AWS App Mesh: User Guide
Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.
# Table of Contents

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

- **Mesher**: 18
  - Creating a Service Mesh: 18

- **Virtual Services**: 19
  - Creating a Virtual Service: 19

- **Virtual nodes**: 20
  - Creating a virtual node: 20
  - Deleting a virtual node: 20

- **Virtual Routers**: 25
  - Creating a Virtual Router: 25
  - Deleting a Virtual Router: 26

- **Routes**: 27
  - Creating a Route: 27
  - Deleting a Route: 31

- **Tutorial: Configure App Mesh integration with Kubernetes**: 32
  - Prerequisites: 32
  - Step 1: Install the integration components: 32
  - Step 2: Deploy App Mesh resources: 34
  - Step 3: Create or update services: 39
  - Step 4: Clean up: 43

- **Envoy**: 44
  - Envoy Configuration Variables: 45
    - Required Variables: 45
    - Optional Variables: 45
    - AWS X-Ray Variables: 45
    - DogStatsD Variables: 45
    - Envoy Stats Variables: 46
  - Access Logs: 46

- **Security**: 47
  - Data Protection: 47
    - Data Encryption: 48
  - Identity and Access Management: 48
    - Audience: 49
    - Authenticating With Identities: 49
    - Managing Access Using Policies: 51
    - How AWS App Mesh Works with IAM: 52
    - Identity-Based Policy Examples: 55
    - Troubleshooting: 57

- **Prerequisites**: 32

- **Steps**: 39
  - 1: Create a mesh and virtual service: 6
  - 2: Create a virtual node: 6
  - 3: Create a virtual router and route: 7
  - 4: Review and create: 9
  - 5: Create additional resources: 9
  - 6: Update services: 13

- **How AWS App Mesh Works with IAM**: 52
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. 18).
- **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. 19).

- **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. 20).

- **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. 25) and Routes (p. 27).

- **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. 44).

## 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 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 AWS App Mesh

This topic helps you use AWS App Mesh with an actual service that is running on Amazon ECS, Kubernetes, or Amazon EC2.

Scenario

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

- Includes 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've 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's proven to be reliable. Once all traffic is being sent to serviceBv2, you want to deprecate serviceB.
- You don't 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've 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 the services hosting your actual 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 can't use the mesh until you have deployed actual services.

If you don't already have services running, you can:

- Create an Amazon ECS service with service discovery.
- Setup an Amazon EKS cluster and deploy a cluster and deploy a sample application to it.
Step 1: Create a mesh and virtual service

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

Create the following resources:

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

You can use the AWS Management Console or the AWS CLI version 1.18.16 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 1.18.16 or higher installed, 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.
   
   ```bash
   aws appmesh create-mesh --mesh-name apps
   ```

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

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

Step 2: Create a virtual node

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

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

AWS Management Console

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

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

**Step 3: Create a virtual router and route**

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

Create the following resources:

- A virtual router named **serviceB**, since the **serviceB.apps.local** virtual service doesn't 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'll change the weight in a later step once you've 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 communications problem.

AWS Management Console

1. For **Virtual router name**, enter `serviceB`.
2. Under **Listener**, specify 80 for **Port** and choose `http2` for **Protocol**.
3. For **Route name**, enter `serviceB`.
4. For **Route type**, choose `http2`.
5. For **Virtual node name**, 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.

   ```
   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": "/"
   }
   }
   }
   ```
Step 4: review and create

Review the settings against the previous instructions.

AWS Management Console

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

AWS CLI

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

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

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

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

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

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

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

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

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

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

Step 5: Create additional resources

To complete the scenario, you need to:

- Create one virtual node named serviceBv2 and another named serviceA. Both virtual nodes listen for requests over HTTP/2 port 80. For the serviceA virtual node, configure a backend of serviceb.apps.local, 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 doesn't 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, enter 80 for Port and select http2 for Protocol.
7. Choose Create virtual node.
8. Choose Create virtual node again, and enter serviceA for the Virtual node name, for Service discovery method, choose DNS, and for DNS hostname, enter servicea.apps.local.
9. Expand Additional configuration.
10. Select Add backend. Enter serviceb.apps.local.
11. Enter 80 for Port, choose http2 for Protocol, and then choose Create virtual node.
12. In the left navigation pane, select Virtual routers and then select the serviceB virtual router from the list.
13. Under Routes, select the route named ServiceB that you created in a previous step, and choose Edit.
14. Under Virtual node name, change the value of Weight for serviceB to 75.
15. Choose Add target, choose serviceBv2 from the drop-down list, and set the value of Weight to 25.
16. Choose Save.
17. In the left navigation pane, select Virtual services and then choose Create virtual service.
18. 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-serviceb2.json with the following contents:

   ```json
   {
     "meshName": "apps",
     "spec": {
       "listeners": [
   ```
Step 5: Create additional resources

```json
{
    "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": {
            "dns": {
                "hostname": "servicea.apps.local"
            }
        },
        "virtualNodeName": "serviceA"
    }
}
```

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 didn't send traffic anywhere, since the serviceB virtual router hadn't been created yet.

a. Create a file named `update-virtual-service.json` with the following contents:
Step 5: Create additional resources

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

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

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

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

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

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

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

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

5. Create the serviceA virtual service.

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

```
{
    "meshName": "apps",
    "spec": {
        "provider": {
            "virtualNode": {
                "virtualNodeName": "serviceA"
            }
        }
    }
}
```
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 Proxy authorization.
- Update each of your existing services to use the Envoy proxy. To update your existing service that is running on Amazon ECS, see Getting Started with App Mesh and Amazon ECS. To update your existing service that is running on Kubernetes, see Getting Started with App Mesh and Kubernetes. To update your existing service that is running on Amazon EC2, complete the steps that follow.

To configure an Amazon EC2 instance as a virtual node member

1. Launch an Amazon EC2 instance with an IAM role that has read access to Amazon ECR. This permissions allows the instance to pull the App Mesh Envoy container image from Amazon Elastic Container Registry. For more information, see Amazon ECR Managed Policies.
2. Connect to your instance via SSH.
3. Install Docker and the AWS CLI on your instance according to your operating system documentation.
4. 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. You can replace `us-west-2` with any supported Region except me-south-1 and ap-east-1.

   ```
   $(aws ecr get-login --no-include-email --region us-west-2 --registry-ids 840364872350)
   ```
   - me-south-1 Region

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.

b. Create the virtual service.

```
aws appmesh create-virtual-service --cli-input-json file://create-virtual-servicea.json
```
Step 6: Update services

5. 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. 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`. You can replace `us-west-2` with any supported Region except `me-south-1` and `ap-east-1` Region.

```
```

- `me-south-1` Region

```
sudo docker run --detach --env APPMESH_VIRTUAL_NODE_NAME=mesh/apps/virtualNode/serviceB -u 1337 --network host 772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.12.2.1-prod
```

- `ap-east-1` Region

```
sudo docker run --detach --env APPMESH_VIRTUAL_NODE_NAME=mesh/apps/virtualNode/serviceB -u 1337 --network host 856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.12.2.1-prod
```

6. Select more below and run the script on your instance to configure the networking policies. Replace the `APPMESH_APP_PORTS` value with the ports that your application code uses for ingress.

```
#!/bin/bash -e

# Start of configurable options
#

#APPMESH_START_ENABLED="0"
APPMESH_IGNORE_UID="1337"
APPMESH_APP_PORTS="8000"
APPMESH_ENVoy_EGRESS_PORT="15001"
APPMESH_ENVoy_INGRESS_PORT="15000"
APPMESH_EGRESS_IGNORED_IP="169.254.169.254,169.254.170.2"

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

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

# Port numbers Application and Envoy are listening on.
if [ -z "APPMESH_ENVOY_INGRESS_PORT" ] || [ -z "APPMESH_ENVOY_EGRESS_PORT" ] || [ -z "APPMESH_APP_PORTS" ]; then
  echo "All of APPMESH_ENVoy_INGRESS_PORT, APPMESH_ENVoy_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 "APPMESH_EGRESS_IGNORED_PORTS" ]; then
  APPMESH_EGRESS_IGNORED_PORTS="22"
else
  APPMESH_EGRESS_IGNORED_PORTS="$APPMESH_EGRESS_IGNORED_PORTS,22"
fi

# End of configurable options
#
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 \
    -m owner --uid-owner $APPMESH_IGNORE_UID \
    -j RETURN
  [ ! -z "APPMESH_IGNORE_GID" ] && 
    iptables -t nat -A APPMESH_EGRESS \
    -m owner --gid-owner $APPMESH_IGNORE_GID \
    -j RETURN
  [ ! -z "APPMESH_EGRESS_IGNORED_PORTS" ] && 
    iptables -t nat -A APPMESH_EGRESS \
    -p tcp \
    -m multiport --dports "$APPMESH_EGRESS_IGNORED_PORTS" \
    -j RETURN
  [ ! -z "APPMESH_EGRESS_IGNORED_IP" ] && 
    iptables -t nat -A APPMESH_EGRESS \
    -p tcp \
    -d "$APPMESH_EGRESS_IGNORED_IP" \
    -j RETURN
  # Redirect everything that is not ignored
  iptables -t nat -A APPMESH_EGRESS \

Step 6: Update services

```bash
-p tcp 
-j REDIRECT --to $APPMESH_ENVOY_EGRESS_PORT

# Apply APPMESH_EGRESS chain to non local traffic
iptables -t nat -A OUTPUT 
-p tcp 
-m addrtype ! --dst-type LOCAL 
-j APPMESH_EGRESS
}

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

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

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
            ;;
```
Step 6: Update services

*)
    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

7. Start your virtual node application code.
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 Management Console, complete the following steps. To create a service mesh using the AWS CLI version 1.18.16 or higher, see the example in the AWS CLI reference for the create-mesh command.

2. Choose Create mesh.
3. For Mesh name, specify a name for your service mesh.
4. Choose Create mesh to finish.
5. (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 ?? (p. 69).
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 in your service mesh using the AWS Management Console, complete the following steps. To create a virtual service using the AWS CLI version 1.18.16 or higher, see the example in the AWS CLI reference for the `create-virtual-service` command.

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. 69) 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.
   - 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 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 (either the full ARN or the truncated resource name) as the APPMESH_VIRTUAL_NODE_NAME 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 mesh/default/virtualNode/simpleapp. This is then mapped to the node.id and node.cluster Envoy parameters.

Note
If you require your Envoy stats or tracing to use a different name, you can override the node.cluster value that is set by APPMESH_VIRTUAL_NODE_NAME with the APPMESH_VIRTUAL_NODE_CLUSTER environment variable.

Creating a virtual node

To create a virtual node using the AWS CLI version 1.18.16 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. 69) with you are listed.
3. Choose Virtual nodes in the left navigation.
4. Choose Create virtual node.
5. For Virtual node name, enter a name for your virtual node.
6. For Service discovery method, choose one of the following options:
   - DNS – Specify the DNS hostname of the actual service that the virtual node represents. The Envoy proxy is deployed in an Amazon VPC. The proxy sends name resolution requests to the DNS server that is configured for the VPC. If the hostname resolves, the DNS server returns one or more IP addresses. For more information about VPC DNS settings, see Using DNS with your VPC. 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 AWSServiceRoleForAppMesh service-linked role (p. 59). For more information about AWS Cloud Map, see the AWS Cloud Map Developer Guide.
   - None – Select if your virtual node doesn't expect any inbound traffic.
7. Select Additional configuration
   - (Optional) To configure default requirements when communicating to backend virtual services, select Client policy defaults.
Proxy authorization (p. 61) 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 the section called “Certificate Requirements” (p. 23).
  - 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 specify a backend virtual service that the virtual node will communicate with, choose Add backend.
- Enter a 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 the section called “Certificate Requirements” (p. 23).
  - 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.

- 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.

8. If your virtual node expects ingress traffic, specify a Port and Protocol for the Listener.

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

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

- For Health check protocol, choose a protocol. If you select grpc, then your service must conform to the GRPC Health Checking Protocol.
- For Health check port, specify the port that the health check should run on.
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. 27) or as a provider in any virtual service (p. 19).

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. 69) 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.

Transport Layer Security (TLS)

In App Mesh, TLS encrypts communication between the Envoy proxies deployed with your applications that are represented as App Mesh virtual nodes. Your application code is not responsible for negotiating a TLS session. The proxy negotiates and terminates 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)

Proxy authorization (p. 61) must be enabled for the Envoy proxy deployed with the application represented by the virtual node. We recommend that when you enable proxy authorization, you restrict access to only the virtual node that you're enabling encryption for.

Certificate Requirements

The Common Name (CN) or Subject Alternative Name (SAN) of the certificate must match specific criteria, depending on how the actual service represented by a virtual node is discovered.

• **DNS** – The certificate CN or one of the SANs must match the value provided in the DNS service discovery settings. You can also use wild cards such as *.apps.local in your certificates. If the certificate CN or SAN does not match the DNS service discovery settings, the connection between Envoys fails with the following error message, as seen from the downstream (calling) Envoy.

```
SSL error: 268435703:SSL routines:OPENSSL_internal:WRONG_VERSION_NUMBER
```

• **AWS Cloud Map** – The certificate CN and SAN are not considered when negotiating TLS. You can use any names for your certificates when using AWS Cloud Map, but we recommend that you create a name that's significant to your virtual node, such as virtual-node-name.apps.local. This name can help you identify the certificate in ACM.

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 account as the virtual node that will use the certificate. 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.

Your account must have the `acm:DescribeCertificate` and `acm-pca:DescribeCertificateAuthority` IAM permissions for the certificate that you use for TLS.

**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](https://aws.amazon.com/certificate-manager/pricing/).

When you enable proxy authorization (p. 61) for the Envoy proxy deployed with the application that a virtual node represents, the IAM role that you use must be assigned the `acm:ExportCertificate` action. In addition to the actions required for proxy authorization, you must add the `acm-pca:GetCertificateAuthorityCertificate` action to the policy for the CA that you use.

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](https://github.com/aws/app-mesh-examples/tree/master/aws-sample-app-tls).

**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 your application code. 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](https://github.com/aws/app-mesh-examples/tree/master/aws-sample-app-tls-with-local-cert).
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 virtual node. Determine how many successful TLS handshakes there were for a virtual node named `serviceA` with the following command.

```
curl -s 'http://servicea.apps.local:9901/stats' | grep ssl.handshake
```

In the following example returned output, there were three handshakes for the virtual node, 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 a virtual node named `serviceb` with the following command.

```
$ curl -s 'http://servicea.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.fail_verify_san: 0
```

For more information about Envoy TLS statistics, see Envoy Listener Statistics.
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 Management Console, complete the following steps. To create a virtual router using the AWS CLI version 1.18.16 or higher, see the example in the AWS CLI reference for the create-virtual-router command.

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. 69) 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.
7. Choose Create virtual router to finish.
Deleting a Virtual Router

To delete a virtual router using the AWS Management Console complete the following steps. 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. 27) or if it is specified as a provider for any virtual service (p. 19).

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. 69) 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**. To delete a virtual router, your account ID must be listed in either the **Mesh owner** or the **Resource owner** columns of the virtual router.
5. In the confirmation box, type **delete** and then 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 Management Console, complete the following steps. To create a route using the AWS CLI version 1.18.16 or later, see the examples in the AWS CLI reference for the create-route command.

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. 69) with you are listed.
3. Choose Virtual routers in the left navigation.
4. Choose the virtual router that you want to associate a new route with. If none are listed, then you need to create a virtual router first.
5. In the Routes table, choose Create route. To create a route, your account ID must be listed as the Resource owner of the route.
6. For Route name, specify the name to use for your route.
7. For Route type, choose the protocol that you want to route. The protocol that you select must match the listener protocol that you selected for your virtual router and the virtual node that you're routing traffic to.
8. Select the protocol that you want to route, enter or select the values that appear, and then select Create route.

gRPC

Route configuration

• (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.

Targets

• For Virtual node name, choose the virtual node that this route will serve traffic to. If none are listed, then you need to create a virtual node first.
• For Weight, choose a relative weight for the route. Select Add target to add additional virtual nodes. The total weight for all targets combined must be less than or equal to 100.
• Choose Additional configuration.

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.

**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.

**Retry policy**

A retry policy enables clients to protect themselves from intermittent network failures or intermittent server-side failures. A retry policy is optional. The retry timeout values define the duration of time between retry attempts.

• 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.

• Select one or more **HTTP retry events**.

• Select a **TCP retry event**.

• Select one or more **gRPC retry events**.

**HTTP and HTTP/2**

**Route configuration**

• (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.

**Targets**

• For **Virtual node name**, choose the virtual node that this route will serve traffic to. If none are listed, then you need to create a virtual node first.

• For **Weight**, choose a relative weight for the route. Select **Add target** to add additional virtual nodes. The total weight for all targets combined must be less than or equal to 100.

• Choose **Additional configuration**.

**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**.
**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.

**Retry policy**

A retry policy enables clients to protect themselves from intermittent network failures or intermittent server-side failures. A retry policy is optional. The retry timeout values define the duration of time between retry attempts.

- 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.
- Select one or more **HTTP retry events**.
- Select a **TCP retry event**.

**TCP**

**Route configuration**

- (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.

**Targets**

- For **Virtual node name**, choose the virtual node that this route will serve traffic to. If none are listed, then you need to create a virtual node first.
- For **Weight**, choose a relative weight for the route. Select **Add target** to add additional virtual nodes. The total weight for all targets combined must be less than or equal to 100.

**App Mesh Preview Channel only - Route Timeout**

By default, the App Mesh proxy has two timeout value types:

- **Per request** – The amount of time that a requester will wait for an upstream target to complete a response. The default value is 15 seconds.
- **Idle** – Bounds the amount of time a connection may be idle. The default value is none.

If the default timeout values don't meet your requirements, then you can specify your own values. You can specify a per request timeout for gRPC, HTTP, and HTTP/2 routes and an idle timeout for gRPC, HTTP, HTTP/2, and TCP routes.

For an end-to-end walk through of using a route timeout, see Timeout Policy Example on GitHub.

**To create a route with a timeout**

1. Add the Preview Channel service model to the AWS CLI with the following command.
aws configure add-model \
  --service-name appmesh-preview \

2. Create a mesh with the following command.

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

3. Create a JSON file named `create-virtual-node.json` with a virtual node configuration.

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

4. Create the virtual node with the following command.

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

5. Create a JSON file named `create-virtual-router.json` with a virtual router configuration.

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

6. Create the virtual router with the following command.

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

7. Create a JSON file named `create-route.json` with a route configuration. In the following configuration, the route has idle and perRequest timeouts.
Valid values for **timeout** are:

- **unit** – A time unit. Valid values are **s** and **ms**.
- **value** – The number of time units.

Specifying 0 disables the **perRequest** timeout. Specifying 0 disables the **idle** timeout.

8. Create the route with the following command.

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

**Deleting a Route**

To delete a route using the AWS Management Console, complete the following steps. To delete a route using the AWS CLI version 1.18.16 or higher, see the example in the AWS CLI reference for the `delete-route` command.

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. 69) 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**.
Tutorial: Configure App Mesh integration with Kubernetes

When you integrate AWS App Mesh with Kubernetes, you manage App Mesh resources, such as virtual services and virtual nodes, 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 following open source components that enable this integration:

- **App Mesh controller for Kubernetes** – The controller is accompanied by the deployment of three Kubernetes custom resource definitions: mesh, virtual service, and virtual node. The controller watches for creation, modification, and deletion of the custom resources and makes changes to the corresponding App Mesh mesh, virtual service (including virtual router and route), and virtual node resources through the App Mesh API. To learn more or contribute to the controller, see the GitHub project.

- **App Mesh sidecar injector for Kubernetes** – The injector installs as a webhook and injects the following containers into Kubernetes pods that are running in specific, labeled namespaces. To learn more or contribute, see the GitHub project.

- **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** – The route manager sets up a pod’s network namespace with iptables rules that route ingress and egress traffic through Envoy.

The features discussed in this topic are available as an open-source beta. This means that these features are well tested. Support for the features will not be dropped, though details may change. If the schema or schematics of a feature changes, instructions for migrating to the next version will be provided. This migration may require deleting, editing, and re-creating Kubernetes API objects.

Prerequisites

- An existing understanding of Kubernetes concepts. For more information, see What is Kubernetes.
- An existing Kubernetes cluster running version 1.13 or later. If you don't have an existing cluster, you can deploy one using the Getting Started with Amazon EKS guide.
- The AWS CLI version 1.18.16 or later installed. To install or upgrade the, 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, you can install it by completing the instructions in Using Helm with Amazon EKS.

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. Add the eks-charts repository to Helm.

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

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

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

3. Create a Kubernetes namespace for the controller.

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

4. Set the following variables. Replace `cluster-name` and `region-code` with the values for your existing cluster.

   ```bash
   export CLUSTER_NAME=cluster-name
   export AWS_REGION=region-code
   ```

5. 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. If you'd prefer to create the provider using the console, see Enabling IAM Roles for Service Accounts on your Cluster.

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

6. 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.

   ```bash
   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. 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.

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

   ```bash
   helm upgrade -i appmesh-controller eks/appmesh-controller
   --namespace appmesh-system
   --set region=$AWS_REGION
   --set serviceAccount.create=false
   --set serviceAccount.name=appmesh-controller
   ```
8. Confirm that the controller version is `v0.3.0` or later. You can view the change log on GitHub.

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

9. Install the App Mesh sidecar injector and create a mesh with the name `my-mesh`. For a list of all configuration options, see Configuration on GitHub.

```
helm upgrade -i appmesh-inject eks/appmesh-inject 
--namespace appmesh-system 
--set mesh.name=my-mesh 
--set mesh.create=true
```

## 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.

**To deploy App Mesh resources**

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

```
kubectl create ns my-app-1
```

2. Create an App Mesh virtual node.

   a. Create a file named `virtual-node.yaml` with the following contents. The file will be used to create an App Mesh virtual node named `my-service-a` in the `my-app-1` namespace. The virtual node represents a Kubernetes service that is created in a later step. The virtual node will communicate to a backend virtual service named `my-service-b.my-app-1.svc.cluster.local`, but it is not created in this tutorial.

```yaml
apiVersion: appmesh.k8s.aws/v1beta1
class: VirtualNode
metadata:
  name: my-service-a
  namespace: my-app-1
spec:
  meshName: my-mesh
  listeners:
  - portMapping:
      port: 9000
      protocol: http
  serviceDiscovery:
    dns:
      hostName: my-service-a.my-app-1.svc.cluster.local
  backends:
  - virtualService:
      virtualServiceName: my-service-b.my-app-1.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 Virtual nodes. To see all available settings for a virtual node that you can set in the preceding spec, run the following command.

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

b. Deploy the virtual node.
Step 2: Deploy App Mesh resources

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

c. View all of the resources in the `my-app-1` namespace.

```
kubectl -n my-app-1 get all
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesh.appmesh.k8s.aws/my-mesh</td>
<td>17m</td>
</tr>
<tr>
<td>NAME</td>
<td>AGE</td>
</tr>
<tr>
<td>virtualnode.appmesh.k8s.aws/my-service-a</td>
<td>16m</td>
</tr>
</tbody>
</table>

This output shows the mesh and the virtual node that exist in the `my-app-1` namespace.

d. Confirm that the virtual node was 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-app-1
```

Output

```json
{
  "virtualNode": {
    "meshName": "my-mesh",
    "metadata": {
      "createdAt": "2020-03-20T08:18:19.510000-05:00",
      "lastUpdatedAt": "2020-03-20T08:18:19.510000-05:00",
      "meshOwner": "111122223333",
      "resourceOwner": "111122223333",
      "uid": "111a11b1-c11d-1e1f-gh11-j11k11l1m711",
      "version": 1
    },
    "spec": {
      "backends": [
        {
          "virtualService": {
            "virtualServiceName": "my-service-b.my-app-1.svc.cluster.local"
          }
        }
      ],
      "listeners": [
        {
          "portMapping": {
            "port": 9000,
            "protocol": "http"
          }
        }
      ]
    }
  }
}
```
Note the value of arn in the preceding output. You will use it in a later step.

3. Create an App Mesh virtual service, virtual router, and route.

a. Create a file named `virtual-service.yaml` with the following contents. 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 the previous step. The value for name is the fully qualified domain name (FQDN) of the actual Kubernetes service that this virtual service abstracts. The service is created in the section called "Step 3: Create or update services" (p. 39). The controller will create the App Mesh virtual service, virtual router, and route resources. You can specify many more capabilities for your routes and use protocols other than http. For more information, see Virtual services, Virtual router, and Route.

```yaml
apiVersion: appmesh.k8s.aws/v1beta1
kind: VirtualService
metadata:
  name: my-service-a.my-app-1.svc.cluster.local
  namespace: my-app-1
spec:
  meshName: my-mesh
  virtualRouter:
    name: my-service-a-virtual-router
  listeners:
  - portMapping:
      port: 9080
        protocol: http
  routes:
  - name: my-service-a-route
    http:
      match:
        prefix: /
      action:
        weightedTargets:
        - virtualNodeName: my-service-a
          weight: 1
```

To see all available settings for a virtual service, virtual router, and route that you can set in the preceding spec, run any of the following commands.

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

b. Create the virtual service.

```bash
kubectl apply -f virtual-service.yaml
```
c. View the virtual service resource.

```
kubectl describe virtualservice my-service-a.my-app-1.svc.cluster.local -n my-app-1
```

Abbreviated output

```
Name: my-service-a.my-app-1.svc.cluster.local
Namespace: my-app-1
Labels: <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
    "apiVersion":"appmesh.k8s.aws/v1beta1","kind":"VirtualService","metadata":{"annotations":{},"name":"my-service-a.my-app-1.svc.cluster.loc..."
API Version: appmesh.k8s.aws/v1beta1
Kind: VirtualService
Spec:
    Mesh Name: my-mesh
    Routes:
        Http:
            Action:
                Weighted Targets:
                    Virtual Node Name: my-service-a
                    Weight: 1
            Match:
                Prefix: /
                Name: my-service-a-route
    Virtual Router:
        Listeners:
            Port Mapping:
                Port: 9080
                Protocol: http
        Name: my-service-a-virtual-router
Status:
    Conditions:
        Last Transition Time: 2020-03-20T13:24:37Z
        Status: True
        Type: VirtualRouterActive
        Last Transition Time: 2020-03-20T13:24:37Z
        Status: True
        Type: RoutesActive
        Last Transition Time: 2020-03-20T13:24:38Z
        Status: True
        Type: VirtualServiceActive
Events: <none>
```

d. Confirm that the virtual service was created in your 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-app-1.svc.cluster.local --mesh-name=mesh=mesh=mesh
```

Output

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

37
e. Confirm that the virtual router was created in your mesh.

**Note**

Though the virtual router created in Kubernetes is `my-service-a-virtual-router`, the name of the virtual router created in App Mesh is `my-service-a-virtual-router-my-app-1`.

```bash
aws appmesh describe-virtual-router --virtual-router-name my-service-a-virtual-router-my-app-1 --mesh-name=my-mesh
```

**Output**

```json
{
  "virtualRouter": {
    "meshName": "my-mesh",
    "metadata": {
      "createdAt": "2020-03-20T08:24:37.285000-05:00",
      "lastUpdatedAt": "2020-03-20T08:24:37.285000-05:00",
      "meshOwner": "111122223333",
      "resourceOwner": "111122223333",
      "uid": "111a11b1-c11d-1e1f-gh11-j11k11l11m711",
      "version": 1
    },
    "spec": {
      "listeners": [
        {
          "portMapping": {
            "port": 9080,
            "protocol": "http"
          }
        }
      ],
      "status": {
        "status": "ACTIVE"
      },
      "virtualRouterName": "my-service-a-virtual-router-my-app-1"
    }
  }
}
```
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 into a namespace that you specify.

To create or update services

1. To enable sidecar injection for the namespace, label the namespace.

   ```
kubectl label namespace my-app-1 appmesh.k8s.aws/sidecarInjectorWebhook=enabled
   ```

2. Enable proxy authorization. We recommend that you enable each Kubernetes deployment to stream the configuration for its own App Mesh virtual node.
a. Create a file named `proxy-auth.json` with the following contents.

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

b. Create the policy.

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

Note the ARN of the policy in the output returned. You'll use it in the next step.

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.

```bash
eksctl create iamserviceaccount \
   --cluster $CLUSTER_NAME \
   --namespace my-app-1 \
   --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. 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.

3. Create a Kubernetes service and deployment. If you have an existing deployment that you want to use with App Mesh, you need to update its namespace to `my-app-1` so that the sidecar containers are automatically added to the pods and the pods are redeployed.

a. Create a file named `example-service.yaml` with the following contents.

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-a
  namespace: my-app-1
---
apiVersion: v1
kind: Service
metadata:
  name: my-service-a
  namespace: my-app-1
labels:
  app: nginx
spec:
  selector:
```
app: nginx
ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
---
apIVersion: apps/v1
kind: Deployment
metadata:
  name: my-service-a
  namespace: my-app-1
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      serviceAccountName: my-service-a
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
You can override the default behavior of the injector for individual pods. For example, notice in the preceding spec that the name of the deployment is my-service-a. By default, this name must be the same as the name of the virtual node that you created in the section called “Step 2: Deploy App Mesh resources” (p. 34). If you want the name of the virtual node to be different than the name of the deployment, then you must add an annotation to your spec for the virtualNode setting. To familiarize yourself with the settings that you can override, see Default behavior and how to override on GitHub.

b. Deploy the service.

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

c. View the service and deployment.

```bash
kubectl -n my-app-1 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-658c47c864-g9p9l</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>26s</td>
</tr>
<tr>
<td>my-service-a-658c47c864-grn8w</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>26s</td>
</tr>
<tr>
<td>my-service-a-658c47c864-1c5q6</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.

```bash
kubectl -n my-app-1 describe pod my-service-a-7fd6966748-79674
```

Abbreviated output

| Name: my-service-a-658c47c864-g9p9l |
Namespace:  my-app-1

...  
Init Containers:  
  proxyinit:  
    Container ID:  docker://5f45b51566681be12eb851d48a47199e5a09d8661a16296bca826c93d3703ae6  
    Image ID:  docker-pullable://111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-proxy-route-manager@sha256:1111111111111111111111111111111111111111111111111111111111111111  
    Port:  <none>  
    Host Port:  <none>  
    State:  Terminated  
    Reason:  Completed  
    Exit Code:  0  
    Started:  Fri, 20 Mar 2020 08:37:05 -0500  
    Finished:  Fri, 20 Mar 2020 08:37:06 -0500  
    Ready:  True  
    Restart Count:  0  
    Requests:  
      cpu:  10m  
      memory:  32Mi  
    Environment:  
      APPMESH_START_ENABLED:  1  
      APPMESH_IGNORE_UID:  1337  
      APPMESH_ENVOY_INGRESS_PORT:  15000  
      APPMESH_ENVOY_EGRESS_PORT:  15001  
      APPMESH_APP_PORTS:  80  
      APPMESH_EGRESS_IGNORED_IP:  169.254.169.254  
      APPMESH_EGRESS_IGNORED_PORTS:  22  
      AWS_ROLE_ARN:  arn:aws:iam::111122223333:role/eksctl-appmesh-1-addon-iamserviceaccount-my-Role1-MVK5HIHHL4WT  
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/serviceaccount/token  
    Mounts:  
      /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)  
      /var/run/secrets/kubernetes.io/serviceaccount from my-service-a-token-s29ls (ro)  

Containers:  
  nginx:  
    Container ID:  docker://022c01578fa711129213aa0de9df0748ff410f468739edc872bb1888843020bd88  
    Image:  nginx:1.14.2  
    Image ID:  docker-pullable://nginx@sha256:1111111111111111111111111111111111111111111111111111111111111111  
    Port:  80/TCP  
    Host Port:  0/TCP  
    State:  Running  
    Started:  Fri, 20 Mar 2020 08:37:11 -0500  
    Ready:  True  
    Restart Count:  0  
    Requests:  
      cpu:  10m  
      memory:  32Mi  
    Environment:  
      AWS_ROLE_ARN:  arn:aws:iam::111122223333:role/eksctl-appmesh-1-addon-iamserviceaccount-my-Role1-MVK5HIHHL4WT  
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/serviceaccount/token  
    Mounts:  
      /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)  
      /var/run/secrets/kubernetes.io/serviceaccount from my-service-a-token-s29ls (ro)  

  envoy:  
    Container ID:  docker://742934a0722734762890ae925a4c143b739ba114ab33acbf14919c4069773f492  
    Image ID:  docker-pullable://840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy@sha256:1111111111111111111111111111111111111111111111111111111111111111  
    Port:  <none>  
    Host Port:  <none>  
    State:  Terminated  
    Reason:  Completed  
    Exit Code:  0  
    Started:  Fri, 20 Mar 2020 08:37:05 -0500  
    Finished:  Fri, 20 Mar 2020 08:37:06 -0500  
    Ready:  True  
    Restart Count:  0  
    Requests:  
      cpu:  10m  
      memory:  32Mi  
    Environment:  
      APPMESH_START_ENABLED:  1  
      APPMESH_IGNORE_UID:  1337  
      APPMESH_ENVOY_INGRESS_PORT:  15000  
      APPMESH_ENVOY_EGRESS_PORT:  15001  
      APPMESH_APP_PORTS:  80  
      APPMESH_EGRESS_IGNORED_IP:  169.254.169.254  
      APPMESH_EGRESS_IGNORED_PORTS:  22  
      AWS_ROLE_ARN:  arn:aws:iam::111122223333:role/eksctl-appmesh-1-addon-iamserviceaccount-my-Role1-MVK5HIHHL4WT  
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/serviceaccount/token  
    Mounts:  
      /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)  
      /var/run/secrets/kubernetes.io/serviceaccount from my-service-a-token-s29ls (ro)
<table>
<thead>
<tr>
<th>Image ID:</th>
<th>docker-pullable://840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy@sha256:1111111111111111111111111111111111111111111111111111111111111111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port:</td>
<td>9901/TCP</td>
</tr>
<tr>
<td>Host Port:</td>
<td>0/TCP</td>
</tr>
<tr>
<td>State:</td>
<td>Running</td>
</tr>
<tr>
<td>Started:</td>
<td>Fri, 20 Mar 2020 08:37:18 -0500</td>
</tr>
<tr>
<td>Ready:</td>
<td>True</td>
</tr>
<tr>
<td>Restart Count:</td>
<td>0</td>
</tr>
<tr>
<td>Requests:</td>
<td></td>
</tr>
<tr>
<td>cpu:</td>
<td>10m</td>
</tr>
<tr>
<td>memory:</td>
<td>32Mi</td>
</tr>
<tr>
<td>Environment:</td>
<td>APPMESH_VIRTUAL_NODE_NAME: mesh/my-mesh/virtualNode/my-service-a-my-app-1</td>
</tr>
<tr>
<td></td>
<td>APPMESH_PREVIEW: 0</td>
</tr>
<tr>
<td></td>
<td>ENVOY_LOG_LEVEL: info</td>
</tr>
<tr>
<td></td>
<td>AWS_REGION: us-west-2</td>
</tr>
<tr>
<td></td>
<td>AWS_ROLE_ARN: arn:aws:iam::111122223333:role/eksctl-appmesh-l-addon-iamserviceaccount-my-Role1-MVK5HIHHL4WT</td>
</tr>
<tr>
<td></td>
<td>AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/serviceaccount/token</td>
</tr>
<tr>
<td></td>
<td>serviceaccount/token</td>
</tr>
<tr>
<td>Mounts:</td>
<td>/var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)</td>
</tr>
<tr>
<td></td>
<td>/var/run/secrets/kubernetes.io/serviceaccount from my-service-a-token-s291s (ro)</td>
</tr>
</tbody>
</table>

In the preceding output, you can see that the proxyinit and envoy containers were added to the pod.

### Step 4: Clean up

Remove all of the example resources created in this tutorial. The controller also removes the resources that were created in App Mesh.

```
kubectl delete namespace my-app-1
```

(Optional) You can remove the Kubernetes integration components.

```
helm delete --purge appmesh-controller
helm delete --purge appmesh-inject
```
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 virtual nodes. 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.

To use the App Mesh container image, specify one of the following addresses, depending on the Region that you want to pull the image from.

- All supported Regions other than me-south-1 and ap-east-1. You can replace us-west-2 with any Region other than me-south-1 and ap-east-1.
  

- me-south-1 Region:
  
  772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.12.2.1-prod

- ap-east-1 Region:
  
  856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.12.2.1-prod

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.

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.

APPMESH_VIRTUAL_NODE_NAME

When you add the Envoy container to a task group, set this environment variable to the name of the virtual node that the task group represents: for example, 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

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.

XRAY_DAEMON_PORT

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

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.
STATSD_PORT

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

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.

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.

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.

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 above image shows a logging path of /dev/stdout for Envoy access logs. The code block below shows the JSON representation that you could 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, so you can export them to a log storage and processing service like CloudWatch Logs using standard Docker log drivers (such as awslogs). 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.
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.

**Topics**

- Data Protection in AWS App Mesh (p. 47)
- Identity and Access Management for AWS App Mesh (p. 48)
- Logging App Mesh API Calls with AWS CloudTrail (p. 64)
- Compliance Validation for AWS App Mesh (p. 66)
- Resilience in AWS App Mesh (p. 66)
- Infrastructure Security in App Mesh (p. 67)
- Configuration and Vulnerability Analysis in AWS App Mesh (p. 68)

Data Protection in AWS App Mesh

AWS App Mesh conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.
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), so that 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.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.

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.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

## 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.

## Identity and Access Management for AWS App Mesh

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. 49)
- Authenticating With Identities (p. 49)
- Managing Access Using Policies (p. 51)
- How AWS App Mesh Works with IAM (p. 52)
- AWS App Mesh Identity-Based Policy Examples (p. 55)
Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work 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. 57).

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. 52).

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. 55).

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 The IAM Console and Sign-in Page 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 or your IAM user name. You can access AWS programmatically using your root user or IAM user 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.

- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.
Applications running on Amazon EC2 – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the IAM User Guide.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the IAM User Guide.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the IAM User Guide.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. 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, role, or group. These policies control what actions that identity 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 such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access Control Lists (ACLs)

Access control lists (ACLs) are a type of policy that controls 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. 52)
- App Mesh Resource-Based Policies (p. 54)
- Authorization Based on App Mesh Tags (p. 54)
- App Mesh IAM Roles (p. 54)

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

The Action element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used 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.

```json
"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.

```json
"Action": "appmesh:Describe*"
```

To see a list of App Mesh actions, see Actions Defined by AWS App Mesh in the IAM User Guide.

Resources

The Resource element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

The App Mesh mesh resource has the following ARN.

```
arn:#{Partition}:appmesh:#{Region}:#{Account}:mesh/#{MeshName}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the mesh named `apps` in the `region-code` Region in your statement, use the following ARN.

```
arn:aws:appmesh:region-code:1112223333:mesh/apps
```

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. 55).

**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. 57).

**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. 59).

### 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",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:GetUser"
      ],
      "Resource": "*
    }
  ]
}
```

Create a mesh

This example shows how you can create a policy that allows a user to create a mesh for an account, in any Region.

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

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

Creating App Mesh Meshes With Restricted Tags

You can use tags in your IAM policies to control what tags can be passed in the IAM request. You can specify which tag key-value pairs can be added, changed, or removed from an IAM user or role. This example shows how you might create a policy that allows creating a mesh, but only if the mesh is created with a tag named `teamName` and a value of `booksTeam`.

```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**.

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.
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, wJalrXUttnFEMI/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. 52).
- 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.

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. 27) defined for all routers in the mesh.
2. Delete all virtual routers (p. 25) in the mesh.
3. Delete all virtual services (p. 19) in the mesh.
4. Delete all virtual nodes (p. 20) in the mesh.
5. Delete the mesh (p. 18).

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.
Proxy Authorization

Proxy authorization authorizes the Envoy (p. 44) 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 virtual nodes from the App Mesh Envoy Management Service. An example of a configuration that the proxy can read from a virtual node is the private certificate of a virtual node that has Transport Layer Security (TLS) (p. 22) enabled. 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 your microservice on.

Create IAM Policy

If you want all virtual nodes in a service mesh to be able to read the configuration for all virtual nodes, skip to the section called “Create IAM role” (p. 61). If you want to limit the virtual nodes that configuration can be read from by individual virtual nodes, then you need to create one or more IAM policies. Limiting the virtual nodes 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 virtual nodes.

Create IAM role

If you want all virtual nodes in a service mesh to be able to read the configuration for all virtual nodes, you only need to create one IAM role. If you want to limit the virtual nodes that configuration can be read from by individual virtual nodes, 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 your service 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. Create the IAM roles and associate the roles with Kubernetes service accounts. For more information, see Creating an IAM Role and Policy for your Service Account and Specifying an IAM Role for your Service Account.
- **Amazon ECS** – Select AWS service, select Elastic Container Service, and then select the Elastic Container Service Task use case when creating your IAM role.
- **Amazon EC2** – Select AWS service, select EC2, and then select the EC2 use case when creating your IAM role. This applies whether you host your microservices directly on an Amazon EC2 instance or on Kubernetes running on an instance.
For more information about how to create an IAM role, see Creating a Role for an AWS Service.

**Attach IAM Policy**

If you want all virtual nodes in a service mesh to be able to read the configuration for all virtual nodes, then attach the `AWSAppMeshEnvoyAccess` managed IAM policy to the IAM role that you created in a previous step. If you want to limit the virtual nodes that configuration can be read from by individual virtual nodes, then attach each policy that you created to each role that you created. For more information about attaching a custom or managed IAM policy to an IAM role, see Adding IAM Identity Permissions.

**Attach IAM Role**

Attach each IAM role to the appropriate compute resource:

- **Amazon EKS** – If you attached the policy to the role attached to your worker nodes, you can skip this step. If you created separate roles, then assign each role to a separate Kubernetes service account, and assign each service account to an individual Kubernetes pod deployment spec that includes the Envoy proxy. For more information, see Specifying an IAM Role for your Service Account in the Amazon EKS User Guide and Configure Service Accounts for Pods in the Kubernetes documentation.

- **Amazon ECS** – Attach an Amazon ECS Task Role to the task definition that includes the Envoy proxy. The task can be deployed with the EC2 or Fargate launch type. For more information about how to create an Amazon ECS Task Role and attach it to a task, see Specifying an IAM Role for your Tasks.

- **Amazon EC2** – The IAM role must be attached to the Amazon EC2 instance that hosts the Envoy proxy. For more information about how to attach a role to an Amazon EC2 instance, see I've created an IAM role, and now I want to assign it to an EC2 instance.

**Confirm permission**

Confirm that the `appmesh:StreamAggregatedResources` permission is assigned to the compute resource that you host your microservice 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 virtual nodes. If the policy is attached to the role assigned to the worker nodes, select the **Amazon EC2** tab, and complete the steps found there for your worker node instances. To determine which IAM role is assigned to a Kubernetes pod, complete the following steps.

1. View the details of a Kubernetes deployment that includes the pod that you want to confirm that a Kubernetes service account is assigned to. The following command views the details for a deployment named `my-deployment`.

   ```
   kubectl describe deployment my-deployment
   ```

   In the returned output note the value to the right of **Service Account**:. If a line that starts with **Service Account**: doesn't exist, then a custom Kubernetes service account isn't currently assigned to the deployment. You'll need to assign one. For more information, see Configure Service Accounts for Pods in the Kubernetes documentation.

2. View the details of the service account returned in the previous step. The following command views the details of a service account named `my-service-account`.

   ```
   kubectl describe serviceaccount my-service-account
   ```
Provided the Kubernetes service account is associated to an AWS Identity and Access Management role, one of the lines returned will look similar to the following example.

```
Annotations:        eks.amazonaws.com/role-arn=arn:aws:iam::123456789012:role/my-deployment
```

In the previous example `my-deployment` is the name of the IAM role that the service account is associated with. If the service account output doesn’t contain a line similar to the example above, then the Kubernetes service account isn’t associated to an AWS Identity and Access Management account and you need to associate it to one. For more information, see Specifying an IAM Role for your Service Account.

3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
4. In the left navigation, select Roles. Select the name of the IAM role that you noted in a previous step.
5. Confirm that either the custom policy you created previously, or the AWSAppMeshEnvoyAccess managed policy is listed. If neither policy is attached, attach an IAM policy (p. 62) 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. 61) 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 virtual node 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 the section called “Create IAM Policy” (p. 61).
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. 61) 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. 62) 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. 61). 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 virtual node 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 the section called “Create IAM Policy” (p. 61).
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. 61)

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**. If you want to attach a custom IAM policy but don't have one, then you need to **create the custom IAM policy**. (p. 61). 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 virtual node 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 the section called “Create IAM Policy”**. (p. 61)

5. Repeat the previous steps for each instance that you host the Envoy proxy on.

---

**Logging App Mesh API Calls 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](#).

---

**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](#).

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**
- **CloudTrail Supported Services and Integrations**
- **Configuring Amazon SNS Notifications for CloudTrail**
- **Receiving CloudTrail Log Files from Multiple Regions** and **Receiving CloudTrail Log Files from Multiple Accounts**

All App Mesh actions are logged by CloudTrail and are documented in **API Actions**. 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.

**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",
        "userId": "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": "00000000-0000-0000-0000-000000000000",
        "spec": {
            "egressFilter": {
                "type": "DROP_ALL"
            }
        }
    },
    "responseElements": {
        "mesh": {
            "meshName": "my-mesh",
            "status": {
                "status": "ACTIVE"
            },
            "metadata": {
                "version": 1,
                "lastUpdatedDate": "Oct 18, 2019 3:00:49 PM",
                "uid": "00000000-0000-0000-0000-000000000000"
            }
        }
    }
}
```
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).

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.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

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 Whitepaper** – This whitepaper 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.

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.

**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* whitepaper.

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. 67).

**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](p. 67).

**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](p. 67) 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](p. 67) 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['
```

Configuration and Vulnerability Analysis in AWS App Mesh

App Mesh vends a managed Envoy proxy Docker container image (p. 44) 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.

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.
- When an owner or consumer creates or updates a resource that references other AWS resources, such as AWS Cloud Map or AWS Certificate Manager, the only resources that can be referenced are resources in the same account. For an example of a virtual node that references a TLS certificate, see the TLS encryption (p. 22) topic in this guide.
- 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. 20).
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 ??? (p. 18).

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 ???? (p. 78). The AWS Resource Access Manager service also has quotas. For more information, see Service Quotas.
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 Service Endpoints 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.
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. 75) 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>
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. 75). 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. 27) 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. 75).

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.

```
aws service-quotas list-aws-default-service-quotas
   --query 'Quotas[*].{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}'
   --service-code appmesh
   --output table
```

To work more with service quotas using the AWS CLI, see the Service Quotas AWS CLI Command Reference.
## Document History for App Mesh

The following table describes the major updates and new features for the *AWS App Mesh User Guide*. We also update the documentation frequently to address the feedback that you send us.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS encryption</td>
<td><em>(App Mesh Preview Channel only)</em> 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><em>(App Mesh Preview Channel only)</em> 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><em>(App Mesh Preview Channel only)</em> 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><em>(App Mesh Preview Channel only)</em> 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</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>App Mesh Interface VPC Endpoints (AWS PrivateLink)</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>Added AWS Cloud Map as a virtual node service discovery method</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>Create App Mesh resources automatically in Kubernetes</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>App Mesh General Availability (p. 79)</td>
<td>The App Mesh service is now generally available for production use.</td>
<td>March 27, 2019</td>
</tr>
<tr>
<td>App Mesh API update (p. 79)</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>App Mesh initial release (p. 79)</td>
<td>Initial documentation for service public preview</td>
<td>November 28, 2018</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.