엔드포인트 \(\)AWS PrivateLink](#) 단원을 참조하십시오.

## App Mesh 인터페이스 VPC 엔드포인트 ()AWS PrivateLink

인터페이스 VPC 엔드포인트를 사용하도록 Amazon ECS를 구성하여 Amazon VPC의 보안 상태를 향상시킬 수 있습니다. 인터페이스 엔드포인트는 프라이빗 IP 주소를 사용하여 App Mesh API에 비공개로 액세스할 수 있는 기술인 에 의해 AWS PrivateLink 구동됩니다. PrivateLinkAmazon VPC와 App Mesh 사이의 모든 네트워크 트래픽을 아마존 네트워크로 제한합니다.

반드시 구성할 필요는 PrivateLink 없지만 구성하는 것이 좋습니다. VPC 엔드포인트에 대한 자세한 내용 PrivateLink 및 인터페이스는 VPC 엔드포인트를 통한 서비스 [액세스](#)를 참조하십시오. AWS PrivateLink

### App Mesh 인터페이스 VPC 엔드포인트에 대한 고려 사항

App Mesh용 인터페이스 VPC 엔드포인트를 설정하기 전에 다음 사항을 고려하세요.

- Amazon VPC에 인터넷 게이트웨이가 없고 작업에서 awslogs 로그 드라이버를 사용하여 로그 정보를 CloudWatch Logs로 보내는 경우 Logs용 인터페이스 VPC 엔드포인트를 생성해야 합니다. CloudWatch 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 인터페이스 VPC 엔드포인트가 있는 CloudWatch 로그 [사용](#)을 참조하십시오.
- VPC 엔드포인트는 리전 간 요청을 지원하지 AWS 않습니다. App Mesh에 대해 API 호출을 실행하려는 동일한 리전에서 엔드포인트를 생성해야 합니다.
- VPC 엔드포인트는 Amazon Route 53을 통해 Amazon이 제공하는 DNS만 지원합니다. 자신의 DNS를 사용하는 경우에는 조건적인 DNS 전송을 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [DHCP 옵션 세트](#)를 참조하십시오.
- VPC 엔드포인트에 연결된 보안 그룹은 Amazon VPC의 프라이빗 서브넷에서 443 포트로 들어오는 연결을 허용해야 합니다.

#### Note

Envoy 연결의 경우 엔드포인트 정책을 VPC 엔드포인트에 연결하여 App Mesh에 대한 액세스를 제어하는 방법(예: 서비스 이름 `com.amazonaws.Region.appmesh-envoy-management` 사용)은 지원되지 않습니다.

추가 고려 사항 및 제한 사항은 [인터페이스 엔드포인트 가용 영역 고려 사항](#) 및 [인터페이스 엔드포인트 속성 및 제한 사항](#)을 참조하세요.

## App Mesh를 위한 인터페이스 VPC 엔드포인트 생성

App Mesh 서비스에 대한 인터페이스 VPC 엔드포인트를 생성하려면 Amazon VPC 사용 설명서의 [인터페이스 엔드포인트 생성](#) 절차를 사용합니다. 메시 작업을 수행하기 위해 App Mesh의 퍼블릭 Envoy Management Service 및 `com.amazonaws.Region.appmesh`에 연결하려면 Envoy 프록시의 서비스 이름으로 `com.amazonaws.Region.appmesh-envoy-management`를 지정합니다.

### Note

### 미국 동부 (오하이오) AWS 지역과 같이 `us-east-2` App Mesh에서 지원하는 지역의 지역 식별자를 나타냅니다.

App Mesh가 지원되는 모든 리전에서 App Mesh에 대한 인터페이스 VPC 엔드포인트를 정의할 수 있지만, 각 리전의 모든 가용 영역에 대해 엔드포인트를 정의하지는 못할 수 있습니다. 리전의 인터페이스 VPC 엔드포인트에서 지원되는 가용 영역을 알아보려면 [describe-vpc-endpoint-services](#) 명령을 사용하거나 `aws`를 사용하십시오. AWS Management Console 예를 들어 다음 명령은 미국 동부(오하이오) 리전 내에 App Mesh 인터페이스 VPC 엔드포인트를 배포할 수 있는 가용 영역을 반환합니다.

```
aws --region us-east-2 ec2 describe-vpc-endpoint-services --query 'ServiceDetails[?ServiceName==`com.amazonaws.us-east-2.appmesh-envoy-management`].AvailabilityZones[]'
```

```
aws --region us-east-2 ec2 describe-vpc-endpoint-services --query 'ServiceDetails[?ServiceName==`com.amazonaws.us-east-2.appmesh`].AvailabilityZones[]'
```

## AWS App Mesh의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

App Mesh 컨트롤 영역 인스턴스는 여러 가용 영역에 걸쳐 실행되어고가용성을 보장합니다. App Mesh는 비정상적인 컨트롤 영역 인스턴스를 자동으로 감지하여 교체하고, 자동화된 버전 업그레이드 및 패치를 제공합니다.

## AWS App Mesh의 재해 복구

App Mesh 서비스는 고객 데이터의 백업을 관리합니다. 백업을 관리하기 위해 수행할 작업은 없습니다. 백업된 데이터는 암호화됩니다.

## 의 구성 및 취약성 분석 AWS App Mesh

App Mesh는 마이크로서비스와 함께 배포하는 관리형 [Envoy 프록시 Docker 컨테이너 이미지](#)를 제공합니다. App Mesh는 컨테이너 이미지가 최신 취약성 및 성능 패치로 패치되도록 합니다. App Mesh는 이미지를 제공하기 전에 App Mesh 기능 세트를 기준으로 새로운 Envoy 프록시 릴리스를 테스트합니다.

업데이트된 컨테이너 이미지 버전을 사용하려면 마이크로서비스를 업데이트해야 합니다. 다음은 이미지의 최신 버전입니다.

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.6.1-prod
```

# App Mesh 문제 해결

이 장에서는 App Mesh 사용 시 발생할 수 있는 일반적인 문제 및 문제 해결 모범 사례에 대해 설명합니다. 다음 영역 중 하나를 선택하여 해당 영역의 모범 사례와 일반적인 문제를 검토하세요.

## 주제

- [App Mesh 문제 해결 모범 사례](#)
- [App Mesh 설치 문제 해결](#)
- [App Mesh 연결 문제 해결](#)
- [App Mesh 스케일링 문제 해결](#)
- [App Mesh 관찰성 문제 해결](#)
- [App Mesh 보안 문제 해결](#)
- [App Mesh Kubernetes 문제 해결](#)

## App Mesh 문제 해결 모범 사례

App Mesh 사용 시 발생하는 문제를 해결하려면 이 주제의 모범 사례를 따르는 것이 좋습니다.

### Envoy 프록시 관리 인터페이스 활성화

Envoy 프록시에는 구성 및 통계를 검색하고 연결 트레이닝 등의 기타 관리 기능을 수행하는 데 사용할 수 있는 관리 인터페이스가 함께 제공됩니다. 자세한 내용은 Envoy 설명서의 [관리 인터페이스](#)를 참조하세요.

관리형 [Envoy 이미지](#)를 사용하는 경우 기본적으로 포트 9901에서 관리 엔드포인트가 활성화됩니다. [App Mesh 설치 문제 해결](#)에 제공된 예제는 예제 관리 엔드포인트 URL을 `http://my-app.default.svc.cluster.local:9901/`로 표시합니다.

#### Note

관리 엔드포인트는 퍼블릭 인터넷에 절대 노출되지 않아야 합니다. 또한 `ENVOY_ADMIN_ACCESS_LOG_FILE` 환경 변수에 의해 기본적으로 `/tmp/envoy_admin_access.log`로 설정되는 관리 엔드포인트 로그를 모니터링하는 것이 좋습니다.

## 지표 오프로드를 위한 Envoy DogStats D 통합 활성화

Envoy 프록시는 OSI 계층 4 및 계층 7 트래픽과 내부 프로세스 상태에 대한 통계를 오프로드하도록 구성할 수 있습니다. 이 항목에서는 메트릭이나 Prometheus. 같은 CloudWatch 싱크로 메트릭을 오프로드하지 않고 이러한 통계를 사용하는 방법을 설명하지만, 모든 애플리케이션의 중앙 위치에 이러한 통계를 보관하면 문제를 진단하고 동작을 더 빠르게 확인할 수 있습니다. 자세한 내용은 [Amazon CloudWatch 메트릭스 사용](#) 및 [Prometheus](#) 설명서를 참조하십시오.

에 정의된 파라미터를 설정하여 DogStats D 지표를 구성할 수 있습니다. [DogStatsD 변수](#) DogStatsD에 대한 자세한 내용은 [DogStatsD](#) 설명서를 참조하십시오. [Amazon ECS의 기본 설명과 함께 App Mesh에서 AWS CloudWatch](#) 메트릭으로 메트릭을 오프로드하는 데모를 찾을 수 있습니다. GitHub

## 액세스 로그 활성화

애플리케이션 간에 전송되는 트래픽에 대한 세부 정보를 검색하려면 [가상 노드](#) 및 [가상 게이트웨이](#)에서 액세스 로그를 활성화하는 것이 좋습니다. 자세한 내용은 Envoy 설명서의 [액세스 로깅](#)을 참조하세요. 로그는 OSI 계층 4 및 계층 7 트래픽 동작에 대한 자세한 정보를 제공합니다. Envoy의 기본 형식을 사용하는 경우 다음 구문 분석을 사용하여 Logs [Insights로 CloudWatch](#) 액세스 로그를 분석할 수 있습니다.

```
parse @message "[*] \"* * *\" * * * * * * * * * * *\" as StartTime, Method, Path, Protocol, ResponseCode, ResponseFlags, BytesReceived, BytesSent, DurationMillis, UpstreamServiceTimeMillis, ForwardedFor, UserAgent, RequestId, Authority, UpstreamHost
```

## 사전 프로덕션 환경에서 Envoy 디버그 로깅 활성화

사전 프로덕션 환경에서 Envoy 프록시의 로그 수준을 debug로 설정하는 것이 좋습니다. 디버그 로그는 관련 App Mesh 구성을 프로덕션 환경으로 전환하기 전에 문제를 식별하는 데 도움이 될 수 있습니다.

[Envoy 이미지](#)를 사용하는 경우 ENVY\_LOG\_LEVEL 환경 변수를 통해 로그 수준을 debug로 설정할 수 있습니다.

### Note

프로덕션 환경에서 debug 수준을 사용하는 것은 권장하지 않습니다. [수준을 로 설정하면 로깅이 debug 증가하여 성능 및 로그와 같은 솔루션으로 오프로드되는 로그의 전체 비용에 영향을 미칠 수 있습니다.](#) [CloudWatch](#)

Envoy의 기본 형식을 사용하는 경우 다음 구문 분석 문을 사용하여 Logs [Insights로 프로세스 CloudWatch 로그](#)를 분석할 수 있습니다.

```
parse @message "[*][*][*][*] [*] *" as Time, Thread, Level, Name, Source, Message
```

## App Mesh 제어 영역을 사용하여 Envoy 프록시 연결 모니터링

Envoy 지표 `control_plane.connected_state`를 모니터링하여 Envoy 프록시가 App Mesh 제어 영역과 통신하여 동적 구성 리소스를 가져오는지 확인하는 것이 좋습니다. 자세한 내용은 [관리 서버](#)를 참조하세요.

## App Mesh 설치 문제 해결

이 주제에서는 App Mesh 설치 시 발생할 수 있는 일반적인 문제를 자세히 설명합니다.

### Envoy 컨테이너 이미지를 끌어올 수 없음

#### 증상

Amazon ECS 태스크에서 다음 오류 메시지가 나타납니다. 다음 메시지의 Amazon ECR **## ID** 및 **#**은 컨테이너 이미지를 끌어온 원본 Amazon ECR 리포지토리에 따라 다를 수 있습니다.

```
CannotPullContainerError: Error response from daemon: pull access denied for 840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy, repository does not exist or may require 'docker login'
```

#### 해결 방법

이 오류는 사용 중인 태스크 실행 역할에 Amazon ECR과 통신할 권한이 없으며 리포지토리에서 Envoy 컨테이너 이미지를 끌어올 수 없음을 나타냅니다. Amazon ECS 태스크에 할당된 태스크 실행 역할에는 다음 명령문이 포함된 IAM 정책이 필요합니다.

```
{
 "Action": [
 "ecr:BatchCheckLayerAvailability",
 "ecr:GetDownloadUrlForLayer",
 "ecr:BatchGetImage"
],
 "Resource": "arn:aws:ecr:us-west-2:111122223333:repository/aws-appmesh-envoy",
 "Effect": "Allow"
}
```

```

},
{
 "Action": "ecr:GetAuthorizationToken",
 "Resource": "*",
 "Effect": "Allow"
}

```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## App Mesh Envoy Management Service에 연결할 수 없음

### 증상

Envoy 프록시가 App Mesh Envoy Management Service에 연결할 수 없습니다. 현재 표시되는 내용은 다음과 같습니다.

- 연결 거부 오류
- 연결 시간 초과
- App Mesh Envoy Management Service 엔드포인트를 확인하는 동안 오류 발생
- gRPC 오류

### 해결 방법

Envoy 프록시가 인터넷 또는 프라이빗 [VPC 엔드포인트](#)에 액세스할 수 있고 [보안 그룹](#)이 포트 443에서 아웃바운드 트래픽을 허용하는지 확인합니다. App Mesh의 퍼블릭 Envoy Management Service 엔드포인트는 정규화된 도메인 이름(FQDN) 형식을 따릅니다.

```

App Mesh Production Endpoint
appmesh-envoy-management.Region-code.amazonaws.com

App Mesh Preview Endpoint
appmesh-preview-envoy-management.Region-code.amazonaws.com

```

아래 명령을 사용하여 EMS 연결을 디버깅할 수 있습니다. 이렇게 하면 유효하지만 비어 있는 gRPC 요청이 Envoy Management Service에 전송됩니다.

```

curl -v -k -H 'Content-Type: application/grpc' -X POST https://
appmesh-envoy-management.Region-code.amazonaws.com:443/
envoy.service.discovery.v3.AggregatedDiscoveryService/StreamAggregatedResources

```

이러한 메시지를 다시 받으면 Envoy Management Service에 대한 연결이 정상적으로 작동합니다. gRPC 관련 오류를 디버깅하려면 [Envoy가 오류 텍스트를 나타내며 App Mesh Envoy Management Service에서 연결이 끊김](#)을 참조하세요.

```
grpc-status: 16
grpc-message: Missing Authentication Token
```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## Envoy가 오류 텍스트를 나타내며 App Mesh Envoy Management Service에서 연결이 끊김

### 증상

Envoy 프록시가 App Mesh Envoy Management Service에 연결하여 해당 구성을 받을 수 없습니다. Envoy 프록시 로그에는 다음과 같은 로그 항목이 포함되어 있습니다.

```
gRPC config stream closed: gRPC status code, message
```

### 해결 방법

대부분의 경우 로그의 메시지 부분에 문제가 표시되어야 합니다. 다음 표에는 표시될 수 있는 가장 일반적인 gRPC 상태 코드, 원인 및 해결 방법이 나와 있습니다.

| gRPC 상태 코드 | 원인                                                     | 해결 방법                                                                                       |
|------------|--------------------------------------------------------|---------------------------------------------------------------------------------------------|
| 0          | Envoy Management Service와의 연결을 정상적으로 끊습니다.             | 문제가 없습니다. App Mesh는 이 상태 코드를 사용하여 Envoy 프록시의 연결을 끊는 경우가 있습니다. Envoy는 다시 연결하여 업데이트를 계속 받습니다. |
| 3          | 메시 엔드포인트(가상 노드 또는 가상 게이트웨이) 또는 관련 리소스 중 하나를 찾을 수 없습니다. | Envoy 구성을 다시 확인하여 해당 구성이 나타내는 App Mesh 리소스의 적절한 이름이 있는지 확인합니다. App Mesh 리소스가 다른 리소스         |

| gRPC 상태 코드 | 원인                                                                           | 해결 방법                                                                                                                                              |
|------------|------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
|            |                                                                              | (예: AWS Cloud Map 네임스페이스 또는 ACM 인증서) 와 통합되어 있는 경우 해당 AWS 리소스가 존재하는지 확인하세요.                                                                         |
| 7          | Envoy 프록시는 Envoy Management Service에 연결하거나 관련 리소스를 검색하는 등의 작업을 수행할 권한이 없습니다. | App Mesh 및 기타 서비스에 대한 적절한 정책 설명이 포함된 <a href="#">IAM 정책을 생성하고</a> 해당 정책을 Envoy 프록시가 Envoy Management Service에 연결하는 데 사용하는 IAM 사용자 또는 역할에 연결해야 합니다. |
| 8          | 특정 App Mesh 리소스의 Envoy 프록시 수가 계정 수준 서비스 할당량을 초과합니다.                          | 기본 계정 할당량과 할당량 증가를 요청하는 방법에 대한 자세한 내용은 <a href="#">App Mesh 서비스 할당량</a> 섹션을 참조하세요.                                                                 |

| gRPC 상태 코드 | 원인                                      | 해결 방법                                                                                                                                                                                                                                                                                                                                                                       |
|------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 16         | Envoy 프록시에는 AWS에 대한 유효한 인증 자격 증명이 없습니다. | Envoy가 IAM 사용자 또는 역할을 통해 AWS 서비스에 연결할 수 있는 적절한 자격 증명을 가지고 있는지 확인합니다. Envoy 프로세스에서 1024 이상의 파일 설명자를 사용하는 경우 버전 v1.24 이하의 Envoy에서 알려진 문제인 <a href="#">#24136</a> 으로 인해 자격 증명을 가져오지 못합니다. 이 문제는 Envoy가 높은 트래픽 볼륨을 제공할 때 발생합니다. 디버그 수준에서 Envoy 로그의 텍스트 "A libcurl function was given a bad argument"를 확인하여 이 문제를 확인할 수 있습니다. 이 문제를 완화하려면 Envoy 버전 v1.25.1.0-prod 이상으로 업그레이드하세요. |

[Amazon CloudWatch Insights](#)에서 다음 쿼리를 사용하여 Envoy 프록시의 상태 코드 및 메시지를 관찰할 수 있습니다.

```
filter @message like /gRPC config stream closed/
| parse @message "gRPC config stream closed: *, *" as StatusCode, Message
```

[제공된 오류 메시지가 도움이 되지 않았거나 문제가 여전히 해결되지 않은 경우 문제를 여는 것을 고려해 보십시오. GitHub](#)

## Envoy 컨테이너 상태 확인, 준비 상태 프로브 또는 활성화 프로브 실패

### 증상

Envoy 프록시가 Amazon ECS 태스크, Amazon EC2 인스턴스 또는 Kubernetes 포드에서 상태 확인에 실패합니다. 예를 들어, 다음 명령으로 Envoy 관리 인터페이스를 쿼리하면 LIVE 이외의 상태가 표시 됩니다.

```
curl -s http://my-app.default.svc.cluster.local:9901/server_info | jq '.state'
```

## 해결 방법

다음은 Envoy 프록시가 반환하는 상태에 따른 수정 단계 목록입니다.

- PRE\_INITIALIZING 또는 INITIALIZING - Envoy 프록시가 아직 구성을 받지 않았거나 App Mesh Envoy Management Service에 연결하여 구성을 검색할 수 없습니다. 연결을 시도할 때 Envoy 는 Envoy Management Service에서 오류를 수신할 수 있습니다. 자세한 내용은 [Envoy가 오류 텍스트를 나타내며 App Mesh Envoy Management Service에서 연결이 끊김](#)의 오류를 참조하세요.
- DRAINING - Envoy 프록시는 Envoy 관리 인터페이스의 /healthcheck/fail 또는 / drain\_listeners 요청에 대한 응답으로 연결을 드레이닝하기 시작했습니다. Amazon ECS 태스크, Amazon EC2 인스턴스 또는 Kubernetes 포드를 종료하려는 경우가 아니면 관리 인터페이스에서 이러한 경로를 호출하지 않는 것이 좋습니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## 로드 밸런서에서 메시 엔드포인트로의 상태 확인이 실패함

### 증상

컨테이너 상태 확인 또는 준비 상태 프로브에서 메시 엔드포인트가 정상으로 간주되지만 로드 밸런서에서 메시 엔드포인트로의 상태 확인이 실패합니다.

### 해결 방법

이 문제를 해결하려면 다음 태스크를 완료합니다.

- 메시 엔드포인트와 연결된 [보안 그룹](#)이 상태 확인을 위해 구성된 포트의 인바운드 트래픽을 수락하는지 확인합니다.
- 수동으로 요청할 경우(예: [VPC 내 Bastion Host](#)에서) 상태 확인이 일관되게 성공하는지 확인하세요.
- 가상 노드의 상태 확인을 구성하는 경우 애플리케이션에 상태 확인 엔드포인트를 구현하는 것이 좋습니다(예: HTTP의 경우 /ping). 이렇게 하면 Envoy 프록시와 애플리케이션 모두 로드 밸런서에서 라우팅할 수 있습니다.

- 필요한 기능에 따라 가상 노드에 모든 유형의 Elastic Load Balancer를 사용할 수 있습니다. 자세한 내용은 [Elastic Load Balancing 기능](#)을 참조하세요.
- [가상 게이트웨이](#)에 대해 상태 확인을 구성하는 경우 가상 게이트웨이의 수신기 포트에 대한 TCP 또는 TLS 상태 확인에 [Network Load Balancer](#)를 사용하는 것이 좋습니다. 이렇게 하면 가상 게이트웨이 리스너가 부트스트랩되어 연결을 수락할 준비가 됩니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## 가상 게이트웨이는 포트 1024 이하에서 트래픽을 허용하지 않음

### 증상

가상 게이트웨이는 포트 1024 이하에서는 트래픽을 허용하지 않지만 1024보다 큰 포트 번호에서는 트래픽을 허용합니다. 예를 들어 다음 명령으로 Envoy 통계를 쿼리하면 0이 아닌 값이 수신됩니다.

```
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "update_rejected"
```

로그에서 권한 있는 포트에 바인딩하지 못했음을 설명하는 다음 텍스트와 비슷한 텍스트가 표시될 수 있습니다.

```
gRPC config for type.googleapis.com/envoy.api.v2.Listener rejected: Error adding/
updating listener(s) lds_ingress_0.0.0.0_port_<port num>: cannot bind '0.0.0.0:<port
num>': Permission denied
```

### 해결 방법

이 문제를 해결하려면 게이트웨이에 지정된 사용자에게 Linux 기능 CAP\_NET\_BIND\_SERVICE가 있어야 합니다. 자세한 내용은 Linux 프로그래머 설명서의 [기능](#), ECS 태스크 정의 파라미터의 [Linux 파라미터](#) 및 Kubernetes 설명서의 [컨테이너 기능 설정](#)을 참조하세요.

#### Important

Fargate는 1024보다 큰 포트 값을 사용해야 합니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## App Mesh 연결 문제 해결

이 주제에서는 App Mesh 연결에서 발생할 수 있는 일반적인 문제를 자세히 설명합니다.

### 가상 서비스의 DNS 이름을 확인할 수 없음

#### 증상

애플리케이션이 연결을 시도하는 가상 서비스의 DNS 이름을 확인할 수 없습니다.

#### 해결 방법

이는 알려진 문제입니다. 자세한 내용은 [호스트 VirtualServices 이름별 이름 지정 또는 FQDN GitHub 문제를](#) 참조하십시오. App Mesh의 가상 서비스에는 어떤 이름도 지정 가능합니다. 가상 서비스 이름에 대한 DNS A 레코드가 있고 애플리케이션이 가상 서비스 이름을 확인할 수 있는 한, Envoy에서 요청을 프록시하여 적절한 대상으로 라우팅합니다. 이 문제를 해결하려면 가상 서비스 이름으로 루프백이 아닌 IP 주소(예: 10.10.10.10)에 DNS A 레코드를 추가합니다. 다음 조건에 따라 DNS A 레코드를 추가할 수 있습니다.

- Amazon Route 53에서 이름 뒤에 프라이빗 호스팅 영역 이름이 붙은 경우
- 애플리케이션 컨테이너의 /etc/hosts 파일 내부
- 관리하는 타사 DNS 서버에서

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의](#)하세요.

### 가상 서비스 백엔드에 연결할 수 없음

#### 증상

애플리케이션이 가상 노드의 백엔드로 정의된 가상 서비스에 연결을 설정할 수 없습니다. 연결을 설정하려고 하면 연결이 완전히 실패하거나 애플리케이션 관점에서 보면 요청이 HTTP 503 응답 코드를 나타내며 실패할 수 있습니다.

#### 해결 방법

애플리케이션에 전혀 연결되지 않는 경우(HTTP 503 응답 코드가 반환되지 않음), 다음과 같이 하세요.

- 컴퓨팅 환경이 App Mesh에서 작동하도록 설정되었는지 확인하세요.

- Amazon ECS의 경우 적절한 [프록시 구성](#)을 활성화해야 합니다. 자세한 내용은 [App end-to-end Mesh 및 Amazon ECS 시작하기](#)를 참조하십시오.
- Amazon EKS를 포함한 Kubernetes의 경우 Helm을 통해 최신 App Mesh 컨트롤러를 설치해야 합니다. 자세한 내용은 Helm Hub의 [App Mesh 컨트롤러](#) 또는 [자습서: App Mesh와의 App Mesh 통합 구성](#)을 참조하세요.
- Amazon EC2의 경우 App Mesh 트래픽을 프록시하도록 Amazon EC2 인스턴스를 설정했는지 확인하세요. 자세한 내용은 [서비스 업데이트](#)를 참조하세요.
- 컴퓨팅 서비스에서 실행 중인 Envoy 컨테이너가 App Mesh Envoy Management Service에 성공적으로 연결되었는지 확인합니다. 필드 `control_plane.connected_state`의 Envoy 통계를 확인하여 이러한 연결을 확인할 수 있습니다. `control_plane.connected_state`에 대한 자세한 내용은 문제 해결 모범 사례의 [Envoy 프록시 연결 모니터링](#)을 참조하세요.

Envoy가 처음에 연결을 설정할 수 있었지만 이후에 연결이 끊겼다가 다시 연결되지 않은 경우, 연결이 끊긴 이유를 해결하려면 [Envoy와 App Mesh Envoy Management Service 간의 연결 끊김](#) 오류 텍스트를 참조하세요.

애플리케이션이 연결되지만 요청이 HTTP 503 응답 코드를 나타내며 실패하는 경우, 다음을 시도해 보세요.

- 연결하려는 가상 서비스가 메시에 있는지 확인하세요.
- 가상 서비스에 공급자(가상 라우터 또는 가상 노드)가 있는지 확인하세요.
- Envoy를 HTTP 프록시로 사용할 때 Envoy 통계를 통해 송신 트래픽이 올바른 대상 대신, `cluster.cds_egress*_mesh-allow-all`로 들어오는 것이 확인되면 Envoy가 `filter_chains`를 통해 요청을 제대로 라우팅하지 못할 가능성이 큼니다. 이것은 검증되지 않은 가상 서비스 이름을 사용한 결과일 수 있습니다. Envoy 프록시는 해당 이름을 통해 다른 가상 서비스와 통신하므로 실제 서비스의 서비스 검색 이름을 가상 서비스 이름으로 사용하는 것이 좋습니다.

자세한 내용은 [가상 서비스](#)를 참조하세요.

- Envoy 프록시 로그를 검사하여 다음과 같은 오류 메시지가 있는지 확인합니다.
  - No healthy upstream - Envoy 프록시가 라우팅하려는 가상 노드에 확인된 엔드포인트가 없거나 정상 엔드포인트가 없습니다. 대상 가상 노드의 서비스 검색 및 상태 확인 설정이 올바른지 확인하세요.

백엔드 가상 서비스를 배포하거나 스케일링하는 동안 서비스에 대한 요청이 실패하는 경우 [가상 서비스에 가상 노드 공급자가 있는 경우 일부 요청은 HTTP 상태 코드 503을 나타내며 실패합니다.](#)의 지침을 따르세요.

- No cluster match for URL - 이 문제는 가상 라우터 공급자에서 정의된 경로에 의해 정의된 기준과 일치하지 않는 가상 서비스로 요청이 전송될 때 발생할 가능성이 큼니다. 경로와 HTTP 요청 헤더가 올바른지 확인하여 애플리케이션의 요청이 지원되는 경로로 전송되는지 확인합니다.
- No matching filter chain found - 이 문제는 요청이 잘못된 포트의 가상 서비스로 전송될 때 발생할 가능성이 큼니다. 애플리케이션의 요청이 가상 라우터에 지정된 것과 동일한 포트를 사용하고 있는지 확인합니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## 외부 서비스에 연결할 수 없음

### 증상

애플리케이션이 메시 외부의 서비스(예: amazon.com)에 연결할 수 없습니다.

### 해결 방법

기본적으로 App Mesh는 메시 내 애플리케이션에서 메시 외부의 대상으로 향하는 아웃바운드 트래픽을 허용하지 않습니다. 외부 서비스와의 통신을 활성화하는 옵션에는 다음 두 가지가 있습니다.

- 메시 리소스의 [아웃바운드 필터](#)를 ALLOW\_ALL로 설정합니다. 이 설정을 사용하면 메시 내의 모든 애플리케이션이 메시 내부 또는 외부의 모든 대상 IP 주소와 통신할 수 있습니다.
- 가상 서비스, 가상 라우터, 경로 및 가상 노드를 사용하여 메시의 외부 서비스를 모델링합니다. 예를 들어 외부 서비스 example.com을 모델링하려면 가상 라우터를 사용하여 이름이 example.com인 가상 서비스를 생성하고, DNS 서비스 검색 호스트 이름이 example.com인 가상 노드로 모든 트래픽을 보내는 경로를 생성할 수 있습니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## MySQL 또는 SMTP 서버에 연결할 수 없음

### 증상

가상 노드 정의를 사용하여 SMTP 서버 또는 MySQL 데이터베이스와 같은 모든 대상(Mesh EgressFilter type =ALLOW\_ALL)에 대한 아웃바운드 트래픽을 허용하면 애플리케이션에서의 연결이 실패합니다. 예를 들어, 다음은 MySQL 서버에 연결하려고 할 때 나타나는 오류 메시지입니다.

```
ERROR 2013 (HY000): Lost connection to MySQL server at 'reading initial communication packet', system error: 0
```

## 해결 방법

이 문제는 알려진 문제이며 App Mesh 이미지 버전 1.15.0 이상을 사용하면 해결됩니다. 자세한 내용은 [App GitHub Mesh를 사용하여 MySQL에 연결할 수 없음 문제를](#) 참조하십시오. 이 오류는 App Mesh에서 구성한 Envoy의 아웃바운드 리스너가 Envoy TLS Inspector 리스너 필터를 추가하기 때문에 발생합니다. 자세한 내용은 Envoy 설명서에서 [TLS Inspector](#)를 참조하세요. 이 필터는 클라이언트에서 보낸 첫 번째 패킷을 검사하여 연결이 TLS를 사용하고 있는지 여부를 평가합니다. 그러나 MySQL과 SMTP를 사용하는 경우 서버는 연결 후 첫 번째 패킷을 전송합니다. MySQL에 대한 자세한 내용은 MySQL 설명서의 [초기 핸드셰이크](#)를 참조하세요. 서버가 첫 번째 패킷을 전송하기 때문에 필터 검사는 실패합니다.

사용 중인 Envoy 버전에 따라 이 문제를 해결하려면:

- App Mesh 이미지 Envoy 버전이 1.15.0 이상인 경우 MySQL, SMTP, MSSQL 등과 같은 외부 서비스를 애플리케이션 가상 노드의 백엔드로 모델링하지 마세요.
- App Mesh 이미지 Envoy 버전이 1.15.0 이전인 경우 MySQL용 서비스에서 APPMESH\_EGRESS\_IGNORED\_PORTS의 값 목록에 SMTP에 사용하는 포트로서 포트 3306을 추가합니다.

### Important

표준 SMTP 포트는 25, 587 및 465이지만 사용 중인 포트만 APPMESH\_EGRESS\_IGNORED\_PORTS에 추가해야 하며 세 포트를 모두 추가해서는 안 됩니다.

자세한 내용은 Kubernetes용 [서비스 업데이트](#), Amazon ECS용 [서비스 업데이트](#) 또는 Amazon EC2용 [서비스 업데이트](#)를 참조하세요.

문제가 여전히 해결되지 않은 경우 기존 문제와 관련하여 발생한 [GitHub 문제](#)에 대한 세부 정보를 제공하거나 [AWS Support](#)에 문의할 수 있습니다.

## App Mesh에서 TCP 가상 노드 또는 가상 라우터로 모델링된 서비스에 연결할 수 없음

### 증상

애플리케이션은 App Mesh [PortMapping](#) 정의의 TCP 프로토콜 설정을 사용하는 백엔드에 연결할 수 없습니다.

## 해결 방법

이는 알려진 문제입니다. 자세한 내용은 [같은 포트에 있는 여러 TCP 대상으로 라우팅 켜기를](#) 참조하십시오. GitHub OSI 계층 4에서 Envoy 프록시에 제공되는 정보의 제한으로 인해 App Mesh는 현재 TCP로 모델링된 여러 백엔드 대상이 동일한 포트를 공유하는 것을 허용하지 않습니다. TCP 트래픽이 모든 백엔드 대상에 적절하게 라우팅될 수 있도록 하려면 다음을 수행합니다.

- 모든 대상이 고유한 포트를 사용하고 있는지 확인합니다. 백엔드 가상 서비스에 가상 라우터 공급자를 사용하는 경우 라우팅되는 가상 노드의 포트를 변경하지 않고도 가상 라우터 포트를 변경할 수 있습니다. 이렇게 하면 Envoy 프록시가 가상 노드에 정의된 포트를 계속 사용하는 동안 애플리케이션이 가상 라우터 포트에서 연결을 열 수 있습니다.
- TCP로 모델링된 대상이 MySQL 서버이거나, 서버가 연결 후 첫 번째 패킷을 보내는 다른 TCP 기반 프로토콜이 있는 경우 [MySQL 또는 SMTP 서버에 연결할 수 없음](#) 섹션을 참조하세요.

문제가 여전히 해결되지 않은 경우 기존 문제와 관련하여 발생한 [GitHub 문제에](#) 대한 세부 정보를 제공하거나 [AWS Support에](#) 문의할 수 있습니다.

## 가상 노드의 가상 서비스 백엔드로 나열되지 않은 서비스에 성공적으로 연결됨

### 증상

애플리케이션은 가상 노드에서 가상 서비스 백엔드로 지정되지 않은 대상에 연결하여 트래픽을 전송할 수 있습니다.

### 해결 방법

App Mesh API에서 모델링되지 않은 대상에 대한 요청이 성공하는 경우 메시의 [아웃바운드 필터](#) 유형이 ALLOW\_ALL로 설정되었기 때문일 가능성이 큼니다. 아웃바운드 필터가 ALLOW\_ALL로 설정되면 모델링된 대상(백엔드)과 일치하지 않는 애플리케이션의 아웃바운드 요청이 애플리케이션에서 설정한 대상 IP 주소로 전송됩니다.

메시에서 모델링되지 않은 대상으로의 트래픽을 허용하지 않으려면 아웃바운드 필터 값을 DROP\_ALL로 설정하는 것이 좋습니다.

#### Note

메시 아웃바운드 필터 값을 설정하면 메시 내의 모든 가상 노드에 영향을 줍니다.

도메인이 아닌 아웃바운드 트래픽에 대해서는 TLS를 DROP\_ALL 구성하고 egress\_filter 활성화할 수 없습니다. AWS

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

가상 서비스에 가상 노드 공급자가 있는 경우 일부 요청은 HTTP 상태 코드 **503**을 나타내며 실패합니다.

### 증상

가상 라우터 공급자 대신 가상 노드 공급자를 사용하는 가상 서비스 백엔드에 대한 애플리케이션 요청 중 일부가 실패합니다. 가상 서비스에 가상 라우터 공급자를 사용하는 경우 요청은 실패하지 않습니다.

### 해결 방법

이는 알려진 문제입니다. 자세한 내용은 가상 [서비스를 위한 가상 노드 공급자 재시도 정책](#)을 참조하십시오. GitHub 가상 노드를 가상 서비스 공급자로 사용하는 경우 가상 서비스의 클라이언트가 사용할 기본 재시도 정책을 지정할 수 없습니다. 이와 비교할 때 가상 라우터 공급자는 하위 경로 리소스의 속성이므로 재시도 정책을 지정할 수 있습니다.

가상 노드 공급자에 대한 요청 실패를 줄이려면 가상 라우터 공급자를 대신 사용하고, 해당 경로에 재시도 정책을 지정합니다. 애플리케이션에 대한 요청 실패를 줄이는 다른 방법은 [App Mesh 모범 사례](#) 섹션을 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## Amazon EFS 파일 시스템에 연결할 수 없음

### 증상

Amazon EFS 파일 시스템을 볼륨으로 사용하여 Amazon ECS 태스크를 구성하면 다음 오류를 발생하면서 태스크가 시작되지 않습니다.

```
ResourceInitializationError: failed to invoke EFS utils commands to set up EFS volumes:
 stderr: mount.nfs4: Connection refused : unsuccessful EFS utils command execution;
 code: 32
```

### 해결 방법

이는 알려진 문제입니다. 이 오류는 Amazon EFS로의 NFS 연결이 태스크의 컨테이너가 시작되기 전에 발생하기 때문에 나타납니다. 이 트래픽은 프록시 구성을 통해 지금은 실행되지 않는 Envoy로 라우팅됩니다. 시작 순서 때문에 NFS 클라이언트가 Amazon EFS 파일 시스템에 연결하지 못하고 태스크가 시작되지 않습니다. 이 문제를 해결하려면 Amazon ECS 태스크 정의의 프록시 구성에서 EgressIgnoredPorts 설정 값 목록에 포트 2049를 추가합니다. 자세한 내용은 [프록시 구성](#)을 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## 서비스에 성공적으로 연결되지만 수신 요청은 Envoy의 액세스 로그, 추적 또는 지표에 나타나지 않음

### 증상

애플리케이션이 다른 애플리케이션에 연결하여 요청을 보낼 수는 있지만 액세스 로그나 Envoy 프록시의 추적 정보에서 들어오는 요청을 볼 수 없습니다.

### 해결 방법

이는 알려진 문제입니다. 자세한 내용은 Github의 [iptables 규칙 설정](#) 문제를 참조하세요. Envoy 프록시는 해당 가상 노드가 수신 중인 포트에 향하는 인바운드 트래픽만 가로칩니다. 다른 포트에 대한 요청은 Envoy 프록시를 우회하여 뒤에 있는 서비스에 직접 도달합니다. Envoy 프록시가 서비스의 인바운드 트래픽을 가로채도록 하려면 가상 노드와 서비스가 동일한 포트에서 수신하도록 설정해야 합니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## 컨테이너 수준에서 HTTP\_PROXY/HTTPS\_PROXY 환경 변수를 설정해도 예상대로 작동하지 않습니다.

### 증상

다음에서 HTTP\_PROXY/HTTPS\_PROXY가 환경 변수로 설정된 위치에 따라 다음이 적용됩니다.

- App Mesh가 활성화된 태스크 정의의 앱 컨테이너: App Mesh 서비스의 네임스페이스로 전송되는 요청은 Envoy 사이드카에서 HTTP 500 오류 응답을 받습니다.
- App Mesh가 활성화된 태스크 정의의 Envoy 컨테이너: Envoy 사이드카에서 나오는 요청은 HTTP/HTTPS 프록시 서버를 거치지 않으며 환경 변수가 작동하지 않습니다.

## 해결 방법

### 앱 컨테이너의 경우:

App Mesh는 태스크 내의 트래픽이 Envoy 프록시를 통과하도록 허용하는 방식으로 작동합니다. HTTP\_PROXY/HTTPS\_PROXY 구성은 컨테이너 트래픽이 다른 외부 프록시를 통과하도록 구성하여 이 동작을 무시합니다. Envoy는 여전히 트래픽을 가로채지만 외부 프록시를 사용한 메시 트래픽의 프록시는 지원하지 않습니다.

메시가 아닌 모든 트래픽을 프록시하려면 다음 예제와 같이 메시의 CIDR/네임스페이스, localhost 및 자격 증명의 엔드포인트를 포함하도록 NO\_PROXY를 설정하세요.

```
NO_PROXY=localhost,127.0.0.1,169.254.169.254,169.254.170.2,10.0.0.0/16
```

### Envoy 컨테이너의 경우:

Envoy는 일반 프록시를 지원하지 않습니다. 이러한 변수는 설정하지 않는 것이 좋습니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

**경로 제한 시간을 설정한 후에도 업스트림 요청 제한 시간이 초과됩니다.**

## 증상

다음에 대해 제한 시간을 정의한 경우 해당 결과가 발생합니다.

- 경로에 대해 제한 시간을 정의했으나 여전히 업스트림 요청 제한 시간 오류가 발생합니다.
- 가상 노드 리스너에 대해 제한 시간을 정의하고 경로의 제한 시간 및 재시도 제한 시간을 정의했지만 여전히 업스트림 요청 제한 시간 오류가 발생합니다.

## 해결 방법

15초를 초과하는 지연 시간이 긴 요청을 성공적으로 완료하려면 경로 및 가상 노드 리스너 수준 모두에서 제한 시간을 지정해야 합니다.

경로 제한 시간을 기본값인 15초보다 크게 지정하는 경우 모든 참여 가상 노드의 리스너에도 제한 시간을 지정해야 합니다. 그러나 제한 시간을 기본값보다 낮은 값으로 줄이는 경우 가상 노드에서 제한 시간을 업데이트할 수도 있습니다. 가상 노드 및 경로 설정 옵션에 대한 자세한 내용은 [가상 노드 및 경로](#)를 참조하세요.

재시도 정책을 지정한 경우 요청 제한 시간에 지정하는 기간은 항상 재시도 제한 시간에 재시도 정책에서 정의한 최대 재시도 횟수를 곱한 값보다 크거나 같아야 합니다. 이렇게 하면 모든 재시도가 포함된 요청을 성공적으로 완료할 수 있습니다. 자세한 내용은 [경로](#)를 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub](#) [문의하세요](#).

## Envoy는 HTTP 잘못된 요청으로 응답합니다.

### 증상

Envoy는 Network Load Balancer(NLB)를 통해 전송된 모든 요청에 대해 HTTP 400 잘못된 요청으로 응답합니다. Envoy 로그를 확인하면 다음과 같은 내용을 확인할 수 있습니다.

- 디버그 로그:

```
dispatch error: http/1.1 protocol error: HPE_INVALID_METHOD
```

- 액세스 로그:

```
"- - HTTP/1.1" 400 DPE 0 11 0 - "-" "-" "-" "-" "
```

### 해결 방법

해결 방법은 NLB의 [대상 그룹 속성](#)에서 프록시 프로토콜 버전 2(PPv2)를 비활성화하는 것입니다.

현재 PPv2는 App Mesh 제어 영역을 사용하여 실행되는 가상 게이트웨이 및 가상 노드 Envoy에서 지원되지 않습니다. Kubernetes에서 AWS 로드 밸런서 컨트롤러를 사용하여 NLB를 배포하는 경우 다음 속성을 다음과 같이 설정하여 PPv2를 비활성화합니다. `false`

```
service.beta.kubernetes.io/aws-load-balancer-target-group-attributes:
 proxy_protocol_v2.enabled
```

NLB 리소스 속성에 대한 자세한 내용은 [AWS 로드 밸런서 컨트롤러 주석](#)을 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub](#) [문의하세요](#).

## 제한 시간을 제대로 구성할 수 없습니다.

### 증상

가상 노드 리스너에서 제한 시간을 구성하고 가상 노드 백엔드로 향하는 경로에서 제한 시간을 구성한 후에도 요청 제한 시간이 15초 이내에 초과됩니다.

## 해결 방법

백엔드 목록에 올바른 가상 서비스가 포함되어 있는지 확인하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## App Mesh 스케일링 문제 해결

이 주제에서는 App Mesh 스케일링 시 발생할 수 있는 일반적인 문제를 자세히 설명합니다.

가상 노드/가상 게이트웨이의 복제본을 50개 이상으로 스케일링할 경우 연결이 실패하고 컨테이너 상태 확인이 실패합니다.

### 증상

가상 노드/가상 게이트웨이에 대해 Amazon ECS 태스크, Kubernetes 포드 또는 Amazon EC2 인스턴스 등의 복제본 수를 50개 이상으로 스케일링하면 현재 실행 중인 새 Envoy에 대한 Envoy 컨테이너 상태 확인이 실패하기 시작합니다. 가상 노드/가상 게이트웨이로 트래픽을 전송하는 다운스트림 애플리케이션은 HTTP 상태 코드 503을 나타내는 요청 실패를 확인하기 시작합니다.

### 해결 방법

가상 노드/가상 게이트웨이당 Envoy 수에 대한 App Mesh의 기본 할당량은 50입니다. 실행 중인 Envoy의 수가 이 할당량을 초과하면 현재 실행 중인 새 Envoy가 gRPC 상태 코드 8(RESOURCE\_EXHAUSTED)을 사용하여 App Mesh의 Envoy Management Service에 연결하지 못합니다. 이 할당량을 늘릴 수 있습니다. 자세한 정보는 [App Mesh 서비스 할당량](#)을 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

가상 서비스 백엔드가 수평적으로 스케일 아웃되거나 스케일 인될 때 **503**을 나타내며 요청이 실패함

### 증상

백엔드 가상 서비스가 수평적으로 스케일 아웃되거나 스케일 인될 경우 다운스트림 애플리케이션의 요청은 HTTP 503 상태 코드를 나타내며 실패합니다.

## 해결 방법

App Mesh는 애플리케이션을 수평적으로 스케일링하는 동안 실패 사례를 줄일 수 있는 몇 가지 접근 방식을 권장합니다. 이러한 오류를 방지하는 방법에 대한 자세한 내용은 [App Mesh 모범 사례](#) 섹션을 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub](#) [문의하세요](#).

## 로드가 증가하면 Envoy 컨테이너가 segfault를 나타내며 충돌함

### 증상

트래픽 로드가 높으면 세분화 오류(Linux 종료 코드 139)로 인해 Envoy 프록시가 충돌합니다. Envoy 프로세스 로그에는 다음과 같은 명령문이 포함됩니다.

```
Caught Segmentation fault, suspect faulting address 0x0"
```

### 해결 방법

Envoy 프록시가 한 프로세스에서 한 번에 열 수 있는 파일 수 제한을 나타내는 운영 체제의 기본 nofile ulimit를 위반했을 수 있습니다. 이 위반은 트래픽으로 인해 더 많은 연결이 발생하여 운영 체제 소켓이 추가로 소모되기 때문에 발생합니다. 이 문제를 해결하려면 호스트 운영 체제에서 ulimit nofile 값을 늘리세요. Amazon ECS를 사용하는 경우 태스크 정의의 [리소스 제한 설정](#)에 있는 [Ulimit 설정](#)을 통해 이 제한을 변경할 수 있습니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub](#) [문의하세요](#).

## 기본 리소스 증가는 서비스 제한에 반영되지 않습니다.

### 증상

App Mesh 리소스의 기본 제한을 늘린 후에는 서비스 제한을 확인할 때 새 값이 반영되지 않습니다.

### 해결 방법

새 제한은 현재 표시되지 않지만 고객은 여전히 해당 제한을 적용할 수 있습니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub](#) [문의하세요](#).

## 방대한 상태 확인 호출로 인해 애플리케이션이 충돌합니다.

### 증상

가상 노드에 대한 활성 상태 확인을 활성화한 후 상태 확인 호출 수가 증가합니다. 애플리케이션에 대한 상태 확인 호출의 양이 크게 증가하여 애플리케이션이 충돌합니다.

## 해결 방법

활성 상태 확인이 활성화되면 다운스트림의 각 Envoy 엔드포인트(클라이언트)는 라우팅 결정을 내리기 위해 업스트림 클러스터의 각 엔드포인트(서버)에 상태 요청을 보냅니다. 따라서 총 상태 확인 요청 수는  $\text{number of client Envoys} * \text{number of server Envoys} * \text{active health check frequency}$ 가 됩니다.

이 문제를 해결하려면 상태 확인 프로브의 빈도를 수정합니다. 이렇게 하면 상태 확인 프로브의 총 용량이 줄어듭니다. 활성 상태 확인 외에도 App Mesh에서 [이상치 탐지](#)를 수동 상태 확인의 수단으로 구성할 수 있습니다. 이상치 탐지를 사용하면 연속적인 5xx호 응답을 기준으로 특정 호스트를 제거할 시기를 구성할 수 있습니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## App Mesh 관찰성 문제 해결

이 주제에서는 App Mesh 관찰성과 관련해서 발생할 수 있는 일반적인 문제를 자세히 설명합니다.

### 내 애플리케이션에 대한 AWS X-Ray 흔적을 볼 수 없습니다.

#### 증상

App Mesh의 애플리케이션이 X-Ray 콘솔 또는 API에 X-Ray 추적 정보를 표시하지 않습니다.

#### 해결 방법

App Mesh에서 X-Ray를 사용하려면 애플리케이션, 사이드카 컨테이너 및 X-Ray 서비스 간의 통신이 가능하도록 구성 요소를 올바르게 구성해야 합니다. 다음 단계에 따라 X-Ray가 올바르게 설정되었는지 확인합니다.

- App Mesh 가상 노드 리스너 프로토콜이 TCP로 설정되어 있지 않은지 확인합니다.
- 애플리케이션과 함께 배포된 X-Ray 컨테이너가 UDP 포트 2000을 노출하고 사용자 1337 권한으로 실행되는지 확인합니다. 자세한 내용은 의 [Amazon ECS X-Ray 예제](#)를 참조하십시오. GitHub
- Envoy 컨테이너에 추적이 활성화되어 있는지 확인합니다. [App Mesh Envoy 이미지](#)를 사용하는 경우 `ENABLE_ENVOY_XRAY_TRACING` 환경 변수를 1 값으로 설정하고 `XRAY_DAEMON_PORT` 환경 변수를 2000으로 설정하여 X-Ray를 활성화할 수 있습니다.

- [언어별 SDK](#) 중 하나를 사용하여 애플리케이션 코드에서 X-Ray를 계측한 경우 해당 언어의 가이드에 따라 X-Ray가 올바르게 구성되었는지 확인합니다.
- 이전 항목이 모두 올바르게 구성된 경우 X-Ray 컨테이너 로그에서 오류를 검토하고 [AWS X-Ray 문제 해결](#)의 지침을 따르세요. App Mesh에서의 X-Ray 통합에 대한 자세한 설명은 [X-Ray와 App Mesh 통합](#)에서 확인할 수 있습니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## Amazon 메트릭에서 내 애플리케이션에 대한 Envoy 메트릭을 볼 수 없습니다. CloudWatch

### 증상

App Mesh의 애플리케이션은 Envoy 프록시에서 생성한 지표를 지표로 내보내지 않습니다. CloudWatch

### 해결 방법

App Mesh에서 CloudWatch 메트릭을 사용하는 경우 Envoy 프록시, CloudWatch 에이전트 사이드 카 및 메트릭 서비스 간의 통신을 가능하게 하려면 몇 가지 구성 요소를 올바르게 구성해야 합니다. CloudWatch 다음 단계에 따라 Envoy 프록시에 대한 CloudWatch 지표가 올바르게 설정되었는지 확인하십시오.

- App Mesh의 CloudWatch 에이전트 이미지를 사용하고 있는지 확인하십시오. 자세한 내용은 [App Mesh CloudWatch 에이전트](#) 쉘을 참조하십시오 GitHub.
- 플랫폼별 사용 지침에 따라 App Mesh용 CloudWatch 에이전트를 적절하게 구성했는지 확인하십시오. 자세한 내용은 [App Mesh CloudWatch 에이전트](#) 쉘을 참조하십시오 GitHub.
- 이전 항목이 모두 올바르게 구성된 경우 CloudWatch 에이전트 컨테이너 로그에서 오류를 검토하고 [CloudWatch 에이전트 문제 해결](#)에 제공된 지침을 따르십시오.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## AWS X-Ray 추적에 대한 사용자 지정 샘플링 규칙을 구성할 수 없음

### 증상

애플리케이션에서 X-추적을 사용하고 있지만 추적에 대한 샘플링 규칙을 구성할 수 없습니다.

### 해결 방법

App Mesh Envoy는 현재 동적 X-Ray 샘플링 구성을 지원하지 않으므로 다음과 같은 해결 방법을 사용할 수 있습니다.

Envoy 버전이 1.19.1 이상인 경우, 다음 옵션을 사용할 수 있습니다.

- 샘플링 속도만 설정하려면 Envoy 컨테이너에서 XRAY\_SAMPLING\_RATE 환경 변수를 사용하세요. 값은 0 ~ 1.00(100%) 범위의 십진수로 지정해야 합니다. 자세한 정보는 [AWS X-Ray 변수](#)를 참조하세요.
- X-Ray 추적기에 대한 현지화된 사용자 지정 샘플링 규칙을 구성하려면 XRAY\_SAMPLING\_RULE\_MANIFEST 환경 변수를 사용하여 Envoy 컨테이너 파일 시스템에서 파일 경로를 지정합니다. 자세한 내용은 AWS X-Ray 개발자 안내서의 [샘플링 규칙](#)을 참조하세요.

Envoy 버전이 1.19.1 이전 버전인 경우 다음을 수행하세요.

- ENVOY\_TRACING\_CFG\_FILE 환경 변수를 사용하여 샘플링 속도를 변경합니다. 자세한 정보는 [Envoy 구성 변수](#)를 참조하세요. 사용자 지정 추적 구성을 지정하고 로컬 샘플링 규칙을 정의합니다. 자세한 내용은 [Envoy X-Ray 구성](#)을 참조하세요.
- ENVOY\_TRACING\_CFG\_FILE 환경 변수에 대한 사용자 지정 추적 구성 예제:

```
tracing:
 http:
 name: envoy.tracers.xray
 typedConfig:
 "@type": type.googleapis.com/envoy.config.trace.v3.XRayConfig
 segmentName: foo/bar
 segmentFields:
 origin: AWS::AppMesh::Proxy
 aws:
 app_mesh:
 mesh_name: foo
 virtual_node_name: bar
 daemonEndpoint:
 protocol: UDP
 address: 127.0.0.1
 portValue: 2000
 samplingRuleManifest:
 filename: /tmp/sampling-rules.json
```

- samplingRuleManifest 속성의 샘플링 규칙 매니페스트 구성에 대한 자세한 내용은 [Go용 X-Ray SDK 구성](#)을 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## App Mesh 보안 문제 해결

이 주제에서는 App Mesh 보안과 관련해서 발생할 수 있는 일반적인 문제를 자세히 설명합니다.

### TLS 클라이언트 정책을 사용하여 백엔드 가상 서비스에 연결할 수 없음

#### 증상

가상 노드의 가상 서비스 백엔드에 TLS 클라이언트 정책을 추가하면 해당 백엔드에 대한 연결이 실패합니다. 백엔드 서비스로 트래픽을 보내려고 하면 HTTP 503 응답 코드와 upstream connect error or disconnect/reset before headers. reset reason: connection failure 오류 메시지를 나타내면서 요청이 실패합니다.

#### 해결 방법

문제의 근본 원인을 파악하려면 문제를 진단하는 데 도움이 되는 Envoy 프록시 프로세스 로그를 사용하는 것이 좋습니다. 자세한 정보는 [사전 프로덕션 환경에서 Envoy 디버그 로깅 활성화](#)를 참조하세요. 다음 목록을 사용하여 연결 실패의 원인을 확인하세요.

- [가상 서비스 백엔드에 연결할 수 없음](#)에서 언급한 오류를 차단하고 백엔드에 대한 연결이 성공하는지 확인하세요.
- Envoy 프로세스 로그에서 다음 오류(디버그 수준에서 기록됨)를 찾아보세요.

```
TLS error: 268435581:SSL routines:OPENSSL_internal:CERTIFICATE_VERIFY_FAILED
```

이 오류는 다음 원인 중 하나 이상으로 인해 발생할 수 있습니다.

- TLS 클라이언트 정책 신뢰 번들에 정의된 인증 기관 중 하나에서 인증서에 서명하지 않았습니다.
- 인증서가 더 이상 유효하지 않습니다(만료됨).
- 주체 대체 이름(SAN)이 요청된 DNS 호스트 이름과 일치하지 않습니다.
- 백엔드 서비스에서 제공하는 인증서가 유효하고, TLS 클라이언트 정책 신뢰 번들에 있는 인증 기관 중 하나에서 서명했으며, [전송 계층 보안 \(TLS\)](#)에서 정의한 기준을 충족하는지 확인하세요.
- 아래와 같은 오류가 표시되면 요청이 Envoy 프록시를 우회하여 애플리케이션에 직접 도달하고 있다는 의미입니다. 트래픽을 전송할 때 Envoy의 통계는 변경되지 않으므로 Envoy가 트래픽을 해독하지 않게 됩니다. 가상 노드의 프록시 구성에서 애플리케이션이 수신하는 올바른 값이 AppPorts에 포함되어 있는지 확인합니다.

```
upstream connect error or disconnect/reset before headers. reset reason:
connection failure, transport failure reason: TLS error: 268435703:SSL
routines:OPENSSL_internal:WRONG_VERSION_NUMBER
```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#). 보안 취약점을 발견했다고 생각되거나 App Mesh의 보안에 대해 궁금한 점이 있으면 [AWS 취약성 보고 지침](#)을 참조하세요.

## 애플리케이션에서 TLS를 시작할 때 백엔드 가상 서비스에 연결할 수 없음

### 증상

Envoy 프록시 대신 애플리케이션에서 TLS 세션을 시작하면 백엔드 가상 서비스로의 연결이 실패합니다.

### 해결 방법

이는 알려진 문제입니다. 자세한 내용은 [기능 요청: 다운스트림 애플리케이션과 업스트림 프록시 간의 TLS 협상](#) 문제를 참조하십시오. GitHub App Mesh에서 TLS 시작이 현재 Envoy 프록시에서는 지원되지만 애플리케이션에서는 지원되지 않습니다. Envoy에서 TLS 시작 지원을 사용하려면 애플리케이션에서 TLS 시작을 비활성화해야 합니다. 이렇게 하면 Envoy가 아웃바운드 요청 헤더를 읽고 TLS 세션을 통해 적절한 대상으로 요청을 전달할 수 있습니다. 자세한 정보는 [전송 계층 보안 \(TLS\)](#)을 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#). 보안 취약점을 발견했다고 생각되거나 App Mesh의 보안에 대해 궁금한 점이 있으면 [AWS 취약성 보고 지침](#)을 참조하세요.

## Envoy 프록시 간의 연결이 TLS를 사용하고 있음을 어설션할 수 없음

### 증상

애플리케이션이 가상 노드 또는 가상 게이트웨이 리스너에서 TLS 종료를 활성화하거나 백엔드 TLS 클라이언트 정책에서 TLS 시작을 활성화했지만 Envoy 프록시 간의 연결이 TLS 협상 세션을 통해 발생하고 있음을 어설션할 수 없습니다.

### 해결 방법

이 해결 방법에 정의된 단계는 Envoy 관리 인터페이스와 Envoy 통계를 사용합니다. 이러한 항목을 구성하는 데 도움이 필요하면 [Envoy 프록시 관리 인터페이스 활성화 및 지표 오프로드를 위한 Envoy DogStats D 통합 활성화](#)를 참조하세요. 다음 통계 예제는 단순성을 위해 관리 인터페이스를 사용합니다.

- TLS 종료를 수행하는 Envoy 프록시의 경우:
  - 다음 명령을 사용하여 Envoy 구성에서 TLS 인증서가 부트스트랩되었는지 확인합니다.

```
curl http://my-app.default.svc.cluster.local:9901/certs
```

반환된 출력에서 TLS 종료에 사용된 인증서에 대한 항목이 `certificates[].cert_chain` 아래에 하나 이상 표시됩니다.

- 다음 예제 명령 및 출력에서 볼 수 있듯이 프록시 리스너에 대한 성공적인 인바운드 연결 수가 SSL 핸드셰이크 수에 재사용된 SSL 세션 수를 더한 값과 정확히 같은지 확인합니다.

```
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep
"listener.0.0.0.0_15000" | grep downstream_cx_total
listener.0.0.0.0_15000.downstream_cx_total: 11
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep
"listener.0.0.0.0_15000" | grep ssl.connection_error
listener.0.0.0.0_15000.ssl.connection_error: 1
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep
"listener.0.0.0.0_15000" | grep ssl.handshake
listener.0.0.0.0_15000.ssl.handshake: 9
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep
"listener.0.0.0.0_15000" | grep ssl.session_reused
listener.0.0.0.0_15000.ssl.session_reused: 1
Total CX (11) - SSL Connection Errors (1) == SSL Handshakes (9) + SSL Sessions
Re-used (1)
```

- TLS 시작을 수행하는 Envoy 프록시의 경우:
  - 다음 명령을 사용하여 Envoy 구성에서 TLS 트러스트 스토어가 부트스트랩되었는지 확인합니다.

```
curl http://my-app.default.svc.cluster.local:9901/certs
```

TLS 시작 시 백엔드 인증서를 검증하는 데 사용된 인증서에 대한 항목이 `certificates[].ca_certs` 아래에 하나 이상 표시됩니다.

- 다음 예제 명령 및 출력에서 볼 수 있듯이 백엔드 클러스터에 대한 성공적인 아웃바운드 연결 수가 SSL 핸드셰이크 수에 재사용된 SSL 세션 수를 더한 값과 정확히 같은지 확인합니다.

```

curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "virtual-node-
name" | grep upstream_cx_total
cluster.cds_egress_mesh-name_virtual-node-name_protocol_port.upstream_cx_total: 11
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "virtual-node-
name" | grep ssl.connection_error
cluster.cds_egress_mesh-name_virtual-node-name_protocol_port.ssl.connection_error:
1
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "virtual-node-
name" | grep ssl.handshake
cluster.cds_egress_mesh-name_virtual-node-name_protocol_port.ssl.handshake: 9
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "virtual-node-
name" | grep ssl.session_reused
cluster.cds_egress_mesh-name_virtual-node-name_protocol_port.ssl.session_reused: 1
Total CX (11) - SSL Connection Errors (1) == SSL Handshakes (9) + SSL Sessions
Re-used (1)

```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#). 보안 취약점을 발견했다고 생각되거나 App Mesh의 보안에 대해 궁금한 점이 있으면 [AWS 취약성 보고 지침](#)을 참조하세요.

## Elastic Load Balancing을 통한 TLS 문제 해결

### 증상

가상 노드에 대한 트래픽을 암호화하도록 Application Load Balancer 또는 Network Load Balancer를 구성하려고 하면 연결 및 로드 밸런서 상태 확인이 실패할 수 있습니다.

### 해결 방법

문제의 근본 원인을 파악하려면 다음을 확인해야 합니다.

- TLS 종료를 수행하는 Envoy 프록시의 경우 구성 오류를 제거해야 합니다. [TLS 클라이언트 정책을 사용하여 백엔드 가상 서비스에 연결할 수 없음](#)에서 위에 제공된 단계를 따릅니다.
- 로드 밸런서의 경우 TargetGroup:의 구성을 확인해야 합니다.
  - TargetGroup 포트가 가상 노드의 정의된 리스너 포트와 일치하는지 확인합니다.
  - HTTP를 통해 서비스에 대한 TLS 연결을 시작하는 Application Load Balancer의 경우 TargetGroup 프로토콜이 HTTPS로 설정되어 있는지 확인합니다. 상태 확인을 사용하는 경우 HealthCheckProtocol이 HTTPS로 설정되어 있는지 확인합니다.

- TCP를 통해 서비스에 대한 TCP 연결을 시작하는 Network Load Balancer의 경우 TargetGroup 프로토콜이 TLS로 설정되어 있는지 확인합니다. 상태 확인을 사용하는 경우 HealthCheckProtocol이 TCP로 설정되어 있는지 확인합니다.

#### Note

TargetGroup을 업데이트하려면 TargetGroup 이름을 변경해야 합니다.

이를 올바르게 구성했으면 로드 밸런서가 Envoy 프록시에 제공된 인증서를 사용하여 서비스에 대한 보안 연결을 제공해야 합니다.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#). 보안 취약점을 발견했다고 생각되거나 App Mesh의 보안에 대해 궁금한 점이 있으면 [AWS 취약성 보고 지침](#)을 참조하세요.

## App Mesh Kubernetes 문제 해결

이 주제에서는 Kubernetes에서 App Mesh를 사용할 때 발생할 수 있는 일반적인 문제를 자세히 설명합니다.

### Kubernetes에서 생성된 App Mesh 리소스를 App Mesh에서 찾을 수 없음

#### 증상

Kubernetes 사용자 지정 리소스 정의 (CRD) 를 사용하여 App Mesh 리소스를 생성했지만, 또는 API를 사용할 때 생성한 리소스가 App Mesh에 표시되지 않습니다. AWS Management Console

#### 해결 방법

App Mesh용 Kubernetes 컨트롤러의 오류가 원인일 수 있습니다. [자세한 내용은 문제 해결을 참조하십시오](#). GitHub 컨트롤러 로그에서 컨트롤러가 리소스를 생성할 수 없음을 나타내는 오류나 경고가 있는지 확인합니다.

```
kubectl logs -n appmesh-system -f \
$(kubectl get pods -n appmesh-system -o name | grep controller)
```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## Envoy 사이드카를 삽입한 후 포드의 준비 상태 및 활성화 상태 확인이 실패함

### 증상

애플리케이션용 포드는 이전에 성공적으로 실행되었지만 Envoy 사이드카를 포드에 삽입한 후에는 준비 상태 및 활성화 상태 확인이 실패하기 시작합니다.

### 해결 방법

포드에 삽입된 Envoy 컨테이너가 App Mesh의 Envoy Management Service로 부트스트랩되었는지 확인하세요. [Envoy가 오류 텍스트를 나타내며 App Mesh Envoy Management Service에서 연결이 끊길](#)에서 오류 코드를 참조하여 오류를 확인할 수 있습니다. 다음 명령을 사용하여 Envoy 로그에서 관련 포드를 검사할 수 있습니다.

```
kubectl logs -n appmesh-system -f \
 $(kubectl get pods -n appmesh-system -o name | grep controller) \
 | grep "gRPC config stream closed"
```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## 포드가 AWS Cloud Map 인스턴스로 등록 또는 등록 취소되지 않음

### 증상

Kubernetes 포드는 수명 주기의 AWS Cloud Map 일부로 등록 또는 등록 취소되지 않습니다. 포드가 성공적으로 시작되어 트래픽을 처리할 준비가 되었지만 수신이 불가능할 수 있습니다. 포드가 종료되더라도 클라이언트는 여전히 IP 주소를 유지하고 트래픽을 보내려고 시도하지만 실패할 수 있습니다.

### 해결 방법

이는 알려진 문제입니다. 자세한 내용은 문제가 있는 [Kubernetes에서 Pod가 자동 등록/등록 취소되지 않음](#)을 참조하십시오. AWS Cloud Map GitHub 포드, App Mesh 가상 노드 및 AWS Cloud Map 리소스 간의 관계로 인해 [쿠버네티스용 App Mesh 컨트롤러가 비동기화되어 리소스가 손실될](#) 수 있습니다. 예를 들어, 연결된 포드를 종료하기 전에 가상 노드 리소스를 Kubernetes에서 삭제하면 이러한 문제가 발생할 수 있습니다.

이 문제를 완화하려면:

- Kubernetes용 App Mesh 컨트롤러의 최신 버전을 실행하고 있는지 확인합니다.

- 가상 노드 정의에서 AWS Cloud Map namespaceName 및 가 올바른지 확인하세요serviceName.
- 가상 노드 정의를 삭제하기 전에 연결된 포드를 모두 삭제해야 합니다. 가상 노드와 연결된 포드를 식별하는 데 도움이 필요하면 [App Mesh 리소스에 대한 포드가 실행 중인 위치를 확인할 수 없음](#) 섹션을 참조하세요.
- 문제가 지속되면 다음 명령을 실행하여 컨트롤러 로그에서 근본적인 문제를 파악하는 데 도움이 될 수 있는 오류가 있는지 확인하세요.

```
kubectl logs -n appmesh-system \
 $(kubectl get pods -n appmesh-system -o name | grep appmesh-controller)
```

- 컨트롤러 포드를 다시 시작하려면 다음 명령을 사용해 보세요. 이렇게 하면 동기화 문제가 해결될 수 있습니다.

```
kubectl delete -n appmesh-system \
 $(kubectl get pods -n appmesh-system -o name | grep appmesh-controller)
```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## App Mesh 리소스에 대한 포드가 실행 중인 위치를 확인할 수 없음

### 증상

Kubernetes 클러스터에서 App Mesh를 실행하는 경우 운영자는 지정된 App Mesh 리소스에 대해 워크로드 또는 포드가 실행 중인 위치를 확인할 수 없습니다.

### 해결 방법

Kubernetes 포드 리소스에는 해당 리소스가 연결된 메시 및 가상 노드가 주석으로 표시됩니다. 다음 명령을 사용하여 지정된 가상 노드 이름에 대해 실행 중인 포드를 쿼리할 수 있습니다.

```
kubectl get pods --all-namespaces -o json | \
 jq '.items[] | { metadata } | select(.metadata.annotations."appmesh.k8s.aws/virtualNode" == "virtual-node-name")'
```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## 포드가 어떤 App Mesh 리소스로 실행되고 있는지 확인할 수 없음

### 증상

Kubernetes 클러스터에서 App Mesh를 실행하는 경우 운영자는 지정된 포드가 어떤 App Mesh 리소스로 실행되고 있는지 확인할 수 없습니다.

### 해결 방법

Kubernetes 포드 리소스에는 해당 리소스가 연결된 메시 및 가상 노드가 주석으로 표시됩니다. 다음 명령을 통해 포드를 직접 쿼리하여 메시 및 가상 노드 이름을 출력할 수 있습니다.

```
kubectl get pod pod-name -n namespace -o json | \
jq '{ "mesh": .metadata.annotations."appmesh.k8s.aws/mesh",
"virtualNode": .metadata.annotations."appmesh.k8s.aws/virtualNode" }'
```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

클라이언트 Envoy는 IMDSv1이 비활성화된 상태에서 App Mesh Envoy Management Service와 통신할 수 없습니다.

### 증상

IMDSv1이 비활성화되면 클라이언트 Envoy는 App Mesh 제어 영역(Envoy Management Service)과 통신할 수 없습니다. v1.24.0.0-prod 이전의 App Mesh Envoy 버전에서는 IMDSv2 지원이 제공되지 않습니다.

### 해결 방법

이 문제를 해결하려면 다음 세 가지 중 한 가지 방법을 시도하면 됩니다.

- IMDSv2가 지원되는 App Mesh Envoy 버전 v1.24.0.0-prod 이상으로 업그레이드하세요.
- Envoy가 실행 중인 인스턴스에서 IMDSv1을 다시 활성화합니다. IMDSv1 복원에 대한 지침은 [인스턴스 메타데이터 옵션 구성](#)을 참조하세요.
- 서비스가 Amazon EKS에서 실행 중인 경우 자격 증명을 가져올 때는 서비스 계정용 IAM 역할 (IRSA)을 사용하는 것이 좋습니다. IRSA 활성화 지침은 [서비스 계정의 IAM 역할](#)을 참조하세요.

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

App Mesh가 활성화되고 Envoy가 삽입된 경우 IRSA가 애플리케이션 컨테이너에서 작동하지 않음

### 증상

Amazon EKS용 App Mesh 컨트롤러를 사용하여 Amazon EKS 클러스터에서 App Mesh를 활성화하면 Envoy와 proxyinit 컨테이너가 애플리케이션 포드에 삽입됩니다. 애플리케이션은 IRSA를 가정할 수 없으며 대신 node role을 가정합니다. 포드 세부 정보를 설명하면 AWS\_WEB\_IDENTITY\_TOKEN\_FILE 또는 AWS\_ROLE\_ARN 환경 변수가 애플리케이션 컨테이너에 포함되어 있지 않다는 것을 알 수 있습니다.

## 해결 방법

AWS\_WEB\_IDENTITY\_TOKEN\_FILE 또는 AWS\_ROLE\_ARN 환경 변수가 정의된 경우 webhook는 포드를 건너뛵니다. 이러한 변수 중 어느 것도 제공하지 마세요. webhook이 자동으로 삽입합니다.

```
reservedKeys := map[string]string{
 "AWS_ROLE_ARN": "",
 "AWS_WEB_IDENTITY_TOKEN_FILE": "",
}
...
for _, env := range container.Env {
 if _, ok := reservedKeys[env.Name]; ok {
 reservedKeysDefined = true
 }
}
```

문제가 여전히 해결되지 않으면 문제를 제기하거나 [AWS Support에 GitHub 문의하세요](#).

## App Mesh 프리뷰 채널

App Mesh 프리뷰 채널은 us-west-2 리전에서 제공되는 App Mesh 서비스의 독특한 변형입니다. 프리뷰 채널은 곧 출시될 기능이 개발되면 시험해 볼 수 있도록 공개합니다. 프리뷰 채널의 기능을 사용할 때 GitHub를 통해 피드백을 제공하여 기능의 작동 방식을 조정할 수 있습니다. 프리뷰 채널에서 기능이 완벽하게 작동하고 필요한 통합 및 점검이 모두 완료되면 프로덕션 App Mesh 서비스로 전환됩니다.

AWS App Mesh 프리뷰 채널은 베타 서비스이며 해당 조건이 [AWS 서비스 약관](#)에 정의되어 있으므로 모든 기능은 프리뷰입니다. 프리뷰 채널 참여에는 계약 및 AWS 서비스 약관, 특히 유니버설 AWS 및 베타 서비스 참여 약관이 적용되며 기밀로 유지됩니다.

프리뷰 채널에 대해 자주 묻는 질문은 다음과 같습니다.

### 프리뷰 채널이란 무엇인가요?

프리뷰 채널은 새로운 서비스 기능이 정식 출시되기 전에 시험해보고 피드백을 제공할 수 있는 퍼블릭 서비스 엔드포인트입니다. 프리뷰 채널의 서비스 엔드포인트는 표준 프로덕션 엔드포인트와 분리되어 있습니다. AWS CLI, 프리뷰 채널용 서비스 모델 파일 및 AWS CLI에 대한 명령 입력 파일을 사용하여 엔드포인트와 상호 작용합니다. 프리뷰 채널을 사용하면 현재 프로덕션 인프라에 영향을 주지 않으면서 새로운 기능을 사용해 볼 수 있습니다. App Mesh가 고객의 가장 중요한 요구 사항을 충족할 수 있도록 App Mesh 팀에 [피드백을 제공](#)하는 것이 좋습니다. 프리뷰 채널에서 제공되는 기능에 대한 사용자의 피드백은 최상의 서비스를 제공할 수 있도록 App Mesh의 기능을 구체화하는 데 도움이 될 수 있습니다.

### 프리뷰 채널은 프로덕션 App Mesh와 어떻게 다른가요?

다음 표에는 프리뷰 채널과 다른 App Mesh 서비스의 측면이 나와 있습니다.

| 속성                                | App Mesh 프로덕션 서비스                                        | App Mesh 프리뷰 채널 서비스                                              |
|-----------------------------------|----------------------------------------------------------|------------------------------------------------------------------|
| Frontend endpoint                 | appmesh.us-west-2.<br>amazonaws.com                      | appmesh-preview.us-west-2.a<br>mazonaws.com                      |
| Envoy management service endpoint | appmesh-envoy-manage<br>ment.us-west-2.a<br>mazonaws.com | appmesh-preview-envoy-<br>management.us-west-2.am<br>azonaws.com |

|                   |                          |                                                                                                  |
|-------------------|--------------------------|--------------------------------------------------------------------------------------------------|
| CLI               | AWS App Mesh list-meshes | AWS App Mesh-preview list-meshes (only available after adding the Preview Channel service model) |
| Signing name      | appmesh                  | appmesh-preview                                                                                  |
| Service principal | appmesh.amazonaws.com    | appmesh-preview.amazonaws.com                                                                    |

### Note

App Mesh 프로덕션 서비스 표의 예제에는 us-west-2 리전이 나열되어 있지만 프로덕션 서비스는 대부분의 리전에서 사용할 수 있습니다. App Mesh 프로덕션 서비스를 사용할 수 있는 모든 리전 목록은 [AWS App Mesh 엔드포인트 및 할당량](#)을 참조하세요. 하지만 App Mesh 프리뷰 채널 서비스는 us-west-2 리전에서만 사용할 수 있습니다.

## 프리뷰 채널의 기능을 사용하려면 어떻게 해야 하나요?

1. 다음 명령을 사용하여 미리보기 채널 기능이 포함된 프리뷰 채널 서비스 모델을 AWS CLI에 추가합니다.

```
aws configure add-model \
 --service-name appmesh-preview \
 --service-model https://raw.githubusercontent.com/aws/aws-app-mesh-roadmap/main/appmesh-preview/service-model.json
```

2. 기능에 대한 [AWS App Mesh 사용 설명서](#)에 제공된 JSON 예제 및 지침을 기준으로 기능이 포함된 JSON 파일을 생성합니다.
3. 적절한 AWS CLI 명령 및 명령 입력 파일을 사용하여 기능을 구현합니다. 예를 들어 다음 명령은 *route.json* 파일을 사용하여 프리뷰 채널 기능이 포함된 경로를 생성합니다.

```
aws appmesh-preview create-route --cli-input-json file://route.json
```

4. Envoy 컨테이너를 Amazon ECS 태스크 정의, Kubernetes 포드 사양 또는 Amazon EC2 인스턴스에 추가할 때 해당 컨테이너의 구성 변수로 APPMESH\_PREVIEW = 1을 추가합니다. 이 변수를 사

용하면 Envoy 컨테이너가 프리뷰 채널 엔드포인트와 통신할 수 있습니다. 구성 변수를 추가하는 방법에 대한 자세한 내용은 [Amazon ECS의 서비스 업데이트](#), [Kubernetes의 서비스 업데이트](#) 및 [Amazon EC2의 서비스 업데이트](#)를 참조하세요.

## 피드백을 제공하려면 어떻게 해야 하나요?

기능에 대한 설명서에서 링크된 [App Mesh 로드맵 GitHub 리포지토리](#) 문제에 직접 피드백을 제공할 수 있습니다.

## 프리뷰 채널의 기능에 대한 피드백을 언제까지 제공해야 하나요?

피드백 기간은 도입되는 기능의 규모와 복잡성에 따라 달라집니다. 프리뷰 엔드포인트의 기능 릴리스와 프로덕션의 기능 릴리스 사이에 14일의 의견 제공 기간을 유지할 계획입니다. App Mesh 팀은 특정 기능에 대한 피드백 기간을 연장할 수 있습니다.

## 프리뷰 채널에는 어떤 수준의 지원이 제공되나요?

App Mesh [GitHub 로드맵 문제](#)에 피드백과 버그 보고서를 직접 제공하는 것이 좋지만, 공유해야 할 민감한 데이터가 있거나 일반 사용자에게 공개하는 것이 안전하지 않다고 생각되는 문제를 발견할 수 있다는 점을 이해합니다. 이러한 문제의 경우 AWS Support에 문의하거나 App Mesh 팀에 직접 [이메일을 보내](#) 피드백을 제공할 수 있습니다.

## 프리뷰 채널 엔드포인트에서 내 데이터는 안전한가요?

예. 프리뷰 채널에는 표준 프로덕션 엔드포인트와 동일한 수준의 보안이 제공됩니다.

## 내 구성을 사용할 수 있는 기간은 얼마나 되나요?

프리뷰 채널에서는 30일 동안 메시지를 사용할 수 있습니다. 메시지가 생성된 후 30일이 지나면 메시지를 나열하거나, 읽거나, 삭제할 수만 있습니다. 30일 후에 리소스를 생성하거나 업데이트하려고 하면 메시지가 보관된다는 BadRequest 예외 메시지가 표시됩니다.

## 프리뷰 채널을 사용하려면 어떤 도구를 사용할 수 있나요?

프리뷰 채널 서비스 모델 파일 및 명령 입력 파일과 함께 AWS CLI를 사용할 수 있습니다. 기능 사용에 대한 자세한 내용은 [프리뷰 채널의 기능을 사용하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요. AWS

CLI 명령 옵션, AWS Management Console, SDK 또는 AWS CloudFormation을 사용하여 프리뷰 채널 기능을 사용할 수는 없습니다. 하지만 프로덕션 서비스에 기능이 출시되면 모든 도구를 사용할 수 있습니다.

## 프리뷰 채널 API의 이후 버전과의 호환성이 제공되나요?

아니요. API는 피드백에 따라 변경될 수 있습니다.

## 프리뷰 채널 기능은 완전한가요?

아니요. 새 API 객체는 AWS Management Console, AWS CloudFormation 또는 AWS CloudTrail에 완전히 통합되지 않을 수 있습니다. 프리뷰 채널에서 기능이 구체화되고 곧 일반 공급될 예정이면 통합 기능을 사용할 수 있게 될 것입니다.

## 프리뷰 채널 기능에 대한 설명서가 제공되나요?

예. 프리뷰 채널 기능에 대한 설명서는 프로덕션 설명서에 포함됩니다. 예를 들어, 경로 리소스에 대한 기능이 프리뷰 채널에 릴리스되는 경우 해당 기능을 사용하는 방법에 대한 정보는 기존 [경로](#) 리소스 문서에 제공됩니다. 프리뷰 채널 기능은 프리뷰 채널에서만 사용할 수 있다는 레이블이 붙어 있습니다.

## 프리뷰 채널에서 새 기능을 사용할 수 있는 시점을 어떻게 알 수 있나요?

프리뷰 채널에 새로운 기능이 도입되면 [App Mesh 문서 기록](#)에 항목이 추가됩니다. 페이지를 정기적으로 검토하거나 [App Mesh 문서 기록 RSS 피드](#)를 구독할 수 있습니다. 또한 AWS App Mesh 로드맵 GitHub 리포지토리의 [문제](#)를 검토할 수 있습니다. 프리뷰 채널 서비스 모델 JSON 파일의 다운로드 링크는 프리뷰 채널로 릴리스될 때 문제에 추가됩니다. 모델 및 기능 사용 방법에 대한 자세한 내용은 [프리뷰 채널의 기능을 사용하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요.

## 기능이 프로덕션 서비스로 전환되는 시점을 어떻게 알 수 있나요?

프리뷰 채널에서만 기능을 사용할 수 있다는 내용을 담은 App Mesh 설명서의 텍스트가 제거되고 [App Mesh 문서 기록](#)에 항목이 추가됩니다. 페이지를 정기적으로 검토하거나 [App Mesh 문서 기록 RSS 피드](#)를 구독할 수 있습니다.

# App Mesh 서비스 할당량

AWS App Mesh는 중앙 위치에서 할당량을 보고 관리할 수 있는 AWS 서비스인 Service Quotas와 통합됩니다. 서비스 할당량을 제한이라고도 합니다. 자세한 내용은 Service Quotas 사용 설명서의 [Service Quotas라는?](#)을 참조하세요.

Service Quotas를 사용하면 모든 App Mesh 서비스 할당량의 값을 쉽게 찾을 수 있습니다.

AWS Management Console을 사용하여 App Mesh 서비스 할당량을 보려면

1. Service Quotas 콘솔(<https://console.aws.amazon.com/servicequotas/>)을 엽니다.
2. 탐색 창에서 AWS 서비스를 선택합니다.
3. AWS 서비스 목록에서 AWS App Mesh를 검색하여 선택합니다.

서비스 할당량 목록에서 서비스 할당량 이름, 적용된 값(제공된 경우), AWS 기본 할당량 및 할당량 값 조정 가능 여부를 확인할 수 있습니다.

4. 설명과 같은 서비스 할당량에 대한 추가 정보를 보려면 할당량 이름을 선택합니다.

할당량 증가를 요청하려면 Service Quotas 사용 설명서의 [할당량 증가 요청](#)을 참조하세요.

AWS CLI를 사용하여 App Mesh 서비스 할당량을 보려면

다음 명령을 실행합니다.

```
aws service-quotas list-aws-default-service-quotas \
 --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
 --service-code appmesh \
 --output table
```

AWS CLI를 사용하여 서비스 할당량에 대한 추가 태스크를 수행하려면 [Service Quotas AWS CLI 명령 참조](#)를 참조하세요.

# App Mesh의 문서 기록

다음 표에서 AWS App Mesh 사용 설명서의 중요한 업데이트 및 새 기능이 나와 있습니다. 사용자로부터 받은 의견을 수렴하기 위해 설명서가 자주 업데이트됩니다.

| 변경 사항                                                      | 설명                                                                                                                         | 날짜            |
|------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|---------------|
| <a href="#">업데이트된 AWSAppMeshFullAccess 정책</a>              | TagResource 및 UntagResource API에 대한 액세스를 AWSAppMeshFullAccess 허용하도록 업데이트되었습니다.                                             | 2024년 4월 24일  |
| <a href="#">CloudTrail 통합 설명서가 업데이트되었습니다.</a>              | App Mesh를 통합하여 API 활동을 CloudTrail 기록하는 방법을 설명하는 설명서가 업데이트되었습니다.                                                            | 2024년 3월 28일  |
| <a href="#">업데이트된 정책</a>                                   | AWSServiceRoleForAppMesh AWS Cloud Map DiscoverInstancesRevision API에 대한 액세스를 AWSAppMeshServiceRolePolicy 허용하도록 업데이트되었습니다. | 2023년 10월 12일 |
| <a href="#">App Mesh에 대한 VPC 엔드포인트 정책 지원</a>               | App Mesh에서 이제 VPC 엔드포인트 정책을 지원합니다.                                                                                         | 2023년 5월 11일  |
| <a href="#">App Mesh의 다중 리스너</a>                           | App Mesh는 이제 여러 리스너를 지원합니다.                                                                                                | 2022년 8월 18일  |
| <a href="#">App Mesh의 IPv6</a>                             | App Mesh는 이제 IPv6를 지원합니다.                                                                                                  | 2022년 5월 18일  |
| <a href="#">CloudTrail App Mesh Envoy 관리 서비스에 대한 로깅 지원</a> | 이제 App Mesh는 App Mesh Envoy 관리 서비스에 대한                                                                                     | 2022년 3월 18일  |

|                                                 |                                                                          |               |
|-------------------------------------------------|--------------------------------------------------------------------------|---------------|
|                                                 | CloudTrail 로깅 지원을 지원합니다.                                                 |               |
| <a href="#">App Mesh Agent for Envoy</a>        | App Mesh는 이제 Agent for Envoy를 지원합니다.                                     | 2022년 2월 25일  |
| <a href="#">App Mesh의 다중 리스너</a>                | ( <a href="#">App Mesh 프리뷰 채널</a> 만 해당) App Mesh에 대해 여러 리스너를 구현할 수 있습니다. | 2021년 11월 23일 |
| <a href="#">App Mesh의 ARM64 지원</a>              | App Mesh는 이제 ARM64를 지원합니다.                                               | 2021년 11월 19일 |
| <a href="#">App Mesh의 지표 확장</a>                 | App Mesh에 대한 지표 확장을 구현할 수 있습니다.                                          | 2021년 10월 29일 |
| <a href="#">수신 트래픽 개선 구현</a>                    | 호스트 이름 및 경로에 대해 호스트 이름 및 헤더 일치와 재작성을 구현할 수 있습니다.                         | 2021년 6월 14일  |
| <a href="#">상호 TLS 인증 구현</a>                    | 상호 TLS 인증을 구현할 수 있습니다.                                                   | 2021년 2월 4일   |
| <a href="#">af-south-1의 리전 시작</a>               | App Mesh는 이제 af-south-1 리전에서 사용할 수 있습니다.                                 | 2021년 1월 22일  |
| <a href="#">상호 TLS 인증 구현</a>                    | ( <a href="#">App Mesh 프리뷰 채널</a> 만 해당) 상호 TLS 인증을 구현할 수 있습니다.           | 2020년 11월 23일 |
| <a href="#">가상 게이트웨이 리스너를 위한 연결 풀링 구현</a>       | 가상 게이트웨이 리스너에 대한 연결 풀링을 구현할 수 있습니다.                                      | 2020년 11월 5일  |
| <a href="#">가상 노드 리스너에 대한 연결 풀링 및 이상치 탐지 구현</a> | 가상 노드 리스너에 대해 연결 풀링 및 이상치 탐지를 구현할 수 있습니다.                                | 2020년 11월 5일  |
| <a href="#">eu-south-1의 리전 시작</a>               | App Mesh는 이제 eu-south-1 리전에서 사용할 수 있습니다.                                 | 2020년 10월 21일 |

|                                                                                   |                                                                                                       |              |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|--------------|
| <a href="#">가상 게이트웨이 리스너를 위한 연결 풀링 구현</a>                                         | ( <a href="#">App Mesh 프리뷰 채널</a> 만 해당) 가상 게이트웨이 리스너에 대한 연결 풀링을 구현할 수 있습니다.                           | 2020년 9월 28일 |
| <a href="#">가상 노드 리스너에 대한 연결 풀링 및 이상치 탐지 구현</a>                                   | ( <a href="#">App Mesh 프리뷰 채널</a> 만 해당) 가상 노드 리스너에 대한 연결 풀링 및 이상치 탐지를 구현할 수 있습니다.                     | 2020년 9월 28일 |
| <a href="#">메시 인바운드를 위한 가상 게이트웨이 및 게이트웨이 경로 생성</a>                                | 메시 외부에 있는 리소스가 메시 내부에 있는 리소스와 통신할 수 있도록 합니다.                                                          | 2020년 7월 10일 |
| <a href="#">Kubernetes용 App Mesh 컨트롤러를 사용하여 Kubernetes에서 App Mesh 리소스 생성 및 관리</a> | Kubernetes에서 App Mesh 리소스를 생성하고 관리할 수 있습니다. 또한 컨트롤러는 사용자가 배포한 포드에 Envoy 프로세스 및 init 컨테이너를 자동으로 삽입합니다. | 2020년 6월 18일 |
| <a href="#">가상 노드 리스너 및 경로에 제한 시간 값 추가</a>                                        | 가상 노드 리스너 및 <a href="#">경로</a> 에 제한 시간 값을 추가할 수 있습니다.                                                 | 2020년 6월 18일 |
| <a href="#">가상 노드 리스너에 제한 시간 값 추가</a>                                             | ( <a href="#">App Mesh 프리뷰 채널</a> 만 해당) 가상 노드 리스너에 제한 시간 값을 추가할 수 있습니다.                               | 2020년 5월 29일 |
| <a href="#">메시 인바운드를 위한 가상 게이트웨이 생성</a>                                           | ( <a href="#">App Mesh 프리뷰 채널</a> 만 해당) 메시 외부의 리소스가 메시 내부의 리소스와 통신할 수 있도록 합니다.                        | 2020년 4월 8일  |

|                                                               |                                                                                                                                       |               |
|---------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|---------------|
| <a href="#"><u>TLS 암호화</u></a>                                | (App Mesh Preview Channel 전용) TLS를 사용하여 가상 노드 간 통신을 암호화하려면 해당 인증 기관 AWS Private Certificate Authority 또는 자체 인증 기관의 인증서를 사용하십시오.       | 2020년 1월 17일  |
| <a href="#"><u>메쉬를 다른 계정과 공유</u></a>                          | (App Mesh 프리뷰 채널만 해당) 다른 계정과 메쉬를 공유할 수 있습니다. 임의의 계정에서 생성된 리소스는 메쉬의 다른 리소스와 통신할 수 있습니다.                                                | 2020년 1월 17일  |
| <a href="#"><u>경로에 제한 시간 값 추가</u></a>                         | (App Mesh 프리뷰 채널만 해당) 경로에 제한 시간 값을 추가할 수 있습니다.                                                                                        | 2020년 1월 17일  |
| <a href="#"><u>AWS 아웃포스트에 App Mesh 프록시 생성</u></a>             | 아웃포스트에서 App Mesh Envoy 프록시를 생성할 수 있습니다. AWS                                                                                           | 2019년 12월 3일  |
| <a href="#"><u>경로, 가상 라우터, 가상 노드에 대한 HTTP/2 및 gRPC 지원</u></a> | HTTP/2 및 gRPC 프로토콜을 사용하는 트래픽을 라우팅할 수 있습니다. <a href="#"><u>가상 노드</u></a> 와 <a href="#"><u>가상 라우터</u></a> 에 이러한 프로토콜용 리스너를 추가할 수도 있습니다. | 2019년 10월 25일 |
| <a href="#"><u>재시도 정책</u></a>                                 | 클라이언트는 재시도 정책을 통해 간헐적 네트워크 장애 또는 간헐적 서버 측 오류로부터 스스로를 보호할 수 있습니다. 경로에 재시도 로직을 추가할 수 있습니다.                                              | 2019년 9월 10일  |

|                                                               |                                                                                                                                                                                                                          |              |
|---------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| <a href="#"><u>TLS 암호화</u></a>                                | (App Mesh 프리뷰 채널만 해당) TLS를 사용하여 가상 노드 간 통신을 암호화합니다.                                                                                                                                                                      | 2019년 9월 6일  |
| <a href="#"><u>HTTP 헤더 기반 라우팅</u></a>                         | 요청에 포함된 HTTP 헤더의 존재 여부와 해당 값을 기준으로 트래픽을 라우팅합니다.                                                                                                                                                                          | 2019년 8월 15일 |
| <a href="#"><u>App Mesh 프리뷰 채널의 가용성</u></a>                   | App Mesh 프리뷰 채널은 App Mesh 서비스가 독특하게 변형된 것입니다. 프리뷰 채널은 곧 출시될 기능을 개발하면서 시험해 볼 수 있도록 공개합니다. 미리보기 채널의 기능을 사용할 때 기능을 통해 피드백을 제공하여 기능의 작동 방식을 GitHub 조정할 수 있습니다.                                                               | 2019년 7월 19일 |
| <a href="#"><u>앱 메시 인터페이스 VPC 엔드포인트 (AWS PrivateLink)</u></a> | 인터페이스 VPC 엔드포인트를 사용하도록 App Mesh를 구성하여 VPC의 보안 태세를 향상시킬 수 있습니다. 인터페이스 엔드포인트는 프라이빗 IP 주소를 사용하여 App Mesh API에 비공개로 액세스할 수 있는 기술인 에 의해 AWS PrivateLink 구동됩니다. PrivateLink VPC와 App Mesh 사이의 모든 네트워크 트래픽을 Amazon 네트워크로 제한합니다. | 2019년 6월 14일 |

|                                                     |                                                                                                                                               |               |
|-----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| <a href="#">가상 노드 서비스 검색 AWS Cloud Map 방법으로 추가됨</a> | DNS를 AWS Cloud Map 지정하거나 가상 노드 서비스 검색 방법으로 지정할 수 있습니다. 서비스 검색에 AWS Cloud Map 를 사용하려면 계정에 App Mesh <a href="#">서비스 연결 역할</a> 이 있어야 합니다.        | 2019년 6월 13일  |
| <a href="#">Kubernetes에서 App Mesh 리소스를 자동으로 생성</a>  | App Mesh 리소스를 생성하고, Kubernetes에서 리소스를 생성할 때 App Mesh 사이드카 컨테이너 이미지를 Kubernetes 배포에 자동으로 추가합니다.                                                | 2019년 6월 11일  |
| <a href="#">App Mesh 일반 공급</a>                      | 이제 App Mesh 서비스가 프로덕션 용도로 일반 공급됩니다.                                                                                                           | 2019년 3월 27일  |
| <a href="#">App Mesh API 업데이트</a>                   | 사용 편의성을 개선하기 위해 App Mesh API가 업데이트되었습니다. 자세한 내용은 <a href="#">[FEATURE] 가상 라우터에 리스너 추가 및 [BUG] 일치하지 않는 포트 블랙홀이 있는 대상 가상 노드에 대한 경로를 참조</a> 하세요. | 2019년 3월 7일   |
| <a href="#">App Mesh 초기 릴리스</a>                     | 서비스 퍼블릭 프리뷰를 위한 최초 설명서                                                                                                                        | 2018년 11월 28일 |

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