ユーザーガイド

AWS App Mesh

AWS App Mesh: ユーザーガイド

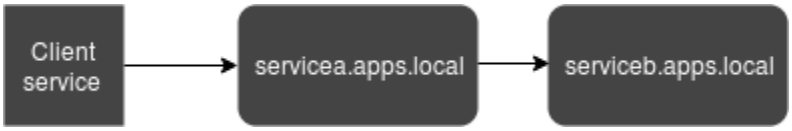
Table of Contents

AWS App Mesh とは?

AWS App Mesh は、サービスのモニタリングとコントロールを容易にするサービスメッシュです。サービスメッシュは、サービス間通信の処理専用のインフラストラクチャレイヤーであり、通常、アプリケーションコードと一緒にデプロイされる一連の軽量ネットワークプロキシを介して行われます。App Mesh は、サービスの通信方法を標準化し、エンドツーエンドの可視性を提供して、アプリケーションの高可用性を確保するのに役立ちます。App Mesh を使用すると、アプリケーション内のすべてのサービスについて一貫した可視性とネットワークトラフィックコントロールを実現できます。

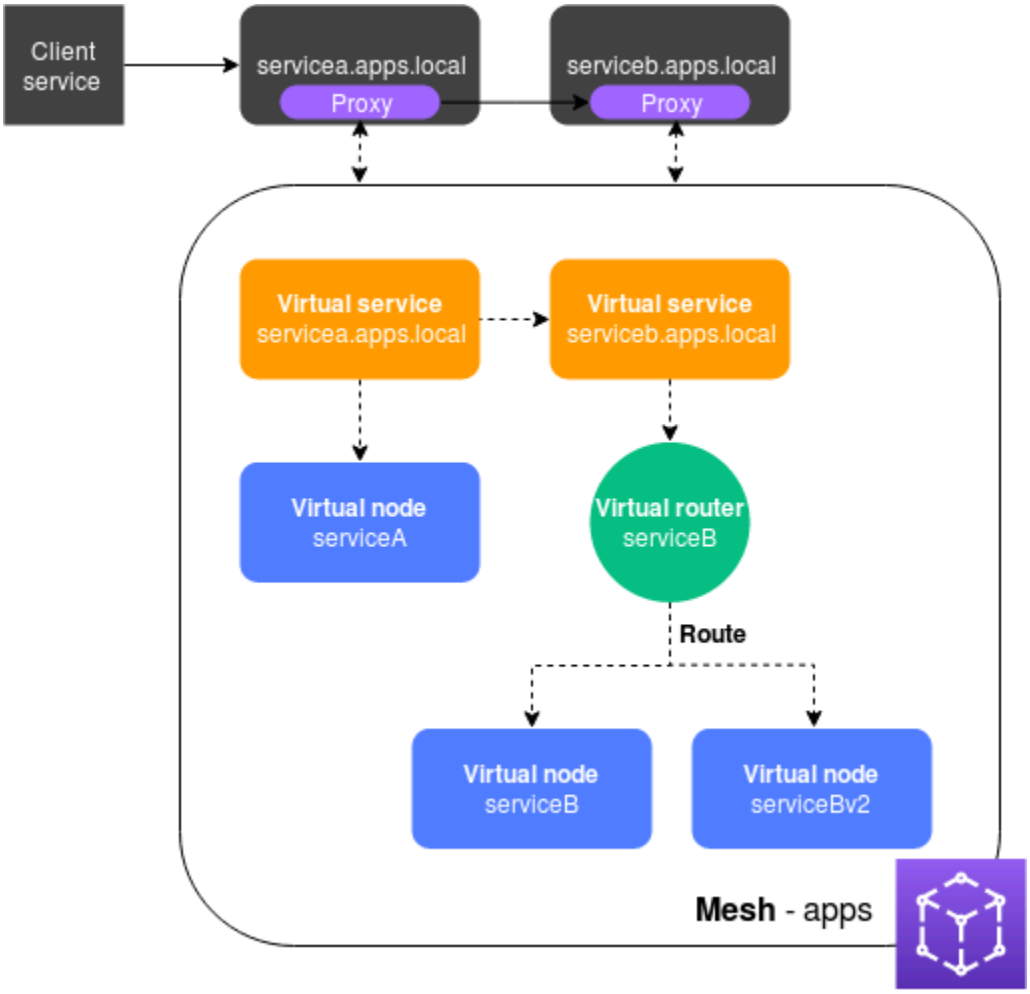
App Mesh をサンプルアプリケーションに追加する

App Mesh を使用しない、次の簡単なアプリケーション例を考えてみましょう。2つのサービスは、AWS Fargate、Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes Service (Amazon EKS)、Amazon Elastic Compute Cloud (Amazon EC2) インスタンス上の Kubernetes、または Docker を使用する AmazonEC2 インスタンスで実行できます。



この図では、`serviceA` と `serviceB` の両方が `apps.local` 名前空間を介して検出可能です。例えば `servicebv2.apps.local` という名前の `serviceb.apps.local` の新しいバージョンをデプロイすることにしたとします。次に、`servicea.apps.local` からのトラフィックの何パーセントかを `serviceb.apps.local` に、何パーセントかを `servicebv2.apps.local` に誘導したいとします。`servicebv2` が順調に実行されていることが確認できたら、そこに 100 パーセントのトラフィックを送ります。

App Mesh は、アプリケーションコードや登録されたサービス名を変更することなく、これを行うのに役立ちます。このサンプルアプリケーションで App Mesh を使用すると、メッシュは、次の図のようになります。



この設定では、サービスは、相互に直接通信しなくなります。代わりに、プロキシを介して相互に通信します。`servicea.apps.local` サービスとともにデプロイされたプロキシは、App Mesh 設定を読み取り、設定に基づいて、トラフィックを `serviceb.apps.local` または `servicebv2.apps.local` に送信します。

App Mesh のコンポーネント

App Mesh は、前の例に示すように、次のコンポーネントで設定されています。

- サービスメッシュ - サービスメッシュは、その中に存在するサービス間のネットワークトラフィックの論理的な境界です この例では、メッシュは apps という名前で、メッシュの他のすべてのリソースが含まれています。詳細については、「サービスメッシュ」を参照してください。
- 仮想サービス - 仮想サービスは、実際のサービスを抽象化したもので、仮想ノードが直接または間接的に仮想ルーターによって提供するものです。図では、2 つの仮想サービスが 2 つの実際のサービスを表しています。仮想サービスの名前は、実際のサービスの検出可能な名前です。仮想サービ

スと実際のサービスの名前が同じ場合、複数のサービスは、AppMesh が実装される前に使用していたのと同じ名前を使用して相互に通信できます。詳細については、「仮想サービス」を参照してください。

- 仮想ノード — 仮想ノードは、Amazon ECS や Kubernetes サービスなどの検出可能なサービスへの論理ポインタとして機能します。仮想サービスごとに、少なくとも 1 つの仮想ノードがあります。図では、`servicea.apps.local` 仮想サービスは、`serviceA` という名前の仮想ノードの設定情報を取得します。`serviceA` 仮想ノードは、サービス検出用の `servicea.apps.local` という名前で設定されます。`serviceb.apps.local` 仮想サービスは、`serviceB` という仮想ルーターを経由して `serviceB` と `serviceBv2` の仮想ノードにトラフィックをルーティングするように設定されています。詳細については、「仮想ノード」を参照してください。
- 仮想ルーターとルート – 仮想ルーターは、メッシュ内の 1 つ以上の仮想サービスのトラフィックを処理します。ルートは仮想ルータに関連付けられます。仮想ルーターとルート - 仮想ルーターは、メッシュ内の1つまたは複数の仮想サービス用のトラフィックを処理します。先ほどの図では、`serviceB` 仮想ルーターは、トラフィックの何割かを `serviceB` 仮想ノードに、何割かを `serviceBv2` 仮想ノードに向ける経路を持っています。特定の仮想ノードにルーティングされるトラフィックの割合を設定し、時間経過とともに変化させることができます。HTTP ヘッダー、URL パス、または gRPC サービス、メソッド名などの条件に基づいてトラフィックをルーティングできます。応答にエラーがある場合に、接続を再試行するように再試行ポリシーを設定できます。例えば、図の例では、ルートの再試行ポリシーで、`serviceb.apps.local` が特定のタイプのエラーを返した場合に、`serviceb.apps.local` への接続を5回再試行し、再試行の間隔を10秒にするように指定できます。詳細については、「仮想ルーター」と「ルート」を参照してください。
- プロキシ — メッシュとそのリソースを作成した後、プロキシを使用するようにサービスを設定します。プロキシは App Mesh 設定を読み取り、トラフィックを適切に転送します。図では、`servicea.apps.local` から `serviceb.apps.local` へのすべてのコミュニケーションは、各サービスとともにデプロイされたプロキシを経由します。サービスは、App Mesh を導入する前に使用したものと同じサービスディスカバリ名を使用して相互に通信します。プロキシは App Mesh 設定を読み取るため、2 つのサービスが相互に通信する方法を制御できます。App Mesh の設定を変更する場合は、サービス自体またはプロキシを変更または再デプロイする必要はありません。詳細については、「Envoy イメージ」を参照してください。

開始方法

App Mesh を使用するには、AWS Fargate、Amazon ECS、Amazon EKS、Kubernetes on Amazon EC2、または Amazon EC2 with Docker で既存のサービスが実行されている必要があります。

App Mesh の使用を開始するには、次のいずれかのガイドを参照してください。

- 「App Mesh と Amazon ECS の使用を開始する」
- 「App Mesh と Kubernetes の使用を開始する」
- 「App Mesh と Amazon EC2 の使用を開始する」

App Mesh にアクセスする

次の方法で App Mesh を使用できます。

AWS Management Console

コンソールは、AppMeshリソースの管理に使用できるブラウザベースのインターフェイスです。App Mesh コンソールは、https://console.aws.amazon.com/appmesh/ で開くことができます。

AWS CLI

一連のさまざまな AWS 製品用のコマンドを提供し、Windows、Mac、および Linux でサポートされています。開始するには、「AWS Command Line Interface ユーザーガイド」を参照してください。App Mesh の詳細については、「AWS CLI コマンドリファレンス」の appmesh を参照してください。

AWS Tools for Windows PowerShell

PowerShell 環境でスクリプトを作成するユーザー向けに、さまざまな AWS 製品用のコマンドが用意されています。開始するには、「AWS Tools for Windows PowerShell ユーザーガイド」を参照してください。App Mesh の cmdlets に関する詳細は、「AWS Tools for PowerShell Cmdlet リファレンス」の「App Mesh」を参照してください。

AWS CloudFormation

必要なすべての AWS リソースを記述するテンプレートを作成できます。テンプレートを使用して、AWS CloudFormation は、リソースをプロビジョニングおよび設定します。開始するには、「AWS CloudFormation ユーザーガイド」を参照してください。App Mesh リソースタイプの詳細については、「App Mesh リソースタイプ」の「AWS CloudFormation テンプレートリファレンス」を参照してください。

AWS SDK

また、さまざまなプログラミング言語から AppMesh にアクセスできるSDKも提供しています。SDK は、自動的に、次のようなタスクを処理します。

- サービスリクエストに暗号署名する
- リクエストを再試行する
- エラーレスポンスの処理をする

利用可能なSDKの詳細については、「アマゾンウェブサービス用のツール」を参照してください。

App Mesh API の詳細については、「AWS App MeshAPI リファレンス」を参照してください。

App Mesh の使用を開始する

App Mesh は、Amazon ECS、Kubernetes (独自の Amazon EC2 インスタンスにデプロイする、または Amazon EKS で実行しているもの) 、および Amazon EC2 にデプロイするアプリケーションで使用できます。App Mesh の使用を開始するには、App Mesh で使用するアプリケーションをデプロイしているサービスの 1 つをを選択します。スタートガイドの1つを完了した後は、他のサービスのアプリケーションがAppMeshでも機能するようにすることができます。

トピック

AWS App Mesh と Amazon ECS の開始方法

このトピックでは、Amazon ECS AWS App Mesh で実行されている実際のサービスで を使用する方法について説明します。このチュートリアルでは、複数の App Mesh リソースタイプのベーシックな機能について説明します。

シナリオ

App Mesh の使用方法を説明するために、次の特性を持つアプリケーションがあると仮定します。

- `serviceA` および `serviceB` という名前の 2 つのサービスで構成されています。
- どちらのサービスも、`apps.local` という名前の名前空間にメンバー登録されます。
- `ServiceA` は、HTTP/2、ポート 80 を介して `serviceB` と通信します。
- すでに `serviceB` のバージョン 2 をデプロイし、`serviceBv2` 名前空間に `apps.local` という名前でメンバー登録しました。

次の要件があります。

- serviceA から serviceB にトラフィックの 75% を送信し、serviceBv2 にトラフィックの 25% を送信して、serviceA からトラフィックの100%を送信する前に、serviceBv2 にバグがないことを検証します。
- トラフィックの重み付けを簡単に調整して、信頼性が証明されたら、トラフィックの 100% が serviceBv2 へ転送されるようにします。すべてのトラフィックが serviceBv2 に送信されたら、serviceB を切断します。
- 上記の要件を満たすために、実際のサービスの既存のアプリケーションコードまたはサービスディスカバリ登録を変更する必要はありません。

要件を満たすために、仮想サービス、仮想ノード、仮想ルーター、およびルートで、App Mesh サービスメッシュを作成することにします。メッシュを実装した後、サービスを更新して、Envoy プロキシを使用します。更新されると、サービスは相互に直接ではなく、Envoy プロキシを介して相互に通信します。

前提条件

- App Mesh の概念を既に理解している。詳細については、「AWS App Mesh とは?」を参照してください。
- Amazon ECS の概念に関する既存の理解。詳細については、Amazon Elastic Container Service デベロッパーガイドの「Amazon ECS とは」を参照してください。
- App Mesh は、DNS、 AWS Cloud Mapまたはその両方に登録されている Linux サービスをサポートします。この入門ガイドを使用するには、DNS に登録されている3つの既存のサービスをお勧めします。このトピックの手順は、既存のサービスが、serviceA、serviceB、serviceBv2 という名前で、すべてのサービスが apps.local という名前の名前空間を介して検出可能であることを前提としています。

  サービスが存在しない場合でもサービスメッシュとそのリソースを作成できますが、実際のサービスをデプロイするまでメッシュを使用することはできません。Amazon ECS でのサービスディスカバリの詳細については、「サービスディスカバリ」を参照してください。サービスディスカバリを使用して Amazon ECS サービスを作成するには、「チュートリアル: サービスディスカバリを使用したサービスの作成」を参照してください。サービスをまだ実行していない場合は、「サービスディスカバリを使用した Amazon ECS サービスを作成する」を参照してください。

ステップ 1: メッシュと仮想サービスを作成する

サービスメッシュは、サービス間のネットワークトラフィックの論理的な境界であり、サービスはその中に存在します。詳細については、「サービスメッシュ」を参照してください。仮想サービスは、実際のサービスを抽象化したものです。詳細については、「仮想サービス」を参照してください。

次の リソースを作成します。

- シナリオ内のすべてのサービスが apps 名前空間にメンバー登録されているため、apps.local という名前のメッシュ。
- serviceb.apps.local という名前の仮想サービス。仮想サービスは、その名前で検出可能なサービスを表しているため、別の名前をリファレンスするようにコードを変更したくないためです。servicea.apps.local という名前の仮想サービスが、次のステップで追加されます。

AWS Management Console または AWS CLI バージョン 1.18.116 以降または 2.0.38 以降を使用して、次のステップを完了できます。を使用している場合は AWS CLI、 aws --version コマンドを使用してインストールされている AWS CLI バージョンを確認します。バージョン 1.18.116 以降、または 2.0.38 以降をインストールしていない場合は、AWS CLIをインストールまたは更新する必要があります。使用するツールのタブを選択します。

AWS Management Console

1. App Mesh コンソールの初回実行ウィザードを https://console.aws.amazon.com/appmesh/get-started で開きます。
2. [メッシュ名] に **apps** と入力します。
3. [仮想サービス名] に **serviceb.apps.local** と入力します。
4. 続行するには、[次へ] を選択します。

AWS CLI

1. create-mesh コマンドを使用してメッシュを作成します。

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

2. create-virtual-service コマンドを使用して仮想サービスを作成します。

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

ステップ 2: 仮想ノードを作成する

仮想ノードは、実際のサービスの論理ポインタとして機能します。詳細については、「仮想ノード」を参照してください。

仮想ノードの 1 つが serviceB という名前の実際のサービスを表すため、serviceB という名前の仮想ノードを作成します。仮想ノードが表す実際のサービスは、serviceb.apps.local というホスト名を持つ DNS を介して検出可能です。または、 を使用して実際のサービスを検出することもできます AWS Cloud Map。仮想ノードは、ポート 80 で HTTP/2 プロトコルを使用してトラフィックをリッスンします。ヘルスチェックと同様に、その他のプロトコルもサポートされています。次のステップで、serviceA および serviceBv2 の仮想ノードを作成します。

AWS Management Console

1. [仮想ノード名] に **serviceB** と入力します。
2. [サービスディスカバリ] で、[DNS] を選択し、[DNS ホスト名] に **serviceb.apps.local** と入力します。
3. [リスナーの設定] で、[プロトコル] に [http2] を選択し、[ポート]に **80** と入力します。
4. 続行するには、[次へ] を選択します。

AWS CLI

1. 次の内容で、create-virtual-node-serviceb.json という名前のファイルを作成します。

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

```
            }
        ],
        "serviceDiscovery": {
            "dns": {
                "hostname": "serviceB.apps.local"
            }
        }
    },
    "virtualNodeName": "serviceB"
}
```

2. JSON ファイルを入力として使用して、create-virtual-node コマンドで仮想ノードを作成します。

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

ステップ 3: 仮想ルーターとルートを作成する

仮想ルーターは、メッシュ内の 1 つ以上の仮想サービスのトラフィックを送信します。詳細については、「仮想ルーター」および「ルート」を参照してください。

次の リソースを作成します。

- serviceB という名前の仮想ルーター。serviceB.apps.local 仮想サービスは、他のサービスとのアウトバウンド通信を開始しないためです。前に作成した仮想サービスは、実際の serviceb.apps.local サービスの抽象化であることに注意してください。仮想サービスは、仮想ルーターにトラフィックを送信します。仮想ルーターは、ポート 80 で HTTP/2 プロトコルを使用してトラフィックをリッスンします。その他のプロトコルもサポートされています。
- serviceB という名前のルート。このルートはトラフィックの 100% を serviceB 仮想ノードにルーティングします。重み付けは、serviceBv2 仮想ノードを追加した後のステップで行います。このガイドでは説明しませんが、ルートにフィルタ条件を追加したり、通信の問題が発生したときに Envoy プロキシが仮想ノードへのトラフィックの送信を複数回試行する再試行ポリシーを追加したりできます。

AWS Management Console

1. [仮想ルーター名] に **serviceB** と入力します。

2. [リスナーの設定] で、[プロトコル] に [http2] を選択して、[ポート] に **80** を指定します。
3. [ルート名] に **serviceB** と入力します。
4. [ルートタイプ] で、[http2] を選択します。
5. [ターゲット設定] の [仮想ノード名]で、[serviceB] を選択し、[重み]に **100** と入力します。
6. [一致設定] で、[方法] を選択します。
7. 続行するには、[次へ] を選択します。

AWS CLI

1. 仮想ルーターを作成します。

   a. 次の内容で、create-virtual-router.json という名前のファイルを作成します。

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

   b. JSON ファイルを入力として使用し、create-virtual-router コマンドで仮想ルーターを作成します。

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

2. ルートを作成します。

   a. 次の内容で、create-route.json という名前のファイルを作成します。

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

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

b. JSON ファイルを入力として使用し、create-route コマンドでルートを作成します。

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

ステップ 4: 確認して作成する

前のステップと照らし合わせて設定を確認します。

AWS Management Console

いずれかのセクションに変更を加える必要がある場合は、[編集] を選択します。設定が完了したら、[メッシュの作成] を選択します。

[ステータス] 画面には、作成されたすべてのメッシュリソースが表示されます。作成したリソースをコンソールに表示するには、[メッシュの表示] を選択します。

AWS CLI

describe-mesh コマンドで作成したメッシュの設定を確認します。

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

describe-virtual-service コマンドで作成した仮想サービスの設定を確認します。

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

describe-virtual-node コマンドで作成した仮想ノードの設定を確認します。

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

describe-virtual-router コマンドで作成した仮想ルーターの設定を確認します。

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

describe-route コマンドで作成したルートの設定を確認します。

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

ステップ 5: 追加のリソースを作成する

このシナリオを完了するには、次のことを行う必要があります。

- serviceBv2 という名前の仮想ノードと、serviceA という名前の別の仮想ノードを作成します。両方の仮想ノードは、HTTP/2 ポート 80 経由でリクエストをリッスンします。serviceA 仮想ノードには、serviceb.apps.local のバックエンドを設定します。serviceA 仮想ノードからのすべてのアウトバウンドトラフィックは、serviceb.apps.local という名前の仮想サービスに送信されます。このガイドでは説明しませんが、仮想ノードのアクセスログを書き込むファイルパスを指定することもできます。
- servicea.apps.local という名前の追加の仮想サービスを 1 つ作成します。これにより、すべてのトラフィックが serviceA 仮想ノードに直接送信されます。
- 前のステップで作成した serviceB ルートを更新して、トラフィックの 75% を serviceB 仮想ノードに送信し、25% を serviceBv2 仮想ノードに送信します。時間の経過とともに、serviceBv2 が 100% のトラフィックを受信するまで、継続して重みを変更することができます。すべてのトラフィックが serviceBv2 に送信されたら、serviceB 仮想ノードと実際のサービスをシャットダウンして中止することができます。重みを変更しても、serviceb.apps.local 仮想サービス名および実際のサービス名は変更されないため、コードを変更する必要はありません。serviceb.apps.local 仮想サービスは仮想ルーターにトラフィックを送信し、仮想ルーターはトラフィックを仮想ノードにルーティングすることに注意してください。仮想ノードのサービスディスカバリ名は、いつでも変更できます。

AWS Management Console

1. 左のナビゲーションペインで [メッシュ] を選択します。
2. 前のステップで作成した apps メッシュを選択します。
3. 左側のナビゲーションペインで、[仮想ノード] を選択します。
4. [仮想ノードの作成] を選択します。
5. [仮想ノード名] に **serviceBv2** と入力し、[サービスディスカバリ] で [DNS] を選択して、[DNS ホスト名] に **servicebv2.apps.local** と入力します。
6. [リスナーの設定] で、[プロトコル] に [http2] を選択し、[ポート] に **80** を入力します。
7. [仮想ノードの作成] を選択します。
8. [仮想ノードの作成] をもう一度選択します。[仮想ノード名] に **serviceA** と入力してください。[サービスディスカバリ] で [DNS] を選択し、[DNS ホスト名] に **servicea.apps.local** と入力します。
9. [新しいバックエンド] の下の [仮想サービス名の入力] に **serviceb.apps.local** と入力します。
10. [リスナーの設定] で、[プロトコル] に [http2] を選択し、[ポート] に **80** を入力して、[仮想ノードの作成] を選択します。
11. 左側のナビゲーションペインで [仮想ルーター] を選択し、リストから [serviceB] 仮想ルーターを選択します。
12. [ルート] で、前のステップで作成した ServiceB という名前のルートを選択し、[編集] を選択します。
13. [ターゲット] の仮想ノード名で、serviceB の [重み] の値を **75** に変更します。
14. [ターゲットの追加] を選択し、ドロップダウンリストから serviceBv2を選択して、[重み] の値を **25** に設定します。
15. [保存] を選択します。
16. 左側のナビゲーションペインで、[仮想サービス] を選択し、[仮想サービスの作成] を選択します。
17. [仮想サービス名] に **servicea.apps.local** と入力し、[プロバイダー] に [仮想ノード] を選択し、[仮想ノード] に serviceA を選択し、[仮想サービスの作成]を選択します。

AWS CLI

1. serviceBv2 仮想ノードを作成します。

a. 次の内容で、create-virtual-node-servicebv2.json という名前のファイルを作成します。

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

b. 仮想ノードを作成します。

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

2. serviceA 仮想ノードを作成します。

a. 次の内容で、create-virtual-node-servicea.json という名前のファイルを作成します。

```
{
   "meshName" : "apps",
   "spec" : {
      "backends" : [
         {
            "virtualService" : {
               "virtualServiceName" : "serviceb.apps.local"
            }
         }
```

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

b. 仮想ノードを作成します。

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

3. 前のステップで作成した serviceb.apps.local 仮想サービスを更新して、そのトラフィックを serviceB 仮想ルーターに送信します。仮想サービスが最初に作成された時点では、serviceB 仮想ルーターがまだ作成されていないため、トラフィックはどこにも送信されませんでした。

a. 次の内容で、update-virtual-service.json という名前のファイルを作成します。

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

b. update-virtual-service コマンドを使用して、仮想サービスを更新します。

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

4. 前のステップで作成した serviceB ルートを更新します。

   a. 次の内容で、update-route.json という名前のファイルを作成します。

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

   b. update-route コマンドを使用してルートを更新します。

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

5. serviceA 仮想サービスを作成します。

   a. 次の内容で、create-virtual-servicea.json という名前のファイルを作成します。

```
{
```

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

b. 仮想サービスを作成します。

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

メッシュの概要

サービスメッシュを作成する前に、servicea.apps.local、serviceb.apps.local、および servicebv2.apps.local という 3 つの実際のサービスがありました。実際のサービスに加えて、実際のサービスを表す次のリソースを含むサービスメッシュが作成されました。

- 2 つの仮想サービス。プロキシは、仮想ルーターを経由して、servicea.apps.local 仮想サービスからのすべてのトラフィックを serviceb.apps.local 仮想サービスに送信します。
- serviceA、serviceB、および serviceBv2 という名前の 3 つの仮想ノード。Envoy プロキシは、仮想ノードに対して設定されたサービスディスカバリ情報を使用して、実際のサービスの IP アドレスを検索します。
- Envoy プロキシがインバウンドトラフィックの 75% を serviceB 仮想ノードに、25% を serviceBv2 仮想ノードにルーティングするように指定する 1 つのルートを持つ仮想ルーター。

ステップ 6: サービスを更新する

メッシュを作成したら、次のタスクを完了する必要があります。

- 各 Amazon ECS タスクでデプロイする Envoy プロキシに、1 つ以上の仮想ノードの設定を読み取りすることを許可します。プロキシの認証方法の詳細については、「プロキシの認証」を参照してください。
- 既存の各 Amazon ECS タスク定義を更新して、Envoy プロキシを使用します。

認証情報

Envoy コンテナには、App Mesh サービスに送信されるリクエストに署名するための AWS Identity and Access Management 認証情報が必要です。Amazon EC2 起動タイプでデプロイされた Amazon ECS タスクの場合、認証情報はインスタンスのロールまたは、タスクの IAM ロールから取得できます。Linux コンテナの Fargate を使用してデプロイされた Amazon ECS タスクは、インスタンス IAM プロファイル認証情報を提供する Amazon EC2 メタデータサーバーにアクセスできません。認証情報を提供するには、Linux コンテナの Fargate タイプを使用してデプロイされたタスクに IAM タスクのロールをアタッチする必要があります。

タスクが Amazon EC2 起動タイプでデプロイされ、Amazon EC2 メタデータサーバーへのアクセスがブロックされている場合、タスク用の IAM ロールの重要な注釈で説明されているように、タスク IAM ロールもタスクに添付する必要があります。インスタンスまたはタスクに割り当てるロールには、プロキシ認可で説明するように IAM ポリシーが添付されている必要があります。

を使用してタスク定義を更新するには AWS CLI

Amazon ECS AWS CLI コマンド を使用しますregister-task-definition。以下のタスク定義の例は、サービスの App Mesh を設定する方法を示しています。

> **Note**
>
> コンソールを使用した Amazon ECS の App Mesh の設定は利用できません。

タスク定義 json

プロキシ設定

App Mesh を使用するように Amazon ECS サービスを設定するには、サービスのタスク定義に次のプロキシ設定セクションがある必要があります。プロキシ設定 `type` を `APPMESH` に、`containerName` を `envoy` に設定します。これに応じて、次のプロパティ値を設定します。

`IgnoredUID`

Envoy プロキシは、このユーザー ID を使用するプロセスからのトラフィックをルーティングしません。このプロパティ値には任意のユーザー ID を選択できますが、この ID はタスク定義の Envoy コンテナの `user` ID と同じである必要があります。この一致により、Envoy はプロキシ

を使用せずに、それ自体のトラフィックを無視することができます。例では、履歴上の目的で *1337* を使用します。

`ProxyIngressPort`

これは、Envoy プロキシのコンテナのインバウンドポートです。この値は `15000` に設定します。

`ProxyEgressPort`

これは、Envoy プロキシのコンテナのアウトバウンドポートです。この値は `15001` に設定します。

`AppPorts`

アプリケーションコンテナがリッスンするインバウンドポートを指定します。この例では、アプリケーションコンテナはポート *9080* でリッスンします。指定するポートは、仮想ノードリスナーで設定されたポートと一致する必要があります。

`EgressIgnoredIPs`

Envoy は、これらの IP アドレスにトラフィックをプロキシしません。この値を `169.254.170.2,169.254.169.254` に設定することで、Amazon EC2 メタデータサーバーと Amazon ECS タスクのメタデータエンドポイントを無視します。メタデータのエンドポイントは、タスクの認証情報用に IAM ロールを提供します。さらにアドレスを追加できます。

`EgressIgnoredPorts`

コンマで区切られたポートのリストを追加できます。Envoy は、これらのポートにトラフィックをプロキシしません。ポートがない場合でも、ポート 22 は無視されます。

> **Note**
>
> 無視できるアウトバウンドポートの最大数は 15 です。

```
"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"
  }
 ]
}
```

アプリケーションコンテナ Envoy の依存関係

タスク定義のアプリケーションコンテナは開始する前に Envoy プロキシがブートストラップして起動するのを待機する必要があります。これを確実に行うには、各アプリケーションコンテナの定義に dependsOn セクションを設定して、Envoy コンテナが HEALTHY としてレポートするのを待ちます。次のコードは、この依存関係があるアプリケーションコンテナの定義の例を示しています。次の例のすべてのプロパティが必須です。一部のプロパティ値も必須ですが、######なものもあります。

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

```
  "condition": "HEALTHY"
 }]
}
```

Envoy コンテナの定義

Amazon ECS タスク定義には、App Mesh Envoy コンテナイメージを含める必要があります。

me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1、af-south-1 以外のサポートされているリージョンすべて。*Region-code* は、me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1、af-south-1 以外の任意のリージョンに置き換えることができます。

標準

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

me-south-1

標準

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

ap-east-1

標準

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

ap-southeast-3

標準

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-
fips
```

eu-south-1

標準

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

il-central-1

標準

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-
fips
```

af-south-1

標準

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

Public repository

標準

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

Important

App Mesh での使用は、バージョン v1.9.0.0-prod 以降のみサポートされています。

Envoy プロジェクトチームが App Mesh をサポートする変更をマージをするまでは、App Mesh Envoy コンテナイメージを使用する必要があります。詳細については、GitHub 「ロードマップの問題」を参照してください。

次の例のすべてのプロパティが必須です。一部のプロパティ値も必須ですが、######なものもあります。

Note

- Envoy のコンテナの定義は essential とマークされる必要があります。
- Envoy コンテナに 512 CPU ユニットと少なくとも64 MiB のメモリを割り当てるようお勧めします。Fargate では、設定できる最低メモリは 1024 MiB です。
- Amazon ECS サービスの仮想ノード名は、APPMESH_RESOURCE_ARN プロパティの値に設定する必要があります。このプロパティには、Envoy イメージのバージョン 1.15.0 以降が必要です。詳細については、「Envoy」を参照してください。
- user 設定の値は、タスク定義のプロキシ設定の IgnoredUID 値と一致する必要があります。この例では、*1337* を使用します。

- ここに示されているヘルスチェックは、Envoy コンテナが正常にブートストラップするのを待機して、Envoy コンテナが正常な状態であり、アプリケーションコンテナが開始する準備ができていることを Amazon ECS に報告します。
- デフォルトでは、App Mesh は、Envoy によってメトリクスとトレースでそれ自体が参照されるとき、APPMESH_RESOURCE_ARN で指定したリソースの名前を使用します。APPMESH_RESOURCE_CLUSTER 環境変数に独自の名前を設定することで、この動作を上書きできます。このプロパティには、Envoy イメージのバージョン 1.15.0 以降が必要です。詳細については、「Envoy」を参照してください。

次のコードは Envoy コンテナの定義の例を示しています。

```
{
 "name": "envoy",
 "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-
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"
}
```

タスク定義の例

次の Amazon ECS タスク定義例は、上記の例を taskB のタスク定義にマージする方法を示しています。ここでは、 AWS X-Rayの使用の有無にかかわらず、両方の Amazon ECS 起動タイプのタスクを作成するための例を示します。必要に応じて、####な値を変更し、シナリオから taskBv2 および taskA という名前のタスクの定義を作成します。メッシュ名と仮想ノード名を

APPMESH_RESOURCE_ARN 値に置き換え、アプリケーションがリッスンするポートのリストをプロキシ設定の AppPorts 値に置き換えます。デフォルトでは、App Mesh は、Envoy によってメトリクスとトレースでそれ自体が参照されるとき、APPMESH_RESOURCE_ARN で指定したリソースの名前を使用します。APPMESH_RESOURCE_CLUSTER 環境変数に独自の名前を設定することで、この動作を上書きできます。次の例のすべてのプロパティは必須です。一部のプロパティ値も必須ですが、*######*なものもあります。

「認証情報」セクション タスクで説明されているように、Amazon ECS タスクを実行している場合は、既存のタスク IAM ロールを例に追加する必要があります。

> **Important**
>
> Fargate は 1024 より大きいポート値を使用する必要があります。

Example Amazon ECS タスク定義の JSON - Linux コンテナの 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:v1.29.5.0-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" : "awsvpc"
}
```

Example を使用した Amazon ECS タスク定義の JSON AWS X-Ray - Linux コンテナ上の Fargate

X-Ray を使用すると、アプリケーションが処理するリクエストに関するデータ収集が可能になります。また、トラフィックフローを視覚化するために使用できるツールが提供されます。Envoy 用の X-Ray ドライバーを使用すると、Envoy はトレース情報を X-Ray に報告することができます。Envoy の設定で、X-Rayトレースを有効にすることができます。設定に基づいて、Envoy は、サイドカーコンテナとして実行されている X-Ray デーモンにトレースデータを送信し、デーモンは、トレースを X-Ray サービスに転送します。トレースが X-Ray に発行されたら、X-Ray コンソールを使用してサービス呼び出しグラフを視覚化し、トレースの詳細をリクエストできます。次の JSON は、X-Ray の統合を有効にするためのタスク定義を表しています。

```
{


   "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:v1.29.5.0-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 Amazon ECS タスク定義の JSON - EC2 起動タイプ

```
{
  "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:v1.29.5.0-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 を使用した Amazon ECS タスク定義の JSON AWS X-Ray - EC2 起動タイプ

```
{
```

```
  "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:v1.29.5.0-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"
}
```

高度なトピック

App Mesh を使用した canary デプロイ

canary デプロイ/リリースは、アプリケーションの古いバージョンと新しくデプロイされたバージョンの間でトラフィックを切り替えるのに役立ちます。また、新しくデプロイされたバージョンのヘルスも監視します。新しいバージョンに問題がある場合、canary デプロイはトラフィックを古いバージョンに自動的に切り替えることができます。canary デプロイでは、アプリケーションのバージョン間でトラフィックを詳細に制御して切り替えることができます。

App Mesh を使用して Amazon ECS の canary デプロイを実装する方法の詳細については、「App Mesh を使用して Amazon ECS の canary デプロイを使用したパイプラインを作成する」を参照してください。

> **Note**
>
> App Mesh のその他の例とチュートリアルについては、App Mesh サンプルリポジトリを参照してください。

AWS App Mesh および Kubernetes の開始方法

Kubernetes 用 App Mesh コントローラーを使用して Kubernetes AWS App Mesh と統合する場合、メッシュ、仮想サービス、仮想ノード、仮想ルーター、Kubernetes 経由のルートなどの App Mesh リソースを管理します。また、Kubernetes ポッド仕様に App Mesh サイドカーコンテナイメージを自動的に追加できます。このチュートリアルでは、Kubernetes 用 App Mesh コントローラーをインストールして、この統合を有効にする方法について説明します。

コントローラーには、Kubernetes カスタムリソース定義 `meshes`、`virtual services`、`virtual nodes`、`virtual routers` のデプロイが備わっています。コントローラーは、カスタムリソースの作成、変更、削除を監視し、App Mesh API を通じて、対応する App Mesh の the section called “メッシュ” the section called “仮想サービス”、the section called “仮想ノード”、the section called “仮想ゲートウェイ”、the section called “ゲートウェイルート”、the section called “仮想ルーター” (the section called “ルート” を含む)などのリソースを変更します。詳細を確認したり、コントローラーに貢献したりするには、GitHub「」プロジェクトを参照してください。

コントローラーは、指定した名前でラベル付けされた Kubernetes ポッドに、次のコンテナを挿入する Webhook もインストールします。

- App Mesh Envoy プロキシ — Envoy は、App Mesh コントロールプレーンで定義されている設定を使用して、アプリケーショントラフィックの送信先を決定します。
- App Mesh プロキシルートマネージャー– Envoyを介してインバウンドトラフィックとアウトバウンドトラフィックをルーティングするポッドのネットワーク名前空間の `iptables` ルールを更新します。このコンテナは、ポッド内の Kubernetes init コンテナとして実行されます。

前提条件

- App Mesh の概念を既に理解している。詳細については、「AWS App Mesh とは?」を参照してください。
- Kubernetes の概念を既に理解している。詳細については、Kubernetes ドキュメントの「Kubernetes とは」を参照してください。
- 既存の Kubernetes クラスター。既存のクラスターがない場合は、「Amazon EKS ユーザーガイド」の「Amazon EKS の開始方法」を参照してください。Amazon EC2 で独自の Kubernetes クラスターを実行している場合は、Envoy イメージがある Amazon ECR リポジトリに対して Docker が認証されていることを確認してください。詳細については、「Amazon Elastic Container Registry ユーザーガイド」の「Envoy イメージ」、「レジストリの認証」、および「Kubernetes ドキュメント」の「プライベートレジストリからイメージをプルする」を参照してください。
- App Mesh は、DNS に登録されている Linux サービス AWS Cloud Map、またはその両方をサポートしています。この入門ガイドを使用するには、DNS に登録されている3つの既存のサービスをお勧めします。このトピックの手順は、既存のサービスが、`serviceA`、`serviceB`、`serviceBv2` という名前で、すべてのサービスが `apps.local` という名前の名前空間を介して検出可能であることを前提としています。

サービスが存在しない場合でもサービスメッシュとそのリソースを作成できますが、実際のサービスをデプロイするまでメッシュを使用することはできません。

- AWS CLI バージョン 1.18.116 以降または 2.0.38 以降がインストールされている。をインストールまたはアップグレードするには AWS CLI、「 のインストール AWS CLI」を参照してください。
- Kubernetes クラスターと通信するよう設定されている kubectl クライアント。Amazon Elastic Kubernetes Service を使用している場合は、kubectl のインストール手順と kubeconfig ファイルの設定手順を実行してください。
- Helm バージョン 3.0 以降がインストールされています。Helm がインストールされていない場合は、「Amazon EKS ユーザーガイド」の「Amazon EKS での Helm の使用」を参照してください。
- Amazon EKS は現在、IPv4_ONLY および IPv6_ONLY の IP 設定にのみ対応しています。これは、Amazon EKS が IPv4 トラフィックのみまたは IPv6 トラフィックのみを処理できるポッドのみを現在サポートしているためです。

残りのステップでは、実際のサービスが serviceA、serviceB、serviceBv2 という名前で、すべてのサービスが apps.local という名前の名前空間を介して検出可能であることを前提としています。

ステップ 1: 統合コンポーネントをインストールする

App Mesh で使用するポッドをホストする各クラスターに、統合コンポーネントを 1 回インストールします。

統合コンポーネントをインストールするには

1. この手順の残りのステップでは、プレリリースバージョンのコントローラーがインストールされていないクラスターが必要です。プレリリースバージョンをインストールした場合、またはインストールしたかどうかが不明な場合は、プレリリースバージョンがクラスターにインストールされているかどうかを確認するスクリプトをダウンロードし、実行できます。

```
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
```

スクリプトが `Your cluster is ready for upgrade. Please proceed to the installation instructions` を返した場合は、次のステップに進むことができます。別のメッセージが返された場合は、続行する前にアップグレード手順を完了する必要があります。プレリリースバージョンのアップグレードの詳細については、「 でのアップグレード」を参照してください GitHub。

2. `eks-charts` リポジトリを Helm に追加します。

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

3. App Mesh Kubernetes カスタムリソース定義 (CRD) をインストールします。

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

4. コントローラーの Kubernetes 名前空間を作成します。

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

5. 後の手順で使用するために、次の数を設定します。*`cluster-name`* と *`Region-code`* を既存のクラスターの値に置き換えます。

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

6. (オプション) Fargate でコントローラーを実行する場合は、Fargate プロファイルを作成する必要があります。`eksctl` をインストールしていない場合は、「Amazon EKS ユーザーガイド」の「`eksctl` のインストールまたはアップグレード」を参照してください。コンソールを使用してプロファイルを作成する場合は、「Amazon EKS ユーザーガイド」の「Fargate プロファイルの作成」を参照してください。

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

7. クラスターの OpenID 接続 (OIDC) ID プロバイダーを作成します。`eksctl` をインストールしていない場合、「Amazon EKS ユーザーガイド」の「`eksctl` のインストールまたはアップグレード」の手順に従ってインストールできます。コンソールを使用してプロバイダーを作成する場合、「Amazon EKS ユーザーガイド」の「クラスターでのサービスアカウントの IAM ロールの有効化」を参照してください。

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

8. IAM ロールを作成し、 AWSAppMeshFullAccess および AWSCloudMapFullAccess AWS 管理ポリシーをアタッチして、Kubernetes `appmesh-controller` サービスアカウントにバインドします。このロールにより、コントローラーは App Mesh リソースの追加、削除、変更を行うことができます。

> **Note**
>
> コマンドは、自動生成された名前で AWS IAM ロールを作成します。作成された IAM ロール名を指定することはできません。

```
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 または を使用してサービスアカウントを作成する場合は AWS CLI、「Amazon EKS ユーザーガイド」の「サービスアカウントの IAM ロールとポリシーの作成」を参照してください。 AWS Management Console または を使用してアカウント AWS CLI を作成する場合は、ロールを Kubernetes サービスアカウントにマッピングする必要もあります。詳細については、「Amazon EKS ユーザーガイド」の「サービスアカウントの IAM ロールを指定する」を参照してください。

9. App Mesh コントローラーをデプロイします。すべての設定オプションのリストについては、「 の設定」を参照してください GitHub。

   1. プライベートクラスターの App Mesh コントローラーをデプロイするには、まず、リンクされたプライベートサブネットへの App Mesh エンドポイントおよびサービス検出 Amazon VPC エンドポイントを有効にする必要があります。また、`accountId` を設定する必要があります。

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

プライベートクラスターで X-Ray トレースを有効にするには、X-Ray エンドポイントおよび Amazon ECR Amazon VPC エンドポイントを有効にします。コントローラーはデフォルトで `public.ecr.aws/xray/aws-xray-daemon:latest` を使用するため、このイメージをローカルにプルし、ECR 個人リポジトリにプッシュします。

> **Note**
>
> 現在、Amazon VPC エンドポイントは Amazon ECR パブリックリポジトリをサポートしていません。

X-Ray の設定でコントローラーをデプロイする例を以下に示します。

```
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
```

アプリケーションのデプロイを仮想ノードまたはゲートウェイにバインドするときに、X-Ray デーモンが正常に挿入されるかどうかを検証します。

詳細については、「Amazon EKS ユーザーガイド」の「プライベートクラスター」を参照してください。

2. 他のクラスターの App Mesh コントローラーをデプロイします。すべての設定オプションのリストについては、「 の設定」を参照してください 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 クラスターファミリーが IPv6 の場合、App Mesh コントローラーをデプロイするときに、前のコマンド `--set clusterName=$CLUSTER_NAME` に次のオプションを追加してクラスター名を設定してください。

> **Important**
>
> クラスターが `me-south-1`、`ap-east-1`、`ap-southeast-3`、`eu-south-1`、`il-central-1`、または `af-south-1` リージョンにある場合は、前のコマンドに次のオプションを追加する必要があります。
> *`account-id`*と*`Region-code`*を適切な値のセットの 1 つに置き換えます。
>
> - サイドカーイメージの場合:
>   - `--set image.repository=`*`account-id`*`.dkr.ecr.`*`Region-code`*`.amazonaws.com/amazon/appmesh-controller`
>   - 772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
>   - 856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
>   - 909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
>   - 422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
>   - 564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
>   - 924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod

- 古いイメージ URIsは、 の変更ログにあります GitHub。イメージが存在する AWS アカウントがバージョン で変更されましたv1.5.0。以前のバージョンのイメージは、Amazon Elastic Kubernetes Service の Amazon コンテナイメージレジストリにある AWS アカウントでホストされています。
- コントローラーイメージの場合:
  - ```
    --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:v1.13.0
  - 856666278305.dkr.ecr.ap-east-1.amazonaws.com/amazon/appmesh-controller:v1.13.0
  - 909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/amazon/appmesh-controller:v1.13.0
  - 422531588944.dkr.ecr.eu-south-1.amazonaws.com/amazon/appmesh-controller:v1.13.0
  - 564877687649.dkr.ecr.il-central-1.amazonaws.com/amazon/appmesh-controller:v1.13.0
  - 924023996002.dkr.ecr.af-south-1.amazonaws.com/amazon/appmesh-controller:v1.13.0
- サイドカー init イメージの場合:
  - ```
    --set sidecar.image.repository=account-id.dkr.ecr.Region-code.amazonaws.com/aws-appmesh-envoy
    ```
  - 772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-proxy-route-manager:v7-prod
  - 856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-proxy-route-manager:v7-prod
  - 909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-proxy-route-manager:v7-prod
  - 422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-proxy-route-manager:v7-prod
  - 564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-proxy-route-manager:v7-prod

- 924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-proxy-route-manager:v7-prod

> **Important**
>
> App Mesh での使用は、バージョン v1.9.0.0-prod 以降のみサポートされています。

10. コントローラーのバージョンが `v1.4.0` 以降であることを確認します。で変更ログを確認できます GitHub。

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

> **Note**
>
> 実行中のコンテナのログを表示すると、次のテキストを含む行が表示されますが、無視しても問題ありません。
>
> ```
> Neither -kubeconfig nor -master was specified. Using the inClusterConfig.
>  This might not work.
> ```

ステップ2：App Mesh リソースをデプロイするには

Kubernetes でアプリケーションをデプロイするときは、Kubernetes カスタムリソースも作成し、コントローラーが対応する App Mesh リソースを作成できるようにします 次の手順は、App Mesh リソースの一部の機能をデプロイするのに役立ちます。他の App Mesh リソース機能をデプロイするためのマニフェストの例は、 の App Mesh チュートリアルに記載されている多くの機能フォルダのv1beta2サブフォルダにあります GitHub。

> **Important**
>
> コントローラーによって App Mesh リソースが作成されたら、App Mesh リソースの変更または削除は、コントローラーのみを使用して行うことをお勧めします。App Mesh を使用してリソースの変更または削除をする場合、デフォルトでは、コントローラーは、変更または

削除された AppMesh リソースを10時間の間、変更または再作成しません。この期間を短く設定できます。詳細については、「 の設定」を参照してください GitHub。

App Mesh リソースをデプロイするには

1. App Mesh リソースをデプロイする Kubernetes 名前空間を作成します。

   a. 次の内容をコンピュータ上の namespace.yaml という名前のファイルに保存します。

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

   b. 名前空間を作成します。

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

2. App Mesh サービスメッシュを作成します。

   a. 次の内容をコンピュータ上の mesh.yaml という名前のファイルに保存します。このファイルは、*my-mesh* という名前のメッシュリソースを作成するために使用されます。サービスメッシュは、サービス間のネットワークトラフィックの論理的な境界であり、サービスはその中に存在します。

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

   b. メッシュを作成します。

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

c. 作成された Kubernetes メッシュリソースの詳細を表示します。

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

出力

```
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-j11k1l111m711
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. コントローラーによって作成された App Mesh サービスメッシュの詳細を表示します。

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

出力

```
{
    "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-j11k1l111m711",
            "version": 1
        },
        "spec": {},
        "status": {
            "status": "ACTIVE"
        }
    }
}
```

3. App Mesh 仮想ノードを作成します。仮想ノードは、Kubernetes デプロイメントへの論理ポインタとして機能します。

   a. 次の内容をコンピュータ上の `virtual-node.yaml` という名前のファイルに保存します。このファイルは、*my-service-a* という名前の付いた App Mesh 仮想ノードを、*my-apps* 名前空間に作成するために使用されます。仮想ノードは、後のステップで作成される Kubernetes サービスを表します。`hostname` の値は、この仮想ノードが表す実際のサービスの完全修飾 DNS ホスト名です。

```
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
```

仮想ノードには、 end-to-end 暗号化やヘルスチェックなどの機能があり、このチュートリアルでは説明しません。詳細については、「the section called “仮想ノード”」を参照してください。前述の仕様で設定できる仮想ノードで使用可能なすべての設定を表示するには、次のコマンドを実行します。

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

b. 仮想ノードをデプロイします。

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

c. 作成された Kubernetes 仮想ノードリソースの詳細を表示します。

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

出力

```
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-j11k1l111m711
```

```
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-j11k1l111m711
  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. App Mesh でコントローラーによって作成された仮想ノードの詳細を表示します。

> **Note**
>
> Kubernetes で作成される仮想ノードの名前は *my-service-a* ですが、App Mesh で作成される仮想ノードの名前は *my-service-a_my-apps* です。コントローラーは、App Mesh リソースの作成時に、Kubernetes 名前空間名を App Mesh 仮想ノード名に追加します。Kubernetes では、異なる名前空間に同じ名前の仮想ノードを作成できるため、名前空間名が追加されますがますが、App Mesh では仮想ノード名がメッシュ内で一意である必要があります。

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

出力

```
{
    "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-j11k1l111m711",
            "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 仮想ルーターを作成します。仮想ルーターは、メッシュ内の 1 つ以上の仮想サービスのトラフィックを処理します。

   a. 次の内容をコンピュータ上の `virtual-router.yaml` という名前のファイルに保存します。このファイルは、前のステップで作成された `my-service-a` という名前の仮想ノードにトラフィックをルーティングする仮想ルーターを作成するために使用されます。コント

ローラーは App Mesh 仮想ルーターを作成し、リソースをルーティングします。ルートにさらに多くの機能を指定し、http 以外のプロトコルを使用することができます。詳細については、「the section called “仮想ルーター”」および「the section called “ルート”」を参照してください。参照される仮想ノード名は、 Kubernetes 仮想ノード名であり、コントローラーによって App Mesh で作成された App Mesh 仮想ノード名ではないことに注意してください。

```
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
```

(オプション) 前述の仕様で設定できる仮想ルーターに使用できるすべての設定を表示するには、次のコマンドを実行します。

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

前述の仕様で設定できるルートに使用できるすべての設定を表示するには、次のコマンドを実行します。

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

b. 仮想ルーターをデプロイします。

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

c. 作成された Kubernetes 仮想ルーターリソースを表示します。

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

省略された出力

```
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-j11k1l111m711
  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. App Mesh でコントローラーによって作成された仮想ルーターリソースを表示します。コントローラーが App Mesh で仮想ルーターを作成したときに、仮想ルーターの名前に Kubernetes 名前空間名が追加されたため、name の `my-service-a-virtual-router_my-apps` を指定します。

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

出力

```
{
    "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-j11k1l111m711",
            "version": 1
        },
        "spec": {
            "listeners": [
                {
                    "portMapping": {
                        "port": 80,
                        "protocol": "http"
                    }
                }
            ]
        },
        "status": {
```

```
            "status": "ACTIVE"
        },
        "virtualRouterName": "my-service-a-virtual-router_my-apps"
    }
}
```

e. App Mesh でコントローラーによって作成されたルートリソースを表示します。その ルートは Kubernetes の仮想ルーター設定の一部であるため、ルートリソースが Kubernetes で作成されませんでした。ルート情報は、サブステップ c の Kubernetes リソースの詳細に表示されました。ルート名が仮想ルーターに固有であるため、コントローラーは、App Mesh でルートを作成したときに、AppMesh ルート名に Kubernetes 名前空間名を追加しませんでした。

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

出力

```
{
    "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-j11k1l111m711",
            "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 仮想サービスを作成します。仮想サービスは、仮想ノードが仮想ルーターを使用して直接または間接的に提供する実際のサービスを抽象化したものです。依存サービスは、仮想サービスを名前で呼び出します。名前は AppMesh にとって重要ではありませんが、仮想サービスに、仮想サービスが表す実際のサービスの完全修飾ドメイン名を付けるようお勧めします。このように仮想サービスに名前を付けることで、別の名前を参照するようにアプリケーションコードを変更する必要がなくなります。リクエストは、仮想サービスのプロバイダーとして指定されている仮想ノードまたは仮想ルーターにルーティングされます。

   a. 次の内容をコンピュータ上の `virtual-service.yaml` という名前のファイルに保存します。このファイルは、仮想ルータープロバイダーを使用して、前のステップで作成された `my-service-a` という名前の仮想ノードにトラフィックをルーティングする仮想サービスを作成するために使用されます。`spec` の `awsName` に対する値 は、この仮想サービスが抽象化する実際の Kubernetes サービスの完全修飾ドメイン名 (FQDN) です。Kubernetes サービスは「the section called “ステップ 3: サービスを作成または更新する”」で作成されます。詳細については、「the section called “仮想サービス”」を参照してください。

```
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
```

前述の仕様で設定できる仮想サービスに使用できるすべての設定を表示するには、次のコマンドを実行します。

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

b. 仮想サービスを作成します。

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

c. 作成された Kubernetes 仮想サービスリソースの詳細を表示します。

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

出力

```
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-j11k1l111m711
Spec:
  Aws Name:  my-service-a.my-apps.svc.cluster.local
  Mesh Ref:
    Name:  my-mesh
    UID:   111a11b1-c11d-1e1f-gh1i-j11k1l111m711
  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. App Mesh でコントローラーによって作成された仮想サービスリソースの詳細を表示します。仮想サービスの名前が一意の FQDN であるため、Kubernetes コントローラーは、App Mesh で仮想サービスを作成したときに、App Mesh 仮想サービス名に Kubernetes 名前空間名を追加しませんでした。

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

出力

```
{
    "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-j11k1l111m711",
            "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"
    }
}
```

このチュートリアルでは説明していませんが、コントローラーは App Mesh the section called “仮想ゲートウェイ” と the section called “ゲートウェイルート”をデプロイすることもできます。これらのリソースをコントローラーでデプロイするチュートリアルについては、「インバウンドゲートウェイの設定」、または のリソースを含むサンプルマニフェストを参照してください GitHub。

ステップ 3: サービスを作成または更新する

App Mesh で使用するポッドには、App Mesh サイドカーコンテナを追加する必要があります。インジェクターは、指定したラベルでデプロイされたポッドに、自動的にサイドカーコンテナを追加します。

1. プロキシ認証を有効にします。各 Kubernetes デプロイメントを有効にして、独自の App Mesh 仮想ノードの設定のみをストリーミングするようお勧めします。

   a. 次の内容をコンピュータ上の proxy-auth.json という名前のファイルに保存します。*alternate-colored values* は、独自の値に置き換えてください。

```
{
    "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. ポリシーを作成します。

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

c. IAM ロールを作成して、前のステップで作成したポリシーをそれにアタッチし、Kubernetes サービスアカウントを作成した後、ポリシーを Kubernetes サービスアカウントにバインドします。このロールにより、コントローラーは App Mesh リソースの追加、削除、変更を行うことができます。

```
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 または を使用してサービスアカウントを作成する場合は AWS CLI、「Amazon EKS ユーザーガイド」の「サービスアカウントの IAM ロールとポリシーの作成」を参照してください。 AWS Management Console または を使用してアカウント AWS CLI を作成する場合は、ロールを Kubernetes サービスアカウントにマッピングする必要もあります。詳細については、「Amazon EKS ユーザーガイド」の「サービスアカウントの IAM ロールを指定する」を参照してください。

2. (オプション) デプロイを Fargate ポッドにデプロイする場合は、Fargate プロファイルを作成する必要があります。`eksctl` をインストールしていない場合、「Amazon EKS ユーザーガイド」の「`eksctl` のインストールまたはアップグレード」の手順に従ってインストールできます。コンソールを使用してプロファイルを作成する場合は、「Amazon EKS ユーザーガイド」の「Fargate プロファイルの作成」を参照してください。

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

3. Kubernetes サービスとデプロイメントを作成します。App Mesh で使用する既存のデプロイがある場合は、「the section called “ステップ2：App Mesh リソースをデプロイするには”」のサブステップ 3 で行ったように、仮想ノードをデプロイする必要があります。デプロイを更新して、そのラベルが仮想ノードに設定したラベルに一致しているか確認し、サイドカーコンテナが自動的にポッドに追加され、ポッドが再デプロイされるようにします。

a. 次の内容をコンピュータ上の example-service.yaml という名前のファイルに保存します。名前空間名を変更して Fargate ポッドを使用している場合は、名前空間名が Fargate プロファイルで定義した名前空間名と一致していることを確認してください。

```
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**
>
> 仕様の app、matchLabels、selector の値は、3 のサブステップ the section called “ステップ2：App Mesh リソースをデプロイするには” で仮想ノードを作成したときに指定した値と一致する必要があります。一致しないと、サイドカーコンテナがポッドに挿入されません。前の例では、ラベルの値は my-app-1 です。仮想ノードではなく仮想ゲートウェイをデプロイする場合は、Deployment マニフェストには、Envoy コンテナのみを含める必要があります。使用する画像の詳細については、「Envoy」を参照してください。サンプルマンフェストについては、「」のデプロイ例を参照してください GitHub。

b. サービスをデプロイします。

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

c. サービスとデプロイメントを表示します。

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

出力

```
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. デプロイ済みポッドの 1 つの詳細を表示します。

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

省略された出力

```
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
```

上記の出力では、コントローラーによって envoy および proxyinit コンテナがポッドに追加されたことがわかります。サンプルサービスを Fargate にデプロイした場合は、envoy コンテナはコントローラーによってポッドに追加されましたが、proxyinit コンテナはそうではありませんでした。

4. (オプション) Prometheus、Grafana、Jaeger AWS X-Ray、Datadog などのアドオンをインストールします。詳細については、「 の App Mesh アドオン」および「App Mesh ユーザーガイド」の「オブザーバビリティ」セクションを参照してください。 GitHub

> **Note**
>
> App Mesh のその他の例とチュートリアルについては、App Mesh サンプルリポジトリを参照してください。

ステップ 4: クリーンアップする

このチュートリアルで作成したサンプルリソースをすべて削除します。コントローラーは、my-mesh App Mesh サービスメッシュで作成されたリソースも削除します。

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

サンプルサービスの Fargate プロファイルを作成した場合は、それを削除します。

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

メッシュを削除します。

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

(オプション) Kubernetes 統合コンポーネントを削除できます。

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

(オプション) Kubernetes 統合コンポーネントを Fargate にデプロイした場合は、Fargate プロファイルを削除します。

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

AWS App Mesh と Amazon EC2 の開始方法

このトピックでは、Amazon EC2 AWS App Mesh で実行されている実際のサービスで を使用する方法について説明します。このチュートリアルでは、複数の App Mesh リソースタイプのベーシックな機能について説明します。

シナリオ

App Mesh の使用方法を説明するために、次の特性を持つアプリケーションがあると仮定します。

- serviceA および serviceB という名前の 2 つのサービスで構成されています。
- どちらのサービスも、apps.local という名前の名前空間にメンバー登録されます。
- ServiceA は、HTTP/2、ポート 80 を介して serviceB と通信します。
- すでに serviceB のバージョン 2 をデプロイし、serviceBv2 名前空間に apps.local という名前でメンバー登録しました。

次の要件があります。

- serviceA から serviceB にトラフィックの 75% を送信し、serviceBv2 にトラフィックの 25% を送信して、serviceA からトラフィックの100%を送信する前に、serviceBv2 にバグがないことを検証します。
- トラフィックの重み付けを簡単に調整して、信頼性が証明されたら、トラフィックの 100% が serviceBv2 へ転送されるようにします。すべてのトラフィックが serviceBv2 に送信されたら、serviceB を切断します。
- 上記の要件を満たすために、実際のサービスの既存のアプリケーションコードまたはサービスディスカバリ登録を変更する必要はありません。

要件を満たすために、仮想サービス、仮想ノード、仮想ルーター、およびルートで、App Mesh サービスメッシュを作成することにします。メッシュを実装した後、サービスを更新して、Envoy プロキシを使用します。更新されると、サービスは相互に直接ではなく、Envoy プロキシを介して相互に通信します。

前提条件

App Mesh は、DNS、 AWS Cloud Mapまたはその両方に登録されている Linux サービスをサポートします。この「開始方法」を使用するには、DNS に既存のサービスを 3 つメンバー登録しておくようお勧めします。サービスが存在しない場合でもサービスメッシュとそのリソースを作成できますが、実際のサービスをデプロイするまでメッシュを使用することはできません。

サービスをまだ実行していない場合は、Amazon EC2 インスタンスの起動をして、それらにアプリケーションをデプロイできます。詳細については、Amazon EC2 ユーザーガイド」の「チュートリアル: Amazon EC2 Linux インスタンスの開始方法」を参照してください。 Amazon EC2 残りの

ステップでは、実際のサービスが serviceA、serviceB、serviceBv2 という名前で、すべてのサービスが apps.local という名前の名前空間を介して検出可能であることを前提としています。

ステップ 1: メッシュと仮想サービスを作成する

サービスメッシュは、サービス間のネットワークトラフィックの論理的な境界であり、サービスはその中に存在します。詳細については、「サービスメッシュ」を参照してください。仮想サービスは、実際のサービスを抽象化したものです。詳細については、「仮想サービス」を参照してください。

次の リソースを作成します。

- シナリオ内のすべてのサービスが apps 名前空間にメンバー登録されているため、apps.local という名前のメッシュ。
- serviceb.apps.local という名前の仮想サービス。仮想サービスは、その名前で検出可能なサービスを表しているため、別の名前をリファレンスするようにコードを変更したくないためです。servicea.apps.local という名前の仮想サービスが、次のステップで追加されます。

AWS Management Console または AWS CLI バージョン 1.18.116 以降または 2.0.38 以降を使用して、次のステップを完了できます。を使用している場合は AWS CLI、 aws --version コマンドを使用してインストールされている AWS CLI バージョンを確認します。バージョン 1.18.116 以降、または 2.0.38 以降をインストールしていない場合は、AWS CLIをインストールまたは更新する必要があります。使用するツールのタブを選択します。

AWS Management Console

1. App Mesh コンソールの初回実行ウィザードを https://console.aws.amazon.com/appmesh/get-started で開きます。
2. [メッシュ名] に **apps** と入力します。
3. [仮想サービス名] に **serviceb.apps.local** と入力します。
4. 続行するには、[次へ] を選択します。

AWS CLI

1. create-mesh コマンドを使用してメッシュを作成します。

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

2. create-virtual-service コマンドを使用して仮想サービスを作成します。

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

ステップ 2: 仮想ノードを作成する

仮想ノードは、実際のサービスの論理ポインタとして機能します。詳細については、「仮想ノード」を参照してください。

仮想ノードの 1 つが serviceB という名前の実際のサービスを表すため、serviceB という名前の仮想ノードを作成します。仮想ノードが表す実際のサービスは、serviceb.apps.local というホスト名を持つ DNS を介して検出可能です。または、 AWS Cloud Mapを使用して実際のサービスを検出することもできます。仮想ノードは、ポート 80 で HTTP/2 プロトコルを使用してトラフィックをリッスンします。ヘルスチェックと同様に、その他のプロトコルもサポートされています。次のステップで、serviceA および serviceBv2 の仮想ノードを作成します。

AWS Management Console

1. [仮想ノード名] に **serviceB** と入力します。
2. [サービスディスカバリ] で、[DNS] を選択し、[DNS ホスト名] に **serviceb.apps.local** と入力します。
3. [リスナーの設定] で、[プロトコル] に [http2] を選択し、[ポート]に **80** と入力します。
4. 続行するには、[次へ] を選択します。

AWS CLI

1. 次の内容で、create-virtual-node-serviceb.json という名前のファイルを作成します。

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

```
            }
        ],
        "serviceDiscovery": {
            "dns": {
                "hostname": "serviceB.apps.local"
            }
        }
    },
    "virtualNodeName": "serviceB"
}
```

2. JSON ファイルを入力として使用して、create-virtual-node コマンドで仮想ノードを作成します。

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

ステップ 3: 仮想ルーターとルートを作成する

仮想ルーターは、メッシュ内の 1 つ以上の仮想サービスのトラフィックを送信します。詳細については、「仮想ルーター」および「ルート」を参照してください。

次の リソースを作成します。

- serviceB という名前の仮想ルーター。serviceB.apps.local 仮想サービスは、他のサービスとのアウトバウンド通信を開始しないためです。前に作成した仮想サービスは、実際の serviceb.apps.local サービスの抽象化であることに注意してください。仮想サービスは、仮想ルーターにトラフィックを送信します。仮想ルーターは、ポート 80 で HTTP/2 プロトコルを使用してトラフィックをリッスンします。その他のプロトコルもサポートされています。
- serviceB という名前のルート。このルートはトラフィックの 100% を serviceB 仮想ノードにルーティングします。重み付けは、serviceBv2 仮想ノードを追加した後のステップで行います。このガイドでは説明しませんが、ルートにフィルタ条件を追加したり、通信の問題が発生したときに Envoy プロキシが仮想ノードへのトラフィックの送信を複数回試行する再試行ポリシーを追加したりできます。

AWS Management Console

1. [仮想ルーター名] に **serviceB** と入力します。

2. [リスナーの設定] で、[プロトコル] に [http2] を選択して、[ポート] に **80** を指定します。
3. [ルート名] に **serviceB** と入力します。
4. [ルートタイプ] で、[http2] を選択します。
5. [ターゲット設定] の [仮想ノード名]で、[serviceB] を選択し、[重み]に **100** と入力します。
6. [一致設定] で、[方法] を選択します。
7. 続行するには、[次へ] を選択します。

AWS CLI

1. 仮想ルーターを作成します。

   a. 次の内容で、create-virtual-router.json という名前のファイルを作成します。

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

   b. JSON ファイルを入力として使用し、create-virtual-router コマンドで仮想ルーターを作成します。

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

2. ルートを作成します。

   a. 次の内容で、create-route.json という名前のファイルを作成します。

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

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

b. JSON ファイルを入力として使用し、create-route コマンドでルートを作成します。

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

ステップ 4: 確認して作成する

前のステップと照らし合わせて設定を確認します。

AWS Management Console

いずれかのセクションに変更を加える必要がある場合は、[編集] を選択します。設定が完了したら、[メッシュの作成] を選択します。

[ステータス] 画面には、作成されたすべてのメッシュリソースが表示されます。作成したリソースをコンソールに表示するには、[メッシュの表示] を選択します。

AWS CLI

describe-mesh コマンドで作成したメッシュの設定を確認します。

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

describe-virtual-service コマンドで作成した仮想サービスの設定を確認します。

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

describe-virtual-node コマンドで作成した仮想ノードの設定を確認します。

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

describe-virtual-router コマンドで作成した仮想ルーターの設定を確認します。

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

describe-route コマンドで作成したルートの設定を確認します。

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

ステップ 5: 追加のリソースを作成する

このシナリオを完了するには、次のことを行う必要があります。

- serviceBv2 という名前の仮想ノードと、serviceA という名前の別の仮想ノードを作成します。両方の仮想ノードは、HTTP/2 ポート 80 経由でリクエストをリッスンします。serviceA 仮想ノードには、serviceb.apps.local のバックエンドを設定します。serviceA 仮想ノードからのすべてのアウトバウンドトラフィックは、serviceb.apps.local という名前の仮想サービスに送信されます。このガイドでは説明しませんが、仮想ノードのアクセスログを書き込むファイルパスを指定することもできます。
- servicea.apps.local という名前の追加の仮想サービスを 1 つ作成します。これにより、すべてのトラフィックが serviceA 仮想ノードに直接送信されます。
- 前のステップで作成した serviceB ルートを更新して、トラフィックの 75% を serviceB 仮想ノードに送信し、25% を serviceBv2 仮想ノードに送信します。時間の経過とともに、serviceBv2 が 100% のトラフィックを受信するまで、継続して重みを変更することができます。すべてのトラフィックが serviceBv2 に送信されたら、serviceB 仮想ノードと実際のサービスをシャットダウンして中止することができます。重みを変更しても、serviceb.apps.local 仮想サービス名および実際のサービス名は変更されないため、コードを変更する必要はありません。serviceb.apps.local 仮想サービスは仮想ルーターにトラフィックを送信し、仮想ルーターはトラフィックを仮想ノードにルーティングすることに注意してください。仮想ノードのサービスディスカバリ名は、いつでも変更できます。

AWS Management Console

1. 左のナビゲーションペインで [メッシュ] を選択します。
2. 前のステップで作成した apps メッシュを選択します。
3. 左側のナビゲーションペインで、[仮想ノード] を選択します。
4. [仮想ノードの作成] を選択します。
5. [仮想ノード名] に **serviceBv2** と入力し、[サービスディスカバリ] で [DNS] を選択して、[DNS ホスト名] に **servicebv2.apps.local** と入力します。
6. [リスナーの設定] で、[プロトコル] に [http2] を選択し、[ポート] に **80** を入力します。
7. [仮想ノードの作成] を選択します。
8. [仮想ノードの作成] をもう一度選択します。[仮想ノード名] に **serviceA** と入力してください。[サービスディスカバリ] で [DNS] を選択し、[DNS ホスト名] に **servicea.apps.local** と入力します。
9. [新しいバックエンド] の下の [仮想サービス名の入力] に **serviceb.apps.local** と入力します。
10. [リスナーの設定] で、[プロトコル] に [http2] を選択し、[ポート] に **80** を入力して、[仮想ノードの作成] を選択します。
11. 左側のナビゲーションペインで [仮想ルーター] を選択し、リストから [serviceB] 仮想ルーターを選択します。
12. [ルート] で、前のステップで作成した ServiceB という名前のルートを選択し、[編集] を選択します。
13. [ターゲット] の仮想ノード名で、serviceB の [重み] の値を **75** に変更します。
14. [ターゲットの追加] を選択し、ドロップダウンリストから serviceBv2を選択して、[重み] の値を **25** に設定します。
15. [保存] を選択します。
16. 左側のナビゲーションペインで、[仮想サービス] を選択し、[仮想サービスの作成] を選択します。
17. [仮想サービス名] に **servicea.apps.local** と入力し、[プロバイダー] に [仮想ノード] を選択し、[仮想ノード] に serviceA を選択し、[仮想サービスの作成]を選択します。

AWS CLI

1. serviceBv2 仮想ノードを作成します。

a. 次の内容で、create-virtual-node-servicebv2.json という名前のファイルを作成します。

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

b. 仮想ノードを作成します。

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

2. serviceA 仮想ノードを作成します。

a. 次の内容で、create-virtual-node-servicea.json という名前のファイルを作成します。

```
{
   "meshName" : "apps",
   "spec" : {
      "backends" : [
         {
            "virtualService" : {
               "virtualServiceName" : "serviceb.apps.local"
            }
         }
```

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

b. 仮想ノードを作成します。

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

3. 前のステップで作成した serviceb.apps.local 仮想サービスを更新して、そのトラフィックを serviceB 仮想ルーターに送信します。仮想サービスが最初に作成された時点では、serviceB 仮想ルーターがまだ作成されていないため、トラフィックはどこにも送信されませんでした。

   a. 次の内容で、update-virtual-service.json という名前のファイルを作成します。

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

   b. update-virtual-service コマンドを使用して、仮想サービスを更新します。

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

4. 前のステップで作成した serviceB ルートを更新します。

   a. 次の内容で、update-route.json という名前のファイルを作成します。

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

   b. update-route コマンドを使用してルートを更新します。

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

5. serviceA 仮想サービスを作成します。

   a. 次の内容で、create-virtual-servicea.json という名前のファイルを作成します。

```
{
```

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

b. 仮想サービスを作成します。

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

メッシュの概要

サービスメッシュを作成する前に、servicea.apps.local、serviceb.apps.local、および servicebv2.apps.local という 3 つの実際のサービスがありました。実際のサービスに加えて、実際のサービスを表す次のリソースを含むサービスメッシュが作成されました。

- 2 つの仮想サービス。プロキシは、仮想ルーターを経由して、servicea.apps.local 仮想サービスからのすべてのトラフィックを serviceb.apps.local 仮想サービスに送信します。
- serviceA、serviceB、および serviceBv2 という名前の 3 つの仮想ノード。Envoy プロキシは、仮想ノードに対して設定されたサービスディスカバリ情報を使用して、実際のサービスの IP アドレスを検索します。
- Envoy プロキシがインバウンドトラフィックの 75% を serviceB 仮想ノードに、25% を serviceBv2 仮想ノードにルーティングするように指定する 1 つのルートを持つ仮想ルーター。

ステップ 6: サービスを更新する

メッシュを作成したら、次のタスクを完了する必要があります。

- 各サービスでデプロイする Envoy プロキシに、1つ以上の仮想ノードの設定を読み取りすることを許可します。プロキシを承認する方法の詳細については、Envoy プロキシの認可 を参照してください。
- 既存のサービスを更新するには、次のステップを実行します。

Amazon EC2 インスタンスを仮想ノードメンバーとして設定するには

1. IAM ロールを作成します。

   a. 次の内容で、ec2-trust-relationship.json という名前のファイルを作成します。

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

   b. 次のコマンドを使用して IAM ロールを作成します。

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

2. IAM ポリシーを、Amazon ECR からの読み取りと特定のApp Mesh 仮想ノードの設定のみを許可するロールに添付します。

   a. 次の内容の virtual-node-policy.json という名前のファイルを作成します。apps は the section called “ステップ 1: メッシュと仮想サービスを作成する” で作成したメッシュの名前で、serviceB は the section called “ステップ 2: 仮想ノードを作成する” で作成した仮想ノードの名前です。*111122223333* をアカウント ID に置き換え、*us-west-2* をメッシュを作成したリージョンに置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appmesh:StreamAggregatedResources",
            "Resource": [
                "arn:aws:appmesh:us-west-2:111122223333:mesh/apps/
virtualNode/serviceB"
```

```
            ]
        }
    ]
}
```

b. 次のコマンドを使用してポリシーを作成します。

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

c. 前のステップで作成したポリシーをロールに添付して、ロールが AppMesh からの `serviceB` 仮想ノードの設定のみを読み取れるようにします。

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

d. `AmazonEC2ContainerRegistryReadOnly` マネージドポリシーをロールに添付して、Amazon ECR から Envoy コンテナイメージをプルできるようにします。

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

3. 作成した IAM ロールを使用して Amazon EC2 インスタンスの起動を行います。
4. SSH 経由でインスタンスに接続します。
5. オペレーティングシステムのドキュメントに従って、Docker と をインスタンス AWS CLI にインストールします。
6. Docker クライアントがイメージをプルするリージョンの Envoy Amazon ECR リポジトリを認証します。

   - `me-south-1`、`ap-east-1`、`ap-southeast-3`、`eu-south-1`、`il-central-1`、`af-south-1` を除くすべてのリージョン。*`us-west-2`* は、`me-south-1`、`ap-east-1`、`ap-southeast-3`、`eu-south-1`、`il-central-1`、`af-south-1` を除くサポートされているリージョンに置き換えることができます。

```
$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 リージョン

```
$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 リージョン

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

7. 次のいずれかのコマンドを実行して、イメージをプルするリージョンに応じて、インスタンスで App Mesh Envoy コンテナを起動します。*apps* と *serviceB* の値は、シナリオで定義されたメッシュ名と仮想ノード名です。この情報は、App Mesh から読み取りする仮想ノード設定をプロキシに指示します。このシナリオを完了する際には、serviceBv2 および serviceA 仮想ノードで表されるサービスをホストする Amazon EC2 インスタンスについても、これらのステップを完了する必要があります。独自のアプリケーションでは、これらの値を独自の値に置き換えます。

   - me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1、af-south-1 を除くすべてのリージョン。*Region-code* は、me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1、af-south-1 リージョンを除く、サポートされている任意のリージョンに置き換えることができます。*1337* は、0 と 2147483647 の間の任意の値に置き換えることができます。

```
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:v1.29.5.0-prod
```

   - me-south-1 リージョン *1337* は、0 と 2147483647 の間の任意の値に置き換えることができます。

```
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:v1.29.5.0-prod
```

- ap-east-1 リージョン *1337* は、0 と 2147483647 の間の任意の値に置き換えることができます。

```
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:v1.29.5.0-prod
```

> **Note**
>
> APPMESH_RESOURCE_ARN プロパティは、バージョン 1.15.0 またはそれ以降のEnvoy イメージを必要とします 詳細については、「Envoy」を参照してください。

> **Important**
>
> App Mesh での使用は、バージョン v1.9.0.0-prod 以降のみでサポートされています。

8. 次の Show more を選択します。インスタンスで、次の内容の envoy-networking.sh という名前のファイルを作成します。*8000* を、アプリケーションコードが着信トラフィックに使用するポートに置き換えます。APPMESH_IGNORE_UID の値は変更できますが、値は前のステップで指定した値と同じである必要があります (例: 1337)。必要に応じて、アドレスを APPMESH_EGRESS_IGNORED_IP に追加できます。他の行は変更しないでください。

```
#!/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 processess 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

      [ ! -z "$APPMESH_IGNORE_GID" ] && \
          ip6tables -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
         ip6tables -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_ENVOY_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. アプリケーショントラフィックを Envoy プロキシにルーティングする iptables ルールを設定するには、前のステップで作成したスクリプトを実行します。

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

10. 仮想ノードのアプリケーションコードを開始します。

> **Note**
>
> App Mesh のその他の例とチュートリアルについては、App Mesh サンプルリポジトリを参照してください。

App Mesh ロードマップ

これは、AWS App Mesh の実験的な公開ロードマップです。ロードマップにより、お客様は、当社の今後の製品と優先順位について知ることができ、将来の App Mesh の使用方法を計画するのに役立てることができます。このリポジトリには、当社が取り組んでいることに関する情報が含まれており、すべての AWS のお客様が直接フィードバックを提することができます。

App Mesh ロードマップ

App Mesh の例

次のリポジトリでは、動作中の AWS App Mesh を示すエンドツーエンドのチュートリアルや、様々な AWS サービスと連携するためのコード例を見ることができます。

App Mesh の例

App Mesh の概念

App Mesh は、次の概念で構成されています。

- サービスメッシュ
- 仮想サービス
- 仮想ゲートウェイ
- 仮想ノード
- 仮想ルーター

サービスメッシュ

サービスメッシュは、サービス間のネットワークトラフィックの論理的な境界であり、サービスはその中に存在します。サービスメッシュを作成したら、仮想サービス、仮想ノード、仮想ルーター、およびルートを作成して、メッシュ内のアプリケーション間でトラフィックを分散させることができます。

サービスメッシュの作成

> **Note**
>
> メッシュを作成するときは、ネームスペースセレクタを追加する必要があります。名前空間セレクタが空の場合、すべての名前空間が選択されます。名前空間を制限するには、ラベルを使用して App Mesh リソースを作成したメッシュに関連付けます。

AWS Management Console

AWS Management Console を使用してサービスメッシュを作成するには

1. 次の App Mesh コンソールを開きます：https://console.aws.amazon.com/appmesh/。
2. [メッシュを作成] を選択します。
3. [メッシュ名] に、サービスメッシュの名前を指定します。
4. (オプション) [外部トラフィックを許可する] を選択します。デフォルトでは、メッシュ内のプロキシは相互間のトラフィックのみを転送します。外部トラフィックを許可する場合、

メッシュ内のプロキシは、メッシュで定義されているプロキシでデプロイされていないサービスに TCP トラフィックを直接転送します。

> **Note**
>
> ALLOW_ALL を使用する際、仮想化ノードでバックエンドを指定する場合は、その仮想化ノードのすべてのエグレスをバックエンドとして指定する必要があります。それ以外の場合は、その仮想化ノードでは ALLOW_ALL が機能しなくなります。

5. [IP バージョンの設定]

[デフォルトの IP バージョンの動作を上書きする] をオンに切り替えて、メッシュ内のトラフィックに使用する IP バージョンを制御します。デフォルトでは、App Mesh はさまざまな IP バージョンを使用します。

> **Note**
>
> メッシュは、IP 設定をメッシュ内のすべての仮想ノードと仮想ゲートウェイに適用します。この動作は、ノードを作成または編集するときに IP 設定の項目を設定することで、個々の仮想ノードで上書きできます。IPv4 と IPv6 の両方のトラフィックをリッスンできる仮想ゲートウェイの設定は、メッシュにどちらの IP 設定が設定されていても同じであるため、仮想ゲートウェイでは IP 設定を上書きすることはできません。

- デフォルト
  - Envoy の DNS リゾルバーは IPv6 を優先し、IPv4 にフォールバックします。
  - 利用可能な場合は AWS Cloud Map から返された IPv4 アドレスを使用し、フォールバックする場合は IPv6 アドレスを使用します。
  - ローカルアプリ用に作成されたエンドポイントは IPv4 アドレスを使用します。
  - Envoy リスナーはすべての IPv4 アドレスにバインドされます。
- IPv6 優先
  - Envoy の DNS リゾルバーは IPv6 を優先し、IPv4 にフォールバックします。
  - 利用可能な場合は AWS Cloud Map から返された IPv6 アドレスを使用し、フォールバックする場合は IPv4 アドレスを使用します。
  - ローカルアプリ用に作成されたエンドポイントは IPv6 アドレスを使用します。

        - Envoy リスナーはすべての IPv4 および IPv6 アドレスにバインドされます。
    - IPv4 優先
        - Envoy の DNS リゾルバーは IPv4 を優先し、IPv6 にフォールバックします。
        - 利用可能な場合は AWS Cloud Map から返された IPv4 アドレスを使用し、フォールバックする場合は IPv6 アドレスを使用します。
        - ローカルアプリ用に作成されたエンドポイントは IPv4 アドレスを使用します。
        - Envoy リスナーはすべての IPv4 および IPv6 アドレスにバインドされます。
    - IPv6 のみ
        - Envoy の DNS リゾルバーは IPv6 のみを使用します。
        - AWS Cloud Map によって返された IPv6 アドレスのみが使用されます。AWS Cloud Map が IPv4 アドレスを返す場合、IP アドレスは使用されず、空の結果が Envoy に返されます。
        - ローカルアプリ用に作成されたエンドポイントは IPv6 アドレスを使用します。
        - Envoy リスナーはすべての IPv4 および IPv6 アドレスにバインドされます。
    - IPv4 のみ
        - Envoy の DNS リゾルバーは IPv4 のみを使用します。
        - AWS Cloud Map によって返された IPv4 アドレスのみが使用されます。AWS Cloud Map が IPv6 アドレスを返す場合、IP アドレスは使用されず、空の結果が Envoy に返されます。
        - ローカルアプリ用に作成されたエンドポイントは IPv4 アドレスを使用します。
        - Envoy リスナーはすべての IPv4 および IPv6 アドレスにバインドされます。
6. [メッシュを作成] を選択して終了します。
7. (オプション) メッシュを他のアカウントと共有します。共有メッシュを使用すると、異なるアカウントで作成されたリソースが同じメッシュ内で相互に通信できるようになります。詳細については、「共有メッシュの使用」を参照してください。

AWS CLI

AWS CLI を使用してメッシュを作成するには

以下のコマンド (##の値を独自の値に置き換えてください) を使用して、サービスメッシュを作成します。

1. ```
   aws appmesh create-mesh --mesh-name meshName
   ```

2. 出力例:

   ```
   {
       "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 Mesh の AWS CLI を使用してメッシュを作成する方法の詳細については、AWS CLI リファレンスの create-mesh コマンドを参照してください。

メッシュの削除

AWS Management Console

AWS Management Console CLI を使用してト仮想ートウェイを削除するには

1. App Mesh コンソールを https://console.aws.amazon.com/appmesh/ で開きます。
2. 削除するメッシュを選択します。所有し、共有されているすべてのメッシュが一覧表示されます。
3. 確認ボックスで、「**delete**」と入力し、[削除] をクリックします。
   ```

AWS CLI

AWS CLI を使用してメッシュを削除するには

1. 以下のコマンドを使用してメッシュを削除します (*##*の値を独自の値に置き換えてください)。

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

2. 出力例:

```
{
    "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 Mesh の AWS CLI を使用してメッシュを削除する方法の詳細については、AWS CLI リファレンスの delete-mesh コマンドを参照してください。

仮想サービス

仮想サービスは、仮想ノードが仮想ルーターを使用して直接または間接的に提供する実際のサービスを抽象化したものです。依存サービスが `virtualServiceName` を使用して仮想サービスを呼び出し、それらのリクエストが仮想サービスのプロバイダーとして指定されている仮想ノードまたは仮想ルーターにルーティングされます。

仮想サービスを作成する

AWS Management Console

AWS Management Console を使用して仮想サービスを作成するには

1. https://console.aws.amazon.com/appmesh/ で App Mesh コンソールを開きます。
2. 仮想サービスを作成するメッシュを選択します。所有し、共有されているすべてのメッシュが一覧表示されます。
3. 左側のナビゲーションで [仮想サービス] を選択します。
4. [仮想サービスの作成] を選択します。
5. [仮想サービス名] で仮想サービスの名前を選択します。任意の名前を選択できますが、ターゲットとする実際のサービスのサービス検出名 (`my-service.default.svc.cluster.local` など) にして、仮想サービスを実際のサービスに簡単に関連付けることができるようにすることをお勧めします。このようにすると、現在参照されているコードとは異なる名前を参照するようにコードを変更する必要はありません。リクエストが Envoy プロキシに送信される前に、アプリケーションコンテナが名前を正常に解決できる必要があるため、指定する名前は非ループバックIPアドレスに解決される必要があります。アプリまたはプロキシコンテナは、この IP アドレスと通信しないため、非ループバックの任意の IP アドレスを使用できます。プロキシは、App Mesh で設定した名前で他の仮想サービスと通信し、名前が解決される IP アドレスを介しては通信しません。
6. [プロバイダー] で、仮想サービスのプロバイダータイプを選択します。
   - 仮想サービスでトラフィックを複数の仮想ノードに分散させる場合は、[仮想ルーター] を選択してから、使用する仮想ルーターをドロップダウンメニューから選択します。
   - 仮想ルーターを使用せずに仮想サービスを仮想ノードに直接到達させる場合は、[仮想ノード] を選択してから、使用する仮想ノードをドロップダウンメニューから選択します。

> **Note**
>
> App Mesh API を介してそのようなポリシーを定義できない場合でも、App Mesh は、2020 年 7 月 29 日以降に定義した仮想ノードプロバイダーごとにデフォルトの Envoy ルート再試行ポリシーを自動的に作成する場合があります。詳細については、「デフォルトのルート再試行ポリシー」を参照してください。

- この時点で仮想サービスにトラフィックをルーティングさせないようにする場合 (例えば、仮想ノードまたは仮想ルーターがまだ存在していない場合) は、[なし] を選択します。この仮想サービスのプロバイダーは後から更新できます。

7. [仮想サービスの作成] を選択して終了します。

AWS CLI

AWS CLI を使用して仮想サービスを作成するには

以下のコマンドと入力 JSON ファイル (##の値を独自の値に置き換えてください) を使用して、仮想サービスと仮想ノードプロバイダーを作成します。

1.
```
aws appmesh create-virtual-service \
--cli-input-json file://create-virtual-service-virtual-node.json
```

2. create-virtual-service-virtual-node.json の例の内容:

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

3. 出力例:

```
{
    "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 Mesh の AWS CLI を使用して仮想サービスを作成する方法の詳細については、AWS CLI リファレンスの create-virtual-service コマンドを参照してください。

仮想サービスを削除する

> **Note**
>
> ゲートウェイルートで参照されている仮想サービスは削除できません。最初にゲートウェイルートを削除する必要があります。

AWS Management Console

AWS Management Console を使用して仮想サービスを削除するには

1. https://console.aws.amazon.com/appmesh/ で App Mesh コンソールを開きます。
2. 仮想サービスを削除するメッシュを選択します。所有し、共有されているすべてのメッシュが一覧表示されます。
3. 左側のナビゲーションで [仮想サービス] を選択します。
4. 削除する仮想サービスを選択し、右上隅の [削除] をクリックします。アカウントがリソース所有者として一覧されている仮想ゲートウェイのみを削除できます。

5. 確認ボックスで、「**delete**」と入力し、[削除] をクリックします。

AWS CLI

AWS CLI を使用して仮想サービスを削除するには

1. 以下のコマンドを使用して仮想サービスを削除します (*##*の値を独自の値に置き換えてください)。

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

2. 出力例:

```
{
    "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 Mesh の AWS CLI を使用して仮想サービスを削除する方法の詳細については、AWS CLI リファレンスの delete-virtual-service コマンドを参照してください。

仮想ゲートウェイ

仮想ゲートウェイを使用すると、メッシュ外のリソースは、メッシュ内のリソースと通信できます。仮想ゲートウェイは、Amazon ECS サービス、Kubernetes サービス内で、または Amazon EC2 インスタンス上で、実行されている Envoy プロキシを表します。仮想ゲートウェイは、アプリケーションで実行されている Envoy を表す仮想ノードとは異なり、単独でデプロイされた Envoy を表します。

外部リソースは、Envoy を実行するサービスまたはインスタンスに割り当てられた IP アドレスに DNS 名を解決できる必要があります。Envoy は、メッシュ内にある App Mesh リソースのすべてのアプリケーションメッシュ設定にアクセスできます。仮想ゲートウェイでの着信リクエストを処理するための設定は、ゲートウェイルートを使用して指定します。

> **Important**
>
> HTTP または HTTP2 リスナーを持つ仮想ゲートウェイは、着信リクエストのホスト名をゲートウェイルートターゲット仮想サービスの名前に書き換えます。また、ゲートウェイルートからの一致したプレフィックスは、デフォルトで / に書き換えられます。例えば、ゲートウェイルート一致プレフィクスが `/chapter` 、そして、着信リクエストが `/chapter/1` とすると、リクエストは `/1` に書き換えられます 書き換えを設定するには、「ゲートウェイルート」の「ゲートウェイルートの作成」セクションを参照してください。仮想ゲートウェイを作成するときは、`proxyConfiguration` と `user` は設定しないでください。

エンドツーエンドのチュートリアルを完了するには、「インバウンドゲートウェイの設定」を参照してください。

仮想ゲートウェイの作成

> **Note**
>
> 仮想ゲートウェイを作成するときは、作成した仮想ゲートウェイにゲートウェイルートを関連付ける名前空間のリストを識別するラベル付きの名前空間セレクターを追加する必要があります。

AWS Management Console

AWS Management Console を使用して仮想ゲートウェイを作成するには

1. https://console.aws.amazon.com/appmesh/ で App Mesh コンソールを開きます。
2. 仮想ゲートウェイを作成するメッシュを選択します。自分が所有しているメッシュや、共有されているメッシュがすべて一覧表示されます。
3. 左側のナビゲーションで [仮想ルーター] を選択します。
4. [仮想ゲートウェイを作成] を選択します。
5. [仮想ゲートウェイ名] に、仮想ゲートウェイの名前を入力します。
6. (オプションですが、推奨) クライアントポリシーのデフォルトを設定します。
   a. (オプション) Transport Layer Security (TLS)を使用してバーチャルサービスとのみ通信を行う場合は、「TLSの適用」を選択します。
   b. (オプション) [ポート] で、仮想サービスとの TLS 通信を適用する 1 つ以上のポートを指定します。
   c. 検証方法を使用する場合、次のいずれかのオプションを選択します。指定する証明書は、すでに存在し、特定の要件を満たしている必要があります。詳細については、「証明書の要件」を参照してください。
      - AWS Private Certificate Authority ホスティング — 既存の 1 つまたは複数のを選択します。証明書。
      - [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使用して取得するシークレットの名前を入力します。
      - ローカルファイルホスティング — Envoy がデプロイされているファイルシステム上の証明書チェーンファイルへのパスを指定します。

d. (オプション) サブジェクトの別名を入力します。SAN を追加するには、[Add SAN] (SAN を追加) を選択します。SAN は FQDN または URI 形式である必要があります。

e. (オプション) サーバーが要求したときにクライアント証明書を提供し、相互TLS認証を有効にするには、[Provide client certificate] (クライアント証明書の提供) と、次のオプションのいずれかを選択します。相互 TLS の詳細については、App Mesh の「相互 TLS 認証」ドキュメントを参照してください。

- [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使用して取得するシークレットの名前を入力します。
- ローカルファイルホスティング - Envoy がデプロイされているファイルシステムで、証明書チェーンファイルとシークレットキーへのパスを指定します。ローカルファイルで暗号化を使用してサンプルアプリケーションでメッシュをデプロイする完全なエンドツーエンドのチュートリアルについては、GitHub の「ファイル提供の TLS 証明書を使用した TLS の設定」を参照してください。

7. (オプション) ログを設定するには、ログ記録を選択します。Envoy で使用する HTTP アクセスログパスを入力します。`/dev/stdout` パスを使用するようお勧めします。これにより、Docker ログドライバーを使用して Envoy ログを Amazon CloudWatch Logs などのサービスにエクスポートできます。

> **Note**
>
> ログは引き続き、アプリケーション内のエージェントによって取り込まれ、送信先に送信される必要があります。このファイルパスは、Envoy にログを送信する場所を指示するだけです。

8. リスナーを設定します。

a. [プロトコル] を選択し、Envoy がトラフィックをリッスンする [ポート] を指定します。http リスナーは、WebSocket への接続移行を許可します。[リスナーの追加] をクリックすると、複数のリスナーを追加できます。[削除] ボタンをクリックすると、そのリスナーが削除されます。

b. (オプション) 接続プールの有効化

接続プーリングにより、仮想ゲートウェイ Envoy が同時に確立できる接続の数が制限されます。これは、Envoyインスタンスを接続に圧倒されないように保護することを目的としており、アプリケーションのニーズに合わせてトラフィックシェーピングを調整できます。

仮想ゲートウェイリスナーの宛先側の接続プール設定を行うことができます。App Mesh は、クライアント側の接続プールの設定をデフォルトで無限に設定し、メッシュ設定を簡素化します。

> **Note**
>
> connectionPool と connectionPoolportMapping プロトコルは同じである必要があります。リスナープロトコルが grpc または http2 の場合は、maxRequests のみを指定します。リスナープロトコルが http の場合、maxConnections と maxPendingRequests の両方を指定できます。

- [最大接続数] に送信接続の最大数を指定します。
- [Maximum requests] (最大リクエスト数) に、仮想ゲートウェイ Envoy で確立できる並列リクエストの最大数を指定します。
- (オプション) [最大保留リクエスト数] に、Envoy が [最大接続数] をキューに入れた後、オーバーフローするリクエストの数を指定します。デフォルト値は、「2147483647」です。

c. (オプション) リスナーのヘルスチェックを設定する場合は、[ヘルスチェックの有効化] を選択します。

ヘルスチェックポリシーはオプションですが、正常なポリシーに値を指定する場合は、正常なしきい値、Health チェック間隔、ヘルスチェックプロトコル、タイムアウト期間、 および非正常なしきい値の値を指定する必要があります。

- [ヘルスチェックプロトコル] で、プロトコルを選択します。grpc を選択した場合、サービスは GRPC Health CheckingProtocol に準拠している必要があります。
- [ヘルスチェックポート] に、ヘルスチェックを実行するポートを指定します。
- [正常なしきい値] に、リスナーが正常であると宣言するために必要となるヘルスチェック成功の数を指定します。
- [ヘルスチェック間隔] に、各ヘルスチェックの実行間隔をミリ秒単位で指定します。
- [パス] に、ヘルスチェックリクエストの送信先パスを指定します。この値は、ヘルスチェックプロトコルが http または http2 の場合のみ使用されます。この値は、他のプロトコルでは無視されます。

- [タイムアウト期間] に、ヘルスチェックからの応答を受け取るまで待機する時間をミリ秒単位で指定します。
- [非正常なしきい値] に、リスナーが異常であると宣言するために必要となるヘルスチェック失敗の数を指定します。

d. (オプション) 仮想ノードが TLS を使用してこの仮想ゲートウェイと通信するかどうかを指定する場合は、[TLS ターミネーションを有効化] を選択します。

- [モード]で、リスナーで TLS を設定するモードを選択します。
- [証明書メソッド] で、次のいずれかのオプションを選択します。証明書は特定の要件を満たしている必要があります。詳細については、「証明書の要件」を参照してください。
  - [AWS Certificate Manager ホスティング] — 既存の証明書を選択します。
  - [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使用して取得するシークレットの名前を入力します。
  - ローカルファイルホスティング - Envoy がデプロイされているファイルシステム上の証明書チェーンとプライベートキーファイルへのパスを指定します。
- (オプション) クライアントが証明書を提供する場合、相互 TLS 認証を有効にするには、クライアント証明書が必要と次のオプションのいずれかを選択します。相互 TLS の詳細については、App Mesh の「相互 TLS 認証」を参照してください。
  - [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使用して取得するシークレットの名前を入力します。
  - ローカルファイルホスティング — Envoy がデプロイされているファイルシステム上の証明書チェーンファイルへのパスを指定します。
- (オプション) サブジェクトの別名を入力します。SAN を追加するには、[Add SAN] (SAN を追加) を選択します。SAN は FQDN または URI 形式である必要があります。

9. [仮想ゲートウェイを作成] を選択して終了します。

AWS CLI

AWS CLI を使用して仮想ゲートウェイを作成するには

以下のコマンドと入力 JSON ファイル (##の値を独自の値に置き換えてください) を使用して、仮想ゲートウェイを作成します。

1.
```
aws appmesh create-virtual-gateway \
```

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

2. create-virtual-gateway.json の例の内容:

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

3. 出力例:

```
{
    "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 Mesh の AWS CLI を使用して仮想ゲートウェイを作成する方法の詳細については、AWS CLI リファレンスの create-virtual-gateway コマンドを参照してください。

仮想ゲートウェイのデプロイ

Envoy コンテナのみを含む Amazon ECS または Kubernetes サービスをデプロイします。また、Amazon EC2 インスタンスに Envoy コンテナをデプロイすることもできます。詳細については、「App Mesh と Amazon EC2 の開始方法」を参照してください。Amazon ECS にデプロイする方法の詳細については、「App MeshとAmazon ECS の使用を開始する」または「AWS AppMesh と Kubernetes を使用してKubernetesにデプロイする方法」を参照してください。APPMESH_RESOURCE_ARN 環境変数を mesh/*mesh-name*/virtualGateway/*virtual-gateway-name* に設定する必要があります。また、プロキシ設定を指定しないでください。そうすると、プロキシのトラフィックがそれ自体にリダイレクトされないようになります。デフォルトでは、App Mesh は、Envoy がメトリックとトレースで自身を参照しているときに APPMESH_RESOURCE_ARN 指定したリソースの名前を使用します。APPMESH_RESOURCE_CLUSTER 環境変数に独自の名前を設定することで、この動作を上書きできます。

コンテナの複数のインスタンスをデプロイし、Network Load Balancer を設定して、インスタンスへのトラフィックの負荷分散を行うようお勧めします。ロードバランサーのサービスディスカバリ名は、外部サービスが、*myapp.example.com* のような、メッシュ内のリソースにアクセスするために使用する名前です。詳細については、「Network Load Balancer の作成」(Amazon ECS)、「外部ロードバランサーの作成」(Kubernetes)、または「チュートリアル: Amazon EC2 でのアプリケーションの可用性を高める」を参照してください。また、さらに多くの例やチュートリアルについては、App Mesh の例を参照してください。

プロキシ認証を有効にします。詳細については、「Envoy プロキシの認可」を参照してください。

仮想ゲートウェイの削除

AWS Management Console

AWS Management Console CLI を使用してト仮想ートウェイを削除するには

1. App Mesh コンソールを https://console.aws.amazon.com/appmesh/ で開きます。
2. 仮想ゲートウェイを削除するメッシュを選択します。自分が所有しているメッシュや、共有されているメッシュがすべて一覧表示されます。
3. 左側のナビゲーションで [仮想ルーター] を選択します。
4. 削除する仮想ゲートウェイを選択したら、[削除] を選択します。仮想ゲートウェイが関連付けられている場合は、仮想ゲートウェイを削除できません。まず、関連するゲートウェイルートを削除する必要があります。アカウントがリソース所有者として一覧されている仮想ゲートウェイのみを削除できます。
5. 確認ボックスに **delete** と入力し、次に、[削除] を選択します。

AWS CLI

AWS CLI CLI を使用してト仮想ートウェイを削除するには

1. 以下のコマンドを使用して仮想ゲートウェイを削除します (*##*の値を独自の値に置き換えてください)。

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

2. 出力例:

```
{
    "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 Mesh の AWS CLI を使用して仮想ゲートウェイを削除する方法の詳細については、AWS CLI リファレンスの delete-virtual-gateway コマンドを参照してください。

ゲートウェイルート

ゲートウェイルートは、仮想ゲートウェイにアタッチされ、トラフィックを既存の仮想サービスにルーティングします。ルートは、リクエストと一致すると、トラフィックをターゲットの仮想サービスに分散できます。このトピックは、サービスメッシュ内のゲートウェイルートの操作に役立ちます。

ゲートウェイルートの作成

AWS Management Console

AWS Management Console を使用して ゲートウェイルートを作成するには

1. App Mesh コンソールを https://console.aws.amazon.com/appmesh/ で開きます。
2. ゲートウェイルートを作成するメッシュを選択します。自分が所有しているメッシュや、共有されているメッシュがすべて一覧表示されます。
3. 左側のナビゲーションで [仮想ルーター] を選択します。

4. 新しいゲートウェイルートを関連付ける仮想ゲートウェイを選択します。何も表示されていない場合は、最初に仮想ゲートウェイを作成する必要があります。アカウントがリソース所有者として一覧されている仮想ゲートウェイのゲートウェイルートのみを作成できます。
5. [ゲートウェイルート] テーブルで、[ゲートウェイルートを作成] を選択します。
6. [ゲートウェイルート名] に、ゲートウェイルートに使用する名前を指定します。
7. [ゲートウェイルートタイプ] で、http、http2、grpc のいずれかを選択します。
8. 既存の仮想サービス名を選択します。何も表示されない場合は、最初に仮想サービスを作成する必要があります。
9. [仮想サービスプロバイダーポート] のターゲットに対応するポートを選択します。仮想サービスプロバイダーポートは、選択した仮想サービスのプロバイダー (ルーターまたはノード) に複数のリスナーが含まれる場合に必要です。
10. (オプション) [優先度] で、このゲートウェイルートの優先度を指定します。
11. [一致] で、次を指定します。

    - http/http2 を選択したタイプは、次のとおりです。
      - (オプション) [メソッド] - 着信した http/http2 リクエストと照合するメソッドヘッダーを指定します。
      - (オプション) [ポートの一致] - 受信トラフィックのポートを照合します。この仮想ルーターに複数のリスナーがある場合は、ポートを一致させる必要があります。
      - (オプション) [完全一致/サフィックスのホスト名] - ターゲット仮想サービスにルーティングするために、着信リクエストで一致する必要があるホスト名を指定します。
      - (オプション) [プレフィックス/完全一致/正規表現パス] - URL のパスを照合する方法です。
        - [プレフィックスの一致] - デフォルトでは、ゲートウェイルートによって一致したリクエストがターゲット仮想サービスの名前に書き換えられ、一致したプレフィックスが / に書き換えられます。仮想サービスの設定方法によっては、仮想ルーターを使用して、特定のプレフィクスまたはヘッダーに基づいて、異なる仮想ノードにリクエストをルーティングできます。

      > **Important**
      >
      > - `/aws-appmesh*` または `/aws-app-mesh*` のどちらも、プレフィックスの一致に対して指定できません これらのプレフィックスは、将来の App Mesh 内部で使用するために予約されています。

- 複数のゲートウェイルートが定義されている場合、リクエストは最長のプレフィックスを持つルートと一致されます。例えば、2 つのゲートウェイルートが存在し、一方が `/chapter` のプレフィックスを持ち、一方が `/` のプレフィックスを持つ場合、`www.example.com/chapter/` へのリクエストは `/chapter` のプレフィックスを持つゲートウェイルートに一致されることになります。

> **Note**
>
> 有効にするとパス/プレフィックスベースの一致を有効化すると、App Mesh は、パスの正規化 (normalize、merge_slashes) を有効化して、パス混乱の脆弱性を最小限に抑えることができます。
> パス混乱の脆弱性は、リクエストに参加している当事者が異なるパス表現を使用する場合に発生します。

  - [完全一致] - exact パラメータは、ルートの部分一致を無効にし、パスが現在の URL と完全一致である場合にのみルートを返すようにします。
  - [正規表現一致] - 複数の URL がウェブサイト上の 1 つのページを実際に識別する可能性のあるパターンを記述するために使用します。
- (オプション) [クエリパラメータ] - このフィールドでは、クエリパラメータで照合できます。
- (オプション) [ヘッダー] - http と http2 のヘッダーを指定します。受信したリクエストに一致させて、対象の仮想サービスにルーティングする必要があります。

- grpc が選択されたタイプの場合：
  - [ホスト名の一致タイプ] と (オプション) [完全一致/サフィックス一致] - ターゲット仮想サービスにルーティングするために、着信リクエストで照合するホスト名を指定します。
  - [grpc サービス名] - grpc サービスはアプリケーションの API として機能し、ProtoBuf で定義されます。

> **Important**
>
> サービス名に対して `/aws.app-mesh*` または `aws.appmesh` は指定できません これらのサービス名は、将来の App Mesh の内部使用のために予約されています。

- (オプション) [メタデータ] - grpc のメタデータを指定します。ターゲット仮想サービスにルーティングする着信リクエストと一致する必要があります。

12. (オプション) 書き換えに対して、次を設定します。

- http/http2 が選択されたタイプの場合：
  - プレフィックス が選択された一致タイプの場合：
    - [ホスト名の自動書き換えを上書き] - デフォルトでは、ホスト名はターゲット仮想サービスの名前に書き換えられます。
    - プレフィックスの自動書き換えを上書きする - オンにすると、プレフィックス書き換えは、書き換えられたプレフィクスの値を指定します。
  - [完全一致] が選択された一致タイプの場合:
    - [ホスト名の自動書き換えを上書き] - デフォルトでは、ホスト名はターゲット仮想サービスの名前に書き換えられます。
    - [パスの書き換え] - 書き換えられたパスの値を指定します。デフォルトパスはありません。
  - [正規表現] が選択された一致タイプの場合:
    - [ホスト名の自動書き換えを上書き] - デフォルトでは、ホスト名はターゲット仮想サービスの名前に書き換えられます。
    - [パスの書き換え] - 書き換えられたパスの値を指定します。デフォルトパスはありません。
- gRPC が選択されたタイプの場合:
  - [ホスト名の自動書き換えを上書き] - デフォルトでは、ホスト名はターゲット仮想サービスの名前に書き換えられます。

13. [ゲートウェイルートを作成] を選択して終了します。

AWS CLI

AWS CLI を使用して ゲートウェイルートを作成するには

以下のコマンドと入力 JSON ファイル (##の値を独自の値に置き換えてください) を使用して、ゲートウェイルートを作成します。

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. create-gateway-route.json の例の内容:

```
{
    "spec": {
        "httpRoute" : {
            "match" : {
                "prefix" : "/"
            },
            "action" : {
                "target" : {
                    "virtualService": {
                        "virtualServiceName": "serviceA.svc.cluster.local"
                    }
                }
            }
        }
    }
}
```

3. 出力例:

```
{
    "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 Mesh の AWS CLI を使用してゲートウェイルートを作成する方法の詳細については、AWS CLI リファレンスの create-gateway-route コマンドを参照してください。

ゲートウェイルートの削除

AWS Management Console

AWS Management Console 使用してゲートウェイルートを削除するには

1. App Mesh コンソールを https://console.aws.amazon.com/appmesh/ で開きます。
2. ゲートウェイルートを削除するメッシュを選択します。自分が所有しているメッシュや、共有されているメッシュがすべて一覧表示されます。
3. 左側のナビゲーションで [仮想ルーター] を選択します。
4. ゲートウェイルートを削除する仮想ゲートウェイを選択します。
5. [ゲートウェイルート] テーブルで、削除するゲートウェイルートを選択し、[削除]を選択します。アカウントがリソース所有者として一覧表示されている場合にのみ、ゲートウェイルートを削除できます。

6. 確認ボックスで、「**delete**」と入力し、[削除] をクリックします。

AWS CLI

AWS CLI 使用してゲートウェイルートを削除するには

1. 以下のコマンドを使用してゲートウェイルートを削除します (*##*の値を独自の値に置き換えてください)。

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

2. 出力例:

```
{
    "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 Mesh の AWS CLI を使用してゲートウェイルートを削除する方法の詳細については、AWS CLI リファレンスの delete-gateway-route コマンドを参照してください。

仮想ノード

仮想ノードは、Amazon ECS サービスや Kubernetes デプロイメントなどの特定のタスクグループへの論理ポインタとして機能します。仮想ノードを作成するときには、タスクグループのサービス検出メソッドを指定する必要があります。仮想ノードが期待するインバウンドトラフィックはすべて、リスナーとして指定されます。仮想ノードがアウトバウンドトラフィックを送信する仮想サービスは、バックエンドとして指定されます。

新しい仮想ノードのレスポンスメタデータには、仮想ノードに関連付けられている Amazon リソースネーム (ARN) が含まれています。この値を、Amazon ECS タスク定義または Kubernetes ポッド仕様のタスクグループの Envoy プロキシコンテナの `APPMESH_RESOURCE_ARN` 環境変数として設定します。例えば、値は `arn:aws:appmesh:`*`us-west-2`*`:`*`111122223333`*`:mesh/`*`myMesh`*`/virtualNode/`*`myVirtualNode`* になる可能性があります。これは、`node.id` および `node.cluster` Envoy パラメータにマッピングされます。これらの変数を設定するときは、Envoy イメージの `1.15.0` 以降を使用する必要があります。App Mesh Envoy 変数の詳細については、「Envoy」を参照してください。

> **Note**
>
> デフォルトでは、App Mesh は、Envoy によってメトリクスとトレースでそれ自体が参照されるとき、`APPMESH_RESOURCE_ARN` で指定したリソースの名前を使用します。`APPMESH_RESOURCE_CLUSTER` 環境変数に独自の名前を設定することで、この動作を上書きできます。

仮想ノードの作成

AWS Management Console

AWS Management Console で仮想ノードを作成するには

1. https://console.aws.amazon.com/appmesh/ で AppMesh コンソールを開きます。
2. 仮想ノードを作成するメッシュを選択します。自分が所有し、これまでに共有された、すべてのメッシュが一覧表示されます。
3. 左側のナビゲーションで [仮想ノード] を選択します。
4. [仮想ノードの作成] を選択し、仮想ノードの設定を指定します。
5. [仮想ノード名] で、仮想ノードの名前を入力します。
6. [サービスの検出方法] で、次のいずれかのオプションを選択します。

   - DNS — 仮想ノードが表す実際のサービスの [DNS ホスト名] を指定します。Envoy プロキシは、Amazon VPC にデプロイされます。プロキシは、VPC 用に設定された DNS サーバーに、名前解決リクエストを送信します。ホスト名が解決されると、DNS サーバーは 1 つ以上の IP アドレスを返します。VPC の DNS 設定の詳細については、「VPC での DNS の使用」を参照してください。[DNS response type] (DNS レスポンスのタイプ) (オプション) に、DNS リゾルバによって返されるエンドポイントのタイプを指定します。「ロードバランサー」とは、DNS リゾルバがロードバランシングされたエンドポイントのセットを返すことを意味します。「エンドポイント」とは、DNS リゾルバがすべてのエンドポイントを返すことを意味します。デフォルトでは、レスポンスタイプはロードバランサーであると想定されています。

     > **Note**
     >
     > Route53 を使用する場合、ロードバランサーを使用する必要があります。

   - [AWS Cloud Map] — 既存の [サービス名] と [名前空間] を指定します。オプションで、[行の追加] を選択し、キーと値を指定することで、App Mesh が AWS Cloud Map をクエリできる属性を指定することもできます。指定されたすべてのキーと値のペアに一致するインスタンスのみが返されます。AWS Cloud Map を使用するには、アカウントには `AWSServiceRoleForAppMesh` サービスにリンクされたロールがなくてはなりません。AWS Cloud Map の詳細については、「AWS Cloud Map デベロッパーガイド」を参照してください。
   - なし – 仮想ノードがインバウンドトラフィックを予期しない場合に選択します。

7. [IP バージョンの設定]

   [デフォルトの IP バージョンの動作を上書きする] をオンに切り替えて、メッシュ内のトラフィックに使用する IP バージョンを制御します。デフォルトでは、App Mesh はさまざまな IP バージョンを使用します。

   > **Note**
   >
   > 仮想ノードで IP 優先設定を設定すると、そのノード上のメッシュに設定された IP優先設定のみが上書きされます。

   - デフォルト
     - Envoy の DNS リゾルバーは IPv6 を優先し、IPv4 にフォールバックします。
     - 利用可能な場合は AWS Cloud Map から返された IPv4 アドレスを使用し、フォールバックする場合は IPv6 アドレスを使用します。
     - ローカルアプリ用に作成されたエンドポイントは IPv4 アドレスを使用します。
     - Envoy リスナーはすべての IPv4 アドレスにバインドされます。
   - IPv6 優先
     - Envoy の DNS リゾルバーは IPv6 を優先し、IPv4 にフォールバックします。
     - 利用可能な場合は AWS Cloud Map から返された IPv6 アドレスを使用し、フォールバックする場合は IPv4 アドレスを使用します。
     - ローカルアプリ用に作成されたエンドポイントは IPv6 アドレスを使用します。
     - Envoy リスナーはすべての IPv4 および IPv6 アドレスにバインドされます。
   - IPv4 優先
     - Envoy の DNS リゾルバーは IPv4 を優先し、IPv6 にフォールバックします。
     - 利用可能な場合は AWS Cloud Map から返された IPv4 アドレスを使用し、フォールバックする場合は IPv6 アドレスを使用します。
     - ローカルアプリ用に作成されたエンドポイントは IPv4 アドレスを使用します。
     - Envoy リスナーはすべての IPv4 および IPv6 アドレスにバインドされます。
   - IPv6 のみ
     - Envoy の DNS リゾルバーは IPv6 のみを使用します。

    - AWS Cloud Map によって返された IPv6 アドレスのみが使用されます。AWS Cloud Map が IPv4 アドレスを返す場合、IP アドレスは使用されず、空の結果が Envoy に返されます。
    - ローカルアプリ用に作成されたエンドポイントは IPv6 アドレスを使用します。
    - Envoy リスナーはすべての IPv4 および IPv6 アドレスにバインドされます。
  - IPv4 のみ
    - Envoy の DNS リゾルバーは IPv4 のみを使用します。
    - AWS Cloud Map によって返された IPv4 アドレスのみが使用されます。AWS Cloud Map が IPv6 アドレスを返す場合、IP アドレスは使用されず、空の結果が Envoy に返されます。
    - ローカルアプリ用に作成されたエンドポイントは IPv4 アドレスを使用します。
    - Envoy リスナーはすべての IPv4 および IPv6 アドレスにバインドされます。
8. (オプション) クライアントポリシーのデフォルト – バックエンド仮想サービスと通信するときのデフォルトの要件を設定します。

> **Note**
>
> - 既存の仮想ノードに対して Transport Layer Security (TLS) を有効にする場合は、TLS を有効にする既存の仮想ノードと同じサービスを表す新しい仮想ノードを作成することをお勧めします。次に、仮想ルータとルートを使用して、トラフィックを新しい仮想ノードに徐々にシフトします。ルートの作成およびトランジションの重みの調整に関する詳細については、「ルート」を参照してください。既存のトラフィック処理仮想ノードを TLS で更新する場合、更新した仮想ノードの Envoy プロキシが証明書を受信する前に、ダウンストリームクライアントの Envoy プロキシが TLS 検証コンテキストを受信する可能性があります。これにより、ダウンストリームの Envoy プロキシで TLS ネゴシエーションエラーが発生する可能性があります。
> - プロキシ認証は、バックエンドサービスの仮想ノードによって表されるアプリケーションでデプロイされた Envoy プロキシで有効にする必要があります。プロキシ認証を有効にする場合は、この仮想ノードが通信している仮想ノードのみへのアクセスを制限するようお勧めします。

- (オプション) 仮想ノードが Transport Layer Security (TLS) を使用してすべてのバックエンドと通信するようにリクエストする場合には、[TLS の適用] を選択します。
- (オプション) 1 つ以上の特定のポートに対して TLS の使用のみが必要な場合は、[ポート] に数値を入力します。ポートを追加するには、[ポートの追加] を選択します。ポートを指定しない場合、すべてのポートに TLS が適用されます。
- 検証方法を使用する場合、次のいずれかのオプションを選択します。指定する証明書は、すでに存在し、特定の要件を満たしている必要があります。詳細については、「証明書の要件」を参照してください。
  - AWS Private Certificate Authority ホスティング — 既存の 1 つまたは複数のを選択します。証明書。ACM 証明書で暗号化を使用したサンプルアプリケーションでのメッシュデプロイの完全なエンドツーエンドのチュートリアルについては、GitHub の「AWS Certificate Manager での TLS 設定」を参照してください。
  - [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使って取得するシークレットの名前を入力します。
  - ローカルファイルホスティング — Envoy がデプロイされているファイルシステム上の証明書チェーンファイルへのパスを指定します。ローカルファイルで暗号化を使用してサンプルアプリケーションでメッシュをデプロイする完全なエンドツーエンドのチュートリアルについては、GitHub の「ファイルにより提供された TLS 証明書を使用した TLS の設定」を参照してください。
- (オプション) [Subject Alternative Name] (サブジェクトの別名) を入力します。SAN を追加するには、[Add SAN] (SAN を追加) を選択します。SAN は FQDN または URI 形式である必要があります。
- (オプション) サーバーが要求したときにクライアント証明書を提供し、相互TLS認証を有効にするには、[Provide client certificate] (クライアント証明書の提供) と、次のオプションのいずれかを選択します。相互 TLS の詳細については、App Mesh の「相互 TLS 認証」ドキュメントを参照してください。
  - [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使って取得するシークレットの名前を入力します。
  - ローカルファイルホスティング - Envoy がデプロイされているファイルシステムで、証明書チェーンファイルとプライベートキーへのパスを指定します。

9. (オプション) サービスバックエンド — 仮想ノードが通信する App Mesh 仮想サービスを指定します。

   - 仮想ノードが通信する仮想サービスの AppMesh 仮想サービス名または完全な Amazonリソース名 (ARN) を入力します。
   - (オプション) バックエンドに一意の TLS 設定を設定する場合は、[TLS 設定]、次に、[デフォルトのオーバーライド] を選択します。
     - (オプション) 仮想ノードが TLS を使用してすべてのバックエンドと通信する必要がある場合、[TLS の適用] を選択します。
     - (オプション) 1 つ以上の特定のポートに対して TLS の使用のみが必要な場合は、[ポート] に数値を入力します。ポートを追加するには、[ポートの追加] を選択します。ポートを指定しない場合、すべてのポートに TLS が適用されます。
     - 検証方法を使用する場合、次のいずれかのオプションを選択します。指定する証明書は、すでに存在し、特定の要件を満たしている必要があります。詳細については、「証明書の要件」を参照してください。
       - AWS Private Certificate Authority ホスティング — 既存の証明書の 1 つまたは複数を選択します。
       - [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使って取得するシークレットの名前を入力します。
       - ローカルファイルホスティング — Envoy がデプロイされているファイルシステム上の証明書チェーンファイルへのパスを指定します。
     - (オプション) サブジェクトの別名を入力します。SAN を追加するには、[Add SAN] (SAN を追加) を選択します。SAN は FQDN または URI 形式である必要があります。
     - (オプション) サーバーが要求したときにクライアント証明書を提供し、相互TLS認証を有効にするには、[Provide client certificate] (クライアント証明書の提供) と、次のオプションのいずれかを選択します。相互 TLS の詳細については、App Mesh の「相互 TLS 認証」ドキュメントを参照してください。
       - [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使って取得するシークレットの名前を入力します。
       - ローカルファイルホスティング - Envoy がデプロイされているファイルシステムで、証明書チェーンファイルとシークレットキーへのパスを指定します。
   - バックエンドを追加するには、[バックエンドの追加] を選択します。

10. (オプション) ログ記録

ログ記録を設定するには、Envoy が使用する HTTP アクセスログのパスを入力します。/dev/stdout パスを使用するようお勧めします。これにより、Docker ログドライバーを使用して Envoy ログを Amazon CloudWatch Logs などのサービスにエクスポートできます。

> **Note**
>
> ログは引き続き、アプリケーション内のエージェントによって取り込まれ、送信先に送信される必要があります。このファイルパスは、ログの送信先を Envoy に指示するだけです。

11. [リスナー設定]

    リスナーは、HTTP、HTTP/2、GRPC、TCP の各プロトコルをサポートします。HTTPS はサポートされていません。

    a. 仮想ノードがインバウンドトラフィックをリクエストする場合は、リスナー用のポートとプロトコルを指定します。http リスナーは、WebSocket への接続移行を許可します。[リスナーの追加] をクリックすると、複数のリスナーを追加できます。[削除] ボタンをクリックすると、そのリスナーが削除されます。

    b. (オプション) 接続プールの有効化

       接続プーリングにより、Envoy がローカルアプリケーションクラスターと同時に確立できる接続の数が制限されます。これは、ローカルアプリケーションが接続に圧倒されるのを防ぎ、アプリケーションのニーズに合わせてトラフィックシェーピングを調整できるようにすることを目的としています。

       仮想ノードリスナーの宛先側の接続プール設定を行うことができます。App Mesh は、クライアント側の接続プールの設定をデフォルトで無限に設定し、メッシュの設定を簡素化します。

       > **Note**
       >
       > connectionPool と portMapping プロトコルは同じである必要があります。リスナープロトコルが tcp の場合は、maxConnections のみを指定します。リスナープロトコルが grpc または http2 の場合は、maxRequests のみを指定します。リスナープロトコルが http の場合、maxConnections と maxPendingRequests の両方を指定できます。

- [最大接続数] に、アウトバウンド接続の最大数を指定します。
- (オプション) [Maximum pending requests] (最大保留リクエスト数) に、Envoy が [最大接続数] をキューに入れた後、オーバーフローしているリクエストの数を指定します。デフォルト値は、「2147483647」です。

c. (オプション) 外れ値の検出の有効化

クライアント Envoy で適用された外れ値の検出により、クライアントは、既知の既知の障害がある接続に対して、ほぼ即時のアクションを実行できます。これは、アップストリームサービス内の個々のホストのヘルスステータスを追跡する回路ブレーカ実装の一形態です。

外れ値の検出は、アップストリームクラスタ内のエンドポイントが他のクラスターとは異なるパフォーマンスを示しているかどうかを動的に判断し、正常な負荷分散セットからそれらを削除します。

> **Note**
>
> サーバー仮想ノードの外れ値の検出を効果的に設定するには、その仮想ノードのサービス検出方法で AWS Cloud Map または DNS を使用し、レスポンスタイプフィールドを `ENDPOINTS` に設定します。レスポンスタイプを `LOADBALANCER` に設定して DNS サービス検出方法を使用する場合、Envoy プロキシは、アップストリームサービスへのルーティングに 1 つの IP アドレスのみを選択します。これにより、一連のホストから異常なホストを排出する外れ値の検出動作を無効にします。サービスディスカバリタイプに関する Envoy プロキシの動作の詳細については、「サービス検出方法」セクションを参照してください。

- [サーバーエラー] に、排出に必要な連続した 5xx エラーの数を指定します。
- [Outlier detection interval] (外れ値の検出間隔) に、排出のスイープ解析の時間間隔と単位を指定します。
- [Base ejection duration] (ベース突出時間) に、ホストが排出される基本時間と単位を指定します。
- [Ejection percentage] (突出パーセンテージ) に、負荷分散プール内の排出可能なホストの最大パーセンテージを指定します。

d. (オプション) [ヘルスチェックを有効化] — ヘルスチェックポリシーに関する設定を行います。

ヘルスチェックポリシーはオプションですが、ヘルスポリシーに値を指定する場合は、正常なしきい値、ヘルスチェック間隔、ヘルスチェックプロトコル、タイムアウト期間、非正常なしきい値の値を指定する必要があります。

- [ヘルスチェックプロトコル] で、プロトコルを選択します。grpc を選択した場合、サービスは GRPC Health CheckingProtocol に準拠している必要があります。
- [ヘルスチェックポート] に、ヘルスチェックを実行するポートを指定します。
- [正常なしきい値] に、リスナーが正常であると宣言するために必要となるヘルスチェック成功の数を指定します。
- [ヘルスチェック間隔] に、各ヘルスチェックの実行間隔をミリ秒単位で指定します。
- [パス] に、ヘルスチェックリクエストの送信先パスを指定します。この値は、ヘルスチェックプロトコルが `http` または `http2` の場合のみ使用されます。この値は、他のプロトコルでは無視されます。
- [タイムアウト期間] に、ヘルスチェックからの応答を受け取るまで待機する時間をミリ秒単位で指定します。
- [非正常なしきい値] に、リスナーが異常であると宣言するために必要となるヘルスチェック失敗の数を指定します。

e. (オプション) [TLS ターミネーションの有効化] – TLS を使用して他の仮想ノードがこの仮想ノードと通信する方法を設定します。

- [モード]で、リスナーで TLS を設定するモードを選択します。
- [証明書メソッド] で、次のいずれかのオプションを選択します。証明書は特定の要件を満たしている必要があります。詳細については、「証明書の要件」を参照してください。
  - [AWS Certificate Manager ホスティング] — 既存の証明書を選択します。
  - [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使って取得するシークレットの名前を入力します。
  - [ローカルファイルホスティング] — Envoy プロキシがデプロイされているファイルシステムで、証明書チェーンファイルとシークレットキーへのパスを指定します。
- (オプション) [Require client certificates] (クライアント証明書を要求する) とクライアントが証明書を提供するときに相互 TLS 認証を有効にする次のオプションの 1 つを選

択します。相互 TLS の詳細については、App Mesh の「相互 TLS 認証」ドキュメントを参照してください。

- [Envoy Secret Discovery Service (SDS)] ホスティング — Envoy が Secret Discovery Service を使って取得するシークレットの名前を入力します。
- ローカルファイルホスティング — Envoy がデプロイされているファイルシステム上の証明書チェーンファイルへのパスを指定します。

- (オプション) サブジェクトの別名を入力します。SAN を追加するには、[Add SAN] (SAN を追加) を選択します。SAN は FQDN または URI 形式である必要があります。

f. (オプション) [タイムアウト]

> **Note**
>
> デフォルトより大きいタイムアウトを指定する場合は、必ず、デフォルトより大きいタイムアウト値を持つ仮想ルータとルートを設定してください。ただし、タイムアウトをデフォルトより低い値に減らす場合には、ルートでのタイムアウトを更新はオプションとなります。詳細については、「ルート」を参照してください。

- [リクエストのタイムアウト] — リスナーのプロトコルに grpc、http、または http2 を選択した場合には、リクエストのタイムアウトを指定できます。デフォルト値は 15 秒です。値が `0` の場合、タイムアウトが無効になります。
- [アイドル期間] — 任意のリスナープロトコルのアイドル期間を指定できます。デフォルトは 300 秒です。

12. [仮想ルーターの作成] を選択して終了します。

AWS CLI

AWS CLI を使用して仮想ノードを作成するには

以下のコマンドと入力 JSON ファイル (*##*の値を独自の値に置き換えてください) を使用して、サービス検出に DNS を使用する仮想ノードを作成します。

1.
```
aws appmesh create-virtual-node \
--cli-input-json file://create-virtual-node-dns.json
```

2. create-virtual-node-dns.json の例の内容:

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

3. 出力例:

```
{
    "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 Mesh の AWS CLI を使用して仮想ノードを作成する方法の詳細については、AWS CLI リファレンスの create-virtual-node コマンドを参照してください。

仮想ノードの削除

> **Note**
>
> 仮想ノードが任意のルートのターゲットまたは任意の仮想サービスのプロバイダーとして指定されている場合、仮想ノードを削除することはできません。

AWS Management Console

AWS Management Console を使用して仮想ノードを削除するには

1. App Mesh コンソールを https://console.aws.amazon.com/appmesh/ で開きます。
2. 仮想ノードから削除するメッシュを選択します。自分が所有し、これまでに共有された、すべてのメッシュが一覧表示されます。
3. 左側のナビゲーションで [仮想ノード] を選択します。
4. [仮想ノード] テーブルで、削除する仮想ノードを選択し、[削除] を選択します。仮想ノードを削除するには、アカウントIDが仮想ノードのメッシュ所有者またはリソース所有者の列にリストされている必要があります。
5. 確認ボックスで、**delete** と入力し、次に、[削除] を選択します。

AWS CLI

AWS CLI を使用して仮想ノードを削除するには

1. 以下のコマンドを使用して仮想ノードを削除します (*##*の値を独自の値に置き換えてください)。

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

2. 出力例:

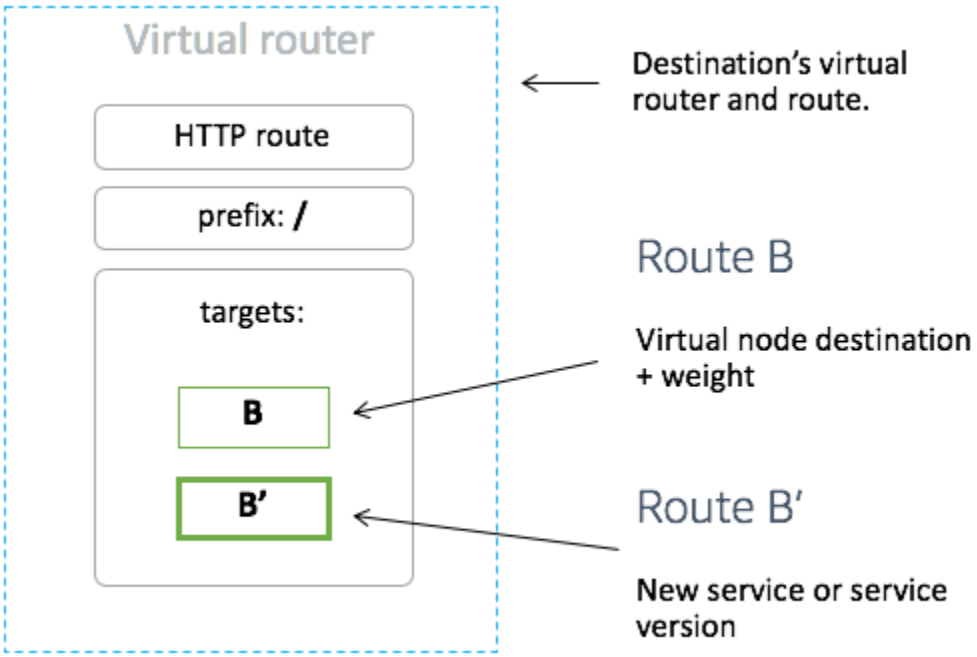
```
{
    "virtualNode": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualNode/nodeName",
            "createdAt": "2022-04-06T09:12:24.348000-05:00",
            "lastUpdatedAt": "2022-04-07T11:03:48.120000-05:00",
            "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 Mesh の AWS CLI を使用して仮想ノードを削除する方法の詳細については、AWS CLI リファレンスの delete-virtual-node コマンドを参照してください。

仮想ルーター

仮想ルーターは、メッシュ内の 1 つ以上の仮想サービスのトラフィックを処理します。仮想ルーターを作成したら、受信リクエストを別の仮想ノードに送る仮想ルーターのルートを作成して関連付けることができます。



仮想ルーターが期待するすべてのインバウンドトラフィックは、リスナーとして指定する必要があります。

仮想ルーターの作成

AWS Management Console

AWS Management Console を使用して仮想ルータを作成するには

> **Note**
>
> 仮想ルーターを作成するときは、作成した仮想ルーターにルートを関連付ける名前空間のリストを識別するラベル付きの名前空間セレクターを追加する必要があります。

1. App Mesh コンソールを https://console.aws.amazon.com/appmesh/ で開きます。
2. 仮想ルーターを作成するメッシュを選択します。所有しているメッシュや、共有されているメッシュがすべて一覧表示されます。
3. 左側のナビゲーションで [仮想ルーター] を選択します。
4. [仮想ルーターの作成] を選択します。
5. [仮想ルーター名] に仮想ルーターの名前を指定します。最大 255 文字の英字、数字、ハイフン、アンダースコアを使用できます。
6. (オプション) [リスナー] 設定には、仮想ルーターの [ポート] および [プロトコル] を指定します。`http` リスナーは、ウェブソケットへの接続移行を許可します。[リスナーの追加] をクリックすると、複数のリスナーを追加できます。[削除] ボタンをクリックすると、そのリスナーが削除されます。
7. [仮想ルーターの作成] を選択して終了します。

AWS CLI

AWS CLI を使用して仮想ルータを作成するには

以下のコマンドと入力 JSON ファイル (##の値を独自の値に置き換えてください) を使用して、仮想ルーターを作成します。

1.
```
aws appmesh create-virtual-router \
     --cli-input-json file://create-virtual-router.json
```
2. create-virtual-router.json の例の内容:
3.
```
{
```

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

4. 出力例:

```
{
    "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 Mesh の AWS CLI を使用して仮想ルーターを作成する方法の詳細については、AWS CLI リファレンスの create-virtual-router コマンドを参照してください。

仮想ルーターを削除する

> **Note**
>
> ルートがある場合、または、それが仮想サービスのプロバイダーとして指定されている場合は、仮想ルーターを削除できません。

AWS Management Console

AWS Management Console を使用して仮想ルータを削除するには

1. App Mesh コンソールをhttps://console.aws.amazon.com/appmesh/で開きます。
2. 仮想ルーターを削除するメッシュを選択します。所有しているメッシュや、共有されているメッシュがすべて一覧表示されます。
3. 左側のナビゲーションで [仮想ルーター] を選択します。
4. [仮想ルーター] テーブルで、削除する仮想ルーターを選択し、[削除] を選択します。仮想ルーターを削除するには、仮想ルーターのメッシュ所有者またはリソース所有者のいずれかの列にアカウント ID が一覧表示されている必要があります。
5. 確認ボックスで、「**delete**」と入力し、[削除] をクリックします。

AWS CLI

AWS CLI を使用して仮想ルータを削除するには

1. 以下のコマンドを使用して仮想ルーターを削除します (*##*の値を独自の値に置き換えてください)。

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

2. 出力例:

```
{
    "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 Mesh の AWS CLI を使用して仮想ルーターを削除する方法の詳細については、AWS CLI リファレンスの delete-virtual-router コマンドを参照してください。

ルート

ルートは、仮想ルーターに関連付けられます。ルートは、仮想ルーターに対するリクエストを一致させ、関連する仮想ノードにトラフィックを分散させるために使用されます。ルートは、リクエストと一致すると、1つ以上のターゲット仮想ノードにトラフィックを分散することができます。各仮想

ノードに対して相対的な重み付けを指定することができます。このトピックは、サービスメッシュ内のルートの操作に役立ちます。

ルートを作成する

AWS Management Console

AWS Management Console を使用したルートの作成

1. App Mesh コンソールをhttps://console.aws.amazon.com/appmesh/で開きます。
2. ルートを作成するメッシュを選択します。所有しているメッシュや、共有されているメッシュがすべて一覧表示されます。
3. 左側のナビゲーションで [仮想ルーター] を選択します。
4. 新しいルートを関連付ける仮想ルーターを選択します。一覧に表示されていない場合は、最初に仮想ルーターを作成する必要があります。
5. [ルート] テーブルで [ルートを作成] を選択します。ルートを作成するには、アカウント ID は、ルートのリソース所有者として一覧表示されている必要があります。
6. [ルート名] に、ルートに使用する名前を指定します。
7. ルートタイプで、ルーティングするプロトコルを選択します。選択したプロトコルは、仮想ルーターとトラフィックをルーティングする仮想ノードで選択したリスナープロトコルと一致する必要があります。
8. (オプション)ルート優先度で、ルートに使用する優先度を0 ~ 1000の範囲で指定します。ルートは、指定された値に基づいて一致させます。ここで、0 は最も高い優先順位になります。
9. (オプション) [追加設定] を選択します。下のプロトコルの中から、ルートタイプ で選択したプロトコルを選択し、コンソールで必要な設定を行います。
10. ターゲットの設定で、トラフィックをルーティングする既存の App Mesh 仮想ノードを選択し、重みを指定します。ターゲットの追加を選択して、ターゲットを追加します。すべてのターゲットのパーセンテージは 100 まで加算する必要があります。仮想ノードが一覧表示されていない場合は、最初に作成する必要があります。選択した仮想ノードに複数のリスナーがある場合は、ターゲットポートが必要です。
11. 一致の設定では、次を指定します。

    一致の設定では、*tcp* の利用はできない

    - http/http2が、選択されたタイプの場合：

- (オプション)メソッド-着信したhttp/http2リクエストに一致させるメソッドヘッダーを指定します。
- (オプション) [ポートの一致] - 受信トラフィックのポートを照合します。この仮想ルーターに複数のリスナーがある場合は、ポートを一致させる必要があります。
- (オプション)プレフィックス/完全一致/正規表現パス-URL のパスを照合する方法です。
  - プレフィックスの一致-デフォルトでは、ゲートウェイルートによって一致したリクエストがターゲット仮想サービスの名前に書き換えられ、一致したプレフィックスが書き換えられます/ 仮想サービスの設定によっては、仮想ルーターを使用して、特定のプレフィックスやヘッダーに基づき、異なる仮想ノードにリクエストをルーティングすることができます。

    > **Note**
    >
    > パス/プレフィックスに基づいた一致を有効にすると、App Mesh はパスの正規化 (normalize_path と merge_slashes) を有効にし、パス混同の脆弱性が発生する確率を最小化します。
    > パス混乱の脆弱性は、リクエストに参加している当事者が異なるパス表現を使用する場合に発生します。

  - 完全一致-exact パラメータは、ルートの部分一致を無効にし、パスが現在の URL と完全一致である場合にのみルートを返すようにします。
  - 正規表現一致-複数の URL が Web サイト上の 1 つのページを実際に識別する可能性のあるパターンを記述するために使用します。
- (オプション) クエリパラメータ - このフィールドでは、クエリパラメータで一致を行うことができます。
- (オプション)ヘッダー-httpとhttp2のヘッダを指定します。受信したリクエストに一致させて、対象の仮想サービスにルーティングする必要があります。

- gRPCが選択されている場合：
  - サービス名-リクエストを一致させる宛先サービス。
  - メソッド名-リクエストを一致させるデスティネーションメソッド。
  - (オプション) メタデータ - メタデータの存在に基づいた `Match` を指定します。リクエストを処理するには、すべてが一致する必要があります。

12. [ルートを作成] を選択します。

AWS CLI

AWS CLI を使用してルートを作成するには

以下のコマンドと入力 JSON ファイル (##の値を独自の値に置き換えてください) を使用して、gRPC ルートを作成します。

1.
```
aws appmesh create-route \
     --cli-input-json file://create-route-grpc.json
```

2. create-route-grpc.json の例の内容:

```
{
    "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. 出力例:

```
{
    "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 Mesh の AWS CLI を使用してルートを作成する方法の詳細については、AWS CLI リファレンスの create-route コマンドを参照してください。

gRPC

(オプション) [一致]

- (オプション) リクエストに一致する宛先サービスの [サービス名] を入力します。名前を指定しない場合は、任意のサービスに対するリクエストを一致させます。
- (オプション) リクエストに一致する宛先メソッドのメソッド名を入力します。名前を指定しない場合は、任意のメソッドに対するリクエストを一致させます。メソッド名を指定する場合は、サービス名を指定する必要があります。

(オプション) [メタデータ]

[メタデータの追加] を選択します。

- (オプション) ルーティングする [メタデータ名] を入力し、[照合タイプ] を選択し、[照合値] を入力します。[反転] を選択すると、その逆が照合されます。例えば、[メタデータ名] を `myMetadata`、[照合タイプ] を [完全一致]、[照合値] を123、そして [反転] を選択している場合、ルートは、123 以外の何かで始まるメタデータ名を持つすべてのリクエストに一致します。
- (オプション) [メタデータの追加] を選択して、メタデータアイテムを 10 個まで追加します。

(オプション)ポリシーを再試行する

再試行ポリシーは、断続的なネットワーク障害や断続的なサーバー側の障害からクライアントを保護することができます。再試行ポリシーはオプションですが、推奨されています。再試行のタイムアウト値は、再試行ごとのタイムアウトを定義します (最初の試行を含む)。再試行ポリシーを定義しない場合、App Mesh は各ルートのデフォルトポリシーを自動的に作成することがあります。詳細については、「デフォルトのルート再試行ポリシー」を参照してください。

- 再試行タイムアウトには、タイムアウト時間の単位数を入力します。プロトコル再試行イベントを選択する場合は、値が必要です。
- 再試行タイムアウトユニットで、単位を選択します。プロトコル再試行イベントを選択する場合は、値が必要です。
- 最大再試行回数に、リクエストが失敗した場合の再試行の最大回数を入力します。プロトコル再試行イベントを選択する場合は、値が必要です。少なくとも 2 つの値を推奨します。
- [HTTP 再試行イベント]を1つ以上選択します。少なくとも [ストリームエラー] と [ゲートウェイエラー] を選択するようお勧めします。
- [TCP 再試行イベント]を選択してください。。

- 1 つまたは複数の[gRPC 再試行イベント]を選択します。少なくとも[キャンセルされました] と [利用できません] を選択するようお勧めします。

(オプション) タイムアウト

- デフォルト値は 15 秒です。[ポリシーを再試行する] を指定した場合、ここで指定する期間は、再試行ポリシーで定義した最大再試行回数を乗じた再試行期間以上でなければならず、再試行ポリシーを完了させることができないため、常に長くする必要があります。15 秒を超える期間を指定する場合は、任意の仮想ノードTargetのリスナーに指定されたタイムアウトも15秒以上であることを確認します。詳細については、「仮想ノード」をご覧ください。
- `0` の値は、タイムアウトを無効にします。
- ルートをアイドル状態にすることができる最大時間。

HTTP と HTTPS

(オプション) 一致

- ルートが必ず一致する [プレフィックス] を指定します。例えば、仮想サービス名が `service-b.local` である場合、ルートと `service-b.local/metrics` へのリクエストを一致させるには、プレフィックスを `/metrics` にする必要があります。指定する/すべてのトラフィックをルーティングします。
- (オプション) [メソッド] を選択します。
- (オプション) [スキーム] を選択します。HTTP2 ルートにのみ適用されます。

(オプション) ヘッダー

- (オプション) [ヘッダーを追加] を選択します。ルーティングするヘッダー名を入力し、[照合タイプ] を選択し、[照合値] を入力します。[反転] を選択すると、その逆に一致します。例えば、`clientRequestId` というヘッダーと `123` という[プレフィックス] を指定し、[Invert] を選択すると、`123` 以外で始まるヘッダーを持つすべてのリクエストに対してルートが一致されます。。
- (オプション) [ヘッダーを追加] を選択します。最大 10 個のヘッダーを追加できます。

(オプション) 再試行ポリシー

再試行ポリシーは、断続的なネットワーク障害や断続的なサーバー側の障害からクライアントを保護することができます。再試行ポリシーはオプションですが、推奨されています。再試行のタイムアウト値は、再試行ごとのタイムアウトを定義します (最初の試行を含む)。再試行ポリシーを定義しない場合、App Mesh は各ルートのデフォルトポリシーを自動的に作成することがあります。詳細については、「デフォルトのルート再試行ポリシー」を参照してください。

- 再試行タイムアウトには、タイムアウト時間の単位数を入力します。プロトコル再試行イベントを選択する場合は、値が必要です。
- 再試行タイムアウトユニットで、単位を選択します。プロトコル再試行イベントを選択する場合は、値が必要です。
- 最大再試行回数に、リクエストが失敗した場合の再試行の最大回数を入力します。プロトコル再試行イベントを選択する場合は、値が必要です。少なくとも 2 つの値を推奨します。
- [HTTP 再試行イベント]を1つ以上選択します。少なくとも [ストリームエラー] と [ゲートウェイエラー] を選択するようお勧めします。
- [TCP 再試行イベント] を選択してください。

(オプション) タイムアウト

- リクエストのタイムアウト – デフォルト値は 15 秒です。再試行ポリシーを指定した場合、ここで指定する期間は、再試行ポリシーが完了することができるように、再試行期間に再試行ポリシーで定義した最大再試行回数を掛けた値以上である必要があります。
- アイドル期間 — デフォルト値は 300 秒です。
- `0` の値は、タイムアウトを無効にします。

> **Note**
>
> デフォルトより大きいタイムアウトを指定する場合は、すべての仮想ノード参加者のリスナーに指定されたタイムアウトがデフォルトよりも大きくなるようにしてください。ただし、タイムアウトをデフォルトよりも低い値に減らす場合は、仮想ノードでタイムアウトを更新することはオプションとなります。詳細については、「仮想ノード」を参照してください。

TCP

(オプション) タイムアウト

- アイドル期間 — デフォルト値は 300 秒です。
- `0` の値は、タイムアウトを無効にします。

ルートの削除

AWS Management Console

AWS Management Console を使用して、ルートを削除するには

1. App Mesh コンソールをhttps://console.aws.amazon.com/appmesh/で開きます。
2. ルートを削除するメッシュを選択します。所有しているメッシュや、共有されているメッシュがすべて一覧表示されます。
3. 左側のナビゲーションで [仮想ルーター] を選択します。
4. ルートを削除するルーターを選択します。
5. [ルート] テーブルで、削除するルートを選択し、右上隅の [削除]を選択します。
6. 確認ボックスで、「**delete**」と入力し、[削除] をクリックします。

AWS CLI

AWS CLI を使用して、ルートを削除するには

1. 以下のコマンドを使用してルートを削除します (*##*の値を独自の値に置き換えてください)。

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

2. 出力例:

```
{
    "route": {
        "meshName": "meshName",
        "metadata": {
```

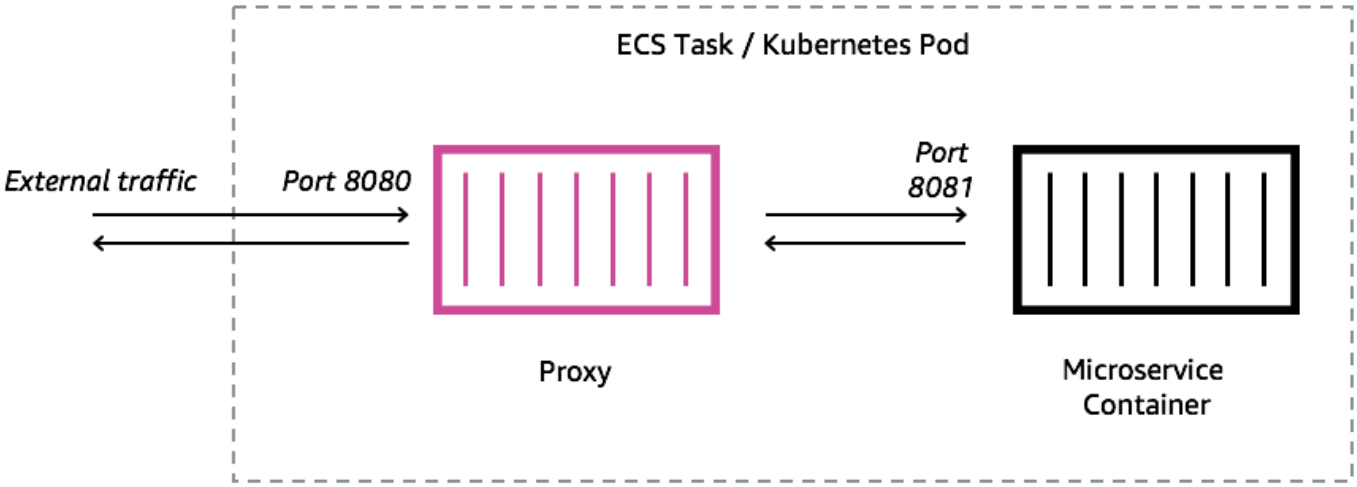
```
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualRouter/routerName/route/routeName",
            "createdAt": "2022-04-06T13:46:54.750000-05:00",
            "lastUpdatedAt": "2022-04-07T10:43:57.152000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "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 Mesh の AWS CLI を使用してルートを削除する方法の詳細については、AWS CLI リファレンスの delete-route コマンドを参照してください。

Envoy イメージ

AWS App Mesh は Envoy プロキシに基づくサービスメッシュです。



仮想ノードや仮想ゲートウェイなど、App Mesh エンドポイントで表される Amazon ECS タスク、Kubernetes ポッド、または Amazon EC2 インスタンスに Envoy プロキシを追加する必要があります。App Mesh は、最新の脆弱性とパフォーマンスの更新でパッチが適用された Envoy プロキシコンテナイメージを提供します。App Mesh は、新しいイメージを使用可能にする前に、App Mesh 機能セットに対して新しい Envoy プロキシリリースをテストします。

Envoy イメージバリアント

App Mesh は、Envoy プロキシコンテナイメージの 2 つのバリアントを提供します。この 2 つの違いは、Envoy プロキシが App Mesh データプレーンと通信する方法と、Envoy プロキシが相互に通信する方法です。1 つは標準イメージで、標準の App Mesh サービスエンドポイントと通信します。もう 1 つのバリアントは FIPS に準拠しており、App Mesh FIPS サービスエンドポイントと通信し、App Mesh サービス間の TLS 通信に FIPS 暗号化を適用します。

次のリストから地域別の画像を選択するか、`aws-appmesh-envoy`という名前の公開リポジトリから画像を選択できます。

Important

- 2023 年 6 月 30 日以降、Envoy イメージ v1.17.2.0-prod 以降のみが App Mesh と互換性があります。より前の Envoy イメージを使用している現在のお客様はv1.17.2.0、既存のエンボイは引き続き互換性がありますが、最新バージョンへの移行を強くお勧めします。
- ベストプラクティスとして、Envoy バージョンを定期的に最新バージョンにアップグレードすることを特にお勧めします。最新のセキュリティパッチ、機能リリース、パフォーマンスの向上により検証されるのは、最新の Envoy バージョンのみです。
- Version 1.17 では、Envoy が大幅に更新されています。詳細については、「Envoy 1.17 へのアップデート/移行」を参照してください。
- バージョン 1.20.0.1 以降は ARM64 と互換性があります。
- IPv6 のサポートには、Envoy バージョン 1.20 以降が必要です。

me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1、af-south-1 以外のサポートされているリージョンすべて。*Region-code* は、me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1、af-south-1 以外の任意のリージョンに置き換えることができます。

標準

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

me-south-1

標準

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

ap-east-1

標準

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

ap-southeast-3

標準

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-
fips
```

eu-south-1

標準

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

il-central-1

標準

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-
fips
```

af-south-1

標準

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

Public repository

標準

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.29.5.0-prod
```

FIPS 準拠

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.29.5.0-prod-fips
```

> **Note**
>
> 512 CPU ユニットと少なくとも 64 MiB のメモリを Envoy コンテナに割り当てるようお勧めします。Fargate では、設定できる最小メモリ容量は 1024 MiB です。コンテナのインサイトやその他の指標から負荷が高いためにリソースが不足していることが判明した場合には、Envoy コンテナへのリソース割り当てを増やすことができます。

> **Note**
>
> v1.22.0.0 以降のすべての aws-appmesh-envoy イメージリリースバージョンは、distroless の Docker イメージとしてビルドされます。この変更は、イメージサイズを小さくし、イメージ内に存在する未使用のパッケージが脆弱性にさらされる可能性を減らすことができるようにするためです。 aws-appmesh-envoy イメージ上に構築していて、一部の AL2 ベースパッケージ (yum など) と機能に依存している場合は、aws-appmesh-envoyイメージ内からバイナリをコピーして AL2 ベースで新しい Docker イメージを構築することをお勧めします。

このスクリプトを実行して、aws-appmesh-envoy:v1.22.0.0-prod-al2: タグ付きのカスタム Docker イメージを生成します。

```
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 .
```

Amazon ECR のこのコンテナイメージへのアクセスは、 AWS Identity and Access Management (IAM) によって制御されます。そのため、IAM を使用して Amazon ECR への読み取りアクセス権があることを確認する必要があります。例えば、Amazon ECS を使用する場合、適切なタスク実行ロールを Amazon ECS タスクに割り当てることができます。特定の Amazon ECR リソースへのアクセスを制限する IAM ポリシーを使用する場合は、aws-appmesh-envoy リポジトリを識別するリージョン固有の Amazon リソースネーム (ARN) へのアクセスを許可していることを確認する必要があります。例えば、us-west-2 リージョンの場合は、次のリソースへのアクセスを許可します。arn:aws:ecr:us-west-2:840364872350:repository/aws-appmesh-envoy。詳細については、「Amazon ECR Managed Policies」を参照してください。Amazon EC2 インスタンスで Docker を使用している場合は、リポジトリに対して Docker を認証します。詳細については、「レジストリの認証」を参照してください。

上流の Envoy イメージにまだマージされていない Envoy の変更に依存する新しい App Mesh 機能をリリースすることがあります。これらの新しい App Mesh 機能を、Envoy の変更を上流にマージする前に使用するには、App Mesh で提供されている Envoy コンテナイメージを使用する必要があります。変更のリストについては、「 Envoy Upstreamラベルの App Mesh GitHub ロードマップの

問題」を参照してください。App Mesh Envoy コンテナイメージを最適なサポートオプションとして使用することをお勧めします。

Envoy 設定変数

次の環境変数を使用して、App Mesh 仮想ノードタスクグループの Envoy コンテナを設定します。

> **Note**
>
> App Mesh Envoy 1.17 は Envoy の v2 xDS API をサポートしていません。Envoy 設定ファイルを受け入れる Envoy 設定変数を使用している場合、最新の v3 xDS API に更新する必要があります。

必須の変数

次の環境変数は、すべての App Mesh Envoy コンテナで必要です。この変数は、Envoy イメージのバージョン `1.15.0` 以降でのみ使用可能です。以前のバージョンのイメージを使用している場合は、代わりに `APPMESH_VIRTUAL_NODE_NAME` 変数を設定する必要があります。

`APPMESH_RESOURCE_ARN`

Envoy コンテナをタスクグループに追加する場合は、この環境変数を、タスクグループが表す仮想ノードまたは仮想ゲートウェイの ARN に設定します。例えば、ARN には、次のリストがあります。

- 仮想ノード – arn:aws:appmesh:*Region-code* : *111122223333*:mesh/*meshName*/virtualNode /*virtualNodeName*
- 仮想ゲートウェイ – arn:aws:appmesh:*Region-code* : *111122223333*:mesh/*meshName*/virtualGateway /*virtualGatewayName*

App Mesh プレビューチャンネルを使用する場合、ARN は、*us-west-2*リージョンを使用し、`appmesh` の代わりに `appmesh-preview`を使用する必要があります。例えば、App Mesh プレビューチャネル内の仮想ノードの ARN は、次のようになります。`arn:aws:appmesh-preview:us-west-2:111122223333:mesh/meshName/virtualNode/virtualNodeName`。

オプションの変数

App Mesh Envoyコンテナでは、次の環境変数がオプションとして用意されています。

`ENVOY_LOG_LEVEL`

Envoy コンテナのログレベルを指定します。

有効な値: `trace`, `debug`, `info`, `warn`, `error`, `critical`, `off`

デフォルト: `info`

`ENVOY_INITIAL_FETCH_TIMEOUT`

初期化プロセス中に Envoy が管理サーバーからの最初の設定応答を待機する時間を指定します。

詳細については、Envoy ドキュメントの「Configuration sources」を参照してください。`0` に設定すると、タイムアウトは発生しません。

デフォルト: `0`

`ENVOY_CONCURRENCY`

Envoy の起動時の `--concurrency` コマンドラインオプションを設定します。これはデフォルトで設定されていません。このオプションは Envoy バージョン `v1.24.0.0-prod` 以上で使用できます。

詳細については、Envoy ドキュメントの「Command line options」を参照してください。

管理者変数

これらの環境変数を使用して、Envoy の管理インターフェイスを設定します。

`ENVOY_ADMIN_ACCESS_PORT`

Envoy がリッスンするカスタム管理ポートを指定します。デフォルト: `9901`。

> **Note**
>
> Envoy 管理ポートは、仮想ゲートウェイまたは仮想ノード上のリスナーポートとは異なる必要があります

`ENVOY_ADMIN_ACCESS_LOG_FILE`

Envoy アクセスログを書き込む先のカスタムパスを指定します。デフォルト: `/tmp/envoy_admin_access.log`。

`ENVOY_ADMIN_ACCESS_ENABLE_IPV6`

Envoy の管理インターフェイスを IPv6 トラフィックを受け入れるように切り替えます。これにより、このインターフェイスは IPv4 と IPv6 のトラフィックの両方を受け入れることができます。デフォルトでは、このフラグは false に設定されており、Envoy は IPv4 トラフィックのみを受信します。この変数は、Envoy イメージのバージョン 1.22.0 以降でのみ使用可能です。

エージェント変数

これらの環境変数を使用して、Envoy 用の AWS App Mesh エージェントを設定します。詳細については、App Mesh の「Envoy 用エージェント」を参照してください。

`APPNET_ENVOY_RESTART_COUNT`

実行中のタスクまたはポッド内の Envoy プロキシプロセスが終了した場合に、エージェントがこのプロセスを再起動する回数を指定します。また、エージェントは Envoy が終了するたびに終了ステータスを記録し、トラブルシューティングを容易にします。この変数のデフォルト値は `0` です。デフォルト値が設定されている場合、エージェントはプロセスの再起動を試行しません。

デフォルト: `0`

最大: `10`

`PID_POLL_INTERVAL_MS`

Envoy プロキシのプロセス状態をエージェントがチェックする間隔をミリ秒単位で指定します。デフォルト値は、`100`です。

デフォルト: `100`

最小: `100`

最大: `1000`

`LISTENER_DRAIN_WAIT_TIME_S`

Envoy プロキシがアクティブな接続が閉じられてからプロセスが終了するまで待機する時間を秒単位で指定します。

デフォルト: 20

最小: 5

最大: 110

APPNET_AGENT_ADMIN_MODE

エージェントの管理インターフェイスサーバーを起動し、TCP アドレスまたは UNIX ソケットにバインドします。

有効な値: tcp、uds

APPNET_AGENT_HTTP_PORT

tcp モードでエージェントの管理インターフェイスバインドするために使用するポートを指定します。uid != 0 の場合、ポートの値が > 1024 であることを確認します。ポートが 65535 より小さいことを確認します。

デフォルト: 9902

APPNET_AGENT_ADMIN_UDS_PATH

uds モードでのエージェントの管理インターフェイスの UNIX ドメインソケットパスを指定します。

デフォルト: /var/run/ecs/appnet_admin.sock

トレースの変数

設定なし、または次のトレースドライバーのいずれかを設定できます。

AWS X-Ray 変数

次の環境変数は、 AWS X-Rayで App Mesh を設定するために使用します。詳細については、AWS X-Ray デベロッパーガイドを参照してください。

ENABLE_ENVOY_XRAY_TRACING

デフォルトのデーモンエンドポイントとして 127.0.0.1:2000 を使用し、X-Ray トレースを有効にします。有効にするには、値を 1 に設定します。デフォルト値は、0です。

XRAY_DAEMON_PORT

ポート値を指定して、デフォルトの X-Ray デーモンポート2000 を上書きします。

`XRAY_SAMPLING_RATE`

サンプリングレートを指定して、X-Ray トレーサのデフォルトのサンプリングレート `0.05` (5%)を上書きします。`1.00` と `0` (100%) の間の小数で値を指定します。`XRAY_SAMPLING_RULE_MANIFEST` が指定されている場合、この値は上書きされます。この変数は、バージョン `v1.19.1.1-prod` 以降の Envoy イメージでサポートされています。

`XRAY_SAMPLING_RULE_MANIFEST`

Envoy コンテナファイルシステム内のファイルパスを指定して、X-Ray トレーサのローカライズされたカスタムサンプリングルールを設定します。詳しくは、『AWS X-Ray デベロッパーガイド』の「サンプリングルール」をご覧ください。この変数は、バージョン `v1.19.1.0-prod` 以降の Envoy イメージでサポートされています。

`XRAY_SEGMENT_NAME`

トレースのセグメント名を指定して、デフォルトの X-Ray セグメント名を上書きします。デフォルトでは、この値は `mesh/resourceName` に設定されます。この変数は、Envoy イメージバージョン `v1.23.1.0-prod` 以降でサポートされています。

Datadog のトレース変数

次の環境変数は、Datadog エージェントトレーサーを使用して App Mesh を設定するのに役立ちます。詳細については、Datadog ドキュメントの「Agent Configuration」を参照してください。

`ENABLE_ENVOY_DATADOG_TRACING`

`127.0.0.1:8126` をデフォルトの Datadog エージェントエンドポイントとして使用し、Datadog トレース収集を有効にします。有効にするには、値を `1` (デフォルト値は `0`)に設定します。

`DATADOG_TRACER_PORT`

ポート値を指定して、デフォルトの Datadog エージェントポート 8126 を上書きします。

`DATADOG_TRACER_ADDRESS`

IP アドレスを指定して、デフォルトの Datadog エージェントアドレス： `127.0.0.1` を上書きします。

`DD_SERVICE`

トレースのサービス名を指定して、デフォルトの Datadog サービス名: `envoy-meshName/virtualNodeName` を上書きします。この変数は、バージョン `v1.18.3.0-prod` 以降の Envoy イメージでサポートされています。

Jaeger のトレース変数

次の環境変数は、Jaeger トレースを使用して App Mesh を設定するために使用します。詳細については、Jaeger ドキュメントの「開始方法」を参照してください。これらの変数は、バージョン `1.16.1.0-prod` 以降の Envoy イメージでサポートされています。

`ENABLE_ENVOY_JAEGER_TRACING`

`127.0.0.1:9411` をデフォルトの Jaeger エンドポイントとして使用して、 Jaeger トレース収集を有効にします。有効にするには、値を `1` (デフォルト値は `0`)に設定します。

`JAEGER_TRACER_PORT`

ポート値を指定して、デフォルトの Jaeger ポート 9411.を上書きします。

`JAEGER_TRACER_ADDRESS`

IP アドレスを指定して、デフォルトの Jaeger アドレス： `127.0.0.1` を上書きします。

`JAEGER_TRACER_VERSION`

コレクターが `JSON` と `PROTO` のいずれのエンコード形式のトレースを必要としているのかを指定します。デフォルトでは、この値は `PROTO` に設定されます。この変数は、Envoy イメージバージョン `v1.23.1.0-prod` 以降でサポートされています。

Envoy トレース変数

独自のトレース設定を使用するには、次の環境変数を設定します。

`ENVOY_TRACING_CFG_FILE`

Envoy コンテナファイルシステム内のファイルパスを指定します。ポリシーの詳細については、Envoy ドキュメントの「 `config.trace.v3.Tracing` 」を参照してください。

> **Note**
>
> トレース設定でトレースクラスターを指定する必要がある場合は、同じトレース設定ファイル内の `static_resources` にある関連するクラスター設定も必ず設定してください。例えば、Zipkin には、トレースコレクターをホストするクラスター名の `collector_cluster` フィールドがあるので、そのクラスターを静的に定義する必要があります。

DogStatsD 変数

DogStatsD で App Mesh を設定するには、次の環境変数を使用します。詳細については、DogStats「D ドキュメント」を参照してください。

`ENABLE_ENVOY_DOG_STATSD`

デフォルトのデーモンエンドポイント`127.0.0.1:8125`として を使用する DogStatsD 統計を有効にします。有効にするには、値を `1` に設定します。

`STATSD_PORT`

デフォルトの DogStatsD デーモンポートを上書きするポート値を指定します。

`STATSD_ADDRESS`

IP アドレス値を指定して、デフォルトの DogStatsD デーモン IP アドレスを上書きします。デフォルト: `127.0.0.1`。この変数は、Envoy イメージのバージョン `1.15.0` 以降でのみ使用可能です。

`STATSD_SOCKET_PATH`

DogStatsD デーモンの UNIX ドメインソケットを指定します。この変数が指定されておらず、 DogStatsD が有効になっている場合、この値はデフォルトで の DogStatsD デーモン IP アドレスポートになります`127.0.0.1:8125`。統計シンク設定を含む `ENVOY_STATS_SINKS_CFG_FILE`変数が指定されている場合、すべての DogStatsD 変数が上書きされます。この変数は、Envoy イメージバージョン `v1.19.1.0-prod` 以降でサポートされています。

App Mesh 変数

次の変数は、App Mesh の設定に役立ちます。

`APPMESH_PREVIEW`

値を 1 に設定して、App Mesh プレビューチャネルエンドポイントに接続します。App Mesh プレビューチャネルの使用の詳細については、「 App Mesh プレビューチャネル 」を参照してください。

`APPMESH_RESOURCE_CLUSTER`

デフォルトでは、App Mesh は、Envoy によってメトリクスとトレースでそれ自体が参照されるとき、`APPMESH_RESOURCE_ARN` で指定したリソースの名前を使用します。`APPMESH_RESOURCE_CLUSTER` 環境変数に独自の名前を設定することで、この動作を上書きできます。この変数は、Envoy イメージのバージョン `1.15.0` 以降でのみ使用可能です。

`APPMESH_METRIC_EXTENSION_VERSION`

値を 1 に設定して、AppMeshメトリック拡張機能を有効にします。App Mesh メトリクス拡張機能の使用方法の詳細については、「 App Mesh のメトリクス拡張機能 」を参照してください。

`APPMESH_DUALSTACK_ENDPOINT`

値を 1 に設定して、App Mesh Dual Stack エンドポイントに接続します。このフラグを設定すると、Envoy はデュアルスタック対応ドメインを使用します。デフォルトでは、このフラグは false に設定されており、`IPv4` ドメインにのみ接続します。この変数は、Envoy イメージのバージョン 1.22.0 以降でのみ使用可能です。

Envoy 統計変数

次の環境変数は、Envoy Stats で App Mesh を設定するために使用します。詳細については、Envoy 統計 のドキュメントを参照してください。

`ENABLE_ENVOY_STATS_TAGS`

App Mesh 定義タグ `appmesh.mesh` と `appmesh.virtual_node` の使用を有効にします。詳細については、Envoy ドキュメントの「config.metrics.v3TagSpecifier」を参照してください。有効にするには、値を 1 に設定します。

`ENVOY_STATS_CONFIG_FILE`

Envoy コンテナファイルシステム内のファイルパスを指定して、デフォルトの Stats タグ設定ファイルを独自の設定ファイルで上書きします。詳細については、「config.metrics.v3」を参照してくださいStatsConfig。

> **Note**
>
> 統計フィルターを含むカスタマイズされた統計設定で設定すると、Envoy がワールドの App Mesh 状態と適切に同期しなくなる状態になる可能性があります。これは、Envoy のバグです。Envoy で統計のフィルターを実行しないようお勧めします。フィルターが絶対に必要な場合のために、この問題のいくつかの回避策をロードマップにリストしました。

`ENVOY_STATS_SINKS_CFG_FILE`

Envoy コンテナファイルシステム内のファイルパスを指定して、デフォルト設定を独自の設定で上書きします。詳細については、Envoy ドキュメントの「config.metrics.v3StatsSink」を参照してください。

廃止された変数

環境変数 `APPMESH_VIRTUAL_NODE_NAME` と `APPMESH_RESOURCE_NAME` は、Envoy バージョン `1.15.0` 以降ではサポートされなくなりました。ただし、既存のメッシュではまだサポートされています。Envoy バージョン `1.15.0` 以降でこれらの変数を使用する代わりに、すべての App Mesh エンドポイントに対して `APPMESH_RESOURCE_ARN` を使用してください。

App Mesh で設定される Envoy のデフォルト値

次のセクションでは、App Mesh によって設定されたルート再試行ポリシーとサーキットブレーカーの Envoy デフォルト値について説明します。

デフォルトのルート再試行ポリシー

2020 年 7 月 29 日より前にアカウントにメッシュがない場合、App Mesh は 2020 年 7 月 29 日以降にアカウント内の任意のメッシュ内のすべての HTTP、HTTP/2、および gRPC リクエストに対して、デフォルトの Envoy ルート再試行ポリシーを自動的に作成します。2020 年 7 月 29 日より前にアカウントにメッシュがある場合、2020 年 7 月 29 日以前、その日現在、または以降に存在した Envoy ルートのデフォルトポリシーは作成されません。これは、AWS サポートでチケットをオープンしない限りです。サポートがチケットを処理すると、App Mesh がチケットが処理された日付以降に作成される Envoy ルートに対してデフォルトポリシーが作成されます。Envoy ルート再試行ポリシーの詳細については、Envoy ドキュメントの「config.route.v3RetryPolicy」を参照してください。

AppMesh は、App Mesh ルートを作成する、あるいはAppMesh 仮想サービスの仮想ノードプロバイダーを定義する、いずれかの場合、Envoy ルートを作成します。App Mesh ルート再試行ポリシーを作成することはできますが、仮想ノードプロバイダーの App Mesh 再試行ポリシーを作成することはできません。

デフォルトのポリシーは App Mesh API を介して表示されません。デフォルトのポリシーは Envoy を介してのみ表示されます。設定を表示するには、次の手順に従います。管理インターフェイスを有効にするそして、`config_dump` のリクエストを Envoy に送ります。このデフォルトのポリシーには、次の設定が含まれます。

- 最大再試行回数 – 2
- gRPC の再試行イベント – `UNAVAILABLE`
- HTTP リトライイベント – `503`

> **Note**
>
> 特定の HTTP エラーコードを検索する App Mesh ルート再試行ポリシーを作成することはできません。ただし、App Mesh ルート再試行ポリシーで `server-error` や `gateway-error` を検索できます。このどちらにも `503` エラーが含まれます。詳細については、「ルート」を参照してください。

- TCP 再試行イベント – `connect-failure` と `refused-stream`

> **Note**
>
> これらのイベントのいずれかを検索する App Mesh ルート再試行ポリシーを作成することはできません。ただし、App Mesh ルート再試行ポリシーで `connection-error` を検索できます。これは `connect-failure` と同じです。詳細については、「ルート」を参照してください。

- [リセット] – アップストリームサーバーがまったく応答しない場合 (切断/リセット/読み取りタイムアウト)、Envoy は再試行を試みます。

デフォルトの回路ブレーカ

App Mesh で Envoy をデプロイすると、一部のサーキットブレーカー設定に Envoy のデフォルト値が設定されます。詳細については、「 cluster」を参照してくださいCircuitBreakers。Envoy ドキュメントのしきい値。この設定は App Mesh API を介して表示されません。設定は Envoy を介してし

てのみ表示されます。設定を表示するには、次の手順に従います。管理インターフェイスを有効にするそして、config_dump のリクエストを Envoy に送ります。

2020 年 7 月 29 日より前にアカウントにメッシュがない場合、2020 年 7 月 29 日以降に作成されたメッシュにデプロイする各 Envoy について、App Mesh は、次の設定の Envoy のデフォルト値を変更して、回路ブレーカーを効果的に無効にします。2020 年 7 月 29 日より前にアカウントにメッシュがある場合、AWS サポート でチケットを開かない限り、App Mesh にデプロイした Envoy のデフォルト値は 2020 年 7 月 29 日以降に設定されます。サポートがチケットを処理すると、次の Envoy 設定の App Mesh のデフォルト値は、チケットが処理された日付以降にデプロイするすべての Envoy で App Mesh によって設定されます。

- **max_requests** – 2147483647
- **max_pending_requests** – 2147483647
- **max_connections** – 2147483647
- **max_retries** – 2147483647

> **Note**
>
> Envoy に Envoy または App Mesh のデフォルトのサーキットブレーカーの値があるかどうかにかかわらず、値を変更することはできません。

Envoy 1.17 へのアップデート/移行

SPIRE でのSecret Discovery Service (SDS)

App Mesh で SPIRE (SPIFFE ランタイム環境) を使用して信頼証明書をサービスに配布する場合は、少なくともバージョン 0.12.0 の SPIRE エージェント (2020 年 12 月にリリース) を使用していることを確認してください。これは、Envoy バージョン 1.16 以降をサポートできる最初のバージョンです。

正規表現の変更

Envoy 1.17 以降、App Mesh は Envoy を RE2 正規表現エンジンを使用して設定します。この変更は、ほとんどのユーザーに対して明白ですが、ルートまたはゲートウェイルートでの一致では、正規表現での先読みまたは後方参照は許可されなくなります。

正と負の先読み取り

正 - 正の先読みは、?= で始まる括弧で囲まれた式です。

```
(?=example)
```

これらは、文字列置換を行うときに最も有用です。なぜなら、文字を一致の一部として消費することなく、文字列を一致させることができるからです。App Mesh では、正規表現による文字列置換がサポートされていないため、これらを通常の一致に置き換えることをお勧めします。

```
(example)
```

負 - 負の先読みは、?! で始まる括弧で囲まれた式です。

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

括弧で囲まれた式は、式の一部が特定の入力と一致しないことを表明するために使用されます。ほとんどの場合、これらはゼロ数値に置き換えることができます。

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

式自体が文字クラスである場合は、クラス全体を単純に否定し、? を使用してオプションとしてマークを付けることができます。

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

ユースケースによっては、これを処理するためにルートを変更することもできます。

```
{
    "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"
                        }
                    }
                ]
            }
        }
    }
}
```

最初のルート一致は、「プレフィックス」で始まり、その後に「サフィックス」が続かないヘッダーを探します。2番目のルートは、「サフィックス」で終わるヘッダーを含め、「プレフィックス」で始まる他のすべてのヘッダーと一致するように機能します。代わりに、負の先読みを削除する方法として、これらを逆にすることができます。

```
{
    "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"
                        }
                    }
                ]
            }
        }
    }
}
```

ここでは、ルートを逆にして「suffix」で終わるヘッダーに高い優先順位を与え、「prefix」で始まる他のすべてのヘッダーは、優先順位の低いルートで一致します。

後方参照

後方参照は、前のカッコで囲まれたグループを繰り返すことで、短い式を記述する方法です。これらには、次の形式があります。

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

後方参照 \ に続く数値は、式内の n 番目の括弧で囲まれたグループのプレースホルダとして機能します。この例では、\1 は、2回目の (group1) 書き込みの代替方法として使用されています。

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

上記の例のように、後方参照を参照するグループに置き換えるだけで、これらを削除することができます。

Envoy 用エージェント

エージェントは、App Mesh 用に販売されている Envoy イメージ内のプロセスマネージャーです。エージェントは Envoy を稼働させ続け、正常性を維持し、ダウンタイムを短縮します。Envoyの統計情報と補助データをフィルタリングして、App Mesh での Envoy プロキシの操作を抽出して表示します。これにより、関連するエラーをより迅速にトラブルシューティングできます。

エージェントを使用すると、プロキシに異常が発生した場合に Envoy プロキシを再起動する回数を設定できます。障害が発生した場合、Envoy が終了するとエージェントは最終的な終了ステータスをログに記録します。この情報は、障害のトラブルシューティングに使用できます。また、エージェントは Envoy の Connection Draining を容易にするため、アプリケーションの障害に対する耐性が高まります。

以下の変数を使用して Envoy 用エージェントを設定します。

- APPNET_ENVOY_RESTART_COUNT - この変数をゼロ以外の値に設定すると、エージェントはポーリング時にプロキシプロセスのステータスが異常であると判断したときに、設定した回数まで Envoy プロキシプロセスを再起動しようとします。これにより、プロキシのヘルスチェックが失敗した場合に、コンテナオーケストレーターによるタスクやポッドの交換に比べて、再起動が速くなるため、ダウンタイムが短縮されます。
- PID_POLL_INTERVAL_MS - この変数を設定する場合、デフォルトは 100 のままです。この値に設定すると、コンテナオーケストレーターのヘルスチェックによるタスクやポッドの交換に比べて、Envoy プロセスの終了時の検出と再起動が速くなります。
- LISTENER_DRAIN_WAIT_TIME_S - この変数を設定する場合は、タスクやポッドを停止するために設定されているコンテナオーケストレーターのタイムアウトを考慮してください。例えば、この値がオーケストレーターのタイムアウトよりも大きい場合、Envoy プロキシは、オーケストレーターがタスクやポッドを強制的に停止するまでの間のみドレインできます。
- APPNET_AGENT_ADMIN_MODE - この変数を tcp または uds に設定すると、エージェントはローカル管理インターフェイスを提供します。この管理インターフェイスは Envoy プロキシと通信するための安全なエンドポイントとして機能し、ヘルスチェックやテレメトリデータのための次の API を提供し、プロキシの動作状態を要約します。
  - GET /status - Envoy の統計情報を照会し、サーバー情報を返します。
  - POST /drain_listeners - すべてのインバウンドリスナーをドレインします。
  - POST /enableLogging?level=<desired_level> - すべてのロガーの Envoy ログ記録レベルを変更します。
  - GET /stats/prometheus - Envoy の統計情報を Prometheus 形式で表示します。

- `GET /stats/prometheus?usedonly` - Envoy が更新した統計情報のみを表示します。

エージェントの設定変数の詳細については、「Envoy 設定変数」を参照してください。

新しい AWS App Mesh エージェントはバージョン `1.21.0.0` 以降の、App Mesh に最適化された Envoy イメージに含まれており、ユーザーのタスクやポッドに追加のリソースを割り当てる必要はありません。

App Mesh 可観測性

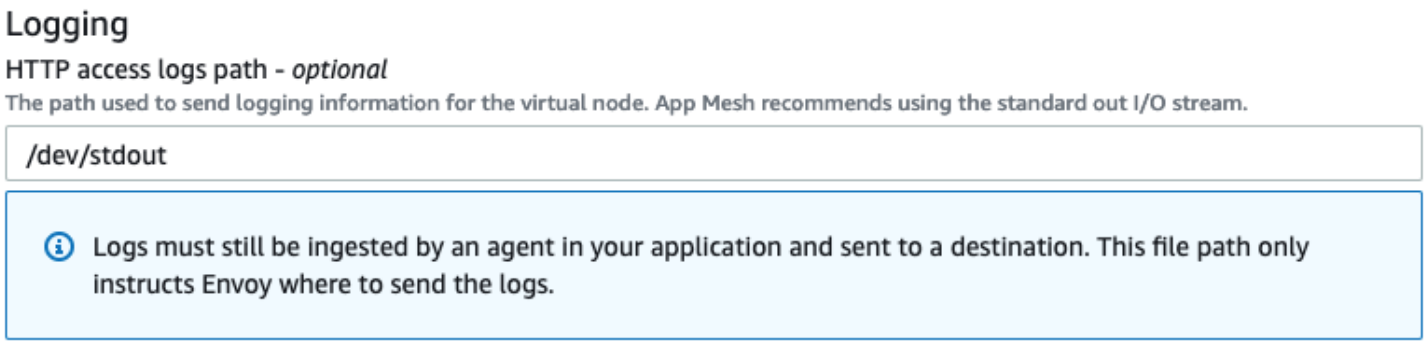
App Meshと連携することで得られるメリットの1つは、マイクロサービスアプリケーションの可視性を高めることができることです。App Mesh は、さまざまなログ記録、メトリクス、トレースソリューションと連携できます。

Envoy プロキシと App Mesh には、アプリケーションとプロキシをより明確に表示するのに役立つ次のタイプのツールが用意されています。

- ログ記録
- メトリクス
- トレース

ログ記録

仮想ノードおよび仮想ゲートウェイを作成する場合、Envoy アクセスログを設定するオプションを使用できます。コンソールでは、これは、仮想ノードと仮想ゲートウェイの作成または編集ワークフローの [ログ記録] セクションに表示されます。



前の図は、Envoy アクセスログの `/dev/stdout` のログ記録パスを示しています。

`format` には、`json` または `text` の 2 つの形式のいずれかとパターンを指定します。`json` はキーペアを受け取って JSON 構造に変換してから Envoy に渡します。

次のコードブロックは、AWS CLIで使用できる JSON 表現を示しています。

```
      "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**
>
> 入力パターンが Envoy で有効であることを確認してください。有効でない場合、Envoy は更新を拒否し、最新の変更を `error state` に保存します。

Envoy アクセスログを `/dev/stdout` に送信すると、それらは Envoy コンテナログと混合されます。`awslogs` などの標準の Docker ログドライバーを使用して、CloudWatch Logsなどのログストレージや処理サービスにそれらをエクスポートできます。詳細については、「Amazon ECS デベロッパーガイド」の「awslogs ログドライバーを使用する」を参照してください。Envoy アクセスログのみをエクスポート (他の Envoy コンテナログを無視する) するには、`ENVOY_LOG_LEVEL` に `off` を設定します。フォーマット文字列 `%REQ_WITHOUT_QUERY(X?Y):Z%` を含めることで、クエリ文字列なしのリクエストを記録できます。例については、「ReqWithoutQuery Formatter」を参照してください。詳細については、Envoy ドキュメントの「アクセスのログ記録」を参照してください。

Kubernetes でのアクセスログの有効化

Kubernetes の App Mesh コントローラーを使用している場合、次の例に示すように、仮想ノードの仕様にログ記録の設定を追加することで、アクセスログを使用して仮想ノードを設定できます。

```
---
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"
```

クラスターには、Fluentd などのログを収集するためのログフォワーダが必要です。詳細については、「CloudWatch Logs へログを送信する DaemonSet として Fluentd を設定する」を参照してください。

Envoy は、そのフィルターから stdout へのさまざまなデバッグログの書き込みもします。これらのログは、App Mesh との通信とサービス間のトラフィックの両方に関するインサイトを得るために役立ちます。特定のログ記録レベルは、ENVOY_LOG_LEVEL 環境変数を使用して設定できます。例えば、次のテキストは、Envoy が特定の HTTP リクエストで一致したクラスターを示すデバッグログの例からのものです。

```
[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 と Cloudwatch

FireLensは、Amazon ECS と AWS Fargate のログを収集するために使用できるコンテナログルーターです。Firelens の使用例は、AWSサンプルリポジトリにあります。

CloudWatch を使用して、ログ情報とメトリクスを収集できます。CloudWatch の詳細については、App Mesh ドキュメントの「エクスポートされるメトリクス」セクションを参照してください。

Envoy メトリクスを使用したアプリケーションの監視

Envoy は、メトリクスを次の主要なカテゴリに分類します。

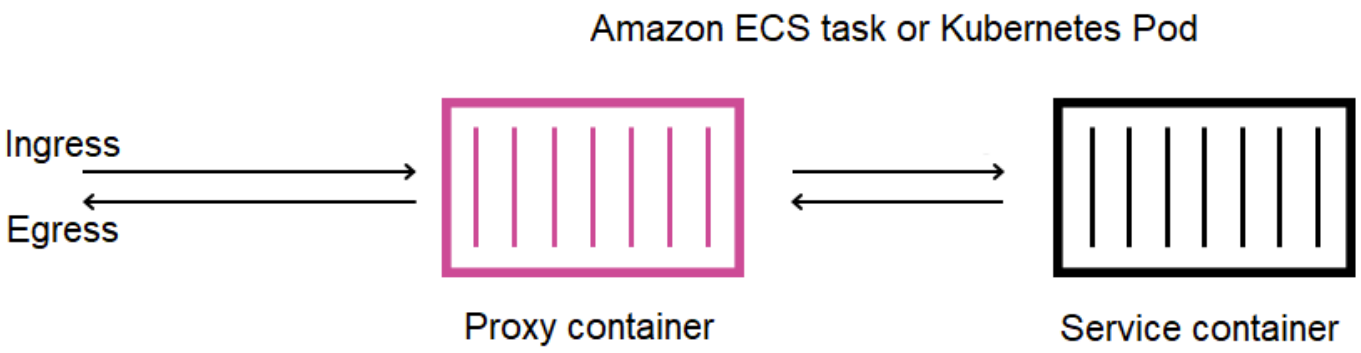
- ダウンストリーム — プロキシに入ってくる接続とリクエストに関連するメトリクスです。

- アップストリーム — プロキシによって行われた送信接続とリクエストに関連するメトリクスです。
- サーバー — Envoy の内部状態を説明するメトリクスです。これには、稼働時間や割り当てられたメモリなどのメトリクスが含まれます。

App Mesh では、プロキシがアップストリームとダウンストリームトラフィックをインターセプトします。例えば、クライアントから受信したリクエストと、サービスコンテナによって行われたリクエストは、Envoy によってダウンストリームトラフィックとして分類されます。これらの異なるタイプのアップストリームトラフィックとダウンストリームトラフィックを区別するために、App Mesh は、サービスに関連するトラフィックの方向に応じて、Envoy メトリクスをさらに分類します。

- Ingress — サービスコンテナにフローする接続とリクエストに関連するメトリクスとリソースです。
- Egress — サービスコンテナから、最終的に Amazon ECS タスクまたは Kubernetes ポッドからフローする接続とリクエストに関するメトリクスとリソースです。

次の図は、プロキシコンテナとサービスコンテナ間の通信を示しています。



リソース命名規則

Envoy がメッシュをどのように表示し、そのリソースが App Mesh で定義したリソースにどのようにマップされるかを理解しておくと便利です。App Mesh が設定する主要な Envoy リソースは次のとおりです。

- リスナー - プロキシがダウンストリーム接続をリッスンするアドレスとポートです。前の図では、App Mesh は Amazon ECS タスクまたは Kubernetes ポッドに入るトラフィックの入力リスナーと、サービスコンテナから出るトラフィックの出力リスナーを作成します。

- クラスター - プロキシが接続してトラフィックをルーティングするアップストリームエンドポイントの名前付きグループです。App Mesh では、サービスコンテナは、サービスが接続できる他のすべての仮想ノードと同様に、クラスターとして表されます。
- ルート - これらは、メッシュで定義するルートに対応します。これには、プロキシがリクエストと一致する条件と、リクエストが送信されるターゲットクラスターが含まれます。
- エンドポイントとクラスターの負荷割り当て - アップストリームクラスタの IP アドレスです。仮想ノードのサービス検出メカニズムとして AWS Cloud Map を使用する場合、App Mesh は、検出されたサービスインスタンスをエンドポイントリソースとしてプロキシに送信します。
- シークレット - これらには、暗号化キーと TLS 証明書が含まれますが、これらに限定されません。クライアント証明書とサーバー証明書のソースとして AWS Certificate Manager を使用する場合、App Mesh はパブリック証明書とプライベート証明書をシークレットリソースとしてプロキシに送信します。

App Mesh は Envoy リソースの名前に一貫したスキームを使用し、メッシュとの関連付けに使用できます。

リスナーとクラスターの命名スキームを理解することは、App Mesh で Envoy のメトリクスを理解する上で重要です。

リスナー名

リスナーは次の形式を使用して命名されます。

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

通常、Envoy で設定されている次のリスナーが表示されます。

- `lds_ingress_0.0.0.0_15000`
- `lds_egress_0.0.0.0_15001`

Kubernetes CNI プラグインまたは IP テーブルルールのいずれかを使用して、Amazon ECS タスクまたは Kubernetes ポッドのトラフィックがポート `15000` と `15001` に送信されます。App Mesh は、入力 (着信) および出力 (発信) トラフィックを受け入れるように、これらの 2 つのリスナーで Envoy を設定します。仮想ノードでリスナーが設定されていない場合、入力リスナーは表示されません。

クラスター名

ほとんどのクラスターは、次の形式を使用します。

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

サービスがそれぞれと通信する仮想ノードには、独自のクラスタがあります。前述のように、App Mesh は Envoy の隣で実行されているサービスのクラスターを作成し、プロキシが入力トラフィックを送信できるようにします。

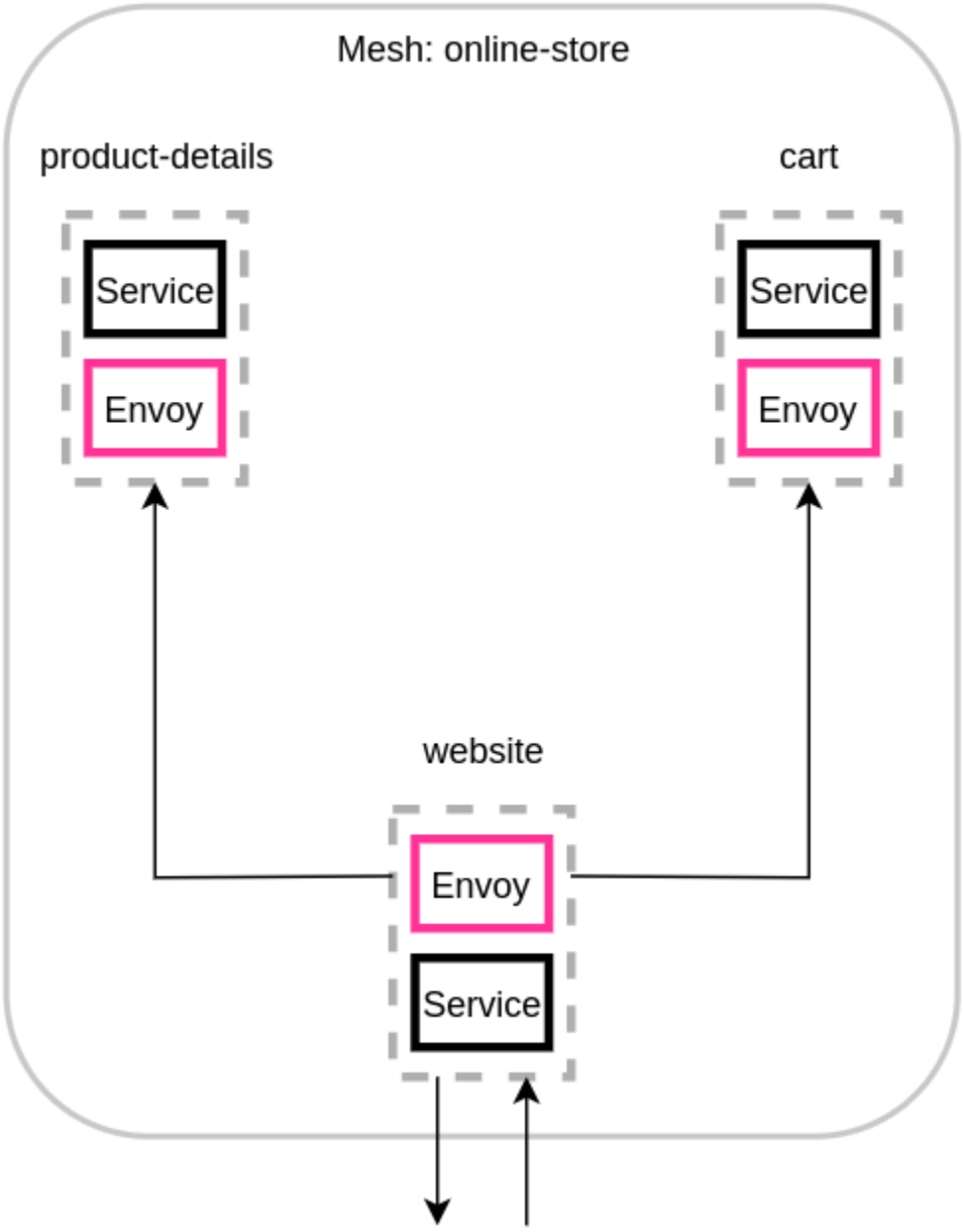
例えば、ポート 8080 で http トラフィックをリッスンする my-virtual-node という名前の仮想ノードがあり、その仮想ノードが my-mesh という名前のメッシュ内にある場合、App Mesh は cds_ingress_my-mesh_my-virtual-node_http_8080 という名前のクラスターを作成します。このクラスタは、my-virtual-node のサービスコンテナへのトラフィックの宛先として機能します。

App Mesh は、次のタイプの追加の特別なクラスターを作成することもできます。これらの他のクラスタは、メッシュで明示的に定義したリソースに必ずしも対応しているとは限りません。

- 他の AWS サービスに到達するために使用されるクラスター。このタイプでは、メッシュは、デフォルトでほとんどの AWS サービスに到達できます: cds_egress_*<mesh name>*_amazonaws
- 仮想ゲートウェイのルーティングを実行するために使用されるクラスタ。これは一般的に無視しても問題ありません:
  - 単一リスナーの場合: cds_ingress_*<mesh name>*_*<virtual gateway name>*_self_redirect_*<protocol>*_*<port>*
  - 複数のリスナーの場合: cds_ingress_*<mesh name>*_*<virtual gateway name>*_self_redirect_*<ingress_listener_port>*_*<protocol>*_*<port>*
- Envoy の Secret Discovery Service を使用してシークレットを取得するときに、TLS など、定義できるエンドポイントのクラスタ: static_cluster_sds_unix_socket です。

アプリケーションメトリクスの例

Envoy で使用可能なメトリクスを説明するために、次のサンプルアプリケーションには 3 つの仮想ノードがあります。メッシュ内の仮想サービス、仮想ルータ、ルートは Envoy のメトリクスに反映されないため、無視できます。この例では、すべてのサービスがポート 8080 で http トラフィックをリッスンします。



環境変数 ENABLE_ENVOY_STATS_TAGS=1 をメッシュで実行されている Envoy プロキシコンテナに追加するようお勧めします。これにより、プロキシによって発行されたメトリクスに、次のメトリクスディメンションが追加されます。

- appmesh.mesh
- appmesh.virtual_node
- appmesh.virtual_gateway

これらのタグは、メッシュ、仮想ノード、または仮想ゲートウェイの名前に設定され、メッシュ内のリソース名を使用してメトリクスをフィルタリングできます。

リソース名

website 仮想ノードのプロキシには、次のリソースがあります。

- 入力トラフィックと出力トラフィック用の 2 つのリスナーには、次があります。
  - `lds_ingress_0.0.0.0_15000`
  - `lds_egress_0.0.0.0_15001`
- 2 つの仮想ノードのバックエンドを表す 2 つの出力クラスター。
  - `cds_egress_online-store_product-details_http_8080`
  - `cds_egress_online-store_cart_http_8080`
- website サービスコンテナの入力クラスター。
  - `cds_ingress_online-store_website_http_8080`

リスナーメトリクスの例

- `listener.0.0.0.0_15000.downstream_cx_active` — Envoy へのアクティブな入力ネットワーク接続の数。
- `listener.0.0.0.0_15001.downstream_cx_active` — Envoy へのアクティブな出力ネットワーク接続の数。アプリケーションが外部サービスに対して行った接続は、この数に含まれます。
- `listener.0.0.0.0_15000.downstream_cx_total` — Envoy への入力ネットワーク接続の総数。
- `listener.0.0.0.0_15001.downstream_cx_total` — Envoy への出力ネットワーク接続の総数。

リスナーメトリクスの完全なセットについては、Envoy のドキュメントの「統計」を参照してください。

クラスターメトリクスの例

- `cluster_manager.active_clusters` — Envoy が少なくとも 1 つの接続を確立したクラスターの総数。
- `cluster_manager.warming_clusters` — Envoy がまだ接続していないクラスターの総数。

次のクラスターメトリクスは、cluster.<cluster name>.<metric name> の形式を使用します。これらのメトリクス名はアプリケーションの例に固有で、website の Envoy コンテナによって発行されます。

- cluster.cds_egress_online-store_product-details_http_8080.upstream_cx_total — website と product-details 間での接続の総数。
- cluster.cds_egress_online-store_product-details_http_8080.upstream_cx_connect_fail — website と product-details 間での失敗した接続の合計数。
- cluster.cds_egress_online-store_product-details_http_8080.health_check.failure — website と product-details 間での失敗したヘルスチェックの総数。
- cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_total — website と product-details 間で行われたリクエストの総数。
- cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_time — website と product-details 間で行われたリクエストにかかる時間。
- cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_2xx — roduct-details から website が受信した HTTP 2xx レスポンスの数。

HTTP メトリクスの完全なセットについては、Envoy のドキュメントの「統計」を参照してください。

管理サーバーのメトリクス

Envoy は、Envoy の管理サーバーとして機能する App Mesh コントロールプレーンへの接続に関連するメトリクスも出力します。プロキシがコントロールプレーンから長時間同期解除された場合に通知する手段として、これらのメトリクスのいくつかを監視することをお勧めします。コントロールプレーンへの接続が失われるか、更新に失敗すると、プロキシが App Mesh API を介して行われたメッシュの変更を含めて、App Mesh からの新しい設定を受信できなくなります。

- control_plane.connected_state — プロキシが App Mesh に接続されている場合、このメトリクスは 1 に設定され、それ以外の場合は 0 に設定されます。

- *.update_rejected — Envoy によって拒否された設定更新の総数。これらは通常、ユーザーの設定ミスによるものです。例えば、Envoy が読み取れないファイルから TLS 証明書を読み取るように App Mesh を設定すると、その証明書のへパスを含む更新は拒否されます。
  - リスナーの更新が拒否された場合、統計は listener_manager.lds.update_rejected になります。
  - クラスターの更新が拒否された場合、統計は cluster_manager.cds.update_rejected になります。
- *.update_success — App Mesh がプロキシに対して正常に行った設定更新の数。これには、新しい Envoy コンテナの起動時に送信される初期設定ペイロードが含まれます。
  - リスナーの更新が成功した場合、統計は listener_manager.lds.update_success になります。
  - クラスターの更新が成功した場合、統計は cluster_manager.cds.update_success になります。

管理サーバーのメトリクスのセットについては、Envoy のドキュメントの「管理サーバー」を参照してください。

エクスポートされるメトリクス

Envoyは、独自の操作と、インバウンドおよびアウトバウンドトラフィックに関するさまざまなディメンションの両方に関する多くの統計を発行します。Envoy の統計の詳細については、Envoy のドキュメントの「統計」を参照してください。これらのメトリクスは、プロキシの管理ポート 上の /stats エンドポイントを介して利用可能になります (通常は 9901)。

stat プレフィックスは、単一リスナーを使用しているのか、複数のリスナーを使用しているのかによって異なります。その違いを示す例を以下に示します。

> **Warning**
>
> 単一リスナーを複数のリスナー機能に更新すると、以下の表に示す stat プレフィックスが更新され、重大な変更が発生する可能性があります。
> Envoy イメージ 1.22.2.1-prod 以降を使用することをお勧めします。これにより、Prometheus エンドポイントで類似のメトリクス名を確認できます。

単一リスナー (SL)/「ingress」リスナープレフィックスが付いた既存の統計情報	複数のリスナー (ML)/「ingress.<protocol>.<port>」リスナープレフィックスが付いた新しい統計情報	
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	
listener.0.0.0.0_15000.http.*ingress*.downstream_rq_2xx	listener.0.0.0.0_15000.http.*ingress.http.5555*.downstream_rq_2xx 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 http.*ingress.http.6666*.downstream_cx_length_ms	

統計エンドポイントの詳細については、Envoy のドキュメントの「統計エンドポイント」を参照してください。管理者インターフェイスの詳細については、「Envoy プロキシ管理インターフェイスを有効にする」を参照してください。

Amazon EKS での App Mesh の Prometheus

Prometheus はオープンソースのシステムモニタリングおよびアラートツールキットです。その機能の 1 つは、他のシステムが使用できるメトリクスを出力するための形式を指定することです。Prometheus の詳細については、Prometheus のドキュメントの「概要」を参照してください。Envoy は、パラメーター `/stats?format=prometheus` を渡すことで、統計エンドポイントを介してメトリクスを発行できます。

Envoy イメージビルド v1.22.2.1-prod を使用している場合、入力リスナー固有の統計情報を示すために 2 つのディメンションが追加されています。

- `appmesh.listener_protocol`
- `appmesh.listener_port`

Prometheus の既存の統計情報と新しい統計情報の比較です。

- 「ingress」リスナープレフィックスが付いた既存の統計情報

```
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>」が付いた新しい統計情報、Appmesh Envoy イメージ v1.22.2.1-prod 以降

```
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="555
 20
```

- 「ingress.<protocol>.<port>」が付いた新しい統計情報、カスタム Envoy イメージビルド

```
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
```

複数のリスナーの場合、特別なクラスター `cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<ingress_listener_port>_<protocol>_<port>` はリスナー固有になります。

- 「ingress」リスナープレフィックスが付いた既存の統計情報

```
envoy_cluster_assignment_stale{appmesh_mesh="multiple-listeners-
mesh",appmesh_virtual_gateway="tellergateway-vg",Mesh="multiple-listeners-
mesh",VirtualGateway="tellergateway-vg",envoy_cluster_name="cds_ingress_multiple-
listeners-mesh_tellergateway-vg_self_redirect_http_15001"} 0
```

- 「ingress.<protocol>.<port>」が付いた新しい統計情報

```
envoy_cluster_assignment_stale{appmesh_mesh="multiple-
listeners-mesh",appmesh_virtual_gateway="tellergateway-
vg",envoy_cluster_name="cds_ingress_multiple-listeners-mesh_tellergateway-
vg_self_redirect_1111_http_15001"} 0
envoy_cluster_assignment_stale{appmesh_mesh="multiple-
listeners-mesh",appmesh_virtual_gateway="tellergateway-
vg",envoy_cluster_name="cds_ingress_multiple-listeners-mesh_tellergateway-
vg_self_redirect_2222_http_15001"} 0
```

Prometheus のインストール

1. EKS リポジトリを Helm に追加します。

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

2. App Mesh Prometheus のインストール

```
helm upgrade -i appmesh-prometheus eks/appmesh-prometheus \
--namespace appmesh-system
```

Prometheus の例

次は、Prometheus 永続的ストレージ用に `PersistentVolumeClaim` を作成する例です。

```
helm upgrade -i appmesh-prometheus eks/appmesh-prometheus \
--namespace appmesh-system \
--set retention=12h \
--set persistentVolumeClaim.claimName=prometheus
```

Prometheus の使用方法のチュートリアル

- EKS を使用した App Mesh — 可観測性：Prometheus

Amazon EKS を使用した Prometheus と Prometheus について詳しく知るには

- Prometheus のドキュメント
- EKS - Prometheus を使用したプレーンメトリクスのコントロール

App Mesh の CloudWatch

Amazon EKS から CloudWatch に Envoy 統計を出力します。

CloudWatch エージェントをクラスターにインストールし、プロキシからメトリクスのサブセットを収集するように設定できます。Amazon EKS クラスターをまだ作成していない場合は、GitHub の「チュートリアル：Amazon EKS を使用した App Mesh」の手順を使用して作成できます。同じチュートリアルに従って、サンプルアプリケーションをクラスターにインストールできます。

クラスターに適切な IAM アクセス許可を設定し、エージェントをインストールするには、「Prometheus メトリクスコレクションを持つ CloudWatch エージェントをインストールする」の手順に従います。デフォルトのインストールには、Envoy 統計の有用なサブセットを取得する Prometheus スクレイプ設定が含まれています。詳細については、「App Mesh の Prometheus メトリクス」を参照してください。。

エージェントが収集するメトリクスを表示するように設定された App Mesh カスタム CloudWatch ダッシュボードを作成するには、「Prometheus メトリクスの表示」チュートリアルの手順に従います。トラフィックが App Mesh アプリケーションに入ると、グラフに対応するメトリクスが入力され始めます。

CloudWatch のメトリクスのフィルタ処理

App Mesh メトリクス拡張機能には、メッシュで定義したリソースの動作に関するインサイトを提供する便利なメトリクスのサブセットが用意されています。CloudWatch エージェントでは Prometheus メトリクスのスクレイピングがサポートされているため、Envoy からプルして CloudWatch に送信するメトリクスを選択するためのスクレープ設定を提供できます。

Prometheus を使用してメトリクスをスクレイピングする例については、「メトリクス拡張機能」チュートリアルを参照してください。

CloudWatch の例

CloudWatch の設定例は、AWSサンプルリポジトリにあります。

CloudWatch を使用するためのチュートリアル

- App Mesh チュートリアルのモニタリングおよびログ記録機能の追加。
- EKS を使用した App Mesh — 可観測性：CloudWatch
- ECS での App Mesh のメトリクス拡張の使用

App Mesh のメトリクス拡張機能

Envoy は、数百のメトリクスをいくつかの異なる次元に分けて生成します。メトリクスは、App Mesh との関連付け方法においては簡単ではありません。仮想サービスの場合、特定の仮想ノードまたは仮想ゲートウェイと通信している仮想サービスを確実に知るメカニズムはありません。

App Mesh メトリクス拡張により、メッシュで実行される Envoy プロキシが強化されます。この機能拡張により、プロキシは、定義したリソースを認識した追加のメトリクスを出力できます。追加のメトリクスのこの小さなサブセットは、App Mesh で定義したリソースの動作をより詳細に把握するのに役立ちます。

App Mesh メトリクス拡張機能を有効にするには、環境変数 `APPMESH_METRIC_EXTENSION_VERSION` を `1` に設定します。

```
APPMESH_METRIC_EXTENSION_VERSION=1
```

Envoy 設定変数の詳細については、Envoy 設定変数 を参照してください。

インバウンドトラフィックに関連するメトリクス

- **`ActiveConnectionCount`**
  - `envoy.appmesh.ActiveConnectionCount` — アクティブな TCP 接続の数。
  - ディメンション — メッシュ、VirtualNode、VirtualGateWay
- **`NewConnectionCount`**
  - `envoy.appmesh.NewConnectionCount` — TCP 接続の合計数。
  - ディメンション — メッシュ、VirtualNode、VirtualGateWay
- **`ProcessedBytes`**

  - envoy.appmesh.ProcessedBytes — ダウンストリームクライアントとの間で送受信された TCP バイトの合計。
  - ディメンション — メッシュ、VirtualNode、VirtualGateWay
- **RequestCount**
  - envoy.appmesh.RequestCount – 処理された HTTP リクエストの数。
  - ディメンション — メッシュ、VirtualNode、VirtualGateWay
- **GrpcRequestCount**
  - envoy.appmesh.GrpcRequestCount – 処理された gPRC リクエストの数。
  - ディメンション — メッシュ、VirtualNode、VirtualGateWay

アウトバウンドトラフィックに関連するメトリクス

アウトバウンドメトリクスは、仮想ノードからのものか仮想ゲートウェイからのものかに基づいて、さまざまなディメンションが表示されます。

- **TargetProcessedBytes**
  - envoy.appmesh.TargetProcessedBytes — Envoy のアップストリームターゲットとの間で送受信された TCP バイトの合計。
  - ディメンション:
    - 仮想ノードのディメンション — メッシュ、VirtualNode、ターゲット仮想サービス、ターゲット仮想ノード
    - 仮想ゲートウェイのディメンション — メッシュ、VirtualGateWay、TargetVirtualService、TargetVirtualNode
- **HTTPCode_Target_2XX_Count**
  - envoy.appmesh.HTTPCode_Target_2XX_Count — 2xx HTTP レスポンスを発生させた、Envoy のアップストリームターゲットへの HTTP リクエストの数。
  - ディメンション:
    - 仮想ノードのディメンション — メッシュ、VirtualNode、ターゲット仮想サービス、ターゲット仮想ノード
    - 仮想ゲートウェイのディメンション — メッシュ、VirtualGateWay、TargetVirtualService、TargetVirtualNode
- **HTTPCode_Target_3XX_Count**

  - envoy.appmesh.HTTPCode_Target_3XX_Count — 3xx HTTP レスポンスを発生させた、Envoy のアップストリームターゲットへの HTTP リクエストの数。
  - ディメンション:
    - 仮想ノードのディメンション — メッシュ、VirtualNode、ターゲット仮想サービス、ターゲット仮想ノード
    - 仮想ゲートウェイのディメンション — メッシュ、VirtualGateWay、TargetVirtualService、TargetVirtualNode
- **HTTPCode_Target_4XX_Count**
  - envoy.appmesh.HTTPCode_Target_4XX_Count — 4xx HTTP レスポンスを発生させた、Envoy のアップストリームターゲットへの HTTP リクエストの数。
  - ディメンション:
    - 仮想ノードのディメンション — メッシュ、VirtualNode、ターゲット仮想サービス、ターゲット仮想ノード
    - 仮想ゲートウェイのディメンション — メッシュ、VirtualGateWay、TargetVirtualService、TargetVirtualNode
- **HTTPCode_Target_5XX_Count**
  - envoy.appmesh.HTTPCode_Target_5XX_Count — 5xx HTTP レスポンスを発生させた、Envoy のアップストリームターゲットへの HTTP リクエストの数。
  - ディメンション:
    - 仮想ノードのディメンション — メッシュ、VirtualNode、ターゲット仮想サービス、ターゲット仮想ノード
    - 仮想ゲートウェイのディメンション — メッシュ、VirtualGateWay、TargetVirtualService、TargetVirtualNode
- **RequestCountPerTarget**
  - envoy.appmesh.RequestCountPerTarget — Envoy のアップストリームターゲットに送信されたリクエストの数。
  - ディメンション:
    - 仮想ノードのディメンション — メッシュ、VirtualNode、ターゲット仮想サービス、ターゲット仮想ノード
    - 仮想ゲートウェイのディメンション — メッシュ、VirtualGateWay、TargetVirtualService、TargetVirtualNode
- **TargetResponseTime**

- `envoy.appmesh.TargetResponseTime` — Envoy のアップストリームターゲットに対してリクエストが行われた時点から完全なレスポンスが受信されるまでの経過時間。
- ディメンション:
  - 仮想ノードのディメンション — メッシュ、VirtualNode、ターゲット仮想サービス、ターゲット仮想ノード
  - 仮想ゲートウェイのディメンション — メッシュ、VirtualGateWay、argetVirtualService、TargetVirtualNode

App Mesh の Datadog

Datadog は、クラウドアプリケーションのエンドツーエンドの監視、メトリクス、および口グ記録のための監視およびセキュリティアプリケーションです。Datadog は、インフラストラクチャ、アプリケーション、およびサードパーティアプリケーションを完全に監視できるようにします。

Datadog のインストール

- EKS - EKS で Datadog をセットアップするには、Datadog のドキュメントの手順に従ってください。
- ECS EC2 - ECS EC2 で Datadog をセットアップするには、Datadog のドキュメントの手順に従ってください。

Datadog の詳細について

- Datadog のドキュメント

トレース

> **Important**
>
> トレースを完全に実装するには、アプリケーションを更新する必要があります。
> 選択したサービスから利用可能なデータをすべて表示させるには、該当するライブラリを使用してアプリケーションを計測する必要があります。

AWS X-Ray で App Mesh を監視する

AWSX-Ray は、アプリケーションが処理するリクエストから収集されたデータの表示、フィルタリング、インサイトを取得するためのツールを提供するサービスです。これらのインサイトは、問題と機会を特定して、アプリを最適化するのに役立ちます。リクエストとレスポンス、およびアプリケーションが他の AWS サービスに対して行うダウンストリームコールに関する詳細情報を表示できます。

X-Ray は App Mesh と統合して、Envoy マイクロサービスを管理します。Envoy からのトレースデータは、コンテナで実行されている X-Ray デーモンに送信されます。

言語に固有のSDKガイドを使用して、アプリケーションコードにX-Ray を実装します。

App Mesh 使用して X-Ray トレースを有効にする

- サービスのタイプに応じて、次のようになります。
  - ECS - Envoy プロキシコンテナ定義で、`ENABLE_ENVOY_XRAY_TRACING` 環境変数に `1` と`XRAY_DAEMON_PORT`環境変数に `2000` を設定します。
  - EKS - App Mesh コントローラーの設定で、`--set tracing.enabled=true` と `--set tracing.provider=x-ray` を含めます。
- X-Ray コンテナで、ポート `2000` を公開し、ユーザー `1337` として実行します。

X-Ray 例

Amazon ECS の Envoy コンテナの定義

```
      {
        "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 EKS 用の App Mesh コントローラーの更新

```
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 を使用するチュートリアル

- AWSX-Ray によるモニタリング
- Amazon EKS を使用した App Mesh — 可観測性：X-Ray
- AWS App Mesh チュートリアルのX-Ray による分散トレース

AWS X-Ray についての詳細を確認する

- AWSX-Ray ドキュメント

App Mesh を使用した AWS X-Ray のトラブルシューティング

- アプリケーションの AWS X-Rayトレースを表示できません。

Amazon EKS を使用した AppMesh の Jaeger

Jaeger はオープンソースで、エンドツーエンドの分散トレースシステムです。ネットワークのプロファイリングやモニタリングに使用できます。Jaeger は、複雑なクラウドネイティブアプリケーションのトラブルシューティングにも役立ちます。

Jaeger をアプリケーションコードに実装するには、Jaeger のドキュメントで言語固有のガイド「トレースライブラリ」を参照してください。

Helm を使用した Jaeger のインストール

1. EKS リポジトリを Helm に追加します。

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

2. App Mesh Jaeger のインストール

```
helm upgrade -i appmesh-jaeger eks/appmesh-jaeger \
--namespace appmesh-system
```

Jaeger の例

次は、`PersistentVolumeClaim` Jaeger 永続的ストレージ作成の例です。

```
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 を使用するためのチュートリアル

- EKS を使用したApp Mesh — 可観測性：Jaeger

Jaeger の詳細を確認する

- Jaeger のドキュメント

トレースの Datadog

Datadog は、メトリクスだけでなくトレースにも使用できます。詳細とインストール手順については、Datadog のドキュメントのアプリケーション言語に固有のガイドを参照してください。

App Mesh ツール

App Mesh は、次のようなツールを使って間接的に API と対話する機能を提供します。

- AWS CloudFormation
- AWS Cloud Development Kit (AWS CDK)
- Kubernetes 用 App Mesh コントローラー
- Terraform

App Mesh と AWS CloudFormation

AWS CloudFormation は、アプリケーションに必要なすべてのリソースを含むテンプレートを作成し、 AWS CloudFormation がリソースを設定してプロビジョニングするサービスです。また、すべての依存関係を設定するため、リソースの管理よりもアプリケーションに集中することができます。

App Mesh AWS CloudFormation で を使用する方法の詳細と例については、 AWS CloudFormation ドキュメントを参照してください。

App Mesh と AWS CDK

AWS CDK は、コードを使用してクラウドインフラストラクチャを定義し、 を使用して AWS CloudFormation プロビジョニングするための開発フレームワークです。 は TypeScript、、Python JavaScript、Java、C#/ などの複数のプログラミング言語 AWS CDK をサポートしています。Net。

App Mesh AWS CDK で を使用する方法の詳細については、「」のAWS CDK ドキュメントを参照してください。

Kubernetes 用 App Mesh コントローラー

Kubernetes 用の App Mesh コントローラーは、Kubernetes クラスターの App Mesh リソースを管理し、サイドカーをポッドに挿入するのに役立ちます。このコントローラーは特に Amazon EKS で使用するためのもので、Kubernetes にネイティブな方法でリソースを管理できます。

App Mesh コントローラーの詳細については、「App Mesh コントローラーのドキュメント」を参照してください。

App Mesh と Terraform

Terraform は、コードソフトウェアツールとしてのオープンソースインフラストラクチャです。Terraform は CLI を介してクラウドサービスを管理し、宣言型設定ファイルを使用して API と対話します。

Terraform でApp Mesh を使用する方法の詳細については、Terraform ドキュメントをご覧ください。

共有メッシュの使用

AWS Resource Access Manager このサービスを使用して、App Mesh AWS メッシュをアカウント間で共有できます。共有メッシュを使用すると、 AWS 異なるアカウントで作成されたリソースが同じメッシュ内で相互に通信できます。

AWS アカウントは、メッシュリソースの所有者、メッシュコンシューマー、またはその両方になることができます。コンシューマーは、アカウントと共有されるメッシュにリソースを作成できます。オーナーは、アカウントが所有する任意のメッシュにリソースを作成できます。メッシュ所有者は以下のタイプのメッシュコンシューマーとメッシュを共有できます。

- AWS 組織内または組織外の特定のアカウント AWS Organizations
- 組織内の組織単位 AWS Organizations
- その組織全体が AWS Organizations

メッシュ共有の手順については、「クロスアカウントメッシュウォークスルーオン GitHub」を参照してください。 end-to-end

メッシュを共有する権限の付与

アカウント間でメッシュを共有する場合、メッシュを共有する IAM プリンシパルに必要な権限と、メッシュ自体に必要なリソースレベルの権限があります。

メッシュを共有する権限を付与する

IAM プリンシパルがメッシュを共有するには、最低限の権限セットが必要です。IAM プリンシパルが共有メッシュを共有したり使用したりするのに必要な権限を持っていることを確認するために、AWSAppMeshFullAccessAWSResourceAccessManagerFullAccessおよび管理された IAM ポリシーを使用することをお勧めします。

カスタム IAM ポリシーを使用する場合
は、、appmesh:PutMeshPolicyappmesh:GetMeshPolicy、アクションが必要です。appmesh:DeleteMeshPolicyこれらは権限のみの IAM アクションです。IAM プリンシパルにこれらの権限が付与されていない場合、サービスを使用してメッシュを共有しようとするとエラーが発生します。 AWS RAM

AWS Resource Access Manager サービスが IAM を使用する方法の詳細については、ユーザーガイドの「IAM AWS RAM の使用方法」を参照してください。AWS Resource Access Manager

メッシュの権限の付与

共有メッシュには以下の権限があります。

- コンシューマーは、アカウントと共有されているメッシュ内のすべてのリソースを一覧表示して記述できます。
- オーナーは、アカウントが所有するメッシュ内のすべてのリソースを一覧表示して説明できます。
- 所有者とコンシューマーは、アカウントが作成したメッシュのリソースを変更できますが、他のアカウントが作成したリソースを変更することはできません。
- コンシューマーは、アカウントが作成したメッシュ内の任意のリソースを削除できます。
- 所有者は、任意のアカウントが作成したメッシュ内の任意のリソースを削除できます。
- 所有者のリソースは、同じアカウント内の他のリソースのみをリファレンスできます。たとえば、AWS Cloud Map AWS Certificate Manager 仮想ノードは仮想ノードの所有者と同じアカウントにある証明書のみを参照できます。
- 所有者とコンシューマーは、アカウントが所有する仮想ノードとして Envoy プロキシを App Mesh に接続できます。
- 所有者は、仮想ゲートウェイと仮想ゲートウェイルートを作成できます。
- 所有者とコンシューマーは、アカウントが作成したメッシュ内のタグを一覧表示したり、リソースにタグ付け/タグ付け解除したりできます。アカウントが作成したものではないメッシュ内のタグを一覧表示したり、リソースにタグ付け/タグ付け解除したりすることはできません。

共有メッシュはポリシーベースの認証を使用します。メッシュは固定された権限で共有されます。これらの権限を選択してリソースポリシーに追加します。オプションの IAM ポリシーを IAM ユーザー/ロールに基づいて選択することもできます。これらのポリシーで許可されている権限の共通部分から、明示的に拒否された権限を除いたものが、プリンシパルのメッシュへのアクセス権になります。

メッシュを共有すると、 AWS Resource Access Manager `AWSRAMDefaultPermissionAppMesh`サービスはという名前の管理ポリシーを作成し、それを以下の権限を提供するApp Mesh に関連付けます。

- `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

メッシュを共有するための前提条件

メッシュを共有するには、以下の前提条件を満たす必要があります。

- AWS メッシュはアカウントで所有している必要があります。すでに共有されているメッシュを共有することはできません。
- 組織または AWS Organizationsの組織単位とメッシュを共有するには、 AWS Organizationsとの共有を有効にする必要があります。詳細については、「AWS RAM ユーザーガイド」の「AWS Organizationsで共有を有効化する」を参照してください。
- サービスは、相互に通信するメッシュリソースを含むアカウント間で接続を共有しているAmazon VPCにデプロイする必要があります。ネットワーク接続を共有する 1 つの方法は、メッシュで使用するすべてのサービスを共有サブネットにデプロイすることです。詳細および制限事項については、「サブネットの共有」を参照してください。

- サービスは DNS または経由で検出できる必要があります。 AWS Cloud Mapサービスディスカバリの詳細については、「仮想ノード」を参照してください。

関連サービス

メッシュ共有は () と統合されます。 AWS Resource Access Manager AWS RAM AWS RAM は、AWS どのアカウントでも、 AWS AWS Organizationsまたはそれを介してリソースを共有できるようにするサービスです。では AWS RAM、リソース共有を作成して所有しているリソースを共有します。リソース共有は、共有するリソースと、それらを共有するコンシューマーを指定します。コンシューマーは、 AWS 個人アカウント、組織単位、または組織全体の場合があります AWS Organizations。

詳細については AWS RAM、『AWS RAM ユーザーガイド』を参照してください。

メッシュを共有する

メッシュを共有すると、異なるアカウントで作成されたメッシュリソースが同じメッシュ内で相互に通信できるようになります。所有するメッシュのみを共有できます。メッシュを共有するには、メッシュをリソース共有に追加する必要があります。リソース共有は、 AWS RAM AWS アカウント間でリソースを共有できるリソースです。リソース共有では、共有対象のリソースと、共有先のコンシューマーを指定します。Amazon Linux コンソールを使用してメッシュを共有する場合は、既存のリソース共有にそれを追加します。メッシュを新しいリソース共有に追加するには、最初に AWS RAM コンソールを使用してリソース共有を作成する必要があります。

AWS Organizations 組織内の組織に所属していて、組織内での共有が有効になっている場合、組織内のコンシューマーには共有メッシュへのアクセス権が自動的に付与されます。それ以外の場合、コンシューマーはリソース共有に参加するための招待を受け取り、招待を受け入れた後に共有メッシュへのアクセスを許可されます。

所有しているメッシュは、 AWS RAM コンソールまたはを使用して共有できます AWS CLI。

AWS RAM 所有しているメッシュをコンソールを使って共有するには

手順については、「AWS RAM ユーザーガイド」の「リソース共有の作成」を参照してください。リソースタイプを選択するときは、[メッシュ] を選択してから、共有するメッシュを選択します。メッシュがリストされていない場合は、最初にメッシュを作成する必要があります。詳細については、「サービスメッシュの作成」を参照してください。

を使用して所有しているメッシュを共有するには AWS CLI

create-resource-shareコマンドを実行します。`--resource-arns` オプションで、共有するメッシュの ARN を指定します。

メッシュの共有解除

メッシュの共有を解除すると、App Mesh はメッシュの以前のコンシューマーによるメッシュへのそれ以降のアクセスを無効にします。ただし、App Mesh はコンシューマーが作成したリソースを削除しません。メッシュの共有が解除されると、メッシュの所有者のみがリソースにアクセスして削除できます。App Mesh は、メッシュ内のリソースを所有していたアカウントが、メッシュの共有解除後に設定情報を受信しないようにします。また、App Mesh は、メッシュ内にリソースを持つアカウントが、メッシュの共有解除後に設定情報を受信しないようにします。メッシュの所有者のみが共有を解除できます。

所有している共有メッシュの共有を解除するには、リソース共有からメッシュを削除する必要があります。そのためには、 AWS RAM コンソールまたはを使用します AWS CLI。

AWS RAM コンソールを使用して所有している共有メッシュの共有を解除するには

手順については、「AWS RAM ユーザーガイド」の「リソース共有の更新」を参照してください。

を使用して所有している共有メッシュの共有を解除するには AWS CLI

disassociate-resource-shareコマンドを実行します。

共有メッシュの特定

オーナーとコンシューマーは Amazon Linux コンソールを使用して共有メッシュとメッシュリソースを識別できます。 AWS CLI

Amazon Linux コンソールを使用して共有メッシュを識別するには

1. App Mesh コンソールを開きます。https://console.aws.amazon.com/appmesh/。
2. 左のナビゲーションペインで [メッシュ] を選択します。各メッシュのメッシュ所有者のアカウント ID は、[メッシュ所有者]列に一覧表示されます。
3. 左側のナビゲーションから、[仮想サービス]、[仮想ルーター]、または[仮想ノード]を選択します。各リソースのメッシュ所有者とリソース所有者のアカウント ID が表示されます。

を使用して共有メッシュを識別するには AWS CLI

`aws appmesh list-meshes` などの `aws appmesh list resource` コマンドを使用します。このコマンドは、所有しているメッシュと共有されているメッシュを返します。`meshOwner` AWS プロパティにはのアカウント ID が表示され`meshOwner`、`resourceOwner` AWS プロパティにはリソース所有者のアカウント ID が表示されます。メッシュリソースに対してコマンドを実行すると、これらのプロパティが返されます。

共有メッシュにアタッチしたユーザー定義のタグは、ユーザー自身の AWS アカウントでのみ利用可能です。メッシュを共有している他のアカウントでは使用できません。別のアカウントでメッシュの `aws appmesh list-tags-for-resource` コマンドを実行しても、アクセスは拒否されます。

請求と使用量測定

メッシュの共有は無料です。

インスタンスクォータ

メッシュにリソースを作成したユーザーに関係なく、メッシュのすべてのクォータは共有メッシュにも適用されます。メッシュの所有者のみがクォータの増加を要求できます。詳細については、「App Mesh Service Quotas」を参照してください。 AWS Resource Access Manager サービスにもクォータがあります。詳細については、「Service Quotas」を参照してください。

AWS App Mesh と統合されたサービス

App Mesh は、他の AWS サービスと連携することで、ビジネスの課題に対してさらに解決策を提供します。このトピックでは、App Mesh を使用して機能を追加するサービス、または App Mesh を使用してタスクを実行するサービスについて説明します。

目次

AWS CloudFormation を使用した App Mesh リソースの作成

App Mesh は AWS CloudFormation と統合されています。これは、 AWS リソースのモデル化とセットアップを支援するサービスです。これにより、リソースやインフラの作成と管理に費やす時間を短縮することができます。App Mesh メッシュなど、必要な AWS リソースをすべて記述したテンプレートを作成すると、 AWS CloudFormation は、これらのリソースのプロビジョニングと設定を行います。

AWS CloudFormation を使用すると、テンプレートを再利用して、App Mesh リソースをいつでも繰り返し設定できます。リソースを一度記述するだけで、複数の AWS アカウントとリージョンに同じリソースを何度もプロビジョニングすることができます。

App Mesh と AWS CloudFormation テンプレート

App Mesh と関連サービスのリソースをプロビジョニングと設定をするためには、AWS CloudFormation テンプレートについて理解している必要があります。テンプレートは、JSON またはYAMLでフォーマットされたテキストファイルです。これらのテンプレートには、AWS CloudFormation スタックにプロビジョニングしたいリソースを記述します。JSON や YAML に不慣れな方は、AWS CloudFormation Designer を使えば、AWS CloudFormation テンプレートを使いこなすことができます。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS CloudFormation Designer とは?」を参照してください。

App Mesh では、AWS CloudFormation でのメッシュ、ルート、仮想ノード、仮想ルーター、仮想サービスの作成がサポートされています。App Mesh リソースの JSONやYAMLテンプレートの例など、詳細については、「 AWS CloudFormation ユーザーガイド」の「App Meshリソースタイプのリファレンス」を参照してください。

AWS CloudFormation の詳細情報

AWS CloudFormation の詳細については、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

AWS Outposts での App Mesh

AWS Outposts は、オンプレミス施設でのネイティブ AWS サービス、インフラ、運用モデルを可能にします。AWS Outposts 環境では、AWS クラウドで使用しているものと同じ AWS API、ツール、インフラストラクチャを使用できます。AWS Outposts での App Mesh は、オンプレミスのデータやアプリケーションに近接して実行する必要がある低レイテンシーのワークロードに最適です。AWS Outposts の詳細については、「AWS Outposts ユーザーガイド」を参照してください。

前提条件

AWS Outposts で App Mesh を使用するための前提条件は、次のとおりです。

- オンプレミスのデータセンターに Outpost をインストールして設定しておく必要があります。
- Outpost とその AWS リージョンとの間に、信頼できるネットワーク接続が必要です。
- Outpost の AWS リージョンで AWS App Mesh がサポートされている必要があります。サポートされているリージョンのリストについては、「AWS 全般のリファレンス」の「AWS App Mesh のエンドポイントとクォータ」を参照してください。

制限事項

AWS Outposts での App Mesh の使用に関する制限事項を次に示します。

- AWS Identity and Access Management、Application Load Balancer、Network Load Balancer、Classic Load Balancer、Amazon Route 53は、Outpostsではなく AWS リージョンで実行されます。これにより、これらのサービスとコンテナ間のレイテンシーが増加します。

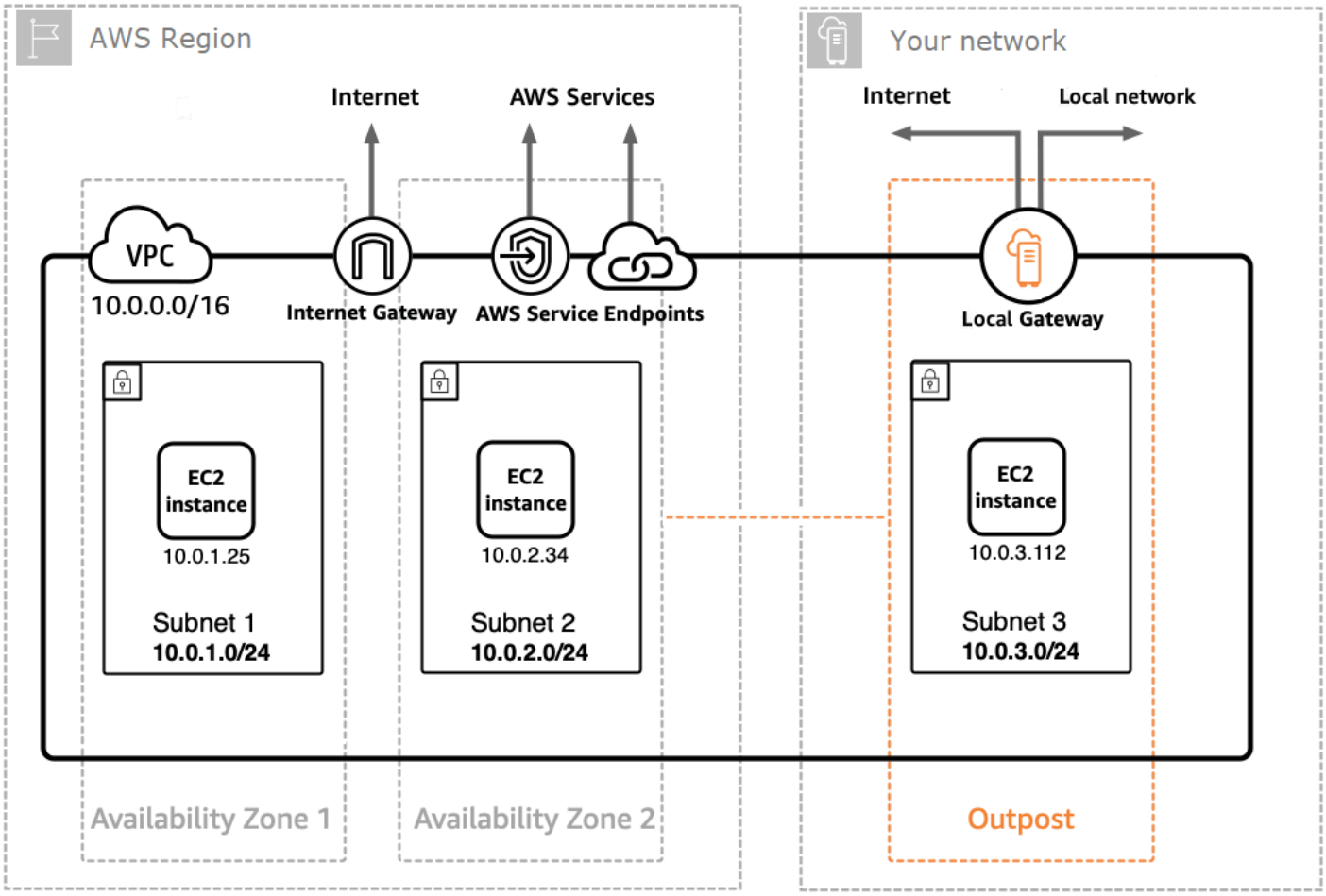
ネットワーク接続に関する考慮事項

次に記載するのは、Amazon EKS AWS Outposts のネットワーク接続に関する考慮事項です。

- Outpost とその AWS リージョン間のネットワーク接続が失われた場合でも、App Mesh Envoy プロキシは実行され続けます。ただし、接続が回復するまで、サービスメッシュを変更することはできません。
- Outpost と AWS リージョン間では、信頼性が高く、可用性が高い、低レイテンシーの接続をするようお勧めします。

Outpost でのApp Mesh Envoy プロキシの作成

Outpost は AWS リージョンの拡張であり、アカウント内の Amazon VPC を複数のアベイラビリティーゾーンと関連する任意の Outpost ロケーションにまたがるように拡張できます。Outpost を設定するとき、サブネットをそれに関連付けて、リージョン VPC 環境をオンプレミス施設に拡張します。Outpost のインスタンスは、サブネットが関連付けられたアベイラビリティーゾーンと同様に、リージョン VPC の一部として表示されます。



Outpost 上に App Mesh Envoy のプロキシを作成するには、Outpost上で動作している Amazon ECS タスクまたは Amazon EKS ポッドに App Mesh Envoy コンテナイメージを追加してください。詳しくは、「Amazon Elastic Container Service Developer Guide」の「Amazon Elastic Container Service on AWS Outposts」、「Amazon EKS User Guide」の「Amazon Elastic Kubernetes Service AWS Outposts」をご参照ください。

App Mesh のベストプラクティス

計画されたデプロイ中および一部のホストの予期しない損失時に、失敗したリクエストをゼロにするという目標を達成するために、このトピックのベストプラクティスでは、次の戦略を実装します。

- 安全なデフォルトの再試行戦略を使用することで、アプリケーションの観点からリクエストが成功する可能性が高くなります。詳細については、「再試行ですべてのルートを計測する」を参照してください。
- 再試行されたリクエストが実際の送信先に送信される可能性を最大化することで、再試行されたリクエストが成功する可能性が高くなります。詳細については、デプロイ速度の調整、スケールインする前にスケールアウトする、およびコンテナのヘルスチェックを実装するを参照してください。

障害を大幅に削減または排除するには、次のすべてのプラクティスでレコメンデーションを実装することをお勧めします。

再試行ですべてのルートを計測する

すべての仮想サービスが仮想ルーターを使用するように設定し、すべてのルートに対してデフォルトの再試行ポリシーを設定します。これにより、ホストを再選択して新しいリクエストを送信することで、リクエストの失敗が軽減されます。再試行ポリシーでは、「maxRetries」に2つ以上の値を指定し、再試行イベントタイプをサポートする各ルートタイプで、再試行イベントタイプごとに次のオプションを指定することを推奨します。

- TCP – connection-error
- HTTP と HTTP/2 – stream-error と gateway-error
- gRPC – cancelled と unavailable

その他の再試行イベントは、リクエストがべき等でない場合など、安全ではない可能性があるため、 case-by-case ベースで検討する必要があります。リクエストの最大レイテンシー(maxRetries * perRetryTimeout)と、より多くの再試行により増えた成功率との間の適切なトレードオフを行うために、maxRetries と perRetryTimeoutの値を検討しテストする必要があります。さらに、Envoyが存在しないエンドポイントに接続しようとする場合、そのリクエストが完全にperRetryTimeoutを消費することを予想する必要があります。再試行ポリシーを設定するには、「ルートを作成する」を参照し、次に、ルートしたいプロトコルを選択します。

> **Note**
>
> 2020年7月29日以降にルートを実装し、再試行ポリシーを指定していない場合、App Meshは、2020年7月29日以降に作成した各ルートに対して、以前のポリシーと同様のデフォルトの再試行ポリシーを自動的に作成している可能性があります。詳細については、「デフォルトのルート再試行ポリシー」を参照してください。

デプロイ速度の調整

ローリングデプロイを使用する場合は、デプロイ全体の速度を下げます。デフォルトでは、Amazon ECSは最低100%の正常なタスクおよび総タスクは200 パーセントのデプロイ戦略を設定します。これによりデプロイでは、2箇所に高いドリフトが発生します。

- 新しいタスクの100%のフリートサイズが、リクエストを完了する前にEnvoyに表示される場合があります (軽減については「コンテナのヘルスチェックを実装する」を参照してください)。
- 古いタスクの100%フリートサイズは、タスクが終了している間にEnvoyに表示される場合があります。

このデプロイ制約で設定されていると、コンテナのオーケストレータは、古い送信先をすべて同時に非表示にし、新しい送信先をすべて表示できる状態になる場合があります。Envoyデータプレーンは結果整合性があるため、データプレーンに表示される一連の送信先がオーケストレーターの視点とは異なる可能性があります。これを軽減するには、最低でも100%の正常なタスクを維持しながら、総タスクを125%に下げることを推奨します。これにより、発散が減少し、再試行の信頼性が向上します。コンテナのランタイムごとに、次の設定を推奨します。

Amazon ECS

サービスに必要な数が2または3の場合は、`maximumPercent`を150パーセントに設定します。それ以外の場合は、「`maximumPercent`」を125パーセントに設定します。

Kubernetes

デプロイの `update strategy` を設定し、`maxUnavailable` を 0 パーセント、`maxSurge` を 25 パーセントに設定します。詳細については、Kubernetes ドキュメントの「Deployments」を参照してください。

スケールインする前にスケールアウトする

スケールアウトとスケールインのどちらも、ある程度の確率でリクエストの再試行に失敗するという結果を招くおそれがあります。スケールアウトを軽減するタスクのレコメンデーションがありますが、スケールインに関する唯一のレコメンデーションは、一度にスケールインするタスクの割合を最小限に抑えることです。古いタスクまたはデプロイをスケーリングインする前に、新しい Amazon ECSタスクまたはKubernetesデプロイをスケールアウトするデプロイ戦略を使用することを推奨します。このスケーリング戦略により、同じ速度を維持しながら、タスクまたはデプロイでのスケーリングの割合を低く抑えることができます。このプラクティスは、Amazon ECSタスクとKubernetesデプロイの両方に適用されます。

コンテナのヘルスチェックを実装する

スケールアップのシナリオでは、Amazon ECS タスク内のコンテナに故障が発生し、最初は応答しないことがあります。コンテナのランタイムごとに、次の提案を推奨します。

Amazon ECS

これを軽減するために、コンテナのヘルスチェックとコンテナの依存関係の順序付けを使用して、送信ネットワーク接続を必要とするコンテナが起動する前に、Envoyが実行されていて正常であることの確認を推奨します。タスク定義でアプリケーションコンテナと Envoy コンテナを正しく設定するには、「コンテナの依存関係」を参照してください。

Kubernetes

Kubernetes ライブネスプローブと準備状況プローブは、Kubernetes 用 App Mesh コントローラーのAWS Cloud Map インスタンスの登録と登録解除では考慮されていないため、なし。詳細については、 GitHub 「問題 #132」を参照してください。

DNS 解決の最適化

サービス検出に DNS を使用している場合は、メッシュを設定するときに DNS 解決を最適化するために、適切な IP プロトコルを選択することが重要です。App Mesh は IPv4と の両方をサポートしIPv6、選択するとサービスのパフォーマンスと互換性に影響を与える可能性があります。インフラストラクチャが をサポートしていない場合はIPv6、デフォルトのIPv6_PREFERRED動作に依存するのではなく、インフラストラクチャに合った IP 設定を指定することをお勧めします。デフォルトのIPv6_PREFERRED動作では、サービスのパフォーマンスが低下する可能性があります。

- IPv6_PREFERRED – これはデフォルト設定です。Envoy は最初に IPv6 アドレスの DNS ルックアップを実行し、IPv6アドレスが見つからないIPv4場合は にフォールバックします。これは、インフラストラクチャが主に をサポートしているIPv6がIPv4、互換性が必要な場合に役立ちます。
- IPv4_PREFERRED – Envoy は最初にIPv4アドレスを検索し、使用可能なIPv4アドレスIPv6がない場合は にフォールバックします。インフラストラクチャが主に をサポートしているIPv4が、ある程度IPv6の互換性がある場合は、この設定を使用します。
- IPv6_ONLY – サービスがIPv6トラフィックのみをサポートしている場合は、このオプションを選択します。Envoy はIPv6アドレスの DNS ルックアップのみを実行し、すべてのトラフィックが経由でルーティングされるようにしますIPv6。
- IPv4_ONLY – サービスがIPv4トラフィックのみをサポートしている場合は、この設定を選択します。Envoy はIPv4アドレスの DNS ルックアップのみを実行し、すべてのトラフィックが 経由でルーティングされるようにしますIPv4。

IP バージョン設定は、メッシュレベルと仮想ノードレベルの両方で設定でき、仮想ノード設定はメッシュレベルで上書きされます。

詳細については、「Service Meshes と仮想ノード」を参照してください。

のセキュリティ AWS App Mesh

のクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ — クラウドで AWS サービスを実行するインフラストラクチャを保護する責任 AWS は にあります AWS 。 AWS また、 では、安全に使用できるサービスも提供しています。AWS コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。 AWS App Meshに適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲内のAWS のサービス」を参照してください。App Mesh は、TLS 証明書のシークレットキーなどの機密情報を含むローカルプロキシに設定を安全に配信する責任があります。
- クラウドのセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、次のようなその他の要因についても責任を負います。
  - データの機密性、企業の要件、および適用される法律と規制。
  - トラフィックが VPC 内のサービス間を通過できるようにするセキュリティグループの設定など、App Mesh データプレーンのセキュリティ設定。
  - App Mesh に関連付けられているコンピューティングリソースの設定。
  - コンピューティングリソースに関連付けられた IAM ポリシーと、それらが App Mesh コントロールプレーンから取得できる設定。

このドキュメントは、App Mesh を使用するときに責任共有モデルを適用する方法を理解するのに役立ちます。次のトピックでは、セキュリティおよびコンプライアンスの目標を達成するために App Mesh を設定する方法を説明します。また、App Mesh リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

App Mesh セキュリティ理念

> お客様は、必要な範囲でセキュリティを調整できるようにする必要があります。プラットフォームが安全性の向上を妨げるものであってはなりません。プラットフォームの機能の安全性はデフォルトで設定されています。

トピック

Transport Layer Security (TLS)

App Mesh では、Transport Layer Security (TLS) は、仮想ノード や 仮想ゲートウェイ などのメッシュエンドポイントによって App Mesh で表される、コンピューティングリソースにデプロイされた Envoy プロキシ間の通信を暗号化します。プロキシは TLS をネゴシエートして終了します。プロキシがアプリケーションと一緒にデプロイされる場合、アプリケーションコードには TLS セッションをネゴシエートする役割はありません。プロキシが、アプリケーションに代わって TLS をネゴシエートします。

App Mesh では、次の方法で TLS 証明書をプロキシに提供できます。

- () によって発行される AWS Certificate Manager (ACM) AWS Private Certificate Authority からのプライベート証明書AWS Private CA。
- 独自の認証局 (CA) によって発行された仮想ノードのローカルファイルシステムに保存されている証明書
- ローカルの Unix ドメインソケットを介して Secrets Discovery Service (SDS) エンドポイントによって提供される証明書。

Envoy プロキシの認可 は、メッシュエンドポイントで表されるデプロイ済みの Envoy プロキシで有効にする必要があります。プロキシ認証を有効にする場合は、暗号化を有効にするメッシュエンドポイントのみへのアクセスに制限するようお勧めします。

証明書の要件

証明書のサブジェクト代替名 (SAN) の 1 つが、メッシュエンドポイントによって表される実際のサービスの検出方法に応じて、特定の基準に一致する必要があります。

- DNS — 証明書 SAN の 1 つが、DNS サービスディスカバリ設定で指定された値と一致する必要があります。*mesh-endpoint.apps.local* というサービスディスカバリ名を持つアプリケーションの場合、その名前と一致する証明書を作成するか、ワイルドカード *.*apps.local* を使用して証明書を作成することができます。
- AWS Cloud Map – 証明書 SANs の 1 つが、 形式を使用して AWS Cloud Map サービス検出設定で指定された値と一致する必要があります*service-name.namespace-name*。serviceName *mesh-endpoint*と namespaceName AWS Cloud Map のサービス検出設定を持つアプリケーションの場合*apps.local*、名前 に一致する証明書*mesh-endpoint.apps.local*、またはワイルドカードを含む証明書を作成できます。 *.*apps.local*.

どちらの検出メカニズムでも、DNS サービスディスカバリ設定に一致する証明書 SAN がない場合、クライアントの Envoy から見て、Envoys 間の接続は、次のエラーメッセージで失敗します。

```
TLS error: 268435581:SSL routines:OPENSSL_internal:CERTIFICATE_VERIFY_FAILED
```

TLS 認証証明書

App Mesh では、TLS 認証を使用する場合、証明書の複数のソースがサポートされます。

AWS Private CA

証明書は、証明書を使用するメッシュエンドポイントと同じリージョンおよび AWS アカウントの ACM に保存する必要があります。CA の証明書は、同じ AWS アカウントに存在する必要はありませんが、メッシュエンドポイントと同じ リージョンに存在する必要があります。がない場合は AWS Private CA、証明書をリクエストする前に作成する必要があります。ACM AWS Private CA を使用して既存の から証明書をリクエストする方法の詳細については、「プライベート証明書のリクエスト」を参照してください。証明書をパブリック証明書にすることはできません。

TLS クライアントポリシーに使用するプライベート CA は、ルートユーザー CA である必要があります。

からの証明書と CAs を使用して仮想ノードを設定するには AWS Private CA、App Mesh の呼び出しに使用するプリンシパル (ユーザーやロールなど) に次の IAM アクセス許可が必要です。

- リスナーの TLS 設定に追加する証明書では、プリンシパルに `acm:DescribeCertificate` のアクセス許可が必要です。
- TLS クライアントポリシーで設定された CA の場合は、プリンシパルに `acm-pca:DescribeCertificateAuthority` のアクセス許可が必要です。

> **Important**
>
> CA を他のアカウントと共有すると、それらのアカウントに意図しない CA への特権が与えられる可能性があります。リソースベースのポリシーを使用して、CA から証明書を発行する必要のないアカウントに対しては `acm-pca:DescribeCertificateAuthority` と `acm-pca:GetCertificateAuthorityCertificate` のみへのアクセスに制限するようお勧めします。

これらのアクセス許可は、プリンシパルにアタッチされている既存の IAM ポリシーに追加するか、新しいプリンシパルとポリシーを作成してポリシーをプリンシパルにアタッチできます。詳細については、「IAM ポリシーの編集」、「IAM ポリシーの作成」、「IAM ID アクセス許可の追加」を参照してください。

> **Note**
>
> 各 のオペレーションは、削除する AWS Private CA まで月額料金が発生します。また、毎月発行するプライベート証明書とエクスポートするプライベート証明書についても料金が発生します。詳細については、「AWS Certificate Manager 料金」を参照してください。

メッシュエンドポイントが表す Envoy プロキシのプロキシ認証を有効にする場合、使用する IAM ロールに、次の IAM アクセス許可を割り当てる必要があります。

- 仮想ノードのリスナーに設定された証明書の場合、ロールには `acm:ExportCertificate` のアクセス許可が必要です。
- TLS クライアントポリシーで設定されている CA の場合、ロールには`acm-pca:GetCertificateAuthorityCertificate` のアクセス許可が必要です。

ファイルシステム

ファイルシステムを使用して Envoy に証明書を配信できます。これを行うには、証明書チェーンとこれに対応するシークレットキーをファイルパスで使用できるようにします。そうすれば、これらのリソースは Envoy サイドカープロキシから利用可能です。

Envoy Secret Discovery Service (SDS)

Envoy は、Secrets Discovery プロトコルを使用して、特定のエンドポイントから TLS 証明書などの機密情報を取得します。このプロトコルの詳細については、Envoy の SDS ドキュメントを参照してください。

App Mesh は、SDS (Secret Discovery Service) が証明書および証明書チェーンのソースとして機能する場合、プロキシに対してローカルな Unix ドメインソケットを使用して SDS エンドポイントとして機能するように Envoy プロキシを設定します。`APPMESH_SDS_SOCKET_PATH` 環境変数を使用して、このエンドポイントへのパスを設定できます。

> **Important**
>
> Unix ドメインソケットを使用した Local Secrets Discovery Service は、App Mesh Envoy プロキシのバージョン 1.15.1.0 以降でサポートされています。
> App Mesh では、gRPC を使用して V2 SDS プロトコルがサポートされています。

SPIFFE ランタイム環境 (SPIRE) との統合

SPIFFE ランタイム環境 (SPIRE) などの既存のツールチェーンを含む、SDS API の任意のサイドカー実装を使用できます。SPIRE は、分散システムの複数のワークロード間で相互 TLS 認証をデプロイできるように設計されています。実行時にワークロードのアイデンティティを証明します。また、SPIRE は、ワークロード固有で一時的に使えて自動的にローテーションするキーと証明書をワークロードに直接送信します。

SPIRE エージェントを Envoy の SDS プロバイダーとして設定する必要があります。相互 TLS 認証を提供するために必要なキーマテリアルを Envoy に直接提供できるようにします。Envoy プロキシの横にあるサイドカーで SPIRE エージェントを実行します。エージェントは、必要に応じて一時的に使えるキーおよび証明書を再生成します。エージェントは、Envoy を証明し、Envoy が SPIRE エージェントによって公開される SDS サーバーに接続するときに、Envoy がどのサービスアイデンティティと CA 証明書を使用できるようにするかを決定します。

このプロセス中に、サービスアイデンティティ と CA 証明書がローテーションされ、更新情報が Envoy にストリーミングされます。Envoy は、その情報を中断やダウンタイムもなく、プライベートキーがファイルシステムに触れることなく、新しい接続にすぐに適用します。

TLS をネゴシエートするための App Mesh による Envoys の設定方法

App Mesh は、メッシュ内にある Envoys 間の通信の設定方法を決定する際に、クライアントとサーバーの両方のメッシュエンドポイント設定を使用します。

クライアントポリシーを使用する場合

クライアントポリシーで TLS の使用が適用され、クライアントポリシー内のポートの 1 つがサーバーのポリシーのポートと一致する場合、クライアントポリシーを使用してクライアントの TLS 検証コンテキストを設定します。例えば、仮想ゲートウェイのクライアントポリシーが仮想ノードのサーバーポリシーと一致する場合、TLS ネゴシエーションは、仮想ゲートウェイのクライアントポリシーで定義された設定を使用して、プロキシ間で試行されます。クライアントポリシーがサーバーのポリシーのポートと一致しない場合、サーバーポリシーの TLS 設定に応じて、プロキシ間の TLS がネゴシエートされる場合とそうでない場合があります。

クライアントポリシーを使用しない場合

クライアントがクライアントポリシーを設定していない場合、またはクライアントポリシーがサーバーのポートと一致しない場合、App Mesh はサーバーを使用して、クライアントから TLS をネゴシエートするかどうか、また、その方法を判断します。例えば、仮想ゲートウェイでクライアントポリシーが指定されておらず、仮想ノードが TLS 終了を設定していない場合、TLS はプロキシ間でネゴシエートされません。クライアントが一致するクライアントポリシーを指定しておらず、サーバーが TLS モード `STRICT` または `PERMISSIVE` で設定されている場合、プロキシは TLS をネゴシエートするように設定されます。TLS 終了時の証明書の提供方法に応じて、次の追加の動作が適用されます。

- ACM 管理の TLS 証明書 — サーバーが ACM 管理された証明書を使用して TLS 終了を設定した場合、App Mesh は、TLS をネゴシエートし、証明書がチェーンアップするルートユーザー CA に対して証明書を検証するようにクライアントを自動的に設定します。
- ファイルベースの TLS 証明書 — サーバーがプロキシのローカルファイルシステムからの証明書を使用して TLS 終了を設定すると、App Mesh は TLS をネゴシエートするようにクライアントを自動的に設定しますが、サーバーの証明書は検証されません。

サブジェクトの別名

オプションで、信頼するサブジェクトの別名 (SAN) のリストを指定することもできます。SAN は FQDN または URI 形式である必要があります。SAN が提供されている場合、Envoy は、提示された証明書のサブジェクトの別名がこのリストのいずれかの名前と一致することを確認します。

終端側のメッシュエンドポイントで SAN を指定しない場合、そのノードの Envoy プロキシはピアクライアント証明書の SAN を検証しません。発信元のメッシュエンドポイントで SAN を指定しない場合、終端側のエンドポイントによって提供される証明書の SAN は、メッシュエンドポイントのサービスディスカバリ設定と一致する必要があります。

詳細については、App Mesh の「TLS: 証明書の要件」を参照してください。

> **Important**
>
> TLS のクライアントポリシーが `not enforced` に設定されている場合にのみ、ワイルドカード SAN を使用できます。クライアント仮想ノードまたは仮想ゲートウェイのクライアントポリシーが TLS を適用するように構成されている場合、ワイルドカード SAN を受け入れることはできません。

暗号化の検証

TLS を有効にしたら、Envoy プロキシにクエリを送信して、通信が暗号化されていることを確認できます。Envoy プロキシは、TLS 通信が正常に機能しているかどうかを理解するのに役立つリソースの統計情報を出力します。例えば、Envoy プロキシは、指定されたメッシュエンドポイントに対して正常にネゴシエートした TLS ハンドシェイクの数に関する統計情報を記録します。次のコマンドを実行して、*my-mesh-endpoint* という名前のメッシュエンドポイントで成功した TLS ハンドシェイクの数を特定します。

```
curl -s 'http://my-mesh-endpoint.apps.local:9901/stats' | grep ssl.handshake
```

次の返された出力例では、メッシュエンドポイントに対して 3 つのハンドシェイクがあったため、通信は暗号化されています。

```
listener.0.0.0.0_15000.ssl.handshake: 3
```

Envoy プロキシは、TLS ネゴシエーションが失敗したときにも統計情報を送信します。メッシュエンドポイントに TLS エラーがあったかどうかを確認します。

```
curl -s 'http://my-mesh-endpoint.apps.local:9901/stats' | grep -e "ssl.*\(fail\|error\)"
```

返された出力例では、複数の統計情報でエラーがゼロだったため、TLS ネゴシエーションは成功しています。

```
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.ssl.fail_verify_san: 0
```

Envoy TLS 統計の詳細については、「Envoy リスナー統計情報」を参照してください。

証明書の更新

AWS Private CA

ACM を使用して証明書を更新すると、更新が完了してから35分以内に接続されているプロキシに更新された証明書が自動的に配信されます。マネージド型更新を使用して、有効期間の期限に近づいた証明書を自動的に更新するようお勧めします。詳細については、「 ユーザーガイド」の「ACM の Amazon 発行証明書のマネージド更新」を参照してください。 AWS Certificate Manager

独自の証明書を使用する

ローカルファイルシステムからの証明書を使用する場合、Envoy は、証明書が変更されても自動的に再ロードしません。Envoy プロセスを再起動するか、再デプロイして、新しい証明書をロードできます。新しい証明書を別のファイルパスに配置し、そのファイルパスで仮想ノードまたはゲートウェイ設定を更新することもできます。

で TLS 認証を使用するように Amazon ECS ワークロードを設定する AWS App Mesh

メッシュを設定すると、TLS 認証を使用できます。ワークロードに追加する Envoy プロキシサイドカーで証明書が使用可能であることを確認します。EBS ボリュームまたは EFS ボリュームを Envoy

サイドカーにアタッチすることも、証明書を保存したり、 AWS Secrets Manager から取得することもできます。

- ファイルベースの証明書のディストリビューションを使用する場合は、EBS ボリュームまたは EFS ボリュームを Envoy サイドカーにアタッチします。証明書とプライベートキーへのパスが、で設定されているパスと一致していることを確認します AWS App Mesh。
- SDS ベースのディストリビューションを使用している場合は、証明書へのアクセス権を持つ Envoy の SDS API を実装するサイドカーを追加します。

> **Note**
>
> SPIRE は Amazon ECS ではサポートされません。

で TLS 認証を使用するように Kubernetes ワークロードを設定する AWS App Mesh

AWS App Mesh Controller for Kubernetes を設定して、仮想ノードと仮想ゲートウェイサービスのバックエンドとリスナーの TLS 認証を有効にできます。ワークロードに追加する Envoy プロキシサイドカーで証明書が使用可能であることを確認します。相互 TLS 認証のチュートリアルセクションで、各ディストリビューションタイプの例を確認できます。

- ファイルベースの証明書のディストリビューションを使用する場合は、EBS ボリュームまたは EFS ボリュームを Envoy サイドカーにアタッチします。証明書とシークレットキーへパスが、コントローラーで設定されているパスと一致していることを確認します。または、ファイルシステム上にマウントされている Kubernetes Secret を使用することもできます。
- SDS ベースのディストリビューションを使用する場合は、Envoy の SDS API を実装するノードローカル SDS プロバイダーを設定する必要があります。Envoy は UDS を介して到達します。EKS AppMesh コントローラーで SDS ベースの mTLS サポートを有効にするには、`enable-sds`フラグを に設定`true`し、ローカル SDS プロバイダーの UDS パスを `sds-uds-path`フラグ経由でコントローラーに提供します。Helm を使用する場合は、コントローラーインストールの一部としてこれらを設定します。

```
--set sds.enabled=true
```

> **Note**
>
> Fargate モードで Amazon Elastic Kubernetes Service (Amazon EKS) を使用している場合は、SPIRE を使用して証明書を配信することはできません。

相互 TLS 認証

相互 TLS (Transport Layer Security) 認証は、TLSのオプションコンポーネントで、双方向のピア認証を提供します。相互 TLS 認証は、TLS 上にセキュリティレイヤーを追加し、お客様のサービスが接続を行うクライアントを確認することを可能にします。

クライアントとサーバーの関係にあるクライアントは、セッションネゴシエーションプロセス中に X.509 証明書も提供します。サーバーは、この証明書を使用してクライアントを識別し、認証します。このプロセスは、証明書が信頼できる認証局 (CA) によって発行されたかどうか、また、証明書が有効な証明書であるかどうかを確認するのに役立ちます。また、証明書のサブジェクト別名 (SAN) を使用してクライアントを識別します。

がサポートするすべてのプロトコルで相互 TLS 認証を有効にできます。 AWS App Mesh TCP、HTTP/1.1、HTTP/2、gRPC があります。

> **Note**
>
> App Mesh を使用して、サービスからの Envoy プロキシ間の通信に対して相互 TLS 認証を設定できます。ただし、アプリケーションとEnvoy プロキシ間の通信は暗号化されません。

相互 TLS 認証証明書

AWS App Mesh 相互 TLS 認証に使用できる 2 つの証明書ソースをサポートします。TLS クライアントポリシーのクライアント証明書とリスナー TLS 設定でのサーバー検証を、次のものからソースできます。

- ファイルシステム—実行中の Envoy プロキシのローカルファイルシステムからの証明書。Envoy に証明書を配布するには、App Mesh API に証明書チェーンとシークレットキーのファイルパスを指定する必要があります。

- Envoy のシークレット・ディスカバリー・サービス (SDS) — SDS を実装し、証明書を Envoy に送信できるようにする B ring-your-own サイドカー。それらには、SPIFFE ランタイム環境 (SPIRE)が含まれます。

> **Important**
>
> App Mesh は、相互 TLS 認証に使用される証明書またはシークレットキーを保存しません。代わりに、Envoy が、それらをメモリに保存します。

メッシュエンドポイントの設定

仮想ノードやゲートウェイなど、メッシュエンドポイントの相互 TLS 認証を設定します。これらのエンドポイントは、証明書を提供し、信頼できる権限を指定します。

これを行うには、クライアントとサーバーの両方に X.509 証明書をプロビジョニングし、TLS 終了と TLS 発信の両方の検証コンテキストで信頼できる機関証明書を明示的に定義する必要があります。

メッシュの内部を信頼する

サーバー側の証明書は仮想ノードリスナー (TLS 終了) で設定され、クライアント側の証明書は仮想ノードサービスバックエンド (TLS 発信) で構成されます。この設定の代わりに、仮想ノードのすべてのサービスバックエンドに対してデフォルトのクライアントポリシーを定義し、必要に応じて特定のバックエンドに対してこのポリシーを上書きできます。仮想ゲートウェイは、そのすべてのバックエンドに適用されるデフォルトのクライアントポリシーでのみ設定できます。

両方のメッシュの仮想ゲートウェイでインバウンドトラフィックの相互 TLS 認証を有効にすることで、異なるメッシュ間で信頼を設定できます。

メッシュの外側を信頼する

TLS 終了の仮想ゲートウェイリスナーでサーバー側の証明書を指定します。仮想ゲートウェイと通信する外部サービスを設定して、クライアント側の証明書を提示します。証明書は、サーバー側の証明書が TLS 発信の仮想ゲートウェイリスナーで使用するのと同じ認定権限 (CA) の 1 つから取得する必要があります。

相互 TLS 認証にサービスを移行する

App Mesh内の既存のサービスを相互 TLS 認証に移行する場合は、これらのガイドラインに従って接続を維持してください。

プレーンテキストで通信するサービスを移行する

1. サーバーエンドポイントの TLS 設定の `PERMISSIVE` モードを有効にします。このモードでは、プレーンテキストトラフィックがエンドポイントに接続できるようになります。
2. サーバーの相互 TLS 認証を設定し、サーバー証明書、トラストチェーン、およびオプションで信頼できる SAN を指定します。
3. TLS 接続を介して通信が行われていることを確認します。
4. クライアントで相互 TLS 認証を設定し、クライアント証明書、トラストチェーン、およびオプションで信頼できるSANを指定します。
5. サーバーの TLS 設定に `STRICT` モードを有効にします。

TLS 経由で通信するサービスの移行

1. クライアントで相互 TLS 設定をし、クライアント証明書、そしてオプションで信頼された SAN を指定します。クライアント証明書は、バックエンドサーバーが要求するまでバックエンドに送信されません。
2. サーバーで相互 TLS 設定をし、トラストチェーン、およびオプションで信頼された SAN を指定します。このため、サーバーは、クライアント証明書をリクエストします。

相互 TLS 認証の検証

Envoy が具体的にどのように TLS 関連の統計を出すかは、「Transport Layer Security: 暗号化の検証」ドキュメントを参照してください。相互 TLS 認証の場合は、次の統計情報を調べる必要があります。

- `ssl.handshake`
- `ssl.no_certificate`
- `ssl.fail_verify_no_cert`
- `ssl.fail_verify_san`

次の 2 つの統計例は、仮想ノードに正常に終了する TLS 接続が、すべて証明書を提供したクライアントから発信されていることを示しています。

```
listener.0.0.0.0_15000.ssl.handshake: 3
```

```
listener.0.0.0.0_15000.ssl.no_certificate: 0
```

次の統計例は、仮想クライアントノード (またはゲートウェイ) からバックエンドの仮想ノードへの接続に失敗したことを表しています。サーバー証明書に記載されているサブジェクト代替名 (SAN) が、クライアントが信頼する SAN のいずれとも一致していません。

```
cluster.cds_egress_my-mesh_my-backend-node_http_9080.ssl.fail_verify_san: 5
```

App Mesh 相互 TLS 認証のウォークスルー

- 相互 TLS 認証のウォークスルー : このウォークスルーでは、App Mesh CLIを使用して、相互 TLS 認証によるカラーアプリを構築する方法について説明します。
- Amazon EKS 相互TLS SDS ベースのウォークスルー:このウォークスルーでは、Amazon EKSと SPIFFE ランタイム環境 (SPIRE) で、相互 TLS SDS ベースの認証を使用する方法を紹介します。
- Amazon EKS 相互 TLS ファイルベースのウォークスルー: このウォークスルーでは、Amazon EKS と SPIFFE ランタイム環境 (SPIRE) で、相互TLSファイルベース認証を使用する方法を紹介します。

が IAM と AWS App Mesh 連携する方法

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、App Mesh リソースを使用するための認証 (サインイン)、認可 (アクセス許可を持つ) できるユーザーを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

- AWS App Mesh アイデンティティベースのポリシーの例
- AWS App Mesh の マネージドポリシー
- App Mesh のサービスリンクロールの使用
- Envoy プロキシの認可
- AWS App Mesh ID とアクセスのトラブルシューティング

対象者

AWS Identity and Access Management (IAM) の使用方法は、App Mesh で行う作業によって異なります。

サービスユーザー - App Mesh サービスを使用してジョブを実行する場合は、管理者が必要な認証情報とアクセス許可を用意します。より多くの App Mesh 機能を使用して作業を行うと、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役立ちます。App Mesh の機能にアクセスできない場合は、「AWS App Mesh ID とアクセスのトラブルシューティング」を参照してください。

サービス管理者 - 社内の App Mesh リソースを担当している場合は、App Mesh へのフルアクセス権を保有しているはずです。サービスのユーザーがどの App Mesh 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。App Mesh による IAM の活用方法について詳しくは、「が IAM と AWS App Mesh 連携する方法」を参照してください。

IAM 管理者 - IAM 管理者は、App Mesh へのアクセスを管理するポリシーを作成する方法の詳細を知りたい場合があります。IAM で使用できる App Mesh アイデンティティベースのポリシーの例を表示するには、「AWS App Mesh アイデンティティベースのポリシーの例」を参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証（ にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレー

ティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させることをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS のサービス完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、IAM ユーザーガイドの「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイド の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「Creating a role for a third-party Identity Provider」(サードパーティーアイデンティティプロバイダー向けロールの作成) を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。

クロスアカウントアクセスのロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントリソースアクセス」を参照してください。

- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストリクエストリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。
- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) AWS がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「マネージドポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーで IAM の AWS マネージドポリシーを使用することはできません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、『Amazon Simple Storage Service デベロッパーガイド』の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS

アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、AWS Organizations ユーザーガイドの「SCP の仕組み」を参照してください。

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシー評価ロジック」を参照してください。

が IAM と AWS App Mesh 連携する方法

App Mesh へのアクセスを管理するために IAM を使用する前に、App Mesh でどの IAM 機能が使用できるかを理解しておく必要があります。App Mesh およびその他の AWS のサービスが IAM と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

トピック

- App Mesh アイデンティティベースのポリシー
- App Mesh でのリソースベースのポリシー
- App Mesh タグに基づく認可
- App Mesh IAM ロール

App Mesh アイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、アクションを許可または拒否する条件を指定できます。App Mesh では、特定のアクション、リソース、

および条件キーがサポートされています。JSON ポリシーで使用するすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素のリファレンス」を参照してください。

アクション

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーのAction要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

App Mesh のポリシーアクションには、アクションの前にプレフィックス appmesh: を使用します。例えば、appmesh:ListMeshes API 操作を使用してアカウント内のメッシュを一覧表示する許可を誰かに付与するには、その appmesh:ListMeshes アクションをポリシーに含めます。ポリシーステートメントには、Action または NotAction の要素を含める必要があります。

単一のステートメントに複数のアクションを指定するには、次のようにコンマで区切ります。

```
"Action": [
      "appmesh:ListMeshes",
      "appmesh:ListVirtualNodes"
]
```

ワイルドカード *を使用して複数のアクションを指定することができます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "appmesh:Describe*"
```

App Mesh アクションのリストを表示するには、「IAM ユーザーガイド」の「AWS App Meshによって定義されたアクション」を参照してください。

リソース

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

操作のリスト化など、リソースレベルの許可がサポートされていないアクションの場合は、ワイルドカード (*) を使用して、ステートメントがすべてのリソースに適用されることを示します。

```
"Resource": "*"
```

App Mesh mesh リソースは、次の ARN を持ちます。

```
arn:${Partition}:appmesh:${Region}:${Account}:mesh/${MeshName}
```

ARN の形式の詳細については、「Amazon リソースネーム (ARNs AWS 「サービス名前空間」を参照してください。

例えば、ステートメントの *Region-code* リージョンに *apps* という名前のメッシュを指定するには、次の ARN を使用します。

```
arn:aws:appmesh:Region-code:111122223333:mesh/apps
```

特定のアカウントに属するすべてのインスタンスを指定するには、ワイルドカード *を使用します。

```
"Resource": "arn:aws:appmesh:Region-code:111122223333:mesh/*"
```

リソースの作成など、一部の App Mesh アクションは、特定のリソースで実行できません。このような場合は、ワイルドカード *を使用する必要があります。

```
"Resource": "*"
```

App Mesh の API アクションの多くが複数のリソースと関連します。例えば、CreateRoute は、仮想ノードターゲットを使用してルートを作成します。この場合、IAM ユーザーはルートと仮想ノードを使用するアクセス許可を持っている必要があります。複数リソースを単一ステートメントで指定するには、ARN をカンマで区切ります。

```
"Resource": [
      "arn:aws:appmesh:Region-code:111122223333:mesh/apps/virtualRouter/serviceB/route/
*",
```

```
        "arn:aws:appmesh:Region-code:111122223333:mesh/apps/virtualNode/serviceB"
]
```

App Mesh リソースタイプとその ARN のリストを表示するには、「IAM ユーザーガイド」の「AWS App Meshで定義されるリソース」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、AWS App Meshで定義されるアクションを参照してください。

条件キー

App Mesh では、いくつかのグローバル条件キーの使用がサポートされています。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」の「AWS グローバル条件コンテキストキー」を参照してください。App Mesh でサポートされるグローバル条件キーのリストを確認するには、「IAM ユーザーガイド」の 「AWS App Meshの条件キー」を参照してください。条件キーで使用できるアクションとリソースについては、「 で定義されるアクション AWS App Mesh」を参照してください。

例

App Mesh アイデンティティベースのポリシーの例を表示するには、「AWS App Mesh アイデンティティベースのポリシーの例」を参照してください。

App Mesh でのリソースベースのポリシー

App Mesh はリソースベースのポリシーをサポートしていません。ただし、 AWS Resource Access Manager （ AWS RAM) サービスを使用してサービス間でメッシュを共有する場合、 AWS サービスによってリソースベースのポリシーがメッシュに適用されます AWS RAM 。詳細については、「メッシュの権限の付与」を参照してください。

App Mesh タグに基づく認可

タグをApp Mesh リソースにアタッチすることも、App Mesh へのリクエストでタグを渡すこともできます。タグに基づいてアクセスを制御するには、appmesh:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件の要素でタグ情報を提供します。App Mesh リソースのタグ付けの詳細については、AWS「リソースのタグ付け」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「制限付きタグを使用した App Mesh メッシュの作成」を参照してください。

App Mesh IAM ロール

IAM ロールは、特定のアクセス許可を持つ AWS アカウント内のエンティティです。

App Mesh での一時的な認証情報の使用

一時的な認証情報を使用して、フェデレーションでサインインする、IAM ロールを引き受ける、またはクロスアカウントロールを引き受けることができます。一時的なセキュリティ認証情報を取得するには、 AssumeRoleやGetFederationトークン などの AWS STS API オペレーションを呼び出します。

App Mesh では、一時認証情報の使用はサポートされています。

サービスリンクロール

サービスにリンクされたロールを使用すると、 AWS サービスは他の サービスのリソースにアクセスして、ユーザーに代わってアクションを実行できます。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。IAM 管理者はサービスリンクロールのアクセス許可を表示できますが、編集することはできません。

App Mesh ではサービスリンクロールはサポートされていません。App Mesh サービスリンクロールの作成または管理の詳細については、「App Mesh のサービスリンクロールの使用」を参照してください。

サービスロール

この機能により、ユーザーに代わってサービスがサービスロールを引き受けることが許可されます。このロールにより、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを完了することが許可されます。サービスロールは、IAM アカウントに表示され、アカウントによって所有されます。つまり、IAM 管理者は、このロールの権限を変更できます。ただし、それにより、サービスの機能が損なわれる場合があります。

App Mesh ではサービスロールはサポートされていません。

AWS App Mesh アイデンティティベースのポリシーの例

デフォルトでは、IAM ユーザーおよびロールには、App Mesh リソースを作成または変更するアクセス許可はありません。また、、 AWS Management Console AWS CLI、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらの権限が必要な IAM ユーザーまたはグループにそのポリシーをアタッチする必要があります。

JSON ポリシードキュメントのこれらの例を使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「JSON タブでのポリシーの作成」を参照してください。

トピック

- ポリシーのベストプラクティス
- App Mesh コンソールの使用
- ユーザーが自分の許可を表示できるようにする
- メッシュの作成
- すべてのメッシュを一覧表示して説明する
- 制限付きタグを使用した App Mesh メッシュの作成

ポリシーのベストプラクティス

アイデンティティベースのポリシーは、ユーザーのアカウントで誰かが App Mesh リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーとアクセス許可」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、 などの特定の を介してサービスアクションが使用される場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素: 条件) を参照してください。

- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

App Mesh コンソールの使用

AWS App Mesh コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 AWS アカウント内の App Mesh リソースの詳細を一覧表示および表示できます。最小限必要な許可よりも厳しく制限されたアイデンティティベースポリシーを作成すると、そのポリシーを添付したエンティティ (IAM ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。ユーザーに対する `AWSAppMeshReadOnly` マネージドポリシーをアタッチできます。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーが自分の許可を表示できるようにする

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

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

```
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "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": "*"
        }
    ]
}
```

メッシュの作成

この例は、リージョンを問わず、アカウントのメッシュをユーザーが作成できるようにするポリシーを作成する方法を示しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appmesh:CreateMesh",
            "Resource": "arn:aws:appmesh:*:123456789012:CreateMesh"
        }
    ]
```

```
}
```

すべてのメッシュを一覧表示して説明する

この例は、すべてのクラスターの一覧表示または記述するための読み取り専用アクセス許可をユーザーに許可するポリシーを作成する方法を示しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appmesh:DescribeMesh",
                "appmesh:ListMeshes"
            ],
            "Resource": "*"
        }
    ]
}
```

制限付きタグを使用した App Mesh メッシュの作成

IAM ポリシーでタグを使用して、IAM リクエストで渡すことができるタグを制御できます。IAM ユーザーまたはロールで追加、変更、または削除できるタグのキー値のペアを指定できます。この例では、*teamName* というタグと *booksTeam* という値でメッシュを作成する場合のみ、メッシュの作成を許可するポリシーを作成する方法を示しています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appmesh:CreateMesh",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/teamName": "booksTeam"
                }
            }
        }
```

```
    ]
}
```

このポリシーをアカウントの IAM ユーザーにアタッチできます。ユーザーがメッシュを作成しようとする場合、メッシュには teamName という名前のタグと booksTeam という値が含まれている必要があります。メッシュにこのタグと値が含まれていない場合、メッシュの作成は失敗します。詳細については、「IAM ユーザーガイド」の 「IAM JSON ポリシーの要素: 条件」を参照してください。

AWS App Mesh の マネージドポリシー

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、多くの一般的なユースケースにアクセス許可を付与するように設計されているため、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できます。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケース別にカスタマーマネージドポリシーを定義して、マネージドポリシーを絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS 管理ポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) が更新されます。 は、新しい AWS のサービス が起動されたとき、または既存のサービスで新しい API AWS オペレーションが使用可能になったときに、 AWS 管理ポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS マネージドポリシー: AWSAppMeshServiceRolePolicy

IAM エンティティに AWSAppMeshServiceRolePolicy をアタッチできます。が使用または管理する AWS サービスとリソースへのアクセスを有効にします AWS App Mesh。

このポリシーのアクセス許可を確認するには、「 マネージドポリシーリファレンスAWSAppMeshServiceRolePolicy」の「」を参照してください。 AWS

AWSAppMeshServiceRolePolicy のアクセス許可の詳細については、「App Mesh のサービスリンクロールにおけるアクセス許可」を参照してください。

AWS マネージドポリシー: AWSAppMeshEnvoyAccess

IAM エンティティに AWSAppMeshEnvoyAccess をアタッチできます。仮想ノード設定にアクセスするための App Mesh Envoy ポリシー。

このポリシーのアクセス許可を確認するには、「 管理ポリシーリファレンスAWSAppMeshEnvoyAccess」の「」を参照してください。 AWS

AWS 管理ポリシー: AWSAppMeshFullAccess

IAM エンティティに AWSAppMeshFullAccess をアタッチできます。 AWS App Mesh APIsおよびへのフルアクセスを提供します AWS Management Console。

このポリシーのアクセス許可を確認するには、「 管理ポリシーリファレンスAWSAppMeshFullAccess」の「」を参照してください。 AWS

AWS マネージドポリシー: AWSAppMeshPreviewEnvoyAccess

IAM エンティティに AWSAppMeshPreviewEnvoyAccess をアタッチできます。仮想ノード設定にアクセスするための App Mesh Preview Envoy ポリシー。

このポリシーのアクセス許可を確認するには、「 マネージドポリシーリファレンスAWSAppMeshPreviewEnvoyAccess」の「」を参照してください。 AWS

AWS 管理ポリシー: AWSAppMeshPreviewServiceRolePolicy

IAM エンティティに AWSAppMeshPreviewServiceRolePolicy をアタッチできます。が使用または管理する AWS サービスとリソースへのアクセスを有効にします AWS App Mesh。

このポリシーのアクセス許可を確認するには、「 管理ポリシーリファレンスAWSAppMeshPreviewServiceRolePolicy」の「」を参照してください。 AWS

AWS 管理ポリシー: AWSAppMeshReadOnly

IAM エンティティに AWSAppMeshReadOnly をアタッチできます。 AWS App Mesh APIsおよび への読み取り専用アクセスを提供します AWS Management Console。

このポリシーのアクセス許可を確認するには、「 管理ポリシーリファレンスAWSAppMeshReadOnly」の「」を参照してください。 AWS

AWS App MeshAWS 管理ポリシーの更新

このサービスがこれらの変更の追跡を開始した AWS App Mesh 以降の の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知については、 AWS App Mesh ドキュメントの履歴ページの RSS フィードをサブスクライブしてください。

変更	説明	日付
AWSAppMeshFullAccess – ポリシーを更新しました。	TagResource および UntagResource APIs API へのアクセスを許可するAWSAppMeshFullAccess ように更新されました。	2024 年 4 月 24 日
AWSAppMeshServiceRolePolicy、 AWSServiceRoleForAppMesh – ポリシーを更新しました。	API へのアクセスを許可するAWSAppMeshServiceRolePolicy ように AWSServiceRoleForAppMesh と を更新しました AWS Cloud Map DiscoverInstancesRevision 。	2023 年 10 月 12 日

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM Identity Center :

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

App Mesh のサービスリンクロールの使用

AWS App Mesh は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスリンクロールは、App Mesh に直接リンクされた一意のタイプの IAM ロールです サービスリンクロールは、App Mesh によって事前に定義され、ユーザーに代わってサービスが他の AWS のサービスを呼び出すために必要なアクセス許可がすべて含まれています。

サービスリンクロールを使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、App Mesh の設定が簡単になります。App Mesh は、サービスリンクロールのアクセス許可を定義します。特に定義されている場合を除き、App Mesh のみがそのロールを引き受けることができます。定義される許可には、信頼ポリシーとアクセス許可ポリシーが含まれており、そのアクセス許可ポリシーを他の IAM エンティティに添付することはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへのアクセス許可を不用意に削除することができないため、App Mesh のリソースを保護することができます。

サービスリンクロールをサポートする他のサービスについては、「IAM と連携する AWS のサービス」を参照して、サービスリンクロール列にはいと表示されているサービスを探してください。そのサービスに対するサービスリンクロールに関するドキュメントを表示するには、リンク付きのはいを選択します。

App Mesh のサービスリンクロールにおけるアクセス許可

App Mesh は、AWSServiceRoleForAppMesh という名前のサービスリンクロールを使用します。このロールにより、App Mesh はユーザーに代わって AWS サービスを呼び出すことができます。

AWSServiceRoleForAppMesh サービスリンクロールは、ロールを継承するために `appmesh.amazonaws.com` サービスを信頼します。

アクセス許可の詳細

- `servicediscovery:DiscoverInstances`-App Mesh がすべての AWS リソースでアクションを完了できるようにします。
- `servicediscovery:DiscoverInstancesRevision`-App Mesh がすべての AWS リソースでアクションを完了できるようにします。

AWSServiceRoleForAppMesh

このポリシーには、以下の許可が含まれています。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "CloudMapServiceDiscovery",
   "Effect": "Allow",
   "Action": [
    "servicediscovery:DiscoverInstances",
    "servicediscovery:DiscoverInstancesRevision"
   ],
   "Resource": "*"
  },
  {
   "Sid": "ACMCertificateVerification",
   "Effect": "Allow",
   "Action": [
    "acm:DescribeCertificate"
   ],
   "Resource": "*"
  }
 ]
}
```

サービスにリンクされたロールの作成、編集、削除をIAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの許可」を参照してください。

App Mesh のサービスリンクロールの作成

2019 年 6 月 5 日以降に、AWS Management Console、AWS CLI、または AWS API でメッシュを作成した場合、App Mesh はサービスリンクロールを作成しています。サービスリンクロールを作成するためには、メッシュの作成に使用した IAM アカウントには、それに添付された `AWSAppMeshFullAccess` IAM ポリシー、または `iam:CreateServiceLinkedRole` アクセス許可を含むそれに添付されたポリシーをが必要です。このサービスにリンクされたロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。メッシュを作成すると、App Mesh はサービスリンクロールを再度作成します。2019 年 6 月 5 日以前に作成されたメッシュのみがアカウントに含まれており、それらのメッシュでサービスにリンクされたロールを使用する場合は、IAM コンソールを使用してロールを作成できます。

App Mesh ユースケースでサービスリンクロールを作成するには、IAM コンソールを使用できます。AWS CLI または AWS API で、`appmesh.amazonaws.com` サービス名を使用してサービスリン

クロールを作成します。詳細については、IAM ユーザーガイドの「サービスリンクロールの作成」を参照してください。このサービスリンクロールを削除する場合、この同じプロセスを使用して、もう一度ロールを作成できます。

App Mesh のサービスリンクロールの編集

で、AWSServiceRoleForAppMesh のサービスリンクロールを編集することはできません。サービスリンクロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

App Mesh でのサービスリンクロールの削除

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、積極的にモニタリングまたは保守されていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスにリンクされたロールのリソースをクリーンアップする必要があります。

> **Note**
>
> リソースを削除しようとしたときに AppMesh サービスがロールを使用している場合、削除が失敗する可能性があります。失敗した場合は、数分待ってから操作を再試行してください。

AWSServiceRoleForAppMesh が使用している App Mesh リソースを削除するには

1. メッシュ内のすべてのルータに定義されているルートをすべて削除します。
2. メッシュ内の仮想ルーターをすべて削します。
3. メッシュ内の仮想サービスをすべて削します。
4. メッシュ内の仮想ノードをすべて削除します。
5. メッシュを削除します。

アカウント内のすべてのメッシュについて、前の手順を完了します。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、サービスリンクロール AWSServiceRoleForAppMesh を削除します。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの削除」を参照してください。

App Mesh サービスリンクロールをサポートするリージョン

App Mesh では、このサービスが利用可能なすべてのリージョンで、サービスリンクロールの使用がサポートされています。詳細については、「App Mesh エンドポイントとクォータ」を参照してください。

Envoy プロキシの認可

プロキシ認可は、Amazon ECS タスク内、Amazon EKS で実行されている Kubernetes ポッド、または Amazon EC2 インスタンスで実行されている Envoy プロキシが、App Mesh Envoy Management Service から1つ以上のメッシュエンドポイントの設定を読み取ることを認可します。2021 年 4 月 26 日以前にすでに Envoy を App Mesh エンドポイントに接続している顧客アカウントの場合、Transport Layer Security (TLS) を使用する仮想ノードと仮想ゲートウェイ (TLS の有無にかかわらず) にはプロキシ認可が必要です。2021 年 4 月 26 日以降に Envoy を App Mesh エンドポイントに接続する顧客アカウントの場合、すべての App Mesh 機能にプロキシ認可が必要です。すべての顧客アカウントが、TLS を使用していない場合でも、すべての仮想ノードのプロキシ認可を有効にして、特定のリソースへの認証に IAM を使用して安全で一貫性のあるエクスペリエンスを提供することをお勧めします。プロキシ認可では、`appmesh:StreamAggregatedResources` アクセス許可は IAM ポリシーで指定されています。ポリシーは IAM ロールに添付する必要があり、その IAM ロールは、プロキシをホストするコンピューティングリソースに添付する必要があります。

IAM ポリシーを作成する

サービスメッシュ内のすべてのメッシュエンドポイントで、すべてのメッシュエンドポイントの設定を読み取れるようにするには、IAM ロールの作成 に進みます。個々のメッシュエンドポイントで設定を読み取りできるメッシュエンドポイントを制限する場合は、1 つ以上の IAM ポリシーを作成する必要があります。設定を読み取りできるメッシュエンドポイントを、特定のコンピューティングリソースで実行されている Envoy プロキシのみに制限することをお勧めします。IAM ポリシーを作成し、`appmesh:StreamAggregatedResources` ポリシーへのアクセス許可を追加します。次のポリシーの例では、`serviceBv1` と `serviceBv2` という名前の仮想ノードの設定をサービスメッシュで読み取りすることを許可します。サービスメッシュで定義されている他の仮想ノードの設定を読み取りすることはできません。IAM ポリシーの作成と編集の詳細については、「IAM ポリシーの作成」と「IAM ポリシーの編集」を参照してください。

```
{
```

```
    "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"
            ]
        }
    ]
}
```

複数のポリシーを作成し、各ポリシーで異なるメッシュエンドポイントへのアクセスを制限できます。

IAM ロールの作成

サービスメッシュ内のすべてのメッシュエンドポイントで、すべてのメッシュエンドポイントの設定を読み取りできるようにするには、1 つの IAM ロールを作成するだけで済みます。個々のメッシュエンドポイントで設定を読み取りできるメッシュエンドポイントを制限する場合は、前の手順で作成したポリシーごとにロールを作成する必要があります。プロキシが実行されるコンピュートリソースの指示を完了します。

- Amazon EKS — 単一のロールを使用する場合は、クラスターの作成時に作成され、ワーカーノードに割り当てられた既存のロールを使用できます。複数のロールを使用するには、クラスターが「クラスターでのサービスアカウントの IAM ロールの有効化」で定義されている要件を満たしている必要があります。IAM ロールを作成し、ロールを Kubernetes サービスアカウントに関連付けます。詳細については、「サービスアカウントの IAM ロールとポリシーの作成」と「サービスアカウントの IAM ロールの指定」を参照してください。
- Amazon ECS – AWS サービスを選択し、Elastic Container Service を選択してから、IAMロールを作成するときに Elastic Container Service Task のユースケースを選択します。
- Amazon EC2 – AWS サービスを選択し、EC2 を選択してから、IAM ロールを作成するときに EC2 ユースケースを選択します。これは、プロキシを Amazon EC2 インスタンスで直接ホストする場合でも、インスタンスで実行されている Kubernetes でホストする場合でも適用されます。

IAM ロールを作成する方法の詳細については、「AWS サービス用のロールの作成」を参照してください。

IAM ポリシーの添付

サービスメッシュ内のすべてのメッシュエンドポイントで、すべてのメッシュエンドポイントの設定を読み取りできるようにするには、`AWSAppMeshEnvoyAccess` マネージド IAM ポリシーを、前のステップで作成した IAM ロールにアタッチします。個々のメッシュエンドポイントで設定を読み込みできるメッシュエンドポイントを制限する場合は、作成した各ポリシーを、作成した各ロールにアタッチします。カスタムまたはマネージド IAM ポリシーを IAM ロールに添付する方法の詳細については、「IAM ID アクセス許可の追加」を参照してください。

IAM ロールを添付する

各 IAM ロールを適切なコンピューティングリソースに添付します。

- Amazon EKS — ワーカーノードに添付されたロールにポリシーを添付した場合は、この手順をスキップできます。個別のロールを作成した場合は、各ロールを個別の Kubernetes サービスアカウントに割り当てて、各サービスアカウントを Envoy プロキシを含む個々の Kubernetes ポッドデプロイ仕様に割り当てます。詳細については、「Amazon EKS ユーザーガイド」の「サービスアカウントの IAM ロールの指定」、および「Kubernetes ドキュメント」の「Pod のサービスアカウントを設定する」を参照してください。
- Amazon ECS — Envoy プロキシを含むタスク定義に Amazon ECS タスクロールを添付します。タスクは EC2 または Fargate 起動タイプでデプロイできます。Amazon ECS タスクロールを作成してタスクにアタッチする方法の詳細については、「タスクの IAM ロールの指定」を参照してください。
- Amazon EC2 — IAM ロールは、Envoy プロキシをホストする Amazon EC2 インスタンスに添付する必要があります。Amazon EC2 インスタンスにロールをアタッチする方法の詳細については、「IAM ロールを作成しましたが、今度は EC2 インスタンスに割り当てたいと思います」を参照してください。

アクセス許可の確認

コンピュートサービス名の1つを選択して、プロキシをホストするコンピュートリソースに `appmesh:StreamAggregatedResources` アクセス許可が割り当てられていることを確認します。

Amazon EKS

カスタムポリシーは、ワーカーノード、個々のポッド、またはその両方に割り当てられるロールに割り当てることができます。ただし、個々のポッドのアクセスを個々のメッシュエンドポイントに制限できるように、個々の Pod にのみポリシーを割り当てることをお勧めします。ポリシーがワーカーノードに割り当てられたロールに添付されている場合は、Amazon EC2 タブをクリックし、ワーカーノードインスタンスの手順を完了します。Kubernetes ポッドに割り当てられている IAM ロールを特定するには、次の手順を実行します。

1. Kubernetes サービスアカウントが割り当てられていることを確認するポッドを含む Kubernetes デプロイの詳細を表示します。次のコマンドは、*my-deployment* という名前のデプロイの詳細を表示します。

   ```
   kubectl describe deployment my-deployment
   ```

   返された出力では、`Service Account:` の右側にある値をメモします。`Service Account:` で始まる行が存在しない場合、カスタム Kubernetes サービスアカウントは現在デプロイメントに割り当てられていません。1 つを割り当てる必要があります。詳細については、「Kubernetes ドキュメント」の「ポッド用にサービスアカウントを設定する」を参照してください。

2. 前のステップで返されたサービスアカウントの詳細を表示します。次のコマンドは、*my-service-account* という名前のサービスアカウントの詳細を表示します。

   ```
   kubectl describe serviceaccount my-service-account
   ```

   Kubernetes サービスアカウントが AWS Identity and Access Management ロールに関連付けられている場合、返された行の 1 つは、次の例のようになります。

   ```
   Annotations:         eks.amazonaws.com/role-arn=arn:aws:iam::123456789012:role/
   my-deployment
   ```

   前の例 `my-deployment` では、サービスアカウントに関連付けられている IAM ロールの名前です。サービスアカウントの出力に上記の例のような行が含まれていない場合、Kubernetes サービスアカウントは AWS Identity and Access Management アカウントに関連付けされていませんから、1 つ関連付けする必要があります。詳細については、「サービスアカウントの IAM ロールの指定」を参照してください。

3. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
4. 左のナビゲーションペインで、[ロール] を選択します。前のステップでメモした IAM ロールの名前を選択します。
5. 以前に作成したカスタムポリシー、または AWSAppMeshEnvoyAccess マネージドポリシーが一覧表示されます。どちらのポリシーもアタッチされていない場合は、IAM ポリシーを IAM ロールにアタッチします。カスタム IAM ポリシーをアタッチしたいが持っていない場合は、必要なアクセス許可を持つカスタム IAM ポリシーを作成する必要があります。カスタム IAM ポリシーが添付されている場合は、ポリシーを選択し、そこに `"Action": "appmesh:StreamAggregatedResources"` が含まれていることを確認します。そうでない場合は、そのアクセス許可をカスタム IAM ポリシーに追加する必要があります。また、特定のメッシュエンドポイントに適切な Amazon リソースネーム (ARN) が表示されていることを確認することもできます。ARN がリストされていない場合は、ポリシーを編集して、リストされた ARN を追加、削除、または変更することができます。詳細については、「IAM ポリシーの編集」および IAM ポリシーを作成する を参照してください。
6. Envoy プロキシを含む各 Kubernetes ポッドについて、上記の手順を繰り返します。

Amazon ECS

1. Amazon ECS コンソールから、[タスク定義] を選択します。
2. Amazon ECS タスクを選択します。
3. [タスク定義名] ページで、タスク定義を選択します。
4. [タスク定義] ページで、[タスクロール] の右側にある IAM ロール名のリンクを選択します。IAM ロールがリストされていない場合は、IAM ロールを作成し、タスク定義を更新してタスクにアタッチする必要があります。
5. [概要] ページの [アクセス許可] タブで、以前に作成したカスタムポリシー、または AWSAppMeshEnvoyAccess マネージドポリシーのいずれかが一覧表示されていることを確認します。どちらのポリシーもアタッチされていない場合は、IAM ポリシーを IAM ロールにアタッチします。カスタム IAM ポリシーをアタッチしたいが持っていない場合は、カスタム IAM ポリシーを作成します。カスタム IAM ポリシーが添付されている場合は、ポリシーを選択し、そこに `"Action": "appmesh:StreamAggregatedResources"` が含まれていることを確認します。そうでない場合は、そのアクセス許可をカスタム IAM ポリシーに追加する必要があります。また、特定のメッシュエンドポイントに適切な Amazon リソースネーム (ARN) が表示されていることを確認することもできます。ARN がリストされていない場合は、ポリシーを編集して、リストされた ARN を追加、削除、または変更することができま

す。詳細については、「IAM ポリシーの編集」および IAM ポリシーを作成する を参照してください。

6. Envoy プロキシを含むタスク定義ごとに、上記の手順を繰り返します。

Amazon EC2

1. Amazon EC2 コンソールから、左側のナビゲーションで [インスタンス] を選択します。
2. Envoy プロキシをホストするインスタンスの 1 つを選択します。
3. [説明] タブで、IAM ロールの右側にある IAM ロール名のリンクを選択します。IAM ロールがリストされていない場合、IAM ロールを作成する必要があります。
4. [概要] ページの [アクセス許可] タブで、以前に作成したカスタムポリシー、または AWSAppMeshEnvoyAccess マネージドポリシーのいずれかが一覧表示されていることを確認します。どちらのポリシーもアタッチされていない場合は、IAM ポリシーを IAM ロールにアタッチします。カスタム IAM ポリシーをアタッチしたいが持っていない場合は、カスタム IAM ポリシーを作成します。カスタム IAM ポリシーが添付されている場合は、ポリシーを選択し、そこに `"Action": "appmesh:StreamAggregatedResources"` が含まれていることを確認します。そうでない場合は、そのアクセス許可をカスタム IAM ポリシーに追加する必要があります。また、特定のメッシュエンドポイントに適切な Amazon リソースネーム (ARN) が表示されていることを確認することもできます。ARN がリストされていない場合は、ポリシーを編集して、リストされた ARN を追加、削除、または変更することができます。詳細については、「IAM ポリシーの編集」および IAM ポリシーを作成する を参照してください。
5. Envoy プロキシをホストしているインスタンスごとに、上記の手順を繰り返します。

AWS App Mesh ID とアクセスのトラブルシューティング

次の情報は、App Mesh と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- App Mesh でアクションを実行する認可がされていない
- AWS アカウント外のユーザーに App Mesh リソースへのアクセスを許可したい

App Mesh でアクションを実行する認可がされていない

からアクションを実行する権限がないと AWS Management Console 通知された場合は、管理者に連絡してサポートを依頼する必要があります。管理者とは、サインイン認証情報を提供した担当者です。

次のエラーは、mateojackson IAM ユーザーがコンソールを使用して *my-mesh* という名前のメッシュに *my-virtual-node* という名前の仮想ノードを作成しようとしたが、appmesh:CreateVirtualNode アクセス許可がない場合に発生します。

```
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
```

この場合、マテオは、管理者にポリシーを更新して、appmesh:CreateVirtualNode アクションを使用して仮想ノードを作成できるようにしてほしいと依頼します。

> **Note**
>
> 仮想ノードはメッシュ内に作成されるため、マテオのアカウントめでも、コンソールで仮想ノードを作成するための appmesh:DescribeMesh および appmesh:ListMeshes アクションが必要です。

AWS アカウント外のユーザーに App Mesh リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、次を参照してください。

- App Mesh でこれらの機能がサポートされているかどうかを確認するには、「が IAM と AWS App Mesh 連携する方法」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。

- リソースへのアクセスをサードパーティー に提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイドの外部認証されたユーザーへのアクセスの提供 (ID フェデレーション) を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いについては、IAM ユーザーガイドの「IAM でのクロスアカウントリソースアクセス」を参照してください。

を使用して AWS App Mesh API 呼び出しをロギングする AWS CloudTrail

AWS App Mesh ユーザーAWS CloudTrail、ロール、 AWS のサービスまたはによって実行されたアクションの記録を提供するサービスと統合されています。 CloudTrail App Mesh のすべての API 呼び出しをイベントとしてキャプチャします。キャプチャされたコールには、 App Mesh コンソールからのコールと、App Mesh API オペレーションへのコードコールが含まれます。によって収集された情報を使用して CloudTrail、App Mesh に対して行われたリクエスト、リクエストが行われた IP アドレス、実行日時、その他の詳細を判断できます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するために役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか
- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

CloudTrail AWS アカウント アカウントを作成した時点で有効になり、 CloudTrail 自動的にイベント履歴にアクセスできるようになります。 CloudTrail イベント履歴では、過去 90 日間に記録された管理イベントの表示、検索、ダウンロード、変更が不可能な記録が記録されます。 AWS リージョン詳細については、『ユーザガイド』の「 CloudTrail AWS CloudTrail イベント履歴の操作」を参照してください。 CloudTrailイベント履歴の閲覧には料金はかかりません。

AWS アカウント 過去 90 日間のイベントを継続的に記録するには、トレイルまたは CloudTrailLake イベントデータストアを作成してください。

CloudTrail トレイル

トレイルを使用すると CloudTrail 、Amazon S3 バケットにログファイルを配信できます。 AWS Management Console を使用して作成されたトレイルはすべてマルチリージョンです。 AWS CLI を使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。 AWS リージョン アカウント内のアクティビティをすべて記録できるため、マルチリージョントレイルを作成することをおすすめします。単一リージョンの証跡を作成する場合、証跡の AWS リージョンに記録されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS アカウントの証跡の作成」および「組織の証跡の作成」を参照してください。

CloudTrail トレイルを作成することで、進行中の管理イベントのコピーを 1 つ無料でAmazon S3 バケットに配信できますが、Amazon S3 ストレージには料金がかかります。 CloudTrail 料金の詳細については、「AWS CloudTrail 料金表」を参照してください。Amazon S3 の料金に関する詳細については、「Amazon S3 の料金」を参照してください。

CloudTrail レイクイベントデータストア

CloudTrail Lake では、イベントに対して SQL ベースのクエリを実行できます。 CloudTrail Lake は行ベースの JSON 形式の既存のイベントを Apache ORC 形式に変換します。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントはイベントデータストアに集約されます。イベントデータストアは、高度なイベントセレクタを適用することによって選択する条件に基いた、イベントのイミュータブルなコレクションです。どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクタが制御します。CloudTrail Lake について詳しくは、『ユーザーガイド』の「AWS CloudTrail Lake での作業」を参照してください。AWS CloudTrail

CloudTrail Lake イベントのデータストアとクエリにはコストがかかります。イベントデータストアを作成する際に、イベントデータストアに使用する料金オプションを選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。料金について詳しくは、「 CloudTrail AWS CloudTrail 料金表」を参照してください。

のApp Mesh 管理イベント CloudTrail

管理イベントは、内のリソースに対して実行される管理操作に関する情報を提供します AWS アカウント。これらのイベントは、コントロールプレーンオペレーションとも呼ばれます。デフォルトでは、 CloudTrail 管理イベントが記録されます。

AWS App Mesh App Mesh コントロールプレーンのすべての操作を管理イベントとして記録します。App Mesh AWS App Mesh がログに記録するコントロールプレーン操作のリストについては CloudTrail、AWS App Mesh API リファレンスをご覧ください。

App Mesh イベントの例

イベントはあらゆるソースからの単一のリクエストを表し、リクエストされた API オペレーションに関する情報、オペレーションの日時、リクエストパラメータなどが含まれます。 CloudTrail ログファイルはパブリック API 呼び出しの順序付けられたスタックトレースではないため、イベントは特定の順序で表示されることはありません。

次の例は、 CloudTrail `StreamAggregatedResources`アクションを示すログエントリを示しています。

```
{
    "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 レコードコンテンツについては、『AWS CloudTrail ユーザーガイド』の「CloudTrailレコードコンテンツ」を参照してください。

でのデータ保護 AWS App Mesh

AWS のデータ保護には、https://aws.amazon.com/compliance/shared-responsibility-model/、 (責任分担モデル) が適用されます AWS App Mesh。このモデルで説明したように、 AWS は、 AWS クラウドすべてを支えるグローバルインフラストラクチャを保護する責任があります。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された記事「AWS 責任共有モデルおよび GDPR」を参照してください。

データ保護のため、 AWS アカウント 認証情報を保護し、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用してリソースと通信します。 AWS TLS 1.2 は必須であり TLS 1.3 がお勧めです。
- を使用して API とユーザーアクティビティのロギングを設定します。 AWS CloudTrail
- AWS 暗号化ソリューションと、 AWS のサービスその中に含まれるデフォルトのセキュリティコントロールをすべて使用してください。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介してアクセスするときに FIPS 140-2 で検証された暗号モジュールが必要な場合は、FIPS エンドポイントを使用してください。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これには、コンソール、API

AWS CLI、または AWS SDK AWS のサービス を使用して App Mesh やその他のアプリケーションを操作する場合も含まれます。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

データ暗号化

データは、App Mesh を使用すると暗号化されます。

保管中の暗号化

デフォルトでは、作成したApp Mesh 構成は保存時に暗号化されます。

転送中の暗号化

App Mesh サービスのエンドポイントは HTTPS プロトコルを使用します。Envoy プロキシと App Mesh Envoy 管理サービス間のすべての通信は暗号化されます。Envoy プロキシと App Mesh Envoy 管理サービス間の通信に FIPS 準拠の暗号化が必要な場合は、Envoy プロキシコンテナイメージの FIPS バリアントを使用できます。詳細については、「Envoy イメージ」を参照してください。

仮想ノード内のコンテナ間の通信は暗号化されませんが、このトラフィックはネットワーク名前空間を離れることはありません。

のコンプライアンス検証 AWS App Mesh

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。

- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド — コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

のインフラストラクチャセキュリティ AWS App Mesh

マネージドサービスとして、 AWS App Mesh AWS グローバルなネットワークセキュリティによって保護されています。 AWS AWS セキュリティサービスとインフラストラクチャの保護方法に

ついては、「AWS Cloud Security」を参照してください。 AWS インフラストラクチャセキュリティのベストプラクティスを使用して環境を設計するには、「Security Pillar AWS Well-Architected Framework におけるインフラストラクチャ保護」を参照してください。

AWS 公開されている API 呼び出しを使用して、ネットワーク経由で App Mesh にアクセスします。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2、できれば TLS 1.3 が必要です。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

App Mesh がインターフェイス VPC エンドポイントを使用するように設定することで、VPC のセキュリティポスチャーを向上させることができます。詳細については、「App Mesh インターフェイス VPC エンドポイント ()AWS PrivateLink」を参照してください。

App Mesh インターフェイス VPC エンドポイント ()AWS PrivateLink

インターフェイス VPC エンドポイントを使用するように Amazon ECS を設定することで、VPC のセキュリティ体制を改善できます。インターフェイスエンドポイントは AWS PrivateLink、プライベート IP アドレスを使用して App Mesh API にプライベートにアクセスできるようにするテクノロジーを利用しています。 PrivateLinkAmazon VPC とApp Mesh 間のすべてのネットワークトラフィックを Amazon ネットワークに制限します。

設定は必須ではありませんが PrivateLink、設定することをおすすめします。VPC PrivateLink エンドポイントの詳細とインターフェースについては、「によるサービスへのアクセス」を参照してください。 AWS PrivateLink

App Mesh インターフェイスの VPC エンドポイントに関する考慮事項

App Mesh 用のインターフェイス VPC エンドポイントを設定する前に、次の考慮事項に注意してください。

- Amazon VPC にインターネットゲートウェイがなく、awslogsタスクがログドライバーを使用してログ情報をログに送信する場合、 CloudWatch Logs CloudWatch 用のインターフェイス VPC エンドポイントを作成する必要があります。詳細については、Amazon CloudWatch Logs ユーザーガイドの「インターフェイス VPC CloudWatch エンドポイントでのログの使用」を参照してください。
- VPC AWS エンドポイントはクロスリージョンリクエストをサポートしていません。エンドポイントには、App Mesh への API コールを発行する予定の同じリージョンに作成することを確認してください。
- VPC エンドポイントでは、Amazon Route 53 を介して Amazon 提供の DNS のみがサポートされています。独自の DNS を使用したい場合は、条件付き DNS 転送を使用できます。詳細については、Amazon VPC ユーザーガイドの「DHCP Options Sets」を参照してください。
- VPC エンドポイントにアタッチされたセキュリティグループは、Amazon VPC のプライベートサブネットからのポート443での着信接続を許可する必要があります。

> **Note**
>
> エンドポイントポリシーを VPC エンドポイントにアタッチして (サービス名 `com.amazonaws.Region.appmesh-envoy-management` を使用するなど)、App Mesh へのアクセスを制御することはサポートされていません。

その他の考慮事項と制限事項については、「インターフェイスエンドポイントのアベイラビリティーゾーンに関する考慮事項」と「インターフェイスエンドポイントのプロパティと制限」を参照してください。

App Mesh のインターフェイス VPC エンドポイントの作成

App Mesh サービスのインターフェース VPC エンドポイントを作成するには、「Amazon VPC ユーザーガイド」の「インターフェースエンドポイント」の作成手順を使用してください。Envoy プロキシで App Mesh のパブリック Envoy 管理サービスに接続するためのサービス名として `com.amazonaws.Region.appmesh-envoy-management` を、メッシュ操作用に `com.amazonaws.Region.appmesh` を指定します。

> **Note**
>
> `######`、App Mesh AWS がサポートするリージョン (米国東部 (オハイオ) リージョンなど`us-east-2`) のリージョン識別子を表します。

App Mesh がサポートされているリージョンでは、App Mesh のインターフェイス VPC エンドポイントを定義できますが、各リージョンのすべてのアベイラビリティーゾーンのエンドポイントを定義できない場合があります。リージョンのインターフェイス VPC エンドポイントでどのアベイラビリティーゾーンがサポートされているかを確認するには、describe-vpc-endpoint-services コマンドを使用するか、を使用します。 AWS Management Consoleたとえば、次のコマンドは、米国東部 (オハイオ) リージョン内の App Mesh インターフェイス VPC エンドポイントをデプロイできるアベイラビリティゾーンを返します：

```
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 Mesh での耐障害性

AWS のグローバルインフラストラクチャは AWS リージョンとアベイラビリティーゾーンを中心に構築されます。AWSリージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立し隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性に優れています。

App Mesh は、高可用性を確保するために、複数のアベイラビリティーゾーンで実行し、高可用性を確保しています。App Mesh は、異常なコントロールプレーンインスタンスを自動的に検出して置換します。また、バージョンアップやパッチ適用を自動的に行います。

AWS App Mesh での災害対策

App Mesh サービスは、顧客データのバックアップを管理します。バックアップを管理するためには、何もする必要はありません。バックアップされたデータは暗号化されます。

での設定と脆弱性の分析 AWS App Mesh

App Mesh は、マイクロサービスと共にデプロイするマネージド Envoy プロキシ Docker コンテナイメージを発行します。App Mesh は、コンテナイメージに最新の脆弱性およびパフォーマンスパッ

チが確実に適用されるようにします。App Mesh は、イメージを利用できるようにする前に、App Mesh 機能セットに対して新しい Envoy プロキシリリースをテストします。

更新されたコンテナイメージのバージョンを使用するには、マイクロサービスを更新する必要があります。次は、イメージの最新バージョンです。

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.29.5.0-prod
```

App Mesh のトラブルシューティング

この章では、トラブルシューティングのベストプラクティスと App Mesh の使用時に発生する可能性のある一般的な問題について説明します。次の領域のいずれかを選択して、その領域のベストプラクティスと一般的な問題を確認します。

トピック

- App Mesh のトラブルシューティングのベストプラクティス
- App Mesh 設定のトラブルシューティング
- App Mesh 接続のトラブルシューティング
- App Mesh スケーリング
- App Mesh の可観測性
- App Mesh セキュリティのトラブルシューティング
- App Mesh Kubernetes のトラブルシューティング

App Mesh のトラブルシューティングのベストプラクティス

このトピックのベストプラクティスに従って、App Mesh を使用する際の問題をトラブルシューティングするようお勧めします。

Envoy プロキシ管理インターフェイスを有効にする

Envoy プロキシには、設定と統計の検出、および Connection Draining などのその他の管理機能を実行するために使用できる管理インターフェイスが付属しています。詳細については、Envoy ドキュメント「管理インターフェイス」を参照してください。

マネージド Envoy イメージ を使用する場合、管理エンドポイントは、デフォルトでポート 9901 が有効化されています。App Mesh 設定のトラブルシューティング の例では、管理エンドポイント URL は、`http://my-app.default.svc.cluster.local:9901/` のように表示されます。

> **Note**
>
> 管理エンドポイントは、パブリックインターネットに公開されることはありません。さらに、`ENVOY_ADMIN_ACCESS_LOG_FILE` 環境変数によって、デフォルトで `/tmp/`

`envoy_admin_access.log` に設定されている管理エンドポイントログをモニタリングするようお勧めします

メトリクスオフロードの Envoy DogStatsD 統合を有効にする

Envoy プロキシは、OSI レイヤー 4 およびレイヤー 7 のトラフィックおよび内部プロセスのヘルスの統計情報をオフロードするように設定できます。このトピックでは、メトリクスや Prometheus などのシンクに CloudWatch メトリクスをオフロードせずにこれらの統計を使用する方法について説明します。これらの統計をすべてのアプリケーションの一元管理された場所に配置すると、問題の診断や動作の迅速な確認に役立ちます。詳細については、「Amazon CloudWatch メトリクスの使用」および Prometheus ドキュメント「」を参照してください。

DogStatsD メトリクスを設定するには、 で定義されているパラメータを設定しますDogStatsD 変数。 DogStatsD の詳細については、DogStatsD のドキュメントを参照してください。AWS CloudWatch メトリクスへのメトリクスオフロードのデモンストレーションは、「」の「Amazon ECS の基本」チュートリアルで確認できます GitHub。

アクセスログの有効化

仮想ノード と 仮想ゲートウェイ でアクセスログを有効にして、アプリケーション間のトラフィックの推移の詳細を確認するようお勧めします。詳細については、Envoy ドキュメントに記載の「アクセスログ」を参照してください。ログには、OSI レイヤー 4 およびレイヤー 7 のトラフィック動作に関する詳細情報が表示されます。Envoy のデフォルト形式を使用すると、次の解析ステートメントを使用して CloudWatch Logs Insights でアクセスログを分析できます。

```
parse @message "[*] \"* * *\" * * * * * * * * * * *" as StartTime, Method, Path,
 Protocol, ResponseCode, ResponseFlags, BytesReceived, BytesSent, DurationMillis,
 UpstreamServiceTimeMillis, ForwardedFor, UserAgent, RequestId, Authority, UpstreamHost
```

本番稼働前の環境で、Envoy デバッグログを有効にする

本番稼働前の環境では、Envoy プロキシのログレベルを debug に設定するようお勧めします。デバッグログは、関連する App Mesh 設定を本番稼働環境に移行する前に、問題を特定する役に立ちます。

Envoy イメージ使用している場合、ログレベルを `ENVOY_LOG_LEVEL` 環境変数で debug に設定できます。

> **Note**
>
> 本番稼働環境での debug レベルの使用はお勧めしていません。レベルを に設定すると、ログdebugが増加し、CloudWatch ログ などのソリューションにオフロードされるログのパフォーマンスと全体のコストに影響する可能性があります。

Envoy のデフォルト形式を使用すると、次の解析ステートメントを使用して CloudWatch Logs Insights でプロセスログを分析できます。

```
parse @message "[*][*][*][*] [*] *" as Time, Thread, Level, Name, Source, Message
```

App Mesh コントロールプレーンで Envoy プロキシ接続を監視する

Envoy メトリクス `control_plane.connected_state` を監視し、Envoy プロキシが App Mesh コントロールプレーンと通信して動的設定リソースを取得しているかを確認することをお勧めします。詳細については、「Management Server」を参照してください。

App Mesh 設定のトラブルシューティング

このトピックでは、App Mesh 設定で発生する可能性のある一般的な問題の詳細を説明します。

Envoy コンテナイメージをプルできない

症状

Amazon ECS タスクで、次のエラーメッセージが表示されます。次のメッセージの Amazon ECR *#### ID* と*#####*は、コンテナイメージを取得した Amazon ECR リポジトリによって異なる場合があります。

```
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'
```

解決方法

このエラーは、使用されているタスク実行ロールに Amazon ECR と通信するアクセス許可がなく、リポジトリから Envoy コンテナイメージをプルできないことを示しています。Amazon ECS タスクに割り当てられたタスク実行ロールには、次のステートメントを含む IAM ポリシーが必要です。

```
{
  "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"
}
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

App Mesh Envoy 管理サービスに接続できない

症状

Envoy プロキシが App Mesh Envoy 管理サービスに接続できません。次が表示されます：

- 接続がエラーを拒否しました
- 接続タイムアウトしました
- App Mesh Envoy 管理サービスエンドポイントの解決中にエラーが発生しました
- gRPC エラー

解決方法

Envoy プロキシがインターネットまたはプライベートVPC エンドポイントにアクセスできること、およびセキュリティグループがポート 443 でのアウトバウンドトラフィックを許可していることを確認してください。App Mesh の公開 Envoy 管理サービスのエンドポイントは、完全修飾ドメイン名 (FQDN、Fully Qualified Domain Name) フォーマットに従います。

```
# App Mesh Production Endpoint
appmesh-envoy-management.Region-code.amazonaws.com

# App Mesh Preview Endpoint
```

```
appmesh-preview-envoy-management.Region-code.amazonaws.com
```

次のコマンドを使用して、EMS への接続をデバッグできます。これにより、有効だが空の gRPC 要求が Envoy 管理サービスに送信されます。

```
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
```

これらのメッセージを受け取った場合、Envoy Management Service への接続は機能しています。gRPC 関連のエラーのデバッグについては、「Envoy が エラーテキストで App Mesh Envoy 管理サービスから切断されました」を参照してください。

```
grpc-status: 16
grpc-message: Missing Authentication Token
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

Envoy がエラーテキストで App Mesh Envoy 管理サービスから切断されました

症状

Envoy プロキシは、App Mesh Envoy 管理サービスへの接続して設定を受信することができません。Envoy プロキシログには、次のようなログエントリが含まれています。

```
gRPC config stream closed: gRPC status code, message
```

解決方法

ほとんどの場合、ログのメッセージ部分は問題を示しているはずです。次の表に、表示される可能性のある最も一般的な gRPC ステータスコード、その原因、および解決策を示します。

gRPC ステータスコード	原因	解決方法
0	Envoy 管理サービスから正常に切断します。	問題はありません。App Mesh は、このステータスコードで Envoy プロキシを切断する

gRPC ステータスコード	原因	解決方法
		ことがあります。Envoy は再接続し、更新を受信し続けます。
3	メッシュエンドポイント (仮想ノードまたは仮想ゲートウェイ) 、または関連するリソースの 1 つが見つかりませんでした。	Envoy 設定を再チェックして、それが表す App Mesh リソースの適切な名前があることをチェックします。App Mesh リソースが、AWS Cloud Map 名前空間やACM証明書などの他の AWS リソースと統合されている場合、それらのリソースが存在することを確認します。
7	Envoy プロキシは、Envoy 管理サービスへの接続や関連リソースの取得などのアクションの実行を許可されていません。	App Mesh やその他のサービスに適切なポリシーステートメントを持つIAMポリシーを作成し、Envoy プロキシが Envoy 管理サービスに接続するために使用している IAM ユーザーまたはロールに、そのポリシーをアタッチしていることを確認してください。
8	特定の App Mesh リソースの Envoy プロキシの数がアカウントレベルのサービスクォータを超えています。	デフォルトのアカウントクォータの詳細および引き上げをリクエストする方法については、「App Mesh Service Quotas」を参照してください。

gRPC ステータスコード	原因	解決方法
16	Envoy プロキシには、AWS の有効な認証資格情報がありません。	Envoy に接続する適切な資格情報があることを確認しますAWSIAM ユーザーまたはロールを介したサービス。Envoy のバージョン `v1.24` 以前の既知の問題 #24136 では、Envoy プロセスが `1024` 個を超えるファイル記述子を使用すると認証情報の取得に失敗します。これは Envoy が大量のトラフィックを処理している場合に発生します。デバッグレベルで Envoy ログのテキスト「`A libcurl function was given a bad argument`」を確認することで、この問題を確認できます。この問題を軽減するには、Envoy バージョン `v1.25.1.0-prod` 以降にアップグレードしてください。

次のクエリを使用して、Amazon CloudWatch Insights で Envoy プロキシのステータスコードとメッセージを確認できます。

```
filter @message like /gRPC config stream closed/
| parse @message "gRPC config stream closed: *, *" as StatusCode, Message
```

表示されたエラーメッセージが役に立たなかったり、問題が解決しない場合は、GitHub 問題を開くことを検討してください。

Envoy コンテナのヘルスチェック、準備状態プローブ、またはライブネスプローブの失敗

症状

Envoy プロキシが Amazon ECS タスク、Amazon EC2 インスタンス、Kubernetes ポッドでヘルスチェックに失敗しています。例えば、次のコマンドを使用して Envoy 管理インターフェースをクエリし、LIVE 以外のステータスを受け取ります。

```
curl -s http://my-app.default.svc.cluster.local:9901/server_info | jq '.state'
```

解決方法

Envoy プロキシによって返されるステータスに応じた修復手順の一覧を次に示します。

- PRE_INITIALIZING または INITIALIZING — Envoy プロキシがまだ設定を受信していないか、まだ接続できないため App Mesh Envoy 管理サービスから設定を取得できない。Envoy 管理サービスから接続しようとしたときに、 Envoy がエラーを受信している可能性があります。詳細については、「Envoy がエラーテキストで App Mesh Envoy 管理サービスから切断されました」を参照してください。
- DRAINING — Envoy プロキシは、Envoy 管理インターフェースの /healthcheck/fail または /drain_listeners リクエストに応答して接続のドレインを開始しました。Amazon ECS タスク、Amazon EC2 インスタンス、または Kubernetes ポッドを終了する場合を除き、管理インターフェイスでこれらのパスを呼び出すことはお勧めしません。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

ロードバランサーからメッシュエンドポイントへのヘルスチェックが失敗している

症状

メッシュエンドポイントは、コンテナヘルスチェックまたは準備プローブによって正常と見なされますが、ロードバランサーからメッシュエンドポイントへのヘルスチェックが失敗しています。

解決方法

この問題を解決するには、次のタスクを完了します。

- メッシュエンドポイントに関連するセキュリティグループが、ヘルスチェック用に設定したポートのインバウンドトラフィックを受け入れていることを確認してください。
- 手動で要求された場合、ヘルスチェックが一貫して成功していることを確認します。例えば、VPC 内の踏み台ホストです。
- 仮想ノードのヘルスチェックを設定する場合は、アプリケーションにヘルスチェックエンドポイントを実装することをお勧めします。例えば、HTTP の場合は /ping などです。これにより、Envoy プロキシとアプリケーションの両方がロードバランサーからルーティング可能になります。
- 必要な機能に応じて、仮想ノードには任意の Elastic Load Balancer タイプを使用できます。詳細については、「Elastic Load Balancing の機能」を参照してください。
- 仮想ゲートウェイのヘルスチェックを設定する場合は、仮想ゲートウェイのリスナーポートに TCP または TLS ヘルスチェックを備えたネットワークロードバランサーを使用することをお勧めします。これにより、仮想ゲートウェイリスナーがブートストラップされ、接続を受け入れる準備ができていることが保証されます。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

仮想ゲートウェイがポート 1024 以下のトラフィックを受け入れない

症状

仮想ゲートウェイは、ポート 1024 以下のトラフィックを受け入れませんが、1024 より大きいポート番号のトラフィックを受け入れます。例えば、次のコマンドを使用して Envoy 統計をクエリし、ゼロ以外の値を受け取ります。

```
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "update_rejected"
```

特権ポートへのバインド障害を示す、次のテキストのようなテキストがログに、表示される場合があります。

```
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
```

解決方法

この問題を解決するには、ゲートウェイに指定したユーザーに linux 機能 CAP_NET_BIND_SERVICE が必要です。詳細については、「Linux プログラマーズマニュアル」の「機能」、「ECS タスク定義パラメータ」の「Linux パラメータ」、および Kubernetes ドキュメントの「コンテナの機能の設定」を参照してください。

> **Important**
>
> Fargate は 1024 より大きいポート値を使用する必要があります。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

App Mesh 接続のトラブルシューティング

このトピックでは、App Mesh 接続で発生する可能性のある一般的な問題の詳細を説明します。

仮想サービスの DNS 名を解決できません

症状

アプリケーションは、接続しようとしている仮想サービスの DNS 名を解決できません。

解決方法

これは既知の問題です。詳細については、「任意のホスト名または FQDN の問題 VirtualServices による名前 GitHub 」を参照してください。App Mesh の仮想サービスには任意の名前を付けることができます。仮想サービス名のDNS A レコードがあり、アプリケーションが仮想サービス名を解決できる限り、リクエストは Envoy によってプロキシされ、適切な宛先にルーティングされます。この問題を解決するには、仮想サービス名の `10.10.10.10` などの非ループバック IP アドレスにDNS A レコードを追加します。DNS A レコードは、次の条件で使用できます。

- プライベートホストゾーン名の末尾に名前が付いている場合には、Amazon Route 53 内
- アプリケーションコンテナの `/etc/hosts` ファイル内
- 管理するサードパーティー DNS サーバー内

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

仮想サービスのバックエンドに接続できない

症状

アプリケーションは、仮想ノードのバックエンドとして定義された仮想サービスへの接続を確立できません。接続を確立しようとすると、接続が完全に失敗したり、アプリケーションの観点から見たリクエストが HTTP 503 レスポンスコードで失敗する可能性があります。

解決方法

アプリケーションがまったく接続に失敗した場合 (HTTP 503 レスポンスコードが返されない場合) は、次の手順を実行します。

- コンピューティング環境が App Mesh で動作するように設定されていることを確認してください。
  - Amazon ECS の場合は、適切なプロキシ設定が有効になっていることを確認してください。end-to-end チュートリアルについては、「App Mesh と Amazon ECS の開始方法」を参照してください。
  - Amazon EKS を含む Kubernetes については、Helm を介して最新のApp Mesh コントローラーがインストールされていることを確認してください。詳細については、Helm Hub の「App Mesh コントローラー」または「チュートリアル: Kubernetes とのApp Mesh 統合を設定する」を参照してください。
  - Amazon EC2 の場合は、App Mesh トラフィックをプロキシするための Amazon EC2 インスタンスを設定していることを確認してください。詳細については、「サービスの更新」を参照してください。
- コンピューティングサービスで実行されている Envoy コンテナが App Mesh Envoy 管理サービスに正常に接続されていることを確認してください。Envoy 統計情報の `control_plane.connected_state` フィールドを確認することで、この問題を確認できます。`control_plane.connected_state` の詳細については、「トラブルシューティングのベストプラクティス」の「Envoy プロキシ接続を監視する」を参照してください。

  Envoy が最初は接続を確立できたが、その後切断され再接続されなかった場合は、「Envoy がエラーテキストで App Mesh Envoy 管理サービスから切断されました」を参照して、接続が切断された理由を確認してください。

アプリケーションが接続してもリクエストが HTTP 503 レスポンスコードで失敗する場合は、次のことを試してください。

- 接続する仮想サービスがメッシュに存在することを確認してください。
- 仮想サービスにプロバイダー (仮想ルーターまたは仮想ノード) があることを確認してください。
- Envoy を HTTP プロキシとして使用しているときに、Envoy の統計情報で、正しい宛先ではなく `cluster.cds_egress_*_mesh-allow-all` への出力トラフィックが見られる場合は、Envoy が `filter_chains` 経由でリクエストを適切にルーティングしていない可能性があります。これは、修飾されていない仮想サービス名を使用したことが原因である可能性があります。Envoy プロキシは他の仮想サービスと名前で通信するため、実際のサービスのサービス検出名を仮想サービス名として使用することをお勧めします。

  詳細については、「仮想サービス」を参照してください。
- Envoy プロキシログで、次のエラーメッセージがないか調べます。
  - `No healthy upstream` — Envoy プロキシがルートしようとしている仮想ノードに、解決済みのエンドポイントがないか、正常なエンドポイントがありません。ターゲット仮想ノードに正しいサービスディスカバリとヘルスチェック設定があることを確認してください。

    バックエンド仮想サービスのデプロイまたはスケーリング中にサービスへのリクエストが失敗した場合は、仮想サービスに仮想ノードプロバイダーがある場合、一部のリクエストが失敗して、HTTP ステータスコード 503 を表示する のガイダンスに従ってください。
  - `No cluster match for URL` — これは、リクエストが、仮想ルーター)プロバイダで定義されたどのルートで定義されている基準にも一致しない、仮想サービスに送信された場合に発生する可能性が多々あります。パスと HTTP リクエストヘッダーが正しいことを確認して、アプリケーションからのリクエストがサポートされているルートに送信されていることを確認してください。
  - `No matching filter chain found` — これは、リクエストが無効なポート上の仮想サービスに送信された場合に発生する可能性が多々あります。アプリケーションからの要求が、仮想ルーターで指定された同じポートを使用していることを確認してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

外部サービスに接続できない

症状

アプリケーションがメッシュ外のサービスに接続できません。例えば、`amazon.com`。

解決方法

デフォルトでは、App Mesh は、メッシュ内のアプリケーションからメッシュ外の宛先へのアウトバウンドトラフィックを許可しません。外部サービスとの通信を有効にするには、次の 2 つのオプションがあります。

- メッシュリソースのアウトバウンドフィルターをALLOW_ALL に設定します。この設定により、メッシュ内のすべてのアプリケーションは、メッシュの内外にあるすべての宛先 IP アドレスと通信を許可されます。
- 仮想サービス、仮想ルーター、ルート、および仮想ノードを使用して、メッシュ内の外部サービスをモデル化します。例えば、外部サービス example.com をモデル化するには、仮想ルーターとルートを使用して example.com という名前の仮想サービスを作成し、DNS サービスディスカバリホスト名が example.com の仮想ノードに、すべてのトラフィックを送信します。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

MySQL または SMTP サーバーに接続できない

症状

SMTP サーバーや仮想ノード定義を使用する MySQL データベースなど、すべての宛先 (メッシュ EgressFilter type=ALLOW_ALL) へのアウトバウンドトラフィックを許可すると、アプリケーションからの接続が失敗します。例として、MySQL サーバーへの接続を試みたときのエラーメッセージを次に示します。

```
ERROR 2013 (HY000): Lost connection to MySQL server at 'reading initial communication
 packet', system error: 0
```

解決方法

この問題は既知の問題であり、App Meshのイメージバージョン1.15.0以降を使用することで解決します。詳細については、「App Mesh で MySQL に接続 GitHub できない」の問題を参照してください。このエラーは、App Mesh で設定したEnvoy の送信リスナーが Envoy TLS Inspector のリスナーフィルターを追加するため発生します。詳細については、Envoy ドキュメントの「TLS Inspector」を参照してください。を参照してください。このフィルターは、クライアントから送信された最初のパケットを調べて、接続が TLS を使用しているかどうかを評価します。ただし、MySQL と SMTP では、サーバーは接続後に最初のパケットを送信します。MySQL の詳細については、MySQL ドキュメントの「初期ハンドシェイク」を参照してください。サーバーが最初のパケットを送信するため、フィルターでの検査は失敗します。

Envoy のバージョンに応じて、この問題を回避するには:

- App Mesh イメージ Envoy のバージョンが 1.15.0 以降の場合は、MySQL、SMTP、MSSQLなどの外部サービスをアプリケーションの仮想ノードのバックエンドとしてモデル化しないでください。
- App Mesh イメージの Envoy のバージョンが 1.15.0 以前の場合は、MySQL のサービスの `APPMESH_EGRESS_IGNORED_PORTS` の値リストに、STMP に使用しているポートとして、ポート `3306` を追加します。

> **Important**
>
> デフォルトの SMTP ポートは 25、587、465 ですが、使用しているポートのみを `APPMESH_EGRESS_IGNORED_PORTS` にに追加する必要があり、3つすべてを追加する必要はありません。

詳細については、Kubernetes の「更新サービス」、Amazon ECS の「更新サービス」、または Amazon EC2 の「更新サービス」を参照してください。

それでも問題が解決しない場合は、既存のGitHub 問題を使用して発生している内容の詳細をお知らせください。または、 AWS サポートにお問い合わせください。

App Mesh で TCP 仮想ノードまたは仮想ルーターとしてモデル化されたサービスに接続できない

症状

アプリケーションは、App Mesh PortMapping 定義の TCP プロトコル設定を使用するバックエンドに接続できません。

解決方法

これは既知の問題です。詳細については、「 の同じポート上の複数の TCP 宛先へのルーティング」を参照してください GitHub。現在、App Mesh では、OSI レイヤー 4 で Envoy プロキシに提供される情報の制限により、TCP としてモデル化された複数のバックエンド宛先を同じポートを共有することは許可されていません。TCP トラフィックがすべてのバックエンド送信先で適切にルーティングされるようにするには、次の手順を実行します。

- すべての宛先が一意のポートを使用していることを確認してください。バックエンド仮想サービスに仮想ルータープロバイダーを使用している場合は、ルーティング先の仮想ノードのポートを変更せずに、仮想ルーターポートを変更できます。これにより、Envoy プロキシが仮想ノードで定義されたポートを引き続き使用する間、アプリケーションは仮想ルーターのポートで接続を開くことができます。
- TCP としてモデル化された送信先が MySQL サーバー、または接続後にサーバが最初のパケットを送信するその他の TCP ベースのプロトコルである場合は、「MySQL または SMTP サーバーに接続できない」を参照してください。

それでも問題が解決しない場合は、既存のGitHub 問題を使用して発生している内容の詳細をお知らせください。または、 AWS サポートにお問い合わせください。

仮想ノードの仮想サービスバックエンドとしてリストされていないサービスへの接続に成功する

症状

アプリケーションは、仮想ノードの仮想サービスバックエンドとして指定されていない送信先に接続してトラフィックを送信できます。

解決方法

App Mesh API でモデル化されていない送信先へのリクエストが成功した場合、メッシュのアウトバウンドフィルタータイプが `ALLOW_ALL` に設定されていることが最も可能性の高い原因となります。アウトバウンドフィルターが `ALLOW_ALL` に設定されている場合、アプリケーションからのアウトバウンドリクエストのうち、モデル化された送信先 (バックエンド) に一致しないものは、アプリケーションが設定した送信先 IP アドレスに送信されることになります。

メッシュでモデル化されていない宛先へのトラフィックを許可しない場合は、アウトバウンドフィルターの値を`DROP_ALL` に設定することを検討してください。。

> **Note**
>
> メッシュアウトバウンドフィルター値を設定すると、メッシュ内のすべての仮想ノードに影響します。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

仮想サービスに仮想ノードプロバイダーがある場合、一部のリクエストが失敗して、 HTTP ステータスコード 503 を表示する

症状

アプリケーションのリクエストの一部は、仮想ルータープロバイダーではなく仮想ノードプロバイダーを使用している仮想サービスバックエンドで失敗します。仮想サービスに仮想ルータープロバイダーを使用する場合、リクエストは失敗しません。

解決方法

これは既知の問題です。詳細については、「 の仮想サービスの仮想ノードプロバイダーでポリシーを再試行する」を参照してください GitHub。仮想ノードを仮想サービスのプロバイダーとして使用する場合、仮想サービスのクライアントが使用するデフォルトの再試行ポリシーを指定することはできません。これに対し、仮想ルーターのプロバイダーでは、リトライポリシーは子ルートリソースのプロパティであるため、指定することが可能です。

仮想ノードプロバイダーへのリクエストの失敗を減らすには、代わりに仮想ルーターのプロバイダーを使用し、そのルートで再試行ポリシーを指定します。アプリケーションへのリクエストの失敗を減らすその他の方法については、「App Mesh のベストプラクティス」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

Amazon EFS ファイルシステムに接続できない

症状

Amazon EFS ファイルシステムをボリュームとして Amazon ECS タスクを設定すると、次のエラーでタスクは失敗し始めます。

```
ResourceInitializationError: failed to invoke EFS utils commands to set up EFS volumes:
 stderr: mount.nfs4: Connection refused : unsuccessful EFS utils command execution;
 code: 32
```

解決方法

これは既知の問題です。このエラーは、タスク内のコンテナが開始される前に Amazon EFS への NFS 接続が発生するために発生します。このトラフィックは、プロキシ設定によって Envoy にルー

ティングされますが、この時点では実行されません。スタートアップの順序が原因で、NFS クライアントは Amazon EFS ファイルシステムへの接続に失敗し、タスクの起動に失敗します。この問題を解決するには、Amazon ECS タスク定義のプロキシ設定の `EgressIgnoredPorts` 設定値リストにポート `2049` を追加します。詳細については、「プロキシ設定ファイル」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

接続は正常にサービスされるが、受信リクエストが Envoy のアクセスログ、トレース、またはメトリクスに表示されない

症状

アプリケーションが別のアプリケーションに接続してリクエストを送信できる場合でも、アクセスログまたは Envoy プロキシのトレース情報で受信リクエストを表示できません。

解決方法

これは既知の問題です。詳細については、GitHub issue の「iptables ルールのセットアップ」を参照してください。Envoy プロキシは、対応する仮想ノードがリッスンしているポートへのインバウンドトラフィックのみをインターセプトします。他のポートへのリクエストは、Envoy プロキシをバイパスし、その背後にあるサービスに直接到達します。Envoy プロキシがサービスのインバウンドトラフィックをインターセプトできるようにするには、仮想ノードとサービスを同じポートでリッスンするように設定する必要があります。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

コンテナレベルで 設定方法 **HTTP_PROXY / HTTPS_PROXY** 環境変数を設定すると、期待どおりに動作しません。

症状

HTTP_PROXY / HTTPS_PROXY が次の環境変数として設定されている場合：

- App Mesh が有効になっているタスク定義のアプリケーションのコンテナでは、App Mesh サービスの名前空間に送信されるリクエストは、Envoy サイドカーから `HTTP 500` エラーレスポンスを受け取ります。

- App Mesh が有効になっているタスク定義の Envoy コンテナでは、Envoy サイドカーから出るリクエストが HTTP / HTTPS プロキシサーバーを通らず、環境変数が動作しなくなります。

解決方法

アプリケーションコンテナの場合:

App Mesh は、タスク内のトラフィックが Envoy プロキシを通過することによって機能します。HTTP_PROXY/HTTPS_PROXY設定は、コンテナのトラフィックが別の外部プロキシを通過するように設定することで、この動作を上書きします。トラフィックは、引き続き Envoy によってインターセプトされますが、外部プロキシを使用したメッシュトラフィックのプロキシはサポートされません。

メッシュ以外のすべてのトラフィックをプロキシする場合は、次の例のように、メッシュの CIDR / 名前空間、ローカルホスト、および認証情報のエンドポイントを含めるように NO_PROXY を設定してください。

```
NO_PROXY=localhost,127.0.0.1,169.254.169.254,169.254.170.2,10.0.0.0/16
```

Envoy コンテナの場合:

Envoy では汎用プロキシはサポートされていません。これらの変数を設定することはお勧めしません。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

ルートのタイムアウトを設定した後でも、アップストリームのリクエストがタイムアウトします。

症状

次のタイムアウトを定義しました。

- ルートですが、アップストリームリクエストのタイムアウトエラーがまだ発生しています。
- 仮想ノードリスナーとタイムアウト、およびルートの再試行タイムアウトですが、アップストリームリクエストのタイムアウトエラーが発生しています。

解決方法

15 秒を超える高レイテンシーのリクエストが正常に完了するには、ルートと仮想ノードのリスナーレベルの両方でタイムアウトを指定する必要があります。

デフォルトの 15 秒より大きいルートのタイムアウトを指定する場合は、すべての参加仮想ノードのリスナーにもタイムアウトが指定されていることを確認してください。ただし、タイムアウトをデフォルトより低い値に減らすと、仮想ノードのタイムアウトを更新することはオプションとなります。仮想ノードとルートを設定するときのオプションの詳細については、「仮想ノード」と「ルート」を参照してください。

再試行ポリシーを指定した場合、リクエストタイムアウトに指定する時間は、常に再試行タイムアウトに再試行ポリシーで定義した最大再試行回数を掛けた値以上である必要があります。これにより、すべての再試行でのリクエストが正常に完了します。詳細については、「ルート」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

Envoy が HTTP Bad request で応答する。

症状

Envoy は、Network Load Balancer (NLB) を介して送信されたすべてのリクエストに対して HTTP 400 Bad request で応答します。Envoy のログを確認すると、次のことがわかります。

- デバッグログ:

```
dispatch error: http/1.1 protocol error: HPE_INVALID_METHOD
```

- アクセスログ:

```
"- - HTTP/1.1" 400 DPE 0 11 0 - "-" "-" "-" "-" "-"
```

解決方法

解決策は、NLB のターゲットグループ属性でプロキシプロトコルバージョン 2 (PPv2) を無効にすることです。

現在のところ、PPv2は、App Mesh コントロールプレーンを使用して実行される仮想ゲートウェイと仮想ノード Envoy ではサポートされていません。Kubernetes の AWS ロードバランサーコント

ローラーを使用して NLB をデプロイする場合は、次の属性を `false` に設定して PPv2 を無効にします。

```
service.beta.kubernetes.io/aws-load-balancer-target-group-attributes:
 proxy_protocol_v2.enabled
```

NLB リソース属性の詳細については、「AWS Load Balancer Controller のアノテーション」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

タイムアウトを適切に設定できない。

症状

仮想ノードリスナーのタイムアウトと、仮想ノードバックエンドへのルートのタイムアウトを設定した後でも、リクエストは 15 秒以内にタイムアウトします。

解決方法

バックエンドリストに正しい仮想サービスが含まれていることを確認してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

App Mesh スケーリング

このトピックでは、App Mesh のスケーリングで発生する可能性のある一般的な問題を詳細に説明します。

仮想ノード/仮想ゲートウェイの 50 レプリカを超えてスケーリングすると、接続が失敗し、コンテナのヘルスチェックが失敗する

症状

仮想ノード/仮想ゲートウェイの Amazon ECS タスク、Kubernetes ポッド、Amazon EC2 インスタンスなどのレプリカの数を 50 個を超えてスケールすると、新規および現在実行中の Envoy の Envoy コンテナヘルスチェックが失敗し始めます。仮想ノード/仮想ゲートウェイにトラフィックを

送信するダウンストリームアプリケーションは、HTTP ステータスコード 503 でリクエストの失敗を確認し始めます。

解決方法

仮想ノード/仮想ゲートウェイあたりのエンボイ数に対する App Mesh のデフォルトのクォータは 50 です。実行中の Envoys の数がこのクォータを超えると、新規で現在実行中の Envoy は、gRPC ステータスコード 8 (RESOURCE_EXHAUSTED)を使用して App Mesh の Envoy 管理サービスに接続できません。このクォータは、引き上げることができます。詳細については、「App Mesh Service Quotas」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

仮想サービスバックエンドが水平方向にスケールアウトまたはスケールインする場合、リクエストが 503 で失敗する

症状

バックエンド仮想サービスが水平方向にスケールアウトまたはスケールインされると、ダウンストリームアプリケーションからのリクエストは失敗し、HTTP 503 ステータスコードを表示します。

解決方法

App Mesh では、アプリケーションを水平方向にスケーリングしながら、障害発生を緩和するためのいくつかのアプローチを推奨しています。これらの障害を防ぐ方法の詳細については、「App Mesh のベストプラクティス」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

ロードが増加すると、Envoy コンテナがセグメンテーション違反でクラッシュする

症状

トラフィックのロードが高い場合、セグメンテーション違反 (Linux 終了コード 139) により Envoy プロキシがクラッシュします。Envoy プロセスログには、次のようなステートメントが含まれています。

```
Caught Segmentation fault, suspect faulting address 0x0"
```

解決方法

Envoy プロキシは、オペレーティングシステムのデフォルトの nofile ulimit に違反している可能性があります。これは、プロセスが一度に開くことができるファイル数の制限です。この違反は、トラフィックがより多くの接続を引き起こし、追加のオペレーティングシステムソケットを消費することが原因です。この問題を解決するには、ホストオペレーティングシステムで ulimit nofile 値を増やします。Amazon ECS を使用している場合は、この制限は、タスク定義のリソース制限設定のUlimit設定を介して変更できます。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

デフォルトリソースの増加がサービスの制限に反映されない

症状

App Mesh リソースのデフォルト制限を増やした後、サービスの制限を確認しても新しい値は反映されません。

解決方法

新しい制限は、現在表示されていませんが、お客様は引き続きそれらを行使できます。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

大量のヘルスチェックコールが原因でアプリケーションがクラッシュする

症状

仮想ノードのアクティブなヘルスチェックを有効にすると、ヘルスチェックのコール数が増加します。アプリケーションに対して行われるヘルスチェックコールのボリュームが大幅に増加したため、アプリケーションがクラッシュします。

解決方法

アクティブなヘルスチェックが有効な場合、ダウンストリーム (クライアント) の各 Envoy エンドポイントは、ルーティングを決定するために、アップストリームのクラスター (サーバー) の各エ

ンドポイントにヘルスリクエストを送信します。その結果、ヘルスチェックのリクエストの総数は、`number of client Envoys * number of server Envoys * active health check frequency` になります。

この問題を解決するには、ヘルスチェックのプローブの頻度を変更します。これにより、ヘルスチェックプローブの総ボリュームが減少します。App Mesh では、アクティブなヘルスチェックに加えて、パッシブヘルスチェックの手段として外れ値の検出を設定できます。外れ値検出を使用して、連続した `5xx` レスポンスに基づいて特定のホストを削除するタイミングを設定します。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

App Mesh の可観測性

このトピックでは、App Mesh の観測性で発生する可能性のある一般的な問題を詳細に説明します。

アプリケーションのト AWS X-Ray トレースを表示できない

症状

App Mesh のアプリケーションが X-Ray コンソールまたは API に X-Ray トレース情報を表示していません。

解決方法

App Mesh で X-Ray を使用するには、アプリケーション、サイドカーコンテナ、および X-Ray サービス間の通信を有効にするようにコンポーネントを正しく設定する必要があります。次の手順を実行して、X-Ray が正しく設定されていることを確認します。

- App Mesh 仮想ノードリスナーのプロトコルが `TCP` に設定されていないことを確認します。
- アプリケーションとともにデプロイされた X-Ray コンテナが UDP ポート `2000` を公開し、ユーザー `1337` として実行されることを確認します。詳細については、「」の「Amazon ECS X-Ray の例」を参照してください GitHub。
- Envoy コンテナでトレースが有効になっていることを確認します。App Mesh Envoy イメージを使用している場合、`ENABLE_ENVOY_XRAY_TRACING` 環境変数を `1` の値に設定し、`XRAY_DAEMON_PORT` 環境変数を `2000` に設定することで、X-Ray を有効にできます。
- 言語固有の SDK の1つを使用してアプリケーションコードに X-Ray を実装した場合は、言語のガイドに従って、正しく設定されていることを確認してください。

- 前の項目がすべて正しく設定されている場合は、X-Ray コンテナのログでエラーがないか確認し、AWS X-Ray のトラブルシューティングのガイダンスに従ってください。App Mesh での X-Ray 統合に関するより詳細な説明は、「X-Ray と App Mesh 統合」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

Amazon メトリクスにアプリケーションの Envoy CloudWatch メトリクスが表示されない

症状

App Mesh のアプリケーションは、Envoy プロキシによって生成されたメトリクスをメトリクスに CloudWatch出力していません。

解決方法

App Mesh で CloudWatch メトリクスを使用する場合は、Envoy プロキシ、 CloudWatch エージェントサイドカー、および CloudWatch メトリクスサービス間の通信を有効にするために、いくつかのコンポーネントを正しく設定する必要があります。Envoy プロキシの CloudWatch メトリクスが正しく設定されていることを確認するには、次のステップを実行します。

- App Mesh の CloudWatch エージェントイメージを使用していることを確認します。詳細については、「」の「App Mesh CloudWatch エージェント」を参照してください GitHub。
- プラットフォーム固有の使用手順に従って、App Mesh 用に CloudWatch エージェントを適切に設定していることを確認します。詳細については、「」の「App Mesh CloudWatch エージェント」を参照してください GitHub。
- 前の項目がすべて正しく設定されている場合は、 CloudWatch エージェントコンテナログにエラーがないか確認し、「エージェントのトラブルシューティング」に記載されているガイダンスに従ってください CloudWatch 。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

AWS X-Ray トレースのカスタムサンプリングルールを設定できない

症状

アプリケーションで X-Ray トレースを使用していますが、トレースのサンプリングルールを設定できません。

解決方法

App Mesh Envoy では、現在、X-Ray の動的サンプリングの設定がサポートされていないため、次の回避策を利用できます。

Envoy のバージョンが 1.19.1 以降の場合は、次のオプションがあります。

- サンプリングレートのみを設定するには、Envoy コンテナで XRAY_SAMPLING_RATE 環境変数を使用します。値は、 0 と 1.00 (100%) の間の10進数で指定する必要があります。詳細については、「AWS X-Ray 変数」を参照してください。
- X-Ray トレーサのローカライズされたカスタムサンプリングルールを設定するには、XRAY_SAMPLING_RULE_MANIFEST 環境変数を使用して、Envoy コンテナファイルシステムのファイルパスを指定します。詳細については、「AWS X-Ray デベロッパーガイド」の「サンプリングルール」を参照してください。

Envoyのバージョンが 1.19.1 以前の場合は、次を実行します。

- ENVOY_TRACING_CFG_FILE 環境変数を使用して、サンプリングレートを変更します。詳細については、「Envoy 設定変数」を参照してください。カスタムトレース設定を指定し、ローカルサンプリングルールを定義します。詳細については、「Envoy X-Ray Config」を参照してください。
- ENVOY_TRACING_CFG_FILE 環境変数のカスタムトレース設定の例：

```
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プロパティのサンプリングルールのマニフェスト設定の詳細については、「X-Ray SDK for Go の設定」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

App Mesh セキュリティのトラブルシューティング

このトピックでは、App Mesh セキュリティで発生する可能性のある一般的な問題を詳細に説明します。

TLS クライアントのポリシーを使用してバックエンド仮想サービスに接続できない

症状

TLS クライアントのポリシーを仮想ノードの仮想サービスバックエンドに追加すると、そのバックエンドへの接続が失敗します。バックエンドサービスにトラフィックを送信しようとすると、リクエストはHTTP 503 レスポンスコードとエラーメッセージ:upstream connect error or disconnect/reset before headers. reset reason: connection failure で失敗します。

解決方法

問題の根本原因を特定するには、問題の診断に役立つ Envoy プロキシプロセスログを使用することをお勧めします。詳細については、「本番稼働前の環境で、Envoy デバッグログを有効にする」を参照してください。次のリストを使用して、接続障害の原因を特定します。

- 仮想サービスのバックエンドに接続できない で説明されているエラーを除外して、バックエンドへの接続が成功していることを確認します。
- Envoy プロセスログで、次のエラー (デバッグレベルで記録される) を探します。

```
TLS error: 268435581:SSL routines:OPENSSL_internal:CERTIFICATE_VERIFY_FAILED
```

次の問題のいずれかが原因で、このエラーが発生します。

- 証明書は、TLS クライアントのポリシーの信頼バンドルで定義されている認証局の 1 つによって署名されていませんでした。
- 証明書は無効です(期限切れ)。
- サブジェクト別名 (SAN) がリクエストされた DNS ホスト名と一致しません。
- バックエンドサービスによって提供される証明書が有効であること、TLS クライアントポリシーの信頼バンドル内のいずれかの認証局によって署名されていること、および Transport Layer Security (TLS) で定義されている基準を満たしていることを確認します。
- 次のようなエラーが表示される場合は、リクエストが Envoy プロキシをバイパスしてアプリケーションに直接到達していることを意味します。トラフィックを送信しても、Envoy の統計は変化せず、Envoy がトラフィックを復号する経路上にいないことがわかります。仮想ノードのプロキシ設定で、AppPorts にアプリケーションがリッスンしている正しい値が含まれていることを確認してください。

```
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
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。セキュリティの脆弱性が見つかったと思われる場合、または App Mesh のセキュリティについて質問がある場合は、「AWS 脆弱性レポートガイドライン」を参照してください。

アプリケーションが TLS を発信しているときにバックエンド仮想サービスに接続できない

症状

Envoy プロキシからではなく、アプリケーションから TLS セッションを開始すると、バックエンド仮想サービスへの接続が失敗します。

解決方法

これは既知の問題です。詳細については、「機能リクエスト: ダウンストリームアプリケーションとアップストリームプロキシの問題間の TLS ネゴシ GitHub エーション」を参照してください。App Mesh では、TLS 発信は、現在 Envoy プロキシからサポートされていますが、アプリケーションからはサポートされていません。Envoy で TLS 発信 Support を使用するには、アプリケーションでの TLS 発信を無効にします。これにより、Envoy はアウトバウンドリクエストヘッダーを読

み取り、TLS のセッションを介してリクエストを適切な宛先に転送できます。詳細については、「Transport Layer Security (TLS)」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、AWS サポートにお問い合わせください。セキュリティの脆弱性が見つかったと思われる場合、または App Mesh のセキュリティについて質問がある場合は、「AWS 脆弱性レポートガイドライン」を参照してください。

Envoy プロキシ間の接続が TLS を使用していることをアサートできません

症状

アプリケーションが仮想ノードまたは仮想ゲートウェイのリスナーで TLS 終了、またはバックエンド TLS クライアントのポリシーで TLS 発信を有効にしていますが、TLS ネゴシエートされたセッションで Envoy プロキシ間の接続が発生していることをアサートできません。

解決方法

この解決法で定義されているステップは、Envoy 管理インターフェイスと Envoy 統計を使用します。これらの設定については、「Envoy プロキシ管理インターフェイスを有効にする」と「メトリクスオフロードの Envoy DogStatsD 統合を有効にする」を参照してください。次の統計例では、簡単にするために管理インターフェイスを使用しています。

- TLS 終了を実行する Envoy プロキシの場合:
  - 次のコマンドを使用して、TLS 証明書が Envoy 設定でブートストラップされていることを確認してください。

    ```
    curl http://my-app.default.svc.cluster.local:9901/certs
    ```

    返される出力には、TLSターミネーションで使用される証明書の `certificates[].cert_chain` の下に少なくとも1つのエントリが表示されます。
  - 次のコマンドと出力の例に示すように、プロキシのリスナーへの正常なインバウンド接続の数が、SSL ハンドシェイクの数に再利用された SSL セッションの数を加えたものと正確に同じであることを確認してください。

    ```
    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 発信を実行する Envoy プロキシの場合:
  - 次のコマンドを使用して、TLS 信頼ストアが Envoy 設定でブートストラップされていることを確認してください。

    ```
    curl http://my-app.default.svc.cluster.local:9901/certs
    ```

    TLS の発信中にバックエンドの証明書を検証する際に使用される証明書について、certificates[].ca_certs の下に少なくとも1つのエントリが表示されます。
  - 次のコマンドと出力の例に示すように、バックエンドのクラスターへの正常なアウトバウンド接続の数が、SSL ハンドシェイクの数に再利用された SSL セッションの数を加えたものと正確に同じであることを確認してください。

    ```
    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)
    ```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、AWS サポートにお問い合わせください。セキュリティの脆弱性が見つかったと思われる場合、または App

Mesh のセキュリティについて質問がある場合は、「AWS 脆弱性レポートガイドライン」を参照してください。

Elastic Load Balancing を使用した TLS のトラブルシューティング

症状

仮想ノードへのトラフィックを暗号化するように Application Load Balancer または Network Load Balancer を設定しようとすると、接続とロードバランサーのヘルスチェックが失敗することがあります。

解決方法

問題の根本原因を特定するには、次の点をチェックする必要があります。

- TLS 終了を実行する Envoy プロキシでは、設定ミスを除外する必要があります。上記の「TLS クライアントのポリシーを使用してバックエンド仮想サービスに接続できない」の手順に従います。
- ロードバランサーの場合は、`TargetGroup`: 設定を確認する必要がある
  - `TargetGroup` ポートが仮想ノードの定義済みリスナーポートと一致していることを確認してください。
  - HTTP を介してサービスへの TLS 接続を発信しているアプリケーションロードバランサーの場合、`TargetGroup` プロトコルが `HTTPS` に設定されていることを確認してください。ヘルスチェックを利用している場合は、`HealthCheckProtocol` が `HTTPS` に設定されていることを確認してください。
  - TCP を介してサービスへの TLS 接続を発信しているネットワークロードバランサーの場合、`TargetGroup` プロトコルが `TLS` に設定されていることを確認してください。ヘルスチェックを利用している場合は、`HealthCheckProtocol` が `TCP` に設定されていることを確認してください。

> **Note**
>
> `TargetGroup` へのすべての更新では、`TargetGroup` 名を変更する必要があります。

これが適切に設定されている場合、ロードバランサーは、Envoy プロキシに提供された証明書を使用して、サービスへの安全な接続を提供する必要があります。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。セキュリティの脆弱性が見つかったと思われる場合、または App

Mesh のセキュリティについて質問がある場合は、「AWS 脆弱性レポートガイドライン」を参照してください。

App Mesh Kubernetes のトラブルシューティング

このトピックでは、Kubernetes で App Mesh を使用するときに発生する可能性のある一般的な問題を詳細に説明します。

Kubernetes で作成されたアプリケーションメッシュリソースが App Mesh 内で見つからない

症状

Kubernetes カスタムリソース定義 (CRD) を使用してApp Mesh リソースを作成しましたが、AWS Management Console またはAPI を使用すると、作成したリソースは App Mesh に表示されません。

解決方法

考えられる原因は、App Mesh の Kubernetes コントローラーのエラーです。詳細については、「 でのトラブルシューティング」を参照してください GitHub。コントローラーのログで、コントローラーがリソースを作成できなかったことを示すエラーまたは警告がないかどうかをチェックしてください。

```
kubectl logs -n appmesh-system -f \
    $(kubectl get pods -n appmesh-system -o name | grep controller)
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

Envoy サイドカーが挿入された後、準備状態とライブネスのチェックに失敗する

症状

アプリケーションのポッドは、以前正常に実行されていましたが、Envoy サイドカーがポッドに挿入された後、準備状態とライブネスのチェックに失敗し始めました。

解決方法

ポッドに挿入された Envoy コンテナが App Mesh の Envoy 管理サービスでブートストラップされていることを確認します。エラーを確認するには、Envoy がエラーテキストで App Mesh Envoy 管理サービスから切断されました でエラーコードを参照します。次のコマンドを使用して、関連するポッドの Envoy ログを調べることができます。

```
kubectl logs -n appmesh-system -f \
    $(kubectl get pods -n appmesh-system -o name | grep controller) \
    | grep "gRPC config stream closed"
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

ポッドが AWS Cloud Map インスタンスとして登録されない、または登録解除される。

症状

Kubernetes ポッドが、ライフサイクルの一部として AWS Cloud Map にメンバー登録されていない、またはメンバー登録解除されています。ポッドが正常にスタートし、トラフィックを処理する準備ができていても、受信できない場合があります。ポッドが終了しても、クライアントはその IP アドレスを保持し、トラフィックを送信しようとする可能性があり、失敗します。

解決方法

これは既知の問題です。詳細については、「問題が発生してもポッドが Kubernetes で自動登録/登録解除されないAWS Cloud Map GitHub 」を参照してください。ポッド、App Mesh 仮想ノード、および AWS Cloud Map リソース間の関係により、Kubernetes の App Mesh コントローラーが非同期になり、リソースが失われる可能性があります。例えば、これは、関連するポッドを終了する前に仮想ノードリソースが Kubernetes から削除された場合に発生する可能性があります。

この問題を軽減するには、次の手順を実行します。

- Kubernetes 用の App Mesh コントローラーの最新バージョンを実行していることを確認してください。
- 仮想ノード定義で AWS Cloud Map、`namespaceName`、`serviceName` が正しいか確認してください。
- 仮想ノード定義を削除する前に、関連するポッドをすべて削除してください。仮想ノードに関連付けられているポッドを特定するための助けが必要な場合は、「App Mesh リソースのポッドが実行されている場所を特定できない」を参照してください。

- 問題が解決しない場合は、次のコマンドを実行して、根本的な問題を明らかにするのに役立つ可能性のあるエラーがないかコントローラーログを調べます。

```
kubectl logs -n appmesh-system \
    $(kubectl get pods -n appmesh-system -o name | grep appmesh-controller)
```

- 次のコマンドを使用してコントローラーのポッド を再起動することを検討してください。これにより、同期の問題が修正される可能性があります。

```
kubectl delete -n appmesh-system \
    $(kubectl get pods -n appmesh-system -o name | grep appmesh-controller)
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

App Mesh リソースのポッドが実行されている場所を特定できない

症状

Kubernetes クラスターで App Mesh を実行すると、オペレータは、特定の App Mesh リソースに対してワークロードまたはポッドが実行されている場所を特定できません。

解決方法

Kubernetes ポッドのリソースは、関連付けられているメッシュと仮想ノードで注釈が付けられます。次のコマンドを使用して、特定の仮想ノード名に対して実行されているポッドをクエリできます。

```
kubectl get pods --all-namespaces -o json | \
    jq '.items[] | { metadata } | select(.metadata.annotations."appmesh.k8s.aws/
virtualNode" == "virtual-node-name")'
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

ポッドが実行されている App Mesh リソースを特定できない

症状

Kubernetes クラスターで App Mesh を実行している場合、オペレータは、特定のポッドが実行されている App Mesh リソースを特定できません。

解決方法

Kubernetes ポッドのリソースには、関連付けられているメッシュと仮想ノードの注釈が付けられます。次のコマンドを使用して、 ポッドに直接クエリを実行することで、メッシュおよび仮想ノード名を出力できます。

```
kubectl get pod pod-name -n namespace -o json | \
    jq '{ "mesh": .metadata.annotations."appmesh.k8s.aws/mesh",
 "virtualNode": .metadata.annotations."appmesh.k8s.aws/virtualNode" }'
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

クライアント Envoy は、IMDSv1 が無効になっていると App Mesh Envoy Management Service と通信できません

症状

`IMDSv1` を無効にすると、クライアントの Envoy は App Mesh コントロールプレーン (Envoy Management Service) と通信できなくなります。`v1.24.0.0-prod` 以前のバージョンの App Mesh Envoy では `IMDSv2` はサポートされていません。

解決方法

この問題を解決するには、次の 3 つのいずれかを実行します。

- `IMDSv2` がサポートされている App Mesh Envoy バージョン `v1.24.0.0-prod` 以降にアップグレードします。
- Envoy が実行されているインスタンスで再度 `IMDSv1` を有効にします。`IMDSv1` の復元方法については、「インスタンスメタデータオプションの設定」を参照してください。
- サービスが Amazon EKS で実行されている場合、認証情報の取得にはサービスアカウントの IAM ロール (IRSA) を使用することをお勧めします。IRSA を有効にする方法については、「サービスアカウントの IAM ロール」を参照してください。

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

App Mesh が有効で、Envoy が挿入されている場合、IRSA はアプリケーションコンテナで動作しません

症状

Amazon EKS 用のApp Mesh コントローラーを利用して Amazon EKS クラスターで App Mesh を有効にした場合、Envoy と `proxyinit` コンテナがアプリケーションポッドに挿入されます。アプリケーションは IRSA を引き受けることができず、代わりに `node role` を引き受けます。ポッドの詳細を確認すると、`AWS_WEB_IDENTITY_TOKEN_FILE` または `AWS_ROLE_ARN` 環境変数のいずれかがアプリケーションコンテナに含まれていないことがわかります。

解決方法

`AWS_WEB_IDENTITY_TOKEN_FILE` または `AWS_ROLE_ARN` 環境変数が定義されている場合、ウェブフックはポッドをスキップします。これらの変数はいずれも指定しないでください。ウェブフックによってこれらの環境変数は自動的に挿入されます。

```
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
        }
```

それでも問題が解決しない場合は、GitHub 問題を開くことを検討してください。または、 AWS サポートにお問い合わせください。

App Mesh プレビューチャネル

App Mesh プレビューチャネルは、us-west-2 リージョンで提供される App Mesh サービスの明確なバリアントです。プレビューチャネルでは、今後開発される機能を公開し、お客様にお試しいただくものです。プレビューチャネルの機能を利用することで、GitHub を介してフィードバックを提供し、機能の動作を形成することができます。プレビューチャネルで機能が完成し、必要な統合とチェックがすべて完了すると、本番環境の App Mesh サービスへと移行します。

AWS App Mesh プレビューチャネルは、ベータサービスです、したがって、AWS サービス規約で定義されているように、すべての機能はプレビューです。お客様のプレビューチャネルへの参加は、AWS および AWS のサービス規約、特にユニバーサルサービスおよびベータサービス参加規約への同意に準拠して機密保持されます。

プレビューチャネルについて、よくあるご質問を次の通りです。

プレビューチャネルとは何ですか？

プレビューチャネルは、サービスの新機能を一般に公開する前にお客様が試用し、フィードバックを提供していただけるようにするパブリックサービスエンドポイントです。プレビューチャネルのサービスエンドポイントは、標準の本番稼働エンドポイントとは別のものです。エンドポイントとのやり取りは、AWS CLI、すなわち、プレビューチャネル用のサービスモデルファイル、AWS CLI 用のコマンド入力ファイルを使って行います。プレビューチャネルでは、現在の本稼働インフラストラクチャに影響を与えることなく、新機能を試すことができます。App Mesh がお客様の最も重要なご要望にお応えできるよう、App Mesh チームへのフィードバックをお願いします。プレビューチャネルの機能に関するフィードバックは、App Mesh の機能開発に役立ちてることができ、これにより、可能な限り最高のサービスを提供できます。

プレビューチャネルは、本番稼働の App Mesh とどう違うのですか？

次の表に、プレビューチャネルとは異なる App Mesh サービスの側面を示します。

側面	App Mesh 本番稼働サービス	App Mesh プレビューチャネルサービス

Frontend endpoint	appmesh.us-west-2.amazonaws.com	appmesh-preview.us-west-2.amazonaws.com
Envoy management service endpoint	appmesh-envoy-management.us-west-2.amazonaws.com	appmesh-preview-envoy-management.us-west-2.amazonaws.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 本番サービスの表の例には、`us-west-2` リージョンの場合、プロダクションサービスは、ほとんどのリージョンで利用できます。App Mesh 本番サービスが使用できるすべてのリージョンのリストについては、「AWS App Mesh エンドポイントとクォータ」を参照してください。ただし、「App Mesh プレビューチャネル」のサービスは、`us-west-2` リージョンでのみ利用可能です。

プレビューチャネルで機能を使用する方法を教えてください。

1. 次のコマンドで、プレビューチャンネル機能を含むサービスモデルを AWS CLI に追加します。

```
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. 機能に関する AWS App Mesh ユーザーガイドに記載されている JSON の例と説明に従った機能が含まれた JSON ファイルを作成します。

3. 適切な AWS CLI コマンドとコマンド入力ファイルを使用して機能を実装します。例えば、次のコマンドは、*route.json* ファイルを使用して、プレビューチャネル機能を備えたルートを作成します。

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

4. Amazon ECS タスク定義、Kubernetes Pod 仕様、またはAmazon EC2 インスタンスへの追加をする際に、Envoy コンテナの設定変数として APPMESH_PREVIEW = 1 を追加します。この変数を使用すると、Envoy コンテナがプレビューチャネルのエンドポイントと通信できるようになります。設定変数の追加方法の詳細については、「Amazon ECS でのサービスの更新」、「Kubernetes でのサービスの更新」および「Amazon EC2 でのサービスの更新」を参照してください。

フィードバックを提供するにはどうすればよいですか?

この機能に関するドキュメントからリンクされている App Mesh ロードマップ GitHub リポジトリの問題に直接フィードバックを提供することができます。

プレビューチャネルの機能に関するフィードバックの期間はどのくらいですか？

フィードバック期間は、導入する機能のサイズと複雑さによって異なります。機能がプレビューエンドポイントにリリースされてから本番環境にリリースされるまで、14日間のコメント期間を設ける予定です。App Mesh チームは、特定の機能のフィードバック期間を延長することができます。

プレビューチャネルにはどのレベルのサポートが提供されていますか。

App Mesh GitHub ロードマップの問題についてフィードバックとバグレポートを直接提供することをお勧めしますが、共有する機密データがある場合や、公開しても安全でないと思われる問題が見つかる場合があることを理解しています。これらの問題については、AWS Support にご連絡いただくか、App Meshチームに直接、電子メールでフィードバックをお寄せください。

プレビューチャネルエンドポイントでデータは安全ですか？

はい。プレビューチャネルには、標準の本番稼働エンドポイントと同じレベルのセキュリティが与えられます。

設定はどれくらいの期間利用可能になりますか？

プレビューチャネルで 30 日間メッシュを操作できます。メッシュが作成されてから 30 日経過すると、メッシュを一覧表示、読み取り、または削除することしかできません。30 日後にリソースを作成または更新しようとすると、メッシュがアーカイブされていることを説明する `BadRequest` 例外が発生します。

プレビューチャネルでの作業にはどのようなツールを使用できますか？

プレビューチャネルのサービスモデルファイルおよびコマンド入力ファイルで AWS CLI を使用できます。機能を用いた操の詳細については、「プレビューチャネルで機能を使用する方法を教えてください。」を参照してください。AWS CLI コマンドオプション、AWS Management Console、SDK、または AWS CloudFormation を使用して、プレビューチャネル機能を操作することはできません。ただし、機能が本番稼働サービス用にリリースされると、すべてのツールを使用できます。

プレビューチャネル API の前方互換性はありますか？

いいえ。API はフィードバックに基づいて変更される場合があります。

プレビューチャネルの機能は完成していますか？

いいえ。新しい API オブジェクトは、AWS Management Console、AWS CloudFormation、または AWS CloudTrail に完全に統合されていない場合があります。プレビューチャネルで機能が固定し、一般公開に近い状態になると、統合が最終的に利用可能になります。

プレビューチャネルの機能に関するドキュメントは入手できますか？

はい。プレビューチャネル機能のドキュメントは、本番稼働用ドキュメントに含まれています。例えば、ルートリソースの機能がプレビューチャネルに公開リリースされた場合、その機能の使用方法に

関する情報は既存のルートリソースドキュメントに記載されることになります。プレビューチャネルの機能は、プレビューチャネルでのみ使用可能とラベル付けされます。

プレビューチャネルで新しい機能がいつ利用可能になるかはどうすればわかりますか？

新しい機能がプレビューチャネルに導入されると、エントリが App Mesh ドキュメントの履歴に追加されます。定期的にページを確認するか、App Mesh ドキュメント履歴 RSS フィードにご登録ください。さらに、AWS App Mesh ロードマップの GitHub レポジトリの課題を確認することができます。プレビューチャネルのサービスモデル JSON ファイルのダウンロードリンクがプレビューチャネルにリリースされると、課題に追加されます。モデルや機能の使い方の詳細については、「プレビューチャネルで機能を使用する方法を教えてください。」を参照してください。

機能が本番稼働用サービスになる時期はどうすればわかりますか？

App Mesh ドキュメントで、この機能がプレビューチャネルでのみ使用可能であることを示すテキストが削除され、App Mesh ドキュメントの履歴に掲載されます。定期的にページを確認するか、App Mesh ドキュメント履歴 RSS フィードにご登録ください。

App Mesh Service Quotas

AWS App Mesh は、Service Quotas と統合されています。Service Quotas は、クォータを一元的な場所から表示および管理できる AWS サービスです。Service Quotas は、制限とも呼ばれます。詳細については、「Service Quotas ユーザーガイド」の「Service Quotas とは」を参照してください。

Service Quotas を使用すると、すべての App Mesh サービスクォータの値を簡単に検索できます。

AWS Management Console を使用して App Mesh サービスクォータを表示するには

1. https://console.aws.amazon.com/servicequotas/ で Service Quotas のコンソールを開きます。
2. ナビゲーションペインで、[AWS サービス] を選択します。
3. [AWS サービス] リストから、[AWS App Mesh]]] を探して選択します。

   Service Quotas の一覧には、サービスクォータ名、適用された値 (使用可能な場合)、AWS デフォルトのクォータ、クォータ値が調整可能かどうかが表示されます。
4. 説明など、Service Quotas に関する追加情報を表示するには、クォータ名を選択します。

クォータの引き上げをリクエストするには、「Service Quotas ユーザーガイド」の「クォータ引き上げリクエスト」を参照してください。

AWS CLI を使用して App Mesh サービスクォータを表示するには

以下のコマンドを実行します。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code appmesh \
    --output table
```

AWS CLIを使用して、さらにサービスクォータの操作を実行するには、「Service Quotas AWS CLI コマンドリファレンス」を参照してください。

App Mesh のドキュメント履歴

次の表は、「AWS App Mesh ユーザーガイド」の主な更新や新機能の一覧です。また、お客様からいただいたフィードバックに対応するために、ドキュメントを頻繁に更新しています。

変更	説明	日付
更新されたAWSAppMeshFullAccess ポリシー	TagResource および UntagResource APIs へのアクセスを許可するAWSAppMeshFullAccess ように更新されました。	2024 年 4 月 24 日
CloudTrail 統合ドキュメントが更新されました	API アクティビティを CloudTrail に記録する App Mesh と の統合について説明するドキュメントが更新されました。	2024 年 3 月 28 日
ポリシーの更新	API へのアクセスを許可するAWSAppMeshServiceRolePolicy ようにAWSServiceRoleForAppMesh と を更新しましたAWS Cloud Map DiscoverInstancesRevision 。	2023 年 10 月 12 日
App Mesh の VPC エンドポイントポリシーのサポート	App Mesh は VPC エンドポイントポリシーをサポートするようになりました。	2023 年 5 月 11 日
App Mesh の複数のリスナー	App Mesh は複数のリスナーをサポートするようになりました。	2022 年 8 月 18 日
App Mesh の IPv6	App Mesh は IPv6 をサポートするようになりました。	2022 年 5 月 18 日

CloudTrail App Mesh Envoy Management Service の口グ記録のサポート	App Mesh は、App Mesh Envoy Management Service の CloudTrail 口グ記録サポートをサポートするようになりました。	2022 年 3 月 18 日
App Mesh の Envoy 用エージェント	App Mesh は Envoy 用エージェントをサポートするようになりました。	2022 年 2 月 25 日
App Mesh の複数のリスナー	(App Mesh プレビューチャネルのみ) App Mesh に複数のリスナーを実装できます。	2021 年 11 月 23 日
App Mesh の ARM64 サポート	App Mesh は ARM64 をサポートするようになりました。	2021 年 11 月 19 日
App Mesh のメトリクス拡張	App Mesh のメトリクス拡張を実装できます。	2021 年 10 月 29 日
着信トラフィックの機能強化の実装	ホスト名とヘッダーの一致、ホスト名とパスの書き換えを実装することができます。	2021 年 6 月 14 日
相互 TLS 認証の実装	相互 TLS 認証を実装できます。	2021 年 2 月 4 日
af-south-1 でのリージョン発売	App Mesh は af-south-1 リージョンで利用可能になりました。	2021 年 1 月 22 日
相互 TLS 認証の実装	(App Mesh プレビューチャネルのみ) 相互 TLS 認証の実装を実装できます。	2020 年 11 月 23 日
仮想ゲートウェイリスナーへの接続プールの実装	仮想ゲートウェイリスナーに接続プールを実装できます。	2020 年 11 月 5 日

仮想ノードリスナーへの接続プールと異常値検出の実装	仮想ノードリスナーに接続プールと異常値検出を実装できます。	2020 年 11 月 5 日
eu-south-1 でのリージョン発売	App Mesh は eu-south-1 リージョンで利用可能になりました。	2020 年 10 月 21 日
仮想ゲートウェイリスナーへの接続プールの実装	(App Mesh プレビューチャネルのみ) 仮想ゲートウェイリスナーに接続プールを実装できます。	2020 年 9 月 28 日
仮想ノードリスナーへの接続プールと異常値検出の実装	(App Mesh プレビューチャネルのみ) 仮想ノードリスナーに接続プールと異常値検出を実装できます。	2020 年 9 月 28 日
メッシュインバウンドの仮想ゲートウェイとゲートウェイルートの作成	メッシュの外側にあるリソースが、メッシュの内側にあるリソースと通信できるようにします。	2020 年 7 月 10 日
Kubernetes 用 App Mesh コントローラーを使用して、Kubernetes 内から App Mesh リソースを作成し管理する	Kubernetes 内から App Mesh リソースを作成し管理できます。また、コントローラーは Envoy プロキシと init コンテナを、デプロイするポッドに自動的に挿入します。	2020 年 6 月 18 日
タイムアウト値を仮想ノードのリスナーとルートに追加する	タイムアウト値を仮想ノードのリスナーとルートに追加できます。	2020 年 6 月 18 日
タイムアウト値を仮想ノードリスナーに追加する	(App Mesh プレビューチャネルのみ)タイムアウト値を仮想ノードリスナーに追加できます。	2020 年 5 月 29 日

メッシュインバウンドの仮想ゲートウェイを作成する	(App Mesh プレビューチャネルのみ) メッシュ外のリソースを有効にして、メッシュ内のリソースと通信します。	2020 年 4 月 8 日
TLS 暗号化	(App Mesh プレビューチャネルのみ) AWS Private Certificate Authority または独自の認証機関からの証明書を使用して、TLS を使用して仮想ノード間の通信を暗号化します。	2020 年 1 月 17 日
メッシュを別のアカウントと共有する	(App Mesh プレビューチャネルのみ) メッシュを別のアカウントと共有できます。任意のアカウントで作成されたリソースは、メッシュ内の他のリソースと通信できます。	2020 年 1 月 17 日
タイムアウト値をルートに追加する	(App Mesh プレビューチャネルのみ) タイムアウト値をルートに追加できます。	2020 年 1 月 17 日
Outpost で App Mesh AWS プロキシを作成する	Outpost で App Mesh Envoy AWS プロキシを作成できます。	2019 年 12 月 3 日
ルート、仮想ルーター、仮想ノードのHTTP / 2およびgRPCサポート	HTTP/2 および gRPC プロトコルを使用するトラフィックをルーティングできます。また、これらのプロトコルのリスナーを仮想ノードと仮想ルーターに追加することもできます。	2019 年 10 月 25 日

再試行ポリシー	再試行ポリシーにより、再試行ポリシーは、断続的なネットワーク障害や断続的なサーバー側の障害からクライアントを保護することができます。再試行ロジックをルートに追加できます。	2019 年 9 月 10 日
TLS 暗号化	(App Mesh プレビューチャネルのみ) TLS を使用して仮想ノード間の通信を暗号化します。	2019 年 9 月 6 日
HTTP ヘッダーベースのルーティング	リクエスト内の HTTP ヘッダーの存在と値に基づいてトラフィックをルーティングします。	2019 年 8 月 15 日
App Mesh プレビューチャネルの可用性	App Mesh プレビューチャネルは、App Mesh サービスの異なるバリアントです。プレビューチャネルは、今後開発される機能を公開し、お客様にお試しいただくものです。プレビューチャネルで機能を使用すると、 を介してフィードバックを提供 GitHub して、機能の動作を形作ることができます。	2019 年 7 月 19 日

App Mesh インターフェイス VPC エンドポイント (AWS PrivateLink)	インターフェイス VPC エンドポイントを使用するように App Mesh を設定することで、VPC のセキュリティポスチャーを強化できます。インターフェイスエンドポイントは AWS PrivateLink、プライベート IP アドレスを使用して App Mesh APIs にプライベートにアクセスできるテクノロジーである を利用しています。 PrivateLink は、VPC と App Mesh 間のすべてのネットワークトラフィックを Amazon ネットワークに制限します。	2019 年 6 月 14 日
仮想ノードサービスの検出方法 AWS Cloud Map として を追加	仮想ノードサービスの検出方法 AWS Cloud Map として DNS または を指定できます。サービスディスカバリで AWS Cloud Map を使用するには、アカウントに App Mesh の サービス連動ロール が含まれている必要があります。	2019 年 6 月 13 日
Kubernetes 内に App Mesh リソースを自動的に作成する	Kubernetes 内にリソースを作成する場合には、App Mesh リソースを作成し、AppMesh サイドカーコンテナイメージを Kubernetes デプロイに自動追加します。	2019 年 6 月 11 日
App Mesh の一般的な可用性	App Meshは、本稼働環境用に一般公開されるようになりました。	2019 年 3 月 27 日

App Mesh API の更新	App Mesh API は、更新されて使いやすくなっています。詳細については、「【機能】リスナーを仮想ルーターに追加する」、「【BUG】一致していないポートブラックホールがある送信先の仮想ノードへのルート」を参照してください。	2019 年 3 月 7 日
App Mesh 初版リリース	サービス公開プレビュー用の初版ドキュメント	2018 年 11 月 28 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。