AWS App Mesh

User Guide
# Table of Contents

**What Is AWS App Mesh?** ................................................................. 1
  - Adding App Mesh to an example application ........................................... 1
  - Components of App Mesh ........................................................................ 2
  - How to get started ..................................................................................... 3
  - Accessing App Mesh .................................................................................. 3

**Getting started** ................................................................................... 5
  - App Mesh and Amazon ECS ....................................................................... 5
    - Scenario .................................................................................................... 5
    - Prerequisites ............................................................................................ 6
    - Step 1: Create a mesh and virtual service ............................................... 6
    - Step 2: Create a virtual node .................................................................... 7
    - Step 3: Create a virtual router and route ............................................... 8
    - Step 4: Review and create ........................................................................ 9
    - Step 5: Create additional resources ....................................................... 10
    - Step 6: Update services .......................................................................... 13

**App Mesh and Kubernetes** .................................................................. 24
  - Prerequisites ............................................................................................ 25
  - Step 1: Install the integration components .............................................. 25
  - Step 2: Deploy App Mesh resources ....................................................... 27
  - Step 3: Create or update services ............................................................ 36
  - Step 4: Clean up ......................................................................................... 41

**App Mesh and Amazon EC2** ............................................................. 41
  - Scenario .................................................................................................... 5
  - Prerequisites ............................................................................................ 6
  - Step 1: Create a mesh and virtual service ............................................... 6
  - Step 2: Create a virtual node .................................................................... 7
  - Step 3: Create a virtual router and route ............................................... 8
  - Step 4: Review and create ........................................................................ 9
  - Step 5: Create additional resources ....................................................... 10
  - Step 6: Update services .......................................................................... 13

**App Mesh Roadmap** ............................................................................ 55

**App Mesh Examples** ............................................................................ 55

**Concepts** ............................................................................................ 56
  - Mesh ............................................................................................................ 56
  - Creating a service mesh .............................................................................. 56
  - Virtual services .......................................................................................... 56
  - Creating a virtual service ........................................................................... 56
  - Virtual gateways .......................................................................................... 57
  - Creating a virtual gateway ......................................................................... 57
  - Deploy virtual gateway .............................................................................. 58
  - Deleting a virtual gateway ........................................................................ 59
  - Gateway routes .......................................................................................... 60
  - Virtual nodes ............................................................................................. 62
  - Creating a virtual node .............................................................................. 62
  - Deleting a virtual node .............................................................................. 62
  - Virtual routers ............................................................................................ 66
  - Creating a virtual router ........................................................................... 66
  - Deleting a virtual router ............................................................................ 67
  - Routes ........................................................................................................ 67

**Best practices** ..................................................................................... 71
  - Instrument all routes with retries ............................................................ 71
  - Adjust deployment velocity ..................................................................... 71
  - Scale out before scale in .......................................................................... 72
  - Implement container health checks ......................................................... 72
Cannot determine what App Mesh resource a pod is running as ............................................. 140
Cannot determine where a pod for an App Mesh resource is running ..................................... 140
Pods not registering or deregistering as AWS Cloud Map instances ........................................ 139
Pods are failing readiness and liveness checks after Envoy sidecar is injected ................................. 138
App Mesh resources created in Kubernetes cannot be found in App Mesh ............................... 138
Unable to assert that connectivity between Envoy proxies is using TLS .................................. 136
Unable to connect to a backend virtual service when application is originating TLS .................. 136
Unable to connect to a backend virtual service with a TLS client policy .................................. 135
Unable to configure custom sampling rules for AWS X-Ray traces ......................................... 135
Unable to connect to an Amazon EFS filesystem ................................................................ 132
Connectivity succeeds to service, but the incoming request does not appear in access logs, traces, or metrics for Envoy ................................................................. 132
Connectivity fails and container health checks fail when scaling beyond 10 replicas for a virtual node ........................................................... 133
Requests fail with 503 when a virtual service backend horizontally scales out or in ................. 133
Envoy container crashes with segfault under increased load ......................................................... 133
Unable to see AWS X-Ray traces for my applications ............................................................... 134
Unable to see Envoy metrics for my applications in Amazon CloudWatch metrics ................... 134
Unable to configure custom sampling rules for AWS X-Ray traces ......................................... 135
Unable to connect to a backend virtual service with a TLS client policy ................................. 135
Unable to connect to a backend virtual service when application is originating TLS ................. 136
Unable to assert that connectivity between Envoy proxies is using TLS ................................ 136
Troubleshooting TLS with Elastic Load Balancing ................................................................. 137
App Mesh resources created in Kubernetes cannot be found in App Mesh ............................... 138
Pods are failing readiness and liveness checks after Envoy sidecar is injected ............................ 138
Pods not registering or deregistering as AWS Cloud Map instances ....................................... 139
Cannot determine where a pod for an App Mesh resource is running ..................................... 140
Cannot determine what App Mesh resource a pod is running as ............................................. 140
What Is AWS App Mesh?

AWS App Mesh is a service mesh that makes it easy to monitor and control services. 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 (p. 56).

- **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 (p. 56).

- **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 (p. 62).

- **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 (p. 66) and Routes (p. 67).

- **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 (p. 73).

## 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 (p. 5)
- Getting started with AWS App Mesh and Kubernetes (p. 24)
- Getting started with AWS App Mesh and Amazon EC2 (p. 41)
- App Mesh Roadmap (p. 55)
- App Mesh Examples (p. 55)

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 . 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 ensure 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 deprecate 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 have decided 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? (p. 1).*
- An existing understanding of Amazon ECS 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 (p. 56).* A virtual service is an abstraction of an actual service. For more information, see *Virtual services (p. 56).*

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**

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.
Step 2: Create a virtual node

A virtual node acts as a logical pointer to an actual service. For more information, see Virtual nodes (p. 62).

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:

   ```json
   {
     "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.

   ```sh
   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 (p. 66) and Routes (p. 67).

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 will listen for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported.
- A route named serviceB. It will route 100 percent of its traffic to the serviceB virtual node. You will change the weight in a later step once you have added 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 **Route configuration**, select serviceB and enter 100 for **Weight**.
6. To continue, choose **Next**.

AWS CLI

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

   ```json
   {
     "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.

   ```shell
   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:

```json
{
    "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.
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, since 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 will send 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 deprecate 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 servicea.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 drop-down 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:

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

   b. Create the virtual node.

   ```bash
   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:

   ```json
   {
     "meshName": "apps",
     "spec": {
       "backends": [
         {
           "virtualService": {
             "virtualServiceName": "serviceb.apps.local"
           }
         }
       ],
       "listeners": [
         {
           "portMapping": {
             "port": 80,
             "protocol": "http2"
           }
         }
       ],
       "serviceDiscovery": {
       
   ```
Step 5: Create additional resources

```
"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" : "/
                }
            }
        }
    }
}
```

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.

**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.

---

```
"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
```

---

**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
the Fargate launch type 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 launch 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 Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. From the navigation bar, choose the Region that contains your task definition.
3. In the navigation pane, choose Task Definitions.
4. 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.
5. 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 and ap-east-1, eu-south-1. You can replace region-code with any Region other than me-south-1 and ap-east-1, eu-south-1.

         | Region           | Envoy Image                                           |
         |------------------|-------------------------------------------------------|
         | me-south-1       | 772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod |
         | ap-east-1        | 856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod |
         | eu-south-1       | 422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod |

      - 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**, ensure that `awsvpc` is selected. To learn more about the `awsvpc` network mode, see [Task Networking with the awsvpc Network Mode](#).

6. Choose **Create**.

7. 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 ingress port for the Envoy proxy container. Set this value to **15000**.

**ProxyEgressPort**

This is the egress port for the Envoy proxy container. Set this value to **15001**.

**AppPorts**

Specify any ingress 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 egress 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 ensure that 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 and ap-east-1, eu-south-1. You can replace `region-code` with any Region other than me-south-1 and ap-east-1, eu-south-1.

840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod
Step 6: Update services

- me-south-1 Region:
  
  772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod

- ap-east-1 Region:

  856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod

- eu-south-1 Region:

  422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod

**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](https://github.com/envoyproxy/envoy). 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.
- 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 (p. 73).
- 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 (p. 73).

The following code shows an Envoy container definition example.

```json
{
  "name": "envoy",
  "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod",
  "essential": true,
  "environment": [{
    "name": "APPMESH_RESOURCE_ARN",
  }],
  "healthCheck": {
    "command": [
      "CMD-SHELL",
      "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
    ],
    "startPeriod": 10,
    "interval": 5,
    "timeout": 2,
    "retries": 3
}
```
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 APPMESHRESOURCEARN 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 APPMESHRESOURCEARN when Envoy is referring to itself in metrics and traces. You can override this behavior by setting the APPMESHRESOURCECLUSTER 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.

Example JSON for Amazon ECS task definition - Fargate launch type

```json
{
  "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": [
        {
          "name": "proxyPort",
          "containerPort": "15000"
        }
      ]
    }
  ]
}
```
Example JSON for Amazon ECS task definition with AWS X-Ray - Fargate launch type

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" : "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:v1.15.1.0-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,
Example JSON for Amazon ECS task definition - EC2 launch type

```json
{
  "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": "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"
}
```
Step 6: Update services

Example JSON for Amazon ECS task definition with AWS X-Ray - EC2 launch type

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.

```json
{
  "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:v1.15.1.0-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 "$LIVE"
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 resources. 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 ingress and egress 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? (p. 1).*
- An existing understanding of Kubernetes concepts. For more information, see *What is Kubernetes* in the Kubernetes documentation.
- An existing Kubernetes cluster running version 1.14 or later. 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*.

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 will check to see whether a pre-release version is installed on your cluster.

   ```bash
   ./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.

   ```bash
   helm repo add eks https://aws.github.io/eks-charts
   ```
3. Install the App Mesh Kubernetes custom resource definitions (CRD).

   ```
   ```

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 \  
   --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.

   ```
   helm upgrade -i appmesh-controller eks/appmesh-controller
   ```
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
```

Important
If your cluster is in the me-south-1 or ap-east-1 Regions, then you need to add the following option to the previous command:

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

Replace `account-id` and `region-code` with one of the appropriate sets of values.

- 772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod
- 856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-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 v1.0.0 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

b. Create the namespace.

```bash
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 will be 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.

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

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

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

Output

<table>
<thead>
<tr>
<th>Name: my-mesh</th>
<th>Namespace: &lt;none&gt;</th>
<th>Labels:</th>
<th>Annotations: kubectl.kubernetes.io/last-applied-configuration:</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Version:  appmesh.k8s.aws/v1beta2</td>
<td>Kind: Mesh</td>
<td>Metadata:</td>
<td></td>
</tr>
<tr>
<td>Creation Timestamp: 2020-06-17T14:51:37Z</td>
<td>Finalizers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generation: 1</td>
<td>Resource Version: 6295</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self Link: /apis/appmesh.k8s.aws/v1beta2/meshes/my-mesh</td>
<td>UID: 11a11b1-c11d-1e1f-gh1i-j1k1l11m711</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spec:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aws Name: my-mesh</td>
<td>Namespace Selector:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Match Labels:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mesh: my-mesh</td>
<td>Status:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditions:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Last Transition Time: 2020-06-17T14:51:37Z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Status:                True
Type:                  MeshActive
Observed Generation:     1
Events:                    <none>

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

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

Output

```json
{
  "mesh": {
    "meshName": "my-mesh",
    "metadata": {
      "createdAt": "2020-06-17T09:51:37.920000-05:00",
      "lastUpdatedAt": "2020-06-17T09:51:37.920000-05:00",
      "meshOwner": "111122223333",
      "resourceOwner": "111122223333",
      "uid": "11111111-1111-1111-1111-111111111111",
      "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 will be 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.

```yaml
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" (p. 62). To see all available settings for a virtual node that you can set in the preceding spec, run the following command.
Step 2: Deploy App Mesh resources

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-app-1"},..."...
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
  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-app-1. 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": {
      "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-gh1j-j11k1111m711",
      "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 will be 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 will create 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" (p. 66) and the section called "Routes" (p. 67). 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
```
Step 2: Deploy App Mesh resources

```yaml
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 any of the following command.

```bash
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 any of the following command.

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

b. Deploy the virtual router.

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

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

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

Abbreviated output

Name:     my-service-a-virtual-router
Namespace: my-app-1
Labels:   <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
  "apiVersion": "appmesh.k8s.aws/v1beta2",
  "kind": "VirtualRouter",
  "metadata": {
    "annotations": {},
    "name": "my-service-a-virtual-router",
  }
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-cl1d-le1f-gh11-j11k11111m711
  Routes:
    Http Route:
      Action:
        Weighted Targets:
Virtual Node Ref:
Name: my-service-a  
Weight: 1
Match:
Prefix: / 
Name: my-service-a-route
Status:
Conditions:
Last Transition Time: 2020-06-17T15:14:01Z
Status: True
Type: VirtualRouterActive
Observed Generation: 1
Route ARNs:
virtualRouter/my-service-a-virtual-router_my-apps/route/my-service-a-route
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-app-1 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.

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

Output
```json
{
  "virtualRouter": {
    "meshName": "my-mesh",
    "metadata": {
      "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-j11k11l1m711",
      "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": {
         "createdAt": "2020-06-17T10:14:01.577000-05:00",
         "lastUpdatedAt": "2020-06-17T10:14:01.577000-05:00",
         "meshOwner": "111122223333",
         "resourceOwner": "111122223333",
         "uid": "111a11b1-111d-1e11f-gh11-j11k1111m711",
         "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 will be 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” (p. 36). For more information, see the section called “Virtual services” (p. 56).
Step 2: Deploy App Mesh resources

```
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

<table>
<thead>
<tr>
<th>Name: my-service-a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace: my-app-1</td>
</tr>
<tr>
<td>Labels: &lt;none&gt;</td>
</tr>
<tr>
<td>Annotations: kubectl.kubernetes.io/last-applied-configuration:</td>
</tr>
<tr>
<td>API Version: appmesh.k8s.aws/v1beta2</td>
</tr>
<tr>
<td>Kind: VirtualService</td>
</tr>
<tr>
<td>Metadata:</td>
</tr>
<tr>
<td>Creation Timestamp: 2020-06-17T15:48:40Z</td>
</tr>
<tr>
<td>Finalizers:</td>
</tr>
<tr>
<td>finalizers.appmesh.k8s.aws/aws-appmesh-resources</td>
</tr>
<tr>
<td>Generation: 1</td>
</tr>
<tr>
<td>Resource Version: 13598</td>
</tr>
<tr>
<td>Self Link: /apis/appmesh.k8s.aws/v1beta2/namespaces/my-apps/virtualservices/my-service-a</td>
</tr>
<tr>
<td>UID: 111a11b1-c11d-1e1f-gh1i-j11k11ll1m71</td>
</tr>
<tr>
<td>Spec:</td>
</tr>
<tr>
<td>Aws Name: my-service-a.my-apps.svc.cluster.local</td>
</tr>
<tr>
<td>Mesh Ref:</td>
</tr>
<tr>
<td>Name: my-mesh</td>
</tr>
<tr>
<td>UID: 111a11b1-c11d-1e1f-gh1i-j11k11ll1m71</td>
</tr>
<tr>
<td>Provider:</td>
</tr>
<tr>
<td>Virtual Router:</td>
</tr>
<tr>
<td>Virtual Router Ref:</td>
</tr>
<tr>
<td>Name: my-service-a-virtual-router</td>
</tr>
<tr>
<td>Status:</td>
</tr>
<tr>
<td>Conditions:</td>
</tr>
<tr>
<td>Last Transition Time: 2020-06-17T15:48:40Z</td>
</tr>
<tr>
<td>Status: True</td>
</tr>
<tr>
<td>Type: VirtualServiceActive</td>
</tr>
<tr>
<td>Observed Generation: 1</td>
</tr>
</tbody>
</table>
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.

   ```json
   {
   }
   ```

   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": {
         "createdAt": "2020-06-17T10:48:40.182000-05:00",
         "lastUpdatedAt": "2020-06-17T10:48:40.182000-05:00",
         "meshOwner": "111122223333",
         "resourceOwner": "111122223333",
         "uid": "111a11b1-clld-1e1f-gh11-j1lkjllm711",
         "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" (p. 57) and the section called "Gateway routes" (p. 60). For a walkthrough of deploying these resources with the controller, see Configuring Ingress 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.

   ```json
   {
   }
   ```
Step 3: Create or update services

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” (p. 27), and 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
```
Step 3: Create or update services

```yaml
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” (p. 27), 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` (p. 73). For a sample manifest, 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>my-service-a-54776556f6</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>10s</td>
</tr>
<tr>
<td>my-service-a-54776556f6-w2kf</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>18s</td>
</tr>
<tr>
<td>my-service-a-54776556f6-zw5kt</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>26s</td>
</tr>
</tbody>
</table>

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

```
kubectl -n my-apps describe pod my-service-a-54776556f6-2cd9
```
## Abbreviated output

<table>
<thead>
<tr>
<th>Name:</th>
<th>my-service-a-54776556f6-2cxd9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace:</td>
<td>my-app-1</td>
</tr>
<tr>
<td>Priority:</td>
<td>0</td>
</tr>
<tr>
<td>Node:</td>
<td>ip-192-168-44-157.us-west-2.compute.internal/192.168.44.157</td>
</tr>
<tr>
<td>Labels:</td>
<td>app=nginx</td>
</tr>
<tr>
<td>Annotations:</td>
<td>kubernetes.io/psp: eks.privileged</td>
</tr>
<tr>
<td>Status:</td>
<td>Running</td>
</tr>
<tr>
<td>IP:</td>
<td>192.168.57.134</td>
</tr>
<tr>
<td>Controlled By:</td>
<td>ReplicaSet/my-service-a-54776556f6</td>
</tr>
</tbody>
</table>

### Init Containers:
- **proxyinit**
  - Container ID: docker://
    - e0c4810d58c21ae0b6e40f6119d25b08f029094d0e01c9411c6cf2a32d77a59
  - 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_EVOY_INGRESS_PORT: 15000
    - APPMESH_EVOY_EGRESS_PORT: 15001
    - APPMESH_APP_PORTS: 80
    - APPMESH_EGRRESS_IGNORED_IP: 169.254.169.254
    - 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://
    - be6359dc6cd3df18a1c87df7b57c2093e1f9db17d5b3a77f22585ce3bcab137a
  - 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

---

39
AWS App Mesh User Guide

Step 3: Create or update services

AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/serviceaccount/token

envoy:
  Container ID: docker://905b55cbf33ef3b3debc51cb448401d2de2e7c2dbfe6a9754a2c49dd55a216b6
  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-lamserviceaccount-my-a-Role1-MMNCVWB6PLON

AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/serviceaccount/token

Events:

<table>
<thead>
<tr>
<th>Type</th>
<th>Reason</th>
<th>Age</th>
<th>From</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>Created container proxyinit</td>
<td>21s</td>
<td>kubelet, ip-192-168-44-157.us-west-2.compute.internal</td>
<td>Created container proxyinit</td>
</tr>
<tr>
<td>Normal</td>
<td>Started container proxyinit</td>
<td>21s</td>
<td>kubelet, ip-192-168-44-157.us-west-2.compute.internal</td>
<td>Started container proxyinit</td>
</tr>
<tr>
<td>Normal</td>
<td>Pulling</td>
<td>20s</td>
<td>kubelet, ip-192-168-44-157.us-west-2.compute.internal</td>
<td>Pulling image &quot;nginx:1.19.0&quot;</td>
</tr>
<tr>
<td>Normal</td>
<td>Pulled</td>
<td>16s</td>
<td>kubelet, ip-192-168-44-157.us-west-2.compute.internal</td>
<td>Successfully pulled image &quot;nginx:1.19.0&quot;</td>
</tr>
<tr>
<td>Normal</td>
<td>Created container nginx</td>
<td>15s</td>
<td>kubelet, ip-192-168-44-157.us-west-2.compute.internal</td>
<td>Created container nginx</td>
</tr>
<tr>
<td>Normal</td>
<td>Started container nginx</td>
<td>15s</td>
<td>kubelet, ip-192-168-44-157.us-west-2.compute.internal</td>
<td>Started container nginx</td>
</tr>
<tr>
<td>Normal</td>
<td>Created container envoy</td>
<td>7s</td>
<td>kubelet, ip-192-168-44-157.us-west-2.compute.internal</td>
<td>Created container envoy</td>
</tr>
<tr>
<td>Normal</td>
<td>Started container envoy</td>
<td>7s</td>
<td>kubelet, ip-192-168-44-157.us-west-2.compute.internal</td>
<td>Started container envoy</td>
</tr>
</tbody>
</table>
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.

## 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.

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

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

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

Delete the mesh.

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

(Optional) You can remove the Kubernetes integration components.

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

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

```bash
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 ensure 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 deprecate `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 have decided 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 (p. 56). A virtual service is an abstraction of an actual service. For more information, see Virtual services (p. 56).

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**

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

---

42
Step 2: Create a virtual node

A virtual node acts as a logical pointer to an actual service. For more information, see Virtual nodes (p. 62).

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:

```json
{
    "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.
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 (p. 66) and Routes (p. 67).

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 will listen for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported.

- A route named serviceB. It will route 100 percent of its traffic to the serviceB virtual node. You will change the weight in a later step once you have added 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 Route configuration, select serviceB and enter 100 for Weight.
6. To continue, choose Next.

**AWS CLI**

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

   ```json
   {
     "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.
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.

```
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, since 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 will send 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 deprecate 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 servicea.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 drop-down 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-serviceb.json` with the following contents:

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

   b. Create the virtual node.

   ```bash
   aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-serviceb2.json
   ```

2. Create the serviceA virtual node.

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

   ```json
   {
       "meshName" : "apps",
       "spec" : {
           "backends" : [
               {
                   "virtualService" : {
                       "virtualServiceName" : "serviceb.apps.local"
                   }
               }
           ],
           "listeners" : [
               {
                   "portMapping" : {
                       "port" : 80,
                       "protocol" : "http2"
                   }
               }
           ],
           "serviceDiscovery" : {
               "dns" : {
                   "hostname" : "serviceA.apps.local"
               }
           },
           "virtualNodeName" : "serviceA"
       }
   }
   ```
Step 5: Create additional resources

b. Create the virtual node.

```bash
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:

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

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

```bash
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:

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

48
Step 6: Update services

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.

Mesh summary

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 (p. 98).
- 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.

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

   b. Create an IAM role with the following command.

   ```bash
   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” (p. 42) and `serviceB` is the name of the virtual node that you created in the section called “Step 2: Create a virtual node” (p. 43). Replace `111122223333` with your account ID and `us-west-2` with the region that you created your mesh in.

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "appmesh:StreamAggregatedResources",
         "Resource": [
         ]
       }
     ]
   }
   ```

   b. Create the policy with the following command.

   ```bash
   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.

   ```bash
   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.

   ```bash
   aws iam attach-role-policy --policy-name 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 and ap-east-1, eu-south-1. You can replace `us-west-2` with any supported Region except me-south-1 and ap-east-1, eu-south-1.
   ```bash
   $ 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
     ```bash
     $ 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
     ```bash
     $ 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 and ap-east-1, eu-south-1. You can replace `region-code` with any supported Region except the me-south-1 and ap-east-1, eu-south-1 Regions. You can replace `1337` with any value between 0 and 2147483647.
     ```bash
     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:v1.15.1.0-prod
     ```
   - me-south-1 Region. You can replace `1337` with any value between 0 and 2147483647.
     ```bash
     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:v1.15.1.0-prod
     ```
   - ap-east-1 Region. You can replace `1337` with any value between 0 and 2147483647.
     ```bash
     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:v1.15.1.0-prod
     ```
sudo docker run --detach --env APPMESH_RESOURCE_ARN=mesh/apps/virtualNode/serviceB -u 1337 --network host 85666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod

**Note**
The `APPMESH_RESOURCE_ARN` property requires version 1.15.0 or later of the Envoy image. For more information, see *Envoy* (p. 73).

**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 ingress. 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.

```bash
#!/bin/bash -e

# Start of configurable options
#

APPMESH_START_ENABLED="0"
APPMESH_IGNORE_UID="1337"
APPMESH_APP_PORTS="8000"
APPMESH_ENVVOY_EGRESS_PORT="15001"
APPMESH_ENVVOY_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"

# Egress traffic from the processes owned by the following UID/GID will be ignored.
if [ -z "" ] && [ -z "" ]; 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 its egress traffic."
  exit 1
fi

# Port numbers Application and Envoy are listening on.
if [ -z "" ] || [ -z "" ]; then
  echo "All of APPMESH_ENVVOY_INGRESS_PORT, APPMESH_ENVVOY_EGRESS_PORT and APPMESH_APP_PORTS variables must be set."
  echo "If any one of them is not set we will not be able to route either ingress, egress, or both directions."
  exit 1
fi

# Comma separated list of ports for which egress traffic will be ignored, we always refuse to route SSH traffic.
if [ -z "" ]; then
  APPMESH_EGRESS_IGNORED_PORTS="22"
else
  APPMESH_EGRESS_IGNORED_PORTS=""APPMESH_EGRESS_IGNORED_PORTS,22"
fi
```
```bash
# End of configurable options
#
APPMESH_LOCAL_ROUTE_TABLE_ID="100"
APPMESH_PACKET_MARK="0x1e7700ce"

function initialize() {
    echo "=== Initializing ==="
    iptables -t mangle -N APPMESH_INGRESS
    iptables -t nat -N APPMESH_INGRESS
    iptables -t nat -N APPMESH_EGRESS
    ip rule add fwmark "$APPMESH_PACKET_MARK" lookup $APPMESH_LOCAL_ROUTE_TABLE_ID
    ip route add local default dev lo table $APPMESH_LOCAL_ROUTE_TABLE_ID
}

function enable_egress_routing() {
    # Stuff to ignore
    [ ! -z "$APPMESH_IGNORE_UID" ] && \
        iptables -t nat -A APPMESH_EGRESS \n        -m owner --uid-owner $APPMESH_IGNORE_UID \n        -j RETURN
    [ ! -z "$APPMESH_IGNORE_GID" ] && \
        iptables -t nat -A APPMESH_EGRESS \n        -m owner --gid-owner $APPMESH_IGNORE_GID \n        -j RETURN
    [ ! -z "$APPMESH_EGRESS_IGNORED_PORTS" ] && \
        iptables -t nat -A APPMESH_EGRESS \n        -p tcp \n        -m multiport --dports "$APPMESH_EGRESS_IGNORED_PORTS" \n        -j RETURN
    [ ! -z "$APPMESH_EGRESS_IGNORED_IP" ] && \
        iptables -t nat -A APPMESH_EGRESS \n        -p tcp \n        -d "$APPMESH_EGRESS_IGNORED_IP" \n        -j RETURN
    # Redirect everything that is not ignored
    iptables -t nat -A APPMESH_EGRESS \n    -p tcp \n    -j REDIRECT --to $APPMESH_ENVOY_EGRESS_PORT
    # Apply APPMESH_EGRESS chain to non local traffic
    iptables -t nat -A OUTPUT \n    -p tcp \n    -m addrtype ! --dst-type LOCAL \n    -j APPMESH_EGRESS
}

function enable_ingress_redirect_routing() {
    # Route everything arriving at the application port to Envoy
    iptables -t nat -A APPMESH_INGRESS \n    -p tcp \n    -m multiport --dports "$APPMESH_APP_PORTS" \n    -j REDIRECT --to-port "$APPMESH_ENVOY_INGRESS_PORT"
    # Apply AppMesh ingress chain to everything non-local
    iptables -t nat -A PREROUTING \n    -p tcp \n    -m addrtype ! --src-type LOCAL \n    -j APPMESH_INGRESS
}
```

function enable_routing() {
    echo "=== Enabling routing ==="
    enable_egress_routing
    enable_ingress_redirect_routing
}

function disable_routing() {
    echo "=== Disabling routing ==="
    iptables -F
    iptables -F -t nat
    iptables -F -t mangle
}

function dump_status() {
    echo "=== Routing rules ==="
    ip rule
    echo "=== AppMesh routing table ==="
    ip route list table $APPMESH_LOCAL_ROUTE_TABLE_ID
    echo "=== iptables FORWARD table ==="
    iptables -L -v -n
    echo "=== iptables NAT table ==="
    iptables -t nat -L -v -n
    echo "=== iptables MANGLE table ==="
    iptables -t mangle -L -v -n
}

function main_loop() {
    echo "=== Entering main loop ==="
    while read -p '> ' cmd; do
        case "$cmd" in
            "quit")
                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.

## 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 repo:

[App Mesh Examples](#)
App Mesh Concepts

App Mesh is composed of the following concepts.

- Service Meshes (p. 56)
- Virtual services (p. 56)
- Virtual gateways (p. 57)
- Virtual nodes (p. 62)
- Virtual routers (p. 66)

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

To create a service mesh using the AWS CLI version 1.18.116 or higher, see the example in the AWS CLI reference for the create-mesh command.

To create a service mesh using the AWS Management Console

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 will only forward traffic between each other. If you allow external traffic, the proxies in the mesh will also forward TCP traffic directly to services that aren't deployed with a proxy that is defined in the mesh.
5. Choose Create mesh to finish.
6. (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 (p. 112).

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

To create a virtual service using the AWS CLI version 1.18.116 or higher, see the example in the AWS CLI reference for the create-virtual-service command.
To create a virtual service using the AWS Management Console

2. Choose the mesh that you want to create the virtual service in. All of the meshes that you own and that have been shared (p. 112) 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 and so that 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 that the names resolve to.
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 (p. 77).
   - 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.

**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. For more details, refer to the Creating a gateway route section from Gateway Routes.
Creating a virtual gateway

To create a virtual gateway using the AWS Management Console

To create a virtual gateway using the AWS CLI version 1.18.116 or higher, see the AWS CLI reference for the create-virtual-gateway command.

2. Choose the mesh that you want to create the virtual gateway in. All of the meshes that you own and that have been shared (p. 112) 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 that you want to enforce TLS communication with virtual services on.
   c. For Certificate discovery 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 (p. 80).
      • AWS Certificate Manager Private Certificate Authority hosting – Select one or more existing Certificates.
      • Local file hosting – Specify the path to the Certificate chain file on the file system where the Envoy is deployed.
   d. Specify a Port and Protocol for the Listener. Each virtual gateway can have only one port and protocol specified. If you need the virtual gateway to route traffic over multiple ports and protocols, then you must create a virtual gateway for each port or protocol.
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 that Envoy will listen for traffic on. The http listener permits connection transition to websockets.
   b. (Optional) Enable connection pool

   Connection pooling limits the number of connections that an Envoy can concurrently establish with all the hosts in the upstream 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 gateway listener. App Mesh sets the client-side connection pool settings to infinite by default, simplifying mesh configuration.
Deploy virtual gateway

Deploy an Amazon ECS or Kubernetes service that contains only the Envoy container (p. 73). You can also deploy the Envoy container on an Amazon EC2 instance. For more information see Getting started with App Mesh and Amazon ECS or Tutorial: Configure App Mesh integration with 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.

Enable proxy authorization for Envoy. For more information, see Envoy Proxy authorization (p. 98).

Deleting a virtual gateway

To delete a virtual gateway using the AWS CLI version 1.18.116 or higher, see the AWS CLI reference for the `delete-virtual-gateway` command.

To delete a virtual gateway using the AWS Management Console

2. Choose the mesh that you want to delete a virtual gateway from. All of the meshes that you own and that have been shared (p. 112) 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.

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

To create a gateway route using the AWS CLI version 1.18.116 or later, see the examples in the AWS CLI reference for the `create-gateway-route` command.

To create a gateway route using the AWS Management Console

2. Choose the mesh that you want to create the gateway route in. All of the meshes that you own and that have been shared (p. 112) with you are listed.
3. Choose Virtual gateways in the left navigation.
4. Choose the virtual gateway that you want to associate a new gateway route with. If none are listed, then you need to create a virtual gateway (p. 58) first. You can only create a gateway route for a virtual gateway that your account is listed as the Resource owner of.
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 one of the following protocols.
   - http or http2 – Specify a Match prefix. 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. For example, if
Gateway routes

you have configured the **Match prefix** to be `/`, when a request is received for `www.example.com/chapter/page`, it would match the prefix, because it has a `/` after the host name. The request would be rewritten to the name of a virtual service, such as `myservicea.svc.cluster.local/chapter/page`.

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. If the **Match prefix** in the gateway route example were `/chapter`, rather than `/`, then:

- A request to `www.example.com/chapter/page` would be re-written to `myservicea.svc.cluster.local/page`. Since `chapter` was the matched prefix, it would be removed in the re-written request.

- A request to `www.example.com/chapter1` would be rewritten to `myservicea.svc.cluster.local/1`. Again, `chapter` was removed, since it was the matched prefix, but `/1` would be added to the rewritten request from the original request.

Provided the virtual service's virtual router has a route defined with a prefix of `/1` or `/page`, the request would be routed to a virtual node. If the virtual service's virtual router doesn't have such a route however, the request would be dropped. This behavior enables you to minimize the number of gateway routes that are needed, but does require you to ensure that you have routes for a virtual service's virtual router for each of the expected prefixes that may be requested. Learn more about Virtual services (p. 56), Virtual routers (p. 66), and Routes (p. 67).

**Important**

- You can't specify either `/aws-appmesh*` or `/aws-app-mesh*` for **Match prefix**. 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.

**grpc** – Specify your gRPC **Service name**. Your gRPC service acts as an API for your application and is defined with ProtoBuf. For more information about gRPC, see grpc.io.

**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.

8. Select an existing **Virtual service name**. If none are listed, then you need to create a virtual service (p. 56) first.

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

## Deleting a gateway route

To delete a route using the AWS CLI version 1.18.116 or higher, see the AWS CLI reference for the `delete-gateway-route` command.

**To delete a gateway route using the AWS Management Console**


2. Choose the mesh that you want to delete a gateway route from. All of the meshes that you own and that have been shared (p. 112) with you are listed.

3. Choose **Virtual gateways** in the left navigation.

4. Choose the virtual gateway that you want to delete a gateway route from.

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 for Resource owner.

6. In the confirmation box, type `delete` and then select **Delete**.
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 (p. 73).

**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

To create a virtual node using the AWS CLI version 1.18.116 or higher, see the example in the AWS CLI reference for the create-virtual-node command.

**To create a virtual node using the AWS Management Console**

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 (p. 112) with you are listed.
3. Choose Virtual nodes in the left navigation.
4. Choose Create virtual node.
5. Specify settings for your virtual node.

   a. **Virtual node configuration**

      - For Virtual node name, enter a name for your virtual node.
      - 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. If the DNS server returns multiple IP addresses, then the Envoy proxy chooses one of the addresses using the Logical DNS service discovery type.

         - **AWS Cloud Map** – Specify an existing Service name and 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 AWS_ServiceRoleForAppMesh service-linked role (p. 96). For more information about AWS Cloud Map, see the AWS Cloud Map Developer Guide. AWS Cloud Map is not available in the eu-south-1 Region.

         - **None** – Select if your virtual node doesn't expect any inbound traffic.

      - (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 (p. 67). 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 (p. 98) 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 **Certificate discovery 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 (p. 80).
  - **AWS Certificate Manager 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.
  - **Local file hosting** – Specify the path to the **Certificate chain** file on the file system where the 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) **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 set **Client policy defaults**, but want to override them with 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 **Certificate discovery 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 (p. 80).
  - **AWS Certificate Manager Private Certificate Authority** hosting – Select one or more existing **Certificates**.
  - **Local file hosting** – Specify the path to the **Certificate chain** file on the file system where the Envoy is deployed.

To add additional backends, select **Add backend**.

- (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.

b. **Listener configuration**

- If your virtual node expects ingress traffic, specify a **Port** and **Protocol** for the **Listener**. The **http** listener permits connection transition to websockets.

- **(Optional) Enable connection pool**

  Connection pooling limits the number of connections that an Envoy can concurrently establish with all the hosts in the upstream 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.

- For **Maximum requests**, specify maximum number of parallel requests that can occur to the upstream cluster.

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

- **(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 should use AWS Cloud Map. Otherwise if using DNS, the Envoy proxy would only elect a single IP address for routing to the upstream service, nullifying 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.

- **(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.

  - **(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 (p. 80).
      - **AWS Certificate Manager hosting** – Select an existing **Certificate**.
      - **Local file hosting** – Specify the path to the **Certificate chain** and **Private key** files on the file system where the Envoy proxy is deployed.
  
  - **(Optional) Timeouts**.
    - **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. If increasing the timeout, make sure that the timeout specified for any route that is used for a **Service backend** is also greater than 15 seconds. For more information, see Routes (p. 67).
    - **Idle duration** – You can specify an idle duration for any listener protocol.

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

### Deleting a virtual node

To delete a virtual node using the AWS CLI, use the `aws appmesh delete-virtual-node` command. For an example of deleting a virtual node using the AWS CLI, see delete-virtual-node.

**Note**

You can't delete a virtual node if it is specified as a target in any **route** (p. 67) or as a provider in any **virtual service** (p. 56).

**To delete a virtual node using the AWS Management Console**

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** (p. 112) 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**.

**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**

To create a virtual router using the AWS CLI version 1.18.116 or higher, see the example in the AWS CLI reference for the `create-virtual-router` command.

**To create a virtual router using the AWS Management Console**

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 (p. 112) 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. For **Listener**, specify a **Port** and **Protocol** for your virtual router. The **http** listener permits connection transition to websockets.
7. Choose **Create virtual router** to finish.
Deleting a virtual router

To delete a virtual router using the AWS CLI, use the `aws appmesh delete-virtual-router` command. For an example of deleting a virtual router using the AWS CLI, see `delete-virtual-router`.

**Note**
you cannot delete a virtual router if it has any routes (p. 67) or if it is specified as a provider for any virtual service (p. 56).

To delete a virtual router using the AWS Management Console

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 (p. 112) 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`.
5. 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.
6. In the confirmation box, type `delete` and then select `Delete`.

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

To create a route using the AWS CLI version 1.18.116 or later, see the examples in the AWS CLI reference for the `create-route` command.

To create a route using the AWS Management Console

2. Choose the mesh that you want to create the route in. All of the meshes that you own and that have been shared (p. 112) 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 (p. 66) one 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. For `Virtual node name`, 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 (p. 62) one first.
10. (Optional) Choose `Additional configuration`. From the following protocols, choose the protocol that you selected for `Route type`, specify settings in the console as desired, and then select `Create route`.

---

**AWS App Mesh User Guide**

Deleting a virtual router

To delete a virtual router using the AWS CLI, use the `aws appmesh delete-virtual-router` command. For an example of deleting a virtual router using the AWS CLI, see `delete-virtual-router`.

**Note**
you cannot delete a virtual router if it has any routes (p. 67) or if it is specified as a provider for any virtual service (p. 56).

To delete a virtual router using the AWS Management Console

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 (p. 112) 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`.
5. In the confirmation box, type `delete` and then select `Delete`.

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

To create a route using the AWS CLI version 1.18.116 or later, see the examples in the AWS CLI reference for the `create-route` command.

To create a route using the AWS Management Console

2. Choose the mesh that you want to create the route in. All of the meshes that you own and that have been shared (p. 112) 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 (p. 66) one 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. For `Virtual node name`, 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 (p. 62) one first.
10. (Optional) Choose `Additional configuration`. From the following protocols, choose the protocol that you selected for `Route type`, specify settings in the console as desired, and then select `Create route`.
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 duration of time between retry attempts. 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 (p. 77).

- 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

- 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. 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 (p. 62).
- Idle duration – 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.

(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 duration of time between retry attempts. 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 (p. 77).

• 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. 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 (p. 62).
• Idle duration – The maximum amount of time that the route can be idle.

TCP

(Optional) Timeouts

• Idle duration – The maximum amount of time that the route can be idle.

Deleting a route

To delete a route using the AWS CLI version 1.18.116 or higher, see the example in the AWS CLI reference for the delete-route command.

To delete a route using the AWS Management Console

2. Choose the mesh that you want to delete a route from. All of the meshes that you own and that have been shared (p. 112) with you are listed.
3. Choose Virtual routers in the left navigation.
4. Choose the router that you want to delete a route from.
5. In the Routes table, choose the route that you want to delete and select Delete.
6. In the confirmation box, type delete and then select Delete.
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 (p. 71).
- 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 (p. 71), Scale out before scale in (p. 72), and Implement container health checks (p. 72).

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 (p. 67) 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 (p. 77).

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 (p. 72) 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.

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.
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 Docker container image and ensures that this container image is patched with the latest vulnerability and performance patches. App Mesh tests a new Envoy proxy release against the App Mesh feature set before making a new container image available to you.

- All supported Regions other than me-south-1 and ap-east-1, eu-south-1. You can replace `region-code` with any Region other than me-south-1 and ap-east-1, eu-south-1.

<table>
<thead>
<tr>
<th>Region</th>
<th>Container Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>840364872350.dkr.ecr.region-code.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod</td>
<td></td>
</tr>
<tr>
<td>me-south-1 Region:</td>
<td></td>
</tr>
<tr>
<td>772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod</td>
<td></td>
</tr>
<tr>
<td>ap-east-1 Region:</td>
<td></td>
</tr>
<tr>
<td>856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod</td>
<td></td>
</tr>
<tr>
<td>eu-south-1 Region:</td>
<td></td>
</tr>
<tr>
<td>422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod</td>
<td></td>
</tr>
</tbody>
</table>

**Important**

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

Access to this container image in Amazon ECR is controlled by AWS Identity and Access Management, so you must use IAM to ensure 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. Further, if you use IAM policies that limit access to specific Amazon ECR resources, then you must ensure 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'd 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 you need to 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. Otherwise, while we recommend that you use the App Mesh Envoy container image as the best supported option, you may use your own Envoy image.

## Envoy configuration variables

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

### 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 (p. 142), 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, warning, error, critical, off

Default: info

### Admin variables

These environment variables let you configure Envoy’s administrative interface.
ENVOY_ADMIN_ACCESS_PORT

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

ENVOY_ADMIN_ACCESS_LOG_FILE

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

Tracing variables

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

AWS X-Ray variables

The following environment variables help you 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 (default value is 0).

XRAY_DAEMON_PORT

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

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.

Envoy tracing variable

The following environment variable enables you 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.v2.Tracing in the Envoy documentation.

DogStatsD variables

The following environment variables help you 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.

**ENVOY_STATS_SINKS_CFG_FILE**

Specify a file path in the Envoy container file system to override the default DogStatsD configuration with your own. For more information, see config.metrics.v2.DogStatsdSink in the Envoy documentation.

---

**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 (p. 142).

**APPMESH_RESOURCE_CLUSTER**

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 variable can only be used with version 1.15.0 or later of the Envoy image.

---

**Envoy stats variables**

The following environment variables help you 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.v2.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.

---

**Deprecated variables**

The environment variables `APPMESH_VIRTUAL_NODE_NAME` and `APPMESH_RESOURCE_NAME` are deprecated in Envoy version 1.15.0 or later, but are 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.
Default route retry policy

If you had no meshes in your account before July 29, 2020, then 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 is created for any Envoy routes that existed before, on, or after July 29, 2020, unless you open a ticket with AWS support. Once support processes the ticket, then the default policy will be 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 (p. 67) or define a virtual node provider for an App Mesh virtual service (p. 56). 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 is not visible through the App Mesh API. The default policy is only visible through Envoy. To view the configuration, enable the administration interface (p. 118) 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, which both include 503 errors. For more information, see Routes (p. 67).

- **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 (p. 67).

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. The settings are not visible through the App Mesh API. The settings are only visible through Envoy. To view the configuration, enable the administration interface (p. 118) 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, then 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**
Whether your Envoys have the Envoy or App Mesh default circuit breaker values, you are not able to modify the values.
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) (p. 79)
- App Mesh Preview Channel only – Mutual TLS Authentication (p. 83)
- How AWS App Mesh works with IAM (p. 86)
- Logging with AWS CloudTrail (p. 103)
- Data protection in AWS App Mesh (p. 105)
- Compliance validation for AWS App Mesh (p. 105)
- Infrastructure security in App Mesh (p. 106)
- Resilience in AWS App Mesh (p. 107)
- Configuration and vulnerability analysis in AWS App Mesh (p. 108)

**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 (p. 62) and Virtual gateways (p. 57). 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 Certificate Manager Private Certificate Authority (ACM PCA).
- A certificate stored on the local file system of a virtual node that is issued by your own certificate authority (CA)

Envoy Proxy authorization (p. 98) 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 namespace `apps.local`, you can create a certificate matching the name `mesh-endpoint.apps.local`, or a certificate with the wild card `*.apps.local`. **AWS Cloud Map is not available in the eu-south-1 Region.**

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
```

For additional requirements, select the issuer of the certificate that you're using.

**ACM PCA**

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 ACM 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 ACM PCA 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 CAs.

To configure a virtual node with certificates and CAs from ACM PCA, 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 ACM PCA 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 (p. 98) 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.

Your own CA
When using a certificate from your own CA, you'll need a public key, a private key, and a certificate chain from the CA. The files must all be stored in a location accessible to the proxy.

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 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.

**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**

**ACM PCA**

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 automatically reload the certificate in response to a move command. For more information about why Envoy only reloads certificates on move and the symbolic link swap example for Envoy's recommended approach for changing files for a running proxy see [core.ConfigSource](#) and [Updating runtime values via symbolic link swap](#) in the Envoy documentation.
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:

- **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.

**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.

- AWS Certificate Manager– AWS Certificate Manager Private Certificate Authority (ACM PCA) can be used along side SPIRE to serve as an upstream Certificate Authority. By using a Private Certificate Authority (PCA), private keys for the CAs are hardware based and not stored in the memory or on the disk. This adds an additional layer of security. For more information, see the SPIRE documentation.

## 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.

### 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**

App Mesh doesn't support wildcard DNS SANs. You need to provide exact names of the endpoints.

### 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 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
```
Configure Amazon ECS workloads to use mutual TLS authentication with AWS App Mesh

You can configure your mesh to use mutual 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 mutual TLS authentication with AWS App Mesh

You can configure the AWS App Mesh Controller for Kubernetes to enable mutual TLS authentication for virtual node service backends and listeners. Make sure that the certificates are available to the Envoy proxy sidecars that you add to your workloads.

- 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, add a sidecar that implements Envoy's SDS API with access to the certificate.

**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.

App Mesh mutual TLS authentication walkthroughs

- **Mutual TLS authentication walkthrough**: This walkthrough describes how you can use the App Mesh Preview CLI to build a color app with mutual TLS authentication.
- **Amazon EKS mutual TLS walkthrough**: This walkthrough shows how you can use mutual TLS 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 (p. 87)
- Authenticating with identities (p. 87)
• Managing access using policies (p. 89)
• How AWS App Mesh works with IAM (p. 91)
• AWS App Mesh identity-based policy examples (p. 93)
• Using service-linked roles for App Mesh (p. 96)
• Envoy Proxy authorization (p. 98)
• Troubleshooting AWS App Mesh identity and access (p. 101)

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 (p. 101).

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 employees 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 (p. 91).

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 (p. 93).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don’t use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the AWS General Reference.

Regardless of the authentication method that you use, you might also 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 Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

AWS account root user

When you first create an AWS account, you begin with a single 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 do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

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:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the IAM User Guide.

- **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.

- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional
Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. 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.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

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 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 (p. 91)
- App Mesh resource-based policies (p. 93)
- Authorization based on App Mesh tags (p. 93)
- App Mesh IAM roles (p. 93)

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.

```
```

To specify all instances that belong to a specific account, use the wildcard (*).

```
```

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": [  
  ]
```

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 (p. 93).

**App Mesh resource-based policies**

App Mesh does not support resource-based policies.

**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 (p. 96).

**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 (p. 96).

**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.
Policy best practices

Identity-based policies are very powerful. They 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 using AWS managed policies** – To start using App Mesh quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the IAM User Guide.

- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the IAM User Guide.

- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition 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.

```json
{
  "Version": "2012-10-17",
```
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::*: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"
            ]
        }
    ]
}
```
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`.

```json
{
  "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.

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.

The role permissions policy allows App Mesh to complete the servicediscovery:DiscoverInstances action on all AWS resources.

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 (p. 67) defined for all routers in the mesh.
2. Delete all virtual routers (p. 66) in the mesh.
3. Delete all virtual services (p. 56) in the mesh.
4. Delete all virtual nodes (p. 62) in the mesh.
5. Delete the mesh (p. 56).

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 (p. 73) 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. Proxy authorization is required for virtual nodes that use Transport Layer Security (TLS) (p. 79) and for virtual gateways (with or without TLS), but will eventually be required for all App Mesh capabilities. Until proxy authorization is required for all App Mesh capabilities, it is only required if specifically mentioned in the documentation for the capability. It is recommended 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 that you host the proxy on.

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 (p. 98). 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 single 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](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html). 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](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html) and [Specifying an IAM Role for your Service Account](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html).

• **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](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html).

### 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](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html).

### 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](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html) in the *Amazon EKS User Guide* and [Configure Service Accounts for Pods](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html) 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](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html).

- **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](https://docs.aws.amazon.com/eks/latest/userguide/iam-roles.html).

### 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`.

    ```bash
    kubectl describe deployment my-deployment
    ```
**Envoy Proxy authorization**

**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 (p. 99) 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 (p. 98) 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 (p. 98).

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 (p. 98) 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 (p. 99) 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 (p. 98). 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 (p. 98).

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 (p. 98).
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 (p. 99) 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 (p. 98). 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 (p. 98).
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 (p. 101)
- I want to view my access keys (p. 102)
- I'm an administrator and want to allow others to access App Mesh (p. 102)
- I want to allow people outside of my AWS account to access my App Mesh resources (p. 102)

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 user name and password.

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.


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 view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you
can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret
access key (for example, wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY). Like a user name and
password, you must use both the access key ID and secret access key together to authenticate your
requests. Manage your access keys as securely as you do your user name and password.

Important
Do not provide your access keys to a third party, even to help find your canonical user ID. By
doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in
a secure location. The secret access key is available only at the time you create it. If you lose your secret
access key, you must add new access keys to your IAM user. You can have a maximum of two access keys.
If you already have two, you must delete one key pair before creating a new one. To view instructions,
see Managing access keys in the IAM User Guide.

I'm an administrator and want to allow others to access App
Mesh

To allow others to access App Mesh, you must create an IAM entity (user or role) for the person or
application that needs access. They will use the credentials for that entity to access AWS. You must then
attach a policy to the entity that grants them the correct permissions in App Mesh.

To get started right away, see Creating your first IAM delegated user and group in the IAM User Guide.

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 (p. 91).
• 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 with AWS CloudTrail

AWS App Mesh works with AWS CloudTrail, a service that provides a record of actions taken by a user, a role, or an AWS service in App Mesh. 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. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for App Mesh. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. 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, which user or account made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/).

**App Mesh information in CloudTrail**

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in App Mesh, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/).

For an ongoing record of events in your AWS account, including events for App Mesh, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for Creating a Trail](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/)
- [CloudTrail Supported Services and Integrations](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/)
- [Configuring Amazon SNS Notifications for CloudTrail](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/)
- [Receiving CloudTrail Log Files from Multiple Regions](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/)

All App Mesh actions are logged by CloudTrail and are documented in [API Actions](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/). For example, calls to the *CreateMesh*, *DescribeMesh*, and *DeleteMesh* actions generate entries in the CloudTrail log files. Actions that App Mesh takes on your behalf, such as creating a service-linked role when you create a mesh, are also logged.

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 or AWS Identity and Access Management (IAM) user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the [CloudTrail userIdentity Element](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/).

**Understanding App Mesh log file entries**

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request
parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they
don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateMesh` action.

```json
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2019-10-18T14:56:49Z"
            }
        },
        "invokedBy": "signin.amazonaws.com"
    },
    "eventTime": "2019-10-18T15:00:49Z",
    "eventSource": "appmesh.amazonaws.com",
    "eventName": "CreateMesh",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "205.251.233.178",
    "userAgent": "signin.amazonaws.com",
    "requestParameters": {
        "meshName": "my-mesh",
        "clientToken": "00000000-0000-0000-0000-0000000000",
        "spec": {
            "egressFilter": {
                "type": "DROP_ALL"
            }
        }
    },
    "responseElements": {
        "mesh": {
            "meshName": "my-mesh",
            "status": {
                "status": "ACTIVE"
            },
            "metadata": {
                "version": 1,
                "lastUpdatedAt": "Oct 18, 2019 3:00:49 PM",
                "uid": "00000000-0000-0000-0000-000000000000",
                "createdAt": "Oct 18, 2019 3:00:49 PM",
                "arn": "arn:aws:iam::123456789012::mesh/my-mesh"
            },
            "spec": {
                "egressFilter": {
                    "type": "DROP_ALL"
                }
            }
        }
    },
    "requestID": "cb8c167e-EXAMPLE",
    "eventID": "e3c6f4ce-EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "apiVersion": "2019-01-25",
    "recipientAccountId": "123456789012"
}
```
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. This content includes 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 user accounts with 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 recommend TLS 1.2 or later.
- 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 personal 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 sensitive identifying information, such as your customers’ account numbers, into free-form 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 App Mesh or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don’t 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

All of the data and configurations that you create are encrypted at rest.

Encryption in transit

App Mesh endpoints use the HTTPS protocol. All communication between the Envoy proxy and the App Mesh Envoy Management Service is encrypted. Communication between containers within virtual nodes is not encrypted, but this traffic doesn’t leave the network namespace.

Compliance validation for AWS App Mesh

Third-party auditors assess the security and compliance of AWS App Mesh as part of multiple AWS compliance programs, such as the European Union’s General Data Protection Regulation (GDPR). App
Mesh is a U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA) eligible service. You can use App Mesh to manage communication between services that contain protected health information (PHI). HIPAA eligibility applies to all AWS Regions where App Mesh is available.

If you have a HIPAA Business Associate Addendum (BAA) in place with AWS, then you can use App Mesh for your HIPAA eligible workloads. If you do not have a BAA in place with AWS, or if have any other questions about running HIPAA-regulated workloads on AWS, then you can contact an AWS business representative. For information and best practices about how to configure AWS HIPAA eligible services to store, process, and transmit PHI, see the Architecting for HIPAA Security and Compliance on Amazon Web Services paper.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

Your compliance responsibility when using App Mesh 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 security- and compliance-focused baseline environments on AWS.
- **Architecting for HIPAA Security and Compliance paper** – This paper describes how companies can use AWS to create HIPAA-compliant applications.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
- **AWS Config** – This AWS 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 that helps you check your compliance with security industry standards and best practices.

### Infrastructure security in App Mesh

As a managed service, App Mesh is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes paper.

You use AWS published API calls to access App Mesh through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). 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.

The Envoy proxy is deployed with a microservice application that is running on an AWS compute service. Each of the compute services are deployed within an Amazon VPC.

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) (p. 106).

### 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.
- Controlling access to App Mesh by attaching an endpoint policy to the VPC endpoint isn't supported. By default, full access to the service will be allowed through the endpoint. For more information, see Controlling Access to Services with VPC Endpoints in the Amazon VPC User Guide.

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.

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 command returns the availability zones to which you can deploy an App Mesh interface VPC endpoint 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[]'
```

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:v1.15.1.0-prod
```
App Mesh observability

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

- Access logs
- Statistics
- Proxy logs

Access logs

When you create your virtual nodes, you have the option to configure Envoy access logs. In the console, this is in the Advanced configuration section of the virtual node create or update workflows.

Logging

HTTP access logs path - optional

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

/dev/stdout

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

The preceding image shows a logging path of /dev/stdout for Envoy access logs. The following code block shows the JSON representation that you can use in the AWS CLI.

```
"logging": {
  "accessLog": {
    "file": {
      "path": "/dev/stdout"
    }
  }
}
```

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. 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.

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

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. 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 (p. 118).

Prometheus on Kubernetes

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.

Amazon CloudWatch supports both the discovery and collection of Prometheus metrics. For more information, see Container Insights Prometheus Metrics. You can enable your Kubernetes and Amazon EKS clusters to emit metrics to CloudWatch. As of July, 2020, CloudWatch support for Prometheus metrics is in the beta stage.

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.

Proxy logs

Envoy 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'
```
Working with shared meshes

A shared mesh allows resources created by different accounts to communicate with each other in the same mesh.

An AWS Identity and Access Management 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.

Shared mesh permissions

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.
- Owners can create virtual gateways and virtual gateway routes.
- 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.

Prerequisites for sharing meshes

The following prerequisites must be met in order to share a mesh:

- 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 (p. 62).
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 App Mesh console, you add it to an existing resource share. To add the mesh to a new resource share, you must first create the resource share using the AWS RAM console.

If you are 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

See Creating a Resource Share in the AWS RAM User Guide. When selecting a resource type, select Meshes, and then select the mesh you want to share. If no meshes are listed, then you need to create a mesh first. For more information, see Creating a service mesh (p. 56).

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 but does not delete the resources created by the consumers. Once 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, and any other accounts with resources in the mesh, from receiving any configuration information after the mesh is unshared. 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

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 App Mesh console and AWS CLI.

**To identify a shared mesh using the App Mesh console**

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.

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 (p. 146). 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 (p. 115)
- App Mesh on AWS Outposts (p. 116)

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 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

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 Docker container image and ensures that this container image is patched with the latest vulnerability and performance patches. App Mesh tests a
new Envoy proxy release against the App Mesh feature set before making a new container image available to you.

- All supported Regions other than me-south-1 and ap-east-1, eu-south-1. You can replace `region-code` with any Region other than me-south-1 and ap-east-1, eu-south-1.

<table>
<thead>
<tr>
<th>Region</th>
<th>Image URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>me-south-1</td>
<td>772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod</td>
</tr>
<tr>
<td>ap-east-1</td>
<td>856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod</td>
</tr>
<tr>
<td>eu-south-1</td>
<td>422531588944.dkr.ecr.eu-south-1.amazonaws.com/aws-appmesh-envoy:v1.15.1.0-prod</td>
</tr>
</tbody>
</table>

**Important**

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

Access to this container image in Amazon ECR is controlled by AWS Identity and Access Management, so you must use IAM to ensure 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. Further, if you use IAM policies that limit access to specific Amazon ECR resources, then you must ensure 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'd 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 you need to 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. Otherwise, while we recommend that you use the App Mesh Envoy container image as the best supported option, you may use your own Envoy image.

## Envoy configuration variables

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

### 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.
Optional variables

APPMESS_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 (p. 142), 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, warning, error, critical, off

Default: info

Admin variables

These environment variables let you configure Envoy's administrative interface.

ENVOY_ADMIN_ACCESS_PORT

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

ENVOY_ADMIN_ACCESS_LOG_FILE

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

Tracing variables

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

AWS X-Ray variables

The following environment variables help you 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
to enable, set the value to 1 (default value is 0).

XRAY_DAEMON_PORT

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

**Envoy tracing variable**

The following environment variable enables you 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.v2.Tracing in the Envoy documentation.

**DogStatsD variables**

The following environment variables help you 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.

**ENVOY_STATS_SINKS_CFG_FILE**

Specify a file path in the Envoy container file system to override the default DogStatsD configuration with your own. For more information, see config.metrics.v2.DogStatsdSink in the Envoy documentation.

**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 (p. 142).

APPMESH_RESOURCE_CLUSTER

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 variable can only be used with version 1.15.0 or later of the Envoy image.

Envoy stats variables

The following environment variables help you 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.v2.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.

Deprecated variables

The environment variables APPMESH_VIRTUAL_NODE_NAME and APPMESH_RESOURCE_NAME are deprecated in Envoy version 1.15.0 or later, but are 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.

Default route retry policy

If you had no meshes in your account before July 29, 2020, then 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 is created for any Envoy routes that existed before, on, or after July 29, 2020, unless you open a ticket with AWS support. Once support processes the ticket, then the default policy will be 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 (p. 67) or define a virtual node provider for an App Mesh virtual service (p. 56). 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 is not visible through the App Mesh API. The default policy is only visible through Envoy. To view the configuration, enable the administration interface (p. 118)
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, which both include 503 errors. For more information, see Routes (p. 67).

- **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 (p. 67).

## 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. The settings are not visible through the App Mesh API. The settings are only visible through Envoy. To view the configuration, enable the administration interface (p. 118) 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, then 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**
  
  Whether your Envoys have the Envoy or App Mesh default circuit breaker values, you are not able to modify the values.

(p. 73), the administration endpoint is enabled by default on port 9901. Examples provided in App Mesh troubleshooting best practices (p. 118) 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 (p. 75). 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 (p. 62) and Virtual gateways (p. 57) 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 (p. 73), 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
```

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"
    ],
    "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

Resolution
Make sure that your Envoy proxy has access to the internet or to a private VPC endpoint (p. 106) and that your security groups allow egress 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
```
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.

<table>
<thead>
<tr>
<th>gRPC status code</th>
<th>Cause</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Graceful disconnect from the Envoy management service.</td>
<td>There is no issue. App Mesh occasionally disconnects Envoy proxies with this status code. Envoy will reconnect and continue receiving updates.</td>
</tr>
<tr>
<td>3</td>
<td>The mesh endpoint (virtual node or virtual gateway), or one of its associated resources, could not be found.</td>
<td>Double check your Envoy configuration to make sure that it has the appropriate 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.</td>
</tr>
<tr>
<td>7</td>
<td>The Envoy proxy is unauthorized to perform an action, such as connect to the Envoy management service, or retrieve associated resources.</td>
<td>Make sure that you create an IAM policy (p. 98) that has the appropriate policy statements 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.</td>
</tr>
<tr>
<td>8</td>
<td>The number of Envoy proxies for a given App Mesh resource exceeds the account-level service quota.</td>
<td>See App Mesh service quotas (p. 146) for information on default account quotas and how to request a quota increase.</td>
</tr>
<tr>
<td>16</td>
<td>The Envoy proxy does not have valid authentication credentials for AWS.</td>
<td>Make sure that the Envoy has appropriate credentials to connect to AWS services through an IAM user or role.</td>
</tr>
</tbody>
</table>
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 liveness 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 (p. 126).
- **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 ingress 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.
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.:<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
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 (p. 98) 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 running on your compute service has successfully connected to the App Mesh Envoy management service. For more information, see Envoy disconnected from App Mesh Envoy management service with error text (p. 126). 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).
- 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 (p. 131).
  - 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.
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 egress 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 `egress 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`.

Unable to connect to a MySQL or SMTP server

Symptoms

When allowing egress 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](https://github.com/aws/app-mesh/issues) GitHub issue. This error occurs because the egress listener in Envoy configured by App Mesh adds the Envoy TLS Inspector listener filter. For more information, see [TLS Inspector](https://www.envoyproxy.io/docs/envoy/latest/inspector/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](https://dev.mysql.com/doc/refman/8.0/en/mysql-initial-handshake.html) in the MySQL documentation. Because the server sends the first packet, inspection at the filter fails.

To work around this issue, either upgrade your Envoy version to AppMesh image 1.15.0 (or later version), or use the following instructions: add port `3306` to the list of values for the `APPMESH_EGRESS_IGNORED_PORTS` in your services for MySQL, and the port you are using for SMTP.

**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](https://docs.aws.amazon.com/appmesh/latest/userguide/k8s-update.html), [Update services for Amazon ECS](https://docs.aws.amazon.com/appmesh/latest/userguide/ecs-update.html), or [Update services for Amazon EC2](https://docs.aws.amazon.com/appmesh/latest/userguide/ec2-update.html).

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 (p. 130).

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 egress filter type has been set to ALLOW_ALL. When the egress 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 egress filter value to DROP_ALL.

**Note**

Setting the mesh egress 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](https://github.com) 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](#) (p. 71).

If your issue is still not resolved, then consider opening a [GitHub issue](https://github.com) or contact [AWS Support](https://aws.amazon.com/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](https://github.com) or contact [AWS Support](https://aws.amazon.com/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 cannot 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](https://github.com) on Github. The Envoy proxy only intercepts ingress 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 ingress 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](https://github.com) or contact [AWS Support](https://aws.amazon.com/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 10 replicas for a virtual node**

**Symptoms**

When you are scaling the number of replicas, such as Amazon ECS tasks, Kubernetes pods, or Amazon EC2 instances, for a virtual node beyond 10, Envoy container health checks for new and currently running Envoy pods begin to fail. Downstream applications sending traffic to the virtual node begin seeing request failures with HTTP status code 503.

**Resolution**

App Mesh's default quota for the number of Envoy pods per virtual node is 10. When the number of running Envoy pods exceeds this quota, new and currently running Envoy pods 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 (p. 146).

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 (p. 71).

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.

**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 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 (p. 73), 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.
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

This is a known issue. For more information, see the Support Dynamic X-Ray sampling configuration issue on GitHub. The Envoy proxy allows configuration of sampling rules through static configuration. For more information, see AWS X-Ray Tracer configuration in the Envoy documentation. The Envoy proxy cannot currently be configured to dynamically configure sampling rules through the App Mesh service APIs.

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 (p. 124). 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 (p. 129).
• 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).
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 egress request headers and forward the request to the appropriate destination through a TLS session. For more information, see Transport Layer Security (TLS) (p. 79).

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 (p. 118) and Enable Envoy DogStatsD integration for metric offload (p. 124). 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.

```bash
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 (p. 135).
- 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 liveness 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 liveness checks begin failing.

Resolution

Make sure that the pod has a mesh and virtual node associated with it. Mesh and virtual node settings are set through annotations on the associated deployment object, as in the following example.

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: default
spec:
  replicas: 3
  template:
    metadata:
      annotations:
        *appmesh*.k8s.aws/mesh: my-mesh
        appmesh.k8s.aws/virtualNode: my-virtual-node
...
```

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 (p. 126). 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 (p. 140).
• 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 | \n        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 | \n        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.
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 (p. 143) 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.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>App Mesh production service</th>
<th>App Mesh Preview Channel service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend endpoint</td>
<td>appmesh.us-west-2.amazonaws.com</td>
<td>appmesh-preview.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td>Envoy management service endpoint</td>
<td>appmesh-envoy-management.us-west-2.amazonaws.com</td>
<td>appmesh-preview-envoy-management.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td>CLI</td>
<td>aws appmesh list-meshes</td>
<td>aws appmesh-preview list-meshes (only available after adding the Preview Channel service model)</td>
</tr>
<tr>
<td>Signing name</td>
<td>appmesh</td>
<td>appmesh-preview</td>
</tr>
<tr>
<td>Service principal</td>
<td>appmesh.amazonaws.com</td>
<td>appmesh-preview.amazonaws.com</td>
</tr>
</tbody>
</table>

142
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.

   ```bash
   aws configure add-model \
   --service-name appmesh-preview \
   ```

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.

   ```bash
   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
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? (p. 143). 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 (p. 67) 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? (p. 143).

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**

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.

```bash
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.

---

AWS App Mesh User Guide
### 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.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement mutual TLS authentication</td>
<td>(App Mesh Preview Channel only) You can implement mutual TLS authentication.</td>
<td>November 23, 2020</td>
</tr>
<tr>
<td>Implement connection pooling for a virtual gateway listener</td>
<td>You can implement connection pooling for a virtual gateway listener.</td>
<td>November 5, 2020</td>
</tr>
<tr>
<td>Implement connection pooling and outlier detection for a virtual node listener</td>
<td>You can implement connection pooling and outlier detection for a virtual node listener.</td>
<td>November 5, 2020</td>
</tr>
<tr>
<td>Region launch in eu-south-1</td>
<td>App Mesh is now available in the eu-south-1 Region. AWS Cloud Map is not available in the eu-south-1 Region.</td>
<td>October 21, 2020</td>
</tr>
<tr>
<td>Implement connection pooling for a virtual gateway listener</td>
<td>(App Mesh Preview Channel only) You can implement connection pooling for a virtual gateway listener.</td>
<td>September 28, 2020</td>
</tr>
<tr>
<td>Implement connection pooling and outlier detection for a virtual node listener</td>
<td>(App Mesh Preview Channel only) You can implement connection pooling and outlier detection for a virtual node listener.</td>
<td>September 28, 2020</td>
</tr>
<tr>
<td>Create a virtual gateway and gateway route for mesh ingress</td>
<td>Enable resources that are outside of a mesh to communicate to resources that are inside of a mesh.</td>
<td>July 10, 2020</td>
</tr>
<tr>
<td>Create and manage App Mesh resources from within Kubernetes with the App Mesh controller for Kubernetes</td>
<td>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.</td>
<td>June 18, 2020</td>
</tr>
<tr>
<td>Add a timeout value to a virtual node listener and route</td>
<td>You can add a timeout value to a virtual node listener and route.</td>
<td>June 18, 2020</td>
</tr>
<tr>
<td>Add a timeout value to a virtual node listener</td>
<td>(App Mesh Preview Channel only) You can add a timeout value to a virtual node listener.</td>
<td>May 29, 2020</td>
</tr>
<tr>
<td>Create a virtual gateway for mesh ingress</td>
<td>(App Mesh Preview Channel only) Enable resources outside</td>
<td>April 8, 2020</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>TLS encryption</td>
<td>Use certificates from an AWS Certificate Manager Private Certificate Authority or your own certificate authority to encrypt communication between virtual nodes using TLS.</td>
<td>January 17, 2020</td>
</tr>
<tr>
<td>Share a mesh with another account</td>
<td>You can share a mesh with another account. Resources created by any account can communicate with other resources in the mesh.</td>
<td>January 17, 2020</td>
</tr>
<tr>
<td>Add a timeout value to a route</td>
<td>You can add a timeout value to a route.</td>
<td>January 17, 2020</td>
</tr>
<tr>
<td>Create an App Mesh proxy on an AWS Outpost</td>
<td>You can create an App Mesh Envoy proxy on an AWS Outpost.</td>
<td>December 3, 2019</td>
</tr>
<tr>
<td>HTTP/2 and gRPC support for routes, virtual routers, and virtual nodes</td>
<td>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.</td>
<td>October 25, 2019</td>
</tr>
<tr>
<td>Retry policy</td>
<td>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.</td>
<td>September 10, 2019</td>
</tr>
<tr>
<td>TLS encryption</td>
<td>Encrypt communication between virtual nodes using TLS.</td>
<td>September 6, 2019</td>
</tr>
<tr>
<td>HTTP header-based routing</td>
<td>Route traffic based on the presence and values of HTTP headers in a request.</td>
<td>August 15, 2019</td>
</tr>
<tr>
<td>Availability of the App Mesh Preview Channel</td>
<td>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.</td>
<td>July 19, 2019</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>App Mesh Interface VPC Endpoints (AWS PrivateLink)</strong></td>
<td>Improve the security posture of your 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 VPC and App Mesh to the Amazon network.</td>
<td>June 14, 2019</td>
</tr>
<tr>
<td><strong>Added AWS Cloud Map as a virtual node service discovery method</strong></td>
<td>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.</td>
<td>June 13, 2019</td>
</tr>
<tr>
<td><strong>Create App Mesh resources automatically in Kubernetes</strong></td>
<td>Create App Mesh resources and add the App Mesh sidecar container images to your Kubernetes deployments automatically when you create resources in Kubernetes.</td>
<td>June 11, 2019</td>
</tr>
<tr>
<td><strong>App Mesh General Availability (p. 147)</strong></td>
<td>The App Mesh service is now generally available for production use.</td>
<td>March 27, 2019</td>
</tr>
<tr>
<td><strong>App Mesh API update (p. 147)</strong></td>
<td>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.</td>
<td>March 7, 2019</td>
</tr>
<tr>
<td><strong>App Mesh initial release (p. 147)</strong></td>
<td>Initial documentation for service public preview</td>
<td>November 28, 2018</td>
</tr>
</tbody>
</table>