aws

User Guide

AWS App Mesh

AWS App Mesh: User Guide

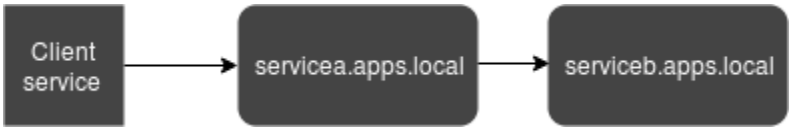
Table of Contents

What Is AWS App Mesh?

AWS App Mesh is a service mesh that makes it easy to monitor and control services. A service mesh is an infrastructure layer dedicated to handling service-to-service communication, usually through an array of lightweight network proxies deployed alongside the application code. App Mesh standardizes how your services communicate, giving you end-to-end visibility and helping to ensure high availability for your applications. App Mesh gives you consistent visibility and network traffic controls for every service in an application.

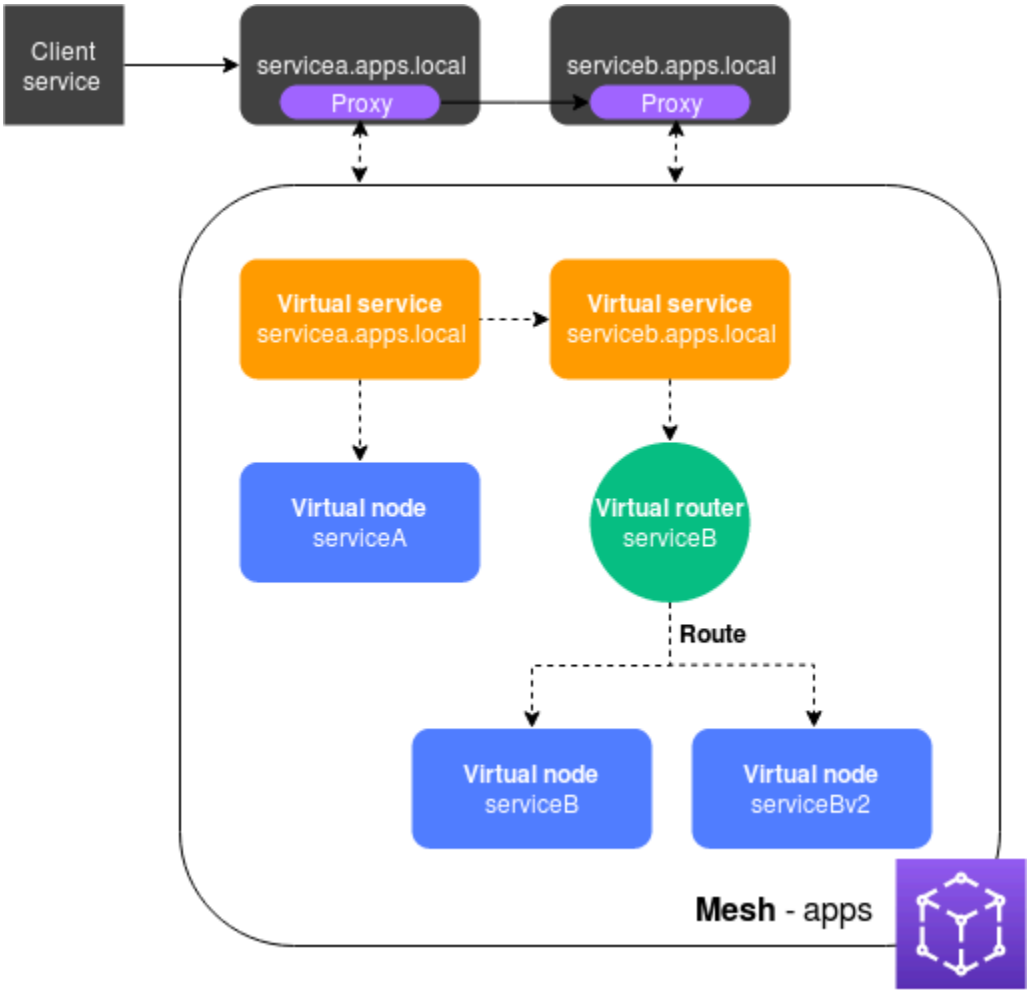
Adding App Mesh to an example application

Consider the following simple example application that doesn't use App Mesh. The two services can be running on AWS Fargate, Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), Kubernetes on Amazon Elastic Compute Cloud (Amazon EC2) instances, or on Amazon EC2 instances with Docker.



In this illustration, both `serviceA` and `serviceB` are discoverable through the `apps.local` namespace. Let's say, for example, you decide to deploy a new version of `serviceb.apps.local` named `servicebv2.apps.local`. Next, you want to direct a percentage of the traffic from `servicea.apps.local` to `serviceb.apps.local` and a percentage to `servicebv2.apps.local`. When you're sure that `servicebv2` is performing well, you want to send 100 percent of the traffic to it.

App Mesh can help you do this without changing any application code or registered service names. If you use App Mesh with this example application, then your mesh might look like the following illustration.



In this configuration, the services no longer communicate with each other directly. Instead, they communicate with each other through a proxy. The proxy deployed with the `servicea.apps.local` service reads the App Mesh configuration and sends traffic to `serviceb.apps.local` or `servicebv2.apps.local` based on the configuration.

Components of App Mesh

App Mesh is made up of the following components, illustrated in the previous example:

- **Service mesh** – A service mesh is a logical boundary for network traffic between the services that reside within it. In the example, the mesh is named `apps`, and it contains all other resources for the mesh. For more information, see Service Meshes.
- **Virtual services** – A virtual service is an abstraction of an actual service that is provided by a virtual node, directly or indirectly, by means of a virtual router. In the illustration, two virtual services represent the two actual services. The names of the virtual services are the discoverable

names of the actual services. When a virtual service and an actual service have the same name, multiple services can communicate with each other using the same names that they used before App Mesh was implemented. For more information, see Virtual services.

- **Virtual nodes** – A virtual node acts as a logical pointer to a discoverable service, such as an Amazon ECS or Kubernetes service. For each virtual service, you will have at least one virtual node. In the illustration, the `servicea.apps.local` virtual service gets configuration information for the virtual node named `serviceA`. The `serviceA` virtual node is configured with the `servicea.apps.local` name for service discovery. The `serviceb.apps.local` virtual service is configured to route traffic to the `serviceB` and `serviceBv2` virtual nodes through a virtual router named `serviceB`. For more information, see Virtual nodes.
- **Virtual routers and routes** – Virtual routers handle traffic for one or more virtual services within your mesh. A route is associated to a virtual router. The route is used to match requests for the virtual router and to distribute traffic to its associated virtual nodes. In the previous illustration, the `serviceB` virtual router has a route that directs a percentage of traffic to the `serviceB` virtual node, and a percentage of traffic to the `serviceBv2` virtual node. You can set the percentage of traffic routed to a particular virtual node and change it over time. You can route traffic based on criteria such as HTTP headers, URL paths, or gRPC service and method names. You can configure retry policies to retry a connection if there is an error in the response. For example, in the illustration, the retry policy for the route can specify that a connection to `serviceb.apps.local` is retried five times, with ten seconds between retry attempts, if `serviceb.apps.local` returns specific types of errors. For more information, see Virtual routers and Routes.
- **Proxy** – You configure your services to use the proxy after you create your mesh and its resources. The proxy reads the App Mesh configuration and directs traffic appropriately. In the illustration, all communication from `servicea.apps.local` to `serviceb.apps.local` goes through the proxy deployed with each service. The services communicate with each other using the same service discovery names that they used before introducing App Mesh. Because the proxy reads the App Mesh configuration, you can control how the two services communicate with each other. When you want change the App Mesh configuration, you don't need to change or redeploy the services themselves or the proxies. For more information, see Envoy image.

How to get started

To use App Mesh you must have an existing service running on AWS Fargate, Amazon ECS, Amazon EKS, Kubernetes on Amazon EC2, or Amazon EC2 with Docker.

To get started with App Mesh, see one of the following guides:

- Getting Started with App Mesh and Amazon ECS
- Getting Started with App Mesh and Kubernetes
- Getting Started with App Mesh and Amazon EC2

Accessing App Mesh

You can work with App Mesh in the following ways:

AWS Management Console

The console is a browser-based interface that you can use to manage App Mesh resources. You can open the App Mesh console at https://console.aws.amazon.com/appmesh/.

AWS CLI

Provides commands for a broad set of AWS products, and is supported on Windows, Mac, and Linux. To get started, see AWS Command Line Interface User Guide. For more information about the commands for App Mesh, see appmesh in the AWS CLI Command Reference.

AWS Tools for Windows PowerShell

Provides commands for a broad set of AWS products for those who script in the PowerShell environment. To get started, see the AWS Tools for Windows PowerShell User Guide. For more information about the cmdlets for App Mesh, see App Mesh in the AWS Tools for PowerShell Cmdlet Reference.

AWS CloudFormation

Enables you to create a template that describes all of the AWS resources that you want. Using the template, AWS CloudFormation provisions and configures the resources for you. To get started, see AWS CloudFormation User Guide. For more information about the App Mesh resource types, see App Mesh Resource Type Reference in the AWS CloudFormation Template Reference.

AWS SDKs

We also provide SDKs that enable you to access App Mesh from a variety of programming languages. The SDKs automatically take care of tasks such as:

- Cryptographically signing your service requests

- Retrying requests
- Handling error responses

For more information about available SDKs, see Tools for Amazon Web Services.

For more information about the App Mesh APIs, see the AWS App Mesh API Reference.

Getting started with App Mesh

You can use App Mesh with applications that you deploy to Amazon ECS, Kubernetes (that you deploy to your own Amazon EC2 instances or running on Amazon EKS), and Amazon EC2. To get started with App Mesh, select one of the services that you have applications deployed to that you want to use with App Mesh. You can always enable applications in the other services to also work with App Mesh after you complete one of the Getting started guides.

Topics

Getting started with AWS App Mesh and Amazon ECS

This topic helps you use AWS App Mesh with an actual service that is running on Amazon ECS. This tutorial covers basic features of several App Mesh resource types.

Scenario

To illustrate how to use App Mesh, assume that you have an application with the following characteristics:

- Consists of two services named `serviceA` and `serviceB`.
- Both services are registered to a namespace named `apps.local`.
- `ServiceA` communicates with `serviceB` over HTTP/2, port 80.
- You have already deployed version 2 of `serviceB` and registered it with the name `serviceBv2` in the `apps.local` namespace.

You have the following requirements:

- You want to send 75 percent of the traffic from `serviceA` to `serviceB` and 25 percent of the traffic to `serviceBv2` to validate that `serviceBv2` is bug free before you send 100 percent of the traffic from `serviceA` to it.
- You want to be able to easily adjust the traffic weighting so that 100 percent of the traffic goes to `serviceBv2` once it is proven to be reliable. Once all traffic is being sent to `serviceBv2`, you want to discontinue `serviceB`.
- You do not want to have to change any existing application code or service discovery registration for your actual services to meet the previous requirements.

To meet your requirements, you decide to create an App Mesh service mesh with virtual services, virtual nodes, a virtual router, and a route. After implementing your mesh, you update your services to use the Envoy proxy. Once updated, your services communicate with each other through the Envoy proxy rather than directly with each other.

Prerequisites

- An existing understanding of App Mesh concepts. For more information, see *What Is AWS App Mesh?*.
- An existing understanding of Amazon ECSs concepts. For more information, see What is Amazon ECS in the Amazon Elastic Container Service Developer Guide.
- App Mesh supports Linux services that are registered with DNS, AWS Cloud Map, or both. To use this getting started guide, we recommend that you have three existing services that are registered with DNS. The procedures in this topic assume that the existing services are named `serviceA`, `serviceB`, and `serviceBv2` and that all services are discoverable through a namespace named `apps.local`.

  You can create a service mesh and its resources even if the services don't exist, but you cannot use the mesh until you have deployed actual services. For more information about service discovery on Amazon ECS, see Service Discovery. To create an Amazon ECS service with service discovery, see Tutorial: Creating a Service Using Service Discovery. If you don't already have services running, you can Create an Amazon ECS service with service discovery.

Step 1: Create a mesh and virtual service

A service mesh is a logical boundary for network traffic between the services that reside within it. For more information, see Service Meshes. A virtual service is an abstraction of an actual service. For more information, see Virtual services.

Create the following resources:

- A mesh named `apps`, since all of the services in the scenario are registered to the `apps.local` namespace.
- A virtual service named `serviceb.apps.local`, since the virtual service represents a service that is discoverable with that name, and you don't want to change your code to reference another name. A virtual service named `servicea.apps.local` is added in a later step.

You can use the AWS Management Console or the AWS CLI version 1.18.116 or higher or 2.0.38 or higher to complete the following steps. If using the AWS CLI, use the `aws --version` command to check your installed AWS CLI version. If you don't have version 1.18.116 or higher or 2.0.38 or higher installed, then you must install or update the AWS CLI. Select the tab for the tool that you want to use.

AWS Management Console

1. Open the App Mesh console first-run wizard at https://console.aws.amazon.com/appmesh/get-started.
2. For **Mesh name**, enter **apps**.
3. For **Virtual service name**, enter **`serviceb.apps.local`**.
4. To continue, choose **Next**.

AWS CLI

1. Create a mesh with the `create-mesh` command.

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

2. Create a virtual service with the `create-virtual-service` command.

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

Step 2: Create a virtual node

A virtual node acts as a logical pointer to an actual service. For more information, see Virtual nodes.

Create a virtual node named `serviceB`, since one of the virtual nodes represents the actual service named `serviceB`. The actual service that the virtual node represents is discoverable through DNS with a hostname of `serviceb.apps.local`. Alternately, you can discover actual services using AWS Cloud Map. The virtual node will listen for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported, as are health checks. You will create virtual nodes for `serviceA` and `serviceBv2` in a later step.

AWS Management Console

1. For **Virtual node name**, enter **`serviceB`**.
2. For **Service discovery method**, choose **DNS** and enter **`serviceb.apps.local`** for **DNS hostname**.
3. Under **Listener configuration**, choose **http2** for **Protocol** and enter **`80`** for **Port**.
4. To continue, choose **Next**.

AWS CLI

1. Create a file named `create-virtual-node-serviceb.json` with the following contents:

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

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

2. Create the virtual node with the create-virtual-node command using the JSON file as input.

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

Step 3: Create a virtual router and route

Virtual routers route traffic for one or more virtual services within your mesh. For more information, see Virtual routers and Routes.

Create the following resources:

- A virtual router named `serviceB`, since the `serviceB.apps.local` virtual service does not initiate outbound communication with any other service. Remember that the virtual service that you created previously is an abstraction of your actual `serviceb.apps.local` service. The virtual service sends traffic to the virtual router. The virtual router listens for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported.
- A route named `serviceB`. It routes 100 percent of its traffic to the `serviceB` virtual node. The weight is in a later step once you add the `serviceBv2` virtual node. Though not covered in this guide, you can add additional filter criteria for the route and add a retry policy to cause the Envoy proxy to make multiple attempts to send traffic to a virtual node when it experiences a communication problem.

AWS Management Console

1. For **Virtual router name,** enter **`serviceB`**.
2. Under **Listener configuration**, choose **http2** for **Protocol** and specify **`80`** for **Port**.

3. For **Route name**, enter **`serviceB`**.
4. For **Route type**, choose **http2**.
5. For **Virtual node name** under **Target configuration**, select `serviceB` and enter **`100`** for **Weight**.
6. Under **Match configuration**, choose a **Method**.
7. To continue, choose **Next**.

AWS CLI

1. Create a virtual router.

   a. Create a file named `create-virtual-router.json` with the following contents:

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

   b. Create the virtual router with the create-virtual-router command using the JSON file as input.

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

2. Create a route.

   a. Create a file named `create-route.json` with the following contents:

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

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

b. Create the route with the create-route command using the JSON file as input.

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

Step 4: Review and create

Review the settings against the previous instructions.

AWS Management Console

Choose **Edit** if you need to make changes in any section. Once you are satisfied with the settings, choose **Create mesh**.

The **Status** screen shows you all of the mesh resources that were created. You can see the created resources in the console by selecting **View mesh**.

AWS CLI

Review the settings of the mesh you created with the describe-mesh command.

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

Review the settings of the virtual service that you created with the describe-virtual-service command.

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

Review the settings of the virtual node that you created with the describe-virtual-node command.

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

Review the settings of the virtual router that you created with the describe-virtual-router command.

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

Review the settings of the route that you created with the describe-route command.

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

Step 5: Create additional resources

To complete the scenario, you need to:

- Create one virtual node named `serviceBv2` and another named `serviceA`. Both virtual nodes listen for requests over HTTP/2 port 80. For the `serviceA` virtual node, configure a backend of `serviceb.apps.local`. All outbound traffic from the `serviceA` virtual node is sent to the virtual service named `serviceb.apps.local`. Though not covered in this guide, you can also specify a file path to write access logs to for a virtual node.
- Create one additional virtual service named `servicea.apps.local`, which sends all traffic directly to the `serviceA` virtual node.
- Update the `serviceB` route that you created in a previous step to send 75 percent of its traffic to the `serviceB` virtual node and 25 percent of its traffic to the `serviceBv2` virtual node. Over time, you can continue to modify the weights until `serviceBv2` receives 100 percent of the traffic. Once all traffic is sent to `serviceBv2`, you can shut down and discontinue the

serviceB virtual node and actual service. As you change weights, your code does not require any modification, because the serviceb.apps.local virtual and actual service names don't change. Recall that the serviceb.apps.local virtual service sends traffic to the virtual router, which routes the traffic to the virtual nodes. The service discovery names for the virtual nodes can be changed at any time.

AWS Management Console

1. In the left navigation pane, select **Meshes**.
2. Select the apps mesh that you created in a previous step.
3. In the left navigation pane, select **Virtual nodes**.
4. Choose **Create virtual node**.
5. For **Virtual node name**, enter **serviceBv2**, for **Service discovery method**, choose **DNS**, and for **DNS hostname**, enter **servicebv2.apps.local**.
6. For **Listener configuration**, select **http2** for **Protocol** and enter **80** for **Port**.
7. Choose **Create virtual node**.
8. Choose **Create virtual node** again. Enter **serviceA** for the **Virtual node name**. For **Service discovery method**, choose **DNS**, and for **DNS hostname**, enter **servicea.apps.local**.
9. For **Enter a virtual service name** under **New backend**, enter **serviceb.apps.local**.
10. Under **Listener configuration**, choose **http2** for **Protocol**, enter **80** for **Port**, and then choose **Create virtual node**.
11. In the left navigation pane, select **Virtual routers** and then select the serviceB virtual router from the list.
12. Under **Routes**, select the route named ServiceB that you created in a previous step, and choose **Edit**.
13. Under **Targets**, **Virtual node name**, change the value of **Weight** for serviceB to **75**.
14. Choose **Add target**, choose serviceBv2 from the dropdown list, and set the value of **Weight** to **25**.
15. Choose **Save**.
16. In the left navigation pane, select **Virtual services** and then choose **Create virtual service**.
17. Enter **servicea.apps.local** for **Virtual service name**, select **Virtual node** for **Provider**, select serviceA for **Virtual node**, and then choose **Create virtual service.**

AWS CLI

1. Create the `serviceBv2` virtual node.

   a. Create a file named `create-virtual-node-servicebv2.json` with the following contents:

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

   b. Create the virtual node.

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

2. Create the `serviceA` virtual node.

   a. Create a file named `create-virtual-node-servicea.json` with the following contents:

```
{
   "meshName" : "apps",
   "spec" : {
      "backends" : [
         {
            "virtualService" : {
```

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

b. Create the virtual node.

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

3. Update the `serviceb.apps.local` virtual service that you created in a previous step to send its traffic to the `serviceB` virtual router. When the virtual service was originally created, it did not send traffic anywhere, since the `serviceB` virtual router had not been created yet.

   a. Create a file named `update-virtual-service.json` with the following contents:

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

```
}
```

b. Update the virtual service with the update-virtual-service command.

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

4. Update the `serviceB` route that you created in a previous step.

a. Create a file named `update-route.json` with the following contents:

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

b. Update the route with the update-route command.

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

5. Create the `serviceA` virtual service.

a. Create a file named `create-virtual-servicea.json` with the following contents:

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

b. Create the virtual service.

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

Mesh summary

Before you created the service mesh, you had three actual services named `servicea.apps.local`, `serviceb.apps.local`, and `servicebv2.apps.local`. In addition to the actual services, you now have a service mesh that contains the following resources that represent the actual services:

- Two virtual services. The proxy sends all traffic from the `servicea.apps.local` virtual service to the `serviceb.apps.local` virtual service through a virtual router.
- Three virtual nodes named `serviceA`, `serviceB`, and `serviceBv2`. The Envoy proxy uses the service discovery information configured for the virtual nodes to look up the IP addresses of the actual services.
- One virtual router with one route that instructs the Envoy proxy to route 75 percent of inbound traffic to the `serviceB` virtual node and 25 percent of the traffic to the `serviceBv2` virtual node.

Step 6: Update services

After creating your mesh, you need to complete the following tasks:

- Authorize the Envoy proxy that you deploy with each Amazon ECS task to read the configuration of one or more virtual nodes. For more information about how to authorize the proxy, see Proxy authorization.
- Update each of your existing Amazon ECS task definitions to use the Envoy proxy.

Credentials

The Envoy container requires AWS Identity and Access Management credentials for signing requests that are sent to the App Mesh service. For Amazon ECS tasks deployed with the Amazon EC2 launch type, the credentials can come from the instance role or from a task IAM role. Amazon ECS tasks deployed with Fargate on Linux containers don't have access to the Amazon EC2 metadata server that supplies instance IAM profile credentials. To supply the credentials, you must attach an IAM task role to any tasks deployed with the Fargate on Linux containers type.

If a task is deployed with the Amazon EC2 launch type and access is blocked to the Amazon EC2 metadata server, as described in the *Important* annotation in IAM Role for Tasks, then a task IAM role must also be attached to the task. The role that you assign to the instance or task must have an IAM policy attached to it as described in Proxy authorization.

To update your task definitions using the AWS Management Console

The following steps only show updating the `taskB` task for the scenario. You also need to update the `taskBv2` and `taskA` tasks by changing the values appropriately.

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. To access the classic classic Amazon ECS console, in the upper-left toggle **New ECS Experience**.

   > **Important**
   >
   > App Mesh integration is only available in the classic Amazon ECS console.

3. From the navigation bar, choose the Region that contains your task definition.
4. In the navigation pane, choose **Task Definitions**.
5. On the **Task Definitions** page, select the box to the left of the task definition to revise. From the pre-requisites and previous steps, you might have task definitions named `taskA`, `taskB`, and `taskBv2`. Select `taskB` and choose **Create new revision**.
6. On the **Create new revision of Task Definition** page, make the following changes to enable App Mesh integration.

a. For **Service Integration**, to configure the parameters for App Mesh integration choose **Enable App Mesh integration** and then do the following:

   i. For **Application container name**, choose the container name to use for the App Mesh application. This container must already be defined within the task definition.

   ii. For **Envoy image**, complete the following task and enter the value that is returned.

   All supported Regions other than `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`. You can replace *`Region-code`* with any Region other than `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`.

   Standard

   ```
   840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-
   envoy:v1.27.3.0-prod
   ```

   FIPS-compliant

   ```
   840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-
   envoy:v1.27.3.0-prod-fips
   ```

   `me-south-1`

   Standard

   ```
   772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-
   envoy:v1.27.3.0-prod
   ```

   FIPS-compliant

   ```
   772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-
   envoy:v1.27.3.0-prod-fips
   ```

   `ap-east-1`

   Standard

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod
```

FIPS-compliant

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod-fips
```

ap-southeast-3

Standard

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod
```

FIPS-compliant

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod-fips
```

eu-south-1

Standard

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod
```

FIPS-compliant

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod-fips
```

il-central-1

Standard

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod
```

FIPS-compliant

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod-fips
```

`af-south-1`

Standard

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod
```

FIPS-compliant

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod-fips
```

`Public repository`

Standard

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

Important

Only version v1.9.0.0-prod or later is supported for use with App Mesh.

iii. For **Mesh name**, choose the App Mesh service mesh to use. In this topic, the name of the mesh that was created is `apps`.

iv. For **Virtual node name**, choose the App Mesh virtual node to use. For example, for the `taskB` task, you would choose the `serviceB` virtual node that you created in a previous step.

v. The value for **Virtual node port** is pre-populated with the listener port that you specified when you created the virtual node.

vi. Choose **Apply**, and then choose **Confirm**. A new Envoy proxy container is created and added to the task definition, and the settings to support the container are also created. The Envoy proxy container then pre-populates the App Mesh **Proxy Configuration** settings for the next step.

b. For **Proxy Configuration**, verify all of the pre-populated values.

c. For **Network Mode**, make sure that `awsvpc` is selected. To learn more about the `awsvpc` network mode, see Task Networking with the `awsvpc` Network Mode.

7. Choose **Create**.

8. Update your service with the updated task definition. For more information, see Updating a service.

The console creates the task definition's json specification. You can modify some of the settings, but not others. For more information, expand the following section.

Task definition json

Proxy configuration

To configure your Amazon ECS service to use App Mesh, your service's task definition must have the following proxy configuration section. Set the proxy configuration `type` to `APPMESH` and the `containerName` to `envoy`. Set the following property values accordingly.

`IgnoredUID`

The Envoy proxy doesn't route traffic from processes that use this user ID. You can choose any user ID that you want for this property value, but this ID must be the same as the `user` ID for the Envoy container in your task definition. This matching allows Envoy to ignore its own traffic without using the proxy. Our examples use *1337* for historical purposes.

`ProxyIngressPort`

This is the inbound port for the Envoy proxy container. Set this value to `15000`.

`ProxyEgressPort`

This is the outbound port for the Envoy proxy container. Set this value to `15001`.

`AppPorts`

Specify any inbound ports that your application containers listen on. In this example, the application container listens on port *9080*. The port that you specify must match the port configured on the virtual node listener.

`EgressIgnoredIPs`

Envoy doesn't proxy traffic to these IP addresses. Set this value to `169.254.170.2,169.254.169.254`, which ignores the Amazon EC2 metadata server and the Amazon ECS task metadata endpoint. The metadata endpoint provides IAM roles for tasks credentials. You can add additional addresses.

`EgressIgnoredPorts`

You can add a comma separated list of ports. Envoy doesn't proxy traffic to these ports. Even if you list no ports, port 22 is ignored.

> **Note**
>
> The maximum number of outbound ports that can be ignored is 15.

```
"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"
  }
 ]
}
```

Application container Envoy dependency

The application containers in your task definitions must wait for the Envoy proxy to bootstrap and start before they can start. To make sure this happens, you set a `dependsOn` section in each application container definition to wait for the Envoy container to report as `HEALTHY`. The following code shows an application container definition example with this dependency. All of the properties in the following example are required. Some of the property values are also required, but some are *`replaceable`*.

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

Envoy container definition

Your Amazon ECS task definitions must contain an App Mesh Envoy container image.

All supported Regions other than `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`. You can replace *`Region-code`* with any Region other than `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`.

Standard

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

`me-south-1`

Standard

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

`ap-east-1`

Standard

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

`ap-southeast-3`

Standard

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

eu-south-1

Standard

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

il-central-1

Standard

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

af-south-1

Standard

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

Public repository

Standard

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

Important

Only version v1.9.0.0-prod or later is supported for use with App Mesh.

You must use the App Mesh Envoy container image until the Envoy project team merges changes that support App Mesh. For additional details, see the GitHub roadmap issue.

All of the properties in the following example are required. Some of the property values are also required, but some are *replaceable*.

Note

- The Envoy container definition must be marked as `essential`.
- We recommend allocating `512` CPU units and at least `64` MiB of memory to the Envoy container. On Fargate the lowest you will be able to set is `1024` MiB of memory.
- The virtual node name for the Amazon ECS service must be set to the value of the `APPMESH_RESOURCE_ARN` property. This property requires version `1.15.0` or later of the Envoy image. For more information, see *Envoy*.
- The value for the `user` setting must match the `IgnoredUID` value from the task definition proxy configuration. In this example, we use *1337*.
- The health check shown here waits for the Envoy container to bootstrap properly before reporting to Amazon ECS that the Envoy container is healthy and ready for the application containers to start.
- By default, App Mesh uses the name of the resource you specified in `APPMESH_RESOURCE_ARN` when Envoy is referring to itself in metrics and traces. You can override this behavior by setting the `APPMESH_RESOURCE_CLUSTER` environment variable with your own name. This property requires version `1.15.0` or later of the Envoy image. For more information, see *Envoy*.

The following code shows an Envoy container definition example.

```
{
 "name": "envoy",
 "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-
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"
}
```

Example task definitions

The following example Amazon ECS task definitions show how to merge the examples from above into a task definition for `taskB`. Examples are provided for creating tasks for both Amazon ECS launch types with or without using AWS X-Ray. Change the *`replaceable`* values, as appropriate, to create task definitions for the tasks named `taskBv2` and `taskA` from the scenario. Substitute your mesh name and virtual node name for the `APPMESH_RESOURCE_ARN` value and a list of ports that your application listens on for the proxy configuration `AppPorts` value. By default, App Mesh uses the name of the resource you specified in `APPMESH_RESOURCE_ARN` when Envoy is referring to itself in metrics and traces. You can override this behavior by setting the `APPMESH_RESOURCE_CLUSTER` environment variable with your own name. All of the properties in the following examples are required. Some of the property values are also required, but some are *`replaceable`*.

If you're running an Amazon ECS task as described in the Credentials section, then you need to add an existing task IAM role, to the examples.

Important

Fargate must use a port value greater than 1024.

Example JSON for Amazon ECS task definition - Fargate on Linux containers

```
{

   "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:v1.27.3.0-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 JSON for Amazon ECS task definition with AWS X-Ray - Fargate on Linux containers

X-Ray allows you to collect data about requests that an application serves and provides tools that you can use to visualize traffic flow. Using the X-Ray driver for Envoy enables Envoy to report tracing information to X-Ray. You can enable X-Ray tracing using the Envoy configuration. Based on the configuration, Envoy sends tracing data to the X-Ray daemon running as a sidecar container and the daemon forwards the traces to the X-Ray service. Once the traces are published to X-Ray, you can use the X-Ray console to visualize the service call graph and request trace details. The following JSON represents a task definition to enable X-Ray integration.

```
{


   "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:v1.27.3.0-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 JSON for Amazon ECS task definition - EC2 launch type

```
{
  "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:v1.27.3.0-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 JSON for Amazon ECS task definition with AWS X-Ray - EC2 launch type

```
{
  "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:v1.27.3.0-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"
```

```
}
```

Advanced topics

Canary deployments using App Mesh

Canary deployments and releases help you switch traffic between an old version of an application and a newly deployed version. It also monitors the health of the newly deployed version. If there are any problems with the new version, the canary deployment can automatically switch traffic back to the old version. Canary deployments give you the ability to switch traffic between application versions with more control.

For more information about how to implement canary deployments for Amazon ECS using App Mesh, see Create a pipeline with canary deployments for Amazon ECS using App Mesh

> **Note**
>
> For more examples and walkthroughs for App Mesh, see the App Mesh examples repository.

Getting started with AWS App Mesh and Kubernetes

When you integrate AWS App Mesh with Kubernetes using the App Mesh controller for Kubernetes, you manage App Mesh resources, such as meshes, virtual services, virtual nodes, virtual routers, and routes through Kubernetes. You also automatically add the App Mesh sidecar container images to Kubernetes pod specifications. This tutorial guides you through the installation of the App Mesh controller for Kubernetes to enable this integration.

The controller is accompanied by the deployment of the following Kubernetes custom resource definitions: `meshes`, `virtual services`, `virtual nodes`, and `virtual routers`. The controller watches for creation, modification, and deletion of the custom resources and makes changes to the corresponding App Mesh the section called “Meshes”, the section called “Virtual services”, the section called “Virtual nodes”, the section called “Virtual gateways”, the section called “Gateway routes”, the section called “Virtual routers” (including the section called “Routes”) resources through the App Mesh API. To learn more or contribute to the controller, see the GitHub project.

The controller also installs a webhook that injects the following containers into Kubernetes pods that are labeled with a name that you specify.

- **App Mesh Envoy proxy** – Envoy uses the configuration defined in the App Mesh control plane to determine where to send your application traffic.
- **App Mesh proxy route manager** – Updates `iptables` rules in a pod's network namespace that route inbound and outbound traffic through Envoy. This container runs as a Kubernetes init container inside of the pod.

Prerequisites

- An existing understanding of App Mesh concepts. For more information, see *What Is AWS App Mesh?*.
- An existing understanding of Kubernetes concepts. For more information, see What is Kubernetes in the Kubernetes documentation.
- An existing Kubernetes cluster. If you don't have an existing cluster, see Getting Started with Amazon EKS in the *Amazon EKS User Guide*. If you're running your own Kubernetes cluster on Amazon EC2, then ensure that Docker is authenticated to the Amazon ECR repository that the Envoy image is in. For more information, see Envoy image, Registry authentication in the Amazon Elastic Container Registry User Guide, and Pull an Image from a Private Registry in the Kubernetes documentation.
- App Mesh supports Linux services that are registered with DNS, AWS Cloud Map, or both. To use this getting started guide, we recommend that you have three existing services that are registered with DNS. The procedures in this topic assume that the existing services are named `serviceA`, `serviceB`, and `serviceBv2` and that all services are discoverable through a namespace named `apps.local`.

  You can create a service mesh and its resources even if the services don't exist, but you cannot use the mesh until you have deployed actual services.
- The AWS CLI version 1.18.116 or later or 2.0.38 or later installed. To install or upgrade the AWS CLI, see Installing the AWS CLI.
- A `kubectl` client that is configured to communicate with your Kubernetes cluster. If you're using Amazon Elastic Kubernetes Service, you can use the instructions for installing `kubectl` and configuring a `kubeconfig` file.

- Helm version 3.0 or later installed. If you don't have Helm installed, see Using Helm with Amazon EKS in the *Amazon EKS User Guide*.
- Amazon EKS currently only supports `IPv4_ONLY` and `IPv6_ONLY` only IP preferences, because Amazon EKS currently only supports pods that are capable of serving either only `IPv4` traffic or only `IPv6` traffic.

The remaining steps assume that the actual services are named `serviceA`, `serviceB`, and `serviceBv2` and that all services are discoverable through a namespace named `apps.local`.

Step 1: Install the integration components

Install the integration components one time to each cluster that hosts pods that you want to use with App Mesh.

To install the integration components

1. The remaining steps of this procedure require a cluster without a pre-release version of the controller installed. If you have installed a pre-release version, or are not sure whether you have, you can download and run a script that checks to see whether a pre-release version is installed on your cluster.

   ```
   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
   ```

   If the script returns `Your cluster is ready for upgrade. Please proceed to the installation instructions` then you can proceed to the next step. If a different message is returned, then you'll need to complete the upgrade steps before continuing. For more information about upgrading a pre-release version, see Upgrade on GitHub.

2. Add the `eks-charts` repository to Helm.

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

3. Install the App Mesh Kubernetes custom resource definitions (CRD).

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

4. Create a Kubernetes namespace for the controller.

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

5. Set the following variables for use in later steps. Replace *cluster-name* and *Region-code* with the values for your existing cluster.

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

6. (Optional) If you want to run the controller on Fargate, then you need to create a Fargate profile. If you don't have `eksctl` installed, see Installing or Upgrading `eksctl` in the *Amazon EKS User Guide*. If you'd prefer to create the profile using the console, see Creating a Fargate profile in the *Amazon EKS User Guide*.

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

7. Create an OpenID Connect (OIDC) identity provider for your cluster. If you don't have `eksctl` installed, you can install it with the instructions in Installing or upgrading `eksctl` in the *Amazon EKS User Guide*. If you'd prefer to create the provider using the console, see Enabling IAM roles for service accounts on your cluster in the *Amazon EKS User Guide*.

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

8. Create an IAM role, attach the AWSAppMeshFullAccess and AWSCloudMapFullAccess AWS managed policies to it, and bind it to the `appmesh-controller` Kubernetes service account. The role enables the controller to add, remove, and change App Mesh resources.

> **Note**
>
> The command creates an AWS IAM role with an auto-generated name. You are not able to specify the IAM role name that is created.

```
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
```

If you prefer to create the service account using the AWS Management Console or AWS CLI, see Creating an IAM role and policy for your service account in the *Amazon EKS User Guide*. If you use the AWS Management Console or AWS CLI to create the account, you also need to map the role to a Kubernetes service account. For more information, see Specifying an IAM role for your service account in the *Amazon EKS User Guide*.

9. **Deploy the App Mesh controller. For a list of all configuration options, see Configuration on GitHub.**

   1. To deploy the App Mesh controller for a private cluster, you have to enable App Mesh and service discovery Amazon VPC endpoints to the linked private subnet first. You're also required to set the `accountId`.

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

      To enable X-Ray tracing in a private cluster, enable the X-Ray and Amazon ECR Amazon VPC endpoints. The controller uses `public.ecr.aws/xray/aws-xray-daemon:latest` by default, so pull this image to local and push it into your personal ECR repository.

      > **Note**
      >
      > Amazon VPC endpoints currently don't support Amazon ECR public repositories.

      The following example shows deploying the controller with configurations for X-Ray.

      ```
      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
```

Verify if the X-Ray daemon is injected successfully when binding the application deployment with your virtual node or gateway.

For more information, see Private Clusters in the *Amazon EKS User Guide*.

2. Deploy the App Mesh controller for other clusters. For a list of all configuration options, see Configuration on 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

If your Amazon EKS cluster family is `IPv6`, please set the cluster name when deploying the App Mesh controller by adding the following option to the previous command `--set clusterName=$CLUSTER_NAME`.

Important

If your cluster is in the `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, or `af-south-1` Regions, then you need to add the following option to the previous command:
Replace *`account-id`* and *`Region-code`* with one of the appropriate sets of values.

- **For the sidecar image:**
  -
```
 --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:v1.27.3.0-prod
  - 856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
  - 909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
  - 422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
  - 564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
  - 924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
- The older image URIs can be found in the change log on GitHub. The AWS accounts on which the images are present have changed in version `v1.5.0`. Older version of the images are hosted on AWS accounts found on the Amazon Elastic Kubernetes Service Amazon container image registries.
- **For the controller image:**
  - ```
    --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:v1.12.3
  - 856666278305.dkr.ecr.ap-east-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 422531588944.dkr.ecr.eu-south-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 564877687649.dkr.ecr.il-central-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 924023996002.dkr.ecr.af-south-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
    ```

- **For the sidecar init image:**
  - `--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

Only version v1.9.0.0-prod or later is supported for use with App Mesh.

10. Confirm that the controller version is `v1.4.0` or later. You can review the change log on GitHub.

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

Note

If you view the log for the running container, you may see a line that includes the following text, which can be safely ignored.

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

Step 2: Deploy App Mesh resources

When you deploy an application in Kubernetes, you also create the Kubernetes custom resources so that the controller can create the corresponding App Mesh resources. The following procedure helps you deploy App Mesh resources with some of their features. You can find example manifests for deploying other App Mesh resource features in the `v1beta2` sub-folders of many of the feature folders listed at App Mesh walkthroughs on GitHub.

> **Important**
>
> Once the controller has created an App Mesh resource, we recommend that you only make changes to or delete the App Mesh resource using the controller. If you make changes to or delete the resource using App Mesh, the controller won't change or recreate the changed or deleted App Mesh resource for ten hours, by default. You can configure this duration to be less. For more information, see Configuration on GitHub.

To deploy App Mesh resources

1. Create a Kubernetes namespace to deploy App Mesh resources to.

   a. Save the following contents to a file named `namespace.yaml` on your computer.

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

   b. Create the namespace.

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

2. Create an App Mesh service mesh.

   a. Save the following contents to a file named `mesh.yaml` on your computer. The file is used to create a mesh resource named *my-mesh*. A service mesh is a logical boundary for network traffic between the services that reside within it.

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

   b. Create the mesh.

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

   c. View the details of the Kubernetes mesh resource that was created.

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

   Output

```
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. View the details about the App Mesh service mesh that the controller created.

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

Output

```
{
    "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. Create an App Mesh virtual node. A virtual node acts as a logical pointer to a Kubernetes deployment.

a. Save the following contents to a file named `virtual-node.yaml` on your computer. The file is used to create an App Mesh virtual node named *my-service-a* in the *my-apps* namespace. The virtual node represents a Kubernetes service that is created in a later step. The value for `hostname` is the fully qualified DNS hostname of the actual service that this virtual node represents.

```
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
```

Virtual nodes have capabilities, such as end-to-end encryption and health checks, that aren't covered in this tutorial. For more information, see the section called “Virtual nodes”. To see all available settings for a virtual node that you can set in the preceding spec, run the following command.

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

b. Deploy the virtual node.

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

c. View the details of the Kubernetes virtual node resource that was created.

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

Output

```
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. View the details of the virtual node that the controller created in App Mesh.

> **Note**
>
> Even though the name of the virtual node created in Kubernetes is *my-service-a*, the name of the virtual node created in App Mesh is *my-service-a_my-apps*. The controller appends the Kubernetes namespace name to the App Mesh virtual node name when it creates the App Mesh resource. The namespace name is added because in Kubernetes you can create virtual nodes with the same name in different namespaces, but in App Mesh a virtual node name must be unique within a mesh.

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

Output

```
{
    "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. Create an App Mesh virtual router. Virtual routers handle traffic for one or more virtual services within your mesh.

   a. Save the following contents to a file named `virtual-router.yaml` on your computer. The file is used to create a virtual router to route traffic to the virtual node named `my-service-a` that was created in the previous step. The controller creates the App Mesh virtual router and route resources. You can specify many more capabilities for your routes and use protocols other than `http`. For more information, see the section called "Virtual routers" and the section called "Routes". Notice that the virtual node name referenced is the Kubernetes virtual node name, not the App Mesh virtual node name that was created in App Mesh by the controller.

```
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
```

(Optional) To see all available settings for a virtual router that you can set in the preceding spec, run the following command.

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

To see all available settings for a route that you can set in the preceding spec, run the following command.

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

b. Deploy the virtual router.

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

c. View the Kubernetes virtual router resource that was created.

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

Abbreviated output

```
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. View the virtual router resource that the controller created in App Mesh. You specify `my-service-a-virtual-router_my-apps` for `name`, because when the controller created the virtual router in App Mesh, it appended the Kubernetes namespace name to the name of the virtual router.

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

Output

```
{
    "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. View the route resource that the controller created in App Mesh. A route resource was not created in Kubernetes because the route is part of the virtual router configuration in Kubernetes. The route information was shown in the Kubernetes resource detail in sub-step c. The controller did not append the Kubernetes namespace name to the App Mesh route name when it created the route in App Mesh because route names are unique to a virtual router.

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

Output

```
{
    "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. Create an App Mesh virtual service. A virtual service is an abstraction of a real service that is provided by a virtual node directly or indirectly by means of a virtual router. Dependent services call your virtual service by its name. Though the name doesn't matter to App Mesh, we recommend naming the virtual service the fully qualified domain name of the actual service that the virtual service represents. By naming your virtual services this way, you don't need to change your application code to reference a different name. The requests are routed to the virtual node or virtual router that is specified as the provider for the virtual service.

   a. Save the following contents to a file named `virtual-service.yaml` on your computer. The file is used to create a virtual service that uses a virtual router provider to route

traffic to the virtual node named `my-service-a` that was created in a previous step. The value for `awsName` in the `spec` is the fully qualified domain name (FQDN) of the actual Kubernetes service that this virtual service abstracts. The Kubernetes service is created in the section called “Step 3: Create or update services”. For more information, see the section called “Virtual services”.

```
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
```

To see all available settings for a virtual service that you can set in the preceding spec, run the following command.

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

b. Create the virtual service.

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

c. View the details of the Kubernetes virtual service resource that was created.

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

Output

```
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. View the details of the virtual service resource that the controller created in App Mesh. The Kubernetes controller did not append the Kubernetes namespace name to the App Mesh virtual service name when it created the virtual service in App Mesh because the virtual service's name is a unique FQDN.

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

Output

```
{
    "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"
    }
}
```

Though not covered in this tutorial, the controller can also deploy App Mesh the section called "Virtual gateways" and the section called "Gateway routes". For a walkthrough of deploying these resources with the controller, see Configuring Inbound Gateway, or a sample manifest that includes the resources on GitHub.

Step 3: Create or update services

Any pods that you want to use with App Mesh must have the App Mesh sidecar containers added to them. The injector automatically adds the sidecar containers to any pod deployed with a label that you specify.

1. Enable proxy authorization. We recommend that you enable each Kubernetes deployment to stream only the configuration for its own App Mesh virtual node.

    a. Save the following contents to a file named `proxy-auth.json` on your computer. Make sure to replace the *`alternate-colored values`* with your own.

```
{
    "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. Create the policy.

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

c. Create an IAM role, attach the policy you created in the previous step to it, create a Kubernetes service account, and bind the policy to the Kubernetes service account. The role enables the controller to add, remove, and change App Mesh resources.

```
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
```

If you prefer to create the service account using the AWS Management Console or AWS CLI, see Creating an IAM Role and policy for your service account in the *Amazon EKS User Guide*. If you use the AWS Management Console or AWS CLI to create the account, you also need to map the role to a Kubernetes service account. For more information, see Specifying an IAM role for your service account in the *Amazon EKS User Guide*.

2. (Optional) If you want to deploy your deployment to Fargate pods, then you need to create a Fargate profile. If you don't have `eksctl` installed, you can install it with the instructions in

Installing or Upgrading `eksctl` in the *Amazon EKS User Guide*. If you'd prefer to create the profile using the console, see Creating a Fargate profile in the *Amazon EKS User Guide*.

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

3. Create a Kubernetes service and deployment. If you have an existing deployment that you want to use with App Mesh, then you need to deploy a virtual node, as you did in sub-step 3 of the section called "Step 2: Deploy App Mesh resources". Update your deployment to make sure that its label matches the label that you set on the virtual node, so that the sidecar containers are automatically added to the pods and the pods are redeployed.

   a. Save the following contents to a file named `example-service.yaml` on your computer. If you change the namespace name and are using Fargate pods, make sure that the namespace name matches the namespace name that you defined in your Fargate profile.

```
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**
>
> The value for the app `matchLabels` `selector` in the spec must match the value that you specified when you created the virtual node in sub-step 3 of the section called "Step 2: Deploy App Mesh resources", or the sidecar containers won't be injected into the pod. In the previous example, the value for the label is `my-app-1`. If you deploy a virtual gateway, rather than a virtual node, then the `Deployment` manifest should include only the Envoy container. For more information about the image to use, see *Envoy*. For a sample manfest, see the deployment example on GitHub.

b. Deploy the service.

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

c. View the service and deployment.

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

Output

```
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. View the details for one of the pods that was deployed.

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

Abbreviated output

```
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
```

In the preceding output, you can see that the `proxyinit` and `envoy` containers were added to the pod by the controller. If you deployed the example service to Fargate, then the `envoy` container was added to the pod by the controller, but the `proxyinit` container was not.

4. (Optional) Install add-ons such as Prometheus, Grafana, AWS X-Ray, Jaeger, and Datadog. For more information, see App Mesh add-ons on GitHub and the Observability section of the App Mesh User Guide.

> **Note**
>
> For more examples and walkthroughs for App Mesh, see the App Mesh examples repository.

Step 4: Clean up

Remove all of the example resources created in this tutorial. The controller also removes the resources that were created in the `my-mesh` App Mesh service mesh.

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

If you created a Fargate profile for the example service, then remove it.

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

Delete the mesh.

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

(Optional) You can remove the Kubernetes integration components.

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

(Optional) If you deployed the Kubernetes integration components to Fargate, then delete the Fargate profile.

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

Getting started with AWS App Mesh and Amazon EC2

This topic helps you use AWS App Mesh with an actual service that is running on Amazon EC2. This tutorial covers basic features of several App Mesh resource types.

Scenario

To illustrate how to use App Mesh, assume that you have an application with the following characteristics:

- Consists of two services named `serviceA` and `serviceB`.
- Both services are registered to a namespace named `apps.local`.
- `ServiceA` communicates with `serviceB` over HTTP/2, port 80.
- You have already deployed version 2 of `serviceB` and registered it with the name `serviceBv2` in the `apps.local` namespace.

You have the following requirements:

- You want to send 75 percent of the traffic from `serviceA` to `serviceB` and 25 percent of the traffic to `serviceBv2` to validate that `serviceBv2` is bug free before you send 100 percent of the traffic from `serviceA` to it.
- You want to be able to easily adjust the traffic weighting so that 100 percent of the traffic goes to `serviceBv2` once it is proven to be reliable. Once all traffic is being sent to `serviceBv2`, you want to discontinue `serviceB`.
- You do not want to have to change any existing application code or service discovery registration for your actual services to meet the previous requirements.

To meet your requirements, you decide to create an App Mesh service mesh with virtual services, virtual nodes, a virtual router, and a route. After implementing your mesh, you update your services to use the Envoy proxy. Once updated, your services communicate with each other through the Envoy proxy rather than directly with each other.

Prerequisites

App Mesh supports Linux services that are registered with DNS, AWS Cloud Map, or both. To use this getting started guide, we recommend that you have three existing services that are registered with DNS. You can create a service mesh and its resources even if the services don't exist, but you cannot use the mesh until you have deployed actual services.

If you don't already have services running, you can launch Amazon EC2 instances and deploy applications to them. For more information, see Tutorial: Getting started with Amazon EC2 Linux instances in the Amazon EC2 User Guide for Linux Instances. The remaining steps assume that the actual services are named `serviceA`, `serviceB`, and `serviceBv2` and that all services are discoverable through a namespace named `apps.local`.

Step 1: Create a mesh and virtual service

A service mesh is a logical boundary for network traffic between the services that reside within it. For more information, see Service Meshes. A virtual service is an abstraction of an actual service. For more information, see Virtual services.

Create the following resources:

- A mesh named `apps`, since all of the services in the scenario are registered to the `apps.local` namespace.
- A virtual service named `serviceb.apps.local`, since the virtual service represents a service that is discoverable with that name, and you don't want to change your code to reference another name. A virtual service named `servicea.apps.local` is added in a later step.

You can use the AWS Management Console or the AWS CLI version 1.18.116 or higher or 2.0.38 or higher to complete the following steps. If using the AWS CLI, use the `aws --version` command to check your installed AWS CLI version. If you don't have version 1.18.116 or higher or 2.0.38 or higher installed, then you must install or update the AWS CLI. Select the tab for the tool that you want to use.

AWS Management Console

1. Open the App Mesh console first-run wizard at https://console.aws.amazon.com/appmesh/get-started.
2. For **Mesh name**, enter **apps**.

3. For **Virtual service name**, enter **`serviceb.apps.local`**.
4. To continue, choose **Next**.

AWS CLI

1. Create a mesh with the `create-mesh` command.

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

2. Create a virtual service with the `create-virtual-service` command.

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

Step 2: Create a virtual node

A virtual node acts as a logical pointer to an actual service. For more information, see Virtual nodes.

Create a virtual node named `serviceB`, since one of the virtual nodes represents the actual service named `serviceB`. The actual service that the virtual node represents is discoverable through DNS with a hostname of `serviceb.apps.local`. Alternately, you can discover actual services using AWS Cloud Map. The virtual node listens for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported, as are health checks. You create virtual nodes for `serviceA` and `serviceBv2` in a later step.

AWS Management Console

1. For **Virtual node name**, enter **`serviceB`**.
2. For **Service discovery method**, choose **DNS** and enter **`serviceb.apps.local`** for **DNS hostname**.
3. Under **Listener configuration**, choose **http2** for **Protocol** and enter **`80`** for **Port**.
4. To continue, choose **Next**.

AWS CLI

1. Create a file named `create-virtual-node-serviceb.json` with the following contents:

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

2. Create the virtual node with the create-virtual-node command using the JSON file as input.

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

Step 3: Create a virtual router and route

Virtual routers route traffic for one or more virtual services within your mesh. For more information, see Virtual routers and Routes.

Create the following resources:

- A virtual router named `serviceB`, since the `serviceB.apps.local` virtual service does not initiate outbound communication with any other service. Remember that the virtual service that you created previously is an abstraction of your actual `serviceb.apps.local` service. The

virtual service sends traffic to the virtual router. The virtual router listens for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported.

- A route named `serviceB`. It routes 100 percent of its traffic to the `serviceB` virtual node. The weight is in a later step once you add the `serviceBv2` virtual node. Though not covered in this guide, you can add additional filter criteria for the route and add a retry policy to cause the Envoy proxy to make multiple attempts to send traffic to a virtual node when it experiences a communication problem.

AWS Management Console

1. For **Virtual router name,** enter **`serviceB`**.
2. Under **Listener configuration**, choose **http2** for **Protocol** and specify **`80`** for **Port**.
3. For **Route name**, enter **`serviceB`**.
4. For **Route type**, choose **http2**.
5. For **Virtual node name** under **Target configuration**, select `serviceB` and enter **`100`** for **Weight**.
6. Under **Match configuration**, choose a **Method**.
7. To continue, choose **Next**.

AWS CLI

1. Create a virtual router.

   a. Create a file named `create-virtual-router.json` with the following contents:

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

```
    "virtualRouterName": "serviceB"
}
```

b. Create the virtual router with the create-virtual-router command using the JSON file as input.

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

2. Create a route.

a. Create a file named `create-route.json` with the following contents:

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

b. Create the route with the create-route command using the JSON file as input.

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

Step 4: Review and create

Review the settings against the previous instructions.

AWS Management Console

Choose **Edit** if you need to make changes in any section. Once you are satisfied with the settings, choose **Create mesh**.

The **Status** screen shows you all of the mesh resources that were created. You can see the created resources in the console by selecting **View mesh**.

AWS CLI

Review the settings of the mesh you created with the describe-mesh command.

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

Review the settings of the virtual service that you created with the describe-virtual-service command.

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

Review the settings of the virtual node that you created with the describe-virtual-node command.

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

Review the settings of the virtual router that you created with the describe-virtual-router command.

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

Review the settings of the route that you created with the describe-route command.

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

Step 5: Create additional resources

To complete the scenario, you need to:

- Create one virtual node named `serviceBv2` and another named `serviceA`. Both virtual nodes listen for requests over HTTP/2 port 80. For the `serviceA` virtual node, configure a backend of `serviceb.apps.local`. All outbound traffic from the `serviceA` virtual node is sent to the virtual service named `serviceb.apps.local`. Though not covered in this guide, you can also specify a file path to write access logs to for a virtual node.
- Create one additional virtual service named `servicea.apps.local`, which sends all traffic directly to the `serviceA` virtual node.
- Update the `serviceB` route that you created in a previous step to send 75 percent of its traffic to the `serviceB` virtual node and 25 percent of its traffic to the `serviceBv2` virtual node. Over time, you can continue to modify the weights until `serviceBv2` receives 100 percent of the traffic. Once all traffic is sent to `serviceBv2`, you can shut down and discontinue the `serviceB` virtual node and actual service. As you change weights, your code does not require any modification, because the `serviceb.apps.local` virtual and actual service names don't change. Recall that the `serviceb.apps.local` virtual service sends traffic to the virtual router, which routes the traffic to the virtual nodes. The service discovery names for the virtual nodes can be changed at any time.

AWS Management Console

1. In the left navigation pane, select **Meshes**.
2. Select the `apps` mesh that you created in a previous step.
3. In the left navigation pane, select **Virtual nodes**.
4. Choose **Create virtual node**.
5. For **Virtual node name**, enter **`serviceBv2`**, for **Service discovery method**, choose **DNS**, and for **DNS hostname**, enter **`servicebv2.apps.local`**.
6. For **Listener configuration**, select **http2** for **Protocol** and enter **`80`** for **Port**.
7. Choose **Create virtual node**.
8. Choose **Create virtual node** again. Enter **`serviceA`** for the **Virtual node name**. For **Service discovery method**, choose **DNS**, and for **DNS hostname**, enter **`servicea.apps.local`**.
9. For **Enter a virtual service name** under **New backend**, enter **`serviceb.apps.local`**.
10. Under **Listener configuration**, choose **http2** for **Protocol**, enter **`80`** for **Port**, and then choose **Create virtual node**.
11. In the left navigation pane, select **Virtual routers** and then select the `serviceB` virtual router from the list.

12. Under **Routes**, select the route named `ServiceB` that you created in a previous step, and choose **Edit**.
13. Under **Targets**, **Virtual node name**, change the value of **Weight** for `serviceB` to **75**.
14. Choose **Add target**, choose `serviceBv2` from the dropdown list, and set the value of **Weight** to **25**.
15. Choose **Save**.
16. In the left navigation pane, select **Virtual services** and then choose **Create virtual service**.
17. Enter **`servicea.apps.local`** for **Virtual service name**, select **Virtual node** for **Provider**, select `serviceA` for **Virtual node**, and then choose **Create virtual service.**

AWS CLI

1. Create the `serviceBv2` virtual node.
   a. Create a file named `create-virtual-node-servicebv2.json` with the following contents:

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

   b. Create the virtual node.

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

2. Create the `serviceA` virtual node.

   a. Create a file named `create-virtual-node-servicea.json` with the following contents:

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

   b. Create the virtual node.

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

3. Update the `serviceb.apps.local` virtual service that you created in a previous step to send its traffic to the `serviceB` virtual router. When the virtual service was originally

created, it did not send traffic anywhere, since the `serviceB` virtual router had not been created yet.

a. Create a file named `update-virtual-service.json` with the following contents:

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

b. Update the virtual service with the update-virtual-service command.

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

4. Update the `serviceB` route that you created in a previous step.

a. Create a file named `update-route.json` with the following contents:

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

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

b. Update the route with the update-route command.

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

5. Create the `serviceA` virtual service.

a. Create a file named `create-virtual-servicea.json` with the following contents:

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

b. Create the virtual service.

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

Mesh summary

Before you created the service mesh, you had three actual services named `servicea.apps.local`, `serviceb.apps.local`, and `servicebv2.apps.local`. In addition to the actual services, you now have a service mesh that contains the following resources that represent the actual services:

- Two virtual services. The proxy sends all traffic from the `servicea.apps.local` virtual service to the `serviceb.apps.local` virtual service through a virtual router.
- Three virtual nodes named `serviceA`, `serviceB`, and `serviceBv2`. The Envoy proxy uses the service discovery information configured for the virtual nodes to look up the IP addresses of the actual services.
- One virtual router with one route that instructs the Envoy proxy to route 75 percent of inbound traffic to the `serviceB` virtual node and 25 percent of the traffic to the `serviceBv2` virtual node.

Step 6: Update services

After creating your mesh, you need to complete the following tasks:

- Authorize the Envoy proxy that you deploy with each service to read the configuration of one or more virtual nodes. For more information about how to authorize the proxy, see Envoy Proxy authorization.
- To update your existing service, complete the steps that follow.

To configure an Amazon EC2 instance as a virtual node member

1. Create an IAM role.

   a. Create a file named `ec2-trust-relationship.json` with the following contents.

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

   b. Create an IAM role with the following command.

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

2. Attach IAM policies to the role that allow it to read from Amazon ECR and only the configuration of a specific App Mesh virtual node.

   a. Create a file named `virtual-node-policy.json` with the following contents. `apps` is the name of the mesh you created in the section called "Step 1: Create a mesh and virtual service" and `serviceB` is the name of the virtual node that you created in the section called "Step 2: Create a virtual node". Replace *111122223333* with your account ID and *us-west-2* with the Region that you created your mesh in.

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

   b. Create the policy with the following command.

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

   c. Attach the policy that you created in the previous step to the role so the role can read the configuration for only the `serviceB` virtual node from App Mesh.

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

   d. Attach the `AmazonEC2ContainerRegistryReadOnly` managed policy to the role so that it can pull the Envoy container image from Amazon ECR.

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

3. Launch an Amazon EC2 instance with the IAM role that you created.
4. Connect to your instance via SSH.
5. Install Docker and the AWS CLI on your instance according to your operating system documentation.
6. Authenticate to the Envoy Amazon ECR repository in the Region that you want your Docker client to pull the image from.

   - All Regions except `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`. You can replace *us-west-2* with any supported Region except `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`.

     ```
     $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` Region

     ```
     $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` Region

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

7. Run one of the following commands to start the App Mesh Envoy container on your instance, depending on which Region you want to pull the image from. The *apps* and *serviceB* values

are the mesh and virtual node names defined in the scenario. This information tells the proxy which virtual node configuration to read from App Mesh. To complete the scenario, you also need to complete these steps for the Amazon EC2 instances that host the services represented by the `serviceBv2` and `serviceA` virtual nodes. For your own application, replace these values with your own.

- All Regions except `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`. You can replace *Region-code* with any supported Region except the `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1` Regions. You can replace *1337* with any value between `0` and 2147483647.

```
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:v1.27.3.0-prod
```

- `me-south-1` Region. You can replace *1337* with any value between `0` and 2147483647.

```
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:v1.27.3.0-prod
```

- `ap-east-1` Region. You can replace *1337* with any value between `0` and 2147483647.

```
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:v1.27.3.0-prod
```

> **Note**
>
> The `APPMESH_RESOURCE_ARN` property requires version `1.15.0` or later of the Envoy image. For more information, see *Envoy*.

> **Important**
>
> Only version v1.9.0.0-prod or later is supported for use with App Mesh.

8. Select Show more below. Create a file named `envoy-networking.sh` on your instance with the following contents. Replace *8000* with the port that your application code uses for incoming traffic. You can change the value for `APPMESH_IGNORE_UID`, but the value must be the same as the value that you specified in the previous step; for example `1337`. You can add additional addresses to `APPMESH_EGRESS_IGNORED_IP` if necessary. Do not modify any other lines.

```
#!/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. To configure `iptables` rules to route application traffic to the Envoy proxy, run the script that you created in the previous step.

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

10. Start your virtual node application code.

> **Note**
>
> For more examples and walkthroughs for App Mesh, see the App Mesh examples repository.

App Mesh Roadmap

This is the experimental public roadmap for AWS App Mesh. The roadmap allows customers to know about our upcoming products and priorities, which helps customers plan how to use App Mesh in the future. This repository contains information about what we are working on and allows all AWS customers to give direct feedback.

App Mesh Roadmap

App Mesh Examples

You can find end-to-end walkthroughs showing AWS App Mesh in action and code examples for integrating with various AWS services in the following repository:

App Mesh Examples

App Mesh Concepts

App Mesh is composed of the following concepts.

- Service Meshes
- Virtual services
- Virtual gateways
- Virtual nodes
- Virtual routers

Service Meshes

A service mesh is a logical boundary for network traffic between the services that reside within it. After you create your service mesh, you can create virtual services, virtual nodes, virtual routers, and routes to distribute traffic between the applications in your mesh.

Creating a service mesh

> **Note**
>
> When creating a Mesh, you must add a namespace selector. If the namespace selector is empty, it selects all namespaces. To restrict the namespaces, use a label to associate App Mesh resources to the created mesh.

AWS Management Console

To create a service mesh using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose **Create mesh**.
3. For **Mesh name**, specify a name for your service mesh.
4. (Optional) Choose **Allow external traffic**. By default, proxies in the mesh only forward traffic between each other. If you allow external traffic, the proxies in the mesh also forward TCP traffic directly to services that aren't deployed with a proxy that is defined in the mesh.

> **Note**
>
> If you specify any backends on a virtual node when using `ALLOW_ALL`, you must specifiy all egress for that virtual node as backends. Otherwise, `ALLOW_ALL` will no longer work for that virtual node.

5. **IP version preference**

Control which IP version should be used for traffic within the mesh by toggling on **Override default IP version behavior**. By default, App Mesh uses a variety of IP versions.

> **Note**
>
> The mesh applies the IP preference to all of the virtual nodes and virtual gateways within a mesh. This behavior can be overridden on a individual virtual node by setting the IP preference when you make or edit the node. The IP preference can't be overridden on a virtual gateway because the configuration for virtual gateways that allows them to listen for both IPv4 and IPv6 traffic is the same regardless of which preference is set on the mesh.

- Default
  - Envoy's DNS resolver prefers `IPv6` and falls back to `IPv4`.
  - We use the `IPv4` address returned by AWS Cloud Map if available and falls back to using the `IPv6` address.
  - The endpoint created for the local app uses an `IPv4` address.
  - The Envoy listeners bind to all `IPv4` addresses.
- IPv6 preferred
  - Envoy's DNS resolver prefers `IPv6` and falls back to `IPv4`.
  - The `IPv6` address returned by AWS Cloud Map is used if available and falls back to using the `IPv4` address
  - The endpoint created for the local app uses an `IPv6` address.
  - The Envoy listeners bind to all `IPv4` and `IPv6` addresses.
- IPv4 preferred

- Envoy's DNS resolver prefers `IPv4` and falls back to `IPv6`.
- We use the `IPv4` address returned by AWS Cloud Map if available and falls back to using the `IPv6` address.
- The endpoint created for the local app uses an `IPv4` address.
- The Envoy listeners bind to all `IPv4` and `IPv6` addresses.

- IPv6 only
  - Envoy's DNS resolver only uses `IPv6`.
  - Only the `IPv6` address returned by AWS Cloud Map is used. If AWS Cloud Map returns an `IPv4` address, no IP addresses are used and empty results are returned to the Envoy.
  - The endpoint created for the local app uses an `IPv6` address.
  - The Envoy listeners bind to all `IPv4` and `IPv6` addresses.
- IPv4 only
  - Envoy's DNS resolver only uses `IPv4`.
  - Only the `IPv4` address returned by AWS Cloud Map is used. If AWS Cloud Map returns an `IPv6` address, no IP addresses are used and empty results are returned to the Envoy.
  - The endpoint created for the local app uses an `IPv4` address.
  - The Envoy listeners bind to all `IPv4` and `IPv6` addresses.

6. Choose **Create mesh** to finish.
7. (Optional) Share the mesh with other accounts. A shared mesh allows resources created by different accounts to communicate with each other in the same mesh. For more information, see Working with shared meshes.

AWS CLI

To create a mesh using the AWS CLI.

Create a service mesh using the following command (replace the *red* values with your own):

1. ```
   aws appmesh create-mesh --mesh-name meshName
   ```
2. Example output:
   ```

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

For more information on creating a mesh with the AWS CLI for App Mesh, see the create-mesh command in the AWS CLI reference.

Deleting a mesh

AWS Management Console

To delete a virtual gateway using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh you want to delete. All of the meshes that you own and that have been shared with you are listed.
3. In the confirmation box, type **`delete`** and then click on **Delete**.

AWS CLI

To delete a mesh using the AWS CLI

1. Use the following command to delete your mesh (replace the *red* values with your own):

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

2. Example output:

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

For more information on deleting a mesh with the AWS CLI for App Mesh, see the delete-mesh command in the AWS CLI reference.

Virtual services

A virtual service is an abstraction of a real service that is provided by a virtual node directly or indirectly by means of a virtual router. Dependent services call your virtual service by its `virtualServiceName`, and those requests are routed to the virtual node or virtual router that is specified as the provider for the virtual service.

Creating a virtual service

AWS Management Console

To create a virtual service using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh in which you want to create the virtual service. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual services** in the left navigation.
4. Choose **Create virtual service**.
5. For **Virtual service name**, choose a name for your virtual service. You can choose any name, but the service discovery name of the real service that you're targeting, such as `my-service.default.svc.cluster.local`, is recommended to make it easier to correlate your virtual services to real services. This way you don't need to change your code to reference a different name than your code currently references. The name that you specify must resolve to a non-loopback IP address because the app container must be able to successfully resolve the name before the request is sent to the Envoy proxy. You can use any non-loopback IP address because neither the app or proxy containers communicate with this IP address. The proxy communicates with other virtual services through the names you've configured for them in App Mesh, not through IP addresses to which the names resolve.
6. For **Provider**, choose the provider type for your virtual service:
   - If you want the virtual service to spread traffic across multiple virtual nodes, select **Virtual router** and then choose the virtual router to use from the drop-down menu.
   - If you want the virtual service to reach a virtual node directly without a virtual router, select **Virtual node** and then choose the virtual node to use from the drop-down menu.

> **Note**
>
> App Mesh may automatically create a default Envoy route retry policy for each virtual node provider that you define on or after July 29, 2020, even though you can't define such a policy through the App Mesh API. For more information, see Default route retry policy.

- If you don't want the virtual service to route traffic at this time (for example, if your virtual nodes or virtual router doesn't exist yet), choose **None**. You can update the provider for this virtual service later.

7. Choose **Create virtual service** to finish.

AWS CLI

To create a virtual service using the AWS CLI.

Create a virtual service with a virtual node provider using the following command and an input JSON file (replace the *red* values with your own):

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

2. Contents of **example** create-virtual-service-virtual-node.json:

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

3. Example output:

```
{
    "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"
    }
}
```

For more information on creating a virtual service with the AWS CLI for App Mesh, see the create-virtual-service command in the AWS CLI reference.

Deleting a virtual service

> **Note**
>
> You can't delete a virtual service that is referenced by a gateway route. You need to delete the gateway route first.

AWS Management Console

To delete a virtual service using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh from which you want to delete a virtual service. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual services** in the left navigation.

4. Choose the virtual service that you want to delete and click on **Delete** in the top right corner. You can only delete a virtual gateway where your account is listed as **Resource owner**.
5. In the confirmation box, type **`delete`** and then click on **Delete**.

AWS CLI

To delete a virtual service using the AWS CLI

1. Use the following command to delete your virtual service (replace the *`red`* values with your own):

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

2. Example output:

```
{
    "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"
    }
}
```

For more information on deleting a virtual service with the AWS CLI for App Mesh, see the delete-virtual-service command in the AWS CLI reference.

Virtual gateways

A virtual gateway allows resources that are outside of your mesh to communicate to resources that are inside of your mesh. The virtual gateway represents an Envoy proxy running in an Amazon ECS service, in a Kubernetes service, or on an Amazon EC2 instance. Unlike a virtual node, which represents Envoy running with an application, a virtual gateway represents Envoy deployed by itself.

External resources must be able to resolve a DNS name to an IP address assigned to the service or instance that runs Envoy. Envoy can then access all of the App Mesh configuration for resources that are inside of the mesh. The configuration for handling the incoming requests at the Virtual Gateway are specified using Gateway Routes.

Important

A virtual gateway with a HTTP or HTTP2 listener rewrites the incoming request's hostname to the Gateway Route target Virtual Service's name, and the matched prefix from the Gateway Route is rewritten to `/`, by default. For example, if you have configured the Gateway route match prefix to `/chapter`, and, if the incoming request is `/chapter/1`, the request would be rewritten to `/1`. To configure rewrites, refer to the Creating a gateway route section from Gateway Routes.
When creating a virtual gateway, `proxyConfiguration` and `user` should not be configured.

To complete an end-to-end walkthrough, see Configuring Inbound Gateway.

Creating a virtual gateway

> **Note**
>
> When creating a Virtual Gateway, you must add a namespace selector with a label to identify the list of namespaces with which to associate Gateway Routes to the created Virtual Gateway.

AWS Management Console

To create a virtual gateway using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh in which you want to create the virtual gateway. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual gateways** in the left navigation.
4. Choose **Create virtual gateway**.
5. For **Virtual gateway name**, enter a name for your virtual gateway.
6. (Optional, but recommended) Configure **Client policy defaults**.
    a. (Optional) Select **Enforce TLS** if you want the gateway to only communicate with virtual services using Transport Layer Security (TLS).
    b. (Optional) For **Ports**, specify one or more ports on which you want to enforce TLS communication with virtual services.
    c. For **Validation method**, select one of the following options. The certificate that you specify must already exist and meet specific requirements. For more information, see Certificate requirements.
        - **AWS Private Certificate Authority** hosting – Select one or more existing **Certificates**.
        - **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret that Envoy fetches using the Secret Discovery Service.
        - **Local file hosting** – Specify the path to the **Certificate chain** file on the file system where Envoy is deployed.

d. (Optional) Enter a **Subject Alternative Name**. To add additional SANs, select **Add SAN**. SANs must be FQDN or URI formatted.

e. (Optional) Select **Provide client certificate** and one of the options below to provide a client certificate when a server requests it and enable mutual TLS authentication. To learn more about mutual TLS, see the App Mesh Mutual TLS Authentication docs.

- **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret that Envoy fetches using the Secret Discovery Service.
- **Local file hosting** – Specify the path to the **Certificate chain** file, as well as the **Private key**, on the file system where Envoy is deployed. For a complete, end-to-end walk through of deploying a mesh with a sample application using encryption with local files, see Configuring TLS with File Provided TLS Certificates on GitHub.

7. (Optional) To configure logging, selected **Logging**. Enter the **HTTP access logs path** that you want Envoy to use. We recommend the `/dev/stdout` path so that you can use Docker log drivers to export your Envoy logs to a service such as Amazon CloudWatch Logs.

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

8. Configure the **Listener**.

a. Select a **Protocol** and specify the **Port** on which Envoy listens for traffic. The **http** listener permits connection transition to websockets. You can click **Add Listener** to add multiple listeners. The **Remove** button will remove that listener.

b. (Optional) **Enable connection pool**

Connection pooling limits the number of connections that the Virtual Gateway Envoy can concurrently establish. It is intended to protect your Envoy instance from being overwhelmed with connections and lets you adjust traffic shaping for the needs of your applications.

You can configure destination-side connection pool settings for a virtual gateway listener. App Mesh sets the client-side connection pool settings to infinite by default, simplifying mesh configuration.

> **Note**
>
> The `connectionPool` and `connectionPoolportMapping` protocols must be the same. If your listener protocol is `grpc` or `http2`, specify `maxRequests` only. If your listener protocol is `http`, you can specify both `maxConnections` and `maxPendingRequests`.

- For **Maximum connections**, specify the maximum number of outbound connections.
- For **Maximum requests**, specify maximum number of parallel requests that can be established with the Virtual Gateway Envoy.
- (Optional) For **Maximum pending requests**, specify the number of overflowing requests after **Maximum connections** that an Envoy queues. The default value is `2147483647`.

c. (Optional) If you want to configure a health check for your listener, then select **Enable health check**.

A health check policy is optional, but if you specify any values for a health policy, then you must specify values for **Healthy threshold**, **Health check interval**, **Health check protocol**, **Timeout period**, and **Unhealthy threshold**.

- For **Health check protocol**, choose a protocol. If you select **grpc**, then your service must conform to the GRPC Health Checking Protocol.
- For **Health check port**, specify the port that the health check should run on.
- For **Healthy threshold**, specify the number of consecutive successful health checks that must occur before declaring the listener healthy.
- For **Health check interval**, specify the time period in milliseconds between each health check execution.
- For **Path**, specify the destination path for the health check request. This value is only used if the **Health check protocol** is `http` or `http2`. The value is ignored for other protocols.
- For **Timeout period**, specify the amount of time to wait when receiving a response from the health check in milliseconds.
- For **Unhealthy threshold**, specify the number of consecutive failed health checks that must occur before declaring the listener unhealthy.

d. (Optional) If you want to specify whether clients communicate with this virtual gateway using TLS, then select **Enable TLS termination**.

- For **Mode**, select the mode that you want TLS to be configured for on the listener.
- For **Certificate method**, select one of the following options. The certificate must meet specific requirements. For more information, see Certificate requirements.
  - **AWS Certificate Manager hosting** – Select an existing **Certificate**.
  - **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret that Envoy fetches using the Secret Discovery Service.
  - **Local file hosting** – Specify the path to the **Certificate chain** and **Private key** files on the file system where Envoy is deployed.
- (Optional) Select **Require client certificate** and one of the options below to enable mutual TLS authentication if the client provides a certificate. To learn more about mutual TLS, see the App Mesh Mutual TLS Authentication docs.
  - **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret that Envoy fetches using the Secret Discovery Service.
  - **Local file hosting** – Specify the path to the **Certificate chain** file on the file system where Envoy is deployed.
- (Optional) Enter a **Subject Alternative Name**. To add additional SANs, select **Add SAN**. SANs must be FQDN or URI formatted.

9. Choose **Create virtual gateway** to finish.

AWS CLI

To create a virtual gateway using the AWS CLI.

Create a virtual gateway using the following command and input JSON (replace the *red* values with your own):

1.
```
aws appmesh create-virtual-gateway \
--mesh-name meshName \
--virtual-gateway-name virtualGatewayName \
--cli-input-json file://create-virtual-gateway.json
```

2. Contents of **example** create-virtual-gateway.json:

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

3. Example output:

```
{
    "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"
```

```
    }
}
```

For more information on creating a virtual gateway with the AWS CLI for App Mesh, see the create-virtual-gateway command in the AWS CLI reference.

Deploy virtual gateway

Deploy an Amazon ECS or Kubernetes service that contains only the Envoy container. You can also deploy the Envoy container on an Amazon EC2 instance. For more information, see Getting started with App Mesh and Amazon EC2. For more information on how to deploy on Amazon ECS see Getting started with App Mesh and Amazon ECS or Getting started with AWS App Mesh and Kubernetes to deploy to Kubernetes. You need to set the `APPMESH_RESOURCE_ARN` environment variable to `mesh/`*`mesh-name`*`/virtualGateway/`*`virtual-gateway-name`* and you must not specify proxy configuration so that the proxy's traffic doesn't get redirected to itself. By default, App Mesh uses the name of the resource you specified in `APPMESH_RESOURCE_ARN` when Envoy is referring to itself in metrics and traces. You can override this behavior by setting the `APPMESH_RESOURCE_CLUSTER` environment variable with your own name.

We recommend that you deploy multiple instances of the container and set up a Network Load Balancer to load balance traffic to the instances. The service discovery name of the load balancer is the name that you want external services to use to access resources that are in the mesh, such as *`myapp.example.com`*. For more information see Creating a Network Load Balancer (Amazon ECS), Creating an External Load Balancer (Kubernetes), or Tutorial: Increase the availability of your application on Amazon EC2. You can also find more examples and walkthroughs in our App Mesh examples.

Enable proxy authorization for Envoy. For more information, see Envoy Proxy authorization.

Deleting a virtual gateway

AWS Management Console

To delete a virtual gateway using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh from which you want to delete a virtual gateway. All of the meshes that you own and that have been shared with you are listed.

3. Choose **Virtual gateways** in the left navigation.
4. Choose the virtual gateway that you want to delete and select **Delete**. You cannot delete a virtual gateway if it has any associated gateway routes. You must delete any associated gateway routes first. You can only delete a virtual gateway where your account is listed as **Resource owner**.
5. In the confirmation box, type **`delete`** and then select **Delete**.

AWS CLI

To delete a virtual gateway using the AWS CLI

1. Use the following command to delete your virtual gateway (replace the *`red`* values with your own):

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

2. Example output:

```
{
    "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"
    }
}
```

For more information on deleting a virtual gateway with the AWS CLI for App Mesh, see the delete-virtual-gateway command in the AWS CLI reference.

Gateway routes

A gateway route is attached to a virtual gateway and routes traffic to an existing virtual service. If a route matches a request, it can distribute traffic to a target virtual service. This topic helps you work with gateway routes in a service mesh.

Creating a gateway route

AWS Management Console

To create a gateway route using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh in which you want to create the gateway route. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual gateways** in the left navigation.
4. Choose the virtual gateway with which you want to associate a new gateway route. If none are listed, then you need to create a virtual gateway first. You can only create a gateway route for a virtual gateway of which your account is listed as the **Resource owner**.
5. In the **Gateway routes** table, choose **Create gateway route**.
6. For **Gateway route name**, specify the name to use for your gateway route.
7. For **Gateway route type** choose either **http**, **http2**, or **grpc**.
8. Select an existing **Virtual service name**. If none are listed, then you need to create a virtual service first.

9. Choose the port that corresponds to the target for **Virtual service provider port**. Virtual service provider port is **required** when the provider (router or node) of the selected virtual service has multiple listeners.
10. (Optional) For **Priority**, specify the priority for this gateway route.
11. For **Match** configuration, specify:

    - If **http/http2** is the selected type:
      - (Optional) **Method** - Specifies the method header to be matched in the incoming **http/http2** requests.
      - (Optional) **Port match** - Match the port for incoming traffic. Port match is **required** if this virtual gateway has multiple listeners.
      - (Optional) **Exact/Suffix hostname** - Specifies the hostname that should be matched on the incoming request to route to the target virtual service.
      - (Optional) **Prefix/Exact/Regex path** - The method of matching the path of the URL.
        - **Prefix match** - A matched request by a gateway route is rewritten to the target virtual service's name and the matched prefix is rewritten to `/`, by default. Depending on how you configure your virtual service, it could use a virtual router to route the request to different virtual nodes, based on specific prefixes or headers.

      > **Important**
      >
      > - You can't specify either `/aws-appmesh*` or `/aws-app-mesh*` for **Prefix match**. These prefixes are reserved for future App Mesh internal use.
      > - If multiple gateway routes are defined, then a request is matched to the route with the longest prefix. For example, if two gateway routes existed, with one having a prefix of `/chapter` and one having a prefix of `/`, then a request for `www.example.com/chapter/` would be matched to the gateway route with the `/chapter` prefix.

      > **Note**
      >
      > If you enable **Path/Prefix** based matching, App Mesh enables path normalization (normalize_path and merge_slashes) to minimize the probability of path confusion vulnerabilities.

Path confusion vulnerabilities occur when parties participating in the request use different path representations.

- **Exact match** - The exact parameter disables the partial matching for a route and makes sure that it only returns the route if the path is an **EXACT** match to the current URL.
- **Regex match** - Used to describe patterns where multiple URLs may actually identify a single page on the website.

- (Optional) **Query parameters** - This field allows you to match on the query parameters.
- (Optional) **Headers** - Specifies the headers for **http** and **http2**. It should match the incoming request to route to the target virtual service..

- If **grpc** is the selected type:
  - **Hostname match type** and (optional) **Exact/Suffix match** - Specifies the hostname that should be matched on the incoming request to route to the target virtual service.
  - **grpc service name** - The **grpc** service acts as an API for your application and is defined with ProtoBuf.

    Important

    You can't specify `/aws.app-mesh*` or `aws.appmesh` for the **Service name**. These service names are reserved for future App Mesh internal use.

  - (Optional) **Metadata** - Specifies the metadata for **grpc**. It should match the incoming request to route to the target virtual service.

12. (Optional) For **Rewrite** configuration:

    - If **http/http2** is the selected type:
      - If **Prefix** is the selected match type:
        - **Override automatic rewrite of hostname** - By default the hostname is rewritten to the target virtual service's name.
        - **Override automatic rewrite of prefix** - When toggled on, **Prefix rewrite** specifies the value of the rewritten prefix.
      - If **Exact Path** is the selected match type:

- **Override automatic rewrite of hostname** - by default the hostname is rewritten to the target virtual service's name.
- **Path rewrite** - Specifies the value of the rewritten path. No default path.

- If **Regex Path** is the selected match type:
  - **Override automatic rewrite of hostname** - by default the hostname is rewritten to the target virtual service's name.
  - **Path rewrite** - Specifies the value of the rewritten path. No default path.

- If **grpc** is the selected type:
  - **Override automatic rewrite of hostname** - By default the hostname is rewritten to the target virtual service's name.

13. Choose **Create gateway route** to finish.

AWS CLI

To create a gateway route using the AWS CLI.

Create a gateway route using the following command and input JSON (replace the *red* values with your own):

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. Contents of **example** create-gateway-route.json:

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

```
                }
            }
        }
    }
}
```

3. Example output:

```
{
    "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"
    }
}
```

For more information on creating a gateway route with the AWS CLI for App Mesh, see the create-gateway-route command in the AWS CLI reference.

Deleting a gateway route

AWS Management Console

To delete a gateway route using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh from which you want to delete a gateway route. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual gateways** in the left navigation.
4. Choose the virtual gateway from which you want to delete a gateway route.
5. In the **Gateway routes** table, choose the gateway route that you want to delete and select **Delete**. You can only delete a gateway route if your account is listed as **Resource owner**.
6. In the confirmation box, type **`delete`** and then click on **Delete**.

AWS CLI

To delete a gateway route using the AWS CLI

1. Use the following command to delete your gateway route (replace the *`red`* values with your own):

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

2. Example output:

```
{
    "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"
    }
}
```

For more information on deleting a gateway route with the AWS CLI for App Mesh, see the delete-gateway-route command in the AWS CLI reference.

Virtual nodes

A virtual node acts as a logical pointer to a particular task group, such as an Amazon ECS service or a Kubernetes deployment. When you create a virtual node, you must specify a service discovery method for your task group. Any inbound traffic that your virtual node expects is specified as a *listener*. Any virtual service that a virtual node sends outbound traffic to is specified as a *backend*.

The response metadata for your new virtual node contains the Amazon Resource Name (ARN) that is associated with the virtual node. Set this value as the `APPMESH_RESOURCE_ARN` environment variable for your task group's Envoy proxy container in your Amazon ECS task definition or Kubernetes pod spec. For example, the value could be `arn:aws:appmesh:us-west-2:111122223333:mesh/myMesh/virtualNode/myVirtualNode`. This is then mapped to the `node.id` and `node.cluster` Envoy parameters. You must be using `1.15.0` or later of the Envoy image when setting this variable. For more information about App Mesh Envoy variables, see *Envoy*.

> **Note**
>
> By default, App Mesh uses the name of the resource you specified in `APPMESH_RESOURCE_ARN` when Envoy is referring to itself in metrics and traces. You can override this behavior by setting the `APPMESH_RESOURCE_CLUSTER` environment variable with your own name.

Creating a virtual node

AWS Management Console

To create a virtual node using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh that you want to create the virtual node in. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual nodes** in the left navigation.
4. Choose **Create virtual node** and then specify settings for your virtual node.
5. For **Virtual node name**, enter a name for your virtual node.
6. For **Service discovery method**, choose one of the following options:

   - **DNS** – Specify the **DNS hostname** of the actual service that the virtual node represents. The Envoy proxy is deployed in an Amazon VPC. The proxy sends name resolution requests to the DNS server that is configured for the VPC. If the hostname resolves, the DNS server returns one or more IP addresses. For more information about VPC DNS settings, see Using DNS with your VPC. For **DNS response type** (optional), specify the types of endpoints returned by the DNS resolver. **Load Balancer** means that the DNS

resolver returns a loadbalanced set of endpoints. **Endpoints** means that the DNS resolver is returning all the endpoints. By default, the response type is assumed to be **Load Balancer**.

> **Note**
>
> If you use **Route53**, you'll need to use **Load Balancer**.

- **AWS Cloud Map** – Specify an existing **Service name** and HTTP **Namespace**. Optionally, you can also specify attributes that App Mesh can query AWS Cloud Map for by selecting **Add row** and specifying a **Key** and **Value**. Only instances that match all of the specified key/value pairs will be returned. To use AWS Cloud Map, your account must have the `AWSServiceRoleForAppMesh` service-linked role. For more information about AWS Cloud Map, see the AWS Cloud Map Developer Guide.
- **None** – Select if your virtual node doesn't expect any inbound traffic.

7. **IP version preference**

Control which IP version should be used for traffic within the mesh by toggling on **Override default IP version behavior**. By default, App Mesh uses a variety of IP versions.

> **Note**
>
> Setting the IP preference on the virtual node only overrides the IP preference set for the mesh on this specific node.

- Default
  - Envoy's DNS resolver prefers `IPv6` and falls back to `IPv4`.
  - We use the `IPv4` address returned by AWS Cloud Map if available and falls back to using the `IPv6` address.
  - The endpoint created for the local app uses an `IPv4` address.
  - The Envoy listeners bind to all `IPv4` addresses.
- IPv6 preferred
  - Envoy's DNS resolver prefers `IPv6` and falls back to `IPv4`.
  - The `IPv6` address returned by AWS Cloud Map is used if available and falls back to using the `IPv4` address

- The endpoint created for the local app uses an `IPv6` address.
- The Envoy listeners bind to all `IPv4` and `IPv6` addresses.

- IPv4 preferred
  - Envoy's DNS resolver prefers `IPv4` and falls back to `IPv6`.
  - We use the `IPv4` address returned by AWS Cloud Map if available and falls back to using the `IPv6` address.
  - The endpoint created for the local app uses an `IPv4` address.
  - The Envoy listeners bind to all `IPv4` and `IPv6` addresses.
- IPv6 only
  - Envoy's DNS resolver only uses `IPv6`.
  - Only the `IPv6` address returned by AWS Cloud Map is used. If AWS Cloud Map returns an `IPv4` address, no IP addresses are used and empty results are returned to the Envoy.
  - The endpoint created for the local app uses an `IPv6` address.
  - The Envoy listeners bind to all `IPv4` and `IPv6` addresses.
- IPv4 only
  - Envoy's DNS resolver only uses `IPv4`.
  - Only the `IPv4` address returned by AWS Cloud Map is used. If AWS Cloud Map returns an `IPv6` address, no IP addresses are used and empty results are returned to the Envoy.
  - The endpoint created for the local app uses an `IPv4` address.
  - The Envoy listeners bind to all `IPv4` and `IPv6` addresses.

8. **(Optional) Client policy defaults – Configure default requirements when communicating to backend virtual services.**

> **Note**
>
> - If you want to enable Transport Layer Security (TLS) for an existing virtual node, then we recommend that you create a new virtual node, which represents the same service as the existing virtual node, on which to enable TLS. Then gradually shift traffic to the new virtual node using a virtual router and route. For more information about creating a route and adjusting weights for the transition, see Routes. If you update an existing, traffic-serving virtual node with TLS, there is

a chance that the downstream client Envoy proxies will receive TLS validation context before the Envoy proxy for the virtual node that you have updated receives the certificate. This can cause TLS negotiation errors on the downstream Envoy proxies.

- Proxy authorization must be enabled for the Envoy proxy deployed with the application represented by the backend service's virtual nodes. We recommend that when you enable proxy authorization, you restrict access to only the virtual nodes that this virtual node is communicating with.

- (Optional) Select **Enforce TLS** if you want to require the virtual node to communicate with all backends using Transport Layer Security (TLS).
- (Optional) If you only want to require the use of TLS for one or more specific ports, then enter a number in **Ports**. To add additional ports, select **Add port**. If you don't specify any ports, TLS is enforced for all ports.
- For **Validation method**, select one of the following options. The certificate that you specify must already exist and meet specific requirements. For more information, see Certificate requirements.
  - **AWS Private Certificate Authority** hosting – Select one or more existing **Certificates**. For a complete, end-to-end walk through of deploying a mesh with a sample application using encryption with an ACM certificate, see Configuring TLS with AWS Certificate Manager on GitHub.
  - **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret Envoy will fetch using the Secret Discovery Service.
  - **Local file hosting** – Specify the path to the **Certificate chain** file on the file system where Envoy is deployed. For a complete, end-to-end walk through of deploying a mesh with a sample application using encryption with local files, see Configuring TLS with File Provided TLS Certificates on GitHub.
- (Optional) Enter a **Subject Alternative Name**. To add additional SANs, select **Add SAN**. SANs must be FQDN or URI formatted.
- (Optional) Select **Provide client certificate** and one of the options below to provide a client certificate when a server requests it and enable mutual TLS authentication . To learn more about mutual TLS, see the App Mesh Mutual TLS Authentication docs.
  - **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret Envoy will fetch using the Secret Discovery Service.

- **Local file hosting** – Specify the path to the **Certificate chain** file, as well as the **Private key**, on the file system where Envoy is deployed.

9. **(Optional) Service backends – Specify the App Mesh virtual service that the virtual node will communicate with.**

   - Enter an App Mesh virtual service name or full Amazon Resource Name (ARN) for the virtual service that your virtual node communicates with.
   - (Optional) If you want to set unique TLS settings for a backend, select **TLS settings** and then select **Override defaults**.
     - (Optional) Select **Enforce TLS** if you want to require the virtual node to communicate with all backends using TLS.
     - (Optional) If you only want to require the use of TLS for one or more specific ports, then enter a number in **Ports**. To add additional ports, select **Add port**. If you don't specify any ports, TLS is enforced for all ports.
     - For **Validation method**, select one of the following options. The certificate that you specify must already exist and meet specific requirements. For more information, see Certificate requirements.
       - **AWS Private Certificate Authority** hosting – Select one or more existing **Certificates**.
       - **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret Envoy will fetch using the Secret Discovery Service.
       - **Local file hosting** – Specify the path to the **Certificate chain** file on the file system where Envoy is deployed.
     - (Optional) Enter a **Subject Alternative Name**. To add additional SANs, select **Add SAN**. SANs must be FQDN or URI formatted.
     - (Optional) Select **Provide client certificate** and one of the options below to provide a client certificate when a server requests it and enable mutual TLS authentication. To learn more about mutual TLS, see the App Mesh Mutual TLS Authentication docs.
       - **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret Envoy will fetch using the Secret Discovery Service.
       - **Local file hosting** – Specify the path to the **Certificate chain** file, as well as the **Private key**, on the file system where Envoy is deployed.
   - To add additional backends, select **Add backend**.

10. (Optional) **Logging**

To configure logging, enter the HTTP access logs path that you want Envoy to use. We recommend the `/dev/stdout` path so that you can use Docker log drivers to export your Envoy logs to a service such as Amazon CloudWatch Logs.

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

11. **Listener configuration**

    Listeners support `HTTP`,`HTTP/2`, `GRPC`, and `TCP` protocols. `HTTPS` is not supported.

    a. If your virtual node expects inbound traffic, specify a **Port** and **Protocol** for the **Listener**. The **http** listener permits connection transition to websockets. You can click **Add Listener** to add multiple listeners. The **Remove** button will remove that listener.

    b. **(Optional) Enable connection pool**

    Connection pooling limits the number of connections that an Envoy can concurrently establish with the local application cluster. It is intended to protect your local application from being overwhelmed with connections and lets you adjust traffic shaping for the needs of your applications.

    You can configure destination-side connection pool settings for a virtual node listener. App Mesh sets the client-side connection pool settings to infinite by default, simplifying mesh configuration.

    > **Note**
    >
    > The connectionPool and portMapping protocols must be the same. If your listener protocol is tcp, specify maxConnections only. If your listener protocol is grpc or http2, specify maxRequests only. If your listener protocol is http, you can specify both maxConnections and maxPendingRequests.

    - For **Maximum connections**, specify the maximum number of outbound connections.

- (Optional) For **Maximum pending requests**, specify the number of overflowing requests after **Maximum connections** that an Envoy will queue. The default value is 2147483647.

c. **(Optional) Enable outlier detection**

Outlier detection applied at the client Envoy allows clients to take near-immediate action on connections with observed known bad failures. It is a form of a circuit breaker implementation that tracks the health status of individual hosts in the upstream service.

Outlier detection dynamically determines whether endpoints in an upstream cluster are performing unlike the others and removes them from the healthy load balancing set.

> **Note**
>
> To effectively configure outlier detection for a server Virtual Node, the service discovery method of that Virtual Node can be either AWS Cloud Map or DNS with the response type field set to `ENDPOINTS`. If you use DNS service discovery method with response type as `LOADBALANCER`, the Envoy proxy would only elect a single IP address for routing to the upstream service. This nullifies the outlier detection behavior of ejecting an unhealthy host from a set of hosts. Refer to the Service discovery method section for more details on the Envoy proxy's behavior in relation to the service discovery type.

- For **Server errors**, specify the number of consecutive 5xx errors required for ejection.
- For **Outlier detection interval**, specify the time interval and unit between ejection sweep analysis.
- For **Base ejection duration**, specify the base amount of time and unit for which a host is ejected.
- For **Ejection percentage**, specify the maximum percentage of hosts in the load balancing pool that can be ejected.

d. **(Optional) Enable health check – Configure settings for a health check policy.**

A health check policy is optional, but if you specify any values for a health policy, then you must specify values for **Healthy threshold**, **Health check interval**, **Health check protocol**, **Timeout period**, and **Unhealthy threshold**.

- For **Health check protocol**, choose a protocol. If you select **grpc**, then your service must conform to the GRPC Health Checking Protocol.
- For **Health check port**, specify the port that the health check should run on.
- For **Healthy threshold**, specify the number of consecutive successful health checks that must occur before declaring the listener healthy.
- For **Health check interval**, specify the time period in milliseconds between each health check execution.
- For **Path**, specify the destination path for the health check request. This value is only used if the **Health check protocol** is `http` or `http2`. The value is ignored for other protocols.
- For **Timeout period**, specify the amount of time to wait when receiving a response from the health check, in milliseconds.
- For **Unhealthy threshold**, specify the number of consecutive failed health checks that must occur before declaring the listener unhealthy.

e. **(Optional) Enable TLS termination – Configure how other virtual nodes communicate with this virtual node using TLS.**

- For **Mode**, select the mode you want TLS to be configured for on the listener.
- For **Certificate method**, select one of the following options. The certificate must meet specific requirements. For more information, see Certificate requirements.
  - **AWS Certificate Manager hosting** – Select an existing **Certificate**.
  - **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret Envoy will fetch using the Secret Discovery Service.
  - **Local file hosting** – Specify the path to the **Certificate chain** file, as well as the **Private key**, on the file system where the Envoy proxy is deployed.
- (Optional) Select **Require client certificates** and one of the options below to enable mutual TLS authentication when a client provides a certificate. To learn more about mutual TLS, see the App Mesh Mutual TLS Authentication docs.

- **Envoy Secret Discovery Service (SDS)** hosting – Enter the name of the secret Envoy will fetch using the Secret Discovery Service.
- **Local file hosting** – Specify the path to the **Certificate chain** file on the file system where Envoy is deployed.

- (Optional) Enter a **Subject Alternative Name**. To add additional SANs, select **Add SAN**. SANs must be FQDN or URI formatted.

f. **(Optional) Timeouts**

> **Note**
>
> If you specify a timeout greater than the default, make sure to set up a virtual router and a route with a timeout greater than the default. However, if you decrease the timeout to a value that is lower than the default, it's optional to update the timeouts at Route. For more information, see Routes.

- **Request timeout** – You can specify a request timeout if you selected **grpc**, **http**, or **http2** for the listener's **Protocol**. The default is 15 seconds. A value of `0` disables the timeout.
- **Idle duration** – You can specify an idle duration for any listener protocol. The default is 300 seconds.

12. Choose **Create virtual node** to finish.

AWS CLI

To create a virtual node using the AWS CLI.

Create a virtual node that uses DNS for service discovery using the following command and an input JSON file (replace the *red* values with your own):

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

2. Contents of **example** create-virtual-node-dns.json:

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

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

3. Example output:

```
{
    "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"
    }
}
```

For more information on creating a virtual node with the AWS CLI for App Mesh, see the create-virtual-node command in the AWS CLI reference.

Deleting a virtual node

> **Note**
>
> You can't delete a virtual node if it is specified as a target in any route or as a provider in any virtual service.

AWS Management Console

To delete a virtual node using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh that you want to delete a virtual node from. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual nodes** in the left navigation.
4. In the **Virtual Nodes** table, choose the virtual node that you want to delete and select **Delete**. To delete a virtual node, your account ID must be listed in either the **Mesh owner** or the **Resource owner** columns of the virtual node.
5. In the confirmation box, type **`delete`** and then select **Delete**.

AWS CLI

To delete a virtual node using the AWS CLI

1. Use the following command to delete your virtual node (replace the *red* values with your own):

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

2. Example output:

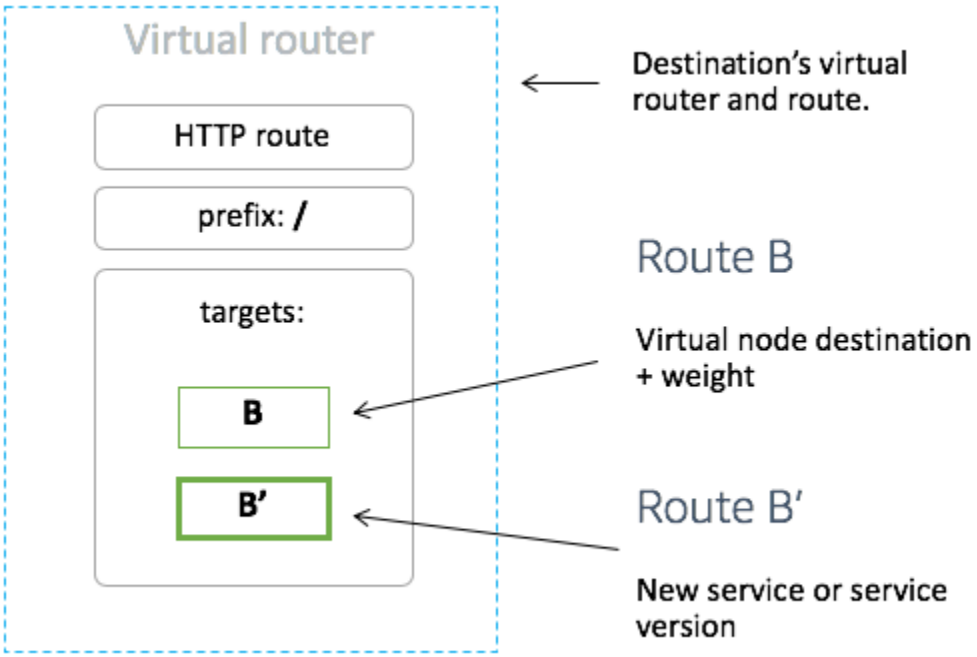
```
{
    "virtualNode": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualNode/nodeName",
            "createdAt": "2022-04-06T09:12:24.348000-05:00",
            "lastUpdatedAt": "2022-04-07T11:03:48.120000-05:00",
            "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"
    }
}
```

For more information on deleting a virtual node with the AWS CLI for App Mesh, see the delete-virtual-node command in the AWS CLI reference.

Virtual routers

Virtual routers handle traffic for one or more virtual services within your mesh. After you create a virtual router, you can create and associate routes for your virtual router that direct incoming requests to different virtual nodes.



Any inbound traffic that your virtual router expects should be specified as a *listener*.

Creating a virtual router

AWS Management Console

To create a virtual router using the AWS Management Console

> **Note**
>
> When creating a Virtual Router, you must add a namespace selector with a label to identify the list of namespaces to associate Routes to the created Virtual Router.

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh that you want to create the virtual router in. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual routers** in the left navigation.
4. Choose **Create virtual router**.
5. For **Virtual router name**, specify a name for your virtual router. Up to 255 letters, numbers, hyphens, and underscores are allowed.
6. (Optional) For **Listener** configuration, specify a **Port** and **Protocol** for your virtual router. The `http` listener permits connection transition to websockets. You can click **Add Listener** to add multiple listeners. The **Remove** button will remove that listener.
7. Choose **Create virtual router** to finish.

AWS CLI

To create a virtual router using the AWS CLI.

Create a virtual router using the following command and input JSON (replace the *red* values with your own):

1. ```
   aws appmesh create-virtual-router \
        --cli-input-json file://create-virtual-router.json
   ```
2. Contents of **example** create-virtual-router.json
3. ```
   {
   ```

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

4. Example output:

```
{
    "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"
    }
```

```
}
```

For more information on creating a virtual router with the AWS CLI for App Mesh, see the create-virtual-router command in the AWS CLI reference.

Deleting a virtual router

> **Note**
>
> You cannot delete a virtual router if it has any routes or if it is specified as a provider for any virtual service.

AWS Management Console

To delete a virtual router using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh that you want to delete a virtual router from. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual routers** in the left navigation.
4. In the **Virtual Routers** table, choose the virtual router that you want to delete and select **Delete** in the top right corner. To delete a virtual router, your account ID must be listed in either the **Mesh owner** or the **Resource owner** columns of the virtual router.
5. In the confirmation box, type **`delete`** and then click on **Delete**.

AWS CLI

To delete a virtual router using the AWS CLI

1. Use the following command to delete your virtual router (replace the *`red`* values with your own):

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

2. Example output:

```
{
    "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"
    }
}
```

For more information on deleting a virtual router with the AWS CLI for App Mesh, see the delete-virtual-router command in the AWS CLI reference.

Routes

A route is associated with a virtual router. The route is used to match requests for the virtual router and to distribute traffic to its associated virtual nodes. If a route matches a request, it can

distribute traffic to one or more target virtual nodes. You can specify relative weighting for each virtual node. This topic helps you work with routes in a service mesh.

Creating a route

AWS Management Console

To create a route using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh that you want to create the route in. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual routers** in the left navigation.
4. Choose the virtual router that you want to associate a new route with. If none are listed, then you need to create a virtual router first.
5. In the **Routes** table, choose **Create route**. To create a route, your account ID must be listed as the **Resource owner** of the route.
6. For **Route name**, specify the name to use for your route.
7. For **Route type**, choose the protocol that you want to route. The protocol that you select must match the listener protocol that you selected for your virtual router and the virtual node that you're routing traffic to.
8. (Optional) For **Route priority**, specify a priority from 0-1000 to use for your route. Routes are matched based on the specified value, where 0 is the highest priority.
9. (Optional) Choose **Additional configuration**. From the protocols down below, choose the protocol that you selected for **Route type** and specify settings in the console as desired.
10. For **Target configuration**, select the existing App Mesh virtual node to route traffic to and specify a **Weight**. You can choose **Add target** to add additional targets. The percentage for all targets must add up to 100. If no virtual nodes are listed, then you must create one first. If the selected virtual node has multiple listeners, **Target port** is **required**.
11. For **Match** configuration, specify:

    ***Match** configuration is not available for `tcp`*

    - If **http/http2** is the selected type:
      - (Optional) **Method** - specifies the method header to be matched in the incoming **http/http2** requests.

- (Optional) **Port match** - Match the port for incoming traffic. Port match is **required** if this virtual router has multiple listeners.
- (Optional) **Prefix/Exact/Regex path** - method of matching the path of the URL.
  - **Prefix match** - a matched request by a gateway route is rewritten to the target virtual service's name and the matched prefix is rewritten to `/`, by default. Depending on how you configure your virtual service, it could use a virtual router to route the request to different virtual nodes, based on specific prefixes or headers.

    > **Note**
    >
    > If you enable **Path/Prefix** based matching, App Mesh enables path normalization (normalize_path and merge_slashes) to minimize the probability of path confusion vulnerabilities.
    > Path confusion vulnerabilities occur when parties participating in the request use different path representations.

  - **Exact match** - the exact param disables the partial matching for a route and makes sure that it only returns the route if the path is an EXACT match to the current url.
  - **Regex match** - used to describe patterns where multiple URLs may actually identify a single page on the website.
- (Optional) **Query parameters** - this field allows you to match on the query parameters.
- (Optional) **Headers** - specifies the headers for **http** and **http2**. It should match the incoming request to route to the target virtual service..

- If **grpc** is the selected type:
  - **Service name** - the destination service for which to match the request.
  - **Method name** - the destination method for which to match the request.
  - (Optional) **Metadata** - specifies the `Match` based on the presence of metadata. All must match for the request to be processed.

12. Select **Create route**.

AWS CLI

To create a route using the AWS CLI.

Create a gRPC route using the following command and input JSON (replace the *red* values with your own):

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

2. Contents of **example** create-route-grpc.json

```
{
    "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. Example output:

```
{
    "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"
  }
}
```

For more information on creating a route with the AWS CLI for App Mesh, see the create-route command in the AWS CLI reference.

gRPC

(Optional) Match

- (Optional) Enter the **Service name** of the destination service to match the request for. If you don't specify a name, requests to any service are matched.
- (Optional) Enter the **Method name** of the destination method to match the request for. If you don't specify a name, requests to any method are matched. If you specify a method name, you must specify a service name.

(Optional) Metadata

Choose **Add metadata**.

- (Optional) Enter the **Metadata name** that you want to route based on, select a **Match type**, and enter a **Match value**. Selecting **Invert** will match the opposite. For example, if you specify a **Metadata name** of `myMetadata`, a **Match type** of **Exact**, a **Match value** of `123`, and select **Invert**, then the route is matched for any request that has a metadata name that starts with anything other than `123`.
- (Optional) Select **Add metadata** to add up to ten metadata items.

(Optional) Retry policy

A retry policy enables clients to protect themselves from intermittent network failures or intermittent server-side failures. A retry policy is optional, but recommended. The retry timeout values define the timeout per retry attempt (including the initial attempt). If you don't define a retry policy, then App Mesh may automatically create a default policy for each of your routes. For more information, see Default route retry policy.

- For **Retry timeout**, enter the number of units for the timeout duration. A value is required if you select any protocol retry event.
- For **Retry timeout unit**, select a unit. A value is required if you select any protocol retry event.
- For **Max retries**, enter the maximum number of retry attempts when the request fails. A value is required if you select any protocol retry event. We recommend a value of at least two.
- Select one or more **HTTP retry events**. We recommend selecting at least **stream-error** and **gateway-error**.
- Select a **TCP retry event**.
- Select one or more **gRPC retry events**. We recommend selecting at least **cancelled** and **unavailable**.

(Optional) Timeouts

- The default is 15 seconds. If you specified a **Retry policy**, then the duration that you specify here should always be greater than or equal to the retry duration multiplied by the **Max retries** that you defined in the **Retry policy** so that your retry policy can complete. If you specify a duration greater than 15 seconds, then make sure that the timeout specified for the listener of any virtual node **Target** is also greater than 15 seconds. For more information, see Virtual Nodes.

- A value of `0` disables the timeout.
- The maximum amount of time that the route can be idle.

HTTP and HTTP/2

(Optional) Match

- Specify the **Prefix** that the route should match. For example, if your virtual service name is `service-b.local` and you want the route to match requests to `service-b.local/metrics`, your prefix should be `/metrics`. Specifying `/` routes all traffic.
- (Optional) Select a **Method**.
- (Optional) Select a **Scheme**. Applicable only for HTTP2 routes.

(Optional) Headers

- (Optional) Select **Add header**. Enter the **Header name** that you want to route based on, select a **Match type**, and enter a **Match value**. Selecting **Invert** will match the opposite. For example, if you specify a header named `clientRequestId` with a **Prefix** of `123`, and select **Invert**, then the route is matched for any request that has a header that starts with anything other than `123`.
- (Optional) Select **Add header**. You can add up to ten headers.

(Optional) Retry policy

A retry policy enables clients to protect themselves from intermittent network failures or intermittent server-side failures. A retry policy is optional, but recommended. The retry timeout values define the timeout per retry attempt (including the initial attempt). If you don't define a retry policy, then App Mesh may automatically create a default policy for each of your routes. For more information, see Default route retry policy.

- For **Retry timeout**, enter the number of units for the timeout duration. A value is required if you select any protocol retry event.
- For **Retry timeout unit**, select a unit. A value is required if you select any protocol retry event.
- For **Max retries**, enter the maximum number of retry attempts when the request fails. A value is required if you select any protocol retry event. We recommend a value of at least two.
- Select one or more **HTTP retry events**. We recommend selecting at least **stream-error** and **gateway-error**.

- Select a **TCP retry event**.

(Optional) Timeouts

- **Request timeout** – The default is 15 seconds. If you specified a **Retry policy**, then the duration that you specify here should always be greater than or equal to the retry duration multiplied by the **Max retries** that you defined in the **Retry policy** so that your retry policy can complete.
- **Idle duration** – The default is 300 seconds.
- A value of `0` disables the timeout.

> **Note**
>
> If you specify a timeout greater than the default, make sure that the timeout specified for the listener for all virtual node participants is also greater than the default. However, if you decrease the timeout to a value that is lower than the default, it's optional to update the timeouts at virtual nodes. For more information, see Virtual Nodes.

TCP

(Optional) Timeouts

- **Idle duration** – The default is 300 seconds.
- A value of `0` disables the timeout.

Deleting a route

AWS Management Console

To delete a route using the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh from which you want to delete a route. All of the meshes that you own and that have been shared with you are listed.
3. Choose **Virtual routers** in the left navigation.
4. Choose the router from which you want to delete a route.

5. In the **Routes** table, choose the route that you want to delete and select **Delete** in the top right corner.
6. In the confirmation box, type **`delete`** and then click on **Delete**.

AWS CLI

To delete a route using the AWS CLI

1. Use the following command to delete your route (replace the *red* values with your own):

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

2. Example output:

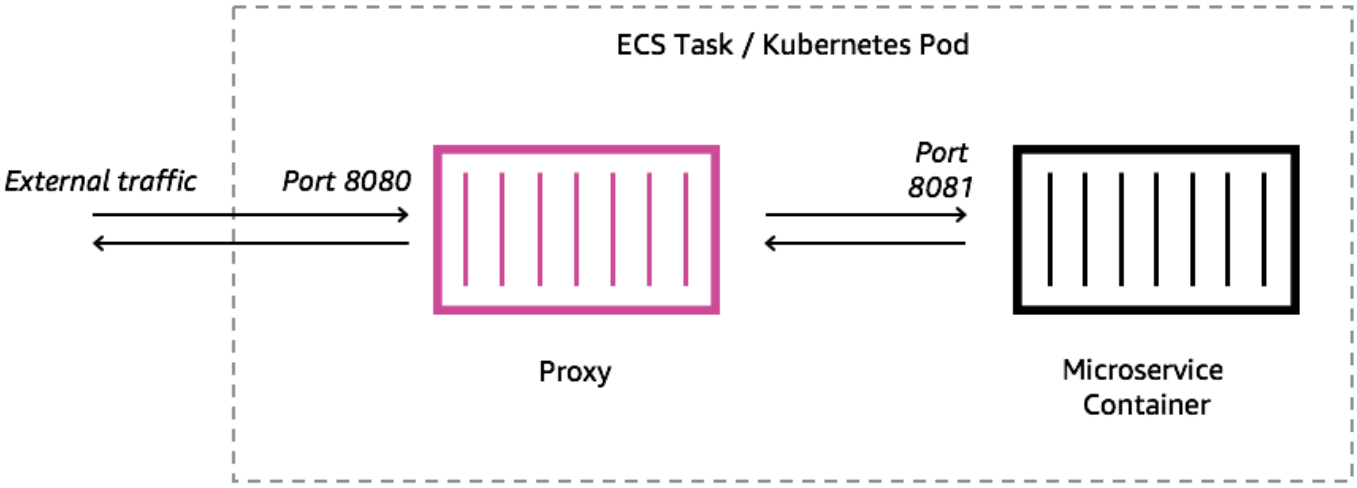
```
{
    "route": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualRouter/routerName/route/routeName",
            "createdAt": "2022-04-06T13:46:54.750000-05:00",
            "lastUpdatedAt": "2022-04-07T10:43:57.152000-05:00",
            "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"
    }
}
```

For more information on deleting a route with the AWS CLI for App Mesh, see the delete-route command in the AWS CLI reference.

Envoy image

AWS App Mesh is a service mesh based on the Envoy proxy.



You must add an Envoy proxy to the Amazon ECS task, Kubernetes pod, or Amazon EC2 instance represented by your App Mesh endpoint, such as a virtual node or virtual gateway. App Mesh vends an Envoy proxy container image that is patched with the latest vulnerability and performance updates. App Mesh tests each new Envoy proxy release against the App Mesh feature set before making a new image available to you.

Envoy image variants

App Mesh provides two variants of the Envoy proxy container image. The differences between the two is how the Envoy proxy communicates to the App Mesh data plane and how the Envoy proxies communicate with each other. One is a standard image, which communicates with the standard App Mesh service endpoints. The other variant is FIPS-compliant, which communicates with the App Mesh FIPS service endpoints and enforces FIPS cryptography in TLS communication between App Mesh services.

You can choose either a Regional image from the list below or an image from our public repository named `aws-appmesh-envoy`.

Important

- Starting from June 30, 2023, only Envoy image `v1.17.2.0-prod` or later is compatible for use with App Mesh. For current customers using an Envoy image before `v1.17.2.0`, although existing envoys will continue to be compatible, we strongly recommend migrating to the latest version.
- As a best practice, upgrading the Envoy version to the latest version on a regular basis is highly recommended. Only the latest Envoy version is validated with the most recent security patches, feature releases, and performance improvements.
- Version `1.17` was a significant update to Envoy. See Updating/migrating to Envoy 1.17 for more details.
- Version `1.20.0.1` or later is `ARM64` compatible.
- For `IPv6` support, Envoy version `1.20` or later is required.

All supported Regions other than `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`. You can replace *`Region-code`* with any Region other than `me-south-1`, `ap-east-1`, `ap-southeast-3`, `eu-south-1`, `il-central-1`, and `af-south-1`.

Standard

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

`me-south-1`

Standard

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

ap-east-1

Standard

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

ap-southeast-3

Standard

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-
fips
```

eu-south-1

Standard

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

il-central-1

Standard

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
564877687649.dkr.ecr.il-central-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-
fips
```

`af-south-1`

Standard

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
924023996002.dkr.ecr.af-south-1.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

`Public repository`

Standard

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.27.3.0-prod
```

FIPS-compliant

```
public.ecr.aws/appmesh/aws-appmesh-envoy:v1.27.3.0-prod-fips
```

Note

We recommend allocating 512 CPU units and at least 64 MiB of memory to the Envoy container. On Fargate the lowest amount of memory that you can set is 1024 MiB of memory. Resource allocation to the Envoy container can be increased if container insights or other metrics indicate insufficient resources due to higher load.

Note

All `aws-appmesh-envoy` image release versions starting from `v1.22.0.0` are built as a distroless Docker image. We made this change so that we could reduce the image size and reduce our vulnerability exposure in unused packages present in the image. If you are

building on top of aws-appmesh-envoy image and are relying on some of the AL2 base packages (e.g. yum) and functionalities, then we suggest you copy the binaries from inside an `aws-appmesh-envoy` image to build a new Docker image with AL2 base.
Run this script to generate a custom docker image with the tag `aws-appmesh-envoy:v1.22.0.0-prod-al2`:

```
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 .
```

Access to this container image in Amazon ECR is controlled by AWS Identity and Access Management (IAM). As a result, you must use IAM to verify that you have read access to Amazon ECR. For example, when using Amazon ECS, you can assign an appropriate task execution role to an Amazon ECS task. If you use IAM policies that limit access to specific Amazon ECR resources, make sure to verify that you allow access to the Region specific Amazon Resource Name (ARN) that identifies the `aws-appmesh-envoy` repository. For example, in the `us-west-2` Region, you allow access to the following resource: `arn:aws:ecr:us-west-2:840364872350:repository/aws-appmesh-envoy`. For more information, see Amazon ECR Managed Policies. If you're using Docker on an Amazon EC2 instance, then authenticate Docker to the repository. For more information, see Registry authentication.

We occasionally release new App Mesh features that depend on Envoy changes that have not been merged to the upstream Envoy images yet. To use these new App Mesh features before the Envoy changes are merged upstream, you must use the App Mesh-vended Envoy container image. For

a list of changes, see the App Mesh GitHub roadmap issues with the `Envoy Upstream` label. We recommend that you use the App Mesh Envoy container image as the best supported option.

Envoy configuration variables

Use the following environment variables to configure the Envoy containers for your App Mesh virtual node task groups.

> **Note**
>
> App Mesh Envoy 1.17 doesn't supports Envoy's **v2 xDS** API. If you're using Envoy configuration variables that accept Envoy config files, they must be updated to the latest **v3 xDS** API.

Required variables

The following environment variable is required for all App Mesh Envoy containers. This variable can only be used with version `1.15.0` or later of the Envoy image. If you're using an earlier version of the image, then you must set the `APPMESH_VIRTUAL_NODE_NAME` variable instead.

`APPMESH_RESOURCE_ARN`

When you add the Envoy container to a task group, set this environment variable to the ARN of the virtual node or the virtual gateway that the task group represents. The following list contains example ARNs:

- **Virtual node** – arn:aws:appmesh:*Region-code*:*111122223333*:mesh/*meshName*/virtualNode/*virtualNodeName*
- **Virtual gateway** – arn:aws:appmesh:*Region-code*:*111122223333*:mesh/*meshName*/virtualGateway/*virtualGatewayName*

When using the App Mesh Preview Channel, ARNs must use the *us-west-2* Region and use `appmesh-preview`, instead of `appmesh`. For example, the ARN of a virtual node in the App Mesh Preview Channel is `arn:aws:`**`appmesh-preview:us-west-2:`***`111122223333`*`:mesh/`*`meshName`*`/virtualNode/`*`virtualNodeName`*.

Optional variables

The following environment variable is optional for App Mesh Envoy containers.

`ENVOY_LOG_LEVEL`

Specifies the log level for the Envoy container.

Valid values: `trace`, `debug`, `info`, `warn`, `error`, `critical`, `off`

Default: `info`

`ENVOY_INITIAL_FETCH_TIMEOUT`

Specifies the amount of time Envoy waits for the first configuration response from the management server during the initialization process.

For more information, see Configuration sources in Envoy Documentation. When set to `0`, there is no timeout.

Default: `0`

`ENVOY_CONCURRENCY`

Sets the `--concurrency` command line option while starting the Envoy. This is not set by default. This option is available from Envoy version `v1.24.0.0-prod` or above.

For more information, see Command line options in Envoy Documentation.

Admin variables

Use these environment variables to configure Envoy's administrative interface.

`ENVOY_ADMIN_ACCESS_PORT`

Specify a custom admin port for Envoy to listen on. Default: `9901`.

> **Note**
>
> The Envoy admin port should be different from any listener port on the virtual gateway or virtual node

`ENVOY_ADMIN_ACCESS_LOG_FILE`

Specify a custom path to write Envoy access logs to. Default: `/tmp/envoy_admin_access.log`.

`ENVOY_ADMIN_ACCESS_ENABLE_IPV6`

Toggles Envoy's administration interface to accept `IPv6` traffic, which allows this interface to accept both `IPv4` and `IPv6` traffic. By default this flag is set to false, and Envoy only listens to `IPv4` traffic. This variable can only be used with Envoy image version 1.22.0 or later.

Agent variables

Use these environment variables to configure the AWS App Mesh Agent for Envoy. For more information, see App Mesh Agent for Envoy.

`APPNET_ENVOY_RESTART_COUNT`

Specifies the number of times that the Agent restarts the Envoy proxy process within a running task or pod if it exits. The Agent also logs the exit status every time Envoy exits to ease troubleshooting. The default value of this variable is `0`. When the default value is set, the Agent doesn't attempt to restart the process.

Default: `0`

Maximum: `10`

`PID_POLL_INTERVAL_MS`

Specifies the interval in milliseconds at which the Envoy proxy's process state is checked by the Agent. The default value is `100`.

Default: `100`

Minimum: `100`

Maximum: `1000`

`LISTENER_DRAIN_WAIT_TIME_S`

Specifies the amount of time in seconds the Envoy proxy waits for active connections to close before the process exits.

Default: `20`

Minimum: `5`

Maximum: `110`

`APPNET_AGENT_ADMIN_MODE`

Starts Agent's management interface server and binds it to either a tcp address or a unix socket.

Valid values: `tcp`, `uds`

`APPNET_AGENT_HTTP_PORT`

Specify a port to be used for binding Agent's management interface in `tcp` mode. Ensure port value is > `1024` if `uid` != `0`. Ensure port is less than `65535`.

Default: `9902`

`APPNET_AGENT_ADMIN_UDS_PATH`

Specify unix domain socket path for Agent's management interface in `uds` mode.

Default: `/var/run/ecs/appnet_admin.sock`

Tracing variables

You can configure none or one of the following tracing drivers.

AWS X-Ray variables

Use the following environment variables to configure App Mesh with AWS X-Ray. For more information, see the AWS X-Ray Developer Guide.

`ENABLE_ENVOY_XRAY_TRACING`

Enables X-Ray tracing using `127.0.0.1:2000` as the default daemon endpoint. To enable, set the value to `1`. The default value is `0`.

`XRAY_DAEMON_PORT`

Specify a port value to override the default X-Ray daemon port: `2000`.

`XRAY_SAMPLING_RATE`

Specify a sampling rate to override the X-Ray tracer's default sampling rate of `0.05` (5%). Specify the value as a decimal between `0` and `1.00` (100%). This value is overridden if `XRAY_SAMPLING_RULE_MANIFEST` is specified. This variable is supported with Envoy images of version `v1.19.1.1-prod` and later.

`XRAY_SAMPLING_RULE_MANIFEST`

Specify a file path in the Envoy container file system to configure the localized custom sampling rules for the X-Ray tracer. For more information, see Sampling rules in the *AWS X-Ray Developer Guide*. This variable is supported with Envoy images of version `v1.19.1.0-prod` and later.

`XRAY_SEGMENT_NAME`

Specify a segment name for traces to override the default X-Ray segment name. By default this value will be set as `mesh/resourceName`. This variable is supported with Envoy image version `v1.23.1.0-prod` or later.

Datadog tracing variables

The following environment variables help you configure App Mesh with the Datadog agent tracer. For more information, see Agent Configuration in the Datadog documentation.

`ENABLE_ENVOY_DATADOG_TRACING`

Enables Datadog trace collection using `127.0.0.1:8126` as the default Datadog agent endpoint. To enable, set the value to `1` (default value is `0`).

`DATADOG_TRACER_PORT`

Specify a port value to override the default Datadog agent port: `8126`.

`DATADOG_TRACER_ADDRESS`

Specify an IP address to override the default Datadog agent address: `127.0.0.1`.

`DD_SERVICE`

Specify a service name for traces to override the default Datadog service name: `envoy-meshName/virtualNodeName`. This variable is supported with Envoy images of version `v1.18.3.0-prod` and later.

Jaeger tracing variables

Use the following environment variables to configure App Mesh with Jaeger tracing. For more information, see Getting Started in the Jaeger documentation. These variables are supported with Envoy images of version `1.16.1.0-prod` and later.

`ENABLE_ENVOY_JAEGER_TRACING`

Enables Jaeger trace collection using `127.0.0.1:9411` as the default Jaeger endpoint. To enable, set the value to `1` (default value is `0`).

`JAEGER_TRACER_PORT`

Specify a port value to override the default Jaeger port: `9411`.

`JAEGER_TRACER_ADDRESS`

Specify an IP address to override the default Jaeger address: `127.0.0.1`.

`JAEGER_TRACER_VERSION`

Specify whether the collector needs traces in `JSON` or `PROTO` encoded format. By default this value will be set to `PROTO`. This variable is supported with Envoy image version `v1.23.1.0-prod` or later.

Envoy tracing variable

Set the following environment variable to use your own tracing configuration.

`ENVOY_TRACING_CFG_FILE`

Specify a file path in the Envoy container file system. For more information, see `config.trace.v3.Tracing` in the Envoy documentation.

> **Note**
>
> If the tracing configuration requires specifying a tracing cluster, make sure to configure the associated cluster configuration under `static_resources` in the same tracing config file. For example, Zipkin has a `collector_cluster` field for the cluster name that hosts the trace collectors, and that cluster needs to be statically defined.

DogStatsD variables

Use the following environment variables to configure App Mesh with DogStatsD. For more information, see the DogStatsD documentation.

`ENABLE_ENVOY_DOG_STATSD`

Enables DogStatsD stats using `127.0.0.1:8125` as the default daemon endpoint. To enable, set the value to `1`.

`STATSD_PORT`

Specify a port value to override the default DogStatsD daemon port.

`STATSD_ADDRESS`

Specify an IP address value to override the default DogStatsD daemon IP address. Default: `127.0.0.1`. This variable can only be used with version `1.15.0` or later of the Envoy image.

`STATSD_SOCKET_PATH`

Specify a unix domain socket for the DogStatsD daemon. If this variable isn't specified and DogStatsD is enabled, then this value defaults to the DogStatsD daemon IP address port of `127.0.0.1:8125`. If the `ENVOY_STATS_SINKS_CFG_FILE` variable is specified containing a stats sinks configuration, it overrides all of the DogStatsD variables. This variable is supported with Envoy image version `v1.19.1.0-prod` or later.

App Mesh variables

The following variables help you configure App Mesh.

`APPMESH_PREVIEW`

Set the value to `1` to connect to the App Mesh Preview Channel endpoint. For more information about using the App Mesh Preview Channel, see App Mesh Preview Channel.

`APPMESH_RESOURCE_CLUSTER`

By default, App Mesh uses the name of the resource that you specified in `APPMESH_RESOURCE_ARN` when Envoy is referring to itself in metrics and traces. You can override this behavior by setting the `APPMESH_RESOURCE_CLUSTER` environment variable with your own name. This variable can only be used with version `1.15.0` or later of the Envoy image.

`APPMESH_METRIC_EXTENSION_VERSION`

Set the value to 1 to enable the App Mesh metrics extension. For more information about using the App Mesh metrics extension, see Metrics extension for App Mesh.

`APPMESH_DUALSTACK_ENDPOINT`

Set the value to 1 to connect to App Mesh Dual Stack endpoint. When this flag is set, Envoy uses our dual stack capable domain. By default this flag is set to false and only connects to our `IPv4` domain. This variable can only be used with Envoy image version 1.22.0 or later.

Envoy stats variables

Use the following environment variables to configure App Mesh with Envoy Stats. For more information, see the Envoy Stats documentation.

`ENABLE_ENVOY_STATS_TAGS`

Enables the use of App Mesh defined tags `appmesh.mesh` and `appmesh.virtual_node`. For more information, see config.metrics.v3.TagSpecifier in the Envoy documentation. To enable, set the value to 1.

`ENVOY_STATS_CONFIG_FILE`

Specify a file path in the Envoy container file system to override the default Stats tags configuration file with your own. For more information, see config.metrics.v3.StatsConfig.

> **Note**
>
> Setting a customized stats configuration that includes stats filters might lead Envoy to enter a state where it will no longer properly synchronize with the App Mesh state of the world. This is a bug in Envoy. Our recommendation is to not perform any filtering of statistics in Envoy. If filtering is absolutely necessary, we have a listed a couple of workarounds in this issue on our roadmap.

`ENVOY_STATS_SINKS_CFG_FILE`

Specify a file path in the Envoy container file system to override the default configuration with your own. For more information, see config.metrics.v3.StatsSink in the Envoy documentation.

Deprecated variables

The environment variables `APPMESH_VIRTUAL_NODE_NAME` and `APPMESH_RESOURCE_NAME` are no longer supported in Envoy version `1.15.0` or later. However, they're still supported for existing meshes. Instead of using these variables with Envoy version `1.15.0` or later, use `APPMESH_RESOURCE_ARN` for all App Mesh endpoints.

Envoy defaults set by App Mesh

The following sections provide information about the Envoy defaults for the route retry policy and circuit breaker that are set by App Mesh.

Default route retry policy

If you had no meshes in your account before July 29, 2020, App Mesh automatically creates a default Envoy route retry policy for all HTTP, HTTP/2, and gRPC requests in any mesh in your account on or after July 29, 2020. If you had any meshes in your account before July 29, 2020, then no default policy was created for any Envoy routes that existed before, on, or after July 29, 2020. This is unless you open a ticket with AWS support. After support processes the ticket, the default policy is created for any future Envoy routes that App Mesh creates on or after the date that the ticket was processed. For more information about Envoy route retry policies, see config.route.v3.RetryPolicy in the Envoy documentation.

App Mesh creates an Envoy route when you either create an App Mesh route or define a virtual node provider for an App Mesh virtual service. Though you can create an App Mesh route retry policy, you can't create an App Mesh retry policy for a virtual node provider.

The default policy isn't visible through the App Mesh API. The default policy is only visible through Envoy. To view the configuration, enable the administration interface and send a request to Envoy for a `config_dump`. The default policy includes the following settings:

- **Max retries** – 2
- **gRPC retry events** – `UNAVAILABLE`
- **HTTP retry events** – `503`

> **Note**
>
> It's not possible to create an App Mesh route retry policy that looks for a specific HTTP error code. However, an App Mesh route retry policy can look for `server-error` or `gateway-error`. Both of these include `503` errors. For more information, see Routes.

- **TCP retry event** – `connect-failure` and `refused-stream`

> **Note**
>
> It's not possible to create an App Mesh route retry policy that looks for either of these events. However, an App Mesh route retry policy can look for `connection-error`, which is equivalent to `connect-failure`. For more information, see Routes.

- **Reset** – Envoy attempts a retry if the upstream server doesn't respond at all (disconnect/reset/read timeout).

Default circuit breaker

When you deploy an Envoy in App Mesh, Envoy default values are set for some of the circuit breaker settings. For more information, see cluster.CircuitBreakers.Thresholds in the Envoy documentation. These settings aren't visible through the App Mesh API. The settings are only visible through Envoy. To view the configuration, enable the administration interface and send a request to Envoy for a `config_dump`.

If you had no meshes in your account before July 29, 2020, then for each Envoy that you deploy in a mesh created on or after July 29, 2020, App Mesh effectively disables circuit breakers by changing the Envoy default values for the settings that follow. If you had any meshes in your account before July 29, 2020, the Envoy default values are set for any Envoy that you deploy in App Mesh on, or after July 29, 2020, unless you open a ticket with AWS support. Once support processes the ticket, then the App Mesh default values for the following Envoy settings are set by App Mesh on all Envoys that you deploy after the date that the ticket is processed:

- **`max_requests`** – 2147483647
- **`max_pending_requests`** – 2147483647
- **`max_connections`** – 2147483647

- **`max_retries`** – 2147483647

> **Note**
>
> No matter if your Envoys have the Envoy or App Mesh default circuit breaker values, you cannot modify the values.

Updating/migrating to Envoy 1.17

Secret Discovery Service with SPIRE

If you're using SPIRE (SPIFFE Runtime Environment) with App Mesh to distribute trust certificates to your services, verify that you're using at least version `0.12.0` of the SPIRE agent (released December 2020). This is the first version that can support Envoy versions after `1.16`.

Regular expression changes

Starting from Envoy `1.17`, App Mesh configures Envoy to use the RE2 regular expression engine by default. This change is apparent to most users, but matches in Routes or Gateway Routes no longer allows look-ahead or back-references in regular expressions.

Positive and Negative look-ahead

Positive - A positive look-ahead is a parenthesized expression that starts with `?=`:

```
(?=example)
```

These have the most utility when doing string replacement because they allow matching a string without consuming the characters as part of the match. Because App Mesh doesn't support regex string replacement, we recommend that you replace these with regular matches.

```
(example)
```

Negative - A negative look-ahead is a parenthesized expression that starts with `?!`.

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

The parenthesized expressions are used to assert that part of the expression doesn't match a given input. In most cases, you can replace these with a zero quantifier.

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

If the expression itself is a character class, you can negate the whole class and mark it optional using ?.

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

Depending on your use-case, you might also be able to change your routes to handle this.

```
{
    "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"
                        }
                    }
```

```
                ]
            }
        }
    }
}
```

The first route match looks for a header that starts with “prefix” but not followed by “suffix.” The second route acts to match all other headers that begin with “prefix,” including those that end in “suffix.” Instead, these can also be reversed as a way to remove the negative look-ahead.

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

```
    }
}
```

This example reverses the routes to provide higher priority to headers that end in “suffix,” and all other headers that start with “prefix” are matched in the lower-priority route.

Back references

A back-reference is a way to write shorter expressions by repeating to a previous parenthesized group. They have this form.

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

A backslash \ followed by a number acts as a placeholder for the n-th parenthesized group in the expression. In this example, \1 is used as an alternative way to write (group1) a second time.

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

These can be removed by simply replacing the back-reference with the group being referenced as in the example.

Agent for Envoy

The Agent is a process manager within the Envoy image that's vended for App Mesh. The Agent ensures Envoy remains running, stays healthy, and reduces downtime. It filters Envoy statistics and ancillary data to provide a distilled view of the Envoy proxy’s operation in App Mesh. This can help you troubleshooting related errors quicker.

You can use the Agent to configure the number of times that you want to restart the Envoy proxy in the event that the proxy becomes unhealthy. If a failure occurs, the Agent logs the conclusive exit status when Envoy exits. You can use this when troubleshooting the failure. The Agent also facilitates Envoy connection draining, which helps make your applications more resilient to failures.

Configure the Agent for Envoy using these variables:

- APPNET_ENVOY_RESTART_COUNT – When this variable is set to a non-zero value, the Agent attempts to restart the Envoy proxy process up to the number that you set when it deems the proxy process status unhealthy on polling. This helps reduce downtime by providing faster

restart compared to a task or pod replacement by the container orchestrator in the case of proxy health check failures.

- `PID_POLL_INTERVAL_MS` – When configuring this variable, the default is kept to `100`. When set to this value, you allow for faster detection and restart of the Envoy process when it exits compared to task or pod replacement through container orchestrator health checks.
- `LISTENER_DRAIN_WAIT_TIME_S` – When configuring this variable, consider the container orchestrator timeout that's set for stopping the task or pod. For example, if this value is greater than the orchestrator timeout, the Envoy proxy can only drain for the duration until the orchestrator forcefully stops the task or pod.
- `APPNET_AGENT_ADMIN_MODE` – When this variable is set to `tcp` or `uds`, the Agent provides a local management interface. This management interface serves as a safe endpoint to interact with the Envoy proxy and provides the following APIs for health checks, telemetry data and summarizes the operating condition of the proxy.
  - `GET /status` – Queries Envoy stats and returns server information.
  - `POST /drain_listeners` – Drains all inbound listeners.
  - `POST /enableLogging?level=<desired_level>` – Change Envoy logging level across all loggers.
  - `GET /stats/prometheus` – Show Envoy statistics in Prometheus format.
  - `GET /stats/prometheus?usedonly` – Only show statistics that Envoy has updated.

For more information about Agent configuration variables, see Envoy configuration variables.

The new AWS App Mesh Agent is included in App Mesh-optimized Envoy images starting from version `1.21.0.0` and requires no additional resource allocation in customer tasks or pods.

App Mesh observability

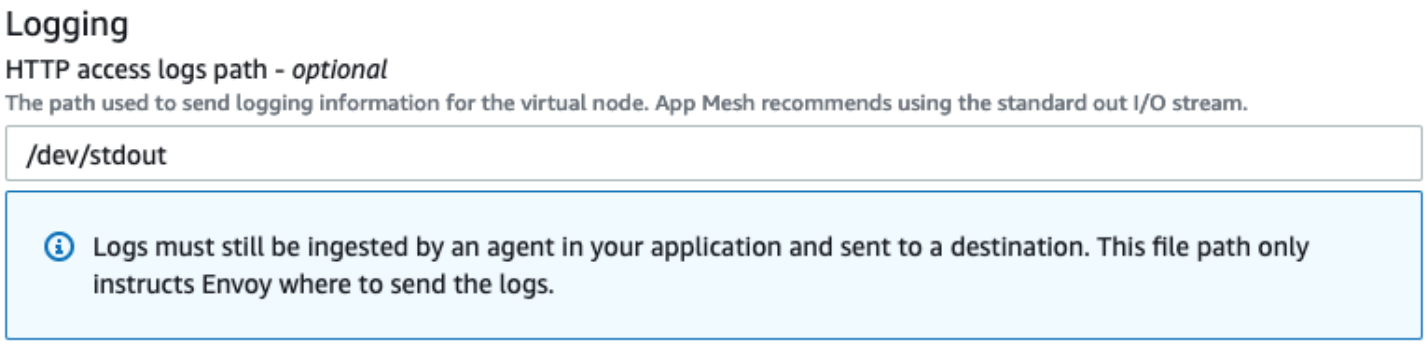
One of the benefits from working with App Mesh is greater visibility into your microservice applications. App Mesh is able to work with many different logging, metric, and tracing solutions.

The Envoy proxy and App Mesh offer the following types of tools to help you gain a clearer view of your applications and proxies:

- Logging
- Metrics
- Tracing

Logging

When you create your virtual nodes and virtual gateways, you have the option to configure Envoy access logs. In the console, this is in the **Logging** section of the virtual node and virtual gateway create or edit workflows.



The preceding image shows a logging path of `/dev/stdout` for Envoy access logs.

For `format`, specify **one** of two possible formats, `json` *or* `text`, and the pattern. `json` takes key pairs and transforms them into JSON struct before passing them to Envoy.

The following code block shows the JSON representation that you can use in the AWS CLI.

```
    "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**
>
> Make sure to check that your input pattern is valid for Envoy, or Envoy will reject the update and store the latest changes in the `error state`.

When you send Envoy access logs to `/dev/stdout`, they are mixed in with the Envoy container logs. You can export them to a log storage and processing service like CloudWatch Logs using standard Docker log drivers such as `awslogs`. For more information, see Using the awslogs Log Driver in the Amazon ECS Developer Guide. To export only the Envoy access logs (and ignore the other Envoy container logs), you can set the `ENVOY_LOG_LEVEL` to `off`. You can log request without query string by including the format string `%REQ_WITHOUT_QUERY(X?Y):Z%`. For examples, see `ReqWithoutQuery` Formatter. For more information, see Access logging in the Envoy documentation.

Enable access logs on Kubernetes

When using the App Mesh Controller for Kubernetes, you can configure virtual nodes with access logging by adding the logging configuration to the virtual node spec, as shown in the following example.

```
---
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"
```

Your cluster must have a log forwarder to collect these logs, such as Fluentd. For more information see, Set up Fluentd as a DaemonSet to send logs to CloudWatch Logs.

Envoy also writes various debugging logs from its filters to `stdout`. These logs are useful for gaining insights into both Envoy's communication with App Mesh and service-to-service traffic. Your specific logging level can be configured using the `ENVOY_LOG_LEVEL` environment variable. For example, the following text is from an example debug log showing the cluster that Envoy matched for a particular HTTP request.

```
[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 and Cloudwatch

Firelens is a container log router you can use to collect logs for Amazon ECS and AWS Fargate. You can find an example of using Firelens in our AWS Samples repository.

You can use CloudWatch to gather logging information as well as metrics. You can find more information on CloudWatch in our Exporting metrics section of the App Mesh docs.

Monitoring your application using Envoy metrics

Envoy classifies its metrics into the following major categories:

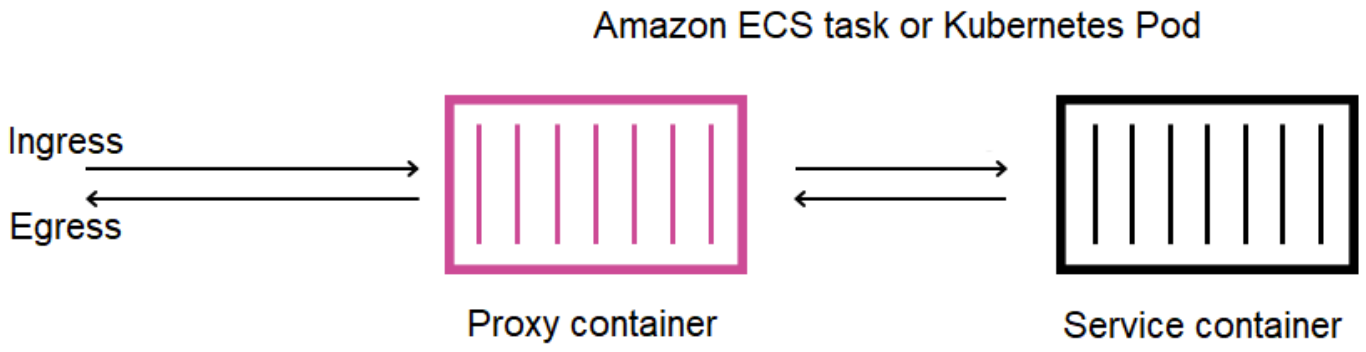
- **Downstream**—Metrics that relate to connections and requests that come into the proxy.

- **Upstream**—Metrics that relate to outgoing connections and requests made by the proxy.
- **Server**—Metrics that describe the internal state of Envoy. These include metrics like uptime or allocated memory.

In App Mesh, the proxy intercepts upstream and downstream traffic. For example, requests received from your clients as well as requests made by your service container are classified as downstream traffic by Envoy. To distinguish between these different types of upstream and downstream traffic, App Mesh further categorizes Envoy metrics depending on the traffic direction relative to your service:

- **Ingress**—Metrics and resources relating to connections and requests that flow to your service container.
- **Egress**—Metrics and resources relating to connections and requests that flow from your service container and ultimately out of your Amazon ECS task or Kubernetes pod.

The following picture shows the communication between the proxy and service containers.



Resource naming conventions

It's useful to understand how Envoy views your mesh and how its resources map back to the resources you define in App Mesh. These are the primary Envoy resources that App Mesh configures:

- **Listeners**—The addresses and ports the proxy listens for downstream connections on. In the previous picture, App Mesh creates an ingress listener for traffic coming into your Amazon ECS task or Kubernetes pod and an egress listener for traffic leaving your service container.

- **Clusters**—A named group of upstream endpoints that the proxy connects and routes traffic to. In App Mesh, your service container is represented as a cluster, as well as all other virtual nodes your service can connect to.
- **Routes**—These correspond to routes you define in your mesh. They contain the conditions by which the proxy matches a request as well as the target cluster a request is sent to.
- **Endpoints and cluster load assignments**—The IP addresses of upstream clusters. When using AWS Cloud Map as your service discovery mechanism for virtual nodes, App Mesh sends discovered service instances as endpoint resources to your proxy.
- **Secrets**—These include, but are not limited to, your encryption keys and TLS certificates. When using AWS Certificate Manager as a source for client and server certificates, App Mesh sends public and private certificates to your proxy as secret resources.

App Mesh uses a consistent scheme for naming Envoy resources that you can use to relate back to your mesh.

Understanding the naming scheme for listeners and clusters is important in understanding Envoy's metrics in App Mesh.

Listener names

Listeners are named using the following format:

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

You will typically see the following listeners configured in Envoy:

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

Using either a Kubernetes CNI plugin or IP tables rules, traffic in your Amazon ECS task or Kubernetes pod is directed to the ports `15000` and `15001`. App Mesh configures Envoy with these two listeners to accept ingress (incoming) and egress (outgoing) traffic. If you do not have a listener configured on your virtual node, you shouldn't see an ingress listener.

Cluster names

Most clusters use the following format:

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

Virtual nodes that your services communicate with each have their own cluster. As mentioned previously, App Mesh creates a cluster for the service running next to Envoy so the proxy can send ingress traffic to it.

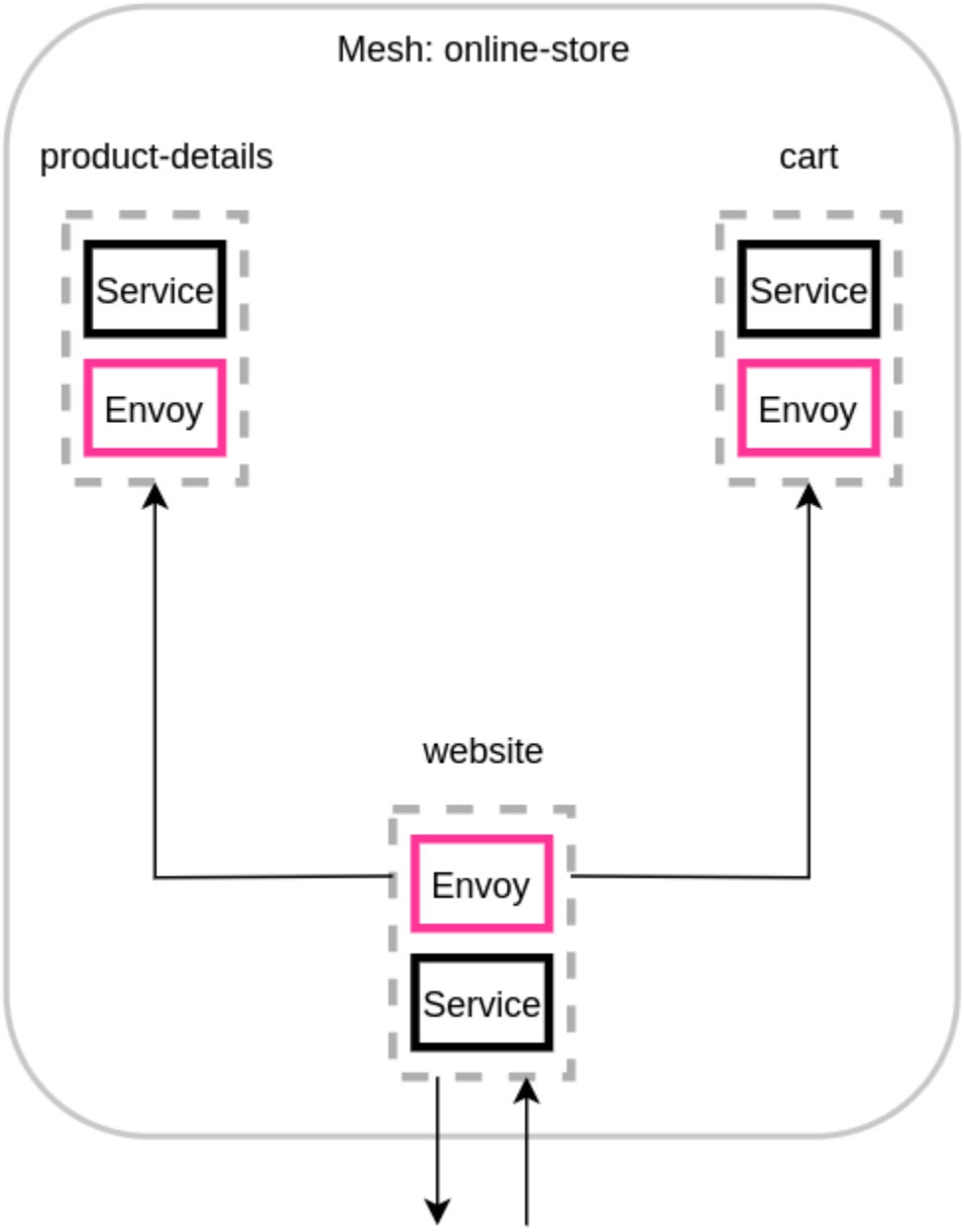
For example, if you have a virtual node named `my-virtual-node` that listens for http traffic on port `8080` and that virtual node is in a mesh named `my-mesh`, App Mesh creates a cluster named `cds_ingress_my-mesh_my-virtual-node_http_8080`. This cluster serves as the destination for traffic into `my-virtual-node`'s service container.

App Mesh may also create the following types of additional special clusters. These other clusters do not necessarily correspond to resources that you explicitly define in your mesh.

- Clusters used to reach other AWS services. This type allows your mesh to reach most AWS services by default: `cds_egress_<mesh name>_amazonaws`.
- Cluster used to perform routing for virtual gateways. This can generally be safely ignored: .
  - For single listeners: `cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<protocol>_<port>`
  - For multiple listeners: `cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<ingress_listener_port>_<protocol>_<port>`
- The cluster who's endpoint you can define, such as TLS, when you retrieve secrets using Envoy's Secret Discovery Service: `static_cluster_sds_unix_socket`.

Example application metrics

To illustrate the metrics available in Envoy, the following sample application has three virtual nodes. The virtual services, virtual routers, and routes in the mesh can be ignored since they are not reflected in Envoy's metrics. In this example, all services listen for http traffic on port 8080.



We recommend adding the environment variable `ENABLE_ENVOY_STATS_TAGS=1` to the Envoy proxy containers running in your mesh. This adds the following metric dimensions to all metrics emitted by the proxy:

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

These tags are set to the name of mesh, virtual node, or virtual gateway to allow filtering metrics using the names of resources in your mesh.

Resource names

The website virtual node's proxy has the following resources:

- Two listeners for ingress and egress traffic:
  - `lds_ingress_0.0.0.0_15000`
  - `lds_egress_0.0.0.0_15001`
- Two egress clusters, representing the two virtual node back ends:
  - `cds_egress_online-store_product-details_http_8080`
  - `cds_egress_online-store_cart_http_8080`
- An ingress cluster for the website service container:
  - `cds_ingress_online-store_website_http_8080`

Example listener metrics

- `listener.0.0.0.0_15000.downstream_cx_active`—Number of active ingress network connections to Envoy.
- `listener.0.0.0.0_15001.downstream_cx_active`—Number of active egress network connections to Envoy. Connections made by your application to external services is included in this count.
- `listener.0.0.0.0_15000.downstream_cx_total`—Total number of ingress network connections to Envoy.
- `listener.0.0.0.0_15001.downstream_cx_total`—Total number of egress network connections to Envoy.

For the full set of listener metrics, see Statistics in the Envoy documentation.

Example cluster metrics

- `cluster_manager.active_clusters`—The total number of clusters that Envoy has established at least one connection to.
- `cluster_manager.warming_clusters`—The total number of clusters that Envoy has yet to connect to.

The following cluster metrics use the format of `cluster.<cluster name>.<metric name>`. These metric names are unique to the application example and are emitted by the website Envoy container:

- `cluster.cds_egress_online-store_product-details_http_8080.upstream_cx_total`—Total number of connections between website and product-details.
- `cluster.cds_egress_online-store_product-details_http_8080.upstream_cx_connect_fail`—Total number of failed connections between website and product-details.
- `cluster.cds_egress_online-store_product-details_http_8080.health_check.failure`—Total number of failed health checks between website and product-details.
- `cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_total`—Total number of requests made between website and product-details.
- `cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_time`—Time taken by requests made between website and product-details.
- `cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_2xx`—Number of HTTP 2xx responses received by website from product-details.

For the full set of HTTP metrics, see Statistics in the Envoy documentation.

Management server metrics

Envoy also emits metrics related to its connection to the App Mesh control plane, which acts as Envoy's management server. We recommend monitoring some of these metrics as a way to notify you when your proxies become desynchronized from the control plane for extended periods of time. Loss of connectivity to the control plane or failed updates prevent your proxies from receiving new configuration from App Mesh, including mesh changes made via App Mesh APIs.

- `control_plane.connected_state`—This metric is set to 1 when the proxy is connected to App Mesh, otherwise it is 0.

- `*.update_rejected`—Total number of configuration updates that are rejected by Envoy. These are usually due to user misconfiguration. For example, if you configure App Mesh to read a TLS certificate from a file that cannot be read by Envoy, the update containing the path to that certificate is rejected.
  - For Listener updated rejected, the stats will be `listener_manager.lds.update_rejected`.
  - For Cluster updated rejected, the stats will be `cluster_manager.cds.update_rejected`.
- `*.update_success`—Number of successful configuration updates made by App Mesh to your proxy. These include the initial configuration payload sent when a new Envoy container is started.
  - For Listener updated success, the stats will be `listener_manager.lds.update_success`.
  - For Cluster updated success, the stats will be `cluster_manager.cds.update_success`.

For the set of management server metrics, see Management Server in the Envoy documentation.

Exporting metrics

Envoy emits many statistics on both its own operation and various dimensions on inbound and outbound traffic. To learn more about Envoy statistics, see Statistics in the Envoy documentation. These metrics are available through the `/stats` endpoint on the proxy's administration port, which is typically `9901`.

The `stat` prefix will be different depending on if you're using single or multiple listeners. Below are some examples to illustrate the differences.

> **Warning**
>
> If you update your single listener to the multiple listener feature, you can face a breaking change due to the updated stat prefix illustrated in the following table.
> We suggest you use Envoy image `1.22.2.1-prod` or later. This allows you to see similar metric names in your Prometheus endpoint.

Single Listener (SL)/Existing stats with "ingress" listener prefix	Multiple Listeners (ML)/New stats with "ingress.<protocol>.<port>" listener prefix	
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	

For more information about the stats endpoint, see Statistics endpoint in the Envoy documentation. For more information about the administration interface, see Enable the Envoy proxy administration interface.

Prometheus for App Mesh with Amazon EKS

Prometheus is an open-source monitoring and alerting toolkit. One of its capabilities is to specify a format for emitting metrics that can be consumed by other systems. For more information about Prometheus, see Overview in the Prometheus documentation. Envoy can emit its metrics via its stats endpoint by passing in the parameter `/stats?format=prometheus`.

For customers that are using Envoy image build v1.22.2.1-prod, there are two additional dimensions to indicate ingress listener specific stats:

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

Below is a comparison between Prometheus existing stats vs new stats.

- Existing stats with "ingress" listener prefix

```
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
```

- New stats with "ingress.<protocol>.<port>" + Appmesh Envoy Image v1.22.2.1-prod or later

```
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
```

- New stats with "ingress.<protocol>.<port>" + custom Envoy Imagebuild

```
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
```

For multiple listeners, the `cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<ingress_listener_port>_<protocol>_<port>` special cluster will be listener specific.

- Existing stats with "ingress" listener prefix

```
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
```

- New stats with "ingress.<protocol>.<port>"

```
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
```

Installing Prometheus

1. Add the EKS repository to Helm:

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

2. Install App Mesh Prometheus

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

Prometheus Example

The following is an example of creating a `PersistentVolumeClaim` for Prometheus persistent storage.

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

Walkthrough for using Prometheus

- App Mesh with EKS—Observability: Prometheus

To learn more about Prometheus and Prometheus with Amazon EKS

- Prometheus Documentation
- **EKS -** Control plane metrics with Prometheus

CloudWatch for App Mesh

Emitting Envoy stats to CloudWatch from Amazon EKS

You can install the CloudWatch Agent to your cluster and configure it to collect a subset of metrics from your proxies. If you do not already have an Amazon EKS cluster, then you can create one with the steps in Walkthrough: App Mesh with Amazon EKS on GitHub. You can install a sample application onto the cluster by following the same walkthrough.

To set the appropriate IAM permissions for your cluster and install the agent, follow the steps in Install the CloudWatch Agent with Prometheus Metrics Collection. The default installation contains a Prometheus scrape configuration which pulls a useful subset of Envoy stats. For more information, see Prometheus Metrics for App Mesh.

To create an App Mesh custom CloudWatch dashboard configured to display the metrics that the agent is collecting, follow the steps in the Viewing Your Prometheus Metrics tutorial. Your graphs will begin to populate with the corresponding metrics as traffic enters the App Mesh application.

Filtering metrics for CloudWatch

The App Mesh metrics extension provides a subset of useful metrics that give you insights into the behaviors of the resources you define in your mesh. Since the CloudWatch agent supports scraping Prometheus metrics, you can provide a scrape configuration to select the metrics you want to pull from Envoy and send to CloudWatch.

You can find an example of scraping metrics using Prometheus in our Metrics Extension walkthrough.

CloudWatch Example

You can find a sample configuration of CloudWatch in our AWS Samples repository.

Walkthroughs for using CloudWatch

- Add monitoring and logging capabilities in our App Mesh workshop.
- App Mesh with EKS—Observability: CloudWatch
- Using App Mesh's metrics extension on ECS

Metrics extension for App Mesh

Envoy generates hundreds of metrics broken down into a few different dimensions. The metrics aren't straightforward in the way they relate back to App Mesh. In the case of virtual services, there is no mechanism to know for sure which virtual service is communicating to a given virtual node or virtual gateway.

The App Mesh metrics extension enhances Envoy proxies running in your mesh. This enhancement allows the proxies to emit additional metrics that are aware of the resources you define. This small subset of additional metrics will help give you greater insight into the behavior of those resources you defined in App Mesh.

To enable the App Mesh metrics extension, set the environment variable `APPMESH_METRIC_EXTENSION_VERSION` to `1`.

```
APPMESH_METRIC_EXTENSION_VERSION=1
```

For more information about Envoy configuration variables, see Envoy configuration variables.

Metrics Related to Inbound Traffic

- **`ActiveConnectionCount`**
  - `envoy.appmesh.ActiveConnectionCount` — Number of active TCP connections.
  - *Dimensions* — Mesh, VirtualNode, VirtualGateway
- **`NewConnectionCount`**
  - `envoy.appmesh.NewConnectionCount` — Total number of TCP connections.
  - *Dimensions* — Mesh, VirtualNode, VirtualGateway

- **`ProcessedBytes`**
  - `envoy.appmesh.ProcessedBytes` — Total TCP bytes sent to and received from downstream clients.
  - *Dimensions* — Mesh, VirtualNode, VirtualGateway
- **`RequestCount`**
  - `envoy.appmesh.RequestCount` — The number of processed HTTP requests.
  - *Dimensions* — Mesh, VirtualNode, VirtualGateway
- **`GrpcRequestCount`**
  - `envoy.appmesh.GrpcRequestCount` — The number of processed gPRC requests.
  - *Dimensions* — Mesh, VirtualNode, VirtualGateway

Metrics Related to Outbound Traffic

You will see different dimensions on your outbound metrics based on if they come from a virtual node or a virtual gateway.

- **`TargetProcessedBytes`**
  - `envoy.appmesh.TargetProcessedBytes` — Total TCP bytes sent to and received from targets upstream of Envoy.
  - *Dimensions*:
    - Virtual node dimensions — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
    - Virtual gateway dimensions — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode
- **`HTTPCode_Target_2XX_Count`**
  - `envoy.appmesh.HTTPCode_Target_2XX_Count` — The number of HTTP requests to a target upstream of Envoy that resulted in a 2xx HTTP response.
  - *Dimensions*:
    - Virtual node dimensions — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
    - Virtual gateway dimensions — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode
- **`HTTPCode_Target_3XX_Count`**
  - `envoy.appmesh.HTTPCode_Target_3XX_Count` — The number of HTTP requests to a target upstream of Envoy that resulted in a 3xx HTTP response.

- *Dimensions*:
  - Virtual node dimensions — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
  - Virtual gateway dimensions — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode
- **`HTTPCode_Target_4XX_Count`**
  - `envoy.appmesh.HTTPCode_Target_4XX_Count` — The number of HTTP requests to a target upstream of Envoy that resulted in a 4xx HTTP response.
  - *Dimensions*:
    - Virtual node dimensions — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
    - Virtual gateway dimensions — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode
- **`HTTPCode_Target_5XX_Count`**
  - `envoy.appmesh.HTTPCode_Target_5XX_Count` — The number of HTTP requests to a target upstream of Envoy that resulted in a 5xx HTTP response.
  - *Dimensions*:
    - Virtual node dimensions — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
    - Virtual gateway dimensions — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode
- **`RequestCountPerTarget`**
  - `envoy.appmesh.RequestCountPerTarget` — The number of requests sent to a target upstream of Envoy.
  - *Dimensions*:
    - Virtual node dimensions — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
    - Virtual gateway dimensions — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode
- **`TargetResponseTime`**
  - `envoy.appmesh.TargetResponseTime` — The time elapsed from when a request is made to a target upstream of Envoy to when the full response is received.
  - *Dimensions*:
    - Virtual node dimensions — Mesh, VirtualNode, TargetVirtualService, TargetVirtualNode
    - Virtual gateway dimensions — Mesh, VirtualGateway, TargetVirtualService, TargetVirtualNode

Datadog for App Mesh

Datadog is a monitoring and security applicaton for end to end monitoring, metrics, and logging of cloud applications. Datadog makes your infrastructure, applications, and third-party applications completely observable.

Installing Datadog

- EKS - To setup Datadog with EKS, follow these steps from the Datadog docs.
- ECS EC2 - To set up Datadog with ECS EC2, follow these steps from the Datadog docs.

To learn more about Datadog

- Datadog Documentation

Tracing

> **Important**
>
> To fully implement tracing, you'll need to update your application.
> To see all the available data from your chosen service, you'll have to instrument your application using the applicable libraries.

Monitor App Mesh with AWS X-Ray

AWS X-Ray is a service that provides tools that let you view, filter, and gain insights into data collected from the requests your application serves. These insights help you identify issues and opportunities to optimize your app. You can see detailed information about requests and responses, and downstream calls your application makes to other AWS services.

X-Ray integrates with App Mesh to manage your Envoy microservices. Trace data from Envoy is sent to the X-Ray daemon running in your container.

Implement X-Ray in your application code using the SDK guide specific to your language.

Enable X-Ray tracing through App Mesh

- **Depending on the type of service:**
  - **ECS -** In the Envoy proxy container definition, set the `ENABLE_ENVOY_XRAY_TRACING` environment variable to `1` and the `XRAY_DAEMON_PORT` environment variable to `2000`.
  - **EKS -** In the App Mesh Controller configuration, include `--set tracing.enabled=true` and `--set tracing.provider=x-ray`.
- In your X-Ray container, expose port `2000` and run as user `1337`.

X-Ray examples

An Envoy container definition for Amazon ECS

```
      {
        "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
      }
```

Updating the App Mesh controller for Amazon EKS

```
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
```

Walkthroughs for using the X-Ray

- Monitor with AWS X-Ray
- App Mesh with Amazon EKS - Observability: X-Ray
- Distributed tracing with X-Ray in the AWS App Mesh Workshop

To learn more about AWS X-Ray

- AWS X-Ray documentation

Troubleshooting AWS X-Ray with App Mesh

- Unable to see AWS X-Ray traces for my applications.

Jaeger for App Mesh with Amazon EKS

Jaeger is an open source, end to end distributed tracing system. It can be used to profile networks and for monitoring. Jaeger can also help you troubleshoot complex cloud native applications.

To implement Jaeger into your application code, you can find the guide specific to your language in the Jaeger documentation tracing libraries.

Installing Jaeger using Helm

1. Add the EKS repository to Helm:

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

2. Install App Mesh Jaeger

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

Jaeger Example

The following is an example of creating a `PersistentVolumeClaim` for Jaeger persistent storage.

```
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
```

Walkthrough for using the Jaeger

- App Mesh with EKS—Observability: Jaeger

To learn more about Jaeger

- Jaeger Documentation

Datadog for tracing

Datadog can be used for tracing as well as metrics. For more information and installation instructions, find the guide specific to your application language in the Datadog documentation.

App Mesh tooling

App Mesh gives customers the ability to interact with its APIs indirectly using tools such as:

- AWS CloudFormation
- AWS Cloud Development Kit (AWS CDK)
- App Mesh Controller for Kubernetes
- Terraform

App Mesh and AWS CloudFormation

AWS CloudFormation is a service that lets you create a template with all the resources you need for your application, and then AWS CloudFormation will configure and provision the resouces for you. It will also configure all the dependencies, so you can focus more on you application and less on managing resources.

For more information and examples on using AWS CloudFormation with App Mesh, see the AWS CloudFormation documentation.

App Mesh and AWS CDK

AWS CDK is a development framework for using code to define your cloud infrastructure and using AWS CloudFormation to provision it. AWS CDK supports multiple programming languages including TypeScript, JavaScript, Python, Java, and C#/.Net.

For more information on using AWS CDK with App Mesh, see the AWS CDK documentation.

App Mesh controller for Kubernetes

The App Mesh controller for Kubernetes helps you to manage your App Mesh resources for a Kubernetes cluster and inject sidecars into pods. This controller is specifically for use with Amazon EKS and allows you to manage your resources in a manner that is native to Kubernetes.

For more information on the App Mesh controller, see the App Mesh Controller documentation.

To see a guide on implementing App Mesh with Amazon EKS using the App Mesh Controller for Kubernetes, check out the Amazon EKS Workshop.

App Mesh and Terraform

Terraform is an open-source infrastructure as code software tool. Terraform can manage cloud services using thier CLI and interacts with APIs using declaritive configuration files.

To see more about using App Mesh with Terraform, check out the Terraform documentation.

Working with shared meshes

You can share your App Mesh meshes across AWS accounts using the AWS Resource Access Manager service. A shared mesh allows resources created by different AWS accounts to communicate with each other in the same mesh.

An AWS account can be a mesh resource owner, a mesh consumer, or both. Consumers can create resources in a mesh that is shared with their account. Owners can create resources in any mesh the account owns. A mesh owner can share a mesh with the following types of mesh consumers.

- Specific AWS accounts inside or outside of its organization in AWS Organizations
- An organizational unit inside its organization in AWS Organizations
- Its entire organization in AWS Organizations

For an end-to-end walk through of sharing a mesh, see Cross-account mesh walk through on GitHub.

Granting permissions to share meshes

When sharing meshes across accounts, there are required permissions for the IAM principal sharing the mesh and required resource-level permissions for the mesh itself.

Granting permission to share a mesh

A minimum set of permissions is required for an IAM principal to share a mesh. We recommend using the `AWSAppMeshFullAccess` and `AWSResourceAccessManagerFullAccess` managed IAM policies to ensure your IAM principals have the required permissions to share and use shared meshes.

If you use a custom IAM policy, the `appmesh:PutMeshPolicy`, `appmesh:GetMeshPolicy`, and `appmesh:DeleteMeshPolicy` actions are required. These are permission-only IAM actions. If an IAM principal doesn't have these permissions granted, an error will occur when attempting to share the mesh using the AWS RAM service.

For more information about the way the AWS Resource Access Manager service uses IAM, see How AWS RAM uses IAM in the *AWS Resource Access Manager User Guide*.

Granting permissions for a mesh

A shared mesh has the following permissions.

- Consumers can list and describe all resources in a mesh that is shared with the account.
- Owners can list and describe all resources in any mesh the account owns.
- Owners and consumers can modify resources in a mesh that the account created, but they cannot modify resources that other another account created.
- Consumers can delete any resource in a mesh that the account created.
- Owners can delete any resource in a mesh that any account created.
- Owner's resources can only reference other resources in the same account. For example, a virtual node can only reference AWS Cloud Map or an AWS Certificate Manager certificate that is in the same account as the virtual node's owner.
- Owners and consumers can connect an Envoy proxy to App Mesh as a virtual node that the account owns.
- Owners can create virtual gateways and virtual gateway routes.
- Owners and consumers can list tags and can tag/untag resources in a mesh that the account created. They can't list tags and tag/untag resources in a mesh that aren't created by the account.

Shared meshes use a policy-based authorization. A mesh is shared with with a fixed set of permissions. These permissions are selected to be added to a resource policy, and an optional IAM policy can also be selected based on IAM user/role. The intersection of permissions allowed in these policies, less any explicit permissions denied, determines a principal's access to the mesh.

When sharing a mesh, the AWS Resource Access Manager service creates a managed policy named `AWSRAMDefaultPermissionAppMesh` and associates it with your App Mesh that provides the following permissions.

- `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`

Prerequisites for sharing meshes

To share a mesh, you must meet the following prerequisites.

- You must own the mesh in your AWS account. You cannot share a mesh that has been shared with you.
- To share a mesh with your organization or an organizational unit in AWS Organizations, you must enable sharing with AWS Organizations. For more information, see Enable Sharing with AWS Organizations in the *AWS RAM User Guide*.
- Your services must be deployed in an Amazon VPC that has shared connectivity across the accounts that include the mesh resources that you want to communicate with each other. One way to share network connectivity is to deploy all of the services that you want to use in your mesh to a shared subnet. For more information and limitations, see Sharing a Subnet.

- Services must be discoverable through DNS or AWS Cloud Map. For more information about service discovery, see Virtual nodes.

Related services

Mesh sharing integrates with AWS Resource Access Manager (AWS RAM). AWS RAM is a service that enables you to share your AWS resources with any AWS account or through AWS Organizations. With AWS RAM, you share resources that you own by creating a *resource share*. A resource share specifies the resources to share, and the consumers with whom to share them. Consumers can be individual AWS accounts, or organizational units or an entire organization in AWS Organizations.

For more information about AWS RAM, see the *AWS RAM User Guide*.

Sharing a mesh

Sharing a mesh enables mesh resources created by different accounts to communicate with each other in the same mesh. You can only share a mesh that you own. To share a mesh, you must add it to a resource share. A resource share is an AWS RAM resource that lets you share your resources across AWS accounts. A resource share specifies the resources to share and the consumers with whom they are shared. When you share a mesh using the Amazon Linux console, you add it to an existing resource share. To add the mesh to a new resource share, create the resource share using the AWS RAM console.

If you're part of an organization in AWS Organizations and sharing within your organization is enabled, consumers in your organization can be automatically granted access to the shared mesh. Otherwise, consumers receive an invitation to join the resource share and are granted access to the shared mesh after accepting the invitation.

You can share a mesh that you own using the AWS RAM console or the AWS CLI.

To share a mesh that you own using the AWS RAM console

For instructions, see Creating a Resource Share in the *AWS RAM User Guide*. When you select a resource type, select **Meshes**, and then select the mesh that you want to share. If no meshes are listed, create a mesh first. For more information, see Creating a service mesh.

To share a mesh that you own using the AWS CLI

Use the create-resource-share command. For the `--resource-arns` option, specify the ARN of the mesh that you want to share.

Unsharing a shared mesh

When you unshare a mesh, App Mesh disables further access to the mesh by former consumers of the mesh. However, App Mesh doesn't delete the resources created by the consumers. After the mesh is unshared, only the mesh owner can access and delete the resources. App Mesh prevents the account that owned resources in the mesh from receiving configuration information after the mesh is unshared. App Mesh also prevents any other accounts with resources in the mesh from receiving configuration information from an unshared mesh. Only the owner of the mesh can unshare it.

To unshare a shared mesh that you own, you must remove it from the resource share. You can do this using the AWS RAM console or the AWS CLI.

To unshare a shared mesh that you own using the AWS RAM console

For instructions, see Updating a Resource Share in the *AWS RAM User Guide*.

To unshare a shared mesh that you own using the AWS CLI

Use the disassociate-resource-share command.

Identifying a shared mesh

Owners and consumers can identify shared meshes and mesh resources using the Amazon Linux console and AWS CLI

To identify a shared mesh using the Amazon Linux console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. From the left navigation, select **Meshes**. The account ID of the mesh owner for each mesh is listed in the **Mesh owner** column.
3. From the left navigation, select **Virtual services**, **Virtual routers**, or **Virtual nodes**. You see the account ID for the **Mesh owner** and **Resource owner** for each of the resources.

To identify a shared mesh using the AWS CLI

Use the `aws appmesh list resource` command, such as `aws appmesh list-meshes`. The command returns the meshes that you own and the meshes that are shared with you. The `meshOwner` property shows the AWS account ID of the `meshOwner` and the `resourceOwner`

property shows the AWS account ID of the resource owner. Any command run against any mesh resource returns these properties.

The user defined tags that you attach to a shared mesh are available only to your AWS account. They're not available to the other accounts that the mesh is shared with. The `aws appmesh list-tags-for-resource` command for a mesh in another account is denied access.

Billing and metering

There are no charges for sharing a mesh.

Instance quotas

All quotas for a mesh also apply to shared meshes, regardless of who created resources in the mesh. Only a mesh owner can request quota increases. For more information, see App Mesh service quotas. The AWS Resource Access Manager service also has quotas. For more information, see Service Quotas.

AWS services integrated with App Mesh

App Mesh works with other AWS services to provide additional solutions for your business challenges. This topic identifies services that either use App Mesh to add functionality, or services that App Mesh uses to perform tasks.

Contents

Creating App Mesh resources with AWS CloudFormation

App Mesh is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want, for example an App Mesh mesh, and AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your App Mesh resources consistently and repeatedly. Just describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

App Mesh and AWS CloudFormation templates

To provision and configure resources for App Mesh and related services, you must understand AWS CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see What is AWS CloudFormation Designer? in the *AWS CloudFormation User Guide*.

App Mesh supports creating meshes, routes, virtual nodes, virtual routers, and virtual services in AWS CloudFormation. For more information, including examples of JSON and YAML templates for your App Mesh resources, see App Mesh resource type reference in the *AWS CloudFormation User Guide*.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- AWS CloudFormation Command Line Interface User Guide

App Mesh on AWS Outposts

AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud. App Mesh on AWS Outposts is ideal for low-latency workloads that need to be run in close proximity to on-premises data and applications. For more information about AWS Outposts, see the AWS Outposts User Guide.

Prerequisites

The following are the prerequisites for using App Mesh on AWS Outposts:

- You must have installed and configured an Outpost in your on-premises data center.
- You must have a reliable network connection between your Outpost and its AWS Region.
- The AWS Region for the Outpost must support AWS App Mesh. For a list of supported Regions, see AWS App Mesh Endpoints and Quotas in the *AWS General Reference*.

Limitations

The following are the limitations of using App Mesh on AWS Outposts:

- AWS Identity and Access Management, Application Load Balancer, Network Load Balancer, Classic Load Balancer, and Amazon Route 53 run in the AWS Region, not on Outposts. This will increase latencies between these services and the containers.

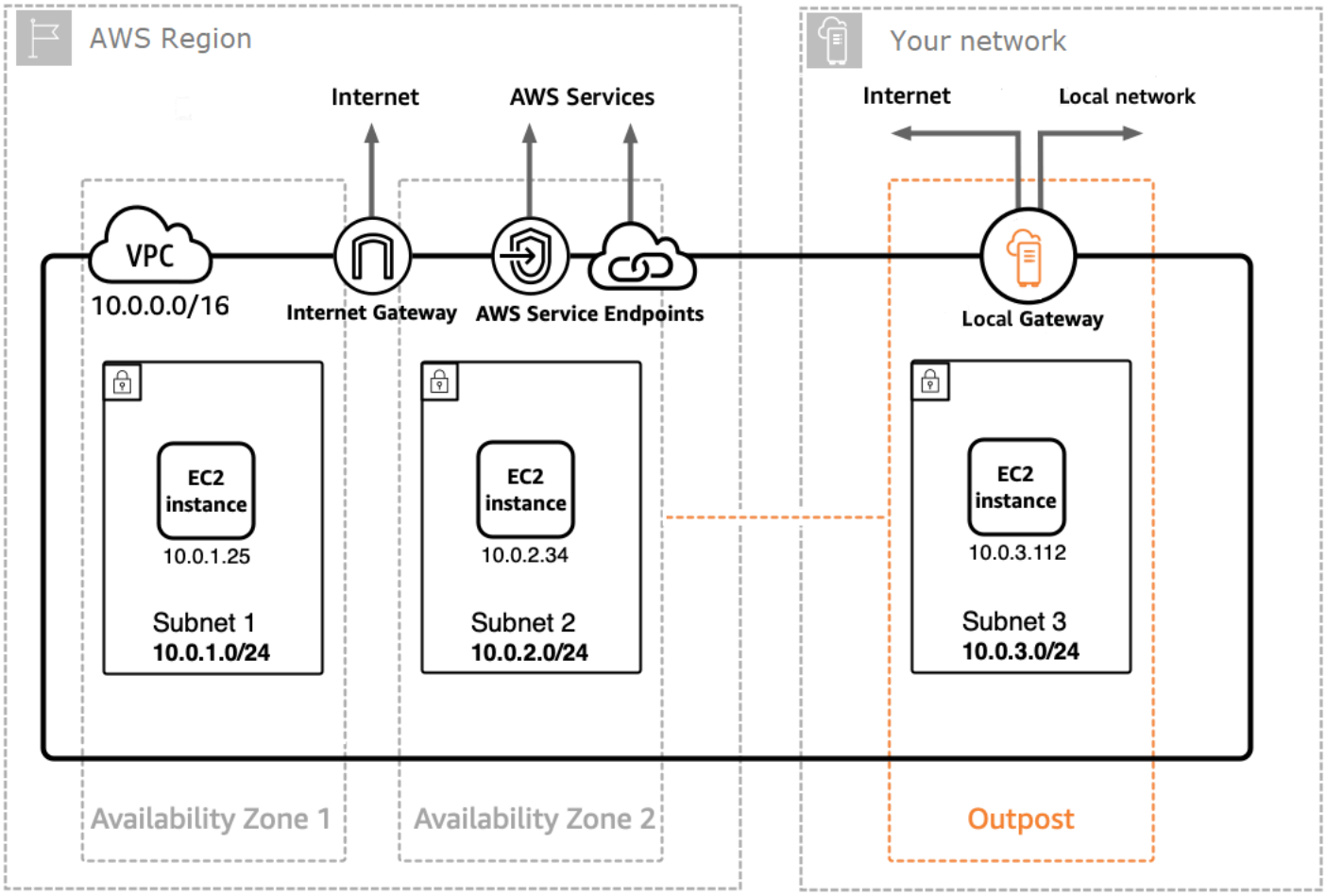
Network connectivity considerations

The following are network connectivity considerations for Amazon EKS AWS Outposts:

- If network connectivity between your Outpost and its AWS Region is lost, the App Mesh Envoy proxies will continue to run. However you will not be able to modify your service mesh until connectivity is restored.
- We recommend that you provide reliable, highly available, and low-latency connectivity between your Outpost and its AWS Region.

Creating an App Mesh Envoy proxy on an Outpost

An Outpost is an extension of an AWS Region, and you can extend an Amazon VPC in an account to span multiple Availability Zones and any associated Outpost locations. When you configure your Outpost, you associate a subnet with it to extend your Regional VPC environment to your on-premises facility. Instances on an Outpost appear as part of your Regional VPC, similar to an Availability Zone with associated subnets.



To create an App Mesh Envoy proxy on an Outpost, add the App Mesh Envoy container image to the Amazon ECS task or Amazon EKS pod running on an Outpost. For more information, see Amazon Elastic Container Service on AWS Outposts in the *Amazon Elastic Container Service Developer Guide* and Amazon Elastic Kubernetes Service on AWS Outposts in the Amazon EKS User Guide.

App Mesh best practices

To achieve the goal of zero failed requests during planned deployments and during the unplanned loss of some hosts, the best practices in this topic implement the following strategy:

- Increase the likelihood that a request will succeed from the perspective of the application by using a safe default retry strategy. For more information, see Instrument all routes with retries.
- Increase the likelihood that a retried request succeeds by maximizing the likelihood that the retried request is sent to an actual destination. For more information, see Adjust deployment velocity, Scale out before scale in, and Implement container health checks.

To significantly reduce or eliminate failures, we recommend that you implement the recommendations in all of the following practices.

Instrument all routes with retries

Configure all virtual services to use a virtual router and set a default retry policy for all routes. This will mitigate failed requests by reselecting a host and sending a new request. For retry policies, we recommend a value of at least two for `maxRetries`, and specifying the following options for each type of retry event in each route type that supports the retry event type:

- **TCP** – `connection-error`
- **HTTP and HTTP/2** – `stream-error` and `gateway-error`
- **gRPC** – `cancelled` and `unavailable`

Other retry events need to be considered on a case-by-case basis as they may not be safe, such as if the request isn't idempotent. You will need to consider and test values for `maxRetries` and `perRetryTimeout` that make the appropriate trade off between the maximum latency of a request (`maxRetries` * `perRetryTimeout`) versus the increased success rate of more retries. Additionally, when Envoy attempts to connect to an endpoint that is no longer present, you should expect that request to consume the full `perRetryTimeout`. To configure a retry policy, see Creating a route and then select the protocol that you want to route.

> **Note**
>
> If you implemented a route on or after July 29, 2020 and didn't specify a retry policy, then App Mesh may have automatically created a default retry policy similar to the previous policy for each route you created on or after July 29, 2020. For more information, see Default route retry policy.

Adjust deployment velocity

When using rolling deployments, reduce the overall deployment velocity. By default, Amazon ECS configures a deployment strategy of a minimum of 100 percent healthy tasks and 200 percent total tasks. On deployment, this results in two points of high drift:

- The 100 percent fleet size of new tasks may be visible to Envoys prior to being ready to complete requests (see Implement container health checks for mitigations).
- The 100 percent fleet size of old tasks may be visible to Envoys while the tasks are being terminated.

When configured with these deployment constraints, container orchestrators may enter a state where they are simultaneously hiding all old destinations and making all new destinations visible. Because your Envoy dataplane is eventually consistent, this can result in periods where the set of destinations visible in your dataplane have diverged from the orchestrator's point of view. To mitigate this, we recommend maintaining a minimum of 100 percent healthy tasks, but lowering total tasks to 125 percent. This will reduce divergence and improve the reliability of retries. We recommend the following settings for different container runtimes:

Amazon ECS

If your service has a desired count of two or three, set `maximumPercent` to 150 percent. Otherwise, set `maximumPercent` to 125 percent.

Kubernetes

Configure your deployment's `update strategy`, setting `maxUnavailable` to 0 percent and `maxSurge` to 25 percent. For more information on deployments, see Kubernetes Deployments documentation.

Scale out before scale in

Scale out and scale in can both result in some probability of failed requests in retries. While there are task recommendations that mitigate scale out, the only recommendation for scale in is to minimize the percentage of scaled in tasks at any one time. We recommend that you use a deployment strategy that scales out new Amazon ECS tasks or Kubernetes deployments prior to scaling in old tasks or deployments. This scaling strategy keeps your percentage of scaled in tasks or deployments lower, while maintaining the same velocity. This practice applies to both Amazon ECS tasks and Kubernetes deployments.

Implement container health checks

In the scale up scenario, containers in an Amazon ECS task may come up out of order and may not be initially responsive. We recommend the following suggestions for different container runtimes:

Amazon ECS

To mitigate this, we recommend using container health checks and container dependency ordering to ensure that Envoy is running and healthy prior to any containers requiring outbound network connectivity starting. To correctly configure an application container and Envoy container in a task definition, see Container dependency.

Kubernetes

None, because Kubernetes liveness and readiness probes are not being considered in registration and de-registration of AWS Cloud Map instances in the App Mesh controller for Kubernetes. For more information, see GitHub issue #132.

Security in AWS App Mesh

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to AWS App Mesh, see AWS Services in Scope by Compliance Program. App Mesh is responsible for securely delivering configuration to local proxies, including secrets such as TLS certificate private keys.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including:
  - The sensitivity of your data, your company's requirements, and applicable laws and regulations.
  - The security configuration of the App Mesh data plane, including the configuration of the security groups that allow traffic to pass between services within your VPC.
  - The configuration of your compute resources associated with App Mesh.
  - The IAM policies associated with your compute resources and what configuration they are allowed to retrieve from the App Mesh control plane.

This documentation helps you understand how to apply the shared responsibility model when using App Mesh. The following topics show you how to configure App Mesh to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your App Mesh resources.

App Mesh security tenet

> Customers should be able to tune the security to the extent they need. Platform should not block them from being more secure. Platform features are secure by default.

Topics

Transport Layer Security (TLS)

In App Mesh, Transport Layer Security (TLS) encrypts communication between the Envoy proxies deployed on compute resources that are represented in App Mesh by mesh endpoints, such as Virtual nodes and Virtual gateways. The proxy negotiates and terminates TLS. When the proxy is deployed with an application, your application code is not responsible for negotiating a TLS session. The proxy negotiates TLS on your application's behalf.

App Mesh allows you to provide the TLS certificate to the proxy in the following ways:

- A private certificate from AWS Certificate Manager (ACM) that is issued by an AWS Private Certificate Authority (AWS Private CA).
- A certificate stored on the local file system of a virtual node that is issued by your own certificate authority (CA)
- A certificate provided by a Secrets Discovery Service (SDS) endpoint over local Unix Domain Socket.

Envoy Proxy authorization must be enabled for the deployed Envoy proxy represented by a mesh endpoint. We recommend that when you enable proxy authorization, you restrict access to only the mesh endpoint that you're enabling encryption for.

Certificate requirements

One of the Subject Alternative Names (SANs) on the certificate must match specific criteria, depending on how the actual service represented by a mesh endpoint is discovered.

- **DNS** – One of the certificate SANs must match the value provided in the DNS service discovery settings. For an application with the service discovery name *`mesh-endpoint.apps.local`*, you can create a certificate matching that name, or a certificate with the wild card `*.`*`apps.local`*.
- **AWS Cloud Map** – One of the certificate SANs must match the value provided in the AWS Cloud Map service discovery settings using the format *`service-name.namespace-name`*. For an application with the AWS Cloud Map service discovery settings of serviceName *`mesh-endpoint`* and the namespaceName *`apps.local`*, you can create a certificate matching the name *`mesh-endpoint.apps.local`*, or a certificate with the wild card `*.`*`apps.local`*.

For both discovery mechanisms, if none of the certificate SANs match the DNS service discovery settings, the connection between Envoys fails with the following error message, as seen from the client Envoy.

```
TLS error: 268435581:SSL routines:OPENSSL_internal:CERTIFICATE_VERIFY_FAILED
```

TLS authentication certificates

App Mesh supports multiple sources for certificates when using TLS authentication.

AWS Private CA

The certificate must be stored in ACM in the same Region and AWS account as the mesh endpoint that will use the certificate. The CA's certificate does not need to be in the same AWS account, but it does still need to be in the same Region as the mesh endpoint. If you don't have an AWS Private CA, then you must create one before you can request a certificate from it. For more information about requesting a certificate from an existing AWS Private CA using ACM, see Request a Private Certificate. The certificate cannot be a public certificate.

The private CAs that you use for TLS client policies must be root user CAs.

To configure a virtual node with certificates and CAs from AWS Private CA, the principal (such as a user or role) that you use to call App Mesh must have the following IAM permissions:

- For any certificates that you add to a listener's TLS configuration, the principal must have the `acm:DescribeCertificate` permission.
- For any CAs configured on a TLS client policy, the principal must have the `acm-pca:DescribeCertificateAuthority` permission.

> **Important**
>
> Sharing CAs with other accounts may give those accounts unintended privileges to the CA. We recommend using resource-based policies to restrict access to just `acm-pca:DescribeCertificateAuthority` and `acm-pca:GetCertificateAuthorityCertificate` for accounts that do not need to issue certificates from the CA.

You can add these permissions to an existing IAM policy that is attached to a principal or create a new principal and policy and attach the policy to the principal. For more information, see Editing IAM Policies, Creating IAM Policies, and Adding IAM Identity Permissions.

> **Note**
>
> You pay a monthly fee for the operation of each AWS Private CA until you delete it. You also pay for the private certificates you issue each month and private certificates that you export. For more information, see AWS Certificate Manager Pricing.

When you enable proxy authorization for the Envoy Proxy that a mesh endpoint represents, the IAM role that you use must be assigned the following IAM permissions:

- For any certificates configured on a virtual node's listener, the role must have the `acm:ExportCertificate` permission.
- For any CAs configured on a TLS client policy, the role must have the `acm-pca:GetCertificateAuthorityCertificate` permission.

File System

You can distribute certificates to Envoy using the file system. You can do this by making the certificate chain and the corresponding private key available on the file path. That way, these resources are reachable from the Envoy sidecar proxy.

Envoy's Secret Discovery Service (SDS)

Envoy fetches secrets like TLS certificates from a specific endpoint through the Secrets Discovery protocol. For more information about this protocol, see Envoy's SDS documentation.

App Mesh configures the Envoy proxy to use a Unix Domain Socket that's local to the proxy to serve as the Secret Discovery Service (SDS) endpoint when SDS serves as the source for your

certificates and certificate chains. You can configure the path to this endpoint by using the `APPMESH_SDS_SOCKET_PATH` environment variable.

> **Important**
>
> Local Secrets Discovery Service using Unix Domain Socket is supported on App Mesh Envoy proxy version 1.15.1.0 and later.
> App Mesh supports V2 SDS protocol using gRPC.

Integrating with SPIFFE Runtime Environment (SPIRE)

You can use any sidecar implementation of the SDS API, including existing toolchains like SPIFFE Runtime Environment (SPIRE). SPIRE is designed to enable the deployment of mutual TLS authentication between multiple workloads in distributed systems. It attests the identity of workloads at runtime. SPIRE also delivers workload-specific, short-lived, and automatically rotating keys and certificates directly to workloads.

You should configure the SPIRE Agent as an SDS provider for Envoy. Allow it to directly supply Envoy with the key material that it needs to provide mutual TLS authentication. Run SPIRE Agents in sidecars next to Envoy proxies. The Agent takes care of re-generating the short-lived keys and certificates as required. The Agent attests Envoy and determines which service identities and CA certificates that it should make available to Envoy when Envoy connects to the SDS server exposed by the SPIRE Agent.

During this process, service identities and CA certificates are rotated, and updates are streamed back to Envoy. Envoy immediately applies them to new connections without any interruptions or downtime and without the private keys ever touching the file system.

How App Mesh configures Envoys to negotiate TLS

App Mesh uses the mesh endpoint configuration of both the client and server when determining how to configure the communication between Envoys in a mesh.

With client policies

When a client policy is enforcing the use of TLS, and one of the ports in the client policy matches the port of the server's policy, the client policy is used to configure the TLS validation

context of the client. For example, if a virtual gateway's client policy matches a virtual node's server policy, TLS negotiation will be attempted between the proxies using the settings defined in the virtual gateway's client policy. If the client policy does not match the port of the server's policy, TLS between the proxies may or may not be negotiated, depending on the server policy's TLS settings.

Without client policies

If the client has not configured a client policy, or the client policy does not match the port of the server, App Mesh will use the server to determine whether or not to negotiate TLS from the client, and how. For example, if a virtual gateway has not specified a client policy, and a virtual node has not configured TLS termination, TLS will not be negotiated between the proxies. If a client has not specified a matching client policy, and a server has been configured with TLS modes `STRICT` or `PERMISSIVE`, the proxies will be configured to negotiate TLS. Depending on how the certificates have been provided for TLS termination, the following additional behavior applies.

- **ACM-managed TLS certificates** – When a server has configured TLS termination using an ACM-managed certificate, App Mesh automatically configures clients to negotiate TLS and validate the certificate against the root user CA that the certificate chains up to.
- **File-based TLS certificates** – When a server has configured TLS termination using a certificate from the proxy's local file system, App Mesh automatically configures a client to negotiate TLS, but the certificate of the server is not validated.

Subject alternative names

You can optionally specify a list of Subject Alternative Names (SANs) to trust. SANs must be in the FQDN or URI format. If SANs are provided, Envoy verifies that the Subject Alternative Name of the presented certificate matches one of the names on this list.

If you don't specify SANs on the terminating mesh endpoint, the Envoy proxy for that node doesn't verify the SAN on a peer client certificate. If you don't specify SANs on the originating mesh endpoint, the SAN on the certificate provided by the terminating endpoint must match the mesh endpoint service discovery configuration.

For more information, see App Mesh TLS: Certificate requirements.

> **Important**
>
> You can only use wildcard SANs if the client policy for TLS is set to `not enforced`. If the client policy for the client virtual node or virtual gateway is configured to enforce TLS, then it can't accept a wildcard SAN.

Verify encryption

Once you've enabled TLS, you can query the Envoy proxy to confirm that communication is encrypted. The Envoy proxy emits statistics on resources that can help you understand if your TLS communication is working properly. For example, the Envoy proxy records statistics on the number of successful TLS handshakes it has negotiated for a specified mesh endpoint. Determine how many successful TLS handshakes there were for a mesh endpoint named *my-mesh-endpoint* with the following command.

```
curl -s 'http://my-mesh-endpoint.apps.local:9901/stats' | grep ssl.handshake
```

In the following example returned output, there were three handshakes for the mesh endpoint, so communication is encrypted.

```
listener.0.0.0.0_15000.ssl.handshake: 3
```

The Envoy proxy also emits statistics when TLS negotiation is failing. Determine whether there were TLS errors for the mesh endpoint.

```
curl -s 'http://my-mesh-endpoint.apps.local:9901/stats' | grep -e "ssl.*\(fail\|error
\)"
```

In the example returned output, there were zero errors for several statistics, so the TLS negotiation succeeded.

```
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
```

For more information about Envoy TLS statistics, see Envoy Listener Statistics.

Certificate renewal

AWS Private CA

When you renew a certificate with ACM, the renewed certificate will be automatically distributed to your connected proxies within 35 minutes of the renewal completion. We recommend using managed renewal to automatically renew certificates nearing the end of their validity period. For more information, see Managed Renewal for ACM's Amazon-Issued Certificates in the AWS Certificate Manager User Guide.

Your own certificate

When using a certificate from the local file system, Envoy will not automatically reload the certificate when it changes. You may either restart or redeploy the Envoy process to load a new certificate. You can also place a newer certificate at a different file path and update the virtual node or gateway configuration with that file path.

Configure Amazon ECS workloads to use TLS authentication with AWS App Mesh

You can configure your mesh to use TLS authentication. Make sure that the certificates are available to Envoy proxy sidecars that you add to your workloads. You can attach an EBS or EFS volume to your Envoy sidecar, or you can store and retrieve certificates from AWS Secrets Manager.

- If you use file-based certificate distribution, attach an EBS or EFS volume to your Envoy sidecar. Make sure that the path to the certificate and private key matches the one that is configured in AWS App Mesh.
- If you're using SDS-based distribution, add a sidecar that implements Envoy's SDS API with access to the certificate.

> **Note**
>
> SPIRE is not supported on Amazon ECS.

Configure Kubernetes workloads to use TLS authentication with AWS App Mesh

You can configure the AWS App Mesh Controller for Kubernetes to enable TLS authentication for virtual node and virtual gateway service backends and listeners. Make sure that the certificates are available to the Envoy proxy sidecars that you add to your workloads. You can see an example for each distribution type in the walkthrough section of Mutual TLS Authentication.

- If you use file-based certificate distribution, attach an EBS or EFS volume to your Envoy sidecar. Make sure that the path to the certificate and private key matches the one configured in the controller. Alternatively, you can use a Kubernetes Secret that is mounted on the file system.
- If you're using SDS-based distribution, you should setup a node local SDS provider that implements Envoy's SDS API. Envoy will reach it over UDS. To enable SDS based mTLS support in the EKS AppMesh controller, set the `enable-sds` flag to `true` and provide the local SDS provider's UDS path to the controller via the `sds-uds-path` flag. If you use helm, you set these as part of your controller installation:

```
--set sds.enabled=true
```

> **Note**
>
> You won't be able to use SPIRE to distribute your certificates if you're using Amazon Elastic Kubernetes Service (Amazon EKS) in Fargate mode.

Mutual TLS authentication

Mutual TLS (Transport Layer Security) authentication is an optional component of TLS that offers two-way peer authentication. Mutual TLS authentication adds a layer of security over TLS and allows your services to verify the client that's making the connection.

The client in the client-server relationship also provides an X.509 certificate during the session negotiation process. The server uses this certificate to identify and authenticate the client. This process helps to verify if the certificate is issued by a trusted certificate authority (CA) and if the certificate is a valid certificate. It also uses the Subject Alternative Name (SAN) on the certificate to identify the client.

You can enable mutual TLS authentication for all the protocols supported by AWS App Mesh. They are TCP, HTTP/1.1, HTTP/2, gRPC.

> **Note**
>
> Using App Mesh, you can configure mutual TLS authentication for communications between Envoy proxies from your services. However, communications between your applications and Envoy proxies are unencrypted.

Mutual TLS authentication certificates

AWS App Mesh supports two possible certificate sources for mutual TLS authentication. Client certificates in a TLS Client Policy and server validation in a listener TLS configuration can be sourced from:

- **File System–** Certificates from the local file system of the Envoy proxy that's being run. To distribute certificates to Envoy, you need to provide file paths for the certificate chain and private key to the App Mesh API.
- **Envoy's Secret Discovery Service (SDS)–** Bring-your-own sidecars that implement SDS and allow certificates to be sent to Envoy. They include the SPIFFE Runtime Environment (SPIRE).

> **Important**
>
> App Mesh doesn't store the certificates or private keys that are used for mutual TLS authentication. Instead, Envoy stores them in memory.

Configure mesh endpoints

Configure mutual TLS authentication for your mesh endpoints, such as virtual nodes or gateways. These endpoints provide certificates and specify trusted authorities.

To do this, you need to provision X.509 certificates for both the client and the server, and explicitly define trusted authority certificates in the validation context of both the TLS termination and TLS origination.

Trust inside of a mesh

Server-side certificates are configured in Virtual Node listeners (TLS termination), and client-side certificates are configured in Virtual Nodes service backends (TLS origination). As an alternative to this configuration, you can define a default client policy for all services backends of a virtual node, and then, if required, you can override this policy for specific backends as needed. Virtual Gateways can only be configured with a default client policy that applies to all of its backends.

You can configure trust across different meshes by enabling mutual TLS authentication for inbound traffic on the Virtual Gateways for both meshes.

Trust outside of a mesh

Specify server-side certificates in the Virtual Gateway listener for TLS termination. Configure the external service that communicates with your Virtual Gateway to present client-side certificates. The certificates should be derived from one of the same certificate authorities (CAs) that the server-side certificates use on the Virtual Gateway listener for TLS origination.

Migrate services to mutual TLS authentication

Follow these guidelines to maintain connectivity when migrating your existing services within App Mesh to mutual TLS authentication.

Migrating services communicating over plaintext

1. Enable `PERMISSIVE` mode for the TLS configuration on the server endpoint. This mode allows plain-text traffic to connect to the endpoint.
2. Configure mutual TLS authentication on your server, specifying the server certificate, trust chain, and optionally the trusted SANs.
3. Confirm communication is happening over a TLS connection.
4. Configure mutual TLS authentication on your clients, specifying the client certificate, trust chain, and optionally the trusted SANs.
5. Enable `STRICT` mode for the TLS configuration on the server.

Migrating services communicating over TLS

1. Configure the mutual TLS settings on your clients, specifying the client certificate and optionally the trusted SANs. The client certificate isn't sent to its backend until after the backend server requests it.
2. Configure the mutual TLS settings on your server, specifying the trust chain and optionally the trusted SANs. For this, your server requests a client certificate.

Verifying mutual TLS authentication

You can refer to the Transport Layer Security: Verify encryption documentation to see how exactly Envoy emits TLS-related statistics. For mutual TLS authentication, you should inspect the following statistics:

- `ssl.handshake`
- `ssl.no_certificate`
- `ssl.fail_verify_no_cert`
- `ssl.fail_verify_san`

The two following examples of statistics together show that successful TLS connections terminating to the virtual node all originated from a client that provided a certificate.

```
listener.0.0.0.0_15000.ssl.handshake: 3
```

```
listener.0.0.0.0_15000.ssl.no_certificate: 0
```

The next example of a statistic shows that the connections from a virtual client node (or gateway) to a backend virtual node failed. The Subject Alternative Name (SAN) that's presented in the server certificate doesn't match any of the SANs trusted by the client.

```
cluster.cds_egress_my-mesh_my-backend-node_http_9080.ssl.fail_verify_san: 5
```

App Mesh mutual TLS authentication walkthroughs

- Mutual TLS authentication walkthrough: This walkthrough describes how you can use the App Mesh CLI to build a color app with mutual TLS authentication.

- Amazon EKS mutual TLS SDS-based walkthrough: This walkthrough shows how you can use mutual TLS SDS-based authentication with Amazon EKS and SPIFFE Runtime Environment (SPIRE).
- Amazon EKS mutual TLS file-based walkthrough: This walkthrough shows how you can use mutual TLS file-based authentication with Amazon EKS and SPIFFE Runtime Environment (SPIRE).

How AWS App Mesh works with IAM

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use App Mesh resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in App Mesh.

Service user – If you use the App Mesh service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more App Mesh features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in App Mesh, see Troubleshooting AWS App Mesh identity and access.

Service administrator – If you're in charge of App Mesh resources at your company, you probably have full access to App Mesh. It's your job to determine which App Mesh features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with App Mesh, see How AWS App Mesh works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to App Mesh. To view example App Mesh identity-based policies that you can use in IAM, see AWS App Mesh identity-based policy examples.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked

roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can

perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user

or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS App Mesh works with IAM

Before you use IAM to manage access to App Mesh, you should understand what IAM features are available to use with App Mesh. To get a high-level view of how App Mesh and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

- App Mesh identity-based policies
- App Mesh resource-based policies
- Authorization based on App Mesh tags
- App Mesh IAM roles

App Mesh identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. App Mesh supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in App Mesh use the following prefix before the action: `appmesh:`. For example, to grant someone permission to list meshes in an account with the `appmesh:ListMeshes` API operation, you include the `appmesh:ListMeshes` action in their policy. Policy statements must include either an `Action` or `NotAction` element.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [
      "appmesh:ListMeshes",
      "appmesh:ListVirtualNodes"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action.

```
"Action": "appmesh:Describe*"
```

To see a list of App Mesh actions, see Actions Defined by AWS App Mesh in the *IAM User Guide*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

The App Mesh `mesh` resource has the following ARN.

```
arn:${Partition}:appmesh:${Region}:${Account}:mesh/${MeshName}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the mesh named *`apps`* in the *`Region-code`* Region in your statement, use the following ARN.

```
arn:aws:appmesh:Region-code:111122223333:mesh/apps
```

To specify all instances that belong to a specific account, use the wildcard (*).

```
"Resource": "arn:aws:appmesh:Region-code:111122223333:mesh/*"
```

Some App Mesh actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

Many App Mesh API actions involve multiple resources. For example, `CreateRoute` creates a route with a virtual node target, so an IAM user must have permissions to use the route and the virtual node. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "arn:aws:appmesh:Region-code:111122223333:mesh/apps/virtualRouter/serviceB/route/
*",
      "arn:aws:appmesh:Region-code:111122223333:mesh/apps/virtualNode/serviceB"
]
```

To see a list of App Mesh resource types and their ARNs, see Resources Defined by AWS App Mesh in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS App Mesh.

Condition keys

App Mesh supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*. To see a list of the global condition keys that App Mesh supports, see Condition Keys for AWS App Mesh in the *IAM User Guide*. To learn with which actions and resources you can use with a condition key, see Actions Defined by AWS App Mesh.

Examples

To view examples of App Mesh identity-based policies, see AWS App Mesh identity-based policy examples.

App Mesh resource-based policies

App Mesh doesn't support resource-based policies. However, if you use the AWS Resource Access Manager (AWS RAM) service to share a mesh across AWS services, a resource-based policy is applied to your mesh by the AWS RAM service. For more information, see Granting permissions for a mesh.

Authorization based on App Mesh tags

You can attach tags to App Mesh resources or pass tags in a request to App Mesh. To control access based on tags, you provide tag information in the condition element of a policy using the `appmesh:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging App Mesh resources, see Tagging AWS Resources.

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Creating App Mesh meshes with restricted tags.

App Mesh IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with App Mesh

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

App Mesh supports using temporary credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

App Mesh supports service-linked roles. For details about creating or managing App Mesh service-linked roles, see Using service-linked roles for App Mesh.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

App Mesh does not support service roles.

AWS App Mesh identity-based policy examples

By default, IAM users and roles don't have permission to create or modify App Mesh resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete App Mesh resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and

functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the App Mesh console

To access the AWS App Mesh console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the App Mesh resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy. You can attach the `AWSAppMeshReadOnly` managed policy to users. For more information, see Adding Permissions to a User in the *IAM User Guide*.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "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": "*"
        }
    ]
}
```

Create a mesh

This example shows how you can create a policy that allows a user to create a mesh for an account, in any Region.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appmesh:CreateMesh",
            "Resource": "arn:aws:appmesh:*:123456789012:CreateMesh"
        }
    ]
}
```

List and describe all meshes

This example shows how you can create a policy that allows a user read-only access to list and describe all meshes.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appmesh:DescribeMesh",
                "appmesh:ListMeshes"
            ],
            "Resource": "*"
        }
    ]
}
```

Creating App Mesh meshes with restricted tags

You can use tags in your IAM policies to control what tags can be passed in the IAM request. You can specify which tag key-value pairs can be added, changed, or removed from an IAM user or role. This example shows how you might create a policy that allows creating a mesh, but only if the mesh is created with a tag named *teamName* and a value of *booksTeam*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appmesh:CreateMesh",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:RequestTag/teamName": "booksTeam"
                }
            }
        }
    ]
}
```

You can attach this policy to the IAM users in your account. If a user attempts to create a mesh, the mesh must include a tag named `teamName` and a value of `booksTeam`. If the mesh does not include this tag and value, the attempt to create the mesh fails. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

AWS managed policies for App Mesh

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AWSAppMeshServiceRolePolicy

You can attach `AWSAppMeshServiceRolePolicy` to your IAM entities. Enables access to AWS Services and resources used or managed by AWS App Mesh.

To view the permissions for this policy, see AWSAppMeshServiceRolePolicy in the *AWS Managed Policy Reference*.

For information on the permission details for the `AWSAppMeshServiceRolePolicy`, see see Service-Linked Role Permissions for App Mesh.

AWS managed policy: AWSAppMeshEnvoyAccess

You can attach `AWSAppMeshEnvoyAccess` to your IAM entities. App Mesh Envoy policy for accessing virtual node configuration.

To view the permissions for this policy, see AWSAppMeshEnvoyAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AWSAppMeshFullAccess

You can attach `AWSAppMeshFullAccess` to your IAM entities. Provides full access to the AWS App Mesh APIs and AWS Management Console.

To view the permissions for this policy, see AWSAppMeshFullAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AWSAppMeshPreviewEnvoyAccess

You can attach `AWSAppMeshPreviewEnvoyAccess` to your IAM entities. App Mesh Preview Envoy policy for accessing virtual node configuration.

To view the permissions for this policy, see AWSAppMeshPreviewEnvoyAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AWSAppMeshPreviewServiceRolePolicy

You can attach `AWSAppMeshPreviewServiceRolePolicy` to your IAM entities. Enables access to AWS Services and resources used or managed by AWS App Mesh.

To view the permissions for this policy, see AWSAppMeshPreviewServiceRolePolicy in the *AWS Managed Policy Reference*.

AWS managed policy: AWSAppMeshReadOnly

You can attach `AWSAppMeshReadOnly` to your IAM entities. Provides read-only access to the AWS App Mesh APIs and AWS Management Console.

To view the permissions for this policy, see AWSAppMeshReadOnly in the *AWS Managed Policy Reference*.

AWS App Mesh updates to AWS managed policies

View details about updates to AWS managed policies for AWS App Mesh since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS App Mesh Document history page.

Change	Description	Date
AWSAppMeshServiceRolePolicy, AWSServiceRoleForAppMesh – Updated policy.	Updated `AWSServiceRoleForAppMesh` and `AWSAppMeshServiceRolePolicy` to allow for	October 12, 2023

Change	Description	Date
	access to the AWS Cloud Map `DiscoverInstancesRevision` API.	

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Creating a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Creating a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Using service-linked roles for App Mesh

AWS App Mesh uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to App Mesh. Service-linked roles are predefined by App Mesh and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up App Mesh easier because you don’t have to manually add the necessary permissions. App Mesh defines the permissions of its service-linked roles, and unless defined otherwise, only App Mesh can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting its related resources. This protects your App Mesh resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for App Mesh

App Mesh uses the service-linked role named **AWSServiceRoleForAppMesh** – The role allows App Mesh to call AWS services on your behalf.

The AWSServiceRoleForAppMesh service-linked role trusts the `appmesh.amazonaws.com` service to assume the role.

Permission details

- `servicediscovery:DiscoverInstances` - Allows App Mesh to complete actions on all AWS resources.
- `servicediscovery:DiscoverInstancesRevision` - Allows App Mesh to complete actions on all AWS resources.

AWSServiceRoleForAppMesh

This policy includes the following permissions:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "CloudMapServiceDiscovery",
   "Effect": "Allow",
   "Action": [
    "servicediscovery:DiscoverInstances",
    "servicediscovery:DiscoverInstancesRevision"
   ],
   "Resource": "*"
  },
  {
   "Sid": "ACMCertificateVerification",
   "Effect": "Allow",
   "Action": [
    "acm:DescribeCertificate"
   ],
```

```
    "Resource": "*"
   }
  ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a service-linked role for App Mesh

If you created a mesh after June 5, 2019 in the AWS Management Console, the AWS CLI, or the AWS API, App Mesh created the service-linked role for you. For the service-linked role to have been created for you, the IAM account that you used to create the mesh must have had the `AWSAppMeshFullAccess` IAM policy attached to it, or a policy attached to it that contained the `iam:CreateServiceLinkedRole` permission. If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a mesh, App Mesh creates the service-linked role for you again. If your account only contains meshes created before June 5, 2019 and you want to use the service-linked role with those meshes, then you can create the role using the IAM console.

You can use the IAM console to create a service-linked role with the **App Mesh** use case. In the AWS CLI or the AWS API, create a service-linked role with the `appmesh.amazonaws.com` service name. For more information, see Creating a Service-Linked Role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for App Mesh

App Mesh does not allow you to edit the AWSServiceRoleForAppMesh service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for App Mesh

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
>
> If the App Mesh service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete App Mesh resources used by the AWSServiceRoleForAppMesh

1. Delete all routes defined for all routers in the mesh.
2. Delete all virtual routers in the mesh.
3. Delete all virtual services in the mesh.
4. Delete all virtual nodes in the mesh.
5. Delete the mesh.

Complete the previous steps for all meshes in your account.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAppMesh service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for App Mesh service-linked roles

App Mesh supports using service-linked roles in all of the Regions where the service is available. For more information, see App Mesh Endpoints and Quotas.

Envoy Proxy authorization

Proxy authorization authorizes the Envoy proxy running within an Amazon ECS task, in a Kubernetes pod running on Amazon EKS, or running on an Amazon EC2 instance to read the configuration of one or more mesh endpoints from the App Mesh Envoy Management Service. For customer accounts who already have Envoys connected to their App Mesh endpoint before 04/26/2021, proxy authorization is required for virtual nodes that use Transport Layer Security (TLS) and for virtual gateways (with or without TLS). For customer accounts who want to connect Envoys to their App Mesh endpoint after 04/26/2021, proxy authorization is required for all App Mesh capabilities. It is recommended for all customer accounts to enable proxy authorization for all virtual nodes, even if they don't use TLS, to have a secure and consistent experience using IAM for authorization to specific resources. Proxy authorization requires that the

`appmesh:StreamAggregatedResources` permission is specified in an IAM policy. The policy must be attached to an IAM role, and that IAM role must be attached to the compute resource on which you host the proxy.

Create IAM policy

If you want all mesh endpoints in a service mesh to be able to read the configuration for all mesh endpoints, then skip to Create IAM role. If you want to limit the mesh endpoints that configuration can be read from by individual mesh endpoints, then you need to create one or more IAM policies. Limiting the mesh endpoints that configuration can be read from to only the Envoy proxy running on specific compute resources is recommended. Create an IAM policy and add the `appmesh:StreamAggregatedResources` permission to the policy. The following example policy allows the configuration of the virtual nodes named `serviceBv1` and `serviceBv2` to be read in a service mesh. Configuration can't be read for any other virtual nodes defined in the service mesh. For more information about creating or editing an IAM policy, see Creating IAM Policies and Edit IAM Policies.

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

You can create multiple policies, with each policy restricting access to different mesh endpoints.

Create IAM role

If you want all mesh endpoints in a service mesh to be able to read the configuration for all mesh endpoints, then you only need to create one IAM role. If you want to limit the mesh endpoints that configuration can be read from by individual mesh endpoints, then you need to create a role

for each policy that you created in the previous step. Complete the instructions for the compute resource that the proxy runs on.

- **Amazon EKS** – If you want to use a singe role, then you can use the existing role that was created and assigned to the worker nodes when you created your cluster. To use multiple roles, your cluster must meet the requirements defined in Enabling IAM Roles for Service Accounts on your Cluster. Create the IAM roles and associate the roles with Kubernetes service accounts. For more information, see Creating an IAM Role and Policy for your Service Account and Specifying an IAM Role for your Service Account.
- **Amazon ECS** – Select **AWS service,** select **Elastic Container Service**, and then select the **Elastic Container Service Task** use case when creating your IAM role.
- **Amazon EC2** – Select **AWS service,** select **EC2**, and then select the **EC2** use case when creating your IAM role. This applies whether you host the proxy directly on an Amazon EC2 instance or on Kubernetes running on an instance.

For more information about how to create an IAM role, see Creating a Role for an AWS Service.

Attach IAM policy

If you want all mesh endpoints in a service mesh to be able to read the configuration for all mesh endpoints, then attach the `AWSAppMeshEnvoyAccess` managed IAM policy to the IAM role that you created in a previous step. If you want to limit the mesh endpoints that configuration can be read from by individual mesh endpoints, then attach each policy that you created to each role that you created. For more information about attaching a custom or managed IAM policy to an IAM role, see Adding IAM Identity Permissions.

Attach IAM role

Attach each IAM role to the appropriate compute resource:

- **Amazon EKS** – If you attached the policy to the role attached to your worker nodes, you can skip this step. If you created separate roles, then assign each role to a separate Kubernetes service account, and assign each service account to an individual Kubernetes pod deployment spec that includes the Envoy proxy. For more information, see Specifying an IAM Role for your Service Account in the *Amazon EKS User Guide* and Configure Service Accounts for Pods in the Kubernetes documentation.
- **Amazon ECS** – Attach an Amazon ECS Task Role to the task definition that includes the Envoy proxy. The task can be deployed with the EC2 or Fargate launch type. For more information

about how to create an Amazon ECS Task Role and attach it to a task, see Specifying an IAM Role for your Tasks.

- **Amazon EC2** – The IAM role must be attached to the Amazon EC2 instance that hosts the Envoy proxy. For more information about how to attach a role to an Amazon EC2 instance, see I've created an IAM role, and now I want to assign it to an EC2 instance.

Confirm permission

Confirm that the `appmesh:StreamAggregatedResources` permission is assigned to the compute resource that you host the proxy on by selecting one of the compute service names.

Amazon EKS

A custom policy may be assigned to the role assigned to the worker nodes, to individual pods, or both. It's recommended however, that you assign the policy only at individual pods, so that you can restrict access of individual pods to individual mesh endpoints. If the policy is attached to the role assigned to the worker nodes, select the **Amazon EC2** tab, and complete the steps found there for your worker node instances. To determine which IAM role is assigned to a Kubernetes pod, complete the following steps.

1. View the details of a Kubernetes deployment that includes the pod that you want to confirm that a Kubernetes service account is assigned to. The following command views the details for a deployment named *my-deployment*.

   ```
   kubectl describe deployment my-deployment
   ```

   In the returned output note the value to the right of `Service Account:`. If a line that starts with `Service Account:` doesn't exist, then a custom Kubernetes service account isn't currently assigned to the deployment. You'll need to assign one. For more information, see Configure Service Accounts for Pods in the Kubernetes documentation.
2. View the details of the service account returned in the previous step. The following command views the details of a service account named *my-service-account*.

   ```
   kubectl describe serviceaccount my-service-account
   ```

   Provided the Kubernetes service account is associated to an AWS Identity and Access Management role, one of the lines returned will look similar to the following example.

```
Annotations:        eks.amazonaws.com/role-arn=arn:aws:iam::123456789012:role/
my-deployment
```

In the previous example `my-deployment` is the name of the IAM role that the service account is associated with. If the service account output doesn't contain a line similar to the example above, then the Kubernetes service account isn't associated to an AWS Identity and Access Management account and you need to associate it to one. For more information, see Specifying an IAM Role for your Service Account.

3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
4. In the left navigation, select **Roles**. Select the name of the IAM role that you noted in a previous step.
5. Confirm that either the custom policy you created previously, or the `AWSAppMeshEnvoyAccess` managed policy is listed. If neither policy is attached, attach an IAM policy to the IAM role. If you want to attach a custom IAM policy but don't have one, then you need to create a custom IAM policy with the required permissions. If a custom IAM policy is attached, select the policy and confirm that it contains `"Action": "appmesh:StreamAggregatedResources"`. If it does not, then you need to add that permission to your custom IAM policy. You can also confirm that the appropriate Amazon Resource Name (ARN) for a specific mesh endpoint is listed. If no ARNs are listed, then you can edit the policy to add, remove, or change the listed ARNs. For more information, see Edit IAM Policies and Create IAM policy.
6. Repeat the previous steps for each Kubernetes pod that contains the Envoy proxy.

Amazon ECS

1. From the Amazon ECS console, choose **Task Definitions**.
2. Select your Amazon ECS task.
3. On the **Task Definition Name** page, select your task definition.
4. On the **Task Definition** page, select the link of the IAM role name that is to the right of **Task Role**. If an IAM role isn't listed, then you need to create an IAM role and attach it to your task by updating your task definition.
5. In the **Summary** page, on the **Permissions** tab, confirm that either the custom policy you created previously, or the `AWSAppMeshEnvoyAccess` managed policy is listed. If

neither policy is attached, attach an IAM policy to the IAM role. If you want to attach a custom IAM policy but don't have one, then you need to create the custom IAM policy. If a custom IAM policy is attached, select the policy and confirm that it contains `"Action": "appmesh:StreamAggregatedResources"`. If it does not, then you need to add that permission to your custom IAM policy. You can also confirm that the appropriate Amazon Resource Name (ARN) for a specific mesh endpoints is listed. If no ARNs are listed, then you can edit the policy to add, remove, or change the listed ARNs. For more information, see Edit IAM Policies and Create IAM policy.

6. Repeat the previous steps for each task definition that contains the Envoy proxy.

Amazon EC2

1. From the Amazon EC2 console, select **Instances** in the left navigation.
2. Select one of your instances that hosts the Envoy proxy.
3. In the **Description** tab, select the link of the IAM role name that is to the right of **IAM role**. If an IAM role isn't listed, then you need to create an IAM role.
4. In the **Summary** page, on the **Permissions** tab, confirm that either the custom policy you created previously, or the `AWSAppMeshEnvoyAccess` managed policy is listed. If neither policy is attached, attach the IAM policy to the IAM role. If you want to attach a custom IAM policy but don't have one, then you need to create the custom IAM policy. If a custom IAM policy is attached, select the policy and confirm that it contains `"Action": "appmesh:StreamAggregatedResources"`. If it does not, then you need to add that permission to your custom IAM policy. You can also confirm that the appropriate Amazon Resource Name (ARN) for a specific mesh endpoints is listed. If no ARNs are listed, then you can edit the policy to add, remove, or change the listed ARNs. For more information, see Edit IAM Policies and Create IAM policy.
5. Repeat the previous steps for each instance that you host the Envoy proxy on.

Troubleshooting AWS App Mesh identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with App Mesh and IAM.

Topics

- I am not authorized to perform an action in App Mesh

- I want to allow people outside of my AWS account to access my App Mesh resources

I am not authorized to perform an action in App Mesh

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following error occurs when the `mateojackson` IAM user tries to use the console to create a virtual node named *`my-virtual-node`* in the mesh named *`my-mesh`* but does not have the `appmesh:CreateVirtualNode` permission.

```
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
```

In this case, Mateo asks his administrator to update his policies to allow him to create a virtual node using the `appmesh:CreateVirtualNode` action.

> **Note**
>
> Since a virtual node is created within a mesh, Mateo's account also requires the `appmesh:DescribeMesh` and `appmesh:ListMeshes` actions to create the virtual node in the console.

I want to allow people outside of my AWS account to access my App Mesh resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether App Mesh supports these features, see How AWS App Mesh works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.

- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging AWS App Mesh API calls using AWS CloudTrail

AWS App Mesh is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures all API calls for App Mesh as events. The calls captured include calls from the App Mesh console and code calls to the App Mesh API operations. Using the information collected by CloudTrail, you can determine the request that was made to App Mesh, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail or a CloudTrail Lake event data store.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region

trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to Apache ORC format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying advanced event selectors. The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see Working with AWS CloudTrail Lake in the *AWS CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the pricing option you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

App Mesh management events in CloudTrail

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

AWS App Mesh logs all App Mesh control plane operations as management events. For a list of the AWS App Mesh control plane operations that App Mesh logs to CloudTrail, see the AWS App Mesh API Reference.

App Mesh event examples

An event represents a single request from any source and includes information about the requested API operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `StreamAggregatedResources` action.


```
{
    "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"
}
```

For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

Data protection in AWS App Mesh

The AWS shared responsibility model applies to data protection in AWS App Mesh. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with App Mesh or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption

Your data is encrypted when using App Mesh.

Encryption at rest

By default, the App Mesh configurations that you create are encrypted at rest.

Encryption in transit

App Mesh service endpoints use the HTTPS protocol. All communication between the Envoy proxy and the App Mesh Envoy Management Service is encrypted. If you require FIPS compliant encryption for the communication between the Envoy proxy and the App Mesh Envoy Management Service, there is a FIPS variant of the Envoy proxy container image you can use. For more information, see Envoy image.

Communication between containers within virtual nodes is not encrypted, but this traffic doesn't leave the network namespace.

Compliance validation for AWS App Mesh

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Infrastructure security in AWS App Mesh

As a managed service, AWS App Mesh is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access App Mesh through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.

- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can improve the security posture of your VPC by configuring App Mesh to use an interface VPC endpoint. For more information, see App Mesh interface VPC endpoints (AWS PrivateLink).

App Mesh interface VPC endpoints (AWS PrivateLink)

You can improve the security posture of your Amazon VPC by configuring App Mesh to use an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access App Mesh APIs by using private IP addresses. PrivateLink restricts all network traffic between your Amazon VPC and App Mesh to the Amazon network.

You're not required to configure PrivateLink, but we recommend it. For more information about PrivateLink and interface VPC endpoints, see Accessing Services Through AWS PrivateLink.

Considerations for App Mesh interface VPC endpoints

Before you set up interface VPC endpoints for App Mesh, be aware of the following considerations:

- If your Amazon VPC doesn't have an internet gateway and your tasks use the `awslogs` log driver to send log information to CloudWatch Logs, you must create an interface VPC endpoint for CloudWatch Logs. For more information, see Using CloudWatch Logs with Interface VPC Endpoints in the *Amazon CloudWatch Logs User Guide*.
- VPC endpoints don't support AWS cross-Region requests. Ensure that you create your endpoint in the same Region where you plan to issue your API calls to App Mesh.
- VPC endpoints only support Amazon-provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see DHCP Options Sets in the *Amazon VPC User Guide*.
- The security group attached to the VPC endpoint must allow incoming connections on port 443 from the private subnet of the Amazon VPC.

> **Note**
>
> Controlling access to App Mesh by attaching an endpoint policy to the VPC endpoint (for example, using the service name `com.amazonaws.Region.appmesh-envoy-management`) isn't supported for Envoy connection.

For additional considerations and limitations, see Interface Endpoint Availability Zone Considerations and Interface Endpoint Properties and Limitations.

Create the interface VPC endpoint for App Mesh

To create the interface VPC endpoint for the App Mesh service, use the Creating an Interface Endpoint procedure in the *Amazon VPC User Guide*. Specify `com.amazonaws.Region.appmesh-envoy-management` for the service name for your Envoy proxy to connect to the App Mesh's public Envoy management service and `com.amazonaws.Region.appmesh` for mesh operations.

> **Note**
>
> *Region* represents the Region identifier for an AWS Region supported by App Mesh, such as `us-east-2` for the US East (Ohio) Region.

Though you can define an interface VPC endpoint for App Mesh in any Region where App Mesh is supported, you may not be able to define an endpoint for all Availability Zones in each Region. To find out which Availability Zones are supported with interface VPC endpoints in a Region, use the describe-vpc-endpoint-services command or use the AWS Management Console. For example, the following commands return the availability zones to which you can deploy an App Mesh interface VPC endpoints within the US East (Ohio) Region:

```
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[]'
```

Resilience in AWS App Mesh

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

App Mesh runs its control plane instances across multiple Availability Zones to ensure high availability. App Mesh automatically detects and replaces unhealthy control plane instances, and it provides automated version upgrades and patching for them.

Disaster recovery in AWS App Mesh

The App Mesh service manages backups of customer data. There is nothing that you need to do to manage backups. The backed-up data is encrypted.

Configuration and vulnerability analysis in AWS App Mesh

App Mesh vends a managed Envoy proxy Docker container image that you deploy with your microservices. App Mesh ensures that the container image is patched with the latest vulnerability and performance patches. App Mesh tests new Envoy proxy releases against the App Mesh feature set before making the images available to you.

You must update your microservices to use the updated container image version. Following is the latest version of the image.

```
840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-prod
```

App Mesh troubleshooting

This chapter discusses troubleshooting best practices and common issues that you might encounter when using App Mesh. Select one of the following areas to review best practices and common issues for that area.

Topics

App Mesh troubleshooting best practices

We recommend that you follow the best practices in this topic to troubleshoot issues when using App Mesh.

Enable the Envoy proxy administration interface

The Envoy proxy ships with an administration interface that you can use to discover configuration and statistics and to perform other administrative functions such as connection draining. For more information, see Administration interface in the Envoy documentation.

If you use the managed Envoy image, the administration endpoint is enabled by default on port 9901. Examples provided in App Mesh setup troubleshooting display the example administration endpoint URL as `http://my-app.default.svc.cluster.local:9901/`.

> **Note**
>
> The administration endpoint should never be exposed to the public internet. Additionally, we recommend monitoring the administration endpoint logs, which

are set by the `ENVOY_ADMIN_ACCESS_LOG_FILE` environment variable to `/tmp/envoy_admin_access.log` by default.

Enable Envoy DogStatsD integration for metric offload

The Envoy proxy can be configured to offload statistics for OSI Layer 4 and Layer 7 traffic and for internal process health. While this topic shows how to use these statistics without offloading the metrics to sinks like CloudWatch metrics and Prometheus., having these statistics in a centralized location for all of your applications can help you diagnose issues and confirm behavior more quickly. For more information, see Using Amazon CloudWatch Metrics and the Prometheus documentation.

You can configure DogStatsD metrics by setting the parameters defined in DogStatsD variables. For more information about DogStatsD, see the DogStatsD documentation. You can find a demonstration of metric offload to AWS CloudWatch metrics in the App Mesh with Amazon ECS basics walk-through on GitHub.

Enable access logs

We recommend enabling access logs on your Virtual nodes and Virtual gateways to discover details about traffic transiting between your applications. For more information, see Access logging in the Envoy documentation. The logs provide detailed information on OSI Layer 4 and Layer 7 traffic behavior. When you use Envoy's default format, you can analyze the access logs with CloudWatch Logs Insights using the following parse statement.

```
parse @message "[*] \"* * *\" * * * * * * * * * * *" as StartTime, Method, Path,
 Protocol, ResponseCode, ResponseFlags, BytesReceived, BytesSent, DurationMillis,
 UpstreamServiceTimeMillis, ForwardedFor, UserAgent, RequestId, Authority, UpstreamHost
```

Enable Envoy debug logging in pre-production environments

We recommend setting the Envoy proxy's log level to debug in a pre-production environment. Debug logs can help you identify issues before you graduate the associated App Mesh configuration to your production environment.

If you're using the Envoy image, you can set the log level to debug through the `ENVOY_LOG_LEVEL` environment variable.

> **Note**
>
> We do not recommend using the debug level in production environments. Setting the level to debug increases the logging and may affect performance and the overall cost of logs offloaded to solutions like CloudWatch Logs.

When you use Envoy's default format, you can analyze the process logs with CloudWatch Logs Insights using the following parse statement:

```
parse @message "[*][*][*][*] [*] *" as Time, Thread, Level, Name, Source, Message
```

Monitor the Envoy Proxy Connectivity with App Mesh control plane

We recommend you monitor the Envoy metrics `control_plane.connected_state` to make sure that the Envoy proxy communicates with the App Mesh control plane to fetch the dynamic configuration resources. For more information, see Management Server.

App Mesh setup troubleshooting

This topic details common issues that you may experience with App Mesh setup.

Cannot pull Envoy container image

Symptoms

You receive the following error message in an Amazon ECS task. The Amazon ECR *account ID* and *Region* in the following message may be different, depending on which Amazon ECR repository that you pulled the container image from.

```
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'
```

Resolution

This error indicates that the task execution role being used does not have permission to communicate to Amazon ECR and cannot pull the Envoy container image from the repository.

The task execution role assigned to your Amazon ECS task needs an IAM policy with the following statements:

```
{
  "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"
}
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Cannot connect to App Mesh Envoy management service

Symptoms

Your Envoy proxy is unable to connect to the App Mesh Envoy management service. You are seeing:

- Connection refused errors
- Connection timeouts
- Errors resolving the App Mesh Envoy management service endpoint
- gRPC errors

Resolution

Make sure that your Envoy proxy has access to the internet or to a private VPC endpoint and that your security groups allow outbound traffic on port 443. App Mesh's public Envoy management service endpoints follow the fully qualified domain name (FQDN) format.

```
# App Mesh Production Endpoint
appmesh-envoy-management.Region-code.amazonaws.com

# App Mesh Preview Endpoint
```

```
appmesh-preview-envoy-management.Region-code.amazonaws.com
```

You can debug your connection to EMS using the command below. This sends a valid, but empty gRPC request to the Envoy Management Service.

```
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
```

If you receive these messages back, your connection to Envoy Management Service is functional. For debugging gRPC related errors, see the errors in Envoy disconnected from App Mesh Envoy management service with error text.

```
grpc-status: 16
grpc-message: Missing Authentication Token
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Envoy disconnected from App Mesh Envoy management service with error text

Symptoms

Your Envoy proxy is unable to connect to the App Mesh Envoy management service and receive its configuration. Your Envoy proxy logs contain a log entry like the following.

```
gRPC config stream closed: gRPC status code, message
```

Resolution

In most cases, the message portion of the log should indicate the problem. The following table lists the most common gRPC status codes that you might see, their causes, and their resolutions.

gRPC status code	Cause	Resolution
0	Graceful disconnect from the Envoy management service.	There is no issue. App Mesh occasionally disconnects Envoy proxies with this status code. Envoy will reconnect

gRPC status code	Cause	Resolution
		and continue receiving updates.
3	The mesh endpoint (virtual node or virtual gateway), or one of its associated resources, could not be found.	Double check your Envoy configuration to make sure that it has the appropria te name of the App Mesh resource that it represents. If your App Mesh resource is integrated with other AWS resources, such as AWS Cloud Map namespaces or ACM certificates, then make sure that those resources exist.
7	The Envoy proxy is unauthori zed to perform an action, such as connect to the Envoy management service, or retrieve associated resources.	Make sure that you create an IAM policy that has the appropriate policy statement s for App Mesh and other services and attach that policy to the IAM user or role that your Envoy proxy is using to connect to the Envoy management service.
8	The number of Envoy proxies for a given App Mesh resource exceeds the account-level service quota.	See App Mesh service quotas for information on default account quotas and how to request a quota increase.

gRPC status code	Cause	Resolution
16	The Envoy proxy does not have valid authentication credentials for AWS.	Make sure that the Envoy has appropriate credentials to connect to AWS services through an IAM user or role. A known issue, #24136, in Envoy for version `v1.24` and before fails to fetch the credentials if Envoy process uses over `1024` file descriptors. This happens when Envoy is serving high traffic volume. You can confirm this issue by checking Envoy logs at debug level for the text "`A libcurl function was given a bad argument`". To mitigate this issue, upgrade to Envoy version `v1.25.1.0-prod` or later.

You can observe the status codes and messages from your Envoy proxy with Amazon CloudWatch Insights by using the following query:

```
filter @message like /gRPC config stream closed/
| parse @message "gRPC config stream closed: *, *" as StatusCode, Message
```

If the provided error message was not helpful, or your issue is still not resolved, then consider opening a GitHub issue.

Envoy container health check, readiness probe, or liveliness probe failing

Symptoms

Your Envoy proxy is failing health checks in an Amazon ECS task, Amazon EC2 instance, or Kubernetes pod. For example, you query the Envoy administration interface with the following command and receive a status other than `LIVE`.

```
curl -s http://my-app.default.svc.cluster.local:9901/server_info | jq '.state'
```

Resolution

The following is a list of remediation steps depending on the status returned by the Envoy proxy.

- `PRE_INITIALIZING` or `INITIALIZING` – The Envoy proxy has yet to receive configuration, or cannot connect and retrieve configuration from App Mesh Envoy management service. The Envoy may be receiving an error from the Envoy management service when trying to connect. For more information, see the errors in Envoy disconnected from App Mesh Envoy management service with error text.
- `DRAINING` – The Envoy proxy has begun draining connections in response to a `/healthcheck/fail` or `/drain_listeners` request on the Envoy administration interface. We do not recommend invoking these paths on the administration interface unless you are about to terminate your Amazon ECS task, Amazon EC2 instance, or Kubernetes pod.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Health check from the load balancer to the mesh endpoint is failing

Symptoms

Your mesh endpoint is considered healthy by the container health check or readiness probe, but the health check from the load balancer to the mesh endpoint is failing.

Resolution

To resolve the issue, complete the following tasks.

- Make sure that the security group associated with your mesh endpoint accepts inbound traffic on the port you've configured for your health check.
- Make sure that the health check is succeeding consistently when requested manually; for example, from a bastion host within your VPC.

- If you are configuring a health check for a virtual node, then we recommend implementing a health check endpoint in your application; for example, /ping for HTTP. This ensures that both the Envoy proxy and your application are routable from the load balancer.
- You can use any elastic load balancer type for the virtual node, depending on the features that you need. For more information, see Elastic Load Balancing features.
- If you are configuring a health check for a virtual gateway, then we recommend using a network load balancer with a TCP or TLS health check on the virtual gateway's listener port. This ensures that the virtual gateway listener is bootstrapped and ready to accept connections.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Virtual gateway not accepting traffic on ports 1024 or less

Symptoms

Your virtual gateway is not accepting traffic on port 1024 or less, but does accept traffic on a port number that is greater than 1024. For example, you query the Envoy stats with the following command and receive a value other than zero.

```
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "update_rejected"
```

You might see text similar to the following text in your logs describing a failure to bind to a privileged port:

```
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
```

Resolution

To resolve the issue, the user specified for the gateway needs to have the linux capability `CAP_NET_BIND_SERVICE`. For more information, see Capabilities in the Linux Programmer's Manual, Linux parameters in ECS Task definition parameters, and Set capabilities for a container in the Kubernetes documentation.

> **Important**
>
> Fargate must use a port value greater than 1024.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

App Mesh connectivity troubleshooting

This topic details common issues that you may experience with App Mesh connectivity.

Unable to resolve DNS name for a virtual service

Symptoms

Your application is unable to resolve the DNS name of a virtual service that it is attempting to connect to.

Resolution

This is a known issue. For more information, see the Name VirtualServices by any hostname or FQDN GitHub issue. Virtual services in App Mesh can be named anything. As long as there is a DNS A record for the virtual service name and the application can resolve the virtual service name, the request will be proxied by Envoy and routed to its appropriate destination. To resolve the issue, add a DNS A record to any non-loopback IP address, such as `10.10.10.10`, for the virtual service name. The DNS A record can be added under the following conditions:

- In Amazon Route 53, if the name is suffixed by your private hosted zone name
- Within the application container's `/etc/hosts` file
- In a third-party DNS server that you manage

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Unable to connect to a virtual service backend

Symptoms

Your application is unable to establish a connection to a virtual service defined as a backend on your virtual node. When attempting to establish a connection, the connection may fail entirely, or the request from the application's perspective may fail with an `HTTP 503` response code.

Resolution

If the application fails to connect at all (no `HTTP 503` response code returned), then do the following:

- Make sure that your compute environment has been set up to work with App Mesh.
  - For Amazon ECS, make sure that you have the appropriate proxy configuration enabled. For an end-to-end walkthrough, see Getting Started with App Mesh and Amazon ECS.
  - For Kubernetes, including Amazon EKS, make sure that you have the latest App Mesh controller installed via Helm. For more information, see App Mesh Controller on Helm Hub or Tutorial: Configure App Mesh integration with Kubernetes.
  - For Amazon EC2, make sure that you have setup your Amazon EC2 instance for proxying App Mesh traffic. For more information, see Update services.
- Make sure that the Envoy container that is running on your compute service has successfully connected to the App Mesh Envoy management service. You can confirm this by checking Envoy stats for the field `control_plane.connected_state`. For more information on `control_plane.connected_state`, see Monitor the Envoy Proxy Connectivity in our **Troubleshooting Best Practices**.

  If the Envoy was able to establish the connection initially, but later was disconnected and never reconnected, see Envoy disconnected from App Mesh Envoy management service with error text to troubleshoot why it was disconnected.

If the application connects but the request fails with an `HTTP 503` response code, try the following:

- Make sure that the virtual service you're connecting to exists in the mesh.
- Make sure that the virtual service has a provider (a virtual router or virtual node).
- When using Envoy as an HTTP Proxy, if you're seeing egress traffic coming into `cluster.cds_egress_*_mesh-allow-all` instead of the correct destination through Envoy stats, most likely Envoy isn't routing requests properly through `filter_chains`. This can be a result of using an unqualified virtual service name. We recommend that you use the service discovery name of the actual service as the virtual service name, because Envoy proxy communicates with other virtual services through their names.

  For more information, see virtual services.
- Inspect the Envoy proxy logs for any of the following error messages:
  - `No healthy upstream` – The virtual node that the Envoy proxy is attempting to route to does not have any resolved endpoints, or it does not have any healthy endpoints. Make sure that the target virtual node has the correct service discovery and health check settings.

If requests to the service are failing during a deployment or scaling of the backend virtual service, follow the guidance in Some requests fail with HTTP status code 503 when a virtual service has a virtual node provider.

- `No cluster match for URL` – This is most likely caused when a request is sent to a virtual service that does not match the criteria defined by any of the routes defined under a virtual router provider. Make sure that the requests from the application are sent to a supported route by ensuring the path and HTTP request headers are correct.
- `No matching filter chain found` – This is most likely caused when a request is sent to a virtual service on an invalid port. Make sure that the requests from the application are using the same port specified on the virtual router.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Unable to connect to an external service

Symptoms

Your application is unable to connect to a service outside of the mesh, such as `amazon.com`.

Resolution

By default, App Mesh does not allow outbound traffic from applications within the mesh to any destination outside of the mesh. To enable communication with an external service, there are two options:

- Set the outbound filter on the mesh resource to `ALLOW_ALL`. This setting will allow any application within the mesh to communicate with any destination IP address inside or outside of the mesh.
- Model the external service in the mesh using a virtual service, virtual router, route, and virtual node. For example, to model the external service `example.com`, you can create a virtual service named `example.com` with a virtual router and route that sends all traffic to a virtual node with a DNS service discovery hostname of `example.com`.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Unable to connect to a MySQL or SMTP server

Symptoms

When allowing outbound traffic to all destinations (Mesh `EgressFilter type=ALLOW_ALL`), such as an SMTP server or a MySQL database using a virtual node definition, the connection from your application fails. As an example, the following is an error message from attempting to connect to a MySQL server.

```
ERROR 2013 (HY000): Lost connection to MySQL server at 'reading initial communication
 packet', system error: 0
```

Resolution

This is a known issue that is resolved by using App Mesh image version 1.15.0 or later. For more information, see the Unable to connect to MySQL with App Mesh GitHub issue. This error occurs because the outbound listener in Envoy configured by App Mesh adds the Envoy TLS Inspector listener filter. For more information, see TLS Inspector in the Envoy documentation. This filter evaluates whether or not a connection is using TLS by inspecting the first packet sent from the client. With MySQL and SMTP, however, the server sends the first packet after connection. For more information about MySQL, see Initial Handshake in the MySQL documentation. Because the server sends the first packet, inspection at the filter fails.

To work around this issue depending on your version of Envoy:

- If your App Mesh image Envoy version is 1.15.0 or later, do not model external services such as **MySQL**, **SMTP**, **MSSQL**, etc. as a backend for your application's virtual node.
- If your App Mesh image Envoy version is prior to 1.15.0, add port `3306` to the list of values for the `APPMESH_EGRESS_IGNORED_PORTS` in your services for **MySQL** and as the port you are using for **STMP**.

> **Important**
>
> While the standard SMTP ports are 25, 587, and 465, you should only add the port you are using to `APPMESH_EGRESS_IGNORED_PORTS` and not all three.

For more information, see Update services for Kubernetes , Update services for Amazon ECS, or Update services for Amazon EC2.

If your issue is still not resolved, then you can provide us with details on what you're experiencing using the existing GitHub issue or contact AWS Support.

Unable to connect to a service modeled as a TCP virtual node or virtual router in App Mesh

Symptoms

Your application is unable to connect to a backend that uses the TCP protocol setting in the App Mesh PortMapping definition.

Resolution

This is a known issue. For more information, see Routing to multiple TCP destinations on the same port on GitHub. App Mesh does not currently allow multiple backend destinations modeled as TCP to share the same port due to restrictions in the information provided to the Envoy proxy at OSI Layer 4. To make sure that TCP traffic can be routed appropriately for all backend destinations, do the following:

- Make sure that all destinations are using a unique port. If you are using a virtual router provider for the backend virtual service, you can change the virtual router port without changing the port on the virtual nodes that it routes to. This allows the applications to open connections on the virtual router port while the Envoy proxy continues to use the port defined in the virtual node.
- If the destination modeled as TCP is a MySQL server, or any other TCP-based protocol in which the server sends the first packets after connection, see Unable to connect to a MySQL or SMTP server.

If your issue is still not resolved, then you can provide us with details on what you're experiencing using the existing GitHub issue or contact AWS Support.

Connectivity succeeds to service not listed as a virtual service backend for a virtual node

Symptoms

Your application is able to connect and send traffic to a destination that is not specified as a virtual service backend on your virtual node.

Resolution

If requests are succeeding to a destination that has not been modeled in the App Mesh APIs, then the most likely cause is that the mesh's outbound filter type has been set to `ALLOW_ALL`. When the outbound filter is set to `ALLOW_ALL`, an outbound request from your application that does not match a modeled destination (backend) will be sent to the destination IP address set by the application.

If you want to disallow traffic to destinations not modeled in the mesh, consider setting the outbound filter value to `DROP_ALL`.

> **Note**
>
> Setting the mesh outbound filter value affects all virtual nodes within the mesh.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Some requests fail with HTTP status code 503 when a virtual service has a virtual node provider

Symptoms

A portion of your application's requests fail to a virtual service backend that is using a virtual node provider instead of a virtual router provider. When using a virtual router provider for the virtual service, requests do not fail.

Resolution

This is a known issue. For more information, see Retry policy on Virtual Node provider for a Virtual Service on GitHub. When using a virtual node as a provider for a virtual service, you cannot specify the default retry policy that you want the clients of your virtual service to use. By comparison, virtual router providers allow retry policies to be specified because they are a property of the child route resources.

To reduce request failures to virtual node providers, use a virtual router provider instead, and specify a retry policy on its routes. For other ways to reduce request failures to your applications, see App Mesh best practices.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Unable to connect to an Amazon EFS filesystem

Symptoms

When configuring an Amazon ECS task with an Amazon EFS filesystem as a volume, the task fails to start with the following error.

```
ResourceInitializationError: failed to invoke EFS utils commands to set up EFS volumes:
 stderr: mount.nfs4: Connection refused : unsuccessful EFS utils command execution;
 code: 32
```

Resolution

This is a known issue. This error occurs because the NFS connection to Amazon EFS occurs before any containers in your task are started. This traffic is routed by the proxy configuration to Envoy, which will not be running at this point. Because of the ordering of startup, the NFS client fails to connecting to the Amazon EFS filesystem and the task fails to launch. To resolve the issue, add port `2049` to the list of values for the `EgressIgnoredPorts` setting in the proxy configuration of your Amazon ECS task definition. For more information, see Proxy configuration.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Connectivity succeeds to service, but the incoming request does not appear in access logs, traces, or metrics for Envoy

Symptoms

Even though your application can connect and send requests to another application, you either can not see incoming requests in the access logs or in tracing information for the Envoy proxy.

Resolution

This is a known issue. From more information, see iptables rules setup issue on Github. The Envoy proxy only intercepts inbound traffic to the port of which its corresponding virtual node is listening on. Requests to any other port will bypass the Envoy proxy and reach to the service behind it directly. In order to let the Envoy proxy intercept the inbound traffic for your service you need to set your virtual node and service to listen on the same port.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Setting the HTTP_PROXY/HTTPS_PROXY environment variables at container level doesn't work as expected.

Symptoms

When HTTP_PROXY/HTTPS_PROXY is set as an environment variable at the:

- App container in the task definition with App Mesh enabled, requests being sent to the namespace of the App Mesh services will get HTTP 500 error responses from the Envoy sidecar.
- Envoy container in task definition with App Mesh enabled, requests coming out of Envoy sidecar will not go through the HTTP/HTTPS proxy server, and the environment variable will not work.

Resolution

For the app container:

App Mesh functions by having traffic within your task go through the Envoy proxy. HTTP_PROXY/HTTPS_PROXY configuration overrides this behavior by configuring container traffic to go through a different external proxy. The traffic will still be intercepted by Envoy, but it doesn't support proxying the mesh traffic using an external proxy.

If you want to proxy all non-mesh traffic, please set NO_PROXY to include your mesh's CIDR/namespace, localhost, and the credential's endpoints like in the following example.

```
NO_PROXY=localhost,127.0.0.1,169.254.169.254,169.254.170.2,10.0.0.0/16
```

For the Envoy container:

Envoy doesn't support a generic proxy. We do not recommend setting these variables.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Upstream request timeouts even after setting the timeout for routes.

Symptoms

You defined the timeout for:

- The routes, but you are still getting an upstream request timeout error.

- The virtual node listener and the timeout and the retry timeout for the routes, but you are still getting an upstream request timeout error.

Resolution

For the high latency requests greater than 15 seconds to complete successfully, you need to specify a timeout at both the route and virtual node listener level.

If you specify a route timeout that is greater than the default 15 seconds, make sure that the timeout is also specified for the listener for all participating virtual nodes. However, if you decrease the timeout to a value that is lower than the default, it's optional to update the timeouts at virtual nodes. For more information about options when setting up virtual nodes and routes, see virtual nodes and routes.

If you specified a **retry policy**, the duration that you specify for the request timeout should always be greater than or equal to the *retry timeout* multiplied by the *max retries* that you defined in the **retry policy**. This allows your request with all the retries to complete successfully. For more information, see routes.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Envoy responds with HTTP Bad request.

Symptoms

Envoy responds with **HTTP 400 Bad request** for all requests sent through the Network Load Balancer (NLB). When we check the Envoy logs, we see:

- Debug logs:

```
dispatch error: http/1.1 protocol error: HPE_INVALID_METHOD
```

- Access logs:

```
"- - HTTP/1.1" 400 DPE 0 11 0 - "-" "-" "-" "-" "-"
```

Resolution

The resolution is to disable the proxy protocol version 2 (PPv2) on your NLB's target group attributes.

As of today the PPv2 is not supported by virtual gateway and virtual node Envoy that are run using the App Mesh control plane. If you deploy NLB using AWS load balancer controller on Kubernetes, then disable PPv2 by setting the following attribute to `false`:

```
service.beta.kubernetes.io/aws-load-balancer-target-group-attributes:
 proxy_protocol_v2.enabled
```

See AWS Load Balancer Controller Annotations for more details about NLB resource attributes.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Unable to configure timeout properly.

Symptoms

Your request timeouts within 15 seconds even after configuring the timeout on the virtual node listener and the timeout on the route towards virtual node backend.

Resolution

Make sure that the correct virtual service is included under the backend list.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

App Mesh scaling troubleshooting

This topic details common issues that you may experience with App Mesh scaling.

Connectivity fails and container health checks fail when scaling beyond 50 replicas for a virtual node/virtual gateway

Symptoms

When you are scaling the number of replicas, such as Amazon ECS tasks, Kubernetes pods, or Amazon EC2 instances, for a virtual node/virtual gateway beyond 50, Envoy container health checks for new and currently running Envoys begin to fail. Downstream applications sending traffic to the virtual node/virtual gateway begin seeing request failures with HTTP status code `503`.

Resolution

App Mesh's default quota for the number of Envoys per virtual node/virtual gateway is 50. When the number of running Envoys exceeds this quota, new and currently running

Envoys fail to connect to App Mesh's Envoy management service with gRPC status code 8 (RESOURCE_EXHAUSTED). This quota can be raised. For more information, see App Mesh service quotas.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Requests fail with 503 when a virtual service backend horizontally scales out or in

Symptoms

When a backend virtual service is horizontally scaled out or in, requests from downstream applications fail with an HTTP 503 status code.

Resolution

App Mesh recommends several approaches to mitigate failure cases while scaling applications horizontally. For detailed information about how to prevent these failures, see App Mesh best practices.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Envoy container crashes with segfault under increased load

Symptoms

Under a high traffic load, the Envoy proxy crashes due to a segmentation fault (Linux exit code 139). The Envoy process logs contain a statement like the following.

```
Caught Segmentation fault, suspect faulting address 0x0"
```

Resolution

The Envoy proxy has likely breached the operating system's default nofile ulimit, the limit on the number of files a process can have open at a time. This breach is due to the traffic causing more connections, which consume additional operating system sockets. To resolve this issue, increase the ulimit nofile value on the host operating system. If you are using Amazon ECS, this limit can be changed through the Ulimit settings on the task definition's resource limits settings.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Increase in default resources is not reflected in Service Limits

Symptoms

After increasing the default limit of App Mesh resources, the new value is not reflected when you look at your service limits.

Resolution

While the new limits aren't currently shown, customers can still exercise them.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Application crashes due to a huge number of health checks calls.

Symptoms

After enabling active health checks for a virtual node, there is an uptick in the number of health check calls. The application crashes due to the greatly increased volume of health check calls made to the application.

Resolution

When active health checking is enabled, each Envoy endpoint of the downstream (client) sends health requests to each endpoint of the upstream cluster (server) in order to make routing decisions. As a result the total number of health check requests would be `number of client Envoys * number of server Envoys * active health check frequency`.

To resolve this issue, modify the frequency of the health check probe, which would reduce the total volume of health check probes. In addition to active health checks, App Mesh allows configuring outlier detection as means of passive health checking. Use outlier detection to configure when to remove a particular host based on consecutive `5xx` responses.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

App Mesh observability troubleshooting

This topic details common issues that you may experience with App Mesh observability.

Unable to see AWS X-Ray traces for my applications

Symptoms

Your application in App Mesh is not displaying X-Ray tracing information in the X-Ray console or APIs.

Resolution

To use X-Ray in App Mesh, you must correctly configure components to enable communication between your application, sidecar containers, and the X-Ray service. Take the following steps to confirm that X-Ray has been set up correctly:

- Make sure the App Mesh Virtual Node listener protocol is not set as `TCP`.
- Make sure that the X-Ray container that is deployed with your application exposes UDP port `2000` and runs as user `1337`. For more information, see the Amazon ECS X-Ray example on GitHub.
- Make sure that the Envoy container has tracing enabled. If you are using the App Mesh Envoy image, you can enable X-Ray by setting the `ENABLE_ENVOY_XRAY_TRACING` environment variable to a value of `1` and the `XRAY_DAEMON_PORT` environment variable to `2000`.
- If you've instrumented X-Ray in your application code with one of the language-specific SDKs , then make sure that it is configured correctly by following the guides for your language.
- If all of the previous items are configured correctly, then review the X-Ray container logs for errors and follow the guidance in Troubleshooting AWS X-Ray. A more detailed explanation of X-Ray integration in App Mesh can be found in Integrating X-Ray with App Mesh.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Unable to see Envoy metrics for my applications in Amazon CloudWatch metrics

Symptoms

Your application in App Mesh is not emitting metrics generated by the Envoy proxy to CloudWatch metrics.

Resolution

When you use CloudWatch metrics in App Mesh, you must correctly configure several components to enable communication between your Envoy proxy, CloudWatch agent sidecar, and the CloudWatch metrics service. Take the following steps to confirm that CloudWatch metrics for Envoy proxy have been setup correctly:

- Make sure that you are using the CloudWatch agent image for App Mesh. For more information, see App Mesh CloudWatch agent on GitHub.
- Make sure that you have configured the CloudWatch agent for App Mesh appropriately by following the platform-specific usage instructions. For more information, see App Mesh CloudWatch agent on GitHub.
- If all of the previous items are configured correctly, then review the CloudWatch agent container logs for errors and follow the guidance provided in Troubleshooting the CloudWatch agent.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Unable to configure custom sampling rules for AWS X-Ray traces

Symptoms

Your application is using X-Ray tracing, but you are unable to configure sampling rules for your traces.

Resolution

Since App Mesh Envoy currently does not support **Dynamic X-Ray sampling configuration**, the following workarounds are available.

If your Envoy version is `1.19.1` or later, you have the following options.

- To only set the sampling rate, use the `XRAY_SAMPLING_RATE` environment variable on the Envoy container. The value should be specified as a decimal between `0` and `1.00` (100%). For more information, see AWS X-Ray variables.
- To configure the localized custom sampling rules for the X-Ray tracer use the `XRAY_SAMPLING_RULE_MANIFEST` environment variable to specify a file path in the Envoy container file system. For more information, see Sampling rules in the *AWS X-Ray Developer Guide*.

If your Envoy version is prior to `1.19.1`, then do the following.

- Use the `ENVOY_TRACING_CFG_FILE` environment variable to change your sampling rate. For more information, see Envoy configuration variables. Specify a custom tracing configuration and define local sampling rules. For more information, see Envoy X-Ray config.

- Custom tracing configuration for the `ENVOY_TRACING_CFG_FILE` environment variable example:

```
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
```

- For details on configuration for the sampling rule manifest in the `samplingRuleManifest` property, see Configuring the X-Ray SDK for Go.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

App Mesh security troubleshooting

This topic details common issues that you may experience with App Mesh security.

Unable to connect to a backend virtual service with a TLS client policy

Symptoms

When adding a TLS client policy to a virtual service backend in a virtual node, connectivity to that backend fails. When attempting to send traffic to the backend service, the requests fail with an `HTTP 503` response code and the error message: `upstream connect error or disconnect/reset before headers. reset reason: connection failure`.

Resolution

In order to determine the root cause of the issue, we recommend using the Envoy proxy process logs to help you diagnose the issue. For more information, see Enable Envoy debug logging in pre-production environments. Use the following list to determine the cause of the connection failure:

- Make sure connectivity to the backend is succeeding by ruling out the errors mentioned in Unable to connect to a virtual service backend.
- In the Envoy process logs, look for the following errors (logged at debug level).

  ```
  TLS error: 268435581:SSL routines:OPENSSL_internal:CERTIFICATE_VERIFY_FAILED
  ```

  This error is caused by one or more of the following reasons:

  - The certificate was not signed by one of the certificate authorities defined in the TLS client policy trust bundle.
  - The certificate is no longer valid (expired).
  - The Subject Alternative Name (SAN) does not match the requested DNS hostname.
  - Make sure that the certificate offered by the backend service is valid, that it is signed by one of the certificate authorities in your TLS client policies trust bundle, and that it meets the criteria defined in Transport Layer Security (TLS).
  - If the error you receive is like the one below, then that means the request is bypassing the Envoy proxy and reaching the application directly. When sending traffic, the stats on Envoy don't change indicating that Envoy isn't on the path to decrypt the traffic. In the proxy configuration of the virtual node, make sure the `AppPorts` contains the correct value that the application is listening on.

    ```
    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
    ```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support. If you believe that you've found a security vulnerability or have questions about App Mesh's security, then see the AWS vulnerability reporting guidelines.

Unable to connect to a backend virtual service when application is originating TLS

Symptoms

When originating a TLS session from an application, instead of from the Envoy proxy, connectivity to a backend virtual service fails.

Resolution

This is a known issue. For more information, see the Feature Request: TLS negotiation between the downstream application and upstream proxy GitHub issue. In App Mesh, TLS origination is currently supported from the Envoy proxy but not from the application. To use TLS origination support at the Envoy, disable TLS origination in the application. This allows the Envoy to read the outbound request headers and forward the request to the appropriate destination through a TLS session. For more information, see Transport Layer Security (TLS).

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support. If you believe that you've found a security vulnerability or have questions about App Mesh's security, then see the AWS vulnerability reporting guidelines.

Unable to assert that connectivity between Envoy proxies is using TLS

Symptoms

Your application has enabled TLS termination on the virtual node or virtual gateway listener, or TLS origination on the backend TLS client policy, but you are unable to assert that connectivity between Envoy proxies is occurring over a TLS-negotiated session.

Resolution

Steps defined in this resolution make use of the Envoy administration interface and Envoy statistics. For help configuring these, see Enable the Envoy proxy administration interface and Enable Envoy DogStatsD integration for metric offload. The following statistics examples use the administration interface for simplicity.

- For the Envoy proxy performing TLS termination:
  - Make sure that the TLS certificate has been bootstrapped in the Envoy configuration with the following command.

    ```
    curl http://my-app.default.svc.cluster.local:9901/certs
    ```

    In the returned output, you should see at least one entry under `certificates[].cert_chain` for the certificate used in TLS termination.

- Make sure that the number of successful inbound connections to the proxy's listener is exactly the same as the number of SSL handshakes plus the number of SSL sessions re-used, as shown by the following example commands and output.

```
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)
```

- For the Envoy proxy performing TLS origination:
  - Make sure that the TLS trust store has been bootstrapped in the Envoy configuration with the following command.

```
curl http://my-app.default.svc.cluster.local:9901/certs
```

    You should see at least one entry under `certificates[].ca_certs` for the certificates used in validating the backend's certificate during TLS origination.

  - Make sure that the number of successful outbound connections to the backend cluster is exactly the same as the number of SSL handshakes plus the number of SSL sessions re-used, as shown by the following example commands and output.

```
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)
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support. If you believe that you've found a security vulnerability or have questions about App Mesh's security, then see the AWS vulnerability reporting guidelines.

Troubleshooting TLS with Elastic Load Balancing

Symptoms

When attempting to configure an Application Load Balancer or Network Load Balancer to encrypt traffic to a virtual node, connectivity and load balancer health checks can fail.

Resolution

In order to determine the root cause of the issue, you need to check the following:

- For the Envoy proxy performing TLS termination, you need to rule out any misconfiguration. Follow the steps provided above in the Unable to connect to a backend virtual service with a TLS client policy.
- For the load balancer, you need to look at the configuration of the `TargetGroup`:
  - Make sure that the `TargetGroup` port matches the virtual node's defined listener port.
  - For Application Load Balancers that are originating TLS connections over HTTP to your service, make sure that the `TargetGroup` protocol is set to `HTTPS`. If health checks are being utilized, make sure that `HealthCheckProtocol` is set to `HTTPS`.
  - For Network Load Balancers that are originating TLS connections over TCP to your service, make sure that the `TargetGroup` protocol is set to `TLS`. If health checks are being utilized, make sure that `HealthCheckProtocol` is set to `TCP`.

> **Note**
>
> Any updates to `TargetGroup` require changing the `TargetGroup` name.

With this configured properly, your load balancer should provide a secure connection to your service using the certificate provided to the Envoy proxy.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support. If you believe that you've found a security vulnerability or have questions about App Mesh's security, then see the AWS vulnerability reporting guidelines.

App Mesh Kubernetes troubleshooting

This topic details common issues that you may experience when you use App Mesh with Kubernetes.

App Mesh resources created in Kubernetes cannot be found in App Mesh

Symptoms

You have created the App Mesh resources using the Kubernetes custom resource definition (CRD), but the resources that you created are not visible in App Mesh when you use the AWS Management Console or APIs.

Resolution

The likely cause is an error in the Kubernetes controller for App Mesh. For more information, see Troubleshooting on GitHub. Check the controller logs for any errors or warnings indicating that the controller could not create any resources.

```
kubectl logs -n appmesh-system -f \
    $(kubectl get pods -n appmesh-system -o name | grep controller)
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Pods are failing readiness and liveliness checks after Envoy sidecar is injected

Symptoms

Pods for your application were previously running successfully, but after the Envoy sidecar is injected into a pod, readiness and liveliness checks begin failing.

Resolution

Make sure that the Envoy container that was injected into the pod has bootstrapped with App Mesh's Envoy management service. You can verify any errors by referencing the error codes in Envoy disconnected from App Mesh Envoy management service with error text. You can use the following command to inspect Envoy logs for the relevant pod.

```
kubectl logs -n appmesh-system -f \
    $(kubectl get pods -n appmesh-system -o name | grep controller) \
    | grep "gRPC config stream closed"
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Pods not registering or deregistering as AWS Cloud Map instances

Symptoms

Your Kubernetes pods are not being registered in or de-registered from AWS Cloud Map as part of their life cycle. A pod may start successfully and be ready to serve traffic, but not receive any. When a pod is terminated, clients may still retain its IP address and attempt to send traffic to it, failing.

Resolution

This is a known issue. For more information, see the Pods don't get auto registered/deregistered in Kubernetes with AWS Cloud Map GitHub issue. Due to the relationship between pods, App Mesh virtual nodes, and AWS Cloud Map resources, the App Mesh controller for Kubernetes may become desynchronized and lose resources. For example, this can happen if a virtual node resource is deleted from Kubernetes before terminating its associated pods.

To mitigate this issue:

- Make sure that you are running the latest version of the App Mesh controller for Kubernetes.
- Make sure that the AWS Cloud Map `namespaceName` and `serviceName` are correct in your virtual node definition.
- Make sure that you delete any associated pods prior to deleting your virtual node definition. If you need help identifying which pods are associated with a virtual node, see Cannot determine where a pod for an App Mesh resource is running.
- If your issue persists, run the following command to inspect your controller logs for errors that may help reveal the underlying issue.

```
kubectl logs -n appmesh-system \
    $(kubectl get pods -n appmesh-system -o name | grep appmesh-controller)
```

- Consider using the following command to restart your controller pods. This may fix synchronization issues.

```
kubectl delete -n appmesh-system \
    $(kubectl get pods -n appmesh-system -o name | grep appmesh-controller)
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Cannot determine where a pod for an App Mesh resource is running

Symptoms

When you run App Mesh on a Kubernetes cluster, an operator cannot determine where a workload, or pod, is running for a given App Mesh resource.

Resolution

Kubernetes pod resources are annotated with the mesh and virtual node that they are associated to. You can query which pods are running for a given virtual node name with the following command.

```
kubectl get pods --all-namespaces -o json | \
    jq '.items[] | { metadata } | select(.metadata.annotations."appmesh.k8s.aws/
virtualNode" == "virtual-node-name")'
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Cannot determine what App Mesh resource a pod is running as

Symptoms

When running App Mesh on a Kubernetes cluster, an operator cannot determine what App Mesh resource a given pod is running as.

Resolution

Kubernetes pod resources are annotated with the mesh and virtual node that they are associated to. You can output the mesh and virtual node names by querying the pod directly using the following command.

```
kubectl get pod pod-name -n namespace -o json | \
    jq '{ "mesh": .metadata.annotations."appmesh.k8s.aws/mesh",
 "virtualNode": .metadata.annotations."appmesh.k8s.aws/virtualNode" }'
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

Client Envoys are not able to communicate with App Mesh Envoy Management Service with IMDSv1 disabled

Symptoms

When `IMDSv1` is disabled, client Envoys aren't able to communicate with the App Mesh control plane (Envoy Management Service). `IMDSv2` support is not available on App Mesh Envoy version before `v1.24.0.0-prod`.

Resolution

To resolve this issue, you can do one of these three things.

- Upgrade to App Mesh Envoy version `v1.24.0.0-prod` or later, which has `IMDSv2` support.
- Re-enable `IMDSv1` on the Instance where Envoy is running. For instructions on restoring `IMDSv1`, see Configure the instance metadata options.
- If your services are running on Amazon EKS, it is recommended to use IAM roles for service accounts (IRSA) for fetching credentials. For instructions to enable IRSA, see IAM roles for service accounts.

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

IRSA does not work on application container when App Mesh is enabled and Envoy is injected

Symptoms

When App Mesh is enabled on an Amazon EKS cluster with the help of the App Mesh controller for Amazon EKS, Envoy and `proxyinit` containers are injected into the application pod. The

application is not able to assume IRSA and instead assumes the node `role`. When we describe the pod details, we then see that either the `AWS_WEB_IDENTITY_TOKEN_FILE` or `AWS_ROLE_ARN` environment variable are not included in the application container.

Resolution

If either `AWS_WEB_IDENTITY_TOKEN_FILE` or `AWS_ROLE_ARN` environment variables are defined, then the webhook will skip the pod. Don't provide either of these variables and the webhook will take care of injecting them for you.

```
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
        }
```

If your issue is still not resolved, then consider opening a GitHub issue or contact AWS Support.

App Mesh Preview Channel

The App Mesh Preview Channel is a distinct variant of the App Mesh service provided in the `us-west-2` Region. The Preview Channel exposes upcoming features for you to try as they are developed. As you use features in the Preview Channel, you can provide feedback via GitHub to shape how the features behave. Once a feature has complete functionality in the Preview Channel, with all of the necessary integrations and checks completed, it will graduate to the production App Mesh service.

The AWS App Mesh Preview Channel is a *Beta Service* and all features are *previews*, as those terms are defined in the AWS Service Terms. Your participation in the Preview Channel is governed by your Agreement with AWS and the AWS Service Terms, in particular, the Universal and Beta Service Participation terms, and is confidential.

The following questions are frequently asked about the Preview Channel.

What is the Preview Channel?

The Preview Channel is a public service endpoint that allows you to try out and provide feedback on new service features before they are generally available. The service endpoint for the Preview Channel is separate from the standard production endpoint. You interact with the endpoint by using the AWS CLI, a service model file for the Preview Channel, and command input files for the AWS CLI. The Preview Channel, allows you to try new features without impacting your current production infrastructure. You're encouraged to provide feedback to the App Mesh team to help ensure that App Mesh meets customers' most important requirements. Your feedback on features while they're in the Preview Channel can help shape the features of App Mesh so that we can deliver the best possible service.

How is the Preview Channel different from production App Mesh?

The following table lists aspects of the App Mesh service that are different from the Preview Channel.

Aspect	App Mesh production service	App Mesh Preview Channel service

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

Though the example in the table for the App Mesh production service lists the `us-west-2` Region, the production service is available in most Regions. For a list of all Regions that the App Mesh production service is available in, see AWS App Mesh Endpoints and Quotas. However, the App Mesh Preview Channel service is available only in the `us-west-2` Region.

How can I use features in the Preview Channel?

1. Add the Preview Channel service model that includes the Preview Channel feature to the AWS CLI with the following command.

```
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. Create a JSON file that includes the feature, based on the JSON example and instructions provided in the AWS App Mesh User Guide for the feature.

3. Implement the feature with the appropriate AWS CLI command and command input file. For example, the following command creates a route with Preview Channel features using the *route.json* file.

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

4. Add `APPMESH_PREVIEW = 1` as a configuration variable for the Envoy container when adding it to your Amazon ECS task definitions, Kubernetes Pod specifications, or Amazon EC2 instances. This variable enables the Envoy container to communicate with the Preview Channel endpoints. For more information about adding configuration variables, see Updating services in Amazon ECS, Updating services in Kubernetes, and Updating services on Amazon EC2.

How do I provide feedback?

You can provide feedback directly on the App Mesh roadmap GitHub repo issue that is linked from the documentation about the feature.

How long do I have to provide feedback on a feature in the Preview Channel?

The feedback period will vary depending on the size and complexity of the feature being introduced. We intend to give a comment period of 14 days between release of a feature to the preview endpoint and release of the feature to production. The App Mesh team may extend the feedback period for specific features.

What level of support is provided for the Preview Channel?

While we encourage you to provide feedback and bug reports directly on the App Mesh GitHub roadmap issue, we understand that you may have sensitive data to share, or you may find an issue that you do not feel is safe to disclose publicly. For these issues, you can contact AWS Support or give feedback by emailing the App Mesh team directly.

Is my data secure on the Preview Channel endpoint?

Yes. The Preview Channel is given the same level of security as the standard production endpoint.

How long will my configuration be available?

You can work with a mesh in the Preview Channel for thirty days. Thirty days after a mesh is created, you can only list, read, or delete the mesh. If you attempt to create or update resources after thirty days, you'll receive a `BadRequest` exception explaining that the mesh is archived.

What tools can I use to work with the Preview Channel?

You can use the AWS CLI with a Preview Channel service model file and command input files. For more information about how to work with features, see How can I use features in the Preview Channel?. You cannot use AWS CLI command options, the AWS Management Console, SDKs, or AWS CloudFormation to work with Preview Channel features. You can use all tools however, once a feature is released to the production service.

Will there be forward compatibility of Preview Channel APIs?

No. APIs may change based on feedback.

Are Preview Channel features complete?

No. New API objects may not be fully integrated into the AWS Management Console, AWS CloudFormation, or AWS CloudTrail. As features solidify in the Preview Channel and near general availability, the integrations will eventually become available.

Is documentation available for Preview Channel features?

Yes. Documentation for Preview Channel features is included in the production documentation. For example, if features for the route resource are released to the Preview Channel, information about how to use the features would be in the existing route resource document. Preview Channel features are labeled as only available in the Preview Channel.

How will I know when new features are available in the Preview Channel?

When new features are introduced in the Preview Channel, an entry is added to the App Mesh Document History. You can review the page regularly or subscribe to the App Mesh Document

History RSS feed. Additionally, you can review the issues for the AWS App Mesh roadmap GitHub repo. A download link for the Preview Channel service model JSON file is added to an issue when it releases to the Preview Channel. For more information about how to use the model and feature, see How can I use features in the Preview Channel?.

How will I know when a feature has graduated to the production service?

The text in the App Mesh documentation noting that the feature is available only in the Preview Channel is removed, and an entry is added to the App Mesh Document History. You can review the page regularly or subscribe to the App Mesh Document History RSS feed.

App Mesh service quotas

AWS App Mesh has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. Service quotas are also referred to as limits. For more information, see What Is Service Quotas? in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of all of the App Mesh service quotas.

To view App Mesh service quotas using the AWS Management Console

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **AWS App Mesh**.

   In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.
4. To view additional information about a service quota, such as the description, choose the quota name.

To request a quota increase, see Requesting a Quota Increase in the *Service Quotas User Guide*.

To view App Mesh service quotas using the AWS CLI

Run the following command.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code appmesh \
    --output table
```

To work more with service quotas using the AWS CLI, see the Service Quotas AWS CLI Command Reference.

Document history for App Mesh

The following table describes the major updates and new features for the *AWS App Mesh User Guide*. We also update the documentation frequently to address the feedback that you send us.

Change	Description	Date
CloudTrail integration documentation updated	The documentation describing the App Mesh integration with CloudTrail to log API activity has been updated.	March 28, 2024
Updated policies	Updated `AWSServiceRoleForAppMesh` and `AWSAppMeshServiceRolePolicy` to allow for access to the AWS Cloud Map `DiscoverInstancesRevision` API.	October 12, 2023
VPC endpoint policy support for App Mesh	App Mesh now supports VPC endpoint policies.	May 11, 2023
Multiple listeners for App Mesh	App Mesh now supports multiple listeners.	August 18, 2022
IPv6 for App Mesh	App Mesh now supports IPv6.	May 18, 2022
CloudTrail logging support for App Mesh Envoy Management Service	App Mesh now supports CloudTrail logging support for App Mesh Envoy Management Service.	March 18, 2022
App Mesh Agent for Envoy	App Mesh now supports Agent for Envoy.	February 25, 2022
Multiple listeners for App Mesh	(App Mesh Preview Channel only) You can implement	November 23, 2021

	multiple listeners for App Mesh.	
ARM64 support for App Mesh	App Mesh now supports ARM64.	November 19, 2021
Metrics extension for App Mesh	You can implement metrics extensions for App Mesh.	October 29, 2021
Implement incoming traffic enhancements	You can implement host name and header match and rewrites for host name and path.	June 14, 2021
Implement mutual TLS authentication	You can implement mutual TLS authentication.	February 4, 2021
Region launch in af-south-1	App Mesh is now available in the af-south-1 Region.	January 22, 2021
Implement mutual TLS authentication	(App Mesh Preview Channel only) You can implement mutual TLS authentication.	November 23, 2020
Implement connection pooling for a virtual gateway listener	You can implement connection pooling for a virtual gateway listener.	November 5, 2020
Implement connection pooling and outlier detection for a virtual node listener	You can implement connection pooling and outlier detection for a virtual node listener.	November 5, 2020
Region launch in eu-south-1	App Mesh is now available in the eu-south-1 Region.	October 21, 2020

Implement connection pooling for a virtual gateway listener	(App Mesh Preview Channel only) You can implement connection pooling for a virtual gateway listener.	September 28, 2020
Implement connection pooling and outlier detection for a virtual node listener	(App Mesh Preview Channel only) You can implement connection pooling and outlier detection for a virtual node listener.	September 28, 2020
Create a virtual gateway and gateway route for mesh inbound	Enable resources that are outside of a mesh to communicate to resources that are inside of a mesh.	July 10, 2020
Create and manage App Mesh resources from within Kubernetes with the App Mesh controller for Kubernetes	You can create and manage App Mesh resources from within Kubernetes. The controller also automatically injects the Envoy proxy and init containers into pods that you deploy.	June 18, 2020
Add a timeout value to a virtual node listener and route	You can add a timeout value to a virtual node listener and route.	June 18, 2020
Add a timeout value to a virtual node listener	(App Mesh Preview Channel only) You can add a timeout value to a virtual node listener.	May 29, 2020
Create a virtual gateway for mesh inbound	(App Mesh Preview Channel only) Enable resources outside of a mesh to communicate to resources inside of a mesh.	April 8, 2020

TLS encryption	(App Mesh Preview Channel only) Use certificates from an AWS Private Certificate Authority or your own certificate authority to encrypt communication between virtual nodes using TLS.	January 17, 2020
Share a mesh with another account	(App Mesh Preview Channel only) You can share a mesh with another account. Resources created by any account can communicate with other resources in the mesh.	January 17, 2020
Add a timeout value to a route	(App Mesh Preview Channel only) You can add a timeout value to a route.	January 17, 2020
Create an App Mesh proxy on an AWS Outpost	You can create an App Mesh Envoy proxy on an AWS Outpost.	December 3, 2019
HTTP/2 and gRPC support for routes, virtual routers, and virtual nodes	You can route traffic that uses the HTTP/2 and gRPC protocols. You can also add a listener for these protocols to virtual nodes and virtual routers.	October 25, 2019

Retry policy	A retry policy enables clients to protect themselves from intermittent network failures or intermittent server-side failures. You can add retry logic to a route.	September 10, 2019
TLS encryption	(App Mesh Preview Channel only) Encrypt communication between virtual nodes using TLS.	September 6, 2019
HTTP header-based routing	Route traffic based on the presence and values of HTTP headers in a request.	August 15, 2019
Availability of the App Mesh Preview Channel	The App Mesh Preview Channel is a distinct variant of the App Mesh service. The Preview Channel exposes upcoming features for you to try as they are developed . As you use features in the Preview Channel, you can provide feedback via GitHub to shape how the features behave.	July 19, 2019

App Mesh Interface VPC Endpoints (AWS PrivateLink)	Improve the security posture of your VPC by configuri ng App Mesh to use an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLi nk, a technology that enables you to privately access App Mesh APIs by using private IP addresses. PrivateLink restricts all network traffic between your VPC and App Mesh to the Amazon network.	June 14, 2019
Added AWS Cloud Map as a virtual node service discovery method	You can specify DNS or AWS Cloud Map as a virtual node service discovery method. To use AWS Cloud Map for service discovery, your account must have the App Mesh service-linked role.	June 13, 2019
Create App Mesh resources automatically in Kubernetes	Create App Mesh resources and add the App Mesh sidecar container images to your Kubernetes deployments automatically when you create resources in Kubernete s.	June 11, 2019
App Mesh General Availability	The App Mesh service is now generally available for production use.	March 27, 2019

App Mesh API update	The App Mesh APIs were updated to improve usability . For more information, see [FEATURE] Add Listeners to Virtual Routers and [BUG] Routes to Target Virtual Nodes with Mismatched Ports Blackhole.	March 7, 2019
App Mesh initial release	Initial documentation for service public preview	November 28, 2018