AWS App Mesh

User Guide

AWS App Mesh: User Guide

Table of Contents

What Is AWS App Mesh?

AWS App Mesh is a service mesh based on the Envoy proxy that makes it easy to monitor and control microservices. App Mesh standardizes how your microservices 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 microservice in an application.

App Mesh supports microservice applications that use service discovery naming for their components. To use App Mesh, you must have an existing application running on AWS Fargate, Amazon ECS, Amazon EKS, Kubernetes on AWS, or Amazon EC2.

Components of App Mesh

App Mesh is made up of the following components:

- **Service mesh** – A service mesh is a logical boundary for network traffic between the services that reside within it. For more information, see Service Meshes (p. 19).
- **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. For more information, see Virtual Services (p. 20).
- **Virtual nodes** – A virtual node acts as a logical pointer to a particular task group, such as an ECS service or a Kubernetes deployment. When you create a virtual node, you must specify the DNS service discovery hostname for your task group. For more information, see Virtual Nodes (p. 22).
- **Envoy proxy** – The Envoy proxy configures your microservice task group to use the App Mesh service mesh traffic rules that you set up for your virtual routers and virtual nodes. You add the Envoy container to your task group after you have created your virtual nodes, virtual routers, routes, and virtual services. For more information, see Envoy Image (p. 25).
- **Virtual routers** – The virtual router handles traffic for one or more virtual services within your mesh. For more information, see Virtual Routers (p. 28).
- **Routes** – A route is associated with a virtual router, and it directs traffic that matches a service name prefix to one or more virtual nodes. For more information, see Routes (p. 30).

How to Get Started

App Mesh supports microservice applications that use service discovery naming for their components. To use App Mesh, you must have an existing application running on AWS Fargate, Amazon ECS, Amazon EKS, Kubernetes on AWS, or Amazon EC2.

For more information about service discovery on Amazon ECS, see Service Discovery in the *Amazon Elastic Container Service Developer Guide*. Kubernetes `kube-dns` is supported. For more information, see DNS for Services and Pods in the Kubernetes documentation.

To get started using the App Mesh console first-run wizard, see Getting Started with AWS App Mesh (p. 3).

Accessing App Mesh

You can work with App Mesh in the following ways:

AWS Management Console

The console is a browser-based interface to manage App Mesh resources. For a tutorial that guides you through the console, see Getting Started with AWS App Mesh (p. 3).

AWS command line tools

You can use the AWS command line tools to issue commands at your system's command line to perform App Mesh and AWS tasks. This can be faster and more convenient than using the console. The command line tools are also useful for building scripts that perform AWS tasks.

AWS provides two sets of command line tools: the AWS Command Line Interface (AWS CLI) and the AWS Tools for Windows PowerShell. For more information, see the AWS Command Line Interface User Guide and the AWS Tools for Windows PowerShell User Guide.

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.

Getting Started with AWS App Mesh

This topic helps you to get started using AWS App Mesh with the AWS Management Console. This getting-started guide helps you to create one of each component that is available in App Mesh. When you're finished with the wizard, you can create additional resources in the AWS Management Console to represent the other services that you want to contain in the mesh.

Topics

Prerequisites

App Mesh supports microservice applications that use service discovery naming for their components. To use App Mesh, you must have an existing application running on AWS Fargate, Amazon ECS, Amazon EKS, Kubernetes on AWS, or Amazon EC2.

For more information about service discovery on Amazon ECS, see Service Discovery in the *Amazon Elastic Container Service Developer Guide*. Kubernetes `kube-dns` and `coredns` are supported. For more information, see DNS for Services and Pods in the Kubernetes documentation.

Step 1: Create a Mesh and Virtual Service

To begin, you must first create a service mesh and a virtual service for one of the microservices in your application.

A service mesh is a logical boundary for network traffic between the services that reside within it. A virtual service represents a service that can be used by virtual nodes in the mesh. Dependent nodes call your virtual service by its virtual service name, and those requests are routed to the virtual node or virtual router that is specified as the provider for the virtual service.

To create a new service mesh and virtual service

1. Open the App Mesh console first-run wizard at https://console.aws.amazon.com/appmesh/get-started.
2. For **Mesh name**, choose a name for your service mesh.
3. For **Virtual service name**, choose a name for your virtual service. We recommend that you use the service discovery name of the real service that you're targeting (such as `my-service.default.svc.cluster.local`).
4. Choose **Next** to proceed.

Step 2: Configure a Virtual Node

In this step, you configure a virtual node to be the end destination for requests that are made to the virtual service that you created earlier.

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 the DNS service discovery hostname for your task group.

Any inbound traffic that your virtual node expects should be specified as a *listener*; you can optionally configure health checks for your virtual node listeners. Any outbound traffic that your virtual node expects to reach should be specified as a *backend*.

To configure a virtual node

1. For **Virtual node name**, choose a name for your virtual node.
2. For **Service discovery method**, choose **DNS** for services that use DNS service discovery and then specify the hostname for **DNS hostname**. Otherwise, choose **None** if your virtual node doesn't expect any ingress traffic.
3. To specify any backends (for egress traffic) for your virtual node, or to configure inbound and outbound access logging information, choose **Additional configuration**.

    a. To specify a backend, choose **Add backend** and enter a virtual service name or full Amazon Resource Name (ARN) for the virtual service that your virtual node communicates with. Repeat this step until all of your virtual node backends are accounted for.

    b. 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.

4. If your virtual node expects ingress traffic, specify a **Port** and **Protocol** for that **Listener**.
5. If you want to configure health checks for your listener, ensure that **Health check enabled** is selected and then complete the following substeps. If not, clear this check box.

    a. For **Health check protocol**, choose to use an HTTP or TCP health check.

    b. For **Health check port**, specify the port that the health check should run on.

    c. For **Healthy threshold**, specify the number of consecutive successful health checks that must occur before declaring the listener healthy.

    d. For **Health check interval**, specify the time period in milliseconds between each health check execution.

    e. For **Path**, specify the destination path for the health check request. This is required only if the specified protocol is HTTP. If the protocol is TCP, this parameter is ignored.

    f. For **Timeout period**, specify the amount of time to wait when receiving a response from the health check, in milliseconds.

    g. For **Unhealthy threshold**, specify the number of consecutive failed health checks that must occur before declaring the listener unhealthy.

6. Chose **Next** to continue.

Step 3: Configure a Virtual Router and Route

In this step, you configure a virtual router to handle requests that are made to the virtual service that you created earlier. You also create a route that is used to match requests and distribute traffic accordingly to its associated virtual node.

To configure a virtual router and route

1. For **Virtual router name**, specify a name for your virtual router.
2. For **Listener**, specify a **Port** and **Protocol** for your virtual router.
3. For **Route name**, specify the name to use for your route.
4. For **Route type**, choose the protocol for your route.
5. For **Virtual node name**, choose the virtual node that this route will serve traffic to.
6. For **Weight**, choose a relative weight for the route. The total weight for all routes must be less than 100.
7. To use HTTP path-based routing, choose **Additional configuration** and then specify the path that the route should match. For example, if your virtual service name is `my-service.local` and you want the route to match requests to `my-service.local/metrics`, your prefix should be `/metrics`.
8. Choose **Next** to proceed.

Step 4: Review and Create

Review your information and choose **Create mesh service** to finish. You can now view each of your App Mesh resources in the console.

Step 5: Create Your Remaining App Mesh Resources

Currently, you should have one of each App Mesh resource for your application. Now you must create the remaining resources for your application.

To create your remaining App Mesh resources

1. Create virtual nodes for each remaining microservice in your application. For more information, see Creating a Virtual Node (p. 22).
2. Create virtual routers for each remaining microservice in your application. For more information, see Creating a Virtual Router (p. 28).
3. Create routes for each remaining microservice in your application. For more information, see Creating a Route (p. 30).
4. Create virtual services for each remaining microservice in your application. For more information, see Creating a Virtual Service (p. 20).

Step 6: Update Your Microservices

App Mesh is a service mesh based on the Envoy proxy. After you create your service mesh, virtual nodes, virtual routers, routes, and virtual services, you must update your microservices to be compatible with App Mesh.

To update your existing microservice application that is running on Amazon ECS, see Getting Started with AWS App Mesh and Amazon ECS (p. 7).

To update your existing microservice application that is running on Kubernetes, see Getting Started with AWS App Mesh and Kubernetes (p. 13).

To update your existing microservice application that is running on Amazon EC2, see Getting Started with AWS App Mesh and Amazon EC2 (p. 15).

Getting Started with AWS App Mesh and Amazon ECS

This topic helps you to use AWS App Mesh with an existing set of microservice applications running on Amazon ECS.

Prerequisites

App Mesh supports microservice applications that use service discovery naming for their components. For more information about service discovery on Amazon ECS, see Service Discovery in the *Amazon Elastic Container Service Developer Guide*.

To use this getting started guide, you must have a microservice application running on Amazon ECS. You must also have the following App Mesh resources, that you can create by completing the steps in the Getting Started with AWS App Mesh (p. 3) guide:

- A service mesh
- Virtual nodes for each microservice in your application
- Virtual routers and routes for each microservice in your application (except for virtual services that are provided by a virtual node directly)
- Virtual services for each microservice in your application

Update Your Microservice Task Definitions

App Mesh is a service mesh based on the Envoy proxy. After you create your service mesh, virtual nodes, virtual routers, routes, and virtual services, you must update the Amazon ECS task definitions for your microservices to be compatible with App Mesh. Complete the steps in the following sections to update your services' task definitions to work with App Mesh.

Proxy Configuration

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

`IgnoredUID`

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

`ProxyIngressPort`

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

`ProxyEgressPort`

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

`AppPorts`

Specify any ingress ports that your application containers listen on. In this example, the application container listens on port `9080`.

`EgressIgnoredIPs`

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

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

Application Container Envoy Dependency

The application containers in your task definitions must wait for the Envoy proxy to bootstrap and start before they can start. To ensure that this happens, you set a `dependsOn` section in each application container definition to wait for the Envoy container to report as `HEALTHY`. The following code block shows an application container definition example with this dependency.

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

```
}
```

Envoy Container Definition

Your Amazon ECS task definitions or Kubernetes pod specs must contain the App Mesh Envoy container image (p. 25):

- `111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-prod`

The Envoy container definition must be marked as `essential`. The virtual node name for the Amazon ECS service should be set to the `APPMESH_VIRTUAL_NODE_NAME`, and the `user` ID value should match the `IgnoredUID` value from the task definition proxy configuration (in this example, we use `1337`). The health check shown here waits for the Envoy container to bootstrap properly before reporting to Amazon ECS that it is healthy and ready for the application containers to start.

The following code block shows an Envoy container definition example.

```
{
 "name": "envoy",
 "image": "111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-prod",
 "essential": true,
 "environment": [{
  "name": "APPMESH_VIRTUAL_NODE_NAME",
  "value": "mesh/meshName/virtualNode/virtualNodeName"
 }],
 "healthCheck": {
  "command": [
   "CMD-SHELL",
   "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
  ],
  "startPeriod": 10,
  "interval": 5,
  "timeout": 2,
  "retries": 3
 },
 "user": "1337"
}
```

Credentials

The Envoy container requires AWS Identity and Access Management credentials for signing requests that are sent to the App Mesh service. For Amazon ECS tasks deployed with the Amazon EC2 launch type, the credentials can come from the instance IAM role or from a task IAM role. Amazon ECS tasks deployed with the Fargate launch type do not have access to the Amazon EC2 metadata server that supplies instance IAM profile credentials. To supply the credentials, you must attach an IAM task role to any tasks deployed with the Fargate launch type. The role doesn't need to have a policy attached to it, but for a task to work properly with App Mesh, the role must be attached to each task deployed with the Fargate launch type. If a task is deployed with the Amazon EC2 launch type and access is blocked to the Amazon EC2 metadata server, as described in the *Important* annotation in IAM Roles for Tasks, then a task IAM role must also be attached to the task.

Example Task Definitions

The following example Amazon ECS task definitions show, in context, the snippets that you can merge with your existing task groups. Substitute your mesh name and virtual node name for the `APPMESH_VIRTUAL_NODE_NAME` value and a list of ports that your application listens on for the proxy configuration `AppPorts` value.

If you're running an Amazon ECS task as described in the section called "Credentials" (p. 9), you need an existing task IAM role. You should also add this line of code to the example task definitions that follow: `"taskRoleArn": "arn:aws:iam::123456789012:role/ecsTaskRole"`

Example JSON for Amazon ECS task definition

```
{
  "family": "appmesh-gateway",
  "memory": "256",
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "envoy",
    "properties": [
      {
        "name": "IgnoredUID",
        "value": "1337"
      },
      {
        "name": "ProxyIngressPort",
        "value": "15000"
      },
      {
        "name": "ProxyEgressPort",
        "value": "15001"
      },
      {
        "name": "AppPorts",
        "value": "9080"
      },
      {
        "name": "EgressIgnoredIPs",
        "value": "169.254.170.2,169.254.169.254"
      }
    ]
  },
  "containerDefinitions": [
    {
      "name": "app",
      "image": "application_image",
      "portMappings": [
        {
          "containerPort": 9080,
          "hostPort": 9080,
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "dependsOn": [
        {
          "containerName": "envoy",
          "condition": "HEALTHY"
        }
      ]
    },
    {
      "name": "envoy",
      "image": "111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-
prod",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/meshName/virtualNode/virtualNodeName"
        }
      ],
```

```
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
        ],
        "startPeriod": 10,
        "interval": 5,
        "timeout": 2,
        "retries": 3
      },
      "user": "1337"
    }
  ],
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

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

X-Ray allows you to collect data about requests that an application serves and provides tools that you can use to visualize traffic flow. Using the X-Ray driver for Envoy enables Envoy to report tracing information to X-Ray. You can enable X-Ray tracing using the Envoy configuration (p. 25). Based on the configuration, Envoy sends tracing data to the X-Ray daemon running as a sidecar container and the daemon forwards the traces to the X-Ray service. Once the traces are published to X-Ray, you can use the X-Ray console to visualize the service call graph and request trace details. The following JSON represents a task definition to enable X-Ray integration.

```
{
  "family": "appmesh-gateway",
  "memory": "256",
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "envoy",
    "properties": [
      {
        "name": "IgnoredUID",
        "value": "1337"
      },
      {
        "name": "ProxyIngressPort",
        "value": "15000"
      },
      {
        "name": "ProxyEgressPort",
        "value": "15001"
      },
      {
        "name": "AppPorts",
        "value": "9080"
      },
      {
        "name": "EgressIgnoredIPs",
        "value": "169.254.170.2,169.254.169.254"
      }
    ]
  },
  "containerDefinitions": [
    {
      "name": "app",
      "image": "application_image",
      "portMappings": [
        {
          "containerPort": 9080,
          "hostPort": 9080,
```

```
          "protocol": "tcp"
        }
      ],
      "essential": true,
      "dependsOn": [
        {
          "containerName": "envoy",
          "condition": "HEALTHY"
        }
      ]
    },
    {
      "name": "envoy",
      "image": "111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-
prod",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/meshName/virtualNode/virtualNodeName"
        },
        {
         "name": "ENABLE_ENVOY_XRAY_TRACING",
         "value": "1"
        }
      ],
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
        ],
        "startPeriod": 10,
        "interval": 5,
        "timeout": 2,
        "retries": 3
      },
      "user": "1337"
    },
    {
      "name": "xray-daemon",
      "image": "amazon/aws-xray-daemon",
      "user": "1337",
      "essential": true,
      "cpu": 32,
      "memoryReservation": 256,
      "portMappings": [
        {
          "containerPort": 2000,
          "protocol": "udp"
        }
      ]
    }
  ],
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

Getting Started with AWS App Mesh and Kubernetes

This topic helps you to use AWS App Mesh with an existing microservice application running on Amazon EKS or Kubernetes on Amazon EC2.

Prerequisites

App Mesh supports microservice applications that use service discovery naming for their components. To use this getting started guide, you must have a microservice application running on Amazon EKS or Kubernetes on AWS.

Kubernetes `kube-dns` and `coredns` are supported. For more information, see DNS for Services and Pods in the Kubernetes documentation.

This guide also assumes that you have completed the Getting Started with AWS App Mesh (p. 3) guide and that you have the following App Mesh resources:

- A service mesh
- Virtual nodes for each microservice in your application
- Virtual routers and routes for each microservice in your application (except for virtual services that are provided by a virtual node directly)
- Virtual services for each microservice in your application

Updating Your Microservice Pod Specifications

App Mesh is a service mesh based on the Envoy proxy. After you create your service mesh, virtual nodes, virtual routers, and routes, you must update your microservices to be compatible with App Mesh.

App Mesh vends the following custom container images that you must add to your Kubernetes pod specifications.

- App Mesh Envoy container image: `111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-prod`
- App Mesh proxy route manager: `111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-proxy-route-manager:v2`

The following is an example Kubernetes pod specification that you can merge with your existing application. Substitute your mesh name and virtual node name for the `APPMESH_VIRTUAL_NODE_NAME` value, and a list of ports that your application listens on for the `APPMESH_APP_PORTS` value. Substitute the Amazon EC2 instance AWS Region for the `AWS_REGION` value.

Example Kubernetes pod spec

```
spec:
  containers:
    - name: envoy
```

```
      image: 111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-prod
      securityContext:
        runAsUser: 1337
      env:
        - name: "APPMESH_VIRTUAL_NODE_NAME"
          value: "mesh/meshName/virtualNode/virtualNodeName"
        - name: "ENVOY_LOG_LEVEL"
          value: "info"
        - name: "AWS_REGION"
          value: "aws_region_name"
  initContainers:
    - name: proxyinit
      image: 111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-proxy-route-
manager:v2
      securityContext:
        capabilities:
          add:
            - NET_ADMIN
      env:
        - name: "APPMESH_START_ENABLED"
          value: "1"
        - name: "APPMESH_IGNORE_UID"
          value: "1337"
        - name: "APPMESH_ENVOY_INGRESS_PORT"
          value: "15000"
        - name: "APPMESH_ENVOY_EGRESS_PORT"
          value: "15001"
        - name: "APPMESH_APP_PORTS"
          value: "application_port_list"
        - name: "APPMESH_EGRESS_IGNORED_IP"
          value: "169.254.169.254"
```

Getting Started with AWS App Mesh and Amazon EC2

This topic helps you to use AWS App Mesh with an existing microservice application running on Amazon EC2 instances.

Prerequisites

App Mesh supports microservice applications that use service discovery naming for their components. To use this getting started guide, you must have a microservice application running on Amazon EC2 instances.

This guide also assumes that you have completed the Getting Started with AWS App Mesh (p. 3) guide, and that you have the following App Mesh resources:

- A service mesh
- Virtual nodes for each microservice in your application
- Virtual routers and routes for each microservice in your application (except for virtual services that are provided by a virtual node directly)
- Virtual services for each microservice in your application

Configure Your Amazon EC2 Instances

AWS App Mesh is a service mesh based on the Envoy proxy. After you create your service mesh, virtual nodes, virtual routers, and routes, you must configure your Amazon EC2 instances to be compatible with App Mesh. The following procedure describes this process.

To configure an Amazon EC2 instance as a virtual node member

1. Launch an Amazon EC2 instance with an IAM role that allows read permissions from Amazon ECR. This is so that the instance can pull the App Mesh Envoy container image. 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 so that your Docker client can pull the container image.

```
$(aws ecr get-login --no-include-email --region us-west-2 --registry-ids 111345817488)
```

5. Run the following command to start the App Mesh Envoy container on your instance. Substitute your mesh name and virtual node name.

```
sudo docker run --detach --env APPMESH_VIRTUAL_NODE_NAME=mesh/meshName/
virtualNode/virtualNodeName  \
```

```
-u 1337 --network host 111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.9.1.0-prod
```

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

# Port numbers Application and Envoy are listening on.
if [ -z "$APPMESH_ENVOY_INGRESS_PORT" ] || [ -z "$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 \
        -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
                ;;
            *)
                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

This topic helps you to create a new service mesh.

To create a new service mesh with the AWS Management Console

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose **Create mesh**.
3. For **Mesh name**, specify a name for your service mesh.
4. Choose **Create mesh** to finish.

To create a new service mesh with the AWS CLI

- The following example AWS CLI command creates a service mesh named `simpleapp`.

```
aws appmesh create-mesh --mesh-name simpleapp
```

> **Note**
> You must use at least version 1.16.133 of the AWS CLI with App Mesh. To install the latest version of the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

For more information about creating service meshes with the AWS CLI, see create-mesh in the *AWS Command Line Interface User Guide*.

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

This topic helps you to create a virtual service in your service mesh.

Creating a virtual service in the AWS Management Console.

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh that you want to create the virtual service in.
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. We recommend that you use the service discovery name of the real service that you're targeting (such as `my-service.default.svc.cluster.local`).
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.

Creating a virtual service in the AWS CLI.

- The following JSON represents a virtual service named `serviceB` that is provided by the virtual router named `serviceB`.

```
{
    "meshName": "simpleapp",
    "spec": {
        "provider": {
            "virtualRouter": {
                "virtualRouterName": "serviceB"
            }
        }
    },
    "virtualServiceName": "serviceB"
}
```

  If you save the preceding JSON as a file, you can create the virtual service with the following AWS CLI command.

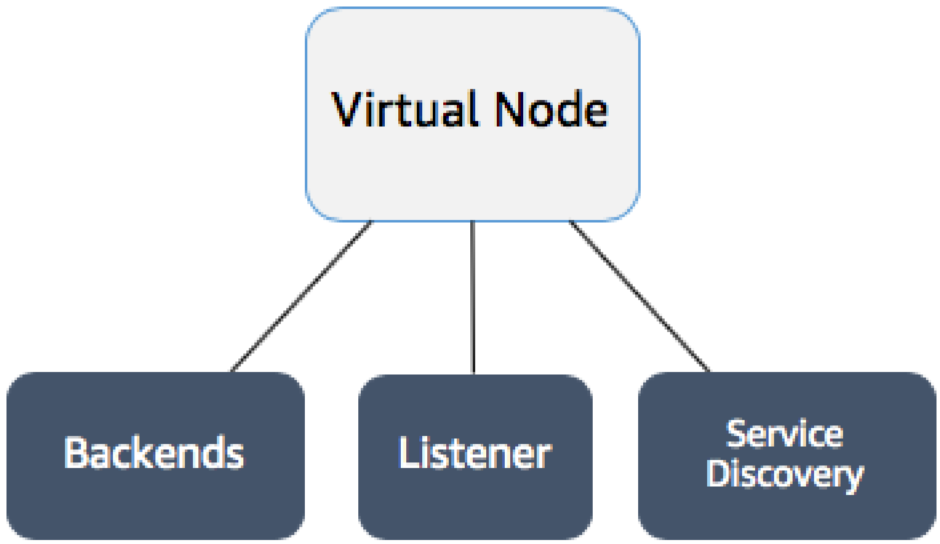
```
aws appmesh create-virtual-service --cli-input-json file://serviceB-virtual-
service.json
```

> **Note**
> You must use at least version 1.16.133 of the AWS CLI with App Mesh. To install the latest version of the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

For more information about creating virtual services with the AWS CLI, see create-virtual-service in the *AWS Command Line Interface User Guide*.

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 the DNS service discovery hostname for your task group. Any inbound traffic that your virtual node expects should be specified as a *listener*. Any outbound traffic that your virtual node expects to reach should be 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 task definition or 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

This topic helps you to create a virtual node in your service mesh.

To create a virtual node in the AWS Management Console.

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.

2. Choose **Virtual nodes** in the left navigation.
3. Choose **Create virtual node**.
4. For **Virtual node name**, choose a name for your virtual node.
5. For **Service discovery method**, choose **DNS** for services that use DNS service discovery and then specify the hostname for **DNS hostname**. Otherwise, choose **None** if your virtual node doesn't expect any ingress traffic.
6. To specify any backends (for egress traffic) for your virtual node, or to configure inbound and outbound access logging information, choose **Additional configuration**.

   a. To specify a backend, choose **Add backend** and enter a virtual service name or full Amazon Resource Name (ARN) for the virtual service that your virtual node communicates with. Repeat this step until all of your virtual node backends are accounted for.
   b. 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.
7. If your virtual node expects ingress traffic, specify a **Port** and **Protocol** for that **Listener**.
8. If you want to configure health checks for your listener, ensure that **Health check enabled** is selected and then complete the following substeps. If not, clear this check box.

   a. For **Health check protocol**, choose to use an HTTP or TCP health check.
   b. For **Health check port**, specify the port that the health check should run on.
   c. For **Healthy threshold**, specify the number of consecutive successful health checks that must occur before declaring the listener healthy.
   d. For **Health check interval**, specify the time period in milliseconds between each health check execution.
   e. For **Path**, specify the destination path for the health check request. This is required only if the specified protocol is HTTP. If the protocol is TCP, this parameter is ignored.
   f. For **Timeout period**, specify the amount of time to wait when receiving a response from the health check, in milliseconds.
   g. For **Unhealthy threshold**, specify the number of consecutive failed health checks that must occur before declaring the listener unhealthy.
9. Chose **Create virtual node** to finish.

Creating a virtual node in the AWS CLI.

- The following JSON represents a virtual node named `serviceA`. This virtual node listens on port 8080, and it expects to send traffic to a virtual service within the mesh named `serviceB.simpleapp.local` as a backend. The task group for the virtual node (the Amazon ECS service with service discovery support, or Kubernetes deployment) uses the `serviceA.simpleapp.local` hostname for service discovery.

```
{
  "meshName": "simpleapp",
  "spec": {
    "backends": [
            {
                "virtualService": {
                    "virtualServiceName": "serviceB.simpleapp.local"
                }
            }
        ],
```

```
    "listeners": [
      {
        "portMapping": {
          "port": 8080,
          "protocol": "http"
        }
      }
    ],
    "serviceDiscovery": {
      "dns": {
        "hostname": "serviceA.simpleapp.local"
      }
    }
  },
  "virtualNodeName": "serviceA"
}
```

If you save the preceding JSON as a file, you can create the virtual node with the following AWS CLI command.

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

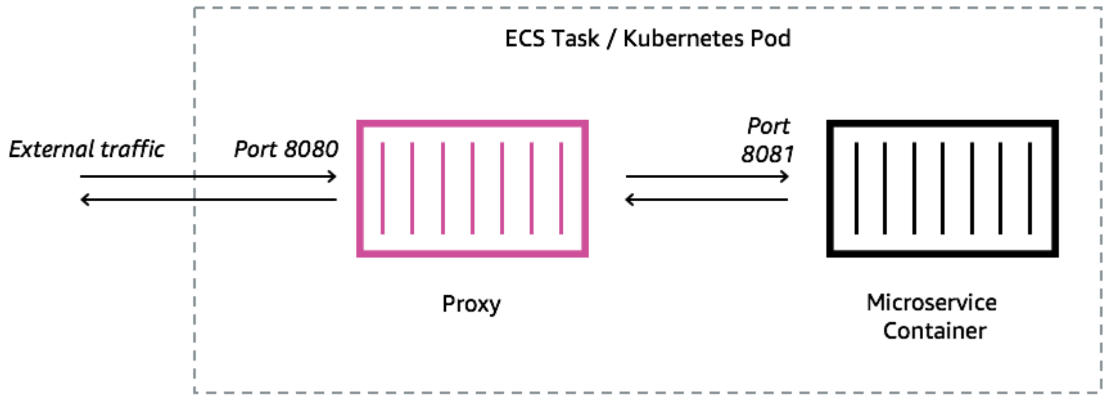
Note

You must use at least version 1.16.133 of the AWS CLI with App Mesh. To install the latest version of the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

For more information about creating virtual nodes with the AWS CLI, see create-virtual-node in the *AWS Command Line Interface User Guide*.

Envoy Image

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



After you create your service mesh, virtual nodes, virtual routers, routes, and virtual services, you add the following App Mesh Envoy container image to the ECS task or Kubernetes pod represented by your App Mesh virtual nodes:

```
111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.9.1.0-prod
```

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

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`

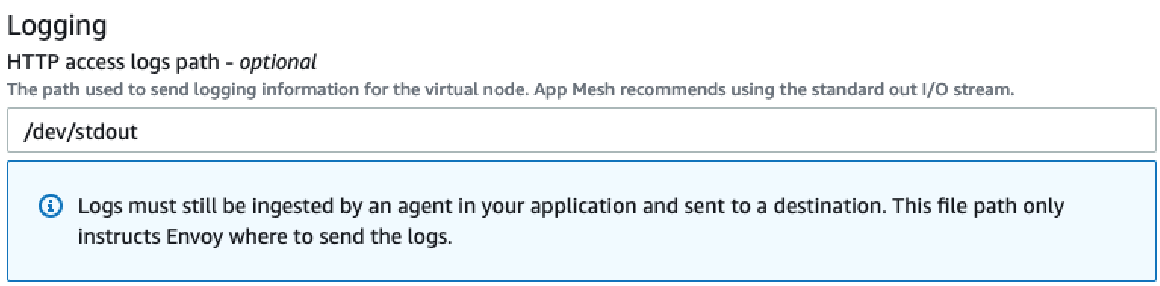
Enables the use of App Mesh defined tags `appmesh.mesh` and `appmesh.virtual_node`. For more information, see config.metrics.v2.TagSpecifier in the Envoy documentation.

`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.



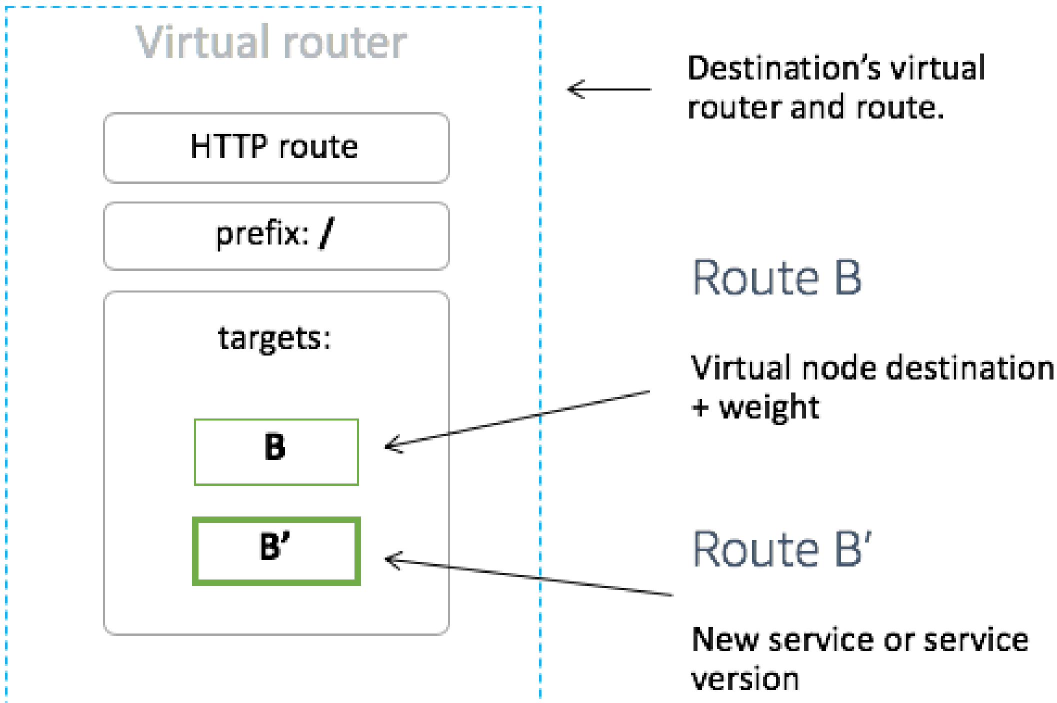
The above image shows a logging path of `/dev/stdout` for Envoy access logs. The code block below shows the JSON representation that you could use in the AWS CLI.

```
"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.

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

This topic helps you to create a virtual router in your service mesh.

Creating a virtual router in the AWS Management Console.

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh that you want to create the virtual router in.
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.

Creating a virtual router in the AWS CLI.

- The following JSON represents a virtual router named `serviceB` that listens for HTTP traffic on port 80.

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

  If you save the preceding JSON as a file, you can create the virtual router with the following AWS CLI command.

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

> **Note**
> You must use at least version 1.16.133 of the AWS CLI with App Mesh. To install the latest version of the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

  For more information about creating virtual routers with the AWS CLI, see create-virtual-router in the *AWS Command Line Interface User Guide*.

Routes

A route is associated with a virtual router, and it's used to match requests for a virtual router and distribute traffic accordingly to its associated virtual nodes.

You can use the `prefix` parameter in your route specification for path-based routing of requests. For example, if your virtual service name is `my-service.local` and you want the route to match requests to `my-service.local/metrics`, your prefix should be `/metrics`.

If your route matches a request, you can distribute traffic to one or more target virtual nodes with relative weighting.

Creating a Route

This topic helps you to create a route in your service mesh.

Creating a route in the AWS Management Console.

1. Open the App Mesh console at https://console.aws.amazon.com/appmesh/.
2. Choose the mesh that you want to create the route in.
3. Choose **Virtual routers** in the left navigation.
4. Choose the router that you want to associate a new route with.
5. In the **Routes** table, choose **Create route**.
6. For **Route name**, specify the name to use for your route.
7. For **Route type**, choose the protocol for your route.
8. For **Virtual node name**, choose the virtual node that this route will serve traffic to.
9. For **Weight**, choose a relative weight for the route. The total weight for all routes must be less than 100.
10. To use HTTP path-based routing, choose **Additional configuration** and then specify the path that the route should match. For example, if your virtual service name is `my-service.local` and you want the route to match requests to `my-service.local/metrics`, your prefix should be `/metrics`.
11. Choose **Create route** to finish.

Creating a route in the AWS CLI.

- The following JSON represents a route named `serviceB-route` for the virtual router `serviceB`. This route directs traffic to two virtual nodes: 90% to `serviceBv1` and 10% to `serviceBv2`. This canary-style deployment allows you to test a new version of your service with a small amount of traffic to decrease the blast radius of new service code.

```
{
  "meshName": "simpleapp",
  "routeName": "serviceB-route",
  "spec": {
    "httpRoute": {
      "action": {
        "weightedTargets": [
          {
            "virtualNode": "serviceBv1",
```

```
            "weight": 90
          },
          {
            "virtualNode": "serviceBv2",
            "weight": 10
          }
        ]
      },
      "match": {
        "prefix": "/"
      }
    }
  },
  "virtualRouterName": "serviceB"
}
```

If you save the preceding JSON as a file, you can create the route with the following AWS CLI command.

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

Note
You must use at least version 1.16.133 of the AWS CLI with App Mesh. To install the latest version of the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

For more information about creating routes with the AWS CLI, see create-route in the *AWS Command Line Interface User Guide*.

AWS App Mesh IAM Policies, Roles, and Permissions

By default, IAM users don't have permission to create or modify AWS App Mesh resources or perform tasks using the App Mesh API. (This means that they also can't do so using the AWS CLI.) To allow IAM users to create or modify service meshes, you must create IAM policies that grant IAM users permissions to use the specific resources and API actions that they need, and then attach those policies to the IAM users or groups that require those permissions.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more information, see Permissions and Policies in the *IAM User Guide*. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

Getting Started

An IAM policy must grant or deny permissions to use one or more App Mesh actions.

Topics

Policy Structure

The following topics explain the structure of an IAM policy.

Topics

Policy Syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows.

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
    }
  ]
```

```
}
```

The following elements make up a statement:

- **Effect** – The *effect* can be `Allow` or `Deny`. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.
- **Action** – The *action* is the specific API action that you're granting or denying permission for.
- **Resource** – The resource that is affected by the action. App Mesh API operations currently don't support resource-level permissions, so you must use the * wildcard to specify that all resources can be affected by the action.
- **Condition** – Conditions are optional. They can be used to control when your policy is in effect.

For more information about example IAM policy statements for App Mesh, see Creating App Mesh IAM Policies (p. 34).

Actions for App Mesh

In an IAM policy statement, you can specify any API action from any service that supports IAM. For App Mesh, use the following prefix with the name of the API action: `appmesh:`. For example: `appmesh:CreateMesh` and `appmesh:DeleteMesh`.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": ["appmesh:action1", "appmesh:action2"]
```

You can also specify multiple actions using wildcards. For example, you can specify all actions whose name begins with the word "Describe" as follows.

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

To specify all App Mesh API actions, use the * wildcard as follows.

```
"Action": "appmesh:*"
```

Checking That Users Have the Required Permissions

After you have created an IAM policy, we recommend that you check whether it grants users the permissions to use the particular API actions and resources that they need. Do this before you put the policy into production.

First, create an IAM user for testing purposes and then attach the IAM policy that you created to the test user. Then make a request as the test user. You can make test requests in the console or with the AWS CLI.

> **Note**
> You can also test your policies with the IAM Policy Simulator. For more information on the policy simulator, see Working with the IAM Policy Simulator in the *IAM User Guide*.

If the policy doesn't grant the user the permissions that you expected or is overly permissive, you can adjust the policy as needed. Retest until you get the desired results.

> **Important**
> It can take several minutes for policy changes to propagate before they take effect. We recommend that you allow 5 minutes to pass before you test your policy updates.

If an authorization check fails, the request returns an encoded message with diagnostic information. You can decode the message using the `DecodeAuthorizationMessage` action. For more information, see DecodeAuthorizationMessage in the *AWS Security Token Service API Reference*, and decode-authorization-message in the *AWS CLI Command Reference*.

Creating App Mesh IAM Policies

You can create specific IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources. For more information, see Permissions and Policies in the *IAM User Guide*. For more information about managing and creating custom IAM policies, see Managing IAM Policies.

If your IAM user doesn't have administrative privileges, you must explicitly add permissions for that user to call the App Mesh API operations.

To create an IAM policy for an App Mesh admin user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies** and then **Create policy**.
3. On the **JSON** tab, paste the following policy.

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

4. Choose **Review policy**.
5. For **Name**, enter your own unique name, such as `AWSAppMeshAdminPolicy`.
6. Choose **Create Policy**.

To attach an IAM policy to a user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** and then choose the user to attach the policy to.
3. Choose **Permissions** and then **Add permissions**.
4. Under **Grant permissions**, choose **Attach existing policies directly**.
5. Select the custom policy that you created in the previous procedure and choose **Next: Review**.
6. Review your details and choose **Add permissions**.

AWS App Mesh Service Limits

The following table provides the default limits that can be changed for App Mesh on an AWS account. For more information, see AWS Service Limits in the *Amazon Web Services General Reference*.

Resource	**Default Limit**
Maximum number of meshes per account	15
Maximum number of virtual services per mesh	200
Maximum number of virtual nodes per mesh	20
Maximum number of virtual routers per mesh	20
Maximum number of routes per virtual router	20
Maximum number of weighted targets per route	10

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.

update-history-change	update-history-description	update-history-date
App Mesh General Availability (p. 36)	The App Mesh service is now generally available for production use.	March 27, 2019
App Mesh API update (p. 36)	The App Mesh APIs were updated to improve usability. For more information, see [FEATURE] Add Listeners to Virtual Routers and [BUG] Routes to Target Virtual Nodes with Mismatched Ports Blackhole.	March 7, 2019
App Mesh initial release (p. 36)	Initial documentation for service public preview	November 28, 2018

AWS Glossary

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