aws

用户指南

AWS App Mesh

AWS App Mesh: 用户指南

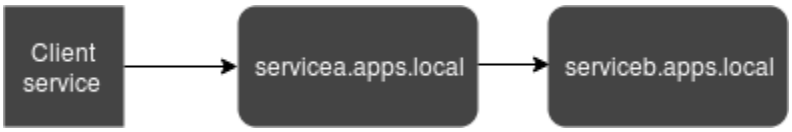
Table of Contents

什么是 AWS App Mesh？

AWS App Mesh 是一种服务网格，可轻松监控和控制服务。服务网格是一个基础设施层，专门用于处理服务到服务的通信，通常通过与应用程序代码一起部署的一系列轻量级网络代理。App Mesh 实现了服务通信方式的标准化，为您提供端到端的可见性，并有助于确保应用程序的高可用性。App Mesh 将为您提供对应用程序中的每个服务的一致可见性和网络流量控制。

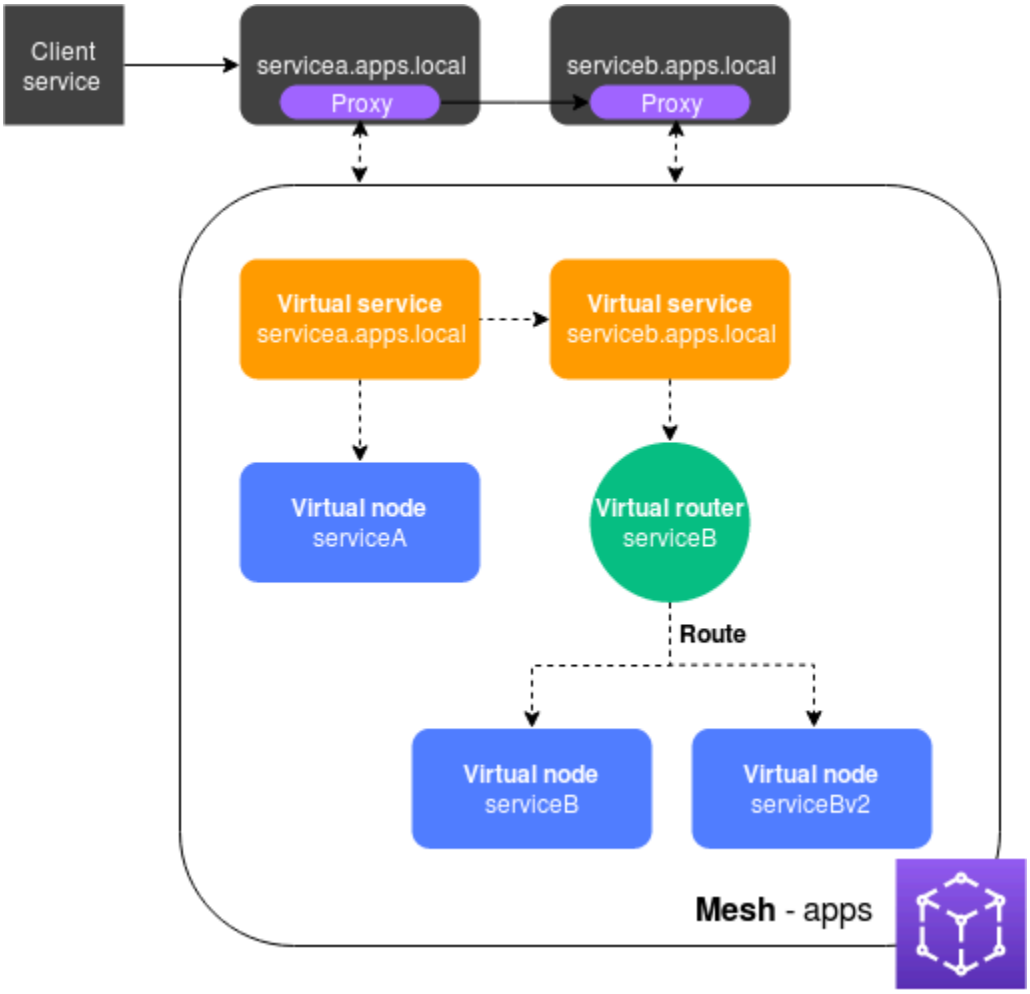
将 App Mesh 添加到示例应用程序中

考虑以下不使用 App Mesh 的简单示例应用程序。这两项服务可以在 AWS Fargate、Amazon Elastic Container Service (Amazon ECS) 、Amazon Elastic Kubernetes Service (Amazon EKS)、Amazon Elastic Compute Cloud (Amazon EC2) 实例上的 Kubernetes 或带 Docker 的 Amazon EC2 实例上运行。



在此插图中，`serviceA` 和 `serviceB` 均可通过 `apps.local` 命名空间发现。例如，假设您决定部署 `serviceb.apps.local` named `servicebv2.apps.local` 的新版本。接下来，您要将一定比例的流量从 `servicea.apps.local` 定向到 `serviceb.apps.local`，将一定比例的流量定向到 `servicebv2.apps.local`。当您确定 `servicebv2` 它表现不错时，您想向它发送100%的流量。

App Mesh 无需更改任何应用程序代码或注册的服务名称即可帮助您完成此操作。如果您在此示例应用程序中使用 App Mesh，则您的网格可能如下图所示。



在此配置中，服务不再直接相互通信。相反，它们通过代理相互通信。与 `servicea.apps.local` 服务一起部署的代理读取 App Mesh 配置，并根据配置向 `serviceb.apps.local` 或 `servicebv2.apps.local` 根据配置发送流量。

App Mesh

App Mesh 由以下组件组成，如上一个示例所示：

- 服务网格 — 一种用于驻留在其内的服务之间的网络流量的逻辑边界。在示例中，网格被命名为 `apps`，它包含网格的所有其他资源。有关更多信息，请参阅 服务网格。
- 虚拟服务 — 一种抽象的实际服务，由虚拟节点直接提供或通过虚拟路由器的方式间接提供。在插图中，两项虚拟服务代表两项实际服务。虚拟服务的名称是实际服务的可发现名称。当虚拟服务和实际服务具有相同名称时，多项服务可以使用与实现 App Mesh 之前相同的名称相互通信。有关更多信息，请参阅 虚拟服务。

- 虚拟节点 — 虚拟节点充当可发现服务的逻辑指针，如 Amazon ECS 或 Kubernetes 服务。对于每项虚拟服务，您将至少有一个虚拟节点。在插图中，`servicea.apps.local` 虚拟服务获取名为 `serviceA` 的虚拟节点的配置信息。`serviceA` 虚拟节点配置了用于服务发现的 `servicea.apps.local` 名称。`serviceb.apps.local` 虚拟服务配置为通过名为 `serviceB` 的虚拟路由器将流量路由到 `serviceB` 和 `serviceBv2` 虚拟节点。有关更多信息，请参阅 虚拟节点。
- 虚拟路由器 — 处理用于您的网格内一项或多个虚拟服务的流量。路由与虚拟路由器关联。该路由用于匹配对虚拟路由器的请求并将流量分发到其关联的虚拟节点。在上图中，`serviceB` 虚拟路由器的路由将一定比例的流量定向到 `serviceB` 虚拟节点，将一定百分比的流量定向到 `serviceBv2` 虚拟节点。您可以设置路由到特定虚拟节点的流量百分比，并随着时间推移进行更改。您可以根据诸如 HTTP 标头、URL 路径或 gRPC 服务和方法名称之类的标准来路由流量。您可以配置重试策略，以便在响应中出现错误时重试连接。例如，在插图中，如果 `serviceb.apps.local` 返回特定类型的错误，`serviceb.apps.local` 路由的重试策略可以指定重试与 连接五次，两次重试之间有十秒钟。有关更多信息，请参阅 虚拟路由器 和 路由。
- 代理 — 创建网格及其资源后，您可以将服务配置为使用代理。代理读取 App Mesh 配置并适当地引导流量。在插图中，所有从 `servicea.apps.local` 到 `serviceb.apps.local` 的通信都通过为每项服务部署的代理进行。这些服务使用与引入 App Mesh 之前相同的服务发现名称相互通信。由于代理会读取 App Mesh 配置，因此您可以控制这两项服务之间的通信方式。如果要更改 App Mesh 配置，则无需更改或重新部署服务本身或代理。有关更多信息，请参阅 Envoy 镜像。

如何开始

要使用 App Mesh，您必须在 AWS Fargate、亚马逊 ECS、亚马逊 EKS、亚马逊 ECS、亚马逊 EC2 上的 Kubernetes 或带有 Docker 的亚马逊 EC2 上运行现有服务。

要开始使用 App Mesh 中的标签，请参阅以下指南：

- App Mesh 和 Amazon ECS 入门
- App Mesh 和 Kubernetes 入门
- App Mesh 和 Amazon EC2 入门

访问 App Mesh

您可以通过以下方式使用 App Mesh：

AWS Management Console

控制台是一个基于浏览器的界面，您可以用它来管理 App Mesh 资源。您可以通过访问 https://console.aws.amazon.com/appmesh/ 打开 App Mesh 控制台。

AWS CLI

提供大量 AWS 产品的相关命令，同时被 Windows、Mac 和 Linux 支持。要开始使用，请参阅 AWS Command Line Interface 用户指南。有关 App Mesh 的更多信息，请参阅 AWS CLI Command Reference 中的 appmesh。

AWS Tools for Windows PowerShell

为在 PowerShell 环境中编写脚本的用户提供大量 AWS 产品的相关命令。要开始使用，请参阅 AWS Tools for Windows PowerShell 用户指南。有关 Amazon Mesh 的 Cmdlet 的更多信息，请参阅AWS Tools for PowerShell Cmdlet Reference中的App Mesh。

AWS CloudFormation

您可以创建一个模板来描述要使用的所有 AWS 资源。使用模板为您 AWS CloudFormation 预置和配置资源。要开始使用，请参阅AWS CloudFormation《用户指南》。有关 App Mesh 资源类型的更多信息，请参阅AWS CloudFormation《模板参考》App Mesh 资源类型参考。

AWS 软件开发工具包

我们还提供 SDK 用于通过各种编程语言访问 App Mesh。软件开发工具包将自动处理任务，例如：

- 使用密码对服务请求签名
- 重试请求
- 处理错误响应

有关可用软件开发工具包的更多信息，请参阅适用于 Amazon Web Services 的工具。

有关使用这些 App Mesh API 的更多信息，请参阅 AWS App Mesh API 参考。

App Mesh 应用程序入门

您可以将 App Mesh 与部署到亚马逊 ECS、Kubernetes（部署到自己的亚马逊 EC2 实例或在 Amazon EKS 上运行）和亚马逊 EC2 上的应用程序一起使用。要开始使用 App Mesh，请选择一个已部署应用程序的服务，然后将其与 App Mesh 一起使用。完成其中一个入门指南后，您始终可以让其他服务中的应用程序也能与 App Mesh 配合使用。

主题

开始使用 AWS App Mesh 和 Amazon ECS

本主题可帮助您 AWS App Mesh 使用在 Amazon ECS 上运行的实际服务。本教程介绍了多种 App Mesh 资源类型的基本功能。

场景

为了说明如何配合使用 App Mesh，假定您有一个具有以下功能的应用程序：

- 由名为 `serviceA` 和 `serviceB` 的两项服务组成。
- 这两项服务均注册到名为 `apps.local` 的命名空间。
- `ServiceA` 通过 HTTP/2 和端口 80 与 `serviceB` 通信。
- 您已部署 `serviceB` 的版本 2，并采用名称 `serviceBv2` 在 `apps.local` 命名空间中注册了该服务。

您的要求如下：

- 您希望将 75% 的流量从 `serviceA` 发送到 `serviceB`，并将 25% 的流量发送到 `serviceBv2`，以验证在将 100% 的流量从 `serviceA` 发送到它之前，`serviceBv2` 没有错误。
- 您希望能够轻松调整流量权重，以便一旦证明 `serviceBv2` 可靠便可将 100% 的流量发送到该服务。将所有流量发送到 `serviceBv2` 后，您希望弃用 `serviceB`。

- 您不想为了满足之前的要求而更改实际服务的任何现有应用程序代码或服务发现注册。

为了满足您的要求，您决定创建包含虚拟服务、虚拟节点、虚拟路由器和路由的 App Mesh 服务网格。实施您的网格后，更新使用 Envoy 代理的服务。更新后，您的服务将通过 Envoy 代理相互通信，而不是直接相互通信。

先决条件

- 已了解 App Mesh 概念。有关更多信息，请参阅 什么是 AWS App Mesh？。
- 对 Amazon 弹性云服务器概念的现有理解。有关更多信息，请参阅《Amazon Elastic Container Service 开发人员指南》中的 Amazon ECS 的定义。
- App Mesh 支持在 DNS 中注册的 Linux 服务 AWS Cloud Map，或者两者兼而有之。要使用此入门指南，我们建议您提供三项已注册到 DNS 的现有服务。该主题的剩余步骤假定现有服务命名为 `serviceA`、`serviceB` 和 `serviceBv2`，并且可以在名为 `apps.local` 的命名空间中发现所有服务。

  即使服务不存在，您也可以创建服务网格及其资源，但在部署实际服务之前，您无法使用网格。有关 Amazon ECS 上服务发现的更多信息，请参阅服务发现。要使用服务发现创建 Amazon ECS 服务，请参阅教程：使用服务发现创建服务。如果您尚未运行服务，则可以创建带有服务发现功能的 Amazon ECS 服务。

步骤 1：创建网格和虚拟服务

服务网格是一种用于驻留在其内的服务之间的网络流量的逻辑边界。有关更多信息，请参阅 服务网格。虚拟服务是实际服务的抽象。有关更多信息，请参阅 虚拟服务。

创建以下 资源：

- 名为 `apps` 的网格，因为此场景中的所有服务均注册到 `apps.local` 命名空间。
- 名为 `serviceb.apps.local` 的虚拟服务，因为虚拟服务表示可以使用该名称发现的服务，并且您不希望更改代码以引用其他名称。稍后的步骤中将添加名为 `servicea.apps.local` 的虚拟服务。

您可以使用或 1.18.116 AWS Management Console 或更高 AWS CLI 版本或 2.0.38 或更高版本来完成以下步骤。如果使用 AWS CLI，请使用`aws --version`命令检查已安装的 AWS CLI 版本。如果您没有安装版本 1.18.116 或更高版本或者没有安装版本 2.0.38 或更高版本，则必须安装或更新 AWS CLI。选择要使用的工具所对应的选项卡。

AWS Management Console

1. 打开 App Mesh 控制台首次运行向导，网址为 https://console.aws.amazon.com/appmesh/get-started。
2. 对于网格名称，输入 **apps**。
3. 对于虚拟服务名称，输入 **serviceb.apps.local**。
4. 要继续，请选择 Next。

AWS CLI

1. 使用 `create-mesh` 命令创建网格。

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

2. 使用 `create-virtual-service` 命令创建虚拟服务。

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

步骤 2：创建虚拟节点

虚拟节点充当实际服务的逻辑指针。有关更多信息，请参阅 虚拟节点。

创建名为 `serviceB` 的虚拟节点，因为某个虚拟节点表示名为 `serviceB` 的实际服务。可使用主机名 `serviceb.apps.local`，通过 DNS 发现虚拟节点所表示的实际服务。或者，您可以使用发现实际的服务。 AWS Cloud Map虚拟节点将采用 HTTP/2 协议在端口 80 上侦听流量。此外，还支持其他协议和运行状况检查。您将在后面的步骤中为 `serviceBv2` 和 `serviceA` 创建虚拟节点。

AWS Management Console

1. 对于虚拟节点名称，输入 **serviceB**。
2. 对于服务发现方法，选择 DNS，并为DNS 主机名输入 **serviceb.apps.local**。
3. 在侦听器配置下，为协议选择 http2，并在端口中，输入 **80**。
4. 要继续，请选择 Next。

AWS CLI

1. 使用以下内容创建名为 create-virtual-node-serviceb.json 的文件：

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

2. 使用 JSON 文件作为输入使用create-virtual-node命令创建虚拟节点。

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

步骤 3：创建虚拟路由器和路由

虚拟路由器路由网格中一个或多个虚拟服务的流量。有关更多信息，请参阅 虚拟路由器 和 路由。

创建以下 资源：

- 名为 serviceB 的虚拟路由器，因为 serviceB.apps.local 虚拟服务不会启动与任何其他服务的出站通信。请记住，您之前创建的虚拟服务是实际 serviceb.apps.local 服务的抽象。虚拟服务将流量发送到虚拟路由器。虚拟路由器采用 HTTP/2 协议在端口 80 上侦听流量。此外，还支持其他协议。

- 名为 serviceB 的路由。它将 100% 的流量路由到serviceB虚拟节点。添加 serviceBv2 虚拟节点后，在稍后的步骤中出现权重。虽然本指南中未作介绍，但您可以为路由添加额外的筛选条件，并添加重试策略，从而使 Envoy 代理在遇到通信问题时会多次尝试将流量发送到虚拟节点。

AWS Management Console

1. 对于虚拟路由器名称，输入 **serviceB**。
2. 在侦听器配置下，为协议选择 http2，并为端口指定 **80**。
3. 对于路由名称，输入 **serviceB**。
4. 对于路由类型，选择 http2。
5. 在目标配置下的虚拟节点名称中，选择 serviceB，并为权重输入 **100**。
6. 在匹配配置下，选择一种方法。
7. 要继续，请选择 Next。

AWS CLI

1. 创建虚拟路由器。

   a. 使用以下内容创建名为 create-virtual-router.json 的文件：

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

   b. 使用 JSON 文件作为输入使用create-virtual-router命令创建虚拟路由器。

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

2. 创建路由。

   a. 使用以下内容创建名为 create-route.json 的文件：

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

   b. 采用 JSON 文件作为输入，使用 create-route 命令创建路由。

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

步骤 4：审核并创建

根据之前的说明审核设置。

AWS Management Console

如果需要对任何部分进行任何更改，请选择编辑。在您对设置感到满意后，选择创建网格。

状态屏幕将显示已创建的所有网格资源。您可以通过选择查看网格来在控制台中查看创建的资源。

AWS CLI

使用 describe-mesh 命令查看所创建网格的设置。

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

查看您使用describe-virtual-service命令创建的虚拟服务的设置。

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

查看您使用describe-virtual-node命令创建的虚拟节点的设置。

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

查看您使用describe-virtual-router命令创建的虚拟路由器的设置。

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

使用 describe-route 命令查看所创建路由的设置。

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

步骤 5：创建其他资源

要完成此场景，您需要执行以下操作：

- 创建名为 serviceBv2 的虚拟节点，以及名为 serviceA 的虚拟节点。这两个虚拟节点均通过 HTTP/2 和端口 80 侦听请求。对于 serviceA 虚拟节点，将后端配置为 serviceb.apps.local。来自 serviceA 虚拟节点的所有出站流量都将发送到名为 serviceb.apps.local 的虚拟服务。虽然本指南中未作介绍，但您还可以为虚拟节点指定用于写入访问日志的文件路径。
- 另外创建一个名为的虚拟服务servicea.apps.local，该服务将所有流量直接发送到serviceA虚拟节点。
- 更新在上一步中创建的 serviceB 路由，以将 75% 的流量发送到 serviceB 虚拟节点，将 25% 的流量发送到 serviceBv2 虚拟节点。随着时间的推移，您可以继续修改权重，直到 serviceBv2 收到 100% 的流量。将所有流量发送到 serviceBv2 后，您可以关闭并弃用 serviceB 虚拟节点

和实际服务。在更改权重时，不需要对代码进行任何修改，因为 serviceb.apps.local 虚拟服务名称和实际服务名称没有改变。回想一下，serviceb.apps.local 虚拟服务将流量发送到虚拟路由器，该路由器将流量路由到虚拟节点。虚拟节点的服务发现名称可以随时更改。

AWS Management Console

1. 在左侧导航窗格中，选择网格。
2. 选择在上一步中创建的 apps 网格。
3. 在左侧导航窗格中，选择虚拟节点。
4. 选择创建虚拟节点。
5. 为虚拟节点名称输入 **serviceBv2**，为服务发现方法选择 DNS，并为 DNS 主机名输入 **servicebv2.apps.local**。
6. 对于侦听器配置，为协议选择 http2，并在端口中，输入 **80**。
7. 选择创建虚拟节点。
8. 再次选择创建虚拟节点。对于虚拟节点名称，输入 **serviceA**。对于服务发现方法，选择 DNS，并为 DNS 主机名输入 **servicea.apps.local**。
9. 在新后端下的输入虚拟服务名称中，输入 **serviceb.apps.local**。
10. 在侦听器配置下，为协议选择 http2，并在端口中输入 **80**，然后选择创建虚拟节点。
11. 在左侧导航窗格中，选择虚拟路由器，然后从列表中选择 serviceB 虚拟路由器。
12. 在路由下，选择在上一步中创建的名为 ServiceB 的路由，然后选择编辑。
13. 在目标、虚拟节点名称下，将 serviceB 的权重值更改为 **75**。
14. 选择添加目标，从下拉列表中选择 serviceBv2，然后将权重值设置为 **25**。
15. 选择保存。
16. 在左侧导航窗格中，选择虚拟服务，然后选择创建虚拟服务。
17. 针对虚拟服务名称输入 **servicea.apps.local**，为提供者选择虚拟节点，为虚拟节点选择 serviceA，然后选择创建虚拟服务。

AWS CLI

1. 创建 serviceBv2 虚拟节点。

   a. 使用以下内容创建名为 create-virtual-node-servicebv2.json 的文件：

```
{
```

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

b. 创建虚拟节点。

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

2. 创建 serviceA 虚拟节点。

a. 使用以下内容创建名为 create-virtual-node-servicea.json 的文件：

```
{
   "meshName" : "apps",
   "spec" : {
      "backends" : [
         {
            "virtualService" : {
               "virtualServiceName" : "serviceb.apps.local"
            }
         }
      ],
      "listeners" : [
         {
            "portMapping" : {
               "port" : 80,
               "protocol" : "http2"
```

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

b. 创建虚拟节点。

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

3. 更新在上一步中创建的 serviceb.apps.local 虚拟服务，以将其流量发送到 serviceB 虚拟路由器。最初创建虚拟服务时，它不会向任何位置发送流量，因为尚未创建 serviceB 虚拟路由器。

   a. 使用以下内容创建名为 update-virtual-service.json 的文件：

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

   b. 使用update-virtual-service命令更新虚拟服务。

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

4. 更新在上一步中创建的 serviceB 路径。

   a. 使用以下内容创建名为 update-route.json 的文件：

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

b. 使用 update-route 命令更新路由。

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

5. 创建 serviceA 虚拟服务。

a. 使用以下内容创建名为 create-virtual-servicea.json 的文件：

```
{
   "meshName" : "apps",
   "spec" : {
      "provider" : {
         "virtualNode" : {
            "virtualNodeName" : "serviceA"
         }
      }
   },
```

```
   "virtualServiceName" : "servicea.apps.local"
}
```

b. 创建虚拟服务。

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

网格摘要

在创建服务网格之前，您具有三个名为 servicea.apps.local、serviceb.apps.local 和 servicebv2.apps.local 的实际服务。除实际服务之外，现在您还具有一个服务网格，其中包含用以表示实际服务的以下资源：

- 两个虚拟服务。代理通过虚拟路由器将所有流量从 servicea.apps.local 虚拟服务发送到 serviceb.apps.local 虚拟服务。
- 三个名为 serviceA、serviceB 和 serviceBv2 的虚拟节点。Envoy 代理使用为虚拟节点配置的服务发现信息来查找实际服务的 IP 地址。
- 一个虚拟路由器，其路由指示 Envoy 代理将 75% 的入站流量路由到 serviceB 虚拟节点，将 25% 的流量路由到 serviceBv2 虚拟节点。

步骤 6：更新服务

创建网格后，您需要完成以下任务：

- 授予随每项 Amazon ECS 任务服务部署的 Envoy 代理读取一个或多个虚拟节点配置的权限。有关如何向代理授权的更多信息，请参阅代理授权。
- 更新每个现有 Amazon ECS 任务定义服务，以使用 Envoy 代理。

凭证

Envoy 容器需要 AWS Identity and Access Management 凭据才能对发送到 App Mesh 服务的请求进行签名。对于使用 Amazon EC2 启动类型部署的 Amazon ECS 任务，凭证可以来自实例角色或任务 IAM 角色。在 Linux 容器上使用 Fargate 部署的 Amazon ECS 任务无法访问提供实例 IAM 配置凭证的 Amazon EC2 元数据服务器。要提供凭证，您必须将 IAM 任务角色附加到在 Linux 容器类型上使用 Fargate 启动类型部署的任何任务。

如果任务以 Amazon EC2 启动类型部署，并且无法访问 Amazon EC2 元数据服务器（如 IAM 任务角色中的重要注释中所述），则还必须为该任务附加任务 IAM 角色。您分配给实例或任务的角色必须附加一个 IAM 策略，如代理授权中所述。

要更新任务定义，请使用 AWS CLI

你使用 Amazon ECS AWS CLI 命令`register-task-definition`。下面的示例任务定义显示了如何为您的服务配置 App Mesh。

> **Note**
>
> 无法通过控制台为 Amazon ECS 配置 App Mesh。

任务定义 json

代理配置

要将您的 Amazon ECS 服务配置为使用 App Mesh，您的服务的任务定义必须具有以下代理配置部分。将代理配置 `type` 设置为 `APPMESH`，并 `containerName` 配置为 `envoy`。相应地设置以下属性值。

`IgnoredUID`

Envoy 代理不会路由来自使用此用户 ID 的进程的流量。您可以为此属性值选择所需的任何用户 ID，但此 ID 必须与任务定义中的 Envoy 容器的 `user` ID 相同。此匹配允许 Envoy 忽略其自己的流量，而无需使用代理。出于历史原因，我们的示例使用 *`1337`*。

`ProxyIngressPort`

这是 Envoy 代理容器的入站端口。将此值设置为 `15000`。

`ProxyEgressPort`

这是 Envoy 代理容器的出口端口。将此值设置为 `15001`。

`AppPorts`

指定您的应用程序容器监听的所有入站端口。在此示例中，应用程序容器在端口 *`9080`* 上侦听。您指定的端口必须与虚拟节点侦听器上配置的端口匹配。

`EgressIgnoredIPs`

Envoy 不会将流量代理至这些 IP 地址。将此值设置为 `169.254.170.2,169.254.169.254`，这会忽略 Amazon EC2 元数据服务器和 Amazon ECS 任务元数据端点。元数据端点为任务凭证提供 IAM 角色。您可以添加其他地址。

`EgressIgnoredPorts`

您可以添加以逗号分隔的端口列表。Envoy 不会将流量转发至这些端口。即使您没有列出任何端口，端口 22 也会被忽略。

> **Note**
>
> 可以忽略的最大出站端口数为 15。

```
"proxyConfiguration": {
 "type": "APPMESH",
 "containerName": "envoy",
 "properties": [{
   "name": "IgnoredUID",
   "value": "1337"
  },
  {
   "name": "ProxyIngressPort",
   "value": "15000"
  },
  {
   "name": "ProxyEgressPort",
   "value": "15001"
  },
  {
   "name": "AppPorts",
   "value": "9080"
  },
  {
   "name": "EgressIgnoredIPs",
   "value": "169.254.170.2,169.254.169.254"
  },
  {
   "name": "EgressIgnoredPorts",
   "value": "22"
  }
```

```
 ]
}
```

应用程序容器 Envoy 依赖项

您的任务定义中的应用程序容器必须等待 Envoy 代理来引导和启动之后才能启动。为了确保这种情况发生，你需要在每个应用程序容器定义中设置一个dependsOn部分，等待 Envoy 容器报告为HEALTHY。以下代码显示了包含此依赖项的应用程序容器定义示例。以下示例中的所有属性都是必需的。某些属性值也是必需的，但某些属性值是*####*。

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

Envoy 容器定义

您的 Amazon ECS 任务定义必须包含 App Mesh Envoy 容器镜像。

除me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1 和 af-south-1 之外的所有受支持地区。您可以用me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1 和 af-south-1之外的任何地区替换*####*。

Standard

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

符合 FIPS 标准

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

me-south-1

Standard

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

符合 FIPS 标准

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

ap-east-1

Standard

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

符合 FIPS 标准

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

ap-southeast-3

Standard

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

符合 FIPS 标准

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

eu-south-1

Standard

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

符合 FIPS 标准

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

`il-central-1`

Standard

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

符合 FIPS 标准

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

`af-south-1`

Standard

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

符合 FIPS 标准

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

`Public repository`

Standard

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

符合 FIPS 标准

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

Important

仅支持将版本 v1.9.0.0-prod 或更高版本与 App Mesh 一起使用。

在 Envoy 项目团队合并支持 App Mesh 的更改之前，您必须使用 App Mesh Envoy 容器镜像。有关更多详细信息，请参阅GitHub 路线图问题。

以下示例中的所有属性都是必需的。某些属性值也是必需的，但某些属性值是####。

Note

- Envoy 容器定义必须标记为 `essential`。
- 我们建议为 Envoy 容器分配 512 个 CPU 单元及至少 64 MiB 的内存。在 Fargate 上，您可以设置的最低内存为 1024MiB。
- Amazon ECS 服务的虚拟节点名称必须设置为 APPMESH_RESOURCE_ARN 属性的值。此属性需要版本`1.15.0`或更高版本的 Envoy 镜像。有关更多信息，请参阅 Envoy。
- `user` 设置的值必须与任务定义代理配置中的 `IgnoredUID` 值匹配。在此示例中，我们使用的是 *1337*。
- 此处显示的运行状况检查等待 Envoy 容器正确引导，然后再向 Amazon ECS报告其运行状况良好且已准备好启动应用程序容器。
- 默认情况下，当 Envoy 在指标和跟踪中引用自身时，App Mesh 使用您在 APPMESH_RESOURCE_ARN 中指定的资源的名称。您可以通过使用自己的名称设置 APPMESH_RESOURCE_CLUSTER 环境变量来覆盖此行为。此属性需要版本 `1.15.0` 或更高版本的 Envoy 镜像。有关更多信息，请参阅 Envoy。

以下代码显示 Envoy 容器定义示例。

```
{
 "name": "envoy",
 "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.27.3.0-
prod",
 "essential": true,
 "environment": [{
  "name": "APPMESH_RESOURCE_ARN",
  "value": "arn:aws:appmesh:us-west-2:111122223333:mesh/apps/virtualNode/serviceB"
 }],
 "healthCheck": {
  "command": [
   "CMD-SHELL",
   "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
  ],
  "startPeriod": 10,
  "interval": 5,
  "timeout": 2,
  "retries": 3
 },
 "user": "1337"
```

```
}
```

示例任务定义

以下示例 Amazon ECS 任务定义显示如何将上述示例合并到 taskB 的任务定义中。本文分别提供了使用或不使用 AWS X-Ray为两种 Amazon ECS 启动类型创建任务的示例。可根据需要更改#### 值，为此场景中名为 taskBv2 和 taskA 的任务创建任务定义。用您的网格名称和虚拟节点名称替换 APPMESH_RESOURCE_ARN 值，并用您的应用程序侦听的端口列表替换代理配置 AppPorts 值。默认情况下，当 Envoy 在指标和跟踪中引用自身时，App Mesh 使用您在 APPMESH_RESOURCE_ARN 中指定的资源的名称。您可以通过使用自己的名称设置 APPMESH_RESOURCE_CLUSTER 环境变量来覆盖此行为。以下示例中的所有属性都是必需的。某些属性值也是必需的，但某些属性值是####。

如果按照“凭证”部分中所述运行 Amazon ECS 任务，那么需要将现有任务 IAM 角色添加到示例中。

> **Important**
>
> Fargate 必须使用大于 1024 的端口值。

Example 亚马逊 ECS 任务定义的 JSON - Linux 容器上的 Fargate

```
{

   "family" : "taskB",
   "memory" : "1024",
   "cpu" : "0.5 vCPU",
   "proxyConfiguration" : {
      "containerName" : "envoy",
      "properties" : [
         {
            "name" : "ProxyIngressPort",
            "value" : "15000"
         },
         {
            "name" : "AppPorts",
            "value" : "9080"
         },
         {
            "name" : "EgressIgnoredIPs",
            "value" : "169.254.170.2,169.254.169.254"
         },
         {
```

```
            "name": "EgressIgnoredPorts",
            "value": "22"
         },
         {
            "name" : "IgnoredUID",
            "value" : "1337"
         },
         {
            "name" : "ProxyEgressPort",
            "value" : "15001"
         }
      ],
      "type" : "APPMESH"
   },
   "containerDefinitions" : [
      {
         "name" : "appName",
         "image" : "appImage",
         "portMappings" : [
            {
               "containerPort" : 9080,
               "protocol" : "tcp"
            }
         ],
         "essential" : true,
         "dependsOn" : [
            {
               "containerName" : "envoy",
               "condition" : "HEALTHY"
            }
         ]
      },
      {
         "name" : "envoy",
         "image" : "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod",
         "essential" : true,
         "environment" : [
            {
               "name" : "APPMESH_VIRTUAL_NODE_NAME",
               "value" : "mesh/apps/virtualNode/serviceB"
            }
         ],
         "healthCheck" : {
```

```
            "command" : [
               "CMD-SHELL",
               "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
            ],
            "interval" : 5,
            "retries" : 3,
            "startPeriod" : 10,
            "timeout" : 2
         },
         "memory" : 500,
         "user" : "1337"
      }
   ],
   "requiresCompatibilities" : [ "FARGATE" ],
   "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
   "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
   "networkMode" : "awsvpc"
}
```

Example 适用于 Amazon ECS 的 JSON 任务定义使用 AWS X-Ray -Linux 容器上的 Fargate

X-Ray 允许您收集有关应用程序服务于的请求的数据，并提供可用于可视化流量的工具。使用适用于 Envoy 的 X-Ray 驱动程序，Envoy 能够向 X-Ray 报告跟踪信息。您可以使用 Envoy 配置启用 X-Ray 跟踪。根据配置，Envoy 将跟踪数据发送到作为 sidecar 容器运行的 X-Ray 进程守护程序，该进程守护程序将跟踪转发给 X-Ray 服务。在将跟踪发布到 X-Ray 后，您可以使用 X-Ray 控制台来可视化服务调用图和请求跟踪详细信息。以下 JSON 表示用于启用 X-Ray 集成的任务定义。

```
{


   "family" : "taskB",
   "memory" : "1024",
   "cpu" : "512",
   "proxyConfiguration" : {
      "containerName" : "envoy",
      "properties" : [
         {
            "name" : "ProxyIngressPort",
            "value" : "15000"
         },
         {
            "name" : "AppPorts",
            "value" : "9080"
```

```
         },
         {
            "name" : "EgressIgnoredIPs",
            "value" : "169.254.170.2,169.254.169.254"
         },
         {
            "name": "EgressIgnoredPorts",
            "value": "22"
         },
         {
            "name" : "IgnoredUID",
            "value" : "1337"
         },
         {
            "name" : "ProxyEgressPort",
            "value" : "15001"
         }
      ],
      "type" : "APPMESH"
   },
   "containerDefinitions" : [
      {
         "name" : "appName",
         "image" : "appImage",
         "portMappings" : [
            {
               "containerPort" : 9080,
               "protocol" : "tcp"
            }
         ],
         "essential" : true,
         "dependsOn" : [
            {
               "containerName" : "envoy",
               "condition" : "HEALTHY"
            }
         ]
      },
      {

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

```
            "environment" : [
               {
                  "name" : "APPMESH_VIRTUAL_NODE_NAME",
                  "value" : "mesh/apps/virtualNode/serviceB"
               },
               {
                  "name": "ENABLE_ENVOY_XRAY_TRACING",
                  "value": "1"
               }
            ],
            "healthCheck" : {
               "command" : [
                  "CMD-SHELL",
                  "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
               ],
               "interval" : 5,
               "retries" : 3,
               "startPeriod" : 10,
               "timeout" : 2
            },
            "memory" : 500,
            "user" : "1337"
         },
         {
            "name" : "xray-daemon",
            "image" : "amazon/aws-xray-daemon",
            "user" : "1337",
            "essential" : true,
            "cpu" : "32",
            "memoryReservation" : "256",
            "portMappings" : [
               {
                  "containerPort" : 2000,
                  "protocol" : "udp"
               }
            ]
         }
      ],
      "requiresCompatibilities" : [ "FARGATE" ],
      "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
      "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
      "networkMode" : "awsvpc"
   }
```

Example Amazon ECS 任务定义的 JSON - EC2 启动类型

```
{
  "family": "taskB",
  "memory": "256",
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "envoy",
    "properties": [
      {
        "name": "IgnoredUID",
        "value": "1337"
      },
      {
        "name": "ProxyIngressPort",
        "value": "15000"
      },
      {
        "name": "ProxyEgressPort",
        "value": "15001"
      },
      {
        "name": "AppPorts",
        "value": "9080"
      },
      {
        "name": "EgressIgnoredIPs",
        "value": "169.254.170.2,169.254.169.254"
      },
      {
        "name": "EgressIgnoredPorts",
        "value": "22"
      }
    ]
  },
  "containerDefinitions": [
    {
      "name": "appName",
      "image": "appImage",
      "portMappings": [
        {
          "containerPort": 9080,
          "hostPort": 9080,
          "protocol": "tcp"
```

```
        }
      ],
      "essential": true,
      "dependsOn": [
        {
          "containerName": "envoy",
          "condition": "HEALTHY"
        }
      ]
    },
    {
      "name": "envoy",
      "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/apps/virtualNode/serviceB"
        }
      ],
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
        ],
        "startPeriod": 10,
        "interval": 5,
        "timeout": 2,
        "retries": 3
      },
      "user": "1337"
    }
  ],
  "requiresCompatibilities" : [ "EC2" ],
  "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
  "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

Example 适用于 Amazon ECS 任务定义的 JSON AWS X-Ray -EC2 启动类型

```
{
```

```
  "family": "taskB",
  "memory": "256",
   "cpu" : "1024",
  "proxyConfiguration": {
    "type": "APPMESH",
    "containerName": "envoy",
    "properties": [
      {
        "name": "IgnoredUID",
        "value": "1337"
      },
      {
        "name": "ProxyIngressPort",
        "value": "15000"
      },
      {
        "name": "ProxyEgressPort",
        "value": "15001"
      },
      {
        "name": "AppPorts",
        "value": "9080"
      },
      {
        "name": "EgressIgnoredIPs",
        "value": "169.254.170.2,169.254.169.254"
      },
      {
        "name": "EgressIgnoredPorts",
        "value": "22"
      }
    ]
  },
  "containerDefinitions": [
    {
      "name": "appName",
      "image": "appImage",
      "portMappings": [
        {
          "containerPort": 9080,
          "hostPort": 9080,
          "protocol": "tcp"
        }
      ],
```

```
      "essential": true,
      "dependsOn": [
        {
          "containerName": "envoy",
          "condition": "HEALTHY"
        }
      ]
    },
    {
      "name": "envoy",
      "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.27.3.0-prod",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/apps/virtualNode/serviceB"
        },
        {
         "name": "ENABLE_ENVOY_XRAY_TRACING",
         "value": "1"
        }
      ],
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "curl -s http://localhost:9901/server_info | grep state | grep -q LIVE"
        ],
        "startPeriod": 10,
        "interval": 5,
        "timeout": 2,
        "retries": 3
      },
      "user": "1337"
    },
    {
      "name": "xray-daemon",
      "image": "amazon/aws-xray-daemon",
      "user": "1337",
      "essential": true,
      "cpu": 32,
      "memoryReservation": 256,
      "portMappings": [
        {
```

```
          "containerPort": 2000,
          "protocol": "udp"
        }
      ]
    }
  ],
  "requiresCompatibilities" : [ "EC2" ],
  "taskRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskRole",
  "executionRoleArn" : "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "networkMode": "awsvpc"
}
```

高级主题

使用 App Mesh 进行金丝雀部署

金丝雀部署和版本可帮助您在应用程序的旧版本和新部署的版本之间切换流量。它还会监控新部署版本的运行状况。如果新版本有任何问题，金丝雀部署可以自动将流量切换回旧版本。利用金丝雀部署，能够更多地控制在应用程序版本之间切换流量。

有关如何使用 App Mesh 为 Amazon ECS 实施金丝雀部署的更多信息，请参阅使用 App Mesh 为亚马逊 ECS 创建带有金丝雀部署的管道

> **Note**
>
> 有关 App Mesh 的更多示例和演练，请参阅 App Mesh 示例存储库。

入门 AWS App Mesh 和 Kubernetes

当你使用适用于 Kubernetes 的 App Mesh 控制器 AWS App Mesh 与 Kubernetes 集成时，你可以管理应用网格资源，例如网格、虚拟服务、虚拟节点、虚拟路由器和通过 Kubernetes 的路由。您也可以自动向 Kubernetes 容器组 (pod) 规范中添加 App Mesh sidecar 容器映像。本教程指导您完成适用于 Kubernetes 的 App Mesh 控制器的安装，以便启用此集成。

以下 Kubernetes 自定义资源定义的部署附带了控制器：`meshes`、`virtual services`、`virtual nodes` 和 `virtual routers`。控制器密切注意自定义资源的创建、修改和删除，并通过 API 更改相应的 App Mesh the section called “网格” the section called “虚拟服务”、the section called “虚拟节点”、the section called “虚拟网关”、the section called “网关路由”、the section called “虚拟路由

器”（包括 the section called “路由”）资源。要了解更多信息或为控制器做出贡献，请参阅GitHub项目。

控制器还会安装一个 Webhook，将以下容器注入标有您指定名称的 Kubernetes 容器组 (pod) 中。

- App Mesh Envoy 代理 - Envoy 使用在 App Mesh 控制面板中定义的配置来确定将应用程序流量发送到何处。
- App Mesh 代理路由管理器 - 更新容器组 (pod) 网络命名空间中通过 Envoy 路由传入和传出流量的 `iptables` 规则。此容器作为 Kubernetes init 容器运行在容器组 (pod) 中。

先决条件

- 已了解 App Mesh 概念。有关更多信息，请参阅 什么是 AWS App Mesh？。
- 已对 Kubernetes 概念有所了解。有关更多信息，请参阅 Kubernetes 文档中的 什么是 Kubernetes。
- 现有 Kubernetes 集群。如果您还没有集群，请参阅《Amazon EKS 用户指南》中的 Amazon EKS 入门。如果您在亚马逊 EC2 上运行自己的 Kubernetes 集群，请确保 Docker 已通过 Envoy 镜像所在的 Amazon ECR 存储库的身份验证。有关更多信息，请参阅 Envoy 镜像、《亚马逊弹性容器注册表用户指南》中的注册表身份验证和 Kubernetes 文档中的从私有注册表中提取镜像。
- App Mesh 支持在 DNS 中注册的 Linux 服务 AWS Cloud Map，或者两者兼而有之。要使用此入门指南，我们建议您提供三项已注册到 DNS 的现有服务。该主题的剩余步骤假定现有服务命名为 `serviceA`、`serviceB` 和 `serviceBv2`，并且可以在名为 `apps.local` 的命名空间中发现所有服务。

  即使服务不存在，您也可以创建服务网格及其资源，但在部署实际服务之前，您无法使用网格。
- 已安装 AWS CLI 版本 1.18.116 或更高版本或 2.0.38 或更高版本。要安装或升级 AWS CLI，请参阅安装 AWS CLI。
- 配置为与您的 Kubernetes 集群通信的 `kubectl` 客户端。如果您正在使用 Amazon Elastic Kubernetes Service，则可以使用用于安装 `kubectl` 和配置 `kubeconfig` 文件的说明。
- Helm 版本 3.0 或更高版本已安装。如果您没有安装 Helm，请参阅《亚马逊 EKS 用户指南》中的将 Helm 与亚马逊 EKS 配合使用。
- Amazon EKS 目前 `IPv6_ONLY` 仅支持 `IPv4_ONLY` 且仅支持 IP 首选项，因为 Amazon EKS 目前仅支持能够仅提供 `IPv4` 流量或仅 `IPv6` 流量的容器组 (pod)。

剩余步骤假定实际服务命名为 `serviceA`、`serviceB` 和 `serviceBv2`，并且可以在名为 `apps.local` 的命名空间中发现所有服务。

步骤 1：安装集成组件

在要托管与 App Mesh 结合使用的容器组 (pod) 的每个集群中，安装一次集成组件。

安装集成组件

1. 此过程的剩余步骤需要一个没有安装预发行版控制器的集群。如果您已安装预发行版，或者不确定是否已安装，则可以下载并运行一个脚本，检查您的集群是否安装了预发行版。

   ```
   curl -o pre_upgrade_check.sh https://raw.githubusercontent.com/aws/eks-charts/
   master/stable/appmesh-controller/upgrade/pre_upgrade_check.sh
   sh ./pre_upgrade_check.sh
   ```

   如果脚本返回 `Your cluster is ready for upgrade. Please proceed to the installation instructions`，则可以继续执行下一步。如果返回不同的消息，则需要先完成升级步骤，然后再继续。有关升级预发行版本的更多信息，请参阅升级。 GitHub

2. 将 `eks-charts` 存储库添加到 Helm。

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

3. 安装 App Mesh Kubernetes 自定义资源定义 (CRD)。

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

4. 为控制器创建一个 Kubernetes 命名空间。

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

5. 设置以下变量，以便在后续步骤中使用。将 *`cluster-name`* 和 *`Region-code`* 替换为现有集群的值。

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

6. （可选）如果您想在 Fargate 上运行控制器，则需要创建一个 Fargate 配置文件。如果您尚未 `eksctl` 安装，请参阅 《Amazon EKS 用户指南》*`eksctl`*中的安装或升级。如果您希望使用控制台创建配置文件，请参阅《Amazon EKS 用户指南》中的创建 Fargate 配置文件。

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

7. 为您的集群创建 OpenID Connect (OIDC) 身份提供商。如果未安装 `eksctl`，则可以按照《Amazon EKS 用户指南》中安装或升级 `eksctl`上的说明安装它。如果您希望使用控制台创建提供商，请参阅《Amazon EKS 用户指南》中的在集群上为服务账户启用 IAM 角色。

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

8. 创建 IAM 角色，为其附加AWSAppMeshFullAccess和AWSCloudMapFullAccess AWS 托管策略，然后将其绑定到 `appmesh-controller` Kubernetes 服务账户。该角色使控制器能够添加、删除和更改 App Mesh 资源。

> **Note**
>
> 该命令使用自动生成的名称创建一个 AWS IAM 角色。您无法指定创建的 IAM 角色名称。

```
eksctl create iamserviceaccount \
    --cluster $CLUSTER_NAME \
    --namespace appmesh-system \
    --name appmesh-controller \
    --attach-policy-arn  arn:aws:iam::aws:policy/
AWSCloudMapFullAccess,arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
    --override-existing-serviceaccounts \
    --approve
```

如果您更喜欢使用 AWS Management Console 或创建服务账户 AWS CLI，请参阅 A mazon EKS 用户指南中的为您的服务账户创建 IAM 角色和策略。如果您使用 AWS Management Console 或 AWS CLI 来创建账户，则还需要将该角色映射到 Kubernetes 服务帐号。有关更多信息，请参阅《Amazon EKS 用户指南》中的为集群上的服务账户指定 IAM 角色。

9. 部署 App Mesh 控制器。有关所有配置选项的列表，请参阅上的配置 GitHub。

   1. 要为私有集群部署 App Mesh 控制器，必须先对链接的私有子网启用 App Mesh 和服务发现 Amazon VPC 端点。您还需要设置 `accountId`。

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

要在私有集群中启用 X-Ray 跟踪，请启用 X-Ray 和 Amazon ECR Amazon VPC 端点。控制器默认使用 `public.ecr.aws/xray/aws-xray-daemon:latest`，因此请将此映像拉到本地并将其推送到您的个人 ECR 存储库。

> **Note**
>
> Amazon VPC 端点目前不支持 Amazon ECR 公有存储库。

以下示例显示如何使用 X-Ray 配置部署控制器。

```
helm upgrade -i appmesh-controller eks/appmesh-controller \
    --namespace appmesh-system \
    --set region=$AWS_REGION \
    --set serviceAccount.create=false \
    --set serviceAccount.name=appmesh-controller \
    --set accountId=$AWS_ACCOUNT_ID \
    --set log.level=debug \
    --set tracing.enabled=true \
    --set tracing.provider=x-ray \
    --set xray.image.repository=your-account-id.dkr.ecr.your-
region.amazonaws.com/your-repository \
    --set xray.image.tag=your-xray-daemon-image-tag
```

在将应用程序部署与虚拟节点或网关绑定时，请验证 X-Ray 进程守护程序是否成功注入。

有关更多信息，请参阅《Amazon EKS 用户指南》中的私有集群。

2. 为其他集群部署 App Mesh 控制器。有关所有配置选项的列表，请参阅上的配置 GitHub。

```
helm upgrade -i appmesh-controller eks/appmesh-controller \
    --namespace appmesh-system \
    --set region=$AWS_REGION \
    --set serviceAccount.create=false \
    --set serviceAccount.name=appmesh-controller
```

> **Note**
>
> 如果您的 Amazon EKS 集群系列是 IPv6，请在部署 App Mesh 控制器时通过在前面的命令中添加以下选项来设置集群名称 `--set clusterName=$CLUSTER_NAME`。

> **Important**
>
> 如果您的集群位于`me-south-1`、`ap-east-1`、`ap-southeast-3`、`eu-south-1`、`il-central-1` 或 `af-south-1` 区域中，则需要在前面的命令中添加以下选项：
> 将*## ID* 和*####*替换为相应的值集之一。
>
> - 对于 sidecar 映像：
>   - ```
>     --set image.repository=account-id.dkr.ecr.Region-code.amazonaws.com/amazon/appmesh-controller
>     ```
>   - 772975370895.dkr。ecr.me-south-1.amazonaws.com/: v1.27.3.0-prod aws-appmesh-envoy
>   - 856666278305.dkr。ecr.ap-east-1.amazonaws.com/: v1.27.3.0-prod aws-appmesh-envoy
>   - 909464085924.dkr。ecr.ap-southeast-3.amazonaws.com/: v1.27.3.0-prod aws-appmesh-envoy
>   - 422531588944.dkr。ecr.eu-south-1.amazonaws.com/: v1.27.3.0-prod aws-appmesh-envoy
>   - 564877687649.dkr。ecr.il-central-1.amazonaws.com/: v1.27.3.0-prod aws-appmesh-envoy
>   - 924023996002.dkr。ecr.af-south-1.amazonaws.com/: v1.27.3.0-prod aws-appmesh-envoy
> - 可以在更改日志中找到较旧的图像 URI。 GitHub显示图像的 AWS 账户版本已更改`v1.5.0`。旧版本的镜像托管在亚马逊 Elastic Kubernetes Service 亚马逊容器镜像注册表上的 AWS 账户中。

- 对于控制器映像：
  - ```
    --set sidecar.image.repository=account-id.dkr.ecr.Region-code.amazonaws.com/aws-appmesh-envoy
    ```
  - 772975370895.dkr.ecr.me-south-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 856666278305.dkr.ecr.ap-east-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 909464085924.dkr.ecr.ap-southeast-3.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 422531588944.dkr.ecr.eu-south-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 564877687649.dkr.ecr.il-central-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
  - 924023996002.dkr.ecr.af-south-1.amazonaws.com/amazon/appmesh-controller:v1.12.3
- 对于 sidecar 初始化映像：
  - ```
    --set sidecar.image.repository=account-id.dkr.ecr.Region-code.amazonaws.com/aws-appmesh-envoy
    ```
  - 772975370895.dkr。ecr.me-south-1.amazonaws.com /-manager: v7-prod aws-appmesh-proxy-route
  - 856666278305.dkr。ecr.ap-east-1.amazonaws.com /-manager: v7-prod aws-appmesh-proxy-route
  - 909464085924.dkr。ecr.ap-southeast-3.amazonaws.com /-manager: v7-prod aws-appmesh-proxy-route
  - 422531588944.dkr。ecr.eu-south-1.amazonaws.com /-manager: v7-prod aws-appmesh-proxy-route
  - 564877687649.dkr。ecr.il-central-1.amazonaws.com /-manager: v7-prod aws-appmesh-proxy-route
  - 924023996002.dkr。ecr.af-south-1.amazonaws.com /-manager: v7-prod aws-appmesh-proxy-route

> **Important**
>
> 仅支持将版本 v1.9.0.0-prod 或更高版本与 App Mesh 一起使用。

10. 确认控制器版本为 `v1.4.0` 或更高版本。您可以登录查看变更日志 GitHub。

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

> **Note**
>
> 如果您查看正在运行的容器的日志，您可能会看到有一行包含以下文本，这行文本可以安全地忽略。
>
> ```
> Neither -kubeconfig nor -master was specified. Using the inClusterConfig.
>  This might not work.
> ```

步骤 2：部署 App Mesh 资源

当您在 Kubernetes 中部署应用程序时，您也会创建 Kubernetes 自定义资源，以便控制器可以创建相应的 App Mesh 资源。以下过程可帮助您部署具有某些功能的 App Mesh 资源。您可以在 App Mesh 演练中列出的许多功能文件夹的`v1beta2`子文件夹中找到部署其他 App Mesh 资源功能的示例清单。GitHub

> **Important**
>
> 控制器创建了 App Mesh 资源后，建议您仅使用控制器对 App Mesh 资源进行更改或删除资源。如果您使用 App Mesh 对资源进行更改或删除，则默认情况下，控制器将在十小时内不会更改或重新创建所更改或删除的 App Mesh 资源。您可以将此持续时间配置为更短的时间。有关更多信息，请参阅上的 “配置” GitHub。

部署 App Mesh 资源

1. 创建要将 App Mesh 资源部署到的 Kubernetes 命名空间。

a. 将以下内容保存到计算机上名为 namespace.yaml 的文件中。

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

b. 创建命名空间。

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

2. 创建 App Mesh 服务网格

a. 将以下内容保存到计算机上名为 mesh.yaml 的文件中。该文件用于创建名为 *my-mesh* 的网格资源。服务网格是一种用于驻留在其内的服务之间的网络流量的逻辑边界。

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

b. 创建网格。

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

c. 查看已创建的 Kubernetes 网格资源的详细信息。

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

输出

```
Name:         my-mesh
Namespace:
Labels:       <none>
```

```
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"appmesh.k8s.aws/
v1beta2","kind":"Mesh","metadata":{"annotations":{},"name":"my-mesh"},"spec":
{"namespaceSelector":{"matchLa...
API Version:  appmesh.k8s.aws/v1beta2
Kind:         Mesh
Metadata:
  Creation Timestamp:  2020-06-17T14:51:37Z
  Finalizers:
    finalizers.appmesh.k8s.aws/mesh-members
    finalizers.appmesh.k8s.aws/aws-appmesh-resources
  Generation:        1
  Resource Version:  6295
  Self Link:         /apis/appmesh.k8s.aws/v1beta2/meshes/my-mesh
  UID:               111a11b1-c11d-1e1f-gh1i-j11k1l111m711
Spec:
  Aws Name:  my-mesh
  Namespace Selector:
    Match Labels:
      Mesh:  my-mesh
Status:
  Conditions:
    Last Transition Time:  2020-06-17T14:51:37Z
    Status:                True
    Type:                  MeshActive
  Mesh ARN:                arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh
  Observed Generation:     1
Events:                    <none>
```

d. 查看有关控制器创建的 App Mesh 服务网格的详细信息。

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

输出

```
{
    "mesh": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh",
            "createdAt": "2020-06-17T09:51:37.920000-05:00",
            "lastUpdatedAt": "2020-06-17T09:51:37.920000-05:00",
            "meshOwner": "111122223333",
```

```
            "resourceOwner": "111122223333",
            "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
            "version": 1
        },
        "spec": {},
        "status": {
            "status": "ACTIVE"
        }
    }
}
```

3. 创建 App Mesh 虚拟节点。虚拟节点充当指向 Kubernetes 部署的逻辑指针。

   a. 将以下内容保存到计算机上名为 `virtual-node.yaml` 的文件中。该文件用于创建命名*my-apps*空间 *my-service-a* 中命名的 App Mesh 虚拟节点。虚拟节点表示在后续步骤中创建的 Kubernetes 服务。`hostname` 的值是此虚拟节点所代表的实际服务的完全限定 DNS 主机名。

```
apiVersion: appmesh.k8s.aws/v1beta2
kind: VirtualNode
metadata:
  name: my-service-a
  namespace: my-apps
spec:
  podSelector:
    matchLabels:
      app: my-app-1
  listeners:
    - portMapping:
        port: 80
        protocol: http
  serviceDiscovery:
    dns:
      hostname: my-service-a.my-apps.svc.cluster.local
```

   虚拟节点具有本教程中未涉及的功能，例如 end-to-end 加密和运行状况检查。有关更多信息，请参阅 the section called “虚拟节点”。若要查看可在上述规范中设置的虚拟节点的所有可用设置，请运行以下命令。

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

   b. 部署虚拟节点。

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

c. 查看已创建的 Kubernetes 虚拟节点资源的详细信息。

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

输出

```
Name:         my-service-a
Namespace:    my-apps
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"appmesh.k8s.aws/
v1beta2","kind":"VirtualNode","metadata":{"annotations":{},"name":"my-service-
a","namespace":"my-apps"},"s...
API Version:  appmesh.k8s.aws/v1beta2
Kind:         VirtualNode
Metadata:
  Creation Timestamp:  2020-06-17T14:57:29Z
  Finalizers:
    finalizers.appmesh.k8s.aws/aws-appmesh-resources
  Generation:        2
  Resource Version:  22545
  Self Link:         /apis/appmesh.k8s.aws/v1beta2/namespaces/my-apps/
virtualnodes/my-service-a
  UID:               111a11b1-c11d-1e1f-gh1i-j11k1l111m711
Spec:
  Aws Name:  my-service-a_my-apps
  Listeners:
    Port Mapping:
      Port:      80
      Protocol:  http
  Mesh Ref:
    Name:  my-mesh
    UID:   111a11b1-c11d-1e1f-gh1i-j11k1l111m711
  Pod Selector:
    Match Labels:
      App:  nginx
  Service Discovery:
    Dns:
      Hostname:  my-service-a.my-apps.svc.cluster.local
Status:
```

```
  Conditions:
    Last Transition Time:  2020-06-17T14:57:29Z
    Status:                True
    Type:                  VirtualNodeActive
  Observed Generation:     2
  Virtual Node ARN:        arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualNode/my-service-a_my-apps
Events:                    <none>
```

d. 查看控制器在 App Mesh 中创建的虚拟节点的详细信息。

> **Note**
>
> 尽管在 Kubernetes 中创建的虚拟节点的名称为 *my-service-a*，但在 App Mesh 中创建的虚拟节点的名称为 *my-service-a_my-apps*。控制器在创建 App Mesh 资源时，将 Kubernetes 命名空间名称附加到 App Mesh 虚拟节点名称。添加命名空间名称是因为在 Kubernetes 中，您可以在不同的命名空间中创建具有相同名称的虚拟节点，但在 App Mesh 中，虚拟节点名称在网格内必须唯一。

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

输出

```
{
    "virtualNode": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualNode/my-service-a_my-apps",
            "createdAt": "2020-06-17T09:57:29.840000-05:00",
            "lastUpdatedAt": "2020-06-17T09:57:29.840000-05:00",
            "meshOwner": "111122223333",
            "resourceOwner": "111122223333",
            "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
            "version": 1
        },
        "spec": {
            "backends": [],
            "listeners": [
```

```
                {
                    "portMapping": {
                        "port": 80,
                        "protocol": "http"
                    }
                }
            ],
            "serviceDiscovery": {
                "dns": {
                    "hostname": "my-service-a.my-apps.svc.cluster.local"
                }
            }
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualNodeName": "my-service-a_my-apps"
    }
}
```

4. 创建 App Mesh 虚拟路由器。虚拟路由器处理用于您的网格内一个或多个虚拟服务的流量。

   a. 将以下内容保存到计算机上名为 `virtual-router.yaml` 的文件中。该文件用于创建虚拟路由器，将流量路由到在上一步中创建的名为 `my-service-a` 的虚拟节点。控制器创建 App Mesh 虚拟路由器和路由资源。您可以为路由指定更多功能，以及使用 `http` 以外的协议。有关更多信息，请参阅 the section called “虚拟路由器” 和 the section called “路由”。请注意，引用的虚拟节点名称是 Kubernetes 虚拟节点名称，而不是控制器在 App Mesh 中创建的 App Mesh 虚拟节点名称。

```
apiVersion: appmesh.k8s.aws/v1beta2
kind: VirtualRouter
metadata:
  namespace: my-apps
  name: my-service-a-virtual-router
spec:
  listeners:
    - portMapping:
        port: 80
        protocol: http
  routes:
    - name: my-service-a-route
      httpRoute:
        match:
```

```
          prefix: /
        action:
          weightedTargets:
            - virtualNodeRef:
                name: my-service-a
              weight: 1
```

（可选）若要查看可在上述规范中设置的虚拟路由器的所有可用设置，请运行以下命令。

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

若要查看可在上述规范中设置的路由的所有可用设置，请运行以下命令。

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

b. 部署虚拟路由器。

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

c. 查看已创建的 Kubernetes 虚拟路由器资源。

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

缩减的输出

```
Name:         my-service-a-virtual-router
Namespace:    my-apps
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"appmesh.k8s.aws/
v1beta2","kind":"VirtualRouter","metadata":{"annotations":{},"name":"my-
service-a-virtual-router","namespac...
API Version:  appmesh.k8s.aws/v1beta2
Kind:         VirtualRouter
...
Spec:
  Aws Name:  my-service-a-virtual-router_my-apps
  Listeners:
    Port Mapping:
      Port:      80
      Protocol:  http
```

```
  Mesh Ref:
    Name:  my-mesh
    UID:   111a11b1-c11d-1e1f-gh1i-j11k1l111m711
  Routes:
    Http Route:
      Action:
        Weighted Targets:
          Virtual Node Ref:
            Name:  my-service-a
          Weight:  1
      Match:
        Prefix:  /
    Name:        my-service-a-route
Status:
  Conditions:
    Last Transition Time:  2020-06-17T15:14:01Z
    Status:                True
    Type:                  VirtualRouterActive
  Observed Generation:     1
  Route AR Ns:
    My - Service - A - Route:  arn:aws:appmesh:us-west-2:111122223333:mesh/my-
mesh/virtualRouter/my-service-a-virtual-router_my-apps/route/my-service-a-route
  Virtual Router ARN:          arn:aws:appmesh:us-west-2:111122223333:mesh/my-
mesh/virtualRouter/my-service-a-virtual-router_my-apps
Events:                        <none>
```

d. 查看控制器在 App Mesh 中创建的虚拟路由器资源。您可以为 name 指定 `my-service-a-virtual-router_my-apps`，因为当控制器在 App Mesh 中创建虚拟路由器时，它会在虚拟路由器的名称后附加 Kubernetes 命名空间名称。

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

输出

```
{
    "virtualRouter": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualRouter/my-service-a-virtual-router_my-apps",
            "createdAt": "2020-06-17T10:14:01.547000-05:00",
```

```
            "lastUpdatedAt": "2020-06-17T10:14:01.547000-05:00",
            "meshOwner": "111122223333",
            "resourceOwner": "111122223333",
            "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
            "version": 1
        },
        "spec": {
            "listeners": [
                {
                    "portMapping": {
                        "port": 80,
                        "protocol": "http"
                    }
                }
            ]
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualRouterName": "my-service-a-virtual-router_my-apps"
    }
}
```

e. 查看控制器在 App Mesh 中创建的路由资源。未在 Kubernetes 中创建路由资源，因为该路由是 Kubernetes 中虚拟路由器配置的一部分。路由信息显示在子步骤 c 的 Kubernetes 资源详细信息中。控制器在 App Mesh 中创建路由时，不将 Kubernetes 命名空间名称附加到 App Mesh 路由名称，因为路由名称对虚拟路由器唯一。

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

输出

```
{
    "route": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualRouter/my-service-a-virtual-router_my-apps/route/my-service-a-route",
            "createdAt": "2020-06-17T10:14:01.577000-05:00",
```

```
            "lastUpdatedAt": "2020-06-17T10:14:01.577000-05:00",
            "meshOwner": "111122223333",
            "resourceOwner": "111122223333",
            "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
            "version": 1
        },
        "routeName": "my-service-a-route",
        "spec": {
            "httpRoute": {
                "action": {
                    "weightedTargets": [
                        {
                            "virtualNode": "my-service-a_my-apps",
                            "weight": 1
                        }
                    ]
                },
                "match": {
                    "prefix": "/"
                }
            }
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualRouterName": "my-service-a-virtual-router_my-apps"
    }
}
```

5. 创建 App Mesh 虚拟服务。虚拟服务是一种抽象的实际服务，由虚拟节点直接提供或通过虚拟路由器的方式间接提供。从属服务按名称调用您的虚拟服务。虽然名称对于 App Mesh 并不重要，但建议将虚拟服务命名为虚拟服务所代表的实际服务的完全限定域名。通过这种方式命名虚拟服务，您无需更改应用程序代码即可引用其他名称。请求路由到指定作为虚拟服务提供程序的虚拟节点或虚拟路由器。

   a. 将以下内容保存到计算机上名为 `virtual-service.yaml` 的文件中。该文件用于创建虚拟服务，使用虚拟路由器提供程序将流量路由到在上一步中创建的名为 `my-service-a` 的虚拟节点。`spec` 中 `awsName` 的值是此虚拟服务所提取的实际 Kubernetes 服务的完全限定域名 (FQDN)。Kubernetes 服务在 the section called “步骤 3：创建或更新服务” 中创建。有关更多信息，请参阅the section called “虚拟服务”。

```
apiVersion: appmesh.k8s.aws/v1beta2
```

```
kind: VirtualService
metadata:
  name: my-service-a
  namespace: my-apps
spec:
  awsName: my-service-a.my-apps.svc.cluster.local
  provider:
    virtualRouter:
      virtualRouterRef:
        name: my-service-a-virtual-router
```

若要查看可在上述规范中设置的虚拟服务的所有可用设置，请运行以下命令。

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

b. 创建虚拟服务。

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

c. 查看已创建的 Kubernetes 虚拟服务资源的详细信息。

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

输出

```
Name:         my-service-a
Namespace:    my-apps
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"appmesh.k8s.aws/
v1beta2","kind":"VirtualService","metadata":{"annotations":{},"name":"my-
service-a","namespace":"my-apps"}...
API Version:  appmesh.k8s.aws/v1beta2
Kind:         VirtualService
Metadata:
  Creation Timestamp:  2020-06-17T15:48:40Z
  Finalizers:
    finalizers.appmesh.k8s.aws/aws-appmesh-resources
  Generation:        1
  Resource Version:  13598
  Self Link:         /apis/appmesh.k8s.aws/v1beta2/namespaces/my-apps/
virtualservices/my-service-a
```

```
  UID:              111a11b1-c11d-1e1f-gh1i-j11k1l111m711
Spec:
  Aws Name:  my-service-a.my-apps.svc.cluster.local
  Mesh Ref:
    Name:  my-mesh
    UID:   111a11b1-c11d-1e1f-gh1i-j11k1l111m711
  Provider:
    Virtual Router:
      Virtual Router Ref:
        Name:  my-service-a-virtual-router
Status:
  Conditions:
    Last Transition Time:  2020-06-17T15:48:40Z
    Status:                True
    Type:                  VirtualServiceActive
  Observed Generation:     1
  Virtual Service ARN:     arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualService/my-service-a.my-apps.svc.cluster.local
Events:                    <none>
```

d. 查看控制器在 App Mesh 中创建的虚拟服务资源的详细信息。Kubernetes 控制器在 App Mesh 中创建虚拟服务时，不将 Kubernetes 命名空间名称附加到 App Mesh 虚拟服务名称，因为虚拟服务的名称是唯一的 FQDN。

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

输出

```
{
    "virtualService": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualService/my-service-a.my-apps.svc.cluster.local",
            "createdAt": "2020-06-17T10:48:40.182000-05:00",
            "lastUpdatedAt": "2020-06-17T10:48:40.182000-05:00",
            "meshOwner": "111122223333",
            "resourceOwner": "111122223333",
            "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
            "version": 1
        },
```

```
        "spec": {
            "provider": {
                "virtualRouter": {
                    "virtualRouterName": "my-service-a-virtual-router_my-apps"
                }
            }
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualServiceName": "my-service-a.my-apps.svc.cluster.local"
    }
}
```

尽管本教程中没有介绍，但控制器也可以部署 App Mesh the section called “虚拟网关” 和 the section called “网关路由”。有关使用控制器部署这些资源的演练，请参阅配置入站网关或包含上 GitHub资源的示例清单。

步骤 3：创建或更新服务

您想要和 App Mesh 配合使用的任何容器组 (pod) 必须已经添加了 App Mesh sidecar 容器。注入器会自动将 sidecar 容器添加到使用您指定的标签部署的任意容器组 (pod) 中。

1. 启用代理授权。我们建议您启用每个 Kubernetes 部署，以仅流式传输自己的 App Mesh 虚拟节点的配置。

   a. 将以下内容保存到计算机上名为 `proxy-auth.json` 的文件中。请确保将 *alternate-colored values* 替换为您自己的值。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appmesh:StreamAggregatedResources",
            "Resource": [
                "arn:aws:appmesh:Region-code:111122223333:mesh/my-mesh/
virtualNode/my-service-a_my-apps"
            ]
        }
    ]
```

```
}
```

b. 创建 策略。

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

c. 创建 IAM 角色，将您在上一步中创建的策略附加到该角色，创建 Kubernetes 服务账户并将策略绑定到该 Kubernetes 服务账户。该角色使控制器能够添加、删除和更改 App Mesh 资源。

```
eksctl create iamserviceaccount \
    --cluster $CLUSTER_NAME \
    --namespace my-apps \
    --name my-service-a \
    --attach-policy-arn  arn:aws:iam::111122223333:policy/my-policy \
    --override-existing-serviceaccounts \
    --approve
```

如果您更喜欢使用 AWS Management Console 或创建服务账户 AWS CLI，请参阅 A mazon EKS 用户指南中的为您的服务账户创建 IAM 角色和策略。如果您使用 AWS Management Console 或 AWS CLI 来创建账户，则还需要将该角色映射到 Kubernetes 服务帐号。有关更多信息，请参阅《Amazon EKS 用户指南》中的为集群上的服务账户指定 IAM 角色。

2. （可选）如果您的部署要部署到 Fargate 容器组 (pod)，则需要创建 Fargate 配置文件。如果未安装 `eksctl`，则可以按照《Amazon EKS 用户指南》中安装或升级 `eksctl`上的说明安装它。如果您希望使用控制台创建配置文件，请参阅《Amazon EKS 用户指南》中的创建 Fargate 配置文件。

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

3. 创建 Kubernetes 服务和部署。如果您有要和 App Mesh 配合使用的现有部署，则需要部署一个虚拟节点，就像在 the section called “步骤 2：部署 App Mesh 资源” 的子步骤 3 中所做的那样。更新您的部署，确保其标签与您在虚拟节点上设置的标签相匹配，这样 sidecar 容器就会自动添加到容器组 (pod) 中，并重新部署容器组 (pod)。

a. 将以下内容保存到计算机上名为 `example-service.yaml` 的文件中。如果您更改命名空间名称并使用 Fargate 容器组 (pod)，请确保命名空间名称与您在 Fargate 配置文件中定义的命名空间名称匹配。

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: my-service-a
  namespace: my-apps
  labels:
    app: my-app-1
spec:
  selector:
    app: my-app-1
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-service-a
  namespace: my-apps
  labels:
    app: my-app-1
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app-1
  template:
    metadata:
      labels:
        app: my-app-1
    spec:
      serviceAccountName: my-service-a
      containers:
      - name: nginx
        image: nginx:1.19.0
        ports:
        - containerPort: 80
```

Important

规范中 app `matchLabels` `selector` 中的值必须与您在 the section called “步骤 2：部署 App Mesh 资源” 的子步骤 3 中创建虚拟节点时指定的值相匹配，否则 sidecar 容器将不会注入到容器组 (pod)中。在前面的示例中，标签的值为 my-

app-1。如果您部署的是虚拟网关而不是虚拟节点，则 Deployment 清单应仅包含 Envoy 容器。有关要使用的镜像的更多信息，请参阅 Envoy。如需查看样本 manfest，请参阅中的部署示例。 GitHub

b. 部署服务。

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

c. 查看服务和部署。

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

输出

```
NAME                                READY   STATUS    RESTARTS   AGE
my-service-a-54776556f6-2cxd9   2/2     Running   0          10s
my-service-a-54776556f6-w26kf   2/2     Running   0          18s
my-service-a-54776556f6-zw5kt   2/2     Running   0          26s
```

d. 查看已部署的容器组 (pod) 之一的详细信息。

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

缩减的输出

```
Name:         my-service-a-54776556f6-2cxd9
Namespace:    my-app-1
Priority:     0
Node:         ip-192-168-44-157.us-west-2.compute.internal/192.168.44.157
Start Time:   Wed, 17 Jun 2020 11:08:59 -0500
Labels:       app=nginx
              pod-template-hash=54776556f6
Annotations:  kubernetes.io/psp: eks.privileged
Status:       Running
IP:           192.168.57.134
IPs:
  IP:           192.168.57.134
Controlled By:  ReplicaSet/my-service-a-54776556f6
Init Containers:
  proxyinit:
```

```
    Container ID:   docker://
e0c4810d584c21ae0cb6e40f6119d2508f029094d0e01c9411c6cf2a32d77a59
    Image:          111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
proxy-route-manager:v2
    Image ID:       docker-pullable://111345817488.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-proxy-route-manager
    Port:           <none>
    Host Port:      <none>
    State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Fri, 26 Jun 2020 08:36:22 -0500
      Finished:     Fri, 26 Jun 2020 08:36:22 -0500
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:     10m
      memory:  32Mi
    Environment:
      APPMESH_START_ENABLED:         1
      APPMESH_IGNORE_UID:            1337
      APPMESH_ENVOY_INGRESS_PORT:    15000
      APPMESH_ENVOY_EGRESS_PORT:     15001
      APPMESH_APP_PORTS:             80
      APPMESH_EGRESS_IGNORED_IP:     169.254.169.254
      APPMESH_EGRESS_IGNORED_PORTS:  22
      AWS_ROLE_ARN:                  arn:aws:iam::111122223333:role/eksctl-app-
mesh-addon-iamserviceaccount-my-a-Role1-NMNCVWB6PL0N
      AWS_WEB_IDENTITY_TOKEN_FILE:   /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
    ...
Containers:
  nginx:
    Container ID:   docker://
be6359dc6ecd3f18a1c87df7b57c2093e1f9db17d5b3a77f22585ce3bcab137a
    Image:          nginx:1.19.0
    Image ID:       docker-pullable://nginx
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 26 Jun 2020 08:36:28 -0500
    Ready:          True
    Restart Count:  0
    Environment:
```

```
      AWS_ROLE_ARN:                 arn:aws:iam::111122223333:role/eksctl-app-
mesh-addon-iamserviceaccount-my-a-Role1-NMNCVWB6PL0N
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
    ...
  envoy:
    Container ID:
 docker://905b55cbf33ef3b3debc51cb448401d24e2e7c2dbfc6a9754a2c49dd55a216b6
    Image:          840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.12.4.0-prod
    Image ID:       docker-pullable://840364872350.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-envoy
    Port:           9901/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 26 Jun 2020 08:36:36 -0500
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:     10m
      memory:  32Mi
    Environment:
      APPMESH_RESOURCE_ARN:         arn:aws:iam::111122223333:mesh/my-mesh/
virtualNode/my-service-a_my-apps
      APPMESH_PREVIEW:              0
      ENVOY_LOG_LEVEL:              info
      AWS_REGION:                   us-west-2
      AWS_ROLE_ARN:                 arn:aws:iam::111122223333:role/eksctl-app-
mesh-addon-iamserviceaccount-my-a-Role1-NMNCVWB6PL0N
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
...
Events:
  Type    Reason     Age   From
   Message
  ----    ------     ----  ----
   -------
  Normal  Pulling    30s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Pulling image "111345817488.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-proxy-route-manager:v2"
  Normal  Pulled     23s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Successfully pulled image "111345817488.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-proxy-route-manager:v2"
```

```
  Normal  Created    21s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Created container proxyinit
  Normal  Started    21s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Started container proxyinit
  Normal  Pulling    20s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Pulling image "nginx:1.19.0"
  Normal  Pulled     16s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Successfully pulled image "nginx:1.19.0"
  Normal  Created    15s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Created container nginx
  Normal  Started    15s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Started container nginx
  Normal  Pulling    15s   kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Pulling image "840364872350.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-envoy:v1.12.4.0-prod"
  Normal  Pulled     8s    kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Successfully pulled image "840364872350.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-envoy:v1.12.4.0-prod"
  Normal  Created    7s    kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Created container envoy
  Normal  Started    7s    kubelet, ip-192-168-44-157.us-
west-2.compute.internal  Started container envoy
```

在前面的输出中，您可以看到控制器已将 `proxyinit` 和 `envoy` 容器添加到容器组 (pod)。如果您将示例服务部署到 Fargate，则控制器会将 `envoy` 容器添加到容器组 (pod) 中，但 `proxyinit` 容器不是。

4. （可选）安装插件，例如 Prometheus、Grafana、Jaeger 和 Datadog。 AWS X-Ray有关更多信息，请参阅 App Mesh 插件 GitHub 和《App Mesh 用户指南》的 “可观察性” 部分。

> **Note**
>
> 有关 App Mesh 的更多示例和演练，请参阅 App Mesh 示例存储库。

步骤 4：清除

删除本教程中创建的所有示例资源。控制器还会删除在 `my-mesh` App Mesh 服务网格中创建的资源。

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

如果您为示例服务创建了 Fargate 配置文件，请将其删除。

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

删除网格。

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

（可选）您可以删除 Kubernetes 集成组件。

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

（可选）如果您将 Kubernetes 集成组件部署到 Fargate，请删除 Fargate 配置文件。

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

开始使用 AWS App Mesh 和 Amazon EC2

本主题可帮助您 AWS App Mesh 使用在 Amazon EC2 上运行的实际服务。本教程介绍了多种 App Mesh 资源类型的基本功能。

场景

为了说明如何配合使用 App Mesh，假定您有一个具有以下功能的应用程序：

- 由名为 `serviceA` 和 `serviceB` 的两项服务组成。
- 这两项服务均注册到名为 `apps.local` 的命名空间。
- `ServiceA` 通过 HTTP/2 和端口 80 与 `serviceB` 通信。
- 您已部署 `serviceB` 的版本 2，并采用名称 `serviceBv2` 在 `apps.local` 命名空间中注册了该服务。

您的要求如下：

- 您希望将 75% 的流量从 `serviceA` 发送到 `serviceB`，并将 25% 的流量发送到 `serviceBv2`，以验证在将 100% 的流量从 `serviceA` 发送到它之前，`serviceBv2` 没有错误。

- 您希望能够轻松调整流量权重，以便一旦证明 serviceBv2 可靠便可将 100% 的流量发送到该服务。将所有流量发送到 serviceBv2 后，您希望弃用 serviceB。
- 您不想为了满足之前的要求而更改实际服务的任何现有应用程序代码或服务发现注册。

为了满足您的要求，您决定创建包含虚拟服务、虚拟节点、虚拟路由器和路由的 App Mesh 服务网格。实施您的网格后，更新使用 Envoy 代理的服务。更新后，您的服务将通过 Envoy 代理相互通信，而不是直接相互通信。

先决条件

App Mesh 支持在 DNS 中注册的 Linux 服务 AWS Cloud Map，或者两者兼而有之。要使用此入门指南，我们建议您提供三项已注册到 DNS 的现有服务。即使服务不存在，您也可以创建服务网格及其资源，但在部署实际服务之前，您无法使用网格。

如果您尚未运行服务，则可以启动 Amazon EC2 实例并向其部署应用程序。有关更多信息，请参阅适用于 Linux 实例的《Amazon EC2 用户指南》中的教程：Amazon EC2 Linux 实例入门。剩余步骤假定实际服务命名为 serviceA、serviceB 和 serviceBv2，并且可以在名为 apps.local 的命名空间中发现所有服务。

步骤 1：创建网格和虚拟服务

服务网格是一种用于驻留在其内的服务之间的网络流量的逻辑边界。有关更多信息，请参阅服务网格。虚拟服务是实际服务的抽象。有关更多信息，请参阅虚拟服务。

创建以下 资源：

- 名为 apps 的网格，因为此场景中的所有服务均注册到 apps.local 命名空间。
- 名为 serviceb.apps.local 的虚拟服务，因为虚拟服务表示可以使用该名称发现的服务，并且您不希望更改代码以引用其他名称。稍后的步骤中将添加名为 servicea.apps.local 的虚拟服务。

您可以使用或 1.18.116 AWS Management Console 或更高 AWS CLI 版本或 2.0.38 或更高版本来完成以下步骤。如果使用 AWS CLI，请使用aws --version命令检查已安装的 AWS CLI 版本。如果您没有安装版本 1.18.116 或更高版本或者没有安装版本 2.0.38 或更高版本，则必须安装或更新 AWS CLI。选择要使用的工具所对应的选项卡。

AWS Management Console

1. 打开 App Mesh 控制台首次运行向导，网址为 https://console.aws.amazon.com/appmesh/get-started。
2. 对于网格名称，输入 **apps**。
3. 对于虚拟服务名称，输入 **serviceb.apps.local**。
4. 要继续，请选择 Next。

AWS CLI

1. 使用 `create-mesh` 命令创建网格。

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

2. 使用 `create-virtual-service` 命令创建虚拟服务。

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

步骤 2：创建虚拟节点

虚拟节点充当实际服务的逻辑指针。有关更多信息，请参阅 虚拟节点。

创建名为 `serviceB` 的虚拟节点，因为某个虚拟节点表示名为 `serviceB` 的实际服务。可使用主机名 `serviceb.apps.local`，通过 DNS 发现虚拟节点所表示的实际服务。或者，也可以使用 AWS Cloud Map 发现实际服务。虚拟节点在端口 80 上使用 HTTP/2 协议监听流量。此外，还支持其他协议和运行状况检查。您将在后面的步骤中为 `serviceA` 和 `serviceBv2` 创建虚拟节点。

AWS Management Console

1. 对于虚拟节点名称，输入 **serviceB**。
2. 对于服务发现方法，选择 DNS，并为DNS 主机名输入 **serviceb.apps.local**。
3. 在侦听器配置下，为协议选择 http2，并在端口中，输入 **80**。
4. 要继续，请选择 Next。

AWS CLI

1. 使用以下内容创建名为 create-virtual-node-serviceb.json 的文件：

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

2. 使用 JSON 文件作为输入使用create-virtual-node命令创建虚拟节点。

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

步骤 3：创建虚拟路由器和路由

虚拟路由器路由网格中一个或多个虚拟服务的流量。有关更多信息，请参阅 虚拟路由器 和 路由。

创建以下 资源：

- 名为 serviceB 的虚拟路由器，因为 serviceB.apps.local 虚拟服务不会启动与任何其他服务的出站通信。请记住，您之前创建的虚拟服务是实际 serviceb.apps.local 服务的抽象。虚拟服务将流量发送到虚拟路由器。虚拟路由器采用 HTTP/2 协议在端口 80 上侦听流量。此外，还支持其他协议。

- 名为 serviceB 的路由。它将 100% 的流量路由到serviceB虚拟节点。添加 serviceBv2 虚拟节点后，在稍后的步骤中出现权重。虽然本指南中未作介绍，但您可以为路由添加额外的筛选条件，并添加重试策略，从而使 Envoy 代理在遇到通信问题时会多次尝试将流量发送到虚拟节点。

AWS Management Console

1. 对于虚拟路由器名称，输入 **serviceB**。
2. 在侦听器配置下，为协议选择 http2，并为端口指定 **80**。
3. 对于路由名称，输入 **serviceB**。
4. 对于路由类型，选择 http2。
5. 在目标配置下的虚拟节点名称中，选择 serviceB，并为权重输入 **100**。
6. 在匹配配置下，选择一种方法。
7. 要继续，请选择 Next。

AWS CLI

1. 创建虚拟路由器。

   a. 使用以下内容创建名为 create-virtual-router.json 的文件：

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

   b. 使用 JSON 文件作为输入使用create-virtual-router命令创建虚拟路由器。

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

2. 创建路由。

   a. 使用以下内容创建名为 `create-route.json` 的文件：

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

   b. 采用 JSON 文件作为输入，使用 create-route 命令创建路由。

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

步骤 4：审核并创建

根据之前的说明审核设置。

AWS Management Console

如果需要对任何部分进行任何更改，请选择编辑。在您对设置感到满意后，选择创建网格。

状态屏幕将显示已创建的所有网格资源。您可以通过选择查看网格来在控制台中查看创建的资源。

AWS CLI

使用 describe-mesh 命令查看所创建网格的设置。

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

查看您使用describe-virtual-service命令创建的虚拟服务的设置。

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

查看您使用describe-virtual-node命令创建的虚拟节点的设置。

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

查看您使用describe-virtual-router命令创建的虚拟路由器的设置。

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

使用 describe-route 命令查看所创建路由的设置。

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

步骤 5：创建其他资源

要完成此场景，您需要执行以下操作：

- 创建名为 serviceBv2 的虚拟节点，以及名为 serviceA 的虚拟节点。这两个虚拟节点均通过 HTTP/2 和端口 80 侦听请求。对于 serviceA 虚拟节点，将后端配置为 serviceb.apps.local。来自 serviceA 虚拟节点的所有出站流量都将发送到名为 serviceb.apps.local 的虚拟服务。虽然本指南中未作介绍，但您还可以为虚拟节点指定用于写入访问日志的文件路径。
- 另外创建一个名为的虚拟服务servicea.apps.local，该服务将所有流量直接发送到serviceA虚拟节点。
- 更新在上一步中创建的 serviceB 路由，以将 75% 的流量发送到 serviceB 虚拟节点，将 25% 的流量发送到 serviceBv2 虚拟节点。随着时间的推移，您可以继续修改权重，直到 serviceBv2 收到 100% 的流量。将所有流量发送到 serviceBv2 后，您可以关闭并弃用 serviceB 虚拟节点

和实际服务。在更改权重时，不需要对代码进行任何修改，因为 serviceb.apps.local 虚拟服务名称和实际服务名称没有改变。回想一下，serviceb.apps.local 虚拟服务将流量发送到虚拟路由器，该路由器将流量路由到虚拟节点。虚拟节点的服务发现名称可以随时更改。

AWS Management Console

1. 在左侧导航窗格中，选择网格。
2. 选择在上一步中创建的 apps 网格。
3. 在左侧导航窗格中，选择虚拟节点。
4. 选择创建虚拟节点。
5. 为虚拟节点名称输入 **serviceBv2**，为服务发现方法选择 DNS，并为 DNS 主机名输入 **servicebv2.apps.local**。
6. 对于侦听器配置，为协议选择 http2，并在端口中，输入 **80**。
7. 选择创建虚拟节点。
8. 再次选择创建虚拟节点。对于虚拟节点名称，输入 **serviceA**。对于服务发现方法，选择 DNS，并为 DNS 主机名输入 **servicea.apps.local**。
9. 在新后端下的输入虚拟服务名称中，输入 **serviceb.apps.local**。
10. 在侦听器配置下，为协议选择 http2，并在端口中输入 **80**，然后选择创建虚拟节点。
11. 在左侧导航窗格中，选择虚拟路由器，然后从列表中选择 serviceB 虚拟路由器。
12. 在路由下，选择在上一步中创建的名为 ServiceB 的路由，然后选择编辑。
13. 在目标、虚拟节点名称下，将 serviceB 的权重值更改为 **75**。
14. 选择添加目标，从下拉列表中选择 serviceBv2，然后将权重值设置为 **25**。
15. 选择保存。
16. 在左侧导航窗格中，选择虚拟服务，然后选择创建虚拟服务。
17. 针对虚拟服务名称输入 **servicea.apps.local**，为提供者选择虚拟节点，为虚拟节点选择 serviceA，然后选择创建虚拟服务。

AWS CLI

1. 创建 serviceBv2 虚拟节点。

   a. 使用以下内容创建名为 create-virtual-node-servicebv2.json 的文件：

```
{
```

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

b. 创建虚拟节点。

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

2. 创建 serviceA 虚拟节点。

a. 使用以下内容创建名为 create-virtual-node-servicea.json 的文件：

```
{
   "meshName" : "apps",
   "spec" : {
      "backends" : [
         {
            "virtualService" : {
               "virtualServiceName" : "serviceb.apps.local"
            }
         }
      ],
      "listeners" : [
         {
            "portMapping" : {
               "port" : 80,
               "protocol" : "http2"
```

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

b. 创建虚拟节点。

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

3. 更新在上一步中创建的 serviceb.apps.local 虚拟服务，以将其流量发送到 serviceB 虚拟路由器。最初创建虚拟服务时，它不会向任何位置发送流量，因为尚未创建 serviceB 虚拟路由器。

   a. 使用以下内容创建名为 update-virtual-service.json 的文件：

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

   b. 使用update-virtual-service命令更新虚拟服务。

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

4. 更新在上一步中创建的 serviceB 路径。

   a. 使用以下内容创建名为 update-route.json 的文件：

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

b. 使用 update-route 命令更新路由。

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

5. 创建 serviceA 虚拟服务。

a. 使用以下内容创建名为 create-virtual-servicea.json 的文件：

```
{
   "meshName" : "apps",
   "spec" : {
      "provider" : {
         "virtualNode" : {
            "virtualNodeName" : "serviceA"
         }
      }
   },
```

```
   "virtualServiceName" : "servicea.apps.local"
}
```

b. 创建虚拟服务。

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

网格摘要

在创建服务网格之前，您具有三个名为 servicea.apps.local、serviceb.apps.local 和 servicebv2.apps.local 的实际服务。除实际服务之外，现在您还具有一个服务网格，其中包含用以表示实际服务的以下资源：

- 两个虚拟服务。代理通过虚拟路由器将所有流量从 servicea.apps.local 虚拟服务发送到 serviceb.apps.local 虚拟服务。
- 三个名为 serviceA、serviceB 和 serviceBv2 的虚拟节点。Envoy 代理使用为虚拟节点配置的服务发现信息来查找实际服务的 IP 地址。
- 一个虚拟路由器，其路由指示 Envoy 代理将 75% 的入站流量路由到 serviceB 虚拟节点，将 25% 的流量路由到 serviceBv2 虚拟节点。

步骤 6：更新服务

创建网格后，您需要完成以下任务：

- 授予随每项服务部署的 Envoy 代理读取一个或多个虚拟节点配置的权限。有关如何向代理授权的更多信息，请参阅 Envoy Proxy 授权。
- 要更新现有服务，请完成以下步骤。

将 Amazon EC2 实例配置为虚拟节点成员

1. 创建一个 IAM 角色。

   a. 使用以下内容创建名为 ec2-trust-relationship.json 的文件。

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 使用以下命令创建 IAM 角色。

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

2. 将 IAM 策略附加到该角色，这将允许它仅从 Amazon ECR 中读取特定 App Mesh 虚拟节点的配置。

a. 使用以下内容创建名为 `virtual-node-policy.json` 的文件。`apps` 是您在 the section called “步骤 1：创建网格和虚拟服务” 中创建的网格的名称，`serviceB` 是您在 the section called “步骤 2：创建虚拟节点” 中创建的虚拟节点的名称。将 *111122223333* 替换为您的账户 ID，并将 *us-west-2* 替换为已在其中创建网格的区域。

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

b. 使用以下命令创建策略。

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

c. 将您在上一步中创建的策略附加到角色，以便角色仅从 App Mesh 中读取 `serviceB`虚拟节点的配置。

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

d. 将 `AmazonEC2ContainerRegistryReadOnly` 托管策略附加到角色，以便它可以从 Amazon ECR 中提取 Envoy 容器映像。

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

3. 启动带创建的 IAM 角色的 Amazon EC2 实例。
4. 通过 SSH 连接到实例。
5. 根据您的操作系统文档， AWS CLI 在您的实例上安装 Docker 和。
6. 对您希望 Docker 客户端从中拉取镜像的区域中的 Envoy 存储库进行身份验证：

- 除 `me-south-1`、`ap-east-1`、`ap-southeast-3`、`eu-south-1`、`il-central-1` 和 `af-south-1` 以外的所有区域。您可以用任何受支持区域替换*us-west-2*，但`me-south-1`、`ap-east-1`、`ap-southeast-3`、`eu-south-1`、`il-central-1`和 `af-south-1` 除外。

```
$aws ecr get-login-password \
    --region us-west-2 \
| docker login \
    --username AWS \
    --password-stdin 840364872350.dkr.ecr.us-west-2.amazonaws.com
```

- `me-south-1` 区域

```
$aws ecr get-login-password \
    --region me-south-1 \
| docker login \
    --username AWS \
    --password-stdin 772975370895.dkr.ecr.me-south-1.amazonaws.com
```

- `ap-east-1` 区域

```
$aws ecr get-login-password \
    --region ap-east-1 \
| docker login \
```

```
    --username AWS \
    --password-stdin 856666278305.dkr.ecr.ap-east-1.amazonaws.com
```

7. 运行下列命令之一，在您的实例上启动 Envoy 容器，具体取决于要从中拉取映像的区域。*apps* 和 *serviceB* 值是场景中定义的网格和虚拟节点名称。此信息告知代理要从 App Mesh 中读取的虚拟节点配置。要完成该场景，您还需要针对托管 serviceBv2 和 serviceA 虚拟节点所表示服务的 Amazon EC2 实例，完成这些步骤。对于您自己的应用程序，请将这些值替换为您自己的值。

   - 除 me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1 和 af-south-1 以外的所有区域。您可以将 *Region-code* 替换为除me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1 和 af-south-1 区域之外的任何受支持区域。您可以将 *1337* 替换为介于 0 和 2147483647 之间的任何值。

     ```
     sudo docker run --detach --env APPMESH_RESOURCE_ARN=mesh/apps/
     virtualNode/serviceB  \
     -u 1337 --network host 840364872350.dkr.ecr.region-code.amazonaws.com/aws-
     appmesh-envoy:v1.27.3.0-prod
     ```

   - me-south-1 区域。您可以将 *1337* 替换为介于 0 和 2147483647 之间的任何值。

     ```
     sudo docker run --detach --env APPMESH_RESOURCE_ARN=mesh/apps/
     virtualNode/serviceB  \
     -u 1337 --network host 772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-
     envoy:v1.27.3.0-prod
     ```

   - ap-east-1 区域。您可以将 *1337* 替换为介于 0 和 2147483647 之间的任何值。

     ```
     sudo docker run --detach --env APPMESH_RESOURCE_ARN=mesh/apps/
     virtualNode/serviceB  \
     -u 1337 --network host 856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-
     envoy:v1.27.3.0-prod
     ```

   > **Note**
   >
   > 该 APPMESH_RESOURCE_ARN 属性需要版本 1.15.0 或更高版本的 Envoy 映像。有关更多信息，请参阅Envoy。

> **Important**
>
> 仅支持将版本 v1.9.0.0-prod 或更高版本与 App Mesh 一起使用。

8. 在下面选择 Show more。使用以下内容在实例上创建名为 envoy-networking.sh 的文件。将 *8000* 替换为应用程序代码用于传入流量的端口。您可以更改 APPMESH_IGNORE_UID 的值，但该值必须与您在上一步中指定的值相同；例如 1337。如有必要，您可以向 APPMESH_EGRESS_IGNORED_IP 添加其他地址。请不要修改任何其他行。

```
#!/bin/bash -e

#
# Start of configurable options
#


#APPMESH_START_ENABLED="0"
APPMESH_IGNORE_UID="1337"
APPMESH_APP_PORTS="8000"
APPMESH_ENVOY_EGRESS_PORT="15001"
APPMESH_ENVOY_INGRESS_PORT="15000"
APPMESH_EGRESS_IGNORED_IP="169.254.169.254,169.254.170.2"

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

# Enable IPv6.
[ -z "$APPMESH_ENABLE_IPV6" ] && APPMESH_ENABLE_IPV6="0"

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

# Port numbers Application and Envoy are listening on.
if [ -z "$APPMESH_ENVOY_EGRESS_PORT" ]; then
```

```
    echo "APPMESH_ENVOY_EGRESS_PORT must be defined to forward traffic from the
 application to the proxy."
    exit 1
fi

# If an app port was specified, then we also need to enforce the proxies ingress
 port so we know where to forward traffic.
if [ ! -z "$APPMESH_APP_PORTS" ] && [ -z "$APPMESH_ENVOY_INGRESS_PORT" ]; then
    echo "APPMESH_ENVOY_INGRESS_PORT must be defined to forward traffic from the
 APPMESH_APP_PORTS to the proxy."
    exit 1
fi

# Comma separated list of ports for which egress traffic will be ignored, we always
 refuse to route SSH traffic.
if [ -z "$APPMESH_EGRESS_IGNORED_PORTS" ]; then
    APPMESH_EGRESS_IGNORED_PORTS="22"
else
    APPMESH_EGRESS_IGNORED_PORTS="$APPMESH_EGRESS_IGNORED_PORTS,22"
fi

#
# End of configurable options
#

function initialize() {
    echo "=== Initializing ==="
    if [ ! -z "$APPMESH_APP_PORTS" ]; then
        iptables -t nat -N APPMESH_INGRESS
        if [ "$APPMESH_ENABLE_IPV6" == "1" ]; then
            ip6tables -t nat -N APPMESH_INGRESS
        fi
    fi
    iptables -t nat -N APPMESH_EGRESS
    if [ "$APPMESH_ENABLE_IPV6" == "1" ]; then
        ip6tables -t nat -N APPMESH_EGRESS
    fi
}

function enable_egress_routing() {
    # Stuff to ignore
    [ ! -z "$APPMESH_IGNORE_UID" ] && \
        iptables -t nat -A APPMESH_EGRESS \
        -m owner --uid-owner $APPMESH_IGNORE_UID \
```

```
        -j RETURN

   [ ! -z "$APPMESH_IGNORE_GID" ] && \
       iptables -t nat -A APPMESH_EGRESS \
       -m owner --gid-owner $APPMESH_IGNORE_GID \
       -j RETURN

   [ ! -z "$APPMESH_EGRESS_IGNORED_PORTS" ] && \
       for IGNORED_PORT in $(echo "$APPMESH_EGRESS_IGNORED_PORTS" | tr "," "\n");
do
         iptables -t nat -A APPMESH_EGRESS \
         -p tcp \
         -m multiport --dports "$IGNORED_PORT" \
         -j RETURN
       done

   if [ "$APPMESH_ENABLE_IPV6" == "1" ]; then
     # Stuff to ignore ipv6
     [ ! -z "$APPMESH_IGNORE_UID" ] && \
         ip6tables -t nat -A APPMESH_EGRESS \
         -m owner --uid-owner $APPMESH_IGNORE_UID \
         -j RETURN

     [ ! -z "$APPMESH_IGNORE_GID" ] && \
         ip6tables -t nat -A APPMESH_EGRESS \
         -m owner --gid-owner $APPMESH_IGNORE_GID \
         -j RETURN

     [ ! -z "$APPMESH_EGRESS_IGNORED_PORTS" ] && \
       for IGNORED_PORT in $(echo "$APPMESH_EGRESS_IGNORED_PORTS" | tr "," "\n");
do
         ip6tables -t nat -A APPMESH_EGRESS \
         -p tcp \
         -m multiport --dports "$IGNORED_PORT" \
         -j RETURN
       done
   fi

   # The list can contain both IPv4 and IPv6 addresses. We will loop over this
list
   # to add every IPv4 address into `iptables` and every IPv6 address into
`ip6tables`.
   [ ! -z "$APPMESH_EGRESS_IGNORED_IP" ] && \
       for IP_ADDR in $(echo "$APPMESH_EGRESS_IGNORED_IP" | tr "," "\n"); do
```

```
            if [[ $IP_ADDR =~ .*:.* ]]
            then
                [ "$APPMESH_ENABLE_IPV6" == "1" ] && \
                    ip6tables -t nat -A APPMESH_EGRESS \
                        -p tcp \
                        -d "$IP_ADDR" \
                        -j RETURN
            else
                iptables -t nat -A APPMESH_EGRESS \
                    -p tcp \
                    -d "$IP_ADDR" \
                    -j RETURN
            fi
        done

    # Redirect everything that is not ignored
    iptables -t nat -A APPMESH_EGRESS \
        -p tcp \
        -j REDIRECT --to $APPMESH_ENVOY_EGRESS_PORT

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

    if [ "$APPMESH_ENABLE_IPV6" == "1" ]; then
        # Redirect everything that is not ignored ipv6
        ip6tables -t nat -A APPMESH_EGRESS \
            -p tcp \
            -j REDIRECT --to $APPMESH_ENVOY_EGRESS_PORT
        # Apply APPMESH_EGRESS chain to non local traffic ipv6
        ip6tables -t nat -A OUTPUT \
            -p tcp \
            -m addrtype ! --dst-type LOCAL \
            -j APPMESH_EGRESS
    fi

}

function enable_ingress_redirect_routing() {
    # Route everything arriving at the application port to Envoy
    iptables -t nat -A APPMESH_INGRESS \
        -p tcp \
```

```
        -m multiport --dports "$APPMESH_APP_PORTS" \
        -j REDIRECT --to-port "$APPMESH_ENVOY_INGRESS_PORT"

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

    if [ "$APPMESH_ENABLE_IPV6" == "1" ]; then
        # Route everything arriving at the application port to Envoy ipv6
        ip6tables -t nat -A APPMESH_INGRESS \
            -p tcp \
            -m multiport --dports "$APPMESH_APP_PORTS" \
            -j REDIRECT --to-port "$APPMESH_ENVOY_INGRESS_PORT"

        # Apply AppMesh ingress chain to everything non-local ipv6
        ip6tables -t nat -A PREROUTING \
            -p tcp \
            -m addrtype ! --src-type LOCAL \
            -j APPMESH_INGRESS
    fi
}

function enable_routing() {
    echo "=== Enabling routing ==="
    enable_egress_routing
    if [ ! -z "$APPMESH_APP_PORTS" ]; then
        enable_ingress_redirect_routing
    fi
}

function disable_routing() {
    echo "=== Disabling routing ==="
    iptables -t nat -F APPMESH_INGRESS
    iptables -t nat -F APPMESH_EGRESS

    if [ "$APPMESH_ENABLE_IPV6" == "1" ]; then
        ip6tables -t nat -F APPMESH_INGRESS
        ip6tables -t nat -F APPMESH_EGRESS
    fi
}

function dump_status() {
```

```
    echo "=== iptables FORWARD table ==="
    iptables -L -v -n
    echo "=== iptables NAT table ==="
    iptables -t nat -L -v -n

    if [ "$APPMESH_ENABLE_IPV6" == "1" ]; then
        echo "=== ip6tables FORWARD table ==="
        ip6tables -L -v -n
        echo "=== ip6tables NAT table ==="
        ip6tables -t nat -L -v -n
    fi
}

function clean_up() {
    disable_routing
    ruleNum=$(iptables -L PREROUTING -t nat --line-numbers | grep APPMESH_INGRESS |
 cut -d " " -f 1)
    iptables -t nat -D PREROUTING $ruleNum

    ruleNum=$(iptables -L OUTPUT -t nat --line-numbers | grep APPMESH_EGRESS | cut
 -d " " -f 1)
    iptables -t nat -D OUTPUT $ruleNum

    iptables -t nat -X APPMESH_INGRESS
    iptables -t nat -X APPMESH_EGRESS

    if [ "$APPMESH_ENABLE_IPV6" == "1" ]; then
        ruleNum=$(ip6tables -L PREROUTING -t nat --line-numbers | grep
 APPMESH_INGRESS | cut -d " " -f 1)
        ip6tables -t nat -D PREROUTING $ruleNum

        ruleNum=$(ip6tables -L OUTPUT -t nat --line-numbers | grep APPMESH_EGRESS |
 cut -d " " -f 1)
        ip6tables -t nat -D OUTPUT $ruleNum

        ip6tables -t nat -X APPMESH_INGRESS
        ip6tables -t nat -X APPMESH_EGRESS
    fi
}

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

```
            "quit")
                clean_up
                break
                ;;
            "status")
                dump_status
                ;;
            "enable")
                enable_routing
                ;;
            "disable")
                disable_routing
                ;;
            *)
                echo "Available commands: quit, status, enable, disable"
                ;;
        esac
    done
}

function print_config() {
    echo "=== Input configuration ==="
    env | grep APPMESH_ || true
}

print_config

initialize

if [ "$APPMESH_START_ENABLED" == "1" ]; then
    enable_routing
fi

main_loop
```

9. 要配置 iptables 规则以将应用程序流量路由到 Envoy 代理，请运行您在上一步中创建的脚本。

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

10. 启动虚拟节点应用程序代码。

> **Note**
>
> 有关 App Mesh 的更多示例和演练，请参阅 App Mesh 示例存储库。

App Mesh 路线图

这是 AWS App Mesh 的实验性公开路线图。该路线图允许客户了解我们即将推出的产品和优先事项，从而帮助客户规划将来如何使用 App Mesh。该存储库包含有关团队工作的信息，并允许所有 AWS 客户提供直接反馈。

App Mesh 路线图

App Mesh 示例

您可以在以下存储库中找到操作 AWS App Mesh 中显示的端到端演练，以及用于与各种 AWS 服务集成的代码示例：

App Mesh 示例

App Mesh 概念

App Mesh 由以下概念组成。

- 服务网格
- 虚拟服务
- 虚拟网关
- 虚拟节点
- 虚拟路由器

服务网格

服务网格是一种用于驻留在其内的服务之间的网络流量的逻辑边界。在创建服务网格后，您可以创建虚拟服务、虚拟节点、虚拟路由器以及用于在网格中的应用程序之间分配流量的路由。

创建服务网格

> **Note**
>
> 创建网格时，必须添加命名空间选择器。如果命名空间选择器为空，则它会选择所有命名空间。要限制命名空间，请使用标签将 App Mesh 资源与创建的网格相关联。

AWS Management Console

使用 AWS Management Console 创建服务网格

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择创建网格。
3. 对于网格名称，为您的服务网格指定一个名称。
4. （可选）选择允许外部流量。默认情况下，网格中的代理仅在彼此之间转发流量。如果您允许外部流量，则网格中的代理还会将 TCP 流量直接转发到未使用网格中定义的代理部署的服务。

> **Note**
>
> 如果您在使用 ALLOW_ALL 时在虚拟节点上指定了任何后端，则必须将该虚拟节点的所有出口指定为后端。否则，ALLOW_ALL 将不再适用于该虚拟节点。

5. IP 版本首选项

通过切换覆盖默认 IP 版本行为，控制网格内的流量应使用哪个 IP 版本。默认情况下，App Mesh 使用各种 IP 版本。

> **Note**
>
> 网格将 IP 首选项应用于网格中的所有虚拟节点和虚拟网关。通过在创建或编辑节点时设置 IP 首选项，可以在单个虚拟节点上覆盖该行为。无法在虚拟网关上覆盖 IP 首选项，这是因为无论在网格上设置哪个首选项，允许它们侦听 IPv4 和 IPv6 流量的虚拟网关的配置都是相同的。

- 默认值
  - Envoy 的 DNS 解析器优先使用 IPv6 并在必要时回退到 IPv4。
  - 我们会优先使用 AWS Cloud Map 返回的 IPv4 地址（如果有），并在必要时回退到使用 IPv6 地址。
  - 为本地应用程序创建的端点使用 IPv4 地址。
  - Envoy 侦听器绑定到所有 IPv4 地址。
- 首选 IPv6
  - Envoy 的 DNS 解析器优先使用 IPv6 并在必要时回退到 IPv4。
  - AWS Cloud Map 返回的 IPv6 地址如果可用，则使用该地址，并在必要时回退到使用 IPv4 地址
  - 为本地应用程序创建的端点使用 IPv6 地址。
  - Envoy 侦听器绑定到所有 IPv4 和 IPv6 地址。
- 首选 IPv4
  - Envoy 的 DNS 解析器优先使用 IPv4 并在必要时回退到 IPv6。
  - 我们会优先使用 AWS Cloud Map 返回的 IPv4 地址（如果有），并在必要时回退到使用 IPv6 地址。

- 为本地应用程序创建的端点使用 IPv4 地址。
- Envoy 侦听器绑定到所有 IPv4 和 IPv6 地址。

- 仅 IPv6
  - Envoy 的 DNS 解析器仅使用 IPv6。
  - 仅使用 AWS Cloud Map 返回的 IPv6 地址。如果 AWS Cloud Map 返回 IPv4 地址，则不使用 IP 地址，并将空结果返回给 Envoy。
  - 为本地应用程序创建的端点使用 IPv6 地址。
  - Envoy 侦听器绑定到所有 IPv4 和 IPv6 地址。
- 仅 IPv4
  - Envoy 的 DNS 解析器仅使用 IPv4。
  - 仅使用 AWS Cloud Map 返回的 IPv4 地址。如果 AWS Cloud Map 返回 IPv6 地址，则不使用 IP 地址，并将空结果返回给 Envoy。
  - 为本地应用程序创建的端点使用 IPv4 地址。
  - Envoy 侦听器绑定到所有 IPv4 和 IPv6 地址。

6. 选择创建网格以完成。
7. （可选）与其他账户共享网格。共享网格允许不同账户创建的资源在同一个网格中相互通信。有关更多信息，请参阅 使用共享网格。。

AWS CLI

使用 AWS CLI 创建网格。

使用以下命令创建服务网格（将*##*值替换为自己的值）：

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

2. 输出示例：

```
{
    "mesh":{
        "meshName":"meshName",
        "metadata":{
            "arn":"arn:aws:appmesh:us-west-2:123456789012:mesh/meshName",
            "createdAt":"2022-04-06T08:45:50.072000-05:00",
            "lastUpdatedAt":"2022-04-06T08:45:50.072000-05:00",
            "meshOwner": "123456789012",
```

```
            "resourceOwner": "123456789012",
            "uid":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version":1
        },
        "spec":{},
        "status":{
            "status":"ACTIVE"
        }
    }
}
```

有关使用 AWS CLI 为 App Mesh 创建网格的更多信息，请参阅AWS CLI 参考资料中的 create-mesh 命令。

删除网格

AWS Management Console

要使用 AWS Management Console 删除虚拟网关

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要删除的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在确认框中，键入 **delete**，然后单击删除。

AWS CLI

使用 AWS CLI 删除网格

1. 使用以下命令删除网格（将*##*值替换为自己的值）：

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

2. 输出示例：

```
{
    "mesh": {
        "meshName": "meshName",
        "metadata": {
```

```
            "arn":"arn:aws:appmesh:us-west-2:123456789012:mesh/meshName",
            "createdAt": "2022-04-06T08:45:50.072000-05:00",
            "lastUpdatedAt": "2022-04-07T11:06:32.795000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "123456789012",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 1
        },
        "spec": {},
        "status": {
            "status": "DELETED"
        }
    }
}
```

有关使用 AWS CLI 为 App Mesh 删除网格的更多信息，请参阅 AWS CLI 参考资料中的删除网格命令。

虚拟服务

虚拟服务是一种抽象的实际服务，由虚拟节点直接提供或通过虚拟路由器的方式间接提供。相关服务通过其 `virtualServiceName` 调用您的虚拟服务，然后这些请求将路由至指定为虚拟服务的提供商的虚拟节点或虚拟路由器。

创建虚拟服务。

AWS Management Console

使用 AWS Management Console 创建虚拟服务

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要在其中创建虚拟服务的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中，选择虚拟服务。
4. 选择创建虚拟服务。
5. 对于虚拟服务名称，为您的虚拟服务选择一个名称。您可以选择任何名称，但建议使用您要定位的真实服务的服务发现名称，如 `my-service.default.svc.cluster.local`，这样可以更轻松地将您的虚拟服务与真实服务关联起来。通过这种方式，您无需更改代码以引用与代码当前引用名称不同的名称。您指定的名称必须解析为非环回 IP 地址，因为在将请求发送到

Envoy 代理之前，应用程序容器必须能够成功解析该名称。您可以使用任何非环回 IP 地址，因为应用程序或代理容器都不会与此 IP 地址通信。代理通过您在 App Mesh 中为其配置的名称与其他虚拟服务进行通信，而不是通过名称解析到的 IP 地址进行通信。

6. 对于提供商，选择您的虚拟服务的提供商类型：

   - 如果您希望虚拟服务跨多个虚拟节点分布流量，则选择虚拟路由器，然后从下拉菜单中选择要使用的虚拟路由器。
   - 如果您希望虚拟服务直接到达虚拟节点，而不使用虚拟路由器，则选择虚拟节点，然后从下拉菜单中选择要使用的虚拟节点。

     > **Note**
     >
     > App Mesh 可能会自动为您在 2020 年 7 月 29 日当天或之后定义的每个虚拟节点提供者创建默认 Envoy 路由重试策略，即使您无法通过 App Mesh API 定义这样的策略。有关更多信息，请参阅 默认路由重试策略。

   - 如果您此时不希望虚拟服务路由流量（例如，如果您的虚拟节点或虚拟路由器不存在），则选择无。您可以稍后更新此虚拟服务的提供商。

7. 选择创建虚拟服务以完成。

AWS CLI

使用 AWS CLI 创建虚拟服务。

使用以下命令和输入 JSON 文件（将##值替换为自己的值），使用虚拟节点提供商创建虚拟服务：

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

2. create-virtual-service-virtual-node.json 示例的内容：

```
{
    "meshName": "meshName",
    "spec": {
        "provider": {
            "virtualNode": {
                "virtualNodeName": "nodeName"
            }
        }
```

```
    },
    "virtualServiceName": "serviceA.svc.cluster.local"
}
```

3. 输出示例：

```
{
    "virtualService": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualService/serviceA.svc.cluster.local",
            "createdAt": "2022-04-06T09:45:35.890000-05:00",
            "lastUpdatedAt": "2022-04-06T09:45:35.890000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 1
        },
        "spec": {
            "provider": {
                "virtualNode": {
                    "virtualNodeName": "nodeName"
                }
            }
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualServiceName": "serviceA.svc.cluster.local"
    }
}
```

有关使用适用于 App Mesh 的 AWS CLI 创建虚拟服务的更多信息，请参阅 AWS CLI 参考资料中的 create-virtual-service 命令。

删除虚拟服务

> **Note**
>
> 您无法删除网关路由引用的虚拟服务。您需要先删除网关路由。

AWS Management Console

使用 AWS Management Console 删除虚拟服务

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要从中删除虚拟服务的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中，选择虚拟服务。
4. 选择要删除的虚拟服务，然后单击右上角的删除。您只能删除您的账户被列为资源所有者的虚拟网关。
5. 在确认框中，键入 **delete**，然后单击删除。

AWS CLI

使用 AWS CLI 删除虚拟服务

1. 使用以下命令删除您的虚拟服务（用您自己的值替换*##*值）：

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

2. 输出示例：

```
{
    "virtualService": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualService/serviceA.svc.cluster.local",
            "createdAt": "2022-04-06T09:45:35.890000-05:00",
            "lastUpdatedAt": "2022-04-07T10:39:42.772000-05:00",
            "meshOwner": "123456789012",
```

```
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 2
        },
        "spec": {
            "provider": {
                "virtualNode": {
                    "virtualNodeName": "nodeName"
                }
            }
        },
        "status": {
            "status": "DELETED"
        },
        "virtualServiceName": "serviceA.svc.cluster.local"
    }
}
```


有关使用适用于 App Mesh 的 AWS CLI 删除虚拟服务的更多信息，请参阅 AWS CLI 参考资料中的 delete-virtual-service 命令。

虚拟网关

虚拟网关允许网格外的资源与网格内的资源进行通信。虚拟网关代表在 Amazon ECS 服务、Kubernetes 服务或 Amazon EC2 实例中运行的某个 Envoy 代理。与虚拟节点（代表使用应用程序运行的 Envoy）不同，虚拟网关代表自己部署的 Envoy。

外部资源必须能够将 DNS 名称解析为分配给运行 Envoy 的服务或实例的 IP 地址。然后，Envoy 可以访问网格内部资源的所有 App Mesh 配置。在虚拟网关处理传入请求的配置是使用网关路由指定的。

> **Important**
>
> 带有 HTTP 或 HTTP2 侦听器的虚拟网关将传入请求的主机名重写为网关路由目标虚拟服务的名称，默认情况下，网关路由中的匹配前缀将重写为 `/`。例如，如果您已将网关路由匹配前缀配置为 `/chapter`，如果传入请求是 `/chapter/1`，则请求将被重写为`/1`。要配置重写，请参阅网关路由中的创建网关路由部分。
>
> 创建虚拟网关时，不应配置`proxyConfiguration` 和 `user`。

要完成端到端演练，请参阅配置入站网关。

创建虚拟网关

> **Note**
>
> 创建虚拟网关时，必须添加带有标签的命名空间选择器，以标识将网关路由与已创建的虚拟网关关联的命名空间列表。

AWS Management Console

使用 AWS Management Console 创建虚拟网关

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要在其中创建虚拟网关。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中选择虚拟网关。
4. 选择创建虚拟网关。
5. 对于虚拟网关名称，请输入虚拟网关的名称。
6. （可选，但建议使用）配置客户端策略默认值。
   a. （可选）如果您希望网关仅使用传输层安全性协议（TLS）与虚拟服务通信，请选择强制 TLS。
   b. （可选）在端口中，指定一个或多个要与虚拟服务执行 TLS 通信的端口。
   c. 对于验证方法，选择下列选项之一。您指定的证书必须已存在且符合特定要求。有关更多信息，请参阅 证书要求。
      - AWS Private Certificate Authority托管 - 选择一个或多个现有证书。
      - Envoy 密钥发现服务 (SDS) 托管 – 输入 Envoy 使用密钥发现服务获取的密钥名称。
      - 本地文件托管 - 指定 Envoy 部署所在文件系统上的证书链文件的路径。
   d. （可选）输入主题备用名称。要添加其他 SAN，请选择添加 SAN。SAN 必须采用 FQDN 或 URI 格式。
   e. （可选）选择提供客户端证书和以下选项之一，以便在服务器请求客户端证书时提供该证书并启用双向 TLS 身份验证。要了解有关双向 TLS 的更多信息，请参阅 App Mesh 双向 TLS 身份验证文档。
      - Envoy 密钥发现服务 (SDS) 托管 – 输入 Envoy 使用密钥发现服务获取的密钥名称。

- 本地文件托管 - 指定 Envoy 部署所在文件系统上的证书链文件和私钥的路径。有关使用本地文件加密的示例应用程序部署网格的完整端到端演练，请参阅 GitHub 上使用文件提供的 TLS 证书配置 TLS。

7. （可选）要配置日志记录，请选择日志记录。输入您希望 Envoy 要使用的 HTTP 访问日志路径。建议您使用 `/dev/stdout` 路径，以便您可以使用 Docker 日志驱动程序来将 Envoy 日志导出至某个服务，如 Amazon CloudWatch Logs。

> **Note**
>
> 日志必须仍由您应用程序中的代理进行接收并发送至目标。此文件路径仅指示 Envoy 要发送日志的位置。

8. 配置侦听器。

   a. 选择协议并指定 Envoy 侦听流量的端口。http 侦听器允许连接转换到 websockets。您可以单击添加侦听器来添加多个侦听器。移除按钮将移除该侦听器。

   b. （可选）启用连接池

   连接池限制虚拟网关 Envoy 可以同时建立的连接数。它旨在保护您的 Envoy 实例免受连接不堪重负，并允许您根据应用程序的需求调整流量整形。

   您可以为虚拟网关侦听器配置目标端连接池设置。App Mesh 默认将客户端连接池设置为无限，从而简化了网格配置。

   > **Note**
   >
   `connectionPool` 和 `connectionPool` PortMapping 协议必须相同。如果您的侦听器协议为 `grpc` 或 `http2`，则仅指定 `maxRequests`。如果您的侦听器协议为 `http`，则可以同时指定 `maxConnections` 和 `maxPendingRequests`。

   - 对于最大连接数，请指定最大出站连接数。
   - 对于最大请求数，指定可与虚拟网关 Envoy 建立的最大并行请求数。
   - （可选）在最大待处理请求数中，指定在达到最大连接数之后，Envoy 队列中排列的溢出请求数。默认值为 2147483647。

   c. （可选）如果要为侦听器配置运行状况检查，请选择启用运行状况检查。

运行状况检查策略是可选的，但是如果您为运行状况策略指定任何值，则必须为正常阈值、运行状况检查间隔、运行状况检查协议、超时时间和运行状况不佳阈值指定值。

- 对于 运行状况检查协议，请选择一个协议。如果选择 grpc，您的服务必须符合 GRPC 运行状况检查协议。
- 对于运行状况检查端口，指定应对其运行状况检查的端口。
- 对于正常阈值，指定在声明侦听器运行状况良好之前，必须出现的连续成功的运行状况检查次数。
- 对于运行状况检查间隔，指定执行每次运行状况检查间隔的时间（毫秒）。
- 对于路径，指定运行状况检查请求的目标路径。仅当运行状况检查协议为 `http` 或 `http2` 时才使用此值。其他协议将忽略此值。
- 对于超时周期，指定接收来自运行状况检查的响应时要等待的时间（毫秒）。
- 对于不正常阈值，指定在声明侦听器运行状况不正常之前，必须出现的连续失败的运行状况检查次数。

d. （可选）如果要指定客户端是否使用 TLS 与此虚拟网关通信，请选择启用 TLS 终止。

- 对于模式，选择要在侦听器上配置 TLS 的模式。
- 对于 Certificate method，执行下列操作之一：证书必须满足具体要求。有关更多信息，请参阅 证书要求。
  - AWS Certificate Manager 托管 — 选择现有证书。
  - Envoy 密钥发现服务 (SDS) 托管 – 输入 Envoy 使用密钥发现服务获取的密钥名称。
  - 本地文件托管 - 指定 Envoy 部署所在文件系统上的证书链和私钥文件的路径。
- （可选）如果客户端提供证书，请选择需要客户端证书和以下选项之一，以启用双向 TLS 身份验证。要了解有关双向 TLS 的更多信息，请参阅 App Mesh 双向 TLS 身份验证文档。
  - Envoy 密钥发现服务 (SDS) 托管 – 输入 Envoy 使用密钥发现服务获取的密钥名称。
  - 本地文件托管 - 指定 Envoy 部署所在文件系统上的证书链文件的路径。
- （可选）输入主题备用名称。要添加其他 SAN，请选择添加 SAN。SAN 必须采用 FQDN 或 URI 格式。

9. 选择创建虚拟网关以完成。

AWS CLI

使用 AWS CLI 创建虚拟网关。

使用以下命令创建虚拟网关并输入 JSON（用您自己的值替换*##*值）：

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

2. create-virtual-gateway.json 示例的内容：

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

3. 输出示例：

```
{
    "virtualGateway": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:123456789012:mesh/meshName/
virtualGateway/virtualGatewayName",
            "createdAt": "2022-04-06T10:42:42.015000-05:00",
            "lastUpdatedAt": "2022-04-06T10:42:42.015000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "123456789012",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 1
        },
        "spec": {
            "listeners": [
```

```
                {
                    "portMapping": {
                        "port": 9080,
                        "protocol": "http"
                    }
                }
            ]
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualGatewayName": "virtualGatewayName"
    }
}
```

有关使用AWS CLI 为 App Mesh 创建虚拟网关的更多信息，请参阅 AWS CLI 参考资料中的create-virtual-gateway 命令。

部署虚拟网关

部署仅包含 Envoy 容器的亚马逊 ECS 或 Kubernetes 服务。您也可以在亚马逊 EC2 实例上部署 Envoy 容器。有关更多信息，请参阅《App Mesh 和 Amazon EC2 入门》。有关如何在 Amazon ECS 上部署的更多信息，请参阅《App Mesh 和 Amazon ECS入门》或《部署到 Kubernetes 的 AWS App Mesh 和 Kubernetes 入门》。您需要将 APPMESH_RESOURCE_ARN 环境变量设置为 mesh/*mesh-name*/virtualGateway/*virtual-gateway-name*，并且不得指定代理配置，这样代理的流量就不会被重定向到自身。默认情况下，当 Envoy 在指标和跟踪中引用自身时，App Mesh 使用您在 APPMESH_RESOURCE_ARN 中指定的资源的名称。您可以通过使用自己的名称设置 APPMESH_RESOURCE_CLUSTER 环境变量来覆盖此行为。

我们建议您部署容器的多个实例，并设置网络负载均衡器来对这些实例的流量进行负载均衡。负载均衡器的服务发现名称是您希望外部服务用来访问网格中的资源（例如 *myapp* .example.com）的名称。有关更多信息，请参阅创建网络负载均衡器 (Amazon ECS)、创建外部负载均衡器 (Kubernetes) 或教程：提高应用程序在 Amazon EC2 上的可用性。此外，您可以在我们的 App Mesh 示例中找到更多示例和演练。

启用代理授权。有关更多信息，请参阅 Envoy Proxy 授权。

删除虚拟网关

AWS Management Console

要使用 AWS Management Console 删除虚拟网关

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要从中删除虚拟网关的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中选择虚拟网关。
4. 选择要删除的虚拟网关，然后选择删除。如果虚拟网关有任何关联的网关路由，则无法将其删除。必须先删除所有关联的网关路由。您只能删除您的账户被列为资源所有者的虚拟网关。
5. 在确认框中，键入 **delete**，然后选择删除。

AWS CLI

要使用 AWS CLI 删除虚拟网关

1. 使用以下命令删除您的虚拟网关（用您自己的值替换*##*值）：

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

2. 输出示例：

```
{
    "virtualGateway": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:123456789012:mesh/meshName/
virtualGateway/virtualGatewayName",
            "createdAt": "2022-04-06T10:42:42.015000-05:00",
            "lastUpdatedAt": "2022-04-07T10:57:22.638000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "123456789012",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 2
        },
        "spec": {
            "listeners": [
```

```
                {
                    "portMapping": {
                        "port": 9080,
                        "protocol": "http"
                    }
                }
            ]
        },
        "status": {
            "status": "DELETED"
        },
        "virtualGatewayName": "virtualGatewayName"
    }
}
```

有关使用 App Mesh 删除虚拟网关的更多信息，请参阅 AWS CLI 参考资料中的 delete-virtual-gateway 命令。AWS CLI

网关路由

网关路由附加到虚拟网关，并将流量路由到现有虚拟服务。如果路由与请求匹配，它可以将流量分配到目标虚拟服务。本主题可帮助您在服务网格中使用网关路由。

创建网关路由

AWS Management Console

使用 AWS Management Console 创建网关路由

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要在其中创建网关路由的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中选择虚拟网关。
4. 选择要与新网关路由关联的虚拟网关。如果没有列出任何网关，则需要先创建一个虚拟网关。您只能为您的账户列为资源所有者的虚拟网关创建网关路由。
5. 在网关路由表中，选择创建网关路由。
6. 对于网关路由名称，指定要用于您网关路由的名称。
7. 对于网关路由类型，请选择 http、http2 或 grpc。
8. 选择现有的虚拟服务名称。如果没有列出任何虚拟服务，则需要先创建一个虚拟服务列表。

9. 为虚拟服务提供商端口选择与目标相对应的端口。当所选虚拟服务的提供商（路由器或节点）有多个侦听器时，需要使用虚拟服务提供商端口。
10. （可选）在优先级中，指定此网关路由的优先级。
11. 对于匹配配置，请指定：

- 如果所选类型为 http/http2：
  - （可选）方法 - 指定要在传入的 http/http2 请求中匹配的方法标头。
  - （可选）端口匹配 - 匹配传入流量的端口。如果此虚拟网关有多个侦听器，则需要匹配端口。
  - （可选）精确/后缀主机名 - 指定在路由到目标虚拟服务的传入请求上应匹配的主机名。
  - （可选）前缀/精确/正则表达式路径 - 匹配网址路径的方法。
    - 前缀匹配 - 默认情况下，网关路由的匹配请求将重写为目标虚拟服务的名称，匹配的前缀将重写为 `/`。根据您配置虚拟服务的方式，它可能会使用虚拟路由器根据特定的前缀或标头将请求路由到不同的虚拟节点。

> **Important**
>
> - 您不能为前缀匹配指定 `/aws-appmesh*` 或 `/aws-app-mesh*`。这些前缀保留供将来的 App Mesh 内部使用。
> - 如果定义了多条网关路由，则会将请求与前缀最长的路由相匹配。例如，如果存在两条网关路由，其中一条的前缀为 `/chapter`，一条的前缀为 `/`，则请求 `www.example.com/chapter/` 将带 `/chapter` 前缀的网关路由进行匹配。

> **Note**
>
> 如果您启用基于路径/前缀的匹配，App Mesh 会启用路径标准化（normalize_path 和 merge_slashes），以最大限度地减少出现路径混淆漏洞的可能性。
> 当参与请求的各方使用不同的路径表示形式时，就会出现路径混淆漏洞。

    - 精确匹配 - exact 参数禁用路径的部分匹配，并确保只有当路径与当前 URL 完全匹配时，它才会返回路由。
    - 正则表达式匹配 - 用于描述多个网址实际上可以识别网站上的单个页面的模式。

- （可选）查询参数 - 此字段允许您匹配查询参数。
- （可选）标头 - 指定 http 和 http2 的标头。它应该与传入的请求相匹配，以路由到目标虚拟服务。

- 如果选择了 grpc 类型：
  - 主机名匹配类型和（可选）精确/后缀匹配 - 指定在路由到目标虚拟服务的传入请求上应匹配的主机名。
  - grpc 服务名称 - grpc 服务充当您的应用程序的 API，是用 ProtoBuf 定义的。

    > **Important**
    >
    > 您不能为服务名称指定 `/aws.app-mesh*` 或 `aws.appmesh`。这些服务名称保留供将来的 App Mesh 内部使用。

  - （可选）元数据 - 指定 grpc 的元数据。它应该与传入的请求相匹配，以路由到目标虚拟服务。

12. （可选）对于重写配置：

- 如果所选类型为 http/http2：
  - 如果 前缀 是选定的匹配类型：
    - 覆盖主机名的自动重写 - 默认情况下，主机名将重写为目标虚拟服务的名称。
    - 覆盖前缀的自动重写 - 开启后， Prefix rewrite 会指定重写前缀的值。
  - 如果精确路径是选定的匹配类型：
    - 覆盖主机名的自动重写 - 默认情况下，主机名将重写为目标虚拟服务的名称。
    - 路径重写 - 指定重写路径的值。没有默认路径。
  - 如果正则表达式路径是选定的匹配类型：
    - 覆盖主机名的自动重写 - 默认情况下，主机名将重写为目标虚拟服务的名称。
    - 路径重写 - 指定重写路径的值。没有默认路径。
- 如果选择了 grpc 类型：
  - 覆盖主机名的自动重写 - 默认情况下，主机名将重写为目标虚拟服务的名称。

13. 选择创建网关路由以完成。

AWS CLI

使用 AWS CLI 创建网关路由。

使用以下命令创建网关路由，然后输入 JSON（用您自己的值替换##值）：

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

2. create-gateway-route.json 示例的内容：

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

3. 输出示例：

```
{
    "gatewayRoute": {
        "gatewayRouteName": "gatewayRouteName",
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualGateway/virtualGatewayName/gatewayRoute/gatewayRouteName",
            "createdAt": "2022-04-06T11:05:32.100000-05:00",
            "lastUpdatedAt": "2022-04-06T11:05:32.100000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 1
        },
```

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

有关使用 App Mesh AWS CLI 创建网关路由的更多信息，请参阅 AWS CLI 参考资料中的 create-gateway-route 命令。

删除网关路由

AWS Management Console

使用 AWS Management Console 删除网关路由

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要从中删除网关路由的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中选择虚拟网关。
4. 选择要从中删除网关路由的虚拟网关。
5. 在网关路由表中，选择要删除的网关路由，然后选择删除。只有当您的账户被列为资源所有者时，您才能删除网关路由。
6. 在确认框中，键入 **delete**，然后单击删除。

AWS CLI

使用 AWS CLI 删除网关路由

1. 使用以下命令删除您的网关路由（用您自己的值替换##值）：

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

2. 输出示例：

```
{
    "gatewayRoute": {
        "gatewayRouteName": "gatewayRouteName",
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualGateway/virtualGatewayName/gatewayRoute/gatewayRouteName",
            "createdAt": "2022-04-06T11:05:32.100000-05:00",
            "lastUpdatedAt": "2022-04-07T10:36:33.191000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 2
        },
        "spec": {
            "httpRoute": {
                "action": {
                    "target": {
                        "virtualService": {
                            "virtualServiceName": "serviceA.svc.cluster.local"
                        }
                    }
                },
                "match": {
                    "prefix": "/"
                }
            }
        },
        "status": {
            "status": "DELETED"
```

```
        },
        "virtualGatewayName": "virtualGatewayName"
    }
}
```

有关使用适用于 App Mesh 的 AWS CLI 删除网关路由的更多信息，请参阅AWS CLI 参考资料中的 delete-gateway-route命令。

虚拟节点

虚拟节点充当特定任务组的逻辑指针，如 Amazon ECS 服务或 Kubernetes 部署。当您创建虚拟节点时，必须为任务组指定服务发现方法。您的虚拟节点预计的任何入站流量被指定为侦听器。虚拟节点向其发送出站流量的任何虚拟服务均指定为后端。

新虚拟节点的响应元数据包含与该虚拟节点关联的 Amazon 资源名称 (ARN)。在 Amazon ECS 任务定义或容器组 (pod) 规范中将此值设置为任务组的 Envoy 代理容器的 `APPMESH_RESOURCE_ARN` 环境变量。例如，值可能为 `arn:aws:appmesh:us-west-2:111122223333:mesh/myMesh/virtualNode/myVirtualNode`。它随后将映射到 `node.id` 和 `node.cluster` Envoy 参数。在设置该变量时，您必须使用 `1.15.0` 或更高版本的 Envoy 映像。有关 App Mesh Envoy 变量的更多信息，请参阅 Envoy。

> **Note**
>
> 默认情况下，当 Envoy 在指标和跟踪中引用自身时，App Mesh 使用您在 `APPMESH_RESOURCE_ARN` 中指定的资源的名称。您可以通过使用自己的名称设置 `APPMESH_RESOURCE_CLUSTER` 环境变量来覆盖此行为。

创建虚拟节点

AWS Management Console

使用 AWS Management Console 创建虚拟节点。

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要在其中创建虚拟节点的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中选择虚拟节点。

4. 选择创建虚拟节点，然后为您的虚拟节点指定设置。
5. 对于虚拟节点名称，输入虚拟节点名称。
6. 对于服务发现方法，选择以下选项之一：

   - DNS-指定虚拟节点所代表的实际服务的 DNS 主机名。Envoy 代理部署在亚马逊 VPC 中。代理向为 VPC 配置的 DNS 服务器发送名称解析请求。如果主机名解析，则 DNS 服务器返回一个或多个 IP 地址。有关 VPC DNS 设置的更多信息，请参阅将 DNS 与您的 VPC 一起使用。对于 DNS 响应类型（可选），指定 DNS 解析器返回的端点类型。负载均衡器表示 DNS 解析器返回一组负载均衡的端点。端点意味着 DNS 解析器正在返回所有端点。默认情况下，假定响应类型为 负载均衡器。

     > **Note**
     >
     > 如果您使用 Route53，则需要使用负载均衡器。

   - AWS Cloud Map— 指定现有的服务名称和 HTTP 命名空间。或者，您也可以通过选择添加行并指定键和值来指定 App Mesh 可以查询 AWS Cloud Map 的属性。只会返回匹配所有键/值对的实例。要使用 AWS Cloud Map，您的账户必须具有`AWSServiceRoleForAppMesh`服务相关角色。有关 AWS Cloud Map 的更多信息，请参阅《AWS Cloud Map 开发人员指南》。
   - 无— 如果您的虚拟节点不希望任何入口流量，请选择该项。

7. IP 版本首选项

   通过切换覆盖默认 IP 版本行为，控制网格内的流量应使用哪个 IP 版本。默认情况下，App Mesh 使用各种 IP 版本。

   > **Note**
   >
   > 在虚拟节点上设置 IP 首选项只会覆盖为该特定节点上的网格设置的 IP 首选项。

   - 默认值
     - Envoy 的 DNS 解析器优先使用 `IPv6` 并在必要时回退到 `IPv4`。
     - 我们会优先使用 AWS Cloud Map 返回的 `IPv4` 地址（如果有），并在必要时回退到使用 `IPv6` 地址。
     - 为本地应用程序创建的端点使用 `IPv4` 地址。

- Envoy 侦听器绑定到所有 IPv4 地址。
- 首选 IPv6
  - Envoy 的 DNS 解析器优先使用 IPv6 并在必要时回退到 IPv4。
  - AWS Cloud Map 返回的 IPv6 地址如果可用，则使用该地址，并在必要时回退到使用 IPv4 地址
  - 为本地应用程序创建的端点使用 IPv6 地址。
  - Envoy 侦听器绑定到所有 IPv4 和 IPv6 地址。
- 首选 IPv4
  - Envoy 的 DNS 解析器优先使用 IPv4 并在必要时回退到 IPv6。
  - 我们会优先使用 AWS Cloud Map 返回的 IPv4 地址（如果有），并在必要时回退到使用 IPv6 地址。
  - 为本地应用程序创建的端点使用 IPv4 地址。
  - Envoy 侦听器绑定到所有 IPv4 和 IPv6 地址。
- 仅 IPv6
  - Envoy 的 DNS 解析器仅使用 IPv6。
  - 仅使用 AWS Cloud Map 返回的 IPv6 地址。如果 AWS Cloud Map 返回 IPv4 地址，则不使用 IP 地址，并将空结果返回给 Envoy。
  - 为本地应用程序创建的端点使用 IPv6 地址。
  - Envoy 侦听器绑定到所有 IPv4 和 IPv6 地址。
- 仅 IPv4
  - Envoy 的 DNS 解析器仅使用 IPv4。
  - 仅使用 AWS Cloud Map 返回的 IPv4 地址。如果 AWS Cloud Map 返回 IPv6 地址，则不使用 IP 地址，并将空结果返回给 Envoy。
  - 为本地应用程序创建的端点使用 IPv4 地址。
  - Envoy 侦听器绑定到所有 IPv4 和 IPv6 地址。

8. （可选）客户端策略默认值 - 配置与后端虚拟服务通信时的默认要求。

> **Note**
>
> - 如果要为现有虚拟节点启用传输层安全性协议 (TLS)，则建议您创建一个新的虚拟节点，来代表与现有虚拟节点相同的服务，以便在其上启用 TLS。然后使用虚拟路由器和路由逐渐将流量转移到新的虚拟节点。有关创建路由和调整过渡权重的更多

信息，请参阅 路由。如果您使用 TLS 更新现有提供流量的虚拟节点，则下游客户端 Envoy 代理可能会在您更新的虚拟节点的 Envoy 代理收到证书之前收到 TLS 验证上下文。这可能会导致下游 Envoy 代理上出现 TLS 协商错误。

- 必须为使用由后端服务的虚拟节点表示的应用程序部署的 Envoy 代理启用代理授权。我们建议您在启用代理授权时，将访问权限限制为仅访问该虚拟节点正在与之通信的虚拟节点。

- （可选）如果要求虚拟节点使用传输层安全性协议 (TLS) 与所有后端通信，请选择强制使用 TLS。
- （可选）如果您只想要求对一个或多个特定端口使用 TLS，请在端口中输入一个数字。要添加其他端口，请选择添加端口。如果您未指定任何端口，则会对所有端口强制执行 TLS。
- 对于验证方法，选择下列选项之一。您指定的证书必须已存在且符合特定要求。有关更多信息，请参阅 证书要求。
  - AWS Private Certificate Authority托管 - 选择一个或多个现有证书。有关使用带有 ACM 证书加密的示例应用程序部署网格的完整端到端演练，请参阅 GitHub 上的 AWS Certificate Manager 配置 TLS。
  - Envoy 密钥发现服务 (SDS) 托管 – 输入 Envoy 将使用密钥发现服务获取的密钥名称。
  - 本地文件托管 – 指定 Envoy 部署所在文件系统上的证书链文件的路径。有关使用本地文件加密的示例应用程序部署网格的完整端到端演练，请参阅 GitHub 上使用文件提供的 TLS 证书配置 TLS。
- （可选）输入主题备用名称。要添加其他 SAN，请选择添加 SAN。SAN 必须采用 FQDN 或 URI 格式。
- （可选）选择提供客户端证书和以下选项之一，以便在服务器请求客户端证书时提供该证书并启用双向 TLS 身份验证。要了解有关双向 TLS 的更多信息，请参阅 App Mesh 双向 TLS 身份验证文档。
  - Envoy 密钥发现服务 (SDS) 托管 – 输入 Envoy 将使用密钥发现服务获取的密钥名称。
  - 本地文件托管 - 指定 Envoy 部署所在文件系统上的证书链文件和私钥的路径。

9. （可选）服务后端 - 指定虚拟节点将与之通信的 App Mesh 虚拟服务。

- 输入您的虚拟节点与之通信的虚拟服务的 App Mesh 虚拟服务名称或完整 Amazon 资源名称 (ARN)。
- （可选）如果要为后端设置唯一的 TLS 设置，请选择 TLS 设置，然后选择覆盖默认值。
  - （可选）如果要求虚拟节点使用 TLS 与所有后端通信，请选择强制使用 TLS。

- （可选）如果您只想要求对一个或多个特定端口使用 TLS，请在端口中输入一个数字。要添加其他端口，请选择添加端口。如果您未指定任何端口，则会对所有端口强制执行 TLS。
- 对于验证方法，选择下列选项之一。您指定的证书必须已存在且符合特定要求。有关更多信息，请参阅 证书要求。
  - AWS Private Certificate Authority托管 — 选择一个或多个现有证书。
  - Envoy 密钥发现服务 (SDS) 托管 — 输入 Envoy 将使用密钥发现服务获取的密钥名称。
  - 本地文件托管 — 指定 Envoy 部署所在文件系统上的证书链文件的路径。
- （可选）输入主题备用名称。要添加其他 SAN，请选择添加 SAN。SAN 必须采用 FQDN 或 URI 格式。
- （可选）选择提供客户端证书和以下选项之一，以便在服务器请求客户端证书时提供该证书并启用双向 TLS 身份验证。要了解有关双向 TLS 的更多信息，请参阅 App Mesh 双向 TLS 身份验证文档。
  - Envoy 密钥发现服务 (SDS) 托管 — 输入 Envoy 将使用密钥发现服务获取的密钥名称。
  - 本地文件托管 — 指定 Envoy 部署所在文件系统上的证书链文件和私钥的路径。
- 要添加其他后端，请选择添加后端。

10. （可选）日志记录

    要配置日志记录，请输入您希望 Envoy 要使用的 HTTP 访问日志路径。建议您使用 `/dev/stdout` 路径，以便您可以使用 Docker 日志驱动程序来将 Envoy 日志导出至某个服务，如 Amazon CloudWatch Logs。

    > **Note**
    >
    > 日志必须仍由您应用程序中的代理进行接收并发送至目标。此文件路径仅指示 Envoy 要发送日志的位置。

11. 侦听器配置

    侦听器支持HTTP、HTTP/2、GRPC 和 TCP协议。不支持 HTTPS。

    a. 如果您的虚拟节点需要入站流量，请为侦听器指定端口和协议。http 侦听器允许连接转换到 websockets。您可以单击添加侦听器来添加多个侦听器。移除按钮将移除该侦听器。

b. （可选）启用连接池

连接池限制 Envoy 可同时与本地应用程序集群建立的连接数。它旨在保护您的本地应用程序免受连接不堪重负，并允许您根据应用程序的需求调整流量整形。

您可以为虚拟节点侦听器配置目标端连接池设置。App Mesh 默认将客户端连接池设置为无限，从而简化了网格配置。

> **Note**
>
> 连接池和端口映射协议必须相同。如果您的侦听器协议为 tcp，则仅指定 maxConnections。如果您的侦听器协议为 grpc 或 http2，则仅指定 maxRequests。如果您的侦听器协议为 http，您可以同时指定 maxConnections 和 maxPendingRequests。

- 对于最大连接数，请指定最大出站连接数。
- （可选）在最大待处理请求数中，指定在达到最大连接数之后，Envoy 队列中将排列的溢出请求数。默认值为 2147483647。

c. （可选）启用异常值检测

在客户端 Envoy 上应用的异常值检测允许客户端对观察到的已知不良故障的连接采取近乎立即的操作。它是一种断路器实现形式，用于跟踪上游服务中各个主机的运行状况。

异常值检测会动态确定上游集群中端点的性能是否与其他集群中的端点不同，并将其从运行正常的负载平衡集中移除。

> **Note**
>
> 要有效地为服务器虚拟节点配置异常值检测，该虚拟节点的服务发现方法可以 AWS Cloud Map 是 DNS，响应类型字段设置为 `ENDPOINTS`。如果您使用响应类型为 DNS 服务发现方法 `LOADBALANCER`，则 Envoy 代理只会选择一个 IP 地址来路由到上游服务。这会使从一组主机中弹出运行状况不佳的主机的异常值检测行为无效。有关 Envoy 代理与服务发现类型相关的行为的更多详细信息，请参阅服务发现方法部分。

- 对于服务器错误，请指定弹出所需的连续 5xx 错误数。

- 对于异常值检测间隔，请指定弹射扫描分析之间的时间间隔和单位。
- 对于基本摘除持续时间，指定摘除主机的基本时间和单位。
- 对于摘除百分比，指定负载平衡池中可摘除的主机的最大百分比。

d. （可选）启用运行状况检查 — 配置运行状况检查策略的设置。

运行状况检查策略是可选的，但是如果您为运行状况策略指定任何值，则必须为正常阈值、运行状况检查间隔、运行状况检查协议、超时时间和运行状况不佳阈值指定值。

- 对于 运行状况检查协议，请选择一个协议。如果选择 grpc，您的服务必须符合 GRPC 运行状况检查协议。
- 对于运行状况检查端口，指定应对其运行状况检查的端口。
- 对于正常阈值，指定在声明侦听器运行状况良好之前，必须出现的连续成功的运行状况检查次数。
- 对于运行状况检查间隔，指定执行每次运行状况检查间隔的时间（毫秒）。
- 对于路径，指定运行状况检查请求的目标路径。仅当运行状况检查协议为 `http` 或 `http2` 时才使用此值。其他协议将忽略此值。
- 对于超时周期，指定接收来自运行状况检查的响应时要等待的时间（毫秒）。
- 对于不正常阈值，指定在声明侦听器运行状况不正常之前，必须出现的连续失败的运行状况检查次数。

e. （可选）启用 TLS 终止 — 配置其他虚拟节点如何使用 TLS 与该虚拟节点通信。

- 对于模式，选择要在侦听器上配置 TLS 的模式。
- 对于 Certificate method，执行下列操作之一：证书必须满足具体要求。有关更多信息，请参阅 证书要求。
  - AWS Certificate Manager托管 — 选择现有证书。
  - Envoy 密钥发现服务 (SDS) 托管 — 输入 Envoy 将使用密钥发现服务获取的密钥名称。
  - 本地文件托管 — 在部署 Envoy 代理的文件系统上指定证书链文件的路径以及私钥。
- （可选）选择需要客户端证书和以下选项之一，以便在客户端提供证书时启用双向 TLS 身份验证。要了解有关双向 TLS 的更多信息，请参阅 App Mesh 双向 TLS 身份验证文档。
  - Envoy 密钥发现服务 (SDS) 托管 — 输入 Envoy 将使用密钥发现服务获取的密钥名称。
  - 本地文件托管 – 指定 Envoy 部署所在文件系统上的证书链文件的路径。

- （可选）输入主题备用名称。要添加其他 SAN，请选择添加 SAN。SAN 必须采用 FQDN 或 URI 格式。

f. （可选）超时

> **Note**
>
> 如果您指定的超时时间大于默认值，请务必设置虚拟路由器和超时时间大于默认值的路由。但是，如果您将超时时间缩短到低于默认值的值，则可以选择更新 Route 的超时时间。有关更多信息，请参阅 路由表。

- 请求超时 — 如果您为侦听器的协议选择了 grpc、http 或 http2，则可以指定请求超时。默认值为 15 秒。0 值禁用超时。
- 空闲时长 — 您可以为任何侦听器协议指定空闲时长。默认值为 300 秒。

12. 选择创建虚拟节点以完成。

AWS CLI

使用 AWS CLI 创建虚拟节点。

使用以下命令和输入 JSON 文件创建使用 DNS 进行服务发现的虚拟节点（用您自己的值替换*##值*）：

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

2. 示例 create-virtual-nod e-dns.json 的内容：

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

```
        "serviceDiscovery": {
            "dns": {
                "hostname": "serviceBv1.svc.cluster.local"
            }
        }
    },
    "virtualNodeName": "nodeName"
}
```

3. 输出示例：

```
{
    "virtualNode": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualNode/nodeName",
            "createdAt": "2022-04-06T09:12:24.348000-05:00",
            "lastUpdatedAt": "2022-04-06T09:12:24.348000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 1
        },
        "spec": {
            "listeners": [
                {
                    "portMapping": {
                        "port": 80,
                        "protocol": "http"
                    }
                }
            ],
            "serviceDiscovery": {
                "dns": {
                    "hostname": "serviceBv1.svc.cluster.local"
                }
            }
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualNodeName": "nodeName"
    }
```

```
}
```

有关使用针对 App Mesh 的AWS CLI 创建虚拟节点的更多信息，请参阅 AWS CLI 参考资料中的 create-virtual-node 命令。

删除虚拟节点

> **Note**
>
> 如果虚拟节点在任何路由中被指定为目标或在任何虚拟服务中指定为提供者，则无法将其删除。

AWS Management Console

要使用 AWS Management Console 删除虚拟节点

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要从中删除虚拟节点的虚拟网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中选择虚拟节点。
4. 在虚拟节点表中，选择要删除的虚拟节点，然后选择删除。要删除虚拟节点，您的账户 ID 必须列在虚拟节点的网格所有者或资源所有者列中。
5. 在确认框中，键入 **delete**，然后选择删除。

AWS CLI

要使用 AWS CLI 删除虚拟节点

1. 使用以下命令删除您的虚拟节点（用您自己的值替换##值）：

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

2. 输出示例：

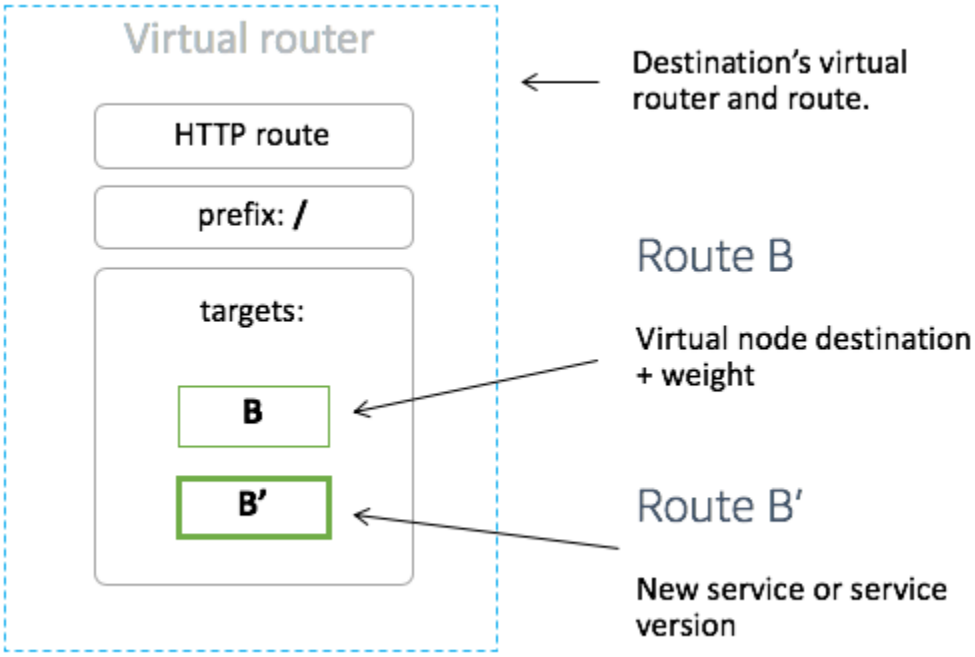
```
{
```

```
    "virtualNode": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualNode/nodeName",
            "createdAt": "2022-04-06T09:12:24.348000-05:00",
            "lastUpdatedAt": "2022-04-07T11:03:48.120000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 2
        },
        "spec": {
            "backends": [],
            "listeners": [
                {
                    "portMapping": {
                        "port": 80,
                        "protocol": "http"
                    }
                }
            ],
            "serviceDiscovery": {
                "dns": {
                    "hostname": "serviceBv1.svc.cluster.local"
                }
            }
        },
        "status": {
            "status": "DELETED"
        },
        "virtualNodeName": "nodeName"
    }
}
```

有关使用针对 App Mesh 的 AWS CLI 删除虚拟节点的更多信息，请参阅 AWS CLI 参考资料中的 delete-virtual-node 命令。

虚拟路由器

虚拟路由器处理用于您的网格内一个或多个虚拟服务的流量。创建虚拟路由器后，您可以为您的虚拟路由器创建并关联路由，以将传入请求定向至不同的虚拟节点。



您的虚拟路由器预计的任何入站流量均应指定为侦听器 。

创建虚拟路由器

AWS Management Console

使用 AWS Management Console 创建虚拟路由器

> **Note**
>
> 创建虚拟路由器时，必须添加带有标签的命名空间选择器，以标识将路由与创建的虚拟路由器关联的命名空间列表。

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要在其中创建虚拟路由器的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中，选择虚拟路由器。

4. 选择创建虚拟路由器。
5. 对于虚拟路由器名称，为您的虚拟路由器指定一个名称。不得超过 255 个字母、数字、连字符和下划线的组合。
6. （可选）对于侦听器，为您的虚拟路由器指定端口和协议。http侦听器允许连接转换到websockets。您可以单击添加侦听器来添加多个侦听器。移除按钮将移除该侦听器。
7. 选择创建虚拟路由器以完成。

AWS CLI

使用 AWS CLI 创建虚拟路由器。

使用以下命令创建虚拟路由器并输入 JSON（用您自己的值替换*##*值）：

1.
```
aws appmesh create-virtual-router \
     --cli-input-json file://create-virtual-router.json
```
2. 示例 create-virtual-router.json 的内容
3.
```
{
    "meshName": "meshName",
    "spec": {
        "listeners": [
            {
                "portMapping": {
                    "port": 80,
                    "protocol": "http"
                }
            }
        ]
    },
    "virtualRouterName": "routerName"
}
```
4. 输出示例：

```
{
    "virtualRouter": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualRouter/routerName",
```

```
            "createdAt": "2022-04-06T11:49:47.216000-05:00",
            "lastUpdatedAt": "2022-04-06T11:49:47.216000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 1
        },
        "spec": {
            "listeners": [
                {
                    "portMapping": {
                        "port": 80,
                        "protocol": "http"
                    }
                }
            ]
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualRouterName": "routerName"
    }
}
```


有关使用适用于 App Mesh 的创建虚拟路由器的 AWS CLI 更多信息，请参阅 AWS CLI 参考资料中的 create-virtual-router 命令。

删除虚拟路由器

> **Note**
>
> 如果虚拟路由器有任何路由，或者如果它被指定为任何虚拟服务的提供商，则无法将其删除。

AWS Management Console

要使用 AWS Management Console 删除虚拟路由器

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要从中删除虚拟路由器的网格。列出了您拥有的所有网格以及已与您共享的所有网格。

3. 在左侧导航中，选择虚拟路由器。
4. 在虚拟路由器表中，选择要删除的虚拟路由器，然后选择右上角的删除。要删除虚拟路由器，您的账户 ID 必须列在虚拟路由器的网格所有者或资源所有者列中。
5. 在确认框中，键入 **delete**，然后单击删除。

AWS CLI

使用 AWS CLI 删除虚拟路由器

1. 使用以下命令删除您的虚拟路由器（将*##*值替换为自己的虚拟路由器）：

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

2. 输出示例：

```
{
    "virtualRouter": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualRouter/routerName",
            "createdAt": "2022-04-06T11:49:47.216000-05:00",
            "lastUpdatedAt": "2022-04-07T10:49:53.402000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 2
        },
        "spec": {
            "listeners": [
                {
                    "portMapping": {
                        "port": 80,
                        "protocol": "http"
                    }
                }
            ]
        },
        "status": {
```

```
            "status": "DELETED"
        },
        "virtualRouterName": "routerName"
    }
}
```

有关使用适用于 App Mesh 的 AWS CLI 删除虚拟路由器的 更多信息，请参阅 AWS CLI 参考资料中的 delete-virtual-router 命令。

路由

路由与虚拟路由器关联。该路由用于匹配对虚拟路由器的请求并将流量分发到其关联的虚拟节点。如果路由与请求匹配，它可以将流量分配到一个或多个目标虚拟节点。您可以为每个虚拟节点指定相对权重。本主题可以帮助您处理服务网格中的路由。

创建路由

AWS Management Console

使用 AWS Management Console 创建路由

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要在其中创建路由的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中，选择虚拟路由器。
4. 选择要将新路由与之关联的虚拟路由器。如果没有列出任何路由器，则需要先创建一个虚拟路由器。
5. 在路由表中，选择创建路由。要创建路由，必须将您的账户 ID 列为该路径的资源所有者。
6. 对于路由名称，指定要用于您的路由的名称。
7. 对于路由类型，选择您要路由的协议。您选择的协议必须与您为虚拟路由器选择的侦听器协议和要将流量路由到的虚拟节点相匹配。
8. （可选）在路由优先级中，从 0-1000 中选择一个优先级以用于您的路由。路由会根据指定的值进行匹配，其中 0 是最高优先级。
9. （可选）选择其他配置。从下面的协议中，选择您为路由类型选择的协议，并根据需要在控制台中指定设置。

10. 对于目标配置，选择要将流量路由到的现有 App Mesh 虚拟节点并指定权重。您可以选择添加目标来添加其他目标。所有目标相加必须等于100。如果没有列出虚拟节点，则必须先创建一个。如果所选虚拟节点有多个侦听器，则需要目标端口。
11. 对于匹配配置，请指定：

    匹配配置不适用于 *tcp*

    - 如果所选类型为 http/http2：
      - （可选）方法 - 指定要在传入的 http/http2 请求中匹配的方法标头。
      - （可选）端口匹配 — 匹配传入流量的端口。如果此虚拟路由器有多个侦听器，则需要匹配端口。
      - （可选）前缀/精确/正则表达式路径 — 匹配网址路径的方法。
        - 前缀匹配 — 默认情况下，网关路由的匹配请求将重写为目标虚拟服务的名称，匹配的前缀将重写为 /。根据您配置虚拟服务的方式，它可能会使用虚拟路由器根据特定的前缀或标头将请求路由到不同的虚拟节点。

          > **Note**
          >
          > 如果您启用基于路径/前缀的匹配，App Mesh 会启用路径标准化（normalize_path 和 merge_slashes），以最大限度地减少出现路径混淆漏洞的可能性。
          > 当参与请求的各方使用不同的路径表示形式时，就会出现路径混淆漏洞。

        - 精确匹配 — exact 参数会禁用路径的部分匹配，并确保只有当路径与当前 url 完全匹配时，才会返回路由。
        - 正则表达式匹配 — 用于描述多个网址实际上可以识别网站上的单个页面的模式。
      - （可选）查询参数 — 此字段允许您匹配查询参数。
      - （可选）标头 - 指定 http 和 http2 的标头。它应该与传入的请求相匹配，以路由到目标虚拟服务。
    - 如果选择了 grpc 类型：
      - 服务名称 — 要匹配请求的目标服务。
      - 方法名称 — 要匹配请求的目标方法。
      - （可选）元数据—根据元数据的存在情况指定 Match。所有内容都必须匹配才能处理请求。
12. 选择创建路由。

AWS CLI

使用 AWS CLI 创建路由。

使用以下命令创建 gRPC 路径并输入 JSON（用您自己的值替换##值）：

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

2. 示例 create-route-grpc.json 的内容

```
{
    "meshName" : "meshName",
    "routeName" : "routeName",
    "spec" : {
       "grpcRoute" : {
          "action" : {
             "weightedTargets" : [
                {
                   "virtualNode" : "nodeName",
                   "weight" : 100
                }
             ]
          },
          "match" : {
             "metadata" : [
                {
                   "invert" : false,
                   "match" : {
                      "prefix" : "123"
                   },
                   "name" : "myMetadata"
                }
             ],
             "methodName" : "nameOfmethod",
             "serviceName" : "serviceA.svc.cluster.local"
          },
          "retryPolicy" : {
             "grpcRetryEvents" : [ "deadline-exceeded" ],
             "httpRetryEvents" : [ "server-error", "gateway-error" ],
             "maxRetries" : 3,
             "perRetryTimeout" : {
                "unit" : "s",
```

```
                "value" : 15
            },
            "tcpRetryEvents" : [ "connection-error" ]
          }
       },
       "priority" : 100
    },
    "virtualRouterName" : "routerName"
}
```

3. 输出示例：

```
{
    "route": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualRouter/routerName/route/routeName",
            "createdAt": "2022-04-06T13:48:20.749000-05:00",
            "lastUpdatedAt": "2022-04-06T13:48:20.749000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 1
        },
        "routeName": "routeName",
        "spec": {
            "grpcRoute": {
                "action": {
                    "weightedTargets": [
                        {
                            "virtualNode": "nodeName",
                            "weight": 100
                        }
                    ]
                },
                "match": {
                    "metadata": [
                        {
                            "invert": false,
                            "match": {
                                "prefix": "123"
                            },
                            "name": "myMetadata"
```

```
                        }
                    ],
                    "methodName": "nameOfMehod",
                    "serviceName": "serviceA.svc.cluster.local"
                },
                "retryPolicy": {
"grpcRetryEvents": [
                        "deadline-exceeded"
                    ],
                    "httpRetryEvents": [
                        "server-error",
                        "gateway-error"
                    ],
                    "maxRetries": 3,
                    "perRetryTimeout": {
                        "unit": "s",
                        "value": 15
                    },
                    "tcpRetryEvents": [
                        "connection-error"
                    ]
                }
            },
            "priority": 100
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualRouterName": "routerName"
    }
}
```

有关使用适用于 App Mesh 的 AWS CLI 创建路径的更多信息，请参阅 AWS CLI 参考资料中的 create-ro ute 命令。

gRPC

（可选）匹配

- （可选）输入目标服务的服务名称以匹配请求。如果没有指定名称，则将匹配对任何服务的请求。

- （可选）输入目标方法的方法名称以匹配请求。如果没有指定名称，则将匹配对任何方法的请求。如果指定方法名称，则必须指定服务名称。

（可选）元数据

选择添加元数据。

- （可选）输入要作为路由依据的元数据名称，选择匹配类型，然后输入匹配值。选择反转将匹配相反情况。例如，如果您将元数据名称指定为myMetadata，将匹配类型指定为精确，将匹配值指定为123，然后选择反转，则该路由将匹配元数据名称以 123 之外的任何值开头的请求。
- （可选）选择添加元数据以添加最多十个元数据项。

（可选）重试策略

重试策略使客户端可以保护自己免受间歇性网络故障或间歇性服务器端故障的影响。重试策略是可选的，但我们建议您这样做。重试超时值定义每次重试尝试（包括初次尝试）的超时时间。如果您未定义重试策略，那么 App Mesh 可能会自动为每条路由创建默认策略。有关更多信息，请参阅 默认路由重试策略。

- 对于重试超时，输入超时持续时间的单位数。如果您选择任何协议重试事件，则必须输入一个值。
- 在重试超时单位中，选择一个单位。如果您选择任何协议重试事件，则必须输入一个值。
- 对于最大重试次数，输入请求失败时进行的最大重试次数。如果您选择任何协议重试事件，则必须输入一个值。我们建议至少将值设置为2。
- 选择一个或多个 HTTP 重试事件。我们建议至少选择流错误和网关错误。
- 选择 TCP 重试事件。
- 选择一个或多个 gRPC 重试事件。我们建议至少选择已取消且不可用。

（可选）超时

- 默认值为 15 秒。如果您指定了重试策略，则在此处指定的持续时间应始终大于或等于重试持续时间乘以您在重试策略中定义的最大重试次数，这样您的重试策略才能完成。如果您指定的持续时间大于 15 秒，请确保为任何虚拟节点 Target 的侦听器指定的超时时间也大于 15 秒。有关更多信息，请参阅虚拟节点。
- 0 值禁用超时。
- 路径可以空闲的最长时间。

HTTP 和 HTTP/2

（可选）匹配

- 指定路由应匹配的前缀。例如，如果您的虚拟服务名称为 `service-b.local`，并且您希望路由将请求匹配到 `service-b.local/metrics`，则前缀应为 `/metrics`。指定所有流量的 `/` 路由。
- （可选）选择一种方法。
- （可选）选择一个方案。仅适用于 HTTP2 路由。

（可选）标头

- （可选）选择添加标头。输入要作为路由依据的标头名称，选择匹配类型，然后输入匹配值。选择反转将匹配相反情况。例如，如果您指定一个带前缀 123 且名为 `clientRequestId` 的标头，然后选择反转，则该路由将与标头是以 123 之外的任何值开头的请求相匹配。
- （可选）选择添加标头。您最多可以添加十个标头。

（可选）重试策略

重试策略使客户端可以保护自己免受间歇性网络故障或间歇性服务器端故障的影响。重试策略是可选的，但我们建议您这样做。重试超时值定义每次重试尝试（包括初次尝试）的超时时间。如果您未定义重试策略，那么 App Mesh 可能会自动为每条路由创建默认策略。有关更多信息，请参阅 默认路由重试策略。

- 对于重试超时，输入超时持续时间的单位数。如果您选择任何协议重试事件，则必须输入一个值。
- 在重试超时单位中，选择一个单位。如果您选择任何协议重试事件，则必须输入一个值。
- 对于最大重试次数，输入请求失败时进行的最大重试次数。如果您选择任何协议重试事件，则必须输入一个值。我们建议至少将值设置为2。
- 选择一个或多个 HTTP 重试事件。我们建议至少选择流错误和网关错误。
- 选择 TCP 重试事件。

（可选）超时

- 请求超时 — 默认超时时间为 15 秒。如果您指定了重试策略，则在此处指定的持续时间应始终大于或等于重试持续时间乘以您在重试策略中定义的最大重试次数，这样您的重试策略才能完成。
- 空闲时长 — 默认值为 300 秒。
- `0` 值禁用超时。

> **Note**
>
> 如果您指定的超时时间大于默认值，请确保为所有虚拟节点参与者侦听器指定的超时时间也大于默认值。但是，如果您将超时时间缩短到低于默认值的值，则可以选择更新虚拟节点的超时时间。有关更多信息，请参阅虚拟节点。

TCP

（可选）超时

- 空闲时长 — 默认值为 300 秒。
- `0` 值禁用超时。

删除路由

AWS Management Console

使用 AWS Management Console 删除路由表

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 选择要从中删除路径的网格。列出了您拥有的所有网格以及已与您共享的所有网格。
3. 在左侧导航中，选择虚拟路由器。
4. 选择要从中删除路由的路由器。
5. 在路径表中，选择要删除的路线，然后选择右上角的删除。
6. 在确认框中，键入 **`delete`**，然后单击删除。

AWS CLI

使用 AWS CLI 删除路由表

1. 使用以下命令删除您的路线（用您自己的值替换*##*值）：

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

2. 输出示例：

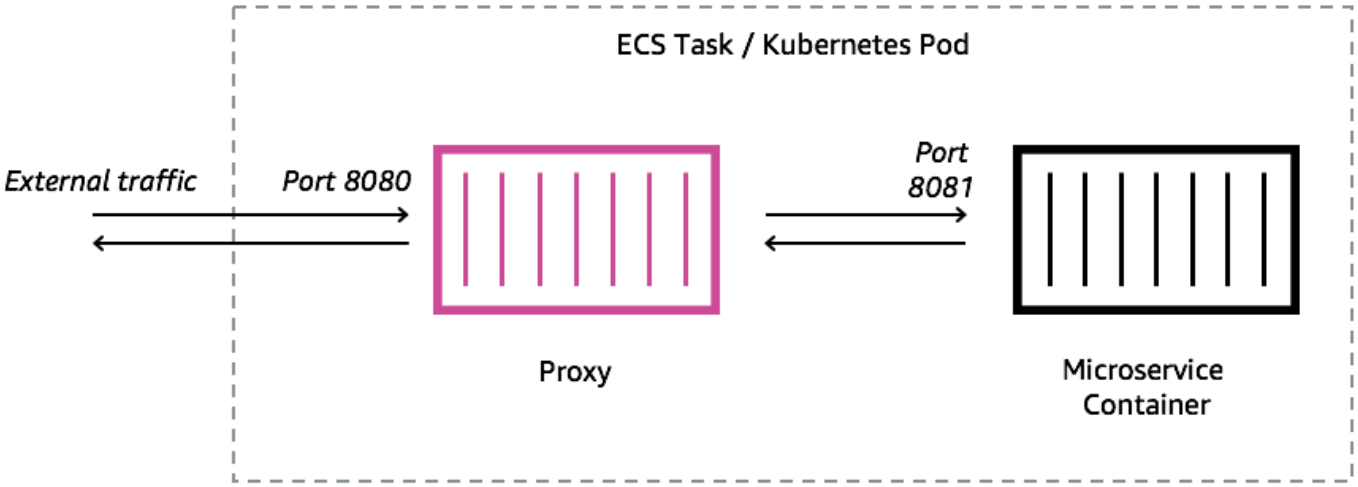
```
{
    "route": {
        "meshName": "meshName",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:210987654321:mesh/meshName/
virtualRouter/routerName/route/routeName",
            "createdAt": "2022-04-06T13:46:54.750000-05:00",
            "lastUpdatedAt": "2022-04-07T10:43:57.152000-05:00",
            "meshOwner": "123456789012",
            "resourceOwner": "210987654321",
            "uid": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "version": 2
        },
        "routeName": "routeName",
        "spec": {
            "grpcRoute": {
                "action": {
                    "weightedTargets": [
                        {
                            "virtualNode": "nodeName",
                            "weight": 100
                        }
                    ]
                },
                "match": {
                    "metadata": [
                        {
                            "invert": false,
                            "match": {
                                "prefix": "123"
                            },
                            "name": "myMetadata"
                        }
                    ],
                    "methodName": "methodName",
                    "serviceName": "serviceA.svc.cluster.local"
                },
                "retryPolicy": {
                    "grpcRetryEvents": [
                        "deadline-exceeded"
                    ],
```

```
                    "httpRetryEvents": [
                        "server-error",
                        "gateway-error"
                    ],
                    "maxRetries": 3,
                    "perRetryTimeout": {
                        "unit": "s",
                        "value": 15
                    },
                    "tcpRetryEvents": [
                        "connection-error"
                    ]
                }
            },
            "priority": 100
        },
        "status": {
            "status": "DELETED"
        },
        "virtualRouterName": "routerName"
    }
}
```

有关使用适用于 App Mesh 删除路径 AWS CLI 的更多信息，请参阅 AWS CLI 参考资料中的 delete-route 命令。

Envoy 镜像

AWS App Mesh 是基于 Envo y 代理的服务网格。



您必须向 Amazon ECS 任务、Kubernetes pod 或用您的 App Mesh 端点表示的 Amazon EC2 实例（例如：虚拟节点或虚拟网关）中添加 Envoy 代理。App Mesh 出售了一个 Envoy 代理容器镜像，该镜像已修补了最新的漏洞和性能更新。在向你提供新图像之前，App Mesh 会根据 App Mesh 功能集测试每个新的 Envoy 代理版本。

特使图像变体

App Mesh 提供了 Envoy 代理容器镜像的两种变体。两者的区别在于 Envoy 代理如何与 App Mesh 数据平面进行通信，以及 Envoy 代理如何相互通信。一个是标准映像，它与标准 App Mesh 服务端点通信。另一种变体符合 FIPS，它与 App Mesh FIPS 服务端点通信，并在 App Mesh 服务之间的 TLS 通信中强制执行 FIPS 加密。

您可以从下面的列表中选择一个地区镜像，也可以从我们的公共存储库中选择一个名为 `aws-appmesh-envoy` 的镜像。

> **Important**
>
> - 从 2023 年 6 月 30 日起，只有 Envoy 镜像`v1.17.2.0-prod`或更高版本可以与 App Mesh 兼容。对于之前使用 Envoy 镜像的当前客户`v1.17.2.0`，尽管现有的 envoy 将继续兼容，但我们强烈建议迁移到最新版本。

- 作为最佳实践，强烈建议定期将 Envoy 版本升级到最新版本。只有最新的 Envoy 版本经过最新安全补丁、功能发布和性能改进的验证。
- 版本 1.17 是对 Envoy 的重大更新。有关更多详细信息，请参阅更新/迁移到 Envoy 1.17。
- 可兼容版本 1.20.0.1 或更高版本 ARM64。
- 如需 IPv6 支持，需要 Envoy 版本 1.20 或更高版本。

除me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1 和 af-south-1 之外的所有受支持地区。您可以用me-south-1、ap-east-1、ap-southeast-3、eu-south-1、il-central-1 和 af-south-1之外的任何地区替换####。

Standard

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

符合 FIPS 标准

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

me-south-1

Standard

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

符合 FIPS 标准

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

ap-east-1

Standard

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

符合 FIPS 标准

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

ap-southeast-3

Standard

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

符合 FIPS 标准

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

eu-south-1

Standard

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

符合 FIPS 标准

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

il-central-1

Standard

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

符合 FIPS 标准

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

af-south-1

Standard

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

符合 FIPS 标准

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

Public repository

Standard

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

符合 FIPS 标准

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

> **Note**
>
> 我们建议为 Envoy 容器分配 512 个 CPU 单元及至少 64 MiB 的内存。Fargate 上可设置的最小内存为 1024 MiB。如果容器洞察或其他指标表明由于负载增加而导致资源不足，则可以增加对 Envoy 容器的资源分配。

> **Note**
>
> 从 `v1.22.0.0` 开始的所有 `aws-appmesh-envoy` 镜像发布版本都是作为 Distroless Docker 映像构建的。我们进行此项更改是为了缩小映像尺寸并降低映像中未使用的软件包中存在的漏洞暴露风险。如果你在 aws-appmesh-envoy 镜像之上构建，并且依赖一些 AL2 基础包（例如 yum）和功能，那么我们建议你从镜像内部复制二进制文件来构建一个基于 AL2 的新 Docker `aws-appmesh-envoy` 镜像。
>
> 运行此脚本生成带有标签的自定义的 Docker 映像 `aws-appmesh-envoy:v1.22.0.0-prod-al2`:
>
> ```
> cat << EOF > Dockerfile
> FROM public.ecr.aws/appmesh/aws-appmesh-envoy:v1.22.0.0-prod as envoy
> ```

```
FROM public.ecr.aws/amazonlinux/amazonlinux:2
RUN yum -y update && \
    yum clean all && \
    rm -rf /var/cache/yum

COPY --from=envoy /usr/bin/envoy /usr/bin/envoy
COPY --from=envoy /usr/bin/agent /usr/bin/agent
COPY --from=envoy /aws_appmesh_aggregate_stats.wasm /
aws_appmesh_aggregate_stats.wasm

CMD [ "/usr/bin/agent" ]
EOF

docker build -f Dockerfile -t aws-appmesh-envoy:v1.22.0.0-prod-al2 .
```

在 Amazon ECR 中访问此容器镜像的权限由 AWS Identity and Access Management (IAM) 控制。因此，您必须使用 IAM 来验证您拥有对 Amazon ECR 的读取权限。例如，在使用 Amazon ECS 时，您可以为 Amazon ECS 任务分配适当的任务执行角色。如果您使用限制访问特定的 Amazon ECR 资源的 IAM 策略，请务必确认您允许访问标识 `aws-appmesh-envoy` 存储库的特定地区的 Amazon 资源名称（ARN）。例如，在 `us-west-2` 地区，您允许访问以下资源：`arn:aws:ecr:us-west-2:840364872350:repository/aws-appmesh-envoy`。有关更多信息，请参阅 Amazon ECR 托管策略。如果您在亚马逊 EC2 实例上使用 Docker，请对 Docker 存储库进行身份验证。有关更多信息，请参阅注册表身份验证。

我们偶尔会发布新的 App Mesh 功能，这些功能依赖于 Envoy 的更改，这些更改尚未合并到上游 Envoy 镜像中。要在上游合并 Envoy 变更之前使用这些新的 App Mesh 功能，您必须使用 App Mesh 提供的 Envoy 容器镜像。有关变更列表，请参阅 App Mesh GitHub 路线图中与`Envoy Upstream`标签有关的问题。我们建议将 App Mesh Envoy 容器的映像作为最佳支持选项。

Envoy 配置变量

使用以下环境变量为您的 App Mesh 虚拟节点任务组配置 Envoy 容器。

> **Note**
>
> App Mesh Envoy 1.17 不支持 Envoy 的 v2 xDS API。如果您使用的是接受 Envoy 配置文件的 Envoy 配置变量，则必须将其更新为最新的 v3 xDS API。

必需的变量

所有 App Mesh Envoy 容器都需要以下环境变量。此变量只能用于 Envoy 映像版本 `1.15.0` 或更高版本。如果您使用的是较早版本的映像，则必须改用 `APPMESH_VIRTUAL_NODE_NAME` 变量。

`APPMESH_RESOURCE_ARN`

在将 Envoy 容器添加到任务组时，请将此环境变量设置为任务组所代表的虚拟节点或虚拟网关的 ARN。下面的列表包含示例 ARN：

- 虚拟节点 – *arn: aws: appmesh#####111122223333: mesh/ meshName / virtualNode/ virtualNodeName*
- 虚拟网关 – *arn: aws: appmesh#####111122223333: mesh/ meshName / VirtualGateway/ virtualGatewayName*

使用 App Mesh 预览频道时，ARN 必须使用 *us-west-2* 地区并使用 `appmesh-preview` 来替代 `appmesh`。例如，App Mesh 预览频道中虚拟节点的 ARN 为 `arn:aws:`**`appmesh-preview:us-west-2:`***`111122223333`*`:mesh/`*`meshName`*`/virtualNode/`*`virtualNodeName`*。

可选变量

以下环境变量对于 App Mesh Envoy 容器是可选的。

`ENVOY_LOG_LEVEL`

指定 Envoy 容器的日志级别。

有效值：`trace`、`debug`、`info`、`warn`、`error`、`critical`、`off`

默认：`info`

`ENVOY_INITIAL_FETCH_TIMEOUT`

指定 Envoy 在初始化过程中等待来自管理服务器的第一个配置响应的时间。

有关更多信息，请参阅 Envoy 文档中的 Configuration 资源。设置为 `0` 时，没有超时。

默认：`0`

`ENVOY_CONCURRENCY`

在启动 Envoy 时设置 `--concurrency` 命令行选项。默认情况下，不设置该选项。此选项在 Envoy 版本 `v1.24.0.0-prod` 或更高版本中可用。

有关更多信息，请参阅命令行选项。

管理员变量

使用这些环境变量来配置 Envoy 的管理接口。

`ENVOY_ADMIN_ACCESS_PORT`

指定 Envoy 要侦听的自定义管理端口。默认值：`9901`。

> **Note**
>
> Envoy 管理端口应不同于虚拟网关或虚拟节点上的任何侦听器端口

`ENVOY_ADMIN_ACCESS_LOG_FILE`

指定写入 Envoy 访问日志的自定义路径。默认值：`/tmp/envoy_admin_access.log`。

`ENVOY_ADMIN_ACCESS_ENABLE_IPV6`

切换 Envoy 的管理界面以接受 IPv6 流量，这样该接口就可以同时接受 IPv4 流量和 IPv6 流量。默认情况下，此标志设置为 false，并且 Envoy 只侦听 IPv4 流量。此变量只能用于 Envoy 镜像版本 1.22.0 或更高版本。

代理变量

使用这些环境变量为 Envoy 配置 AWS App Mesh 代理。有关更多信息，请参阅适用于 Envoy 的 App Mesh 代理。

`APPNET_ENVOY_RESTART_COUNT`

指定代理在正在运行的任务或容器组 (pod) 中重新启动 Envoy 代理进程的次数（如果容器组 (pod) 已退出）。代理还会在每次 Envoy 退出时记录退出状态，以便于故障排除。该变量的默认值为 `0`。设置默认值后，代理不会尝试重新启动该进程。

默认：`0`

最高：`10`

PID_POLL_INTERVAL_MS

指定代理检查 Envoy 代理进程状态的时间间隔（以毫秒为单位）。默认值为 100。

默认：100

最低：100

最高：1000

LISTENER_DRAIN_WAIT_TIME_S

指定 Envoy 代理在进程退出之前等待活动连接关闭的时间（以秒为单位）。

默认：20

最低：5

最高：110

APPNET_AGENT_ADMIN_MODE

启动代理的管理接口服务器并将其绑定到 tcp 地址或 unix 套接字。

有效值：tcp、uds

APPNET_AGENT_HTTP_PORT

指定用于在 tcp 模式下绑定代理管理接口的端口。确保端口值为 > 1024 if uid != 0。确保端口小于65535。

默认：9902

APPNET_AGENT_ADMIN_UDS_PATH

在 uds 模式下为代理的管理接口指定 unix 域套接字路径。

默认：/var/run/ecs/appnet_admin.sock

跟踪变量

您可以不配置以下跟踪驱动程序或其中一个跟踪驱动程序。

AWS X-Ray 变量

使用以下环境变量来配置带 AWS X-Ray的 App Mesh。有关更多信息，请参阅 AWS X-Ray 开发人员指南。

`ENABLE_ENVOY_XRAY_TRACING`

使用 `127.0.0.1:2000` 作为默认进程守护程序端点启用 X-Ray 跟踪。要启用，请将值设置为 `1`。默认值为 `0`。

`XRAY_DAEMON_PORT`

指定端口值以覆盖默认 X-Ray 进程守护程序端口：`2000`。

`XRAY_SAMPLING_RATE`

指定采样率以覆盖 X-Ray tracer 的默认采样率 `0.05` (5%)。将该值指定为介于 `0` 和 `1.00` 之间的小数(100%)。如果已指定 `XRAY_SAMPLING_RULE_MANIFEST`，则该值将被覆盖。版本 `v1.19.1.1-prod` 及更高版本的 Envoy 映像支持此变量。

`XRAY_SAMPLING_RULE_MANIFEST`

在 Envoy 容器文件系统中指定文件路径，为 X-Ray tracer 配置本地化的自定义采样规则。有关更多信息，请参阅《AWS X-Ray 开发人员指南》中的采样规则。版本 `v1.19.1.0-prod` 及更高版本的 Envoy 映像支持此变量。

`XRAY_SEGMENT_NAME`

为轨迹指定分段名称以覆盖默认 X-Ray 分段名称。默认情况下，此值将设置为 `mesh/resourceName`。Envoy 镜像版本 `v1.23.1.0-prod` 或更高版本支持此变量。

Datadog 跟踪变量

以下环境变量可帮助您使用 Datadog 代理追踪器配置 App Mesh。有关更多信息，请参阅 Datadog 文档中的代理配置。

`ENABLE_ENVOY_DATADOG_TRACING`

使用作为默认 Datadog 代理端点的 `127.0.0.1:8126` 启用 Datadog 跟踪收集。要启用，请将该值设置为 `1`（默认值为 `0`）。

`DATADOG_TRACER_PORT`

指定一个端口值以覆盖默认 Datadog 代理端口：`8126`。

`DATADOG_TRACER_ADDRESS`

指定 IP 地址以覆盖默认的 Datadog 代理地址：`127.0.0.1`。

`DD_SERVICE`

为跟踪指定服务名称以覆盖默认的 Datadog 服务名称：`envoy-meshName/virtualNodeName`。版本 `v1.18.3.0-prod` 及更高版本的 Envoy 映像支持此变量。

Jaeger 追踪变量

使用以下环境变量为 App Mesh 配置 Jaeger 追踪。有关更多信息，请参阅 Jaeger 文档中的入门。版本 `1.16.1.0-prod` 及更高版本的 Envoy 镜像支持这些变量。

`ENABLE_ENVOY_JAEGER_TRACING`

使用 `127.0.0.1:9411` 作为默认 Jaeger 端点启用 Jaeger 跟踪收集。要启用，请将该值设置为 `1`（默认值为 `0`）。

`JAEGER_TRACER_PORT`

指定端口值以覆盖默认 Jaeger 端口：`9411`。

`JAEGER_TRACER_ADDRESS`

指定 IP 地址以覆盖默认 Jaeger 地址：`127.0.0.1`。

`JAEGER_TRACER_VERSION`

指定收集器是否需要 `PROTO` 编码格式的 `JSON` 轨迹。默认情况下，该值将设置为 `PROTO`。Envoy 镜像版本 `v1.23.1.0-prod` 或更高版本支持此变量。

特使跟踪变量

将以下环境变量设置为使用您自己的跟踪配置。

`ENVOY_TRACING_CFG_FILE`

在 Envoy 容器文件系统中指定文件路径。有关更多信息，请参阅 Envoy 文档中的`config.trace.v3.Tracing`。

> **Note**
>
> 如果跟踪配置需要指定跟踪集群，请确保在同一个跟踪配置文件 `static_resources` 中配置关联的集群配置。例如，Zipkin 有一个用于托管跟踪收集器的集群名称`collector_cluster`字段，该集群需要静态定义。

DogStatsD 变量

使用以下环境变量将 App Mesh 配置为 DogStats D。有关更多信息，请参阅 DogStatsD 文档。

`ENABLE_ENVOY_DOG_STATSD`

使用`127.0.0.1:8125`作为默认守护程序端点启用 DogStats D 统计信息。要启用，请将值设置为 `1`。

`STATSD_PORT`

指定端口值以覆盖默认 DogStats D 守护程序端口。

`STATSD_ADDRESS`

指定 IP 地址值以覆盖默认 DogStats D 守护程序 IP 地址。默认值：`127.0.0.1`。此变量只能用于 Envoy 映像版本 `1.15.0` 或更高版本。

`STATSD_SOCKET_PATH`

为 DogStats D 守护程序指定 unix 域套接字。如果未指定此变量且启用了 DogStats D，则此值默认为 DogStats D 守护程序 IP 地址端口。`127.0.0.1:8125`如果指定的`ENVOY_STATS_SINKS_CFG_FILE`变量包含 stats sinks 配置，则它将覆盖所有 DogStats D 变量。Envoy 镜像版本 `v1.19.1.0-prod` 或更高版本支持此变量。

App Mesh 变量

以下变量可帮助您配置 App Mesh。

`APPMESH_PREVIEW`

将该值设置为 `1` 到 App Mesh 预览频道端点。有关使用 App Mesh 预览频道的更多信息，请参阅 App Mesh 预览频道。

`APPMESH_RESOURCE_CLUSTER`

默认情况下，当 Envoy 在指标和跟踪中引用自己`APPMESH_RESOURCE_ARN`时，App Mesh 使用您在中指定的资源名称。您可以通过使用自己的名称设置 `APPMESH_RESOURCE_CLUSTER` 环境变量来覆盖此行为。此变量只能用于 Envoy 映像版本 `1.15.0` 或更高版本。

`APPMESH_METRIC_EXTENSION_VERSION`

将该值设置 `1` 为可启用 App Mesh 指标扩展。有关 App Mesh 指标扩展的更多信息，请参阅 App Mesh 的指标扩展。

`APPMESH_DUALSTACK_ENDPOINT`

将该值设置为 1 以连接到 App Mesh 双堆栈端点。设置此标志后，Envoy 将使用我们支持双堆栈的域。默认情况下，此标志设置为 false，并且仅连接到我们的 IPv4 域名。此变量只能用于 Envoy 镜像版本 1.22.0 或更高版本。

特使统计变量

使用以下环境变量为 App Mesh 配置 Envoy 统计信息。有关更多信息，请参阅 Envoy Stats 文档。

`ENABLE_ENVOY_STATS_TAGS`

允许使用 App Mesh 定义的标签 `appmesh.mesh` 和 `appmesh.virtual_node`。有关更多信息，请参阅 config.metrics.v3。 TagSpecifier在特使文档中。要启用，请将值设置为 1。

`ENVOY_STATS_CONFIG_FILE`

在 Envoy 容器文件系统中指定文件路径，用自己的文件路径覆盖默认的 Stats 标签配置文件。有关更多信息，请参阅 config.metrics.v3。 StatsConfig。

> **Note**
>
> 设置包含统计过滤器的自定义统计信息配置可能会导致 Envoy 进入无法再与 App Mesh 世界状态正确同步的状态。这是 Envoy 中的一个错误。我们建议不要在 Envoy 中对统计数据进行任何过滤。如果绝对需要过滤，我们在路线图上列出了本次发布版本的几个解决方法。

`ENVOY_STATS_SINKS_CFG_FILE`

在 Envoy 容器文件系统中指定文件路径，用自己的文件路径覆盖默认配置。有关更多信息，请参阅 config.metrics.v3。 StatsSink在特使文档中。

已弃用的变量

Envoy 版本 `1.15.0` 或更高版本不再支持环境变量 `APPMESH_VIRTUAL_NODE_NAME` 和 `APPMESH_RESOURCE_NAME`。但是，现有网格仍然支持它们。与其在 Envoy 版本 `1.15.0` 或更高版本中使用这些变量，不如将这些变量 `APPMESH_RESOURCE_ARN` 用于所有 App Mesh 端点。

Envoy 默认值由 App Mesh 设置

以下各节提供了有关 App Mesh 设置的路由重试策略和断路器的 Envoy 默认值的信息。

默认路由重试策略

如果在 2020 年 7 月 29 日之前您的账户中没有网格，App Mesh 会在 2020 年 7 月 29 日当天或之后自动为账户中任何网格中的所有 HTTP、HTTP/2 和 gRPC 请求创建默认的 Envoy 路由重试策略。如果您的账户在 2020 年 7 月 29 日之前有任何网格，则不会为 2020 年 7 月 29 日之前、当天或之后存在的任何 Envoy 路线创建默认策略。除非你在AWS 支持下开票。支持部门处理票证后，将为 App Mesh 在处理票证之日或之后创建的任何 future Envoy 路线创建默认策略。有关 Envoy 路由重试策略的更多信息，请参阅 config.route.v3。 RetryPolicy在特使文档中。

当您创建 App Mesh 路由或为 App Mesh 虚拟服务定义虚拟节点提供者时，App Mesh 会创建 Envoy 路由。尽管您可以创建 App Mesh 路由重试策略，但无法为虚拟节点提供商创建 App Mesh 重试策略。

默认策略无法通过 App Mesh API 显示。默认策略只能通过 Envoy 查看。要查看配置，请启用管理界面并向 Envoy 发送请求`config_dump`。此默认策略包含以下设置：

- 最大重试次数 — 2
- gRPC 重试事件 — `UNAVAILABLE`
- HTTP 重试事件 — `503`

> **Note**
>
> 无法创建用于查找特定 HTTP 错误代码的 App Mesh 路由重试策略。但是，App Mesh 路由重试策略可以查找 `server-error` 或 `gateway-error`。这两者都包含 `503` 错误。有关更多信息，请参阅 路由。

- TCP 重试事件 — 以及 `connect-failure` 和 `refused-stream`

> **Note**
>
> 不可能创建用于查找这两个事件的 App Mesh 路由重试策略。但是，App Mesh 路由重试策略可以查找 `connection-error`，这等同于 `connect-failure`。有关更多信息，请参阅路由。

- 重置 — 如果上游服务器根本没有响应（断开连接/重置/读取超时），Envoy 会尝试重试。

默认断路器

当您在 App Mesh 中部署 Envoy 时，某些断路器设置会设置 Envoy 的默认值。有关更多信息，请参阅集群。 CircuitBreakers.Envoy 文档中的阈值。这些设置无法通过 App Mesh API 查看。这些设置只能通过 Envoy 看到。要查看配置，请启用管理界面并向 Envoy 发送请求`config_dump`。

如果在 2020 年 7 月 29 日之前您的账户中没有网格，那么对于您在 2020 年 7 月 29 日当天或之后创建的网格中部署的每个 Envoy，App Mesh 都会通过更改以下设置的 Envoy 默认值来有效地禁用断路器。如果你的账户在 2020 年 7 月 29 日之前有任何网格，则除非你在AWS 支持下开票，否则将为你在 2020 年 7 月 29 日当天或之后在 App Mesh 中部署的任何 Envoy 设置默认值。支持人员处理完工单后，App Mesh 将为处理工单之日之后部署的所有 Envoy 设置的 App Mesh 默认值设置为：

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

> **Note**
>
> 无论您的 Envoy 是 Envoy 还是 App Mesh 的默认断路器值，您都无法修改这些值。

更新/迁移到 Envoy 1.17

带有 SPIRE 的密钥发现服务

如果您使用带 App Mesh 的 SPIRE（SPIFFE 运行时系统环境）向您的服务分发信任证书，请确认您使用的是版本至少为 `0.12.0` 的 SPIRE 代理（2020 年 12 月发布）。这是之后第一个可以支持 Envoy 版本的版本 `1.16`。

正则表达式更改

从 Envoy `1.17` 开始，App Mesh 将 Envoy 配置为默认使用 RE2 正则表达式引擎。对于大多数用户来说，这种变化是显而易见的，但是路由或网关路由中的匹配不再允许在正则表达式中进行正向或反向引用。

正向和负向展望

正向 — 正向前看是一个带括号的表达式，开头为：?=。

```
(?=example)
```

它们在进行字符串替换时最有用，因为它们允许匹配字符串而不必在匹配中消耗字符。由于 App Mesh 不支持替换正则表达式字符串，因此我们建议您将其替换为常规匹配项。

```
(example)
```

负向 — 负向前看是一个带括号的表达式，开头为：?!。

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

带圆括号的表达式用于断言表达式的部分与给定输入不匹配。在大多数情况下，你可以用零量词替换它们。

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

如果表达式本身是一个字符类，则可以？使用否定整个类并将其标记为可选。

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

根据您的用例，您也可以更改路线来处理此问题。

```
{
    "routeSpec": {
        "priority": 0,
        "httpRoute": {
            "match": {
                "headers": [
                    {
                        "name": "x-my-example-header",
                        "match": {
                            "regex": "^prefix(?!suffix)"
                        }
                    }
                ]
            }
```

```
        }
    }
}

{
    "routeSpec": {
        "priority": 1,
        "httpRoute": {
            "match": {
                "headers": [
                    {
                        "name": "x-my-example-header",
                        "match": {
                            "regex": "^prefix"
                        }
                    }
                ]
            }
        }
    }
}
```

第一个路由匹配会查找以“prefix”开头但后面不是“suffix”的标头。第二条路由的作用是匹配所有其他以“prefix”开头的标头，包括那些以“suffix”结尾的标头。取而代之的是，也可以将其逆转，以消除负向前看。

```
{
    "routeSpec": {
        "priority": 0,
        "httpRoute": {
            "match": {
                "headers": [
                    {
                        "name": "x-my-example-header",
                        "match": {
                            "regex": "^prefix.*?suffix"
                        }
                    }
                ]
            }
        }
    }
}
```

```
{
    "routeSpec": {
        "priority": 1,
        "httpRoute": {
            "match": {
                "headers": [
                    {
                        "name": "x-my-example-header",
                        "match": {
                            "regex": "^prefix"
                        }
                    }
                ]
            }
        }
    }
}
```

此示例反转路由，为以“suffix”结尾的标头提供更高的优先级，而所有其他以“prefix”开头的标头在优先级较低的路由中都匹配。

反向引用

反向引用是一种通过向前一个带括号的组重复来编写较短表达式的方法。他们有这样的表格。

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

反斜杠后 \ 跟一个数字充当表达式中第 n 个带括号的组的占位符。在此示例中，\1 被用作第二次写入 (group1) 的替代方式。

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

只需将反向引用替换为所引用的组即可将其删除，如示例所示。

Envoy 代理

代理是为 App Mesh 出售的 Envoy 镜像中的一个进程管理器。代理确保 Envoy 一直保持运行、保持健康并减少停机时间。它会过滤 Envoy 统计数据和辅助数据，以帮助简要了解 Envoy 代理在 App Mesh 中的运行情况。这可以帮助您更快地排除相关错误。

当 Envoy 代理运行状况不佳时，您可以使用代理配置要重启该代理的次数。如果出现故障，代理会在 Envoy 退出时记录最终退出状态。您可以在排除故障时使用它。代理还可以促进 Envoy 连接耗尽，这有助于提高应用程序对故障的弹性。

使用以下变量为 Envoy 配置代理：

- APPNET_ENVOY_RESTART_COUNT — 当此变量设置为非零值时，代理会尝试重新启动 Envoy 代理进程，直到轮询时其认为代理进程状态不正常时设置的数字。在代理运行状况检查失败的情况下，与容器编排工具替换任务或容器组 (pod) 相比，这有助于缩短停机时间。
- PID_POLL_INTERVAL_MS — 配置此变量时，默认值保持为 100。当设置为此值时，与通过容器编排工具运行状况检查替换任务或容器组 (pod) 相比，您可以更快地检测和重启退出 Envoy 进程。
- LISTENER_DRAIN_WAIT_TIME_S — 配置此变量时，请考虑为停止任务或容器组 (pod) 而设置的容器编排工具超时的情况。例如，如果此值大于编排工具超时时间，则 Envoy 代理只能在编排工具强制停止任务或容器组 (pod) 之前的持续时间内耗尽。
- APPNET_AGENT_ADMIN_MODE — 当此变量设置为 tcp 或 uds 时，代理会提供本地管理接口。该管理接口充当与 Envoy 代理交互的安全端点，为运行状况检查、遥测数据提供以下 API，并总结代理的运行状况。
  - GET /status — 查询 Envoy 统计数据并返回服务器信息。
  - POST /drain_listeners — 耗尽所有入站侦听器。
  - POST /enableLogging?level=<desired_level> — 更改所有记录器的 Envoy 日志级别。
  - GET /stats/prometheus — 以 Prometheus 格式显示 Envoy 统计数据。
  - GET /stats/prometheus?usedonly — 仅显示 Envoy 已更新的统计数据。

有关代理配置变量的更多信息，请参阅 Envoy 配置变量。

从版本 1.21.0.0 开始，新 AWS App Mesh 代理包含在 App Mesh 优化的 Envoy 镜像中，无需在客户任务或容器组 (pod) 中分配额外的资源。

App Mesh的可观测性

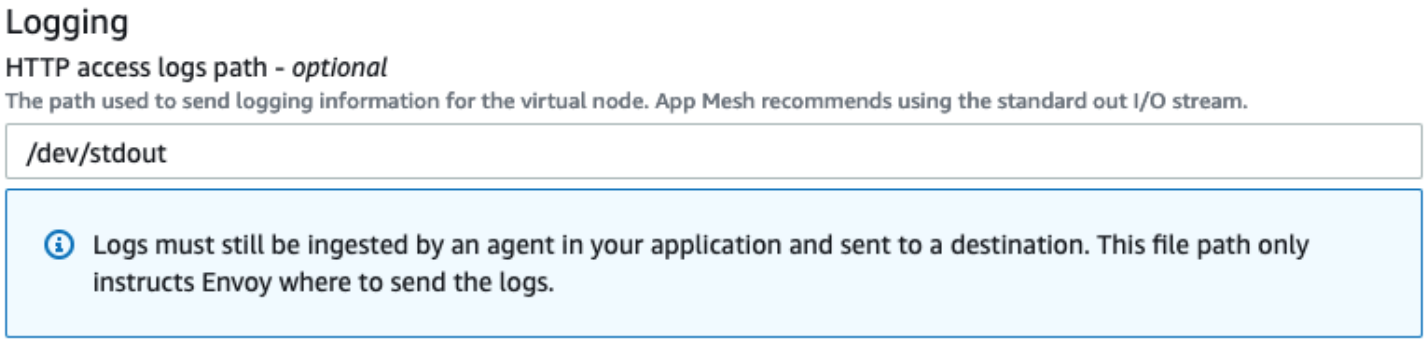
使用 App Mesh 的好处之一是可以更好地了解您的微服务应用程序。App Mesh 能够使用许多不同的日志、指标和跟踪解决方案。

Envoy 代理和 App Mesh 提供以下类型的工具，可帮助您更清楚地了解应用程序和代理：

- 日志记录
- 指标
- 跟踪

日志记录

创建虚拟节点和虚拟网关时，您可以选择配置 Envoy 访问日志。在控制台中，这位于虚拟节点和虚拟网关创建或编辑工作流程的日志记录部分中。



上图显示了 Envoy 访问日志 `/dev/stdout` 的日志记录路径。

对于`format`，请指定两种可能的格式之一，`json` 或 `text`和模式。`json` 获取密钥对并将其转换为 JSON 结构，然后再将其传递给 Envoy。

以下代码块显示了可在 AWS CLI 中使用的 JSON 表示形式。

```
    "logging": {
       "accessLog": {
          "file": {
             "path": "/dev/stdout",
              "format" : {
                  // Exactly one of json or text should be specified
                  "json": [ // json will be implemented with key pairs
                      {
```

```
                        "key": "string",
                        "value": "string"
                    }
                ]
                "text": "string" //e.g. "%LOCAL_REPLY_BODY%:%RESPONSE_CODE%:path=
%REQ(:path)%\n"
            }
        }
    }
}
```

> **Important**
>
> 请务必检查您的输入模式是否对 Envoy 有效，否则 Envoy 将拒绝更新并将最新的更改存储在 `error state` 中。

当您向 `/dev/stdout` 发送 Envoy 访问日志时，它们会与 Envoy 容器日志混在一起。您可以使用标准 Docker 日志驱动程序（例如 `awslogs`）将它们导出到日志存储和处理服务（例如 CloudWatch Logs）。有关更多信息，请参阅《Amazon ES3 开发人员指南》中的使用 awslogs 日志驱动程序。要仅导出 Envoy 访问日志（而忽略其他 Envoy 容器日志），可以 `ENVOY_LOG_LEVEL` 将设置为 `off`。您可以通过包含格式字符串 `%REQ_WITHOUT_QUERY(X?Y):Z%` 来记录不带查询字符串的请求。有关示例，请参阅`ReqWithoutQuery` 格式化程序。有关更多信息，请参阅 Envoy 文档中的访问日志记录。

在 Kubernetes 上启用访问日志

使用适用于 Kubernetes 的 App Mesh Controller 时，您可以通过将日志配置添加到虚拟节点规范中，为虚拟节点配置访问日志，如以下示例所示。

```
---
apiVersion: appmesh.k8s.aws/v1beta2
kind: VirtualNode
metadata:
  name: virtual-node-name
  namespace: namespace
spec:
  listeners:
    - portMapping:
        port: 9080
        protocol: http
```

```
  serviceDiscovery:
    dns:
      hostName: hostname
  logging:
    accessLog:
      file:
        path: "/dev/stdout"
```

您的集群必须有日志转发器才能收集这些日志，例如 Fluentd。有关更多信息，请参阅将 Set up Fluentd 设置为 DaemonSet，以将日志发送到 CloudWatch Logs。

Envoy 还会将各种调试日志从其过滤器写入 `stdout`。这些日志对于深入了解 Envoy 与 App Mesh 的通信以及服务到服务的流量非常有用。可以使用 `ENVOY_LOG_LEVEL` 环境变量配置您的特定日志级别。例如，以下文本来自调试日志示例，该日志显示 Envoy 为特定 HTTP 请求匹配的集群。

```
[debug][router] [source/common/router/router.cc:434] [C4][S17419808847192030829]
 cluster 'cds_ingress_howto-http2-mesh_color_client_http_8080' match for URL '/ping'
```

Firelens 和 Cloudwatch

Firelens 是一款容器日志路由器，您可以用来收集 Amazon ECS 和 AWS Fargate 的日志。您可以在我们的示例存储库中找到使用 Firelens 的AWS示例。

您可以使用 CloudWatch 来收集日志信息和指标。您可以在 App Mesh 文档的导出指标部分找到有关 CloudWatch 的更多信息。

使用 Envoy 指标监控您的应用程序

Envoy 将其指标分为以下主要类别：

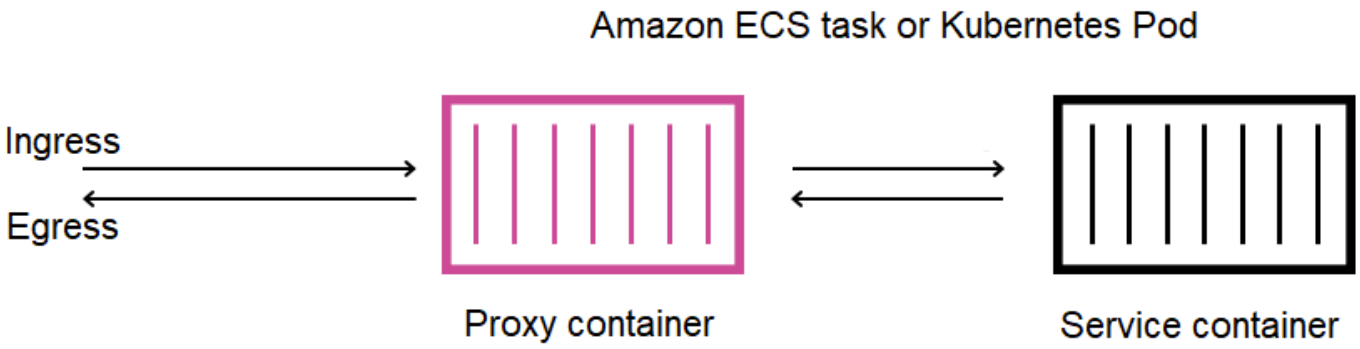
- 下游—与进入代理的连接和请求相关的指标。
- 上游—与传出连接和代理发出的请求相关的指标。
- 服务器—描述 Envoy 内部状态的指标。其中包括诸如正常运行时间或分配内存之类的指标。

在 App Mesh 中，代理会拦截上游和下游流量。例如，Envoy 将从您的客户端收到的请求以及您的服务容器发出的请求归类为下游流量。为了区分这些不同类型的上游和下游流量，App Mesh 根据与您的服务相关的流量方向进一步对 Envoy 指标进行了分类：

- 入口—与流向您的服务容器的连接和请求相关的指标和资源。

- 出口—与来自您的服务容器并最终流出您的 Amazon ECS 任务或 Kubernetes 容器组 (pod) 的连接和请求相关的指标和资源。

下图显示了代理和服务容器之间的通信。



资源命名约定

了解 Envoy 如何查看您的网格以及其资源如何映射回您在 App Mesh 中定义的资源非常有用。以下是 App Mesh 配置的主要 Envoy 资源：

- 侦听器—代理侦听下游连接的地址和端口。在上图中，App Mesh 为进入您的 Amazon ECS 任务或 Kubernetes 容器组 (pod) 的流量创建了一个入口侦听器，并为离开服务容器的流量创建了一个出口侦听器。
- 集群—代理连接和路由流量到的上游端点的命名组。在 App Mesh 中，您的服务容器以及您的服务可以连接到的所有其他虚拟节点都表示为集群。
- 路径—这些路径对应于您在网格中定义的路径。它们包含代理匹配请求所依据的条件以及请求发送到的目标集群。
- 端点和集群负载分配—上游集群的 IP 地址。AWS Cloud Map当使用虚拟节点的服务发现机制时，App Mesh 会将发现的服务实例作为端点资源发送到您的代理。
- 密钥—这些密钥包括但不限于您的加密密钥和 TLS 证书。当 AWS Certificate Manager 用作客户端和服务器证书的来源时，App Mesh 会将公共和私有证书作为秘密资源发送到您的代理。

App Mesh 使用一致的方案来命名 Envoy 资源，您可以使用这些资源与网格相关联。

了解侦听器和集群的命名方案对于理解 Envoy 在 App Mesh 中的指标非常重要。

侦听器名称

使用以下格式命名侦听器：

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

您通常会看到在 Envoy 中配置了以下侦听器：

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

使用 Kubernetes CNI 插件或 IP 表规则，Amazon ECS 任务或 Kubernetes 容器组 (pod) 中的流量将定向到端口 15000 和 15001。App Mesh 将 Envoy 配置为这两个侦听器，以接受入口（传入）和出口（传出）流量。如果您的虚拟节点上没有配置侦听器，则不应看到入口侦听器。

集群名称

大多数集群采用以下格式：

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

您的服务与每个虚拟节点通信的虚拟节点都有自己的集群。如前所述，App Mesh 为在 Envoy 旁边运行的服务创建一个集群，以便代理可以向其发送入口流量。

例如，如果您有一个名为 `my-virtual-node` 的虚拟节点，用于侦听端口 8080 上的 http 流量，并且该虚拟节点位于名为 `my-mesh`的网格中，则 App Mesh 会创建一个名为 `cds_ingress_my-mesh_my-virtual-node_http_8080` 的集群。该集群充当进入 `my-virtual-node` 服务容器的流量的目的地。

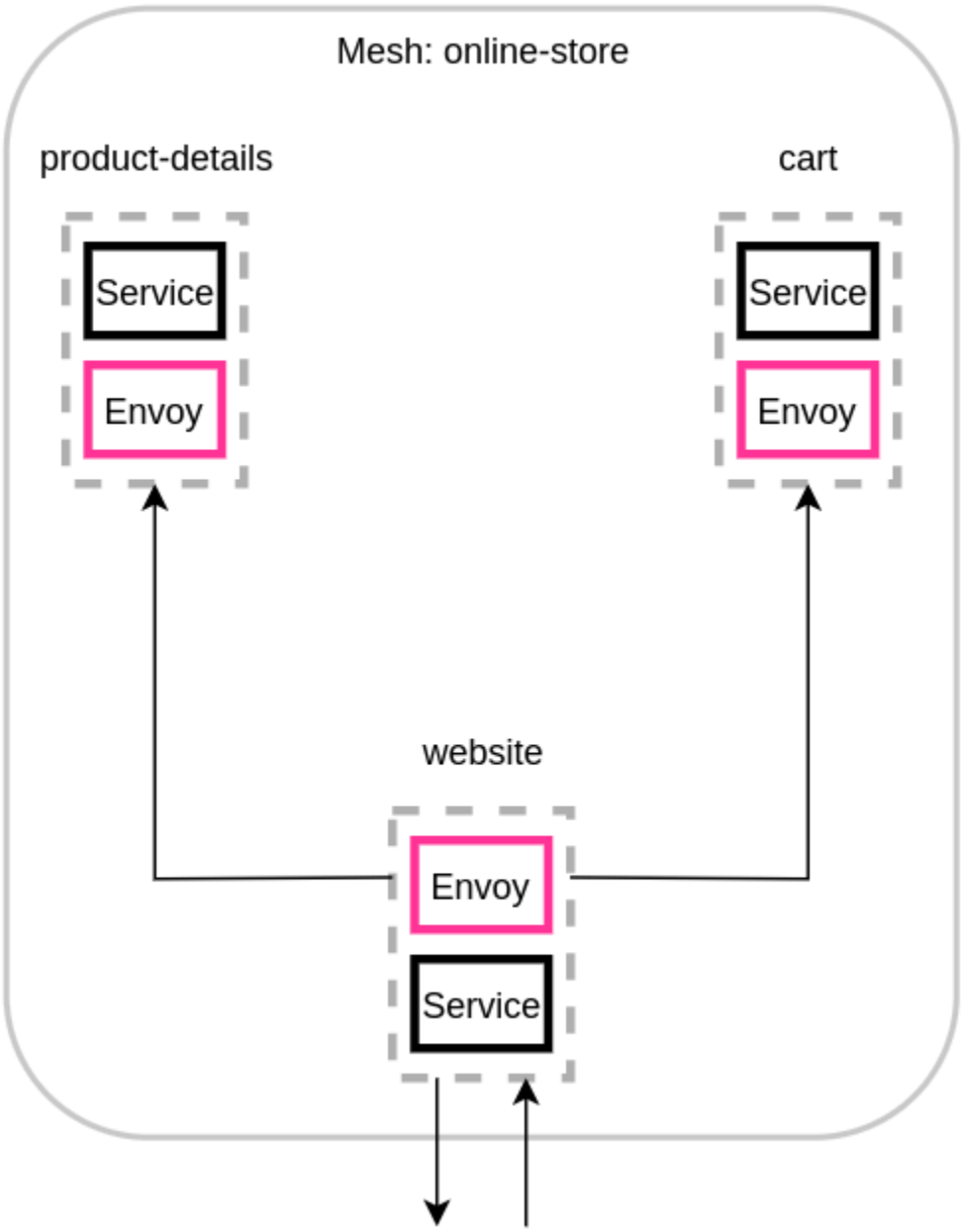
App Mesh 还可以创建以下类型的额外特殊集群。这些其他集群不一定对应于您在网格中明确定义的资源。

- 用于访问其他 AWS 服务的集群。默认情况下，这种类型允许您的网格访问大多数 AWS 服务:`cds_egress_<mesh name>_amazonaws`。
- 用于为虚拟网关执行路由的集群。但一般可以安全地忽略此错误：
  - 对于单个侦听器：`cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<protocol>_<port>`
  - 对于多个侦听器：`cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<ingress_listener_port>_<protocol>_<port>`

- 当您使用 Envoy 密钥发现服务检索密钥时，您可以为其定义端点的集群，例如 TLS：`static_cluster_sds_unix_socket`。

应用程序指标示例

为了说明 Envoy 中可用的指标，以下示例应用程序包含三个虚拟节点。可以忽略网格中的虚拟服务、虚拟路由器和路由，因为它们不会反映在 Envoy 的指标中。在此示例中，所有服务都在端口 8080 上侦听 http 流量。



我们建议将环境变量 ENABLE_ENVOY_STATS_TAGS=1 添加到网格中运行的 Envoy 代理容器中。这可为代理发出的所有指标添加以下指标维度：

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

这些标签设置为网格、虚拟节点或虚拟网关的名称，以允许使用网格中的资源名称筛选指标。

资源名称

网站虚拟节点的代理具有以下资源：

- 两个侦听器，用于侦听入口和出口流量：
  - lds_ingress_0.0.0.0_15000
  - lds_egress_0.0.0.0_15001
- 两个出口集群，代表两个虚拟节点后端：
  - cds_egress_online-store_product-details_http_8080
  - cds_egress_online-store_cart_http_8080
- 网站服务容器的入口集群：
  - cds_ingress_online-store_website_http_8080

侦听器指标示例

- listener.0.0.0.0_15000.downstream_cx_active—与 Envoy 的活跃入口网络连接数。
- listener.0.0.0.0_15001.downstream_cx_active—与 Envoy 的活跃出口网络连接数。您的应用程序与外部服务建立的连接包含在此计数中。
- listener.0.0.0.0_15000.downstream_cx_total—与 Envoy 的入口网络连接总数。
- listener.0.0.0.0_15001.downstream_cx_total—与 Envoy 的出口网络连接总数。

有关全套侦听器指标，请参阅 Envoy 文档中的统计信息。

集群指标示例

- cluster_manager.active_clusters— Envoy 已与之建立至少一个连接的集群总数。

- cluster_manager.warming_clusters— Envoy 尚未连接的集群总数。

以下集群指标使用的格式为 cluster.<cluster name>.<metric name>。这些指标名称是应用程序示例所独有的，由网站 Envoy 容器发出：

- cluster.cds_egress_online-store_product-details_http_8080.upstream_cx_total—网站和产品详情之间的连接总数。
- cluster.cds_egress_online-store_product-details_http_8080.upstream_cx_connect_fail—网站和产品详情之间失败的连接总数。
- cluster.cds_egress_online-store_product-details_http_8080.health_check.failure—网站和产品详情之间运行状况检查失败的总数。
- cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_total—在网站和产品详情之间提出的请求总数。
- cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_time—在网站和产品详情之间提出的请求所花费的时间。
- cluster.cds_egress_online-store_product-details_http_8080.upstream_rq_2xx—网站从产品详情中收到的 HTTP 2xx 响应的数量。

有关完整的 HTTP 指标集，请参阅 Envoy 文档中的统计信息。

管理服务器指标

Envoy 还会发布与其与 App Mesh 控制面板的连接相关的指标，App Mesh 控制面板充当 Envoy 的管理服务器。我们建议您监控其中的一些指标，以便在代理与控制面板长时间不同步时通知您。与控制面板的连接中断或更新失败会使您的代理无法从 App Mesh 接收新配置，包括通过 App Mesh API 进行的网格更改。

- control_plane.connected_state—当代理连接到 App Mesh 时，该指标设置为 1，否则为 0。
- *.update_rejected—Envoy 拒绝的配置更新总数。这通常是由于用户配置错误造成的。例如，如果您将 App Mesh 配置为从 Envoy 无法读取的文件中读取 TLS 证书，则包含该证书路径的更新将被拒绝。
  - 对于已拒绝更新的侦听器，统计数据将为 listener_manager.lds.update_rejected。
  - 对于更新被拒绝的集群，统计数据将为 cluster_manager.cds.update_rejected。

- `*.update_success`—App Mesh 成功更新代理的配置次数。其中包括启用新 Envoy 容器时发送的初始配置有效负载。
  - 如果 Listener 更新成功，则统计数据将为 `listener_manager.lds.update_success`。
  - 要成功更新集群，统计数据将为 `cluster_manager.cds.update_success`。

有关管理服务器指标集，请参阅 Envoy 文档中的管理服务器。

导出指标

Envoy 会发布许多关于其自身操作以及入站和出站流量的各个维度的统计数据。要了解有关 Envoy 统计信息的更多信息，请参阅 Envoy 文档中的统计信息。这些指标可通过代理管理端口上的 `/stats` 端点获得，通常是 `9901`。

`stat` 前缀会有所不同，具体取决于您使用的是单个侦听器还是多个侦听器。以下提供了一些例子来说明其中的区别。

> **Warning**
>
> 如果您将单个侦听器更新为多侦听器功能，由于更新了下表所示的统计前缀，您可能会面临重大变化。
> 我们建议您使用 Envoy 镜像 `1.22.2.1-prod` 或更高版本。这样，您就可以在 Prometheus 端点中看到类似的指标名称。

单侦听器 (SL)/带有“ingress”侦听器前缀的现有统计信息	多侦听器 (ML)/带有“ingress.<protocol>.<port>”侦听器前缀的新统计信息	

`http.*ingress*.rds.rds_ingress_http_5555.version_text`	`http.*ingress.http.5555*.rds.rds_ingress_http_5555.version_text`
	
	`http.*ingress.http.6666*.rds.rds_ingress_http_6666.version_text`	

单侦听器 (SL)/带有“ingress”侦听器前缀的现有统计信息	多侦听器 (ML)/带有“ingress.<protocol>.<port>”侦听器前缀的新统计信息	

`listener.0.0.0.0_15000.http.*ingress*.downstream_rq_2xx`	`listener.0.0.0.0_15000.http.*ingress.http.5555*.downstream_rq_2xx`
	
	`listener.0.0.0.0_15000.http.*ingress.http.6666*.downstream_rq_2xx`	

`http.*ingress*.downstream_cx_length_ms`	`http.*ingress.http.5555*.downstream_cx_length_ms`
	
	`http.*ingress.http.6666*.downstream_cx_length_ms`	

有关统计端点的更多信息，请参阅 Envoy 文档中的统计端点。有关管理界面的更多信息，请参阅 启用 Envoy 代理管理界面。

使用 Amazon EKS 的 App Mesh 的 Prometheus for App Mesh

Prometheus 是一个开源监控和警报工具包。它的一项功能是指定一种发布指标的格式，供其他系统使用。有关 Prometheus 的更多信息，请参阅 Prometheus 文档中的概述。Envoy 可以通过其统计端点传入参数 `/stats?format=prometheus` 来发布其指标。

对于使用 Envoy image build v1.22.2.1-prod 的客户，还有两个额外的维度可以指示入口侦听器的特定统计数据：

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

以下是 Prometheus 现有统计数据与新统计数据的比较。

- 带有“ingress”侦听器前缀的现有统计信息

```
envoy_http_downstream_rq_xx{appmesh_mesh="multiple-listeners-
mesh",appmesh_virtual_node="foodteller-
vn",envoy_response_code_class="2",envoy_http_conn_manager_prefix="ingress"} 931433
```

- 带有“ingress.<protocol>.<port>”的新统计数据 + Appmesh Envoy Image v1.22.2.1-prod 或更高版本

```
envoy_http_downstream_rq_xx{appmesh_mesh="multiple-listeners-
mesh",appmesh_virtual_node="foodteller-
vn",envoy_response_code_class="2",appmesh_listener_protocol="http",appmesh_listener_port="555
 20
```

- 带有“ingress.<protocol>.<port>”的新统计数据 + 自定义 Envoy Imagebuil

```
envoy_http_http_5555_downstream_rq_xx{appmesh_mesh="multiple-listeners-
mesh",appmesh_virtual_node="foodteller-
vn",envoy_response_code_class="2",envoy_http_conn_manager_prefix="ingress"} 15983
```

对于多个侦听器，cds_ingress_<mesh name>_<virtual gateway name>_self_redirect_<ingress_listener_port>_<protocol>_<port> 特殊集群是针对特定侦听器的。

- 带有“ingress”侦听器前缀的现有统计信息

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

- 带有“ingress.<protocol>.<port>”的新统计数据

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

安装 Prometheus

1. 将 EKS 存储库添加到 Helm：

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

2. 安装 App Mesh Prometheus

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

Prometheus 示例

以下是为 Prometheus 永久存储 `PersistentVolumeClaim` 创建的示例。

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

使用 Prometheus 的演练

- 带有 EKS 的 App Mesh — 可观测性：Prometheus

要通过亚马逊 EKS 进一步了解 Prometheus 和 Prometheus

- Prometheus 文档
- EKS — Prometheus 的控制面板指标

App Mesh 的 CloudWatch

从亚马逊 EKS 向 CloudWatch 发送 Envoy 统计数据

您可以将 CloudWatch 代理安装到您的集群中，并将其配置为从代理中收集指标子集。如果您还没有 Amazon EKS 集群，则可以按照 GitHub 上的演练：使用 Amazon EKS 的 App Mesh 中的步骤创建一个集群。您可以按照相同的演练将示例应用程序安装到集群上。

要为您的集群设置适当的 IAM 权限并安装代理，请按照安装带有 Prometheus 指标集合的 CloudWatch 代理中的步骤进行操作。默认安装包含 Prometheus 抓取配置，该配置会提取有用的 Envoy 统计数据子集。有关更多信息，请参阅 App Mesh 的 Prometheus 指标。

要创建配置为显示代理正在收集的指标的 App Mesh 自定义 CloudWatch 控制面板，请按照查看您的 Prometheus 指标教程中的步骤进行操作。当流量进入 App Mesh 应用程序时，您的图表将开始填充相应的指标。

为 CloudWatch 筛选指标

App Mesh 指标扩展提供了一部分有用的指标，可让您深入了解您在网格中定义的资源的行为。由于 CloudWatch 代理支持抓取 Prometheus 指标，因此您可以提供抓取配置来选择要从 Envoy 中提取并发送到 CloudWatch 的指标。

您可以在我们的指标扩展演练中找到使用 Prometheus 抓取指标的示例。

CloudWatch 示例

您可以在我们的 AWS 示例存储库中找到 CloudWatch 的示例配置。

使用 CloudWatch 的演练

- 在我们的 App Mesh 研讨会中添加监控和日志记录功能。
- 带有 EKS 的 App Mesh — 可观测性：CloudWatch
- 在 ECS 上使用 App Mesh 的指标扩展

App Mesh 的指标扩展

Envoy 生成了数百个指标，细分为几个不同的维度。这些指标在与 App Mesh 的关联方面并不简单。就虚拟服务而言，没有机制可以确定哪个虚拟服务正在与给定的虚拟节点或虚拟网关通信。

App Mesh 指标扩展增强了在您的网格中运行的 Envoy 代理。此增强功能允许代理发出其他指标，这些指标可以识别您定义的资源。这小部分额外指标将有助于您更加深入地了解您在 App Mesh 中定义的那些资源的行为。

要启用 App Mesh 指标扩展，请将环境变量 `APPMESH_METRIC_EXTENSION_VERSION` 设置为 `1`。

```
APPMESH_METRIC_EXTENSION_VERSION=1
```

有关 Envoy 环境变量配置的更多信息，请参阅 Envoy 配置变量。

与入站流量相关的指标

- **ActiveConnectionCount**
  - envoy.appmesh.ActiveConnectionCount— 活动的 TCP 连接数。
  - 尺寸 — 网格、虚拟节点、虚拟网关
- **NewConnectionCount**
  - envoy.appmesh.NewConnectionCount— TCP 连接的总数。
  - 尺寸 — 网格、虚拟节点、虚拟网关
- **ProcessedBytes**
  - envoy.appmesh.ProcessedBytes— 发送到下游客户端和从下游客户端接收的 TCP 字节总数。
  - 尺寸 — 网格、虚拟节点、虚拟网关
- **RequestCount**
  - envoy.appmesh.RequestCount — 处理的 HTTP 的请求数。
  - 尺寸 — 网格、虚拟节点、虚拟网关
- **GrpcRequestCount**
  - envoy.appmesh.GrpcRequestCount — 处理的 gPRC 请求数。
  - 尺寸 — 网格、虚拟节点、虚拟网关

与出站流量相关的指标

根据出站指标是来自虚拟节点还是虚拟网关，您将在出站指标上看到不同的维度。

- **TargetProcessedBytes**
  - envoy.appmesh.TargetProcessedBytes— 向 Envoy 上游目标发送和接收的 TCP 字节总数。
  - 维度：
    - 虚拟节点维度 — 网格、虚拟节点、TargetVirtualService、TargetVirtualNode
    - 虚拟网关维度 — 网格、虚拟网关、TargetVirtualService、TargetVirtualNode
- **HTTPCode_Target_2XX_Count**

  - envoy.appmesh.HTTPCode_Target_2XX_Count — 向 Envoy 上游目标发出的导致 2xx HTTP 响应的 HTTP 请求数量。
  - 维度：
    - 虚拟节点维度 — 网格、虚拟节点、TargetVirtualService、TargetVirtualNode
    - 虚拟网关维度 — 网格、虚拟网关、TargetVirtualService、TargetVirtualNode
- **HTTPCode_Target_3XX_Count**
  - envoy.appmesh.HTTPCode_Target_3XX_Count — 向 Envoy 上游目标发出的导致 3xx HTTP 响应的 HTTP 请求数量。
  - 维度：
    - 虚拟节点维度 — 网格、虚拟节点、TargetVirtualService、TargetVirtualNode
    - 虚拟网关维度 — 网格、虚拟网关、TargetVirtualService、TargetVirtualNode
- **HTTPCode_Target_4XX_Count**
  - envoy.appmesh.HTTPCode_Target_4XX_Count — 向 Envoy 上游目标发出的导致 4xx HTTP 响应的 HTTP 请求数量。
  - 维度：
    - 虚拟节点维度 — 网格、虚拟节点、TargetVirtualService、TargetVirtualNode
    - 虚拟网关维度 — 网格、虚拟网关、TargetVirtualService、TargetVirtualNode
- **HTTPCode_Target_5XX_Count**
  - envoy.appmesh.HTTPCode_Target_5XX_Count — 向 Envoy 上游目标发出的导致 5xx HTTP 响应的 HTTP 请求数量。
  - 维度：
    - 虚拟节点维度 — 网格、虚拟节点、TargetVirtualService、TargetVirtualNode
    - 虚拟网关维度 — 网格、虚拟网关、TargetVirtualService、TargetVirtualNode
- **RequestCountPerTarget**
  - envoy.appmesh.RequestCountPerTarget — 发送到 Envoy 上游目标的请求数量。
  - 维度：
    - 虚拟节点维度 — 网格、虚拟节点、TargetVirtualService、TargetVirtualNode
    - 虚拟网关维度 — 网格、虚拟网关、TargetVirtualService、TargetVirtualNode
- **TargetResponseTime**
  - envoy.appmesh.TargetResponseTime — 从向 Envoy 上游的目标发出请求到收到完整响应所经过的时间。

- 维度：
  - 虚拟节点维度 — 网格、虚拟节点、TargetVirtualService、TargetVirtualNode
  - 虚拟网关维度 — 网格、虚拟网关、TargetVirtualService、TargetVirtualNode

App Mesh 的 Datadog

Datadog 是一款监控和安全应用程序，用于云应用程序的端到端监控、指标和日志记录。Datadog 使您的基础架构、应用程序和第三方应用程序完全可被观察。

正在安装 Datadog

- EKS — 要使用 EKS 设置 Datadog，请按照 Datadog 文档中的以下步骤操作。
- ECS EC2 — 要使用 ECS EC2 设置 Datadog，请按照 Datadog 文档中的以下步骤操作。

了解有关 Datadog 的更多信息

- Datadog 文档

跟踪

> **Important**
>
> 为全面实现跟踪，您需要更新应用程序。
> 要查看所选服务中的所有可用数据，您必须使用适用的库来检测您的应用程序。

使用 AWS X-Ray 监控 App Mesh

AWS X-Ray 是一项服务，提供用于查看、筛选和获取您应用程序所服务的请求中收集的数据的工具，并获取数据洞察力。这些见解可帮助您发现问题和机会，从而优化您的应用程序。您可以查看有关请求和响应的详细信息，以及您的应用程序对其他 AWS 服务的下游调用。

X-Ray 与 App Mesh 集成以管理您的 Envoy 微服务。来自 Envoy 的跟踪数据将发送到您的容器中运行的 X-Ray 进程守护程序。

使用具体适用于您的语言的 SDK 指南，在您的应用程序代码中实现 X-Ray。

通过 App Mesh 启用X-Ray 跟踪功能

- 视服务类型而定：
  - ECS — 在 Envoy 代理容器定义中，将 ENABLE_ENVOY_XRAY_TRACING 环境变量设置为 1，将 XRAY_DAEMON_PORT 环境变量设置为 2000。
  - EKS — 在 App Mesh 控制器配置中，包括 `--set tracing.enabled=true` 和 `--set tracing.provider=x-ray`。
- 在 X-Ray 容器中，公开端口 2000 并以用户身份运行 1337。

X-Ray 示例

亚马逊 ECS 的 Envoy 容器定义

```
      {
        "name": "envoy",
        "image": "840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.15.1.0-prod",
        "essential": true,
        "environment": [
          {
            "name": "APPMESH_VIRTUAL_NODE_NAME",
            "value": "mesh/myMesh/virtualNode/myNode"
          },
          {
            "name": "ENABLE_ENVOY_XRAY_TRACING",
            "value": "1"
           }
        ],
        "healthCheck": {
          "command": [
            "CMD-SHELL",
            "curl -s http://localhost:9901/server_info | cut -d' ' -f3 | grep -q live"
            ],
           "startPeriod": 10,
           "interval": 5,
           "timeout": 2,
           "retries": 3
      }
```

更新亚马逊 EKS 的 App Mesh 控制器

```
helm upgrade -i appmesh-controller eks/appmesh-controller \
--namespace appmesh-system \
--set region=${AWS_REGION} \
--set serviceAccount.create=false \
--set serviceAccount.name=appmesh-controller \
--set tracing.enabled=true \
--set tracing.provider=x-ray
```

使用 X-Ray 的演练

- 使用 AWS X-Ray 进行监控
- 带亚马逊 EKS 的 App Mesh — 可观测性：X-Ray
- 在AWS App Mesh研讨会上使用 X-Ray 进行分布式追踪

了解与 AWS X-Ray 相关的更多信息

- AWS X-Ray 文档

使用 App Mesh AWS 对 X-Ray 进行故障排除

- 无法看到我的应用程序的 AWS X-Ray 轨迹。

带 Amazon EKS 的 App Mesh 的 Jaeger 版

Jaeger 是一个开源的、端到端的分布式跟踪系统。它可用于分析网络和进行监控。Jaeger 还可以帮助您排查复杂的云原生应用程序。

要在应用程序代码中实现 Jaeger，您可以在 Jaeger 文档跟踪库中找到具体适用于您的语言的指南。

使用 Helm 安装 Jaeger

1. 将 EKS 存储库添加到 Helm：

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

2. 安装 App Mesh Jaeger

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

Jaeger 示例

以下是为 Jaeger 创建永久存储 `PersistentVolumeClaim` 的示例。

```
helm upgrade -i appmesh-controller eks/appmesh-controller \
--namespace appmesh-system \
--set tracing.enabled=true \
--set tracing.provider=jaeger \
--set tracing.address=appmesh-jaeger.appmesh-system \
--set tracing.port=9411
```

使用 Jaeger 的演练

- 带有 EKS 的 App Mesh — 可观测性：Jaeger

了解有关 Jaeger 的更多信息

- Jaeger 文档

用于跟踪的 Datadog

Datadog 既可以用于跟踪，也可以用于指标。有关更多信息和安装说明，请在 Datadog 文档中找到具体适用于您的应用程序语言的指南。

App Mesh 工具

利用 App Mesh，客户能够使用以下工具间接与其 API 进行交互：

- AWS CloudFormation
- AWS Cloud Development Kit (AWS CDK)
- 适用于 Kubernetes 的 App Mesh 控制器
- Terraform

App Mesh 和 AWS CloudFormation

AWS CloudFormation 是一项服务，可帮助您创建包含应用程序所需的所有资源的模板，然后 AWS CloudFormation 为您配置和配置资源。它还将配置所有依赖关系，因此您可以将更多精力放在应用程序上，而减少对资源管理的关注。

有关与 App Mesh 配合使用的 AWS CloudFormation 更多信息和示例，请参阅AWS CloudFormation文档。

App Mesh 和 AWS CDK

AWS CDK 是一个开发框架，可使用代码来定义您的云基础架构并使用它 AWS CloudFormation 进行配置。 AWS CDK 支持多种编程语言，包括 TypeScript、JavaScript、Python、Java 和 C#/Net。

有关使用带 App Mesh的 AWS CDK 更多信息，请参阅 AWS CDK 文档。

适用于 Kubernetes 的 App Mesh 控制器

适用于 Kubernetes 的 App Mesh 控制器可帮助您管理 Kubernetes 集群的 App Mesh 资源，并将 sidecar 注入容器组 (pod) 中。该控制器专门用于 Amazon EKS，允许您以 Kubernetes 原生的方式管理您的资源。

有关 App Mesh 控制器的更多信息，请参阅 App Mesh 控制器文档。

要查看使用适用于 Kubernetes 的 App Mesh 控制器在 Amazon EKS 上实现 App Mesh 的指南，请查看亚马逊 EKS 研讨会。

App Mesh 和 Terraform

Terraform 是一种开源基础架构即代码软件工具。Terraform 可以使用其 CLI 管理云服务，并使用声明性配置文件与 API 进行交互。

要了解有关在 Terraform 中使用 App Mesh 的更多信息，请查看 Terraform 文档。

使用共享网格。

您可以使用该 AWS Resource Access Manager 服务跨 AWS 账户共享您的 App Mesh 网格。共享网格允许不同 AWS 账户创建的资源在同一个网格中相互通信。

AWS 账户可以是网格资源所有者、网格使用者，或者两者兼而有之。消费者可以在与其账户共享的网格中创建资源。所有者可以在账户拥有的任何网格中创建资源。网格所有者可以与以下类型的网格使用者共享网格。

- 其组织内部或外部的特定 AWS 帐户 AWS Organizations
- 其组织内部的组织单位 AWS Organizations
- 它的整个组织都在 AWS Organizations

有关共享网格的详细介绍，请参阅跨账户网格演练。 end-to-end GitHub

授予共享网格的权限

跨账户共享网格时，共享网格的 IAM 委托人需要权限，网格本身需要资源级权限。

授予共享网格的权限

IAM 委托人需要一组最低权限才能共享网格。我们建议使用`AWSAppMeshFullAccess`和`AWSResourceAccessManagerFullAccess`托管 IAM 策略来确保您的 IAM 委托人拥有共享和使用共享网格所需的权限。

如果您使用自定义 IAM 策略`appmesh:PutMeshPolicy`，则需要`appmesh:GetMeshPolicy`、和`appmesh:DeleteMeshPolicy`操作。这些是仅限许可的 IAM 操作。如果 IAM 委托人未获得这些权限，则在尝试使用该 AWS RAM 服务共享网格时会发生错误。

有关该 AWS Resource Access Manager 服务使用 IAM 的方式的更多信息，请参阅AWS Resource Access Manager 用户指南中的如何 AWS RAM 使用 IAM。

为网格授予权限

共享网格具有以下权限。

- 消费者可以列出和描述与账户共享的网格中的所有资源。
- 所有者可以列出和描述账户拥有的任何网格中的所有资源。

- 所有者和使用者可以修改账户创建的网格中的资源，但他们不能修改其他账户创建的资源。
- 消费者可以删除该账户创建的网格中的任何资源。
- 所有者可以删除网格中任何账户创建的任何资源。
- 所有者的资源只能引用同一账户中的其他资源。例如，虚拟节点只能引用 AWS Cloud Map 或与虚拟节点所有者属于同一账户的 AWS Certificate Manager 证书。
- 所有者和使用者可以将 Envoy 代理作为账户拥有的虚拟节点连接到 App Mesh。
- 所有者可以创建虚拟网关和虚拟网关路由。
- 所有者和消费者可以列出标签，也可以在账户创建的网格中标记/取消标记资源。他们无法在网格中列出不是由账户创建的标签和标记/取消标记资源。

共享网格使用基于策略的授权。使用固定的权限集与网格共享。选择将这些权限添加到资源策略中，也可以根据 IAM 用户/角色选择可选的 IAM 策略。这些策略中允许的权限的交集，减去任何被拒绝的显式权限，决定了主体对网格的访问权限。

共享网格时，该 AWS Resource Access Manager 服务会创建一个名`AWSRAMDefaultPermissionAppMesh`为的托管策略，并将其与提供以下权限的 App Mesh 关联起来。

- `appmesh:CreateVirtualNode`
- `appmesh:CreateVirtualRouter`
- `appmesh:CreateRoute`
- `appmesh:CreateVirtualService`
- `appmesh:UpdateVirtualNode`
- `appmesh:UpdateVirtualRouter`
- `appmesh:UpdateRoute`
- `appmesh:UpdateVirtualService`
- `appmesh:ListVirtualNodes`
- `appmesh:ListVirtualRouters`
- `appmesh:ListRoutes`
- `appmesh:ListVirtualServices`
- `appmesh:DescribeMesh`
- `appmesh:DescribeVirtualNode`
- `appmesh:DescribeVirtualRouter`

- appmesh:DescribeRoute
- appmesh:DescribeVirtualService
- appmesh:DeleteVirtualNode
- appmesh:DeleteVirtualRouter
- appmesh:DeleteRoute
- appmesh:DeleteVirtualService
- appmesh:TagResource
- appmesh:UntagResource

共享网格的先决条件

要共享网格，您必须满足以下先决条件。

- 您必须在自己的 AWS 账户中拥有网格。无法共享已与您共享的网格。
- 要与您的组织或 AWS Organizations内的组织单位共享 AWS Organizations，您必须允许与 共享。有关更多信息，请参阅AWS RAM 《用户指南》中的允许与 AWS Organizations 共享。
- 您的服务必须部署在跨账户共享连接的 Amazon VPC 中，其中包括您要相互通信的网格资源。共享网络连接的一种方法是将您要在网格中使用的所有服务部署到共享子网。有关更多信息和限制，请参阅共享子网。
- 服务必须可通过 DNS 发现或 AWS Cloud Map。有关服务发现的更多信息，请参阅虚拟节点。

相关服务

网格共享与 AWS Resource Access Manager (AWS RAM) 集成。 AWS RAM 是一项服务，可让您与任何 AWS 账户或通过任何账户共享 AWS 资源 AWS Organizations。使用 AWS RAM，您可以通过创建资源共享来共享您拥有的资源。资源共享指定要共享的资源以及与之共享资源的使用者。消费者可以是个人 AWS 帐户、组织单位或整个组织 AWS Organizations。

有关的更多信息 AWS RAM，请参阅《AWS RAM 用户指南》。

共享网格

共享网格可以让不同账户创建的网格资源在同一个网格中相互通信。您只能共享自己拥有的网格。要共享网格，您必须将它添加到资源共享。资源共享是一种 AWS RAM 允许您跨 AWS 账户共享资源的资

源。资源共享指定要共享的资源以及与您共享这些资源的使用者。在使用 Amazon Linux 控制台共享网格时，必须将它添加到现有资源共享。要将网格添加到新的资源共享，使用AWS RAM 控制台创建资源共享。

如果您是组织中的一员， AWS Organizations 并且启用了组织内部共享，则可以自动授予组织中的消费者访问共享网格的权限。否则，使用者会收到加入资源共享的邀请，并在接受邀请后获得对共享网格的访问权限。

您可以使用 AWS RAM 控制台或共享您拥有的网格 AWS CLI。

使用 AWS RAM 控制台共享您拥有的网格

有关说明，请参阅AWS RAM 《用户指南》中的创建资源共享。选择资源类型时，选择网格，然后选择要共享的网格。如果未列出任何网格，请先创建网格。有关更多信息，请参阅 创建服务网格。

要共享您拥有的网格，请使用 AWS CLI

使用 create-resource-share 命令。对于 `--resource-arns` 选项，请指定要共享网格的 ARN。

将已共享的网格取消共享

取消共享网格时，App Mesh 会禁止网格的前使用者进一步访问网格。但是，App Mesh 不会删除使用者创建的资源。取消共享网格后，只有网格所有者才能访问和删除资源。App Mesh 可防止在网格中拥有资源的账户在取消共享网格后接收配置信息。App Mesh 还可以防止任何其他在网格中拥有资源的账户从非共享网格中接收配置信息。只有网格的所有者才能取消共享。

要取消共享您拥有的已共享网格，必须从资源共享中将其删除。您可以使用 AWS RAM 控制台或 AWS CLI.

使用控制台取消共享您拥有的共享网格 AWS RAM

请参阅AWS RAM 《用户指南》中的更新资源共享。

要取消共享您拥有的共享网格，请使用 AWS CLI

使用 disassociate-resource-share 命令。

标识共享的网格

所有者和使用者可以使用 Amazon Linux 控制台识别共享网格和网格资源 AWS CLI

使用 Amazon Linx 控制台识别共享网格

1. 打开 App Mesh 控制台，网址为 https://console.aws.amazon.com/appmesh/。
2. 在左侧导航栏中，选择网格。网格所有者列中列出了每个网格的网格所有者的账户 ID。
3. 从左侧导航栏中，选择虚拟服务、虚拟路由器或虚拟节点。您可以看到网格所有者的账户 ID 和每个资源的资源所有者。

要使用识别共享网格 AWS CLI

使用 `aws appmesh list` *`resource`* 命令，例如 `aws appmesh list-meshes`。
该命令返回您拥有的网格以及与您共享的网格。该`meshOwner`属性显示的 AWS 账户 ID`meshOwner`，`resourceOwner`属性显示资源所有者的 AWS 账户 ID。对任何网格资源运行的任何命令都会返回这些属性。

您附加到共享网格的用户定义标签仅适用于您的 AWS 账户。它们不适用于与之共享网格的其他账户。另一个账户中的网格 `aws appmesh list-tags-for-resource` 命令被拒绝访问。

计费和计量

共享网格不会产生额外费用。

实例配额

网格的所有配额也适用于共享网格，无论谁在网格中创建了资源。只有网格所有者才能申请增加配额。有关更多信息，请参阅 App Mesh 服务限额。该 AWS Resource Access Manager 服务也有配额。有关更多信息，请参阅服务限额。

AWS 与 App Mesh 集成的服务

App Mesh 可与其他 AWS 服务配合使用来提供应对您的业务挑战的更多解决方案。本主题介绍了使用 App Mesh 添加功能的服务或 App Mesh 用于执行任务的服务。

目录

使用 AWS CloudFormation 创建 App Mesh 资源

Amazon ECS 与 AWS CloudFormation 集成，后者是一项服务，可帮助您对 AWS 资源进行建模和设置，这样您只需花较少的时间来创建和管理资源与基础设施。您创建一个描述您所需的所有 AWS 资源（如 App Mesh 集群）的模板，而 AWS CloudFormation 将负责为您设置和配置这些资源。

在您使用 AWS CloudFormation 时，可重复使用您的模板来不断地重复设置您的 App Mesh 资源。仅描述您的资源一次，然后在多个 AWS 账户和区域中反复配置相同的资源。

App Mesh 和 AWS CloudFormation 模板

要为 App Mesh 和相关服务预置和配置资源，您必须了解 AWS CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板描述要在 AWS CloudFormation 堆栈中调配的资源。如果您不熟悉 JSON 或 YAML，可以在 AWS CloudFormation Designer 的帮助下开始使用 AWS CloudFormation 模板。有关更多信息，请参阅 AWS CloudFormation 用户指南中的什么是 AWS CloudFormation Designer？。

App Mesh 支持在 AWS CloudFormation 中创建网格、路由、虚拟节点、虚拟路由器和虚拟服务。有关更多信息（包括 App Mesh 资源的 JSON 和 YAML 模板示例），请参阅《AWS CloudFormation 用户指南》中的 App Mesh 资源类型参考。

了解有关 AWS CloudFormation 的更多信息

要了解有关 AWS CloudFormation 的更多信息，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- AWS CloudFormation 命令行界面用户指南

AWS Outposts 上的 App Mesh

AWS Outposts 在本地设施上启用本机 AWS 服务、基础设施和操作模型。在 AWS Outposts 环境中，您可以使用与 AWS 云中相同的 AWS API、工具和基础设施。AWS Outposts 上的 App Mesh 非常适合需要靠近本地数据和应用程序运行的低延迟工作负载。有关 AWS Outposts 的更多信息，请参阅 AWS Outposts 用户指南。

先决条件

以下是使用 AWS Outposts 上的 App Mesh 的先决条件：

- 您必须已在本地数据中心中安装并配置了 Outpost。
- Outpost 与其 AWS 区域之间必须具有可靠的网络连接。
- Outpost 的 AWS 区域必须支持 AWS App Mesh。有关受支持区域的列表，请参阅AWS 一般参考中的AWS App Mesh 端点和配额。

限制

以下是使用AWS Outpost 上的 App Mesh 的限制：

- AWS Identity and Access Management、应用程序负载均衡器、网络负载均衡器、经典负载均衡器和 Amazon Route 53 在 AWS区域中运行，而不是在 Outpost 上运行。这将增加这些服务和容器之间的延迟。

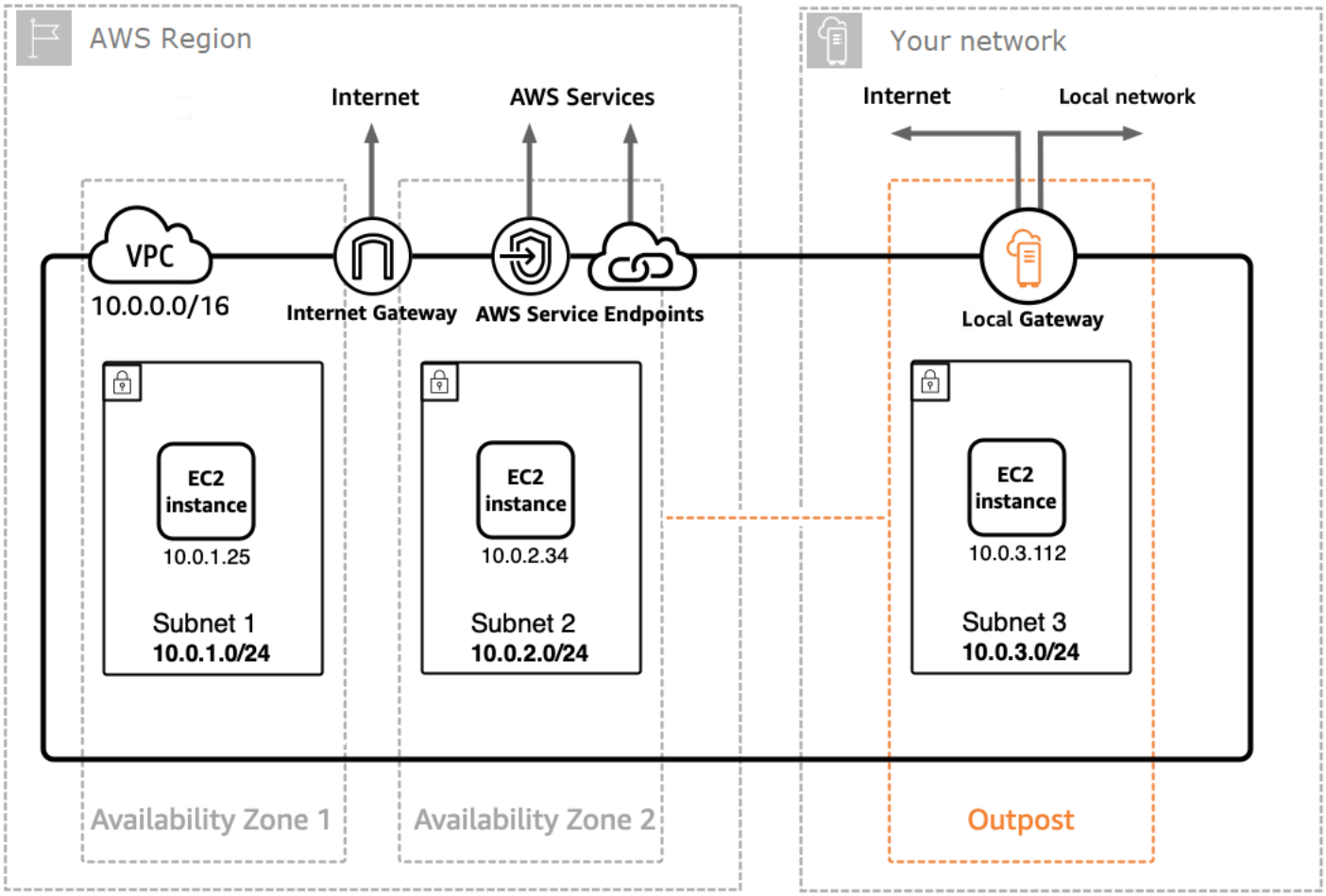
网络连接注意事项

以下是 Amazon EKS AWS Outposts 的网络连接注意事项：

- 如果 Outpost 与其 AWS 区域之间的网络连接丢失，App Mesh Envoy 代理将继续运行。但是，在恢复连接之前，您将无法修改服务网格。
- 建议在 Outpost 与 AWS 区域之间提供可靠、高度可用的低延迟连接。

在 Outpost 上创建 App Mesh Envoy 代理

Outpost 是 AWS 区域的扩展，您可以在账户中扩展 Amazon VPC 以跨越多个可用区和所有关联的 Outpost 位置 。在配置 Outpost 时，您将一个子网与它相关联，以将区域 VPC 环境扩展到本地设施。Outpost 上的实例显示为区域 VPC 的一部分，类似于具有关联子网的可用区。



要在 Outpost 上创建 App Mesh Envoy 代理，请将 App Mesh Envoy 容器镜像添加到亚马逊 ECS 任务或在 Outpost 上运行的 Amazon EKS 容器组 (pod) 中。有关更多信息，请参阅《亚马逊弹性容器服务开发者指南》中的 AWS Outposts 上的亚马逊弹性容器服务，以及亚马逊 EKS 用户指南中的 Outposts 上的亚马逊 Elastic Kubernetes AWS 服务。

App Mesh 最佳实践

为了实现计划部署期间和某些主机计划外丢失期间零失败请求的目标，本主题中的最佳实践实施以下策略：

- 从应用程序的角度来看，使用安全的默认重试策略可以增加请求成功的可能性。有关更多信息，请参阅 通过重试来检测所有路由。
- 通过最大限度地提高重试请求发送到实际目标的可能性，增加重试请求成功的可能性。有关更多信息，请参阅 调整部署速度、在横向缩减之前先进行横向扩展 和 实施容器运行状况检查。

为显著减少或消除故障，我们建议您在以下所有实践中实施这些建议。

通过重试来检测所有路由

将所有虚拟服务配置为使用虚拟路由器，并为所有路由设置默认重试策略。这可通过重新选择主机并发送新请求来缓解请求失败的情况。对于重试策略，我们建议在支持重试事件类型的每种路径类型中为 `maxRetries` 指定至少两个值，并为每种重试事件指定以下选项：

- TCP — `connection-error`
- HTTP 和 HTTP/2 — `stream-error` 以及 `gateway-error`
- gRPC — `cancelled` 以及 `unavailable`

其他重试事件可能不安全，需要根据具体情况考虑这些事件，例如：请求不具有幂等性。您需要考虑并测试 `maxRetries` 和 `perRetryTimeout` 的值，这些值获得请求延迟最大值(`maxRetries` * `perRetryTimeout`)与提高更多重试的成功率之间进行适当的权衡。此外，当 Envoy 尝试连接到已不存在的端点时，您应该预计该请求会消耗全部内容 `perRetryTimeout`。若想配置重试策略，请参阅 创建路由，然后选择要进行路由的协议。

> **Note**
>
> 如果您在 2020 年 7 月 29 日当天或之后实施了路由，但没有指定重试策略，那么 App Mesh 可能会自动为您在 2020 年 7 月 29 日当天或之后创建的每条路径创建与之前策略相似的默认重试策略。有关更多信息，请参阅 默认路由重试策略。

调整部署速度

使用滚动部署时，请降低总体部署速度。默认情况下，Amazon ECS 配置的部署策略至少针对 100% 的正常任务和 200% 的总任务。部署后，会导致两个高漂移点产生：

- 在准备完成请求之前，Envoy 可以看到新任务 100% 规模的实例集（有关缓解措施，请参阅 实施容器运行状况检查）。
- 任务终止期间，Envoy 可以看到旧任务 100% 规模的实例集。

配置这些部署限制后，容器编排工具可能会进入一种状态，它们会同时隐藏所有原目标，同时显示所有新目标。由于您的 Envoy 数据面板最终是一致的，这可能会导致数据面板中可见的目标集与编排工具的角度有所不同。为缓解这种情况，我们建议将正常运行的任务比例保持在至少 100%，但将总任务比例降至 125%。这可以减少分歧，提高重试的可靠性。我们建议针对不同的容器运行时系统使用以下设置。

Amazon ECS

如果您的服务所需计数为两个或三个，请将 `maximumPercent` 设置为 150%。否则，请将 `maximumPercent` 设置为 125%。

Kubernetes

配置您的部署 `update strategy`，设置 `maxUnavailable` 为 0% 和 `maxSurge` 25%。有关部署的更多信息，请参阅 Kubernetes 文档中的部署。

在横向缩减之前先进行横向扩展

横向扩展和横向缩减均可能导致重试时请求失败。虽然有些任务建议可以缓解横向扩展情况，但横向缩减的唯一建议是最大限度地降低在任何时候横向缩减任务的百分比。我们建议您使用部署策略，在横向扩展旧任务或部署之前，先扩展新的 Amazon ECS 任务或 Kubernetes 部署。这种扩展策略可以降低横向缩减任务或部署的缩放百分比，同时保持相同的速度。这种做法适用于亚马逊 ECS 任务和 Kubernetes 部署。

实施容器运行状况检查

在纵向扩展场景中，Amazon ECS 任务中的容器可能会出现故障，且最初可能没有响应。对于不同的容器运行时系统，我们推荐以下建议：

Amazon ECS

为缓解这种情况，我们建议使用容器运行状况检查和容器依赖关系排序，以确保 Envoy 在任何需要出站网络连接的容器启动之前运行良好。要在任务定义中正确配置应用程序容器和 Envoy 容器，请参阅容器依赖关系。

Kubernetes

没有，因为在 Kubernetes 的 App Mesh 控制器中注册和注销实例时，没有考虑 Kubernetes 的AWS Cloud Map存活性和就绪性探头。有关更多信息，请参阅 GitHub 问题 #132。

安全性 AWS App Mesh

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为 AWS 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于 AWS App Mesh的合规性计划，请参阅合规性计划范围内的AWS 服务。App Mesh 负责将配置安全地传送到本地代理，包括 TLS 证书私钥等密钥。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还应对其他因素负责，包括：
  - 您的数据的敏感性、贵公司的要求以及适用的法律和法规。
  - App Mesh 数据面板的安全配置，包括配置安全组以允许流量在您的 VPC 内传入服务。
  - 与 App Mesh 关联的计算资源的配置。
  - 与您的计算资源关联的 IAM 策略以及允许它们从 App Mesh 控制面板检索哪些配置。

此文档将帮助您了解如何在使用 App Mesh 时应用责任共担模式。以下主题说明如何配置 App Mesh 以实现您的安全性和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 App Mesh 资源。

App Mesh 安全原则

> 客户应该能够根据需要调整安全性。平台不应阻止他们提高安全性。默认情况下，平台功能是安全的。

主题

- AWS App Mesh 中的故障恢复能力
- 中的配置和漏洞分析 AWS App Mesh

传输层安全性协议 (TLS)

在 App Mesh 中，传输层安全性协议 (TLS) 对部署在计算资源上的 Envoy 代理之间的通信进行加密，这些资源在 App Mesh 中由网格端点表示，例如 虚拟节点 和 虚拟网关。代理协商并终止 TLS。当代理与应用程序一起部署时，您的应用程序代码不负责协商 TLS 会话。代理代表您的应用程序协商 TLS。

App Mesh 允许您通过以下方式向代理提供 TLS 证书：

- 来自 AWS Certificate Manager (ACM) 的私有证书，由 AWS Private Certificate Authority (AWS Private CA) 颁发。
- 存储在虚拟节点本地文件系统中的证书，由您自己的证书颁发机构 (CA) 颁发
- 由密钥发现服务 (SDS) 端点通过本地 Unix 域套接字提供的证书。

Envoy Proxy 授权 必须为由网格端点表示的已部署的 Envoy 代理启用。我们建议您在启用代理授权时，将访问限制为仅访问您启用加密的网状端点。

证书要求

证书上的主题备用名称 (SAN) 之一必须符合特定标准，具体取决于网状端点所代表的实际服务的发现方式。

- DNS — 其中一个证书 SAN 必须与 DNS 服务发现设置中提供的值相匹配。对于具有服务发现名称的应用程序 *mesh-endpoint.apps.local*，您可以创建与该名称匹配的证书，也可以创建带有通配符的证书 *.*apps.local*。
- AWS Cloud Map— 其中一个证书 SAN 必须与 AWS Cloud Map 服务发现设置中提供的格式相匹配*service-name.namespace-name*。对于 AWS Cloud Map 服务发现设置为 Service *apps.local* Name 和 Namespac *mesh-endpoint* eName 的应用程序，您可以创建与*mesh-endpoint.apps.local*名称匹配的证书，也可以创建带有通配符的证书 *.*apps.local*.

对于这两种发现机制，如果没有一个证书 SAN 与 DNS 服务发现设置匹配，则 Envoy 之间的连接将失败，并显示以下错误消息，如客户端 Envoy 所示。

```
TLS error: 268435581:SSL routines:OPENSSL_internal:CERTIFICATE_VERIFY_FAILED
```

TLS 身份验证证书

使用 TLS 身份验证时，App Mesh 支持多种证书来源。

AWS Private CA

证书必须存储在 ACM 中，其区域和 AWS 账户必须与使用该证书的网格端点相同。CA 的证书不必位于同一个 AWS 账户中，但它仍然需要与网格端点位于同一个区域。如果您没有 AWS 私有 CA，则必须先创建一个，然后才能向其申请证书。有关 AWS Private CA 使用 ACM 向现有证书申请证书的更多信息，请参阅申请私有证书。该证书不能是公共证书。

用于 TLS 客户端策略的私有 CA 必须是根用户 CA。

要使用来自的证书和 CA 配置虚拟节点 AWS Private CA，用于调用 App Mesh 的委托人（例如用户或角色）必须具有以下 IAM 权限：

- 对于添加到侦听器的 TLS 配置中的任何证书，主体必须具有 `acm:DescribeCertificate` 权限。
- 对于在 TLS 客户端策略上配置的任何 CA，主体必须拥有 `acm-pca:DescribeCertificateAuthority` 权限。

> **Important**
>
> 与其他账户共享 CA 可能会给这些账户授予 CA 意想不到的权限。我们建议使用基于资源的策略来限制访问无需从 CA 颁发证书的账户的 `acm-pca:DescribeCertificateAuthority` 和 `acm-pca:GetCertificateAuthorityCertificate`。

您可以将这些权限添加到附加到主体的现有 IAM 策略中，也可以创建新的主体和策略并将该策略附加到主体。有关更多信息，请参阅编辑 IAM 策略、创建 IAM 策略和添加 IAM 身份权限。

> **Note**
>
> 在将其删除 AWS Private CA 之前，您需要为每个操作支付月费。您还需要为每月颁发的私有证书和导出的私有证书付费。有关更多信息，请参阅AWS Certificate Manager 定价。

当您为网状端点所代表的 Envoy 代理启用代理授权时，必须为您使用的 IAM 角色分配以下 IAM 权限：

- 对于在虚拟节点的侦听器上配置的任何证书，该角色必须具有 `acm:ExportCertificate` 权限。
- 对于在 TLS 客户端策略上配置的任何 CA，该角色都必须具有 `acm-pca:GetCertificateAuthorityCertificate` 权限。

文件系统

您可以使用文件系统向 Envoy 分发证书。您可以通过在文件路径上提供证书链和相应的私钥来实现此目的。这样，就可以通过 Envoy sidecar 代理访问这些资源。

Envoy 密钥发现服务 (SDS)

Envoy 通过密钥发现协议从特定端点获取 TLS 证书等密钥。有关此协议的更多信息，请参阅 Envoy 的 SDS 文档。

当 SDS 作为证书和证书链的来源时，App Mesh 将 Envoy 代理配置为使用代理本地的 Unix 域套接字作为密钥发现服务 (SDS) 端点。您可以使用 `APPMESH_SDS_SOCKET_PATH` 环境变量配置此端点的路径。

> **Important**
>
> App Mesh Envoy 代理版本 1.15.1.0 及更高版本支持使用 Unix 域套接字的本地密钥发现服务。
>
> App Mesh 支持使用 gRPC 的 V2 SDS 协议。

与 SPIFFE 运行时系统环境 (SPIRE) 集成

您可以使用 SDS API 的任何 sidecar 实现，包括 SPIFFE 运行时环境 (SPIRE) 等现有工具链。SPIRE 旨在支持在分布式系统中的多个工作负载之间部署双向 TLS 身份验证。它在运行时证明了运行时系统的身份。SPIRE 还为工作负载提供适用于特定工作负载的、短期的、自动轮换的密钥和证书。

您应该将 SPIRE 代理配置为 Envoy 的 SDS 提供商。允许它直接向 Envoy 提供提供双向 TLS 身份验证所需的密钥材料。在 Envoy 代理旁的 sidecar 中运行 SPIRE 代理。代理负责根据需要重新生成短期密钥和证书。代理会证明 Envoy，并确定当 Envoy 连接到 SPIRE 代理暴露的 SDS 服务器时，它应该向 Envoy 提供哪些服务身份和 CA 证书。

在此过程中，将轮换服务身份和 CA 证书，并将更新流式传输回 Envoy。Envoy 会立即将它们应用于新连接，不会出现任何中断或停机，也不会让私钥接触文件系统。

App Mesh 如何配置 Envoy 以协商 TLS

在确定如何配置网格中 Envoy 之间的通信时，App Mesh 使用客户端和服务器的网格端点配置。

使用客户端策略

当客户端策略强制使用 TLS，并且客户端策略中的一个端口与服务器策略的端口匹配时，该客户端策略用于配置客户端的 TLS 验证上下文。例如，如果虚拟网关的客户端策略与虚拟节点的服务器策略相匹配，则将使用虚拟网关的客户端策略中定义的设置在代理之间尝试 TLS 协商。如果客户端策略与服务器策略的端口不匹配，则代理之间的 TLS 可能会协商，也可能不协商，具体取决于服务器策略的 TLS 设置。

没有客户端策略

如果客户端尚未配置客户端策略，或者客户端策略与服务器的端口不匹配，App Mesh 将使用服务器来确定是否与客户端协商 TLS 以及如何协商。例如，如果虚拟网关未指定客户端策略，并且虚拟节点未配置 TLS 终止，则代理之间将不会协商 TLS。如果客户端未指定匹配的客户端策略，并且服务器已配置为 TLS 模式 `STRICT` 或 `PERMISSIVE`，则代理将被配置为协商 TLS。根据为 TLS 终止提供证书的方式，将适用以下其他行为。

- ACM 管理的 TLS 证书 — 当服务器使用 ACM 管理的证书配置 TLS 终止时，App Mesh 会自动将客户端配置为协商 TLS，并根据证书链接到的根用户 CA 验证证书。
- 基于文件的 TLS 证书 — 当服务器使用来自代理本地文件系统的证书配置 TLS 终止时，App Mesh 会自动将客户端配置为协商 TLS，但服务器的证书未经过验证。

主题备用名称

您可以选择指定要信任的主题备用名称 (SAN) 列表。SAN 必须采用 FQDN 或 URI 格式。如果提供了 SAN，Envoy 会验证所提供证书的主题备用名称是否与此列表中的一个名称匹配。

如果您没有在终止网格端点上指定 SAN，则该节点的 Envoy 代理不会验证对等客户端证书上的 SAN。如果您没有在起源网格端点上指定 SAN，则终止端点提供的证书上的 SAN 必须与网格端点服务发现配置匹配。

有关更多信息，请参阅 App Mesh TLS：证书要求。

> **Important**
>
> 只有将 TLS 的客户端策略设置为 `not enforced`，您才能使用通配符 SAN。如果将客户端虚拟节点或虚拟网关的客户端策略配置为强制执行 TLS，则它不能接受通配符 SAN。

验证加密

启用 TLS 后，您可以查询 Envoy 代理以确认通信已加密。Envoy 代理会发出有关资源的统计信息，这些统计数据可以帮助您了解 TLS 通信是否正常运行。例如，Envoy 代理记录其为指定网格端点协商的成功的 TLS 握手次数的统计信息。使用以下命令确定命名的网格端点成功的 TLS 握 *my-mesh-endpoint* 手次数。

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

在以下示例返回的输出中，网格端点进行了三次握手，因此通信已加密。

```
listener.0.0.0.0_15000.ssl.handshake: 3
```

当 TLS 协商失败时，Envoy 代理还会发出统计信息。确定网格端点是否存在 TLS 错误。

```
curl -s 'http://my-mesh-endpoint.apps.local:9901/stats' | grep -e "ssl.*\(fail\|error
\)"
```

在返回的输出示例中，多个统计数据没有错误，因此 TLS 协商成功。

```
listener.0.0.0.0_15000.ssl.connection_error: 0
listener.0.0.0.0_15000.ssl.fail_verify_cert_hash: 0
listener.0.0.0.0_15000.ssl.fail_verify_error: 0
listener.0.0.0.0_15000.ssl.fail_verify_no_cert: 0
listener.0.0.0.0_15000.ssl.ssl.fail_verify_san: 0
```

有关 Envoy TLS 统计信息的更多信息，请参阅 Envoy 侦听器统计信息。

证书续订

AWS Private CA

当您使用 ACM 续订证书时，续订的证书将在续订完成后的 35 分钟内自动分发给您连接的代理。我们建议使用托管续订来自动续订即将到期的证书。有关更多信息，请参阅《用户指南》中的 ACM 亚马逊颁发的证书的托管续订。 AWS Certificate Manager

您自己的证书

使用来自本地文件系统的证书时，Envoy 不会在证书更改时自动重新加载证书。您可以重新启动或重新部署 Envoy 进程来加载新证书。您还可以将较新的证书放在不同的文件路径中，并使用该文件路径更新虚拟节点或网关配置。

将 Amazon ECS 工作负载配置为使用 TLS 身份验证 AWS App Mesh

您可以将网格配置为使用 TLS 身份验证。确保证书可供您添加到工作负载中的 Envoy 代理 sidecar 使用。您可以将 EBS 或 EFS 卷附加到您的 Envoy sidecar，也可以从 AWS Secrets Manager 存储和检索证书。

- 如果您使用基于文件的证书分发，请将 EBS 或 EFS 卷附加到您的 Envoy sidecar。确保证书和私钥的路径与中配置的路径相匹配 AWS App Mesh。
- 如果您使用的是基于 SDS 的发行版，请添加一个实现 Envoy 的 SDS API 并访问证书的 sidecar。

> **Note**
>
> 亚马逊 ECS 不支持 SPIRE。

将 Kubernetes 工作负载配置为使用 TLS 身份验证 AWS App Mesh

您可以将 Kubernetes AWS App Mesh 控制器配置为为虚拟节点和虚拟网关服务后端和侦听器启用 TLS 身份验证。确保证书可供您添加到工作负载中的 Envoy 代理 sidecar 使用。您可以在双向 TLS 身份验证的演练部分中查看每种分发类型的示例。

- 如果您使用基于文件的证书分发，请将 EBS 或 EFS 卷附加到您的 Envoy sidecar。确保证书和私钥的路径与控制器中配置的路径相匹配。或者，您可以使用挂载在文件系统上的 Kubernetes 密钥。
- 如果您使用的是基于 SDS 的发行版，您应该设置一个实现 Envoy 的 SDS API 的节点本地 SDS 提供商。Envoy 将通过 UDS 找到它。要在 EKS AppMesh 控制器中启用基于 SDS 的 mTLS 支持，请将该`enable-sds`标志设置为，`true`并通过该标志提供本地 SDS 提供商到控制器的 UDS 路径。`sds-uds-path`如果您使用 helm，则可以在控制器安装过程中设置以下内容：

```
--set sds.enabled=true
```

> **Note**
>
> 如果您在 Fargate 模式下使用 Amazon Elastic Kubernetes Service (Amazon EKS)，则无法使用 SPIRE 分发证书。

双向 TLS 身份验证

双向 TLS（传输层安全）身份验证是 TLS 的可选组件，可提供双向对等身份验证。双向 TLS 身份验证在 TLS 的基础上增加了一层安全性，并允许您的服务验证正在建立连接的客户端。

客户端-服务器关系中的客户端还会在会话协商过程中提供 X.509 证书。服务器使用此证书来识别和验证客户端。此过程有助于验证证书是否由可信证书颁发机构 (CA) 颁发，以及该证书是否为有效证书。它还使用证书上的主题备用名称 (SAN) 来识别客户端。

您可以为支持的所有协议启用双向 TLS 身份验证 AWS App Mesh。它们是 TCP、HTTP/1.1、HTTP/2、gRPC。

> **Note**
>
> 使用 App Mesh，您可以为来自您的服务的 Envoy 代理之间的通信配置双向 TLS 身份验证。但是，您的应用程序与 Envoy 代理之间的通信未加密。

双向 TLS 身份验证证书

AWS App Mesh 支持双向 TLS 身份验证的两种可能的证书来源。TLS 客户端策略中的客户端证书和侦听器 TLS 配置中的服务器验证可以从以下来源获取：

- 文件系统 - 来自正在运行的 Envoy 代理的本地文件系统的证书。要向 Envoy 分发证书，您需要提供证书链的文件路径和 App Mesh API 的私钥。
- E@@ nvoy 的秘密发现服务 (SDS) — 实现 SDS 并允许向 Envoy 发送证书的 B s ring-your-own idecar。它们包括 SPIFFE 运行时系统环境 (SPIRE)。

> **Important**
>
> App Mesh 不存储用于双向 TLS 身份验证的证书或私钥。相反，Envoy 会将它们存储在内存中。

配置网格端点

为您的网格端点（例如虚拟节点或网关）配置双向 TLS 身份验证。这些端点提供证书并指定受信任的颁发机构。

为此，您需要为客户端和服务器配置 X.509 证书，并在 TLS 终止和 TLS 来源的验证上下文中明确定义可信机构证书。

网格内部的信任

服务器端证书在虚拟节点侦听器（TLS 终止）中配置，客户端证书在虚拟节点服务后端（TLS 发起）中配置。作为此配置的替代方案，您可以为虚拟节点的所有服务后端定义默认的客户端策略，然后，如果需要，可以根据需要为特定后端覆盖此策略。虚拟网关只能使用适用于其所有后端的默认客户端策略进行配置。

您可以通过为两个网格的虚拟网关上的入站流量启用双向 TLS 身份验证来配置不同网格之间的信任。

网格之外的信任

在虚拟网关侦听器中指定服务器端证书以终止 TLS。配置与您的虚拟网关通信的外部服务以提供客户端证书。证书应派生自服务器端证书在虚拟网关侦听器上用于发起 TLS 的相同证书颁发机构 (CA)。

将服务迁移到双向 TLS 身份验证

在 App Mesh 中将现有服务迁移到双向 TLS 身份验证时，请遵循以下准则以保持连接。

迁移服务通过明文进行通信

1. 在服务器端点上启用 TLS 配置 `PERMISSIVE` 模式。此模式允许纯文本流量连接到端点。
2. 在服务器上配置双向 TLS 身份验证，指定服务器证书、信任链以及可选的可信 SAN。
3. 确认通信是通过 TLS 连接进行的。
4. 在您的客户端上配置双向 TLS 身份验证，指定客户端证书、信任链以及可选的可信 SAN。
5. 在服务器上启用 TLS 配置的 `STRICT` 模式。

迁移通过 TLS 通信的服务

1. 在您的客户端上配置双向 TLS 设置，指定客户端证书和可信的 SAN（可选）。直到后端服务器请求客户端证书后，才会将其发送到其后端。
2. 在服务器上配置双向 TLS 设置，指定信任链和可信的 SAN（可选）。为此，您的服务器会请求客户端证书。

验证双向 TLS 身份验证

您可以参阅传输层安全：验证加密文档，了解 Envoy 究竟是如何发出与 TLS 相关的统计数据的。对于双向 TLS 身份验证，您应该检查以下统计信息：

- `ssl.handshake`
- `ssl.no_certificate`
- `ssl.fail_verify_no_cert`
- `ssl.fail_verify_san`

以下两个统计示例共同表明，成功端接到虚拟节点的 TLS 连接都来自提供证书的客户端。

```
listener.0.0.0.0_15000.ssl.handshake: 3
```

```
listener.0.0.0.0_15000.ssl.no_certificate: 0
```

下一个统计数据示例显示，从虚拟客户端节点（或网关）到后端虚拟节点的连接失败。服务器证书中显示的主题备用名称 (SAN) 与客户端信任的任何 SAN 都不匹配。

```
cluster.cds_egress_my-mesh_my-backend-node_http_9080.ssl.fail_verify_san: 5
```

App Mesh 双向 TLS 身份验证演练

- 双向 TLS 身份验证演练：本演练描述了如何使用 App Mesh CLI 构建具有双向 TLS 身份验证的彩色应用程序。
- Amazon EKS 基于双向 TLS SDS 的演练：本演练展示了如何在 Amazon EKS 和 SPIFFE 运行时系统环境 (SPIRE) 中使用基于双向 TLS SDS 的身份验证。
- 亚马逊 EKS 基于双向 TLS 文件的演练：本演练展示了如何在 Amazon EKS 和 SPIFFE 运行时系统环境 (SPIRE) 中使用基于双向 TLS 文件的身份验证。

如何 AWS App Mesh 与 IAM 配合使用

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制可以通过身份验证（登录）和授权（具有权限）使用 App Mesh 资源的人员。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 App Mesh 中所做的工作。

服务用户 – 如果使用 App Mesh 服务来完成任务，则您的管理员会为您提供所需的凭证和权限。当您使用更多 App Mesh 功能来完成工作时，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 App Mesh 中的功能，请参阅 对 AWS App Mesh 身份和访问进行故障排除。

服务管理员 – 如果您在公司负责管理 App Mesh 资源，则您可能具有 App Mesh 的完全访问权限。您有责任确定您的服务用户应访问哪些 App Mesh 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 App Mesh 搭配使用的更多信息，请参阅 如何 AWS App Mesh 与 IAM 配合使用。

IAM 管理员 – 如果您是 IAM 管理员，您可能希望了解有关如何编写策略以管理对 App Mesh 的访问权限的详细信息。要查看您可在 IAM 中使用的 App Mesh 基于身份的策略示例，请参阅 AWS App Mesh 基于身份的策略示例。

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。

当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以根用户身份登录的任务的完整列表，请参阅 IAM 用户指南 中的 需要根用户凭证的任务。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅 IAM 用户指南中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或

AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的 为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅 IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅 IAM 用户指南中的创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Simple Storage Service（Amazon S3）存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅 Amazon Simple Storage Service 开发人员指南 中的访问控制列表（ACL）概览。

其它策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界 - 权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅 IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有特征，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅 AWS Organizations 用户指南中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

如何 AWS App Mesh 与 IAM 配合使用

在使用 IAM 管理对 App Mesh 的访问之前，您应了解哪些 IAM 功能可与 App Mesh 结合使用。要全面了解 App Mesh 和其他 AWS 服务如何与 IAM 配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的 AWS 服务。

主题

App Mesh 基于身份的策略

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。App Mesh 支持特定的操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》 中的 IAM JSON 策略元素参考。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源 执行操作，以及在什么 条件 下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

App Mesh 中的策略操作在操作前使用以下前缀：`appmesh:`。例如，要授予某人使用 `appmesh:ListMeshes` API 操作列出网格的权限，您应将 `appmesh:ListMeshes` 操作纳入他们的策略中。策略语句必须包含 `Action` 或 `NotAction` 元素。

要在单个语句中指定多项 操作，请使用逗号将它们隔开，如下所示。

```
"Action": [
      "appmesh:ListMeshes",
      "appmesh:ListVirtualNodes"
]
```

您也可以使用通配符 (*) 指定多个操作。例如，要指定以单词 `Describe` 开头的所有操作，请包括以下操作。

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

要查看 App Mesh 操作列表，请参阅 《IAM 用户指南》中 AWS App Mesh 定义的操作。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源 执行操作，以及在什么条件 下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

App Mesh mesh 资源拥有以下 ARN：

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

有关 ARN 格式的更多信息，请参阅 A mazon 资源名称 (ARN) 和 AWS 服务命名空间。

例如，要在语句中指定 *Region-code* 区域中名为 *app* 的网格，请使用以下 ARN：

```
arn:aws:appmesh:Region-code:111122223333:mesh/apps
```

要指定属于特定账户的所有实例，请使用通配符 (*)。

```
"Resource": "arn:aws:appmesh:Region-code:111122223333:mesh/*"
```

无法对特定资源执行某些 App Mesh 操作，例如，用于创建资源的操作。在这些情况下，您必须使用通配符（*）。

```
"Resource": "*"
```

许多 App Mesh API 操作涉及多种资源。例如,CreateRoute 创建具有虚拟节点目标的路由，因此 IAM 用户必须有权使用该路由和虚拟节点。要在单个语句中指定多个资源，请使用逗号分隔 ARN。

```
"Resource": [
      "arn:aws:appmesh:Region-code:111122223333:mesh/apps/virtualRouter/serviceB/route/
*",
      "arn:aws:appmesh:Region-code:111122223333:mesh/apps/virtualNode/serviceB"
]
```

要查看 App Mesh 资源类型及其 ARN 的列表，请参阅《IAM 用户指南》中 AWS App Mesh定义的资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 AWS App Mesh定义的操作。

条件键

App Mesh 支持使用某些全局条件键。要查看所有 AWS 全局条件键，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。要查看 App Mesh 支持的全局条件键的列表，请参阅 IAM 用户指南中的 AWS App Mesh的条件键。要了解您可以将哪些操作和资源与条件键一起使用，请参阅由定义的操作 AWS App Mesh。

示例

要查看 App Mesh 基于身份的策略的示例，请参阅 AWS App Mesh 基于身份的策略示例。

基于 App Mesh 资源的策略

App Mesh 不支持基于资源的策略。但是，如果您使用 AWS Resource Access Manager (AWS RAM) 服务跨服务共享网格，则该 AWS 服务会将基于资源的策略应用于您的网格。 AWS RAM 有关更多信息，请参阅 为网格授予权限。

基于 App Mesh 标签的授权

您可以将标签附加到 App Mesh 资源或将请求中的标签传递到 App Mesh。要基于标签控制访问，您需要使用 `appmesh:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关为 App Mesh 资源添加标签的更多信息，请参阅为资源添加标签。 AWS

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅创建带受限标签的 App Mesh 网格。

App Mesh IAM 角色

I AM 角色是您的 AWS 账户中具有特定权限的实体。

将临时凭证与 App Mesh 一并使用

可以使用临时凭证进行联合身份验证登录，分派 IAM 角色或分派跨账户角色。您可以通过调用AssumeRole或之类的 AWS STS API 操作来获取临时安全证书GetFederationToken。

App Mesh 支持使用临时凭证。

服务相关角色

服务相关角色允许 AWS 服务访问其他服务中的资源以代表您完成操作。服务相关角色显示在 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

App Mesh 支持服务相关角色。有关创建或管理 App Mesh 服务相关角色的详细信息，请参阅 将服务相关角色用于 App Mesh。

服务角色

此功能允许服务代表您担任服务角色。此角色允许服务访问其他服务中的资源以代表您完成操作。服务角色显示在 IAM 账户中，并归该账户所有。这意味着，IAM 管理员可以更改该角色的权限。但是，这样做可能会中断服务的功能。

App Mesh 不支持服务角色。

AWS App Mesh 基于身份的策略示例

默认情况下，IAM 用户和角色没有创建或修改 App Mesh 资源的权限。他们也无法使用 AWS Management Console AWS CLI、或 AWS API 执行任务。IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的 在 JSON 选项卡上创建策略。

主题

- 策略最佳实践
- 使用 App Mesh 控制台
- 允许用户查看他们自己的权限
- 创建网格
- 列出并描述所有网格

- 创建带受限标签的 App Mesh 网格

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 App Mesh 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管策略或工作职能的AWS 托管策略。
- 应用最低权限 – 在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

使用 App Mesh 控制台

要访问 AWS App Mesh 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您 AWS 账户中 App Mesh 资源的详细信息。如果您创建的基于身份的策略比所需的最低权限更严格，则无法为具有该策略的实体（IAM 用户或角色）正常运行控制台。您可以将 `AWSAppMeshReadOnly` 托管策略附加到用户 有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

创建网格

此示例显示如何创建一个策略，该策略允许用户在任何区域为账户创建网格。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "appmesh:CreateMesh",
            "Resource": "arn:aws:appmesh:*:123456789012:CreateMesh"
        }
    ]
}
```

列出并描述所有网格

此示例显示如何创建一个策略，该策略允许用户拥有只读访问权限以列出或描述所有网格。

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

创建带受限标签的 App Mesh 网格

您可以在 IAM 策略中使用标签以控制可以在 IAM 请求中传递哪些标签。您可以指定可以在 IAM 用户或角色中添加、更改或删除哪些标签键值对。此示例说明如何创建允许创建网格的策略，但前提是使用名为 *teamName* 且值为 *BooksTeam* 的标签创建网格。

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

```
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/teamName": "booksTeam"
                }
            }
        }
    ]
}
```

您可以将该策略附加到您账户中的 IAM 用户。如果用户尝试创建网格，则网格必须包含名为 `teamName` 且值为 `booksTeam` 的标签。如果网格不包含此标签和值，则创建网格的尝试将失败。有关更多信息，请参阅 IAM 用户指南 中的 IAM JSON 策略元素：条件。

AWS App Mesh 的托管策略

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

AWS 托管策略： AWSAppMeshServiceRolePolicy

您可以将 `AWSAppMeshServiceRolePolicy` 附加到 IAM 实体。允许访问使用或管理的 AWS 服务和资源 AWS App Mesh。

要查看此策略的权限，请参阅《AWS 托管策略参考》AWSAppMeshServiceRolePolicy中的。

有关 `AWSAppMeshServiceRolePolicy` 权限详情的信息，请参阅 App Mesh 的服务相关角色权限。

AWS 托管策略： AWSAppMeshEnvoyAccess

您可以将 `AWSAppMeshEnvoyAccess` 附加到 IAM 实体。用于访问虚拟节点配置的 App Mesh Envoy 策略。

要查看此策略的权限，请参阅《AWS 托管策略参考》AWSAppMeshEnvoyAccess中的。

AWS 托管策略： AWSAppMeshFullAccess

您可以将 AWSAppMeshFullAccess 附加到 IAM 实体。提供对 AWS App Mesh API 和的完全访问权限 AWS Management Console。

要查看此策略的权限，请参阅《AWS 托管策略参考》AWSAppMeshFullAccess中的。

AWS 托管策略： AWSAppMeshPreviewEnvoyAccess

您可以将 AWSAppMeshPreviewEnvoyAccess 附加到 IAM 实体。用于访问虚拟节点配置的 App Mesh Preview Envoy 策略。

要查看此策略的权限，请参阅《AWS 托管策略参考》AWSAppMeshPreviewEnvoyAccess中的。

AWS 托管策略： AWSAppMeshPreviewServiceRolePolicy

您可以将 AWSAppMeshPreviewServiceRolePolicy 附加到 IAM 实体。允许访问使用或管理的 AWS 服务和资源 AWS App Mesh。

要查看此策略的权限，请参阅《AWS 托管策略参考》AWSAppMeshPreviewServiceRolePolicy中的。

AWS 托管策略： AWSAppMeshReadOnly

您可以将 AWSAppMeshReadOnly 附加到 IAM 实体。提供对 AWS App Mesh API 和的只读访问权限 AWS Management Console。

要查看此策略的权限，请参阅《AWS 托管策略参考》AWSAppMeshReadOnly中的。

AWS App MeshAWS 托管策略的更新

查看 AWS App Mesh 自该服务开始跟踪这些更改以来 AWS 托管策略更新的详细信息。有关此页面更改的自动提示，请订阅 AWS App Mesh 文档历史记录页面上的 RSS 源。

更改	描述	日期
AWSAppMeshFullAccess— 更新了政策。	已更新AWSAppMeshFullAccess ，允许访问TagResource 和 UntagResource API。	2024 年 4 月 24 日

更改	描述	日期
AWSAppMeshServiceRolePolicy，AWSServiceRoleForAppMesh— 更新了政策。	已更新`AWSServiceRoleForAppMesh` `AWSAppMeshServiceRolePolicy` 并允许访问 AWS Cloud Map `DiscoverInstancesRevision` API。	2023 年 10 月 12 日

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

将服务相关角色用于 App Mesh

AWS App Mesh 使用 AWS Identity and Access Management（IAM）服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 App Mesh 直接相关。服务相关角色由 App Mesh 预定义，并包含服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可让您更轻松地设置 App Mesh，因为您不必手动添加必要的权限。App Mesh 定义其服务相关角色的权限，除非另外定义，否则只有 App Mesh 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其它 IAM 实体的权限策略。

只有在首先删除服务相关角色的相关资源后，才能删除该角色。这将保护您的 App Mesh 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅使用 IAM 的 AWS 服务并查找 Service-Linked Role（服务相关角色）列中显示为 Yes（是）的服务。请选择 Yes 与查看该服务的服务相关角色文档的链接。

App Mesh 的服务相关角色权限

App Mesh 使用名为 AWSServiceRoleForAppMesh 的服务相关角色 — 该角色允许 App Mesh 代表您调用 AWS 服务。

AWSServiceRoleForAppMesh 服务相关角色信任 `appmesh.amazonaws.com` 服务来担任该角色：

权限详细信息

- `servicediscovery:DiscoverInstances`:允许 App Mesh 完成对所有AWS资源的操作。
- `servicediscovery:DiscoverInstancesRevision`:允许 App Mesh 完成对所有AWS资源的操作。

AWSServiceRoleForAppMesh

此策略包含以下权限：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "CloudMapServiceDiscovery",
   "Effect": "Allow",
   "Action": [
    "servicediscovery:DiscoverInstances",
    "servicediscovery:DiscoverInstancesRevision"
   ],
   "Resource": "*"
  },
  {
   "Sid": "ACMCertificateVerification",
   "Effect": "Allow",
   "Action": [
    "acm:DescribeCertificate"
   ],
   "Resource": "*"
  }
 ]
```

```
}
```

必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅 IAM 用户指南中的服务相关角色权限。

为 App Mesh 创建服务相关角色

如果您在 2019 年 6 月 5 日之后在 AWS Management Console、AWS CLI 或 AWS API 中创建了网格，则 App Mesh 会为您创建服务相关角色。要为您创建服务相关角色，您用于创建网格的 IAM 账户必须已附加了 `AWSAppMeshFullAccess`IAM policy，或者附加了包含该`iam:CreateServiceLinkedRole`权限的策略。如果删除此服务相关角色，然后需要再次创建，可以使用相同流程在账户中重新创建此角色。当您创建网格时，App Mesh 将再次为您创建服务相关角色。如果您的账户仅包含 2019 年 6 月 5 日之前创建的网格，并且您想将服务相关角色用于这些网格，则可以使用 IAM 控制台创建角色。

您可以使用 IAM 控制台创建带App Mesh 用例的服务相关角色。在 AWS CLI 或 AWS API 中，使用 `appmesh.amazonaws.com` 服务名称创建服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的创建服务相关角色。如果您删除了此服务相关角色，则可以使用此相同过程再次创建角色。

编辑 App Mesh 的服务相关角色

App Mesh 不允许您编辑 AWSServiceRoleForAppMesh 服务相关角色。创建服务相关角色后，将无法更改角色名称，因为可能有多个实体引用该角色。但是可以使用 IAM 编辑角色说明。有关更多信息，请参见 IAM 用户指南中的编辑服务相关角色。

删除 App Mesh 的服务相关角色

如果不再需要使用某个需要服务相关角色的特征或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须先清除服务相关角色的资源，然后才能手动删除它。

> **Note**
>
> 如果在您试图删除资源时，App Mesh 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

删除 AWSServiceRoleForAppMesh 使用的 App Mesh 资源

1. 删除为网格中所有路由器定义的所有路由。
2. 删除网格中的所有虚拟路由器。

3. 删除网格中的所有虚拟服务。
4. 删除网格中的所有虚拟节点。
5. 删除网格。

为账户中的所有网格完成前面的步骤。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台、AWS CLI 或 AWS API 删除 AWSServiceRoleForAppMesh 服务相关角色。有关更多信息，请参见 IAM 用户指南中的删除服务相关角色。

App Mesh 服务相关角色的受支持区域

App Mesh 支持在服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 App Mesh 端点和配额。

Envoy Proxy 授权

代理授权授权在 Amazon ECS 任务中、在 Amazon EKS 上运行的 Kubernetes 容器组 (pod)中或在 Amazon EC2 实例上运行的 Envoy 代理从 App Mesh Envoy 管理服务中读取一个或多个网格端点的配置。对于在 2021 年 4 月 26 日之前已将 Envoy 连接到 App Mesh 端点的客户账户，使用传输层安全性协议 (TLS) 的虚拟节点和虚拟网关（带或不带 TLS）需要代理授权。对于想要在 2021 年 4 月 26 日之后将 Envoy 连接到其 App Mesh 端点的客户账户，所有 App Mesh 功能都需要代理授权。建议所有客户账户为所有虚拟节点启用代理授权，即使这些账户不使用 TLS，也要使用 IAM 对特定资源进行授权，获得安全、一致的体验。代理授权要求在 IAM policy 中指定 `appmesh:StreamAggregatedResources` 权限。该策略必须附加到 IAM 角色，并且该 IAM 角色必须附加到您托管代理的计算资源上。

创建 IAM policy

如果您希望服务网格中的所有网格端点都能读取所有网格端点的配置，请跳至 创建 IAM 角色。如果您想限制单个网格端点可以从中读取配置的网格端点，则需要创建一项或多项 IAM 策略。建议将可以读取配置的网格端点限制为仅在特定计算资源上运行的 Envoy 代理。创建 IAM policy 并将 `appmesh:StreamAggregatedResources` 权限添加到策略中。以下示例策略允许在服务网格中读取名为 `serviceBv1` 和 `serviceBv2` 的虚拟节点的配置。无法读取服务网格中定义的任何其他虚拟节点的配置。有关创建或编辑 IAM policy 的更多信息，请参阅《创建 IAM policy》和《编辑 IAM policy》。

```
{
```

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

您可以创建多个策略，每个策略都限制对不同网格端点的访问。

创建 IAM 角色

如果您希望服务网格中的所有网格端点都能读取所有网格端点的配置，则只需创建一个 IAM 角色即可。如果要限制单个网格端点可以从中读取配置的网格端点，则需要为在上一步中创建的每个策略创建一个角色。完成运行代理的计算资源的说明。

- Amazon EKS — 如果您想使用单个角色，则可以使用在创建集群时创建并分配给工作节点的现有角色。要使用多个角色，您的集群必须满足在集群上为服务账户启用 IAM 角色中定义的要求。创建 IAM 角色并将这些角色与 Kubernetes 服务账户关联。有关更多信息，请参阅为您的服务账户创建 IAM 角色和策略以及为您的服务账户指定 IAM 角色。
- Amazon ECS — 选择AWS 服务，选择弹性容器服务，然后在创建 IAM 角色时选择弹性容器服务任务用例。
- Amazon EC2 — 选择AWS 服务，选择 EC2，然后在创建 IAM 角色时选择 EC2 用例。无论您是将代理直接托管在 Amazon EC2 实例上，还是在实例上运行的 Kubernetes 上，这都适用。

有关如何创建 IAM 角色的更多信息，请参阅为 AWS 服务创建一个角色。

附加 IAM policy

如果您希望服务网格中的所有网格端点都能读取所有网状端点的配置，请将 `AWSAppMeshEnvoyAccess` 托管 IAM 策略附加到您在上一步中创建的 IAM 角色上。如果要限制单个网格端点可以从中读取配置的网格端点，请将您创建的每个策略附加到您创建的每个角色上。有关向 IAM 角色附加自定义或托管 IAM policy 的更多信息，请参阅添加 IAM 身份权限。

附加 IAM 角色

将每个 IAM 角色附加到相应的计算资源上：

- Amazon EKS — 如果您将策略附加到工作节点的角色，则可以跳过此步骤。如果您创建了单独的角色，则将每个角色分配给单独的 Kubernetes 服务帐户，并将每个服务帐户分配给包含 Envoy 代理的单个 Kubernetes 容器组 (pod) 部署规范。有关更多信息，请参阅《Amazon EKS 用户指南》中的为服务账户指定 IAM 角色和 Kubernetes 文档中的为容器组 (pod) 配置服务账户。
- Amazon ECS — 将亚马逊 ECS 任务角色附加到包含 Envoy 代理的任务定义中。此任务可以使用 EC2 或 Fargate 启动类型进行部署。有关如何创建 Amazon ECS 任务角色并将其附加到任务的更多信息，请参阅为任务指定 IAM 角色。
- 亚马逊 EC2 — IAM 角色必须附加到托管 Envoy 代理的 Amazon EC2 实例上。有关如何将角色附加到 Amazon EC2 实例的更多信息，请参阅我已创建一个 IAM 角色，现在我想将其分配给 EC2 实例。

确认权限

选择其中一个计算服务名称，确认 `appmesh:StreamAggregatedResources` 权限已分配给托管代理的计算资源。

Amazon EKS

可以将自定义策略分配给分配给 Worker 节点的角色，也可以分配给各个容器组 (pod)，或者两者兼而有之。但是，建议您仅在单个容器组 (pod) 上分配策略，这样您就可以将单个 pod 的访问限制为单个网格端点。如果策略已附加到分配给 Worker 节点的角色上，请选择 Amazon EC2 选项卡，然后完成该选项卡中针对您的工作节点实例找到的步骤。要确定向 Kubernetes 容器组 (pod) 分配了哪个 IAM 角色，请完成以下步骤。

1. 查看 Kubernetes 部署的详细信息，其中包括您要确认已将 Kubernetes 服务帐户分配到的容器组 (pod)。以下命令查看名为 *my-deployment* 的部署的详细信息。

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

   在返回的输出中，记下右边 `Service Account:` 的值。如果以 `Service Account:` 开头的行不存在，则当前未为部署分配自定义 Kubernetes 服务帐户。您需要分配一个。有关更多信息，请参阅 Kubernetes 文档中的为容器组 (pod) 配置服务账户。
2. 查看上一步中返回的服务账户的详细信息。以下命令查看名为 *my-service-account* 的服务帐号的详细信息。

```
kubectl describe serviceaccount my-service-account
```

如果 Kubernetes 服务账号与 AWS Identity and Access Management 角色关联，则返回的其中一行将与以下示例类似。

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

在前面的示例中，`my-deployment` 是与服务账号关联的 IAM 角色的名称。如果服务帐号输出不包含与上面示例类似的行，那么 Kubernetes 服务帐号未与账户关联，您需要将其关联到一个 AWS Identity and Access Management 账户。有关更多信息，请参阅为服务账户指定 IAM 角色。

3. 登录 AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
4. 在左侧导航窗格中，选择角色。选择您在上一步中记下的 IAM 角色的名称。
5. 确认已列出您之前创建的自定义策略或 `AWSAppMeshEnvoyAccess` 托管策略。如果两个策略都未附加，请将 IAM policy 附加到 IAM 角色。如果您想附加自定义 IAM policy 但还没有，则需要创建具有所需权限的自定义 IAM policy。如果附加了自定义 IAM policy，请选择该策略并确认其中包含 `"Action": "appmesh:StreamAggregatedResources"`。如果不是，则需要将该权限添加到您的自定义 IAM policy 中。您还可以确认是否列出了特定网格端点的 Amazon 资源名称 (ARN)。如果未列出 ARN，则可以编辑策略以添加、删除或更改列出的 ARN。要了解更多信息，请参阅编辑 IAM policy 和创建 IAM policy。
6. 对每个包含 Envoy 代理的 Kubernetes 容器组 (pod)，重复上述步骤。

Amazon ECS

1. 从 Amazon ECS 控制台中，选择任务定义。
2. 选择 Amazon ECS 任务。
3. 在任务定义名称页面上，选择任务定义。
4. 在任务定义页面上，选择任务角色右侧的 IAM 角色名称链接。如果未列出 IAM 角色，则需要创建一个 IAM 角色并通过更新任务定义将其附加到您的任务上。
5. 在摘要页面的权限选项卡上，确认已列出您之前创建的自定义策略或 `AWSAppMeshEnvoyAccess` 托管策略。如果两个策略都未附加，请将 IAM policy 附加到 IAM 角色。如果您想附加自定义 IAM policy 但还没有，则需要创建自定义 IAM

policy。如果附加了自定义 IAM policy，请选择该策略并确认其中包含 `"Action": "appmesh:StreamAggregatedResources"`。如果不是，则需要将该权限添加到您的自定义 IAM policy 中。您还可以确认是否列出了特定网格端点的适当 Amazon 资源名称 (ARN)。如果未列出 ARN，则可以编辑策略以添加、删除或更改列出的 ARN。要了解更多信息，请参阅编辑 IAM Policy 和创建 IAM policy。

6. 对于包含 Envoy 代理的每个任务定义，重复前面的步骤。

Amazon EC2

1. 在 Amazon EC2 控制台中，选择左侧导航栏中的实例。
2. 选择一个托管 Envoy 代理的实例。
3. 在描述选项卡中，选择 IAM 角色右侧的 IAM 角色名称链接。如果未列出 IAM 角色，则需要创建一个 IAM 角色。
4. 在摘要页面的权限选项卡上，确认已列出您之前创建的自定义策略或 `AWSAppMeshEnvoyAccess` 托管策略。如果两个策略都未附加，请将 IAM policy 附加到 IAM 角色。如果您想附加自定义 IAM policy 但还没有，则需要创建自定义 IAM policy。如果附加了自定义 IAM policy，请选择该策略并确认其中包含 `"Action": "appmesh:StreamAggregatedResources"`。如果不是，则需要将该权限添加到您的自定义 IAM policy 中。您还可以确认是否列出了特定网格端点的适当 Amazon 资源名称 (ARN)。如果未列出 ARN，则可以编辑策略以添加、删除或更改列出的 ARN。要了解更多信息，请参阅编辑 IAM Policy 和创建 IAM policy。
5. 对托管 Envoy 代理的每个实例重复上述步骤。

对 AWS App Mesh 身份和访问进行故障排除

使用以下信息可帮助您诊断和修复在使用 App Mesh 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 App Mesh 中执行操作
- 我想允许 AWS 账户以外的用户访问我的 App Mesh 资源

我无权在 App Mesh 中执行操作

如果 AWS Management Console 告诉您您无权执行某项操作，则必须联系管理员寻求帮助。管理员是向您提供登录凭证的人。

当 mateojackson IAM 用户尝试使用控制台在网格*my-virtual-node*中创建名为 *my-m* esh 但没有权限的虚拟节点时，会发生以下错误。appmesh:CreateVirtualNode

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: appmesh:CreateVirtualNode on resource: arn:aws:appmesh:us-
east-1:123456789012:mesh/my-mesh/virtualNode/my-virtual-node
```

在这种情况下，Mateo 请求管理员更新其策略，以允许他使用 appmesh:CreateVirtualNode 操作创建虚拟节点 。

> **Note**
>
> 由于虚拟节点是在网格中创建的，因此 Mateo 的账户还需要 appmesh:DescribeMesh 和 appmesh:ListMeshes 操作才能在控制台中创建虚拟节点。

我想允许 AWS 账户以外的用户访问我的 App Mesh 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 App Mesh 是否支持这些功能，请参阅 如何 AWS App Mesh 与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅 IAM 用户指南中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

使用记录 AWS App Mesh API 调用 AWS CloudTrail

AWS App Mesh 与AWS CloudTrail一项服务集成，该服务提供用户、角色或角色所执行操作的记录 AWS 服务。 CloudTrail 将 App Mesh 的所有 API 调用捕获为事件。捕获的调用包含来自 App Mesh

控制台的调用和代码对 App Mesh API 操作的调用。使用收集的信息 CloudTrail，您可以确定向 App Mesh 发出的请求、发出请求的 IP 地址、发出请求的时间以及其他详细信息。

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根用户凭证还是用户凭证发出的。
- 请求是否代表 IAM Identity Center 用户发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

CloudTrail 在您创建账户 AWS 账户 时在您的账户中处于活动状态，并且您自动可以访问 CloudTrail 活动历史记录。 CloudTrail 事件历史记录提供了过去 90 天中记录的管理事件的可查看、可搜索、可下载且不可变的记录。 AWS 区域有关更多信息，请参阅《AWS CloudTrail 用户指南》中的 “使用 CloudTrail 事件历史记录”。查看活动历史记录不 CloudTrail收取任何费用。

要持续记录 AWS 账户 过去 90 天内的事件，请创建跟踪或 CloudTrailLake 事件数据存储。

CloudTrail 步道

跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。使用创建的所有跟踪 AWS Management Console 都是多区域的。您可以通过使用 AWS CLI创建单区域或多区域跟踪。建议创建多区域跟踪，因为您可以捕获账户 AWS 区域 中的所有活动。如果您创建单区域跟踪，则只能查看跟踪的 AWS 区域中记录的事件。有关跟踪的更多信息，请参阅《AWS CloudTrail 用户指南》中的为您的 AWS 账户创建跟踪和为组织创建跟踪。

通过创建跟踪，您可以免费将正在进行的管理事件的一份副本传送到您的 Amazon S3 存储桶，但会收取 Amazon S3 存储费用。 CloudTrail 有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。有关 Amazon S3 定价的信息，请参阅 Amazon S3 定价。

CloudTrail 湖泊事件数据存储

CloudTrail L@@ ak e 允许你对自己的事件运行基于 SQL 的查询。 CloudTrail Lake 将基于行的 JSON 格式的现有事件转换为 Apache ORC 格式。ORC 是一种针对快速检索数据进行优化的列式存储格式。事件将被聚合到事件数据存储中，它是基于您通过应用高级事件选择器选择的条件的不可变的事件集合。应用于事件数据存储的选择器用于控制哪些事件持续存在并可供您查询。有关 CloudTrail Lake 的更多信息，请参阅AWS CloudTrail 用户指南中的使用 AWS CloudTrail Lake。

CloudTrail 湖泊事件数据存储和查询会产生费用。创建事件数据存储时，您可以选择要用于事件数据存储的定价选项。定价选项决定了摄取和存储事件的成本，以及事件数据存储的默认和最长保留期。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

中的 App Mesh 管理事件 CloudTrail

管理事件提供有关对中的资源执行的管理操作的信息 AWS 账户。这些也称为控制层面操作。默认情况下，CloudTrail 记录管理事件。

AWS App Mesh 将所有 App Mesh 控制平面操作记录为管理事件。有关 App Mesh 记录到的 AWS App Mesh 控制平面操作的列表 CloudTrail，请参阅 AWS App Mesh API 参考。

App Mesh 事件示例

事件代表来自任何来源的单个请求，包括有关所请求的 API 操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此事件不会按任何特定顺序出现。

以下示例显示了演示该`StreamAggregatedResources`操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:d060be4ac3244e05aca4e067bfe241f8",
        "arn": "arn:aws:sts::123456789012:assumed-role/Application-TaskIamRole-
C20GBLBRLBXE/d060be4ac3244e05aca4e067bfe241f8",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "invokedBy": "appmesh.amazonaws.com"
    },
    "eventTime": "2021-06-09T23:09:46Z",
    "eventSource": "appmesh.amazonaws.com",
    "eventName": "StreamAggregatedResources",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "appmesh.amazonaws.com",
    "userAgent": "appmesh.amazonaws.com",
    "eventID": "e3c6f4ce-EXAMPLE",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails": {
        "connectionId": "e3c6f4ce-EXAMPLE",
        "nodeArn": "arn:aws:appmesh:us-west-2:123456789012:mesh/CloudTrail-Test/
virtualNode/cloudtrail-test-vn",
        "eventStatus": "ConnectionEstablished",
```

```
        "failureReason": ""
    },
    "eventCategory": "Management"
}
```

有关 CloudTrail 录音内容的信息，请参阅《AWS CloudTrail 用户指南》中的CloudTrail录制内容。

中的数据保护 AWS App Mesh

分 AWS 担责任模型适用于中的数据保护 AWS App Mesh。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的 AWS 责任共担模式和 GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用 multi-factor authentication（MFA）。
- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括使用控制台、API 或 AWS SDK AWS 服务 使用 App Mesh 或其他工具包的情况。 AWS CLI在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

数据加密

使用 App Mesh 时，您的数据会被加密。

静态加密

默认情况下，您创建的 App Mesh 配置是静态加密的。

传输中加密

App Mesh 服务端点使用 HTTPS 协议。Envoy 代理和 App Mesh Envoy 管理服务之间的所有通信均经过加密。如果您需要对 Envoy 代理和 App Mesh Envoy 管理服务之间的通信进行符合 FIPS 的加密，则可以使用 Envoy 代理容器镜像的 FIPS 变体。有关更多信息，请参阅 Envoy 镜像。

虚拟节点内的容器之间的通信不会加密，但这些流量不会离开网络命名空间。

合规性验证 AWS App Mesh

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了部署以安全性和合规性为重点 AWS 的基准环境的步骤。
- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规性 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

  > **Note**
  >
  > 并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合AWS 规资源 — 此工作簿和指南集可能适用于您的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务 并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。

- AWS Security Hub— 这 AWS 服务 可以全面了解您的安全状态 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

中的基础设施安全 AWS App Mesh

作为一项托管服务 AWS App Mesh ，受 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用通过网络访问 App Mesh。客户端必须支持以下内容：

- 传输层安全性协议 (TLS) 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

您可以将 App Mesh 配置为使用接口 VPC 端点以改善 VPC 的安全状况。有关更多信息，请参阅 App Mesh 接口 VPC 端点 (AWS PrivateLink)。

App Mesh 接口 VPC 端点 (AWS PrivateLink)

您可以将 App Mesh 配置为使用接口 VPC 端点以改善 Amazon VPC 的安全状况。接口端点由一项技术提供支持 AWS PrivateLink，该技术使您能够使用私有 IP 地址私密访问 App Mesh API。PrivateLink将您的亚马逊 VPC 和 App Mesh 之间的所有网络流量限制到亚马逊网络。

您无需进行配置 PrivateLink，但我们建议您这样做。有关 PrivateLink 和接口 VPC 终端节点的更多信息，请参阅通过访问服务 AWS PrivateLink。

App Mesh 接口 VPC 端点的注意事项

在为 App Mesh 设置接口 VPC 端点之前，请注意以下事项：

- 如果您的 Amazon VPC 没有互联网网关，并且您的任务使用`awslogs`日志驱动程序向日志发送日志信息，则必须为 CloudWatch CloudWatch 日志创建接口 VPC 终端节点。有关更多信息，请参阅 Amazon CloudWatch CloudWatch 日志用户指南中的将日志与接口 VPC 终端节点配合使用。
- VPC 终端节点不支持 AWS 跨区域请求。确保在计划向 App Mesh 发出 API 调用的同一区域中创建端点。
- VPC 端点仅通过 Amazon Route 53 支持 Amazon 提供的 DNS。如果您希望使用自己的 DNS，可以使用条件 DNS 转发。有关更多信息，请参阅《Amazon VPC 用户指南》中的 DHCP 选项集。
- 附加到 Amazon VPC 端点的安全组必须允许端口 443 上来自 VPC 的私有子网的传入连接。

> **Note**
>
> Envoy 连接不支持通过向 VPC 端点附加端点策略（例如，使用服务名称 `com.amazonaws.Region.appmesh-envoy-management`）来控制对 App Mesh 的访问。

有关其他注意事项和限制，请参阅接口端点可用区域注意事项和接口端点属性和限制。

为 App Mesh 创建接口 VPC 端点

要为 App Mesh 服务创建 VPC 端点，请使用 《Amazon VPC 用户指南》中的创建接口端点过程。`com.amazonaws.Region.appmesh-envoy-management` 为连接到 App Mesh 的公共 Envoy 管理服务的 Envoy 代理以及 `com.amazonaws.Region.appmesh` 网格操作指定服务名称。

> **Note**
>
> *##*表示 App Mesh 支持的 AWS 区域（例如`us-east-2`美国东部（俄亥俄州）区域的区域标识符。

尽管您可以在任何支持 App Mesh 的区域中为 App Mesh 定义接口 VPC 端点，但您可能无法为每个区域中的所有可用区域定义一个端点。要了解某个区域中接口 VPC 终端节点支持哪些可用区，请使用describe-vpc-endpoint-services 命令或使用 AWS Management Console。例如，以下命令返回可在美国东部（俄亥俄州）区域内部署 App Mesh 接口 VPC 端点的可用区域：

```
aws --region us-east-2 ec2 describe-vpc-endpoint-services --query 'ServiceDetails[?
ServiceName==`com.amazonaws.us-east-2.appmesh-envoy-management`].AvailabilityZones[]'
```

```
aws --region us-east-2 ec2 describe-vpc-endpoint-services --query 'ServiceDetails[?
ServiceName==`com.amazonaws.us-east-2.appmesh`].AvailabilityZones[]'
```

AWS App Mesh 中的故障恢复能力

AWS全球基础设施围绕AWS区域和可用区构建。AWS区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

App Mesh 跨多个可用区运行控制面板实例以确保高可用性。App Mesh 可以自动检测和替换运行状况不佳的控制面板实例，并为它们提供自动版本升级和修补。

AWS App Mesh 中的灾难恢复

App Mesh 服务管理客户数据备份。您无需执行任何操作即可管理备份。备份数据已加密。

中的配置和漏洞分析 AWS App Mesh

App Mesh 出售托管的 Envoy 代理 Docker 容器镜像，该镜像随微服务一起部署。App Mesh 确保使用最新的漏洞和性能补丁对容器映像进行修补。在向您提供图像之前，App Mesh 会根据 App Mesh 功能集测试新的 Envoy 代理版本。

您必须更新您的微服务才能使用更新的容器映像版本。以下是该图像的最新版本。

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

App Mesh 故障排除

本章讨论故障排除最佳实践以及使用 App Mesh 时可能遇到的常见问题。选择以下方面之一，查看该方面的最佳实践和常见问题。

主题

App Mesh 故障排除最佳实践

我们建议您遵循本主题下的最佳实践，排除使用 App Mesh 时遇到的问题。

启用 Envoy 代理管理界面

Envoy 代理附带一个管理界面，您可以使用该界面来发现配置和统计数据，以及执行其他管理功能，例如连接耗尽。有关更多信息，请参阅 Envoy 文档中的管理界面。

如果您使用托管 Envoy 镜像，则在默认情况下，管理端点在端口 9901 上处于启用状态。在 App Mesh 设置故障排除 中提供的示例中，示例管理端点 URL 显示为 `http://my-app.default.svc.cluster.local:9901/`。

> **Note**
>
> 管理端点永远不应暴露在公共互联网上。此外，我们建议您监控管理端点日志，这些日志由 `ENVOY_ADMIN_ACCESS_LOG_FILE` 环境变量默认设置为 `/tmp/envoy_admin_access.log`。

启用 Envoy DogStats D 集成以进行指标卸载

可以将 Envoy 代理配置为卸载 OSI 第 4 层和第 7 层流量以及内部进程运行状况的统计数据。虽然本主题展示了如何在不将指标转移到 CloudWatch 指标和 Prometheus. 之类的接收器的情况下使用这些统计信息，但将所有应用程序的这些统计数据放在一个集中的位置可以帮助您更快地诊断问题并确认行为。有关更多信息，请参阅使用亚马逊 CloudWatch指标和 Prometheus 文档。

您可以通过设置中定义的参数来配置 DogStats D 指标DogStatsD 变量。有关 DogStats D 的更多信息，请参阅 DogStatsD 文档。您可以在 A pp Mesh 中找到将指标卸载到AWS CloudWatch 指标的演示，其中包含了 Amazon ECS 基础知识演练。 GitHub

启用访问日志

我们建议您启用访问 虚拟节点 和 虚拟网关 上的日志，发现有关应用程序之间传输的流量的详细信息。有关更多信息，请参阅 Envoy 文档中的访问日志记录。这些日志提供有关 OSI 第 4 层和第 7 层流量行为的详细信息。当你使用 Envoy 的默认格式时，你可以使用以下解析语句通过 Logs Ins ights 分析访问CloudWatch 日志。

```
parse @message "[*] \"* * *\" * * * * * * * * * * *" as StartTime, Method, Path,
 Protocol, ResponseCode, ResponseFlags, BytesReceived, BytesSent, DurationMillis,
 UpstreamServiceTimeMillis, ForwardedFor, UserAgent, RequestId, Authority, UpstreamHost
```

在预生产环境中启用 Envoy 调试日志记录

我们建议在预生产环境中将 Envoy 代理的日志级别设置为 debug。调试日志可以帮助您在将关联的 App Mesh 配置升级到生产环境之前识别问题。

如果您使用的是 Envoy 镜像，则可以通过 ENVOY_LOG_LEVEL 环境变量将日志级别设置为 debug。

> **Note**
>
> 我们建议不在生产环境中使用 debug 级别。将级别设置为debug会增加日志记录，并且可能会影响性能和转移到日志等CloudWatch 解决方案的日志的总体成本。

当你使用 Envoy 的默认格式时，你可以使用以下解析语句通过 Logs Ins ights 分析流程CloudWatch 日志：

```
parse @message "[*][*][*][*] [*] *" as Time, Thread, Level, Name, Source, Message
```

使用 App Mesh 控制面板监控 Envoy 代理连接

我们建议您监控 Envoy 指标 `control_plane.connected_state`，确保 Envoy 代理与 App Mesh 控制面板通信以获取动态配置资源。有关更多信息，请参阅管理服务器。

App Mesh 设置故障排除

本主题详细介绍了您在设置 App Mesh 时可能遇到的常见问题。

无法提取 Envoy 容器镜像

症状

在 Amazon ECS 任务中，您会收到以下错误消息。以下消息中的 Amazon ECR *## ID* 和*##*可能会有所不同，具体取决于您从哪个 Amazon ECR 存储库中提取容器映像。

```
CannotPullContainerError: Error response from daemon: pull access denied
 for 840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy, repository does
 not exist or may require 'docker login'
```

解决方案

此错误表示正在使用的任务执行角色无权与 Amazon ECR 通信，也无法从存储库中提取 Envoy 容器镜像。分配给您的 Amazon ECS 任务的任务执行角色需要包含以下声明的 IAM policy：

```
{
  "Action": [
    "ecr:BatchCheckLayerAvailability",
    "ecr:GetDownloadUrlForLayer",
    "ecr:BatchGetImage"
  ],
  "Resource": "arn:aws:ecr:us-west-2:111122223333:repository/aws-appmesh-envoy",
  "Effect": "Allow"
},
{
  "Action": "ecr:GetAuthorizationToken",
  "Resource": "*",
  "Effect": "Allow"
}
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法连接到 App Mesh Envoy 管理服务

症状

您的 Envoy 代理无法连接到 App Mesh Envoy 管理服务。您将看到：

- 连接被拒绝错误
- 连接超时
- 解析 App Mesh Envoy 管理服务端点时出错
- gRPC 错误

解决方案

确保您的 Envoy 代理可以访问互联网或私有 VPC 端点，并且您的安全组允许端口 443 上的出站流量。App Mesh 的公有 Enoy 管理服务端点遵循完全限定域名（FQDN）格式。

```
# App Mesh Production Endpoint
appmesh-envoy-management.Region-code.amazonaws.com

# App Mesh Preview Endpoint
appmesh-preview-envoy-management.Region-code.amazonaws.com
```

您可以使用以下命令调试与 EMS 的连接。这会向 Envoy 管理服务发送一个有效但空的 gRPC 请求。

```
curl -v -k -H 'Content-Type: application/grpc' -X POST https://
appmesh-envoy-management.Region-code.amazonaws.com:443/
envoy.service.discovery.v3.AggregatedDiscoveryService/StreamAggregatedResources
```

如果您收到这些回复，则表示您与 Envoy 管理服务的连接正常。要调试 gRPC 相关错误，请参阅 Envoy 中与 App Mesh Envoy 管理服务断开连接的错误以及错误文本。

```
grpc-status: 16
grpc-message: Missing Authentication Token
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

Envoy 已断开与 App Mesh Envoy 管理服务的连接，并显示错误

症状

您的 Envoy 代理无法连接到 App Mesh Envoy 管理服务并接收其配置。您的 Envoy 代理日志包含如下所示的日志条目。

```
gRPC config stream closed: gRPC status code, message
```

解决方案

在大多数情况下，日志的消息部分应指出问题所在。下表列出了您可能看到的最常见的 gRPC 状态代码、其原因和解决方法。

gRPC 状态码	原因	解决方案
0	优雅地断开与 Envoy 管理服务的连接。	没有消息。App Mesh 偶尔会断开带有此状态代码的 Envoy 代理的连接。Envoy 将重新连接并继续接收更新。
3	找不到网格端点（虚拟节点或虚拟网关）或其关联资源之一。	仔细检查您的 Envoy 配置，确保它具有其所代表的 App Mesh 资源的相应名称。如果您的 App Mesh 资源与其他 AWS 资源（例如 AWS Cloud Map 命名空间或 ACM 证书）集成，请确保这些资源存在。
7	Envoy 代理无权执行操作，例如连接到 Envoy 管理服务或检索相关资源。	请务必创建包含适用于 App Mesh 和其他服务的相应策略声明的 IAM policy，并将该策略附加到您的 Envoy 代理用于连接 Envoy 管理服务的 IAM 用户或角色。
8	给定 App Mesh 资源的 Envoy 代理数量超过了账户级别的服务限额。	有关默认账户配额以及如何请求增加配额的更多信息，请参阅 App Mesh 服务限额。

gRPC 状态码	原因	解决方案
16	Envoy 代理没有有效的 AWS 身份验证凭证。	确保 Envoy 拥有适当的凭证，可以通过 IAM 用户或角色连接到 AWS 服务。如果 Envoy 进程使用 1024 文件描述符，则版本 v1.24 及之前版本的 Envoy 中存在一个已知问题 #24136 无法获取凭证。当 Envoy 提供高流量服务时，就会发生这种情况。您可以通过在调试级别查看 Envoy 日志中是否有文本“A libcurl function was given a bad argument”来确认此问题。要缓解此问题，请升级到 Envoy 版本 v1.25.1.0-prod 或更高版本。

您可以使用以下查询通过 Amazon CloudWatch Insigh ts 查看来自 Envoy 代理的状态代码和消息：

```
filter @message like /gRPC config stream closed/
| parse @message "gRPC config stream closed: *, *" as StatusCode, Message
```

如果提供的错误消息无济于事，或者您的问题仍未解决，请考虑提出GitHub 问题。

Envoy 容器运行状况检查、就绪探测或活跃度探测失败

症状

您的 Envoy 代理在 Amazon ECS 任务、Amazon EC2 实例或 Kubernetes 容器组 (pod) 中的运行状况检查失败。例如，您使用以下命令查询 Envoy 管理界面，但收到的状态不是 LIVE。

```
curl -s http://my-app.default.svc.cluster.local:9901/server_info | jq '.state'
```

解决方案

以下是补救步骤列表，具体取决于 Envoy 代理返回的状态。

- PRE_INITIALIZING 或者 INITIALIZING — Envoy 代理尚未接收配置，或者无法从 App Mesh Envoy 管理服务连接和检索配置。Envoy 在尝试连接时可能会收到来自 Envoy 管理服务的错误。有关更多信息，请查看 Envoy 已断开与 App Mesh Envoy 管理服务的连接，并显示错误 中的错误文本。
- DRAINING：Envoy 代理已开始耗尽连接，以响应 Envoy 管理界面上的 /healthcheck/fail 或 /drain_listeners 请求。除非您即将终止您的 Amazon ECS 任务、Amazon EC2 实例或 Kubernetes 容器组 (pod)，否则我们不建议您在管理界面上调用这些路径。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

从负载均衡器到网格端点的运行状况检查失败

症状

容器运行状况检查或就绪探测器认为您的网格端点运行状况良好，但是从负载均衡器到网格端点的运行状况检查失败。

解决方案

要解决此问题，请完成以下任务。

- 确保与您的网格端点关联的安全组接受您为运行状况检查配置的端口上的入站流量。
- 请确保在手动请求运行状况检查时始终如一地成功；例如，来自您的 VPC 内的堡垒主机。
- 如果您正在为虚拟节点配置运行状况检查，我们建议您在应用程序中实现运行状况检查端点；例如，/ping for HTTP。这可确保 Envoy 代理和您的应用程序均可从负载均衡器路由。
- 根据所需的功能，您可以针对虚拟节点使用任何类型的弹性负载均衡器。有关更多信息，请参阅 弹性负载均衡功能。
- 如果您正在为虚拟网关配置运行状况检查，我们建议使用网络负载均衡器，并在虚拟网关的侦听器端口上进行 TCP 或 TLS 运行状况检查。这样可以确保虚拟网关侦听器已启动并准备好接受连接。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

虚拟网关不接受端口 1024 或更少的流量

症状

您的虚拟网关不接受端口 1024 或更小的流量，但接受端口号大于 1024 的流量。例如，使用以下命令查询 Envoy 统计数据并收到一个非零值。

```
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "update_rejected"
```

您可能会在日志中看到类似于以下文本的文本，描述无法绑定到特权端口：

```
gRPC config for type.googleapis.com/envoy.api.v2.Listener rejected: Error adding/
updating listener(s) lds_ingress_0.0.0.0_port_<port num>: cannot bind '0.0.0.0:<port
 num>': Permission denied
```

解决方案

要解决此问题，为网关指定的用户需要具有 linux 功能 CAP_NET_BIND_SERVICE。有关更多信息，请参阅《Linux 程序员手册》中的功能、ECS 任务定义参数中的 Linux 参数和 Kubernetes 文档中的设置容器功能。

> **Important**
>
> Fargate 必须使用大于 1024 的端口值。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

App Mesh 连接故障排除

本主题详细介绍了您在使用 App Mesh 连接时可能遇到的常见问题。

无法解析虚拟服务的 DNS 名称

症状

您的应用程序无法解析它正在尝试连接的虚拟服务的 DNS 名称。

解决方案

这是一个已知问题。有关更多信息，请参阅 “ VirtualServices 按任意主机名命名” 或 FQDN GitHub 问题。App Mesh 中的虚拟服务可以被命名为任何名称。只要虚拟服务名称中有 DNS A 记录，并且应用程序可以解析虚拟服务名称，Envoy 就会代理请求并将其路由到相应的目的地。要解决此问题，请在任何非环回 IP 地址（例如 `10.10.10.10` 虚拟服务名称）中添加 DNS A 记录。在以下条件下可以添加 DNS A 记录：

- 在 Amazon Route 53 中，如果名称以您的私有托管区域名称为后缀

- 在应用程序容器的 /etc/hosts 文件中
- 在您管理的第三方 DNS 服务器中

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法连接到虚拟服务后端

症状

您的应用程序无法与虚拟节点上定义为后端的虚拟服务建立连接。尝试建立连接时，连接可能完全失败，或者从应用程序的角度来看，请求可能会失败并显示 HTTP 503 响应代码。

解决方案

如果应用程序根本无法连接（未返回 HTTP 503 响应代码），请执行以下操作：

- 确保您的计算环境已设置为可与 App Mesh 配合使用。
  - 对于 Amazon ECS，请确保已启用相应的代理配置。有关演 end-to-end 练，请参阅 App Mesh 和 Amazon ECS 入门。
  - 对于 Kubernetes，包括亚马逊 EKS，请确保通过 Helm 安装了最新的 App Mesh 控制器。有关更多信息，请参阅 Helm Hub 上的 App Mesh 控制器或教程：配置与 Kubernetes 的应用程序网格集成。
  - 对于亚马逊 EC2，请确保已设置用于代理 App Mesh 流量的亚马逊 EC2 实例。有关更多信息，请参阅更新服务。
- 确保在您的计算服务上运行的 Envoy 容器已成功连接到 App Mesh Envoy 管理服务。您可以通过查看该领域的 Envoy 统计数据来确认这一点 control_plane.connected_state。有关更多信息 control_plane.connected_state，请参阅故障排除最佳实践中的监控 Envoy 代理连接。

  如果 Envoy 最初能够建立连接，但后来断开连接且从未重新连接，请参阅 Envoy 与 App Mesh Envoy 管理服务断开连接，并附上错误文本，以解决断开连接的原因。

如果应用程序连接但请求失败并显示 HTTP 503 响应代码，请尝试以下操作：

- 确保您要连接的虚拟服务存在于网格中。
- 确保虚拟服务有提供商（虚拟路由器或虚拟节点）。
- 使用 Envoy 作为 HTTP 代理时，如果您看到出口流量通过 Envoy 统计数据进入 cluster.cds_egress_*_mesh-allow-all 而非正确的目的地，那么 Envoy 很可能没有正确路

由请求。filter_chains这可能是由于使用了不合格的虚拟服务名称所致。我们建议您使用实际服务的服务发现名称作为虚拟服务名称，因为 Envoy 代理通过其他虚拟服务的名称与其他虚拟服务进行通信。

有关更多信息，请参阅虚拟服务。

- 检查 Envoy 代理日志中是否出现以下任何错误消息：
  - No healthy upstream：Envoy 代理尝试路由到的虚拟节点没有任何已解析的端点，或者它没有任何运行正常的端点。确保目标虚拟节点的服务发现和运行状况检查设置正确。

    如果在部署或扩展后端虚拟服务的过程中对该服务的请求失败，请按照 当虚拟服务有虚拟节点提供程序 503 时，某些请求会失败，并显示 HTTP 状态码 中的指导进行操作。
  - No cluster match for URL：这很可能是当向虚拟服务发送的请求与虚拟路由器提供商下定义的任何路由所定义的标准都不匹配时引起的。通过确保路径和 HTTP 请求标头正确，确保将来自应用程序的请求发送到受支持的路由。
  - No matching filter chain found：这很可能是通过无效端口向虚拟服务发送请求时造成的。确保来自应用程序的请求使用虚拟路由器上指定的相同端口。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法连接到外部服务

症状

您的应用程序无法连接到网格之外的服务，例如 amazon.com。

解决方案

默认情况下，App Mesh 不允许从网格内应用程序到网格之外任何目的地的出站流量。要启用与外部服务的通信，有两个选项：

- 将网格资源的出站过滤器设置为 ALLOW_ALL。此设置将允许网格内的任何应用程序与网格内外的任何目标 IP 地址进行通信。
- 使用虚拟服务、虚拟路由器、路由和虚拟节点对网格中的外部服务进行建模。例如，要对外部服务进行建模 example.com，您可以创建一个名为 example.com 的虚拟服务，该虚拟路由器和路由将所有流量发送到 DNS 服务发现主机名为 example.com 的虚拟节点。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法连接到 MySQL 或 SMTP 服务器

症状

当允许到所有目的地（网格 `EgressFilter type=ALLOW_ALL`）的出站流量时，例如使用虚拟节点定义的 SMTP 服务器或 MySQL 数据库，您应用程序的连接将失败。例如，以下是尝试连接到 MySQL 服务器时出现的错误消息。

```
ERROR 2013 (HY000): Lost connection to MySQL server at 'reading initial communication
 packet', system error: 0
```

解决方案

这是一个已知问题，可通过使用 App Mesh 镜像版本 1.15.0 或更高版本来解决。有关更多信息，请参阅无法通过 App Mesh 连接到 MySQL GitHub 问题。之所以出现此错误，是因为由 App Mesh 配置的 Envoy 中的出站侦听器添加了 Envoy TLS Inspector 侦听器过滤器。有关更多信息，请参阅 Envoy 文档中的 TLS Inspector。此过滤器通过检查从客户端发送的第一个数据包来评估连接是否使用 TLS。但是，在 MySQL 和 SMTP 中，服务器会在连接后发送第一个数据包。有关 MySQL 的更多信息，请参阅 MySQL 文档中的首次握手。由于服务器发送第一个数据包，因此过滤器检查失败。

要根据您的 Envoy 版本解决此问题，请执行以下操作：

- 如果您的 App Mesh 镜像 Envoy 版本为 1.15.0 或更高版本，请勿将 MySQL、SMTP、MSSQL 等外部服务建模为应用程序虚拟节点的后端。
- 如果您的 App Mesh 镜像 Envoy 版本低于 1.15.0，请将端口 `3306` 添加到您的 MySQL 服务的值列表`APPMESH_EGRESS_IGNORED_PORTS` 中，并作为您用于 STMP 的端口。

> **Important**
>
> 虽然标准 SMTP 端口是 25、587 和 465，但您只应添加正在使用的端口，而不是全部添加 `APPMESH_EGRESS_IGNORED_PORTS` 三个端口。

有关更多信息，请参阅更新适用于 Kubernetes 的服务、Amazon ECS 的更新服务或亚马逊 EC2 的更新服务。

如果您的问题仍未解决，则可以向我们详细说明您在使用现有GitHub 问题时遇到的情况，或者联系 Su AWSpp ort。

无法连接到 App Mesh 中建模为 TCP 虚拟节点或虚拟路由器的服务

症状

您的应用程序无法连接到使用 App Mesh PortMapping定义中的 TCP 协议设置的后端。

解决方案

这是一个已知问题。有关更多信息，请参阅路由到同一端口上的多个 TCP 目的地 GitHub。由于 OSI 第 4 层向 Envoy 代理提供的信息存在限制，App Mesh 目前不允许建模为 TCP 的多个后端目标共享同一个端口。为确保可将 TCP 流量正确路由到所有后端目标，请执行以下操作：

- 确保所有目的地都使用唯一端口。如果您使用虚拟路由器提供商来提供后端虚拟服务，则可以更改虚拟路由器端口，而无需更改其路由到的虚拟节点上的端口。这样，应用程序可在虚拟路由器端口上打开连接，而 Envoy 代理可继续使用虚拟节点中定义的端口。
- 如果建模为 TCP 的目标是 MySQL 服务器，或者任何其他基于 TCP 的协议，服务器在连接后使用该协议发送第一批数据包，请参见 无法连接到 MySQL 或 SMTP 服务器。

如果您的问题仍未解决，则可以向我们详细说明您在使用现有GitHub 问题时遇到的情况，或者联系 Su AWSpp ort。

成功连接到未列为虚拟节点虚拟服务后端的服务

症状

您的应用程序能够连接并发送流量到您的虚拟节点上未指定为虚拟服务后端的目的地。

解决方案

如果请求成功发送到未在 App Mesh API 中建模的目标，则最有可能的原因是网格的出站过滤器类型已设置为 `ALLOW_ALL`。当出站筛选器设置为 `ALLOW_ALL` 时，您的应用程序发出的与建模目的地（后端）不匹配的出站请求将发送到该应用程序设置的目标 IP 地址。

如果要禁止流量前往未在网格中建模的目的地，请考虑将出站过滤器值设置为 `DROP_ALL`。

> **Note**
>
> 设置网格出站过滤器值会影响网格中的所有虚拟节点。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

当虚拟服务有虚拟节点提供程序 503 时，某些请求会失败，并显示 HTTP 状态码

症状

您的应用程序的一部分请求无法发送到使用虚拟节点提供商而非虚拟路由器提供商的虚拟服务后端。使用虚拟路由器提供商提供虚拟服务时，请求不会失败。

解决方案

这是一个已知问题。有关更多信息，请参阅上针对虚拟服务的虚拟节点提供商的重试策略。 GitHub使用虚拟节点作为虚拟服务提供者时，您无法指定希望虚拟服务的客户端使用的默认重试策略。相比之下，虚拟路由器提供商允许指定重试策略，因为它们是子路由资源的属性。

要减少向虚拟节点提供商请求失败的情况，请改用虚拟路由器提供商，并在其路由上指定重试策略。有关减少应用程序请求失败的其他方法，请参阅App Mesh 最佳实践。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法连接到 Amazon EFS 文件系统

症状

使用 Amazon EFS 文件系统作为卷配置 Amazon ECS 任务时，该任务无法启动，并出现以下错误。

```
ResourceInitializationError: failed to invoke EFS utils commands to set up EFS volumes:
 stderr: mount.nfs4: Connection refused : unsuccessful EFS utils command execution;
 code: 32
```

解决方案

这是一个已知问题。之所以出现此错误，是因为与 Amazon EFS 的 NFS 连接发生在任务中的任何容器启动之前。此流量通过代理配置路由到 Envoy，此时不会运行。由于启动顺序的原因，NFS 客户端无法连接到 Amazon EFS 文件系统，任务也无法启动。要解决此问题，请在 Amazon ECS 任务定义的代理配置中`EgressIgnoredPorts`设置的值列表中添加端口 2049。有关更多信息，请参阅 代理配置。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

连接成功服务，但传入的请求未出现在 Envoy 的访问日志、跟踪记录或指标中

症状

尽管您的应用程序可以连接其他应用程序并向其发送请求，但您要么无法在访问日志中看到传入的请求，要么无法在 Envoy 代理的跟踪信息中看到传入的请求。

解决方案

这是一个已知问题。如需更多信息，请参阅 Github 上的 iptables 规则设置问题。Envoy 代理仅拦截其相应虚拟节点正在侦听的端口的入站流量。对任何其他端口的请求都将绕过 Envoy 代理，直接访问其背后的服务。为了让 Envoy 代理拦截服务的入站流量，您需要将虚拟节点和服务设置为在同一端口上侦听。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

在容器级别设置HTTP_PROXY/HTTPS_PROXY环境变量不如预期的那样起作用。

症状

当 HTTP_PROXY/HTTPS_PROXY 被设置为环境变量时：

- 任务定义中的应用程序容器启用了 App Mesh，发送到 App Mesh 服务命名空间的请求将从 Envoy sidecar `HTTP 500` 获得错误响应。
- 任务定义中的 Envoy 容器启用了 App Mesh，从 Envoy sidecar 发出的请求将不会通过HTTP/HTTPS代理服务器，环境变量也无法工作。

解决方案

对于应用程序容器：

App Mesh 的功能是让任务中的流量通过 Envoy 代理。HTTP_PROXY/HTTPS_PROXY配置通过将容器流量配置为通过不同的外部代理来覆盖此行为。Envoy 仍会拦截流量，但不支持使用外部代理代理代理网格流量。

如果要代理所有非网格流量，请设置 NO_PROXY 为包括网格的 CIDR/命名空间、本地主机和凭证的端点，如下例所示。

```
NO_PROXY=localhost,127.0.0.1,169.254.169.254,169.254.170.2,10.0.0.0/16
```

对于 Envoy 容器：

Envoy 不支持通用代理。我们不建议设置这些变量。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

即使设置了路由的超时时间，上游请求也会超时。

症状

您为以下各项定义了超时：

- 路由，但您仍然收到上游请求超时错误。
- 虚拟节点侦听器以及路由的超时和重试超时，但您仍会收到上游请求超时错误。

解决方案

要使大于 15 秒的高延迟请求成功完成，您需要在路由和虚拟节点侦听器级别指定超时时间。

如果您指定的路由超时大于默认的 15 秒，请确保同时为所有参与的虚拟节点的侦听器指定了超时时间。但是，如果您将超时时间缩短到低于默认值的值，则可以选择更新虚拟节点的超时时间。有关设置虚拟节点和路由时选项的更多信息，请参阅虚拟节点和路由。

如果您指定了重试策略，则您为请求超时指定的持续时间应始终大于或等于重试超时乘以您在重试策略中定义的最大重试次数。这样，您的请求就可以成功完成所有重试操作。有关更多信息，请参阅路线。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

Envoy 回复了 HTTP 错误的请求。

症状

对于通过网络负载均衡器 (NLB) 发送的所有请求，Envoy 用 HTTP 400 错误请求进行响应。当我们查看 Envoy 日志时，我们会看到：

- 调试日志：

```
dispatch error: http/1.1 protocol error: HPE_INVALID_METHOD
```

- 访问日志：

```
"- - HTTP/1.1" 400 DPE 0 11 0 - "-" "-" "-" "-" "-"
```

解决方案

解决方案是在 NLB 的目标组属性上禁用代理协议版本 2 (PPv2)。

到目前为止，使用 App Mesh 控制面板运行的虚拟网关和虚拟节点 Envoy 不支持 PPv2。如果您在 Kubernetes 上使用 AWS 负载均衡器控制器部署 NLB，请通过将以下属性设置为 `false` 来禁用 PPv2：

```
service.beta.kubernetes.io/aws-load-balancer-target-group-attributes:
 proxy_protocol_v2.enabled
```

有关 NLB 资源属性的更多详细信息，请参阅AWS 负载均衡器控制器注释。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法正确配置超时。

症状

即使在虚拟节点侦听器上配置了超时时间，在通往虚拟节点后端的路由上配置了超时，您的请求仍会在 15 秒内超时。

解决方案

确保后端列表中包含正确的虚拟服务。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

App Mesh 缩放疑难解答

本主题详细介绍了您在扩展 App Mesh 时可能遇到的常见问题。

将虚拟节点/虚拟网关的副本扩展到 50 个以上时，连接失败且容器运行状况检查失败

症状

当您将虚拟节点/虚拟网关的副本数量（例如 Amazon ECS 任务、Kubernetes 容器组 (pod) 或 Amazon EC2 实例）扩展到 50 以上时，Envoy 对新的和当前正在运行的 Envoy 的容器运行状况检查开始失败。向虚拟节点/虚拟网关发送流量的下游应用程序开始看到请求失败并显示 HTTP 状态码 503。

解决方案

App Mesh 对每个虚拟节点/虚拟网关的 Envoy 数量的默认配额为 50。当正在运行的 Envoy 数量超过此配额时，新的和当前正在运行的 Envoy 将无法通过 gRPC 状态码 8(RESOURCE_EXHAUSTED)连接到 App Mesh 的 Envoy 管理服务。这个配额可以提高。有关更多信息，请参见 App Mesh 服务限额。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

503当虚拟服务后端水平向外扩展或向内扩展时，请求会失败

症状

当后端虚拟服务横向扩展或向内扩展时，来自下游应用程序的请求会失败，并 HTTP 503 显示状态码。

解决方案

App Mesh 推荐了几种方法来缓解故障情况，同时水平扩展应用程序。有关如何防止这些故障的详细信息，请参阅 App Mesh 最佳实践。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

在负载增加的情况下，Envoy 容器因段错误而崩溃

症状

在高流量负载下，Envoy 代理由于分段错误（Linux 退出代码 139）而崩溃。Envoy 进程日志包含如下语句。

```
Caught Segmentation fault, suspect faulting address 0x0"
```

解决方案

Envoy 代理可能违反了操作系统的默认 nofile ulimit，即一个进程一次可以打开的文件数量的限制。这种漏洞是由于流量导致了更多的连接，从而消耗了额外的操作系统插槽。要解决此问题，请增加主机操作系统上的 ulimit nofile 值。如果您使用的是 Amazon ECS，则可以通过任务定义的资源限制设置上的 Ulimit 设置来更改此限制。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

默认资源的增加未反映在服务限制中

症状

在增加 App Mesh 资源的默认限制后，当您查看服务限制时，新值不会反映出来。

解决方案

虽然目前未显示新的限制，但客户仍然可以行使这些限制。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

由于大量的运行状况检查调用，应用程序崩溃。

症状

为虚拟节点启用主动运行状况检查后，运行状况检查调用的数量会增加。由于向应用程序发出的运行状况检查调用的数量大大增加，应用程序崩溃。

解决方案

启用主动运行状况检查后，下游（客户端）的每个 Envoy 端点都会向上游集群（服务器）的每个端点发送运行状况请求，以便做出路由决策。因此，运行状况检查请求的总数将为 `number of client Envoys * number of server Envoys * active health check frequency`。

要解决此问题，请修改运行状况检查探测器的频率，这样可以减少运行状况检查探测器的总量。除了主动运行状况检查外，App Mesh 还允许将异常值检测配置为被动运行状况检查的手段。使用异常值检测来配置何时根据连续`5xx`响应移除特定主机。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

App Mesh 可观测性故障排除

本主题详细介绍了您在 App Mesh 可观测性方面可能遇到的常见问题。

看不到我的应用程序的 AWS X-Ray 痕迹

症状

您在 App Mesh 中的应用程序未在 X-Ray 控制台或 API 中显示 X-Ray 跟踪信息。

解决方案

要在 App Mesh 中使用 X-Ray，必须正确配置组件，以实现应用程序、sidecar 容器和 X-Ray 服务之间的通信。请执行以下步骤确认 X-Ray 已正确设置：

- 确保 App Mesh 虚拟节点侦听器协议未设置为 TCP。
- 确保与您的应用程序一起部署的 X-Ray 容器公开 UDP 端口2000并以用户身份运行。1337有关更多信息，请参阅上的 Amazon ECS X-Ray 示例 GitHub。
- 确保 Envoy 容器已启用追踪。如果您使用的是 App Mesh Envoy 镜像，则可以通过将环境变量设置为将ENABLE_ENVOY_XRAY_TRACING环境变量设置为1 并将 XRAY_DAEMON_PORT环境变量设置为2000来启用 X-Ray。
- 如果您使用某个特定语言的 SDK 在应用程序代码中对 X-Ray 进行了检测，请按照您的语言指南确保其配置正确。
- 如果前面的所有项目都配置正确，请查看 X-Ray 容器日志中是否存在错误，并按照疑难解答中的指导进行操作AWS X-Ray。有关在 App Mesh 中集成 X-Ray 的更详细说明，请参阅将 X-Ray 与 App Mesh 集成。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法在 Amazon 指标中查看我的应用程序的 Envoy CloudWatch 指标

症状

你在 App Mesh 中的应用程序没有向指标发出 Envoy 代理生成的 CloudWatch指标。

解决方案

在 App Mesh 中使用 CloudWatch 指标时，必须正确配置多个组件，以实现您的 Envoy 代理、CloudWatch 代理 sidecar 和 CloudWatch 指标服务之间的通信。请执行以下步骤确认 Envoy 代理的 CloudWatch 指标设置正确：

- 确保您使用的是 App Mesh 的 CloudWatch 代理映像。有关更多信息，请参阅上的 App Mesh CloudWatch 代理 GitHub。
- 确保按照平台特定的使用说明正确配置了 App Mesh 的 CloudWatch 代理。有关更多信息，请参阅上的 App Mesh CloudWatch 代理 GitHub。
- 如果前面的所有项目都配置正确，请查看 CloudWatch 代理容器日志中是否存在错误，并按照 CloudWatch 代理故障排除中提供的指导进行操作。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法为 AWS X-Ray 跟踪配置自定义采样规则

症状

您的应用程序正在使用 X-Ray 跟踪，但您无法为跟踪配置采样规则。

解决方案

由于 App Mesh Envoy 目前不支持动态 X-Ray 采样配置，因此可以使用以下解决方法。

如果您的 Envoy 版本为 `1.19.1` 或更高版本，您可以选择以下选项。

- 要仅设置采样率，请使用 Envoy 容器上的 `XRAY_SAMPLING_RATE` 环境变量。该值应指定为介于 `0` 和之间的小数 `1.00` (100%)。有关更多信息，请参阅 AWS X-Ray 变量。
- 要为 X-Ray tracer 配置本地化的自定义采样规则，请使用 `XRAY_SAMPLING_RULE_MANIFEST` 环境变量指定 Envoy 容器文件系统中的文件路径。有关更多信息，请参阅《AWS X-Ray 开发人员指南》中的采样规则。

如果您的 Envoy 版本早于 `1.19.1`，请执行以下操作。

- 使用 `ENVOY_TRACING_CFG_FILE` 环境变量来更改采样率。有关更多信息，请参阅 Envoy 配置变量。指定自定义跟踪配置并定义本地采样规则。有关更多信息，请参阅 Envoy Cay 配置。
- `ENVOY_TRACING_CFG_FILE` 环境变量的自定义跟踪配置示例：

```
tracing:
   http:
     name: envoy.tracers.xray
     typedConfig:
       "@type": type.googleapis.com/envoy.config.trace.v3.XRayConfig
       segmentName: foo/bar
       segmentFields:
         origin: AWS::AppMesh::Proxy
         aws:
           app_mesh:
             mesh_name: foo
             virtual_node_name: bar
       daemonEndpoint:
             protocol: UDP
             address: 127.0.0.1
```

```
            portValue: 2000
      samplingRuleManifest:
            filename: /tmp/sampling-rules.json
```

- 有关 samplingRuleManifest 属性中采样规则清单配置的详细信息，请参阅配置 X-Ray SDK for Go。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

App Mesh 安全故障排除

本主题详细介绍了您在使用 App Mesh 安全时可能遇到的常见问题。

无法通过 TLS 客户端策略连接到后端虚拟服务

症状

向虚拟节点中的虚拟服务后端添加 TLS 客户端策略时，与该后端的连接会失败。尝试向后端服务发送流量时，请求失败并显示 HTTP 503 响应代码和错误消息：upstream connect error or disconnect/reset before headers. reset reason: connection failure。

解决方案

为确定问题的根本原因，我们建议使用 Envoy 代理进程日志来帮助您诊断问题。有关更多信息，请参阅 在预生产环境中启用 Envoy 调试日志记录。使用以下列表来确定连接失败的原因：

- 排除 无法连接到虚拟服务后端 中提到的错误，确保与后端连接成功。
- 在 Envoy 进程日志中，查找以下错误（记录在调试级别）。

```
TLS error: 268435581:SSL routines:OPENSSL_internal:CERTIFICATE_VERIFY_FAILED
```

  该错误可能是由以下一个或多个问题导致的：

  - 该证书未由 TLS 客户端策略信任包中定义的证书颁发机构之一签名。
  - 证书不再有效（已过期）。
  - 主题备用名称 (SAN) 与请求的 DNS 主机名不匹配。
  - 确保后端服务提供的证书有效，由您的 TLS 客户端策略信任包中的一个证书颁发机构签名，并且符合中定义的标准 传输层安全性协议 (TLS)。

- 如果您收到的错误与下面的错误类似，则表示该请求正在绕过 Envoy 代理并直接到达应用程序。发送流量时，Envoy 上的统计数据不会发生变化，这表明 Envoy 不在解密流量的路上。在虚拟节点的代理配置中，确保 `AppPorts` 包含应用程序正在侦听的正确值。

```
upstream connect error or disconnect/reset before headers. reset reason:
 connection failure, transport failure reason: TLS error: 268435703:SSL
 routines:OPENSSL_internal:WRONG_VERSION_NUMBER
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。如果您认为自己发现了安全漏洞或对 App Mesh 的安全性有疑问，请查看AWS 漏洞报告指南。

应用程序源自 TLS 时无法连接到后端虚拟服务

症状

从应用程序而不是从 Envoy 代理发起 TLS 会话时，与后端虚拟服务的连接会失败。

解决方案

这是一个已知问题。有关更多信息，请参阅功能请求：下游应用程序和上游代理之间的 TLS 协商 GitHub 问题。在 App Mesh 中，目前支持 Envoy 代理发起 TLS，但不支持应用程序发起 TLS。要在 Envoy 上使用 TLS 发起支持，请在应用程序中禁用 TLS 起源。这样，Envoy 可读取出站请求标头，并通过 TLS 会话将请求转发到相应的目的地。有关更多信息，请参见 传输层安全性协议 (TLS)。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。如果您认为自己发现了安全漏洞或对 App Mesh 的安全性有疑问，请查看AWS 漏洞报告指南。

无法断言 Envoy 代理之间的连接正在使用 TLS

症状

您的应用程序已在虚拟节点或虚拟网关侦听器上启用 TLS 终止，或者在后端 TLS 客户端策略上启用 TLS 启动，但是您无法断言 Envoy 代理之间的连接是通过 TLS 协商的会话进行的。

解决方案

本解析中定义的步骤使用了 Envoy 管理界面和 Envoy 统计信息。有关配置这些内容的帮助，请参阅 启用 Envoy 代理管理界面 和 启用 Envoy DogStats D 集成以进行指标卸载。为简单起见，以下统计示例使用了管理界面。

- 对于执行 TLS 终止的 Envoy 代理：

- 使用以下命令确保已在 Envoy 配置中引导 TLS 证书。

```
curl http://my-app.default.svc.cluster.local:9901/certs
```

在返回的输出中，您应看到至少一个条目 `certificates[].cert_chain` 指定 TLS 端点。

- 确保代理侦听器的成功入站连接数与 SSL 握手次数加上重复使用的 SSL 会话数完全相同，如以下示例命令和输出所示。

```
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep
 "listener.0.0.0.0_15000" | grep downstream_cx_total
listener.0.0.0.0_15000.downstream_cx_total: 11
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep
 "listener.0.0.0.0_15000" | grep ssl.connection_error
listener.0.0.0.0_15000.ssl.connection_error: 1
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep
 "listener.0.0.0.0_15000" | grep ssl.handshake
listener.0.0.0.0_15000.ssl.handshake: 9
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep
 "listener.0.0.0.0_15000" | grep ssl.session_reused
listener.0.0.0.0_15000.ssl.session_reused: 1
# Total CX (11) - SSL Connection Errors (1) == SSL Handshakes (9) + SSL Sessions
 Re-used (1)
```

- 对于执行 TLS 发起的 Envoy 代理：
  - 使用以下命令确保已在 Envoy 配置中引导 TLS 信任库。

```
curl http://my-app.default.svc.cluster.local:9901/certs
```

您应该在 `certificates[].ca_certs` 下方看到至少一个条目，显示在发起 TLS 期间用于验证后端证书的证书。

- 确保成功连接到后端集群的出站连接数与 SSL 握手次数加上重复使用的 SSL 会话数完全相同，如以下示例命令和输出所示。

```
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "virtual-node-
name" | grep upstream_cx_total
cluster.cds_egress_mesh-name_virtual-node-name_protocol_port.upstream_cx_total: 11
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "virtual-node-
name" | grep ssl.connection_error
cluster.cds_egress_mesh-name_virtual-node-name_protocol_port.ssl.connection_error:
 1
```

```
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "virtual-node-
name" | grep ssl.handshake
cluster.cds_egress_mesh-name_virtual-node-name_protocol_port.ssl.handshake: 9
curl -s http://my-app.default.svc.cluster.local:9901/stats | grep "virtual-node-
name" | grep ssl.session_reused
cluster.cds_egress_mesh-name_virtual-node-name_protocol_port.ssl.session_reused: 1
# Total CX (11) - SSL Connection Errors (1) == SSL Handshakes (9) + SSL Sessions
 Re-used (1)
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。如果您认为自己发现了安全漏洞或对 App Mesh 的安全性有疑问，请查看AWS 漏洞报告指南。

使用 Elastic Load Balancing 排除

症状

尝试配置应用程序负载均衡器或网络负载均衡器以加密虚拟节点的流量时，连接和负载均衡器运行状况检查可能会失败。

解决方案

为确定问题的根本原因，您需要检查以下内容：

- 对于执行 TLS 终止的 Envoy 代理，您需要排除任何配置错误。然后，遵循 无法通过 TLS 客户端策略连接到后端虚拟服务 中的步骤。
- 对于负载均衡器，您需要查看负载均衡器的配置 TargetGroup：
  - 确保该 TargetGroup 端口与虚拟节点定义的侦听器端口相匹配。
  - 对于通过 HTTP 向您的服务发起 TLS 连接的应用程序负载均衡器，请确保将 TargetGroup 协议设置为 HTTPS。如果正在使用运行状况检查，请确保将 HealthCheckProtocol 设置为 HTTPS。
  - 对于通过 TCP 向您的服务发起 TLS 连接的网络负载均衡器，请确保将 TargetGroup 协议设置为 TLS。如果正在使用运行状况检查，请确保将 HealthCheckProtocol 设置为 TCP。

> **Note**
>
> 任何更新 TargetGroup 都需要更改 TargetGroup 名称。

正确配置后，您的负载均衡器应使用提供给 Envoy 代理的证书为您的服务提供安全连接。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。如果您认为自己发现了安全漏洞或对 App Mesh 的安全性有疑问，请查看AWS 漏洞报告指南。

App Mesh Kubernetes 故障排除

本主题详细介绍了在 Kubernetes 中使用 App Mesh 时可能遇到的常见问题。

在 Kubernetes 中创建的应用网格资源无法在 App Mesh 中找到

症状

您已经使用 Kubernetes 自定义资源定义 (CRD) 创建了 App Mesh 资源，但是当您使用 AWS Management Console 或 API 时，您创建的资源在应用网格中不可见。

解决方案

可能的原因是 App Mesh 的 Kubernetes 控制器出现错误。有关更多信息，请参阅上的 “故障排除” GitHub。检查控制器日志中是否存在任何表明控制器无法创建任何资源的错误或警告。

```
kubectl logs -n appmesh-system -f \
    $(kubectl get pods -n appmesh-system -o name | grep controller)
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

注入 Envoy sidecar 后，容器组 (pod) 无法进行就绪和活跃度检查

症状

您的应用程序的 pod 之前已成功运行，但是在 Envoy sidecar 注入容器组 (pod) 后，就绪和活跃度检查开始失败。

解决方案

确保注入到容器组 (pod) 的 Envoy 容器已通过 App Mesh 的 Envoy 管理服务进行引导。您可以通过参考 Envoy 已断开与 App Mesh Envoy 管理服务的连接，并显示错误 中的错误代码来验证任何错误。您可以使用以下命令检查相关容器组 (pod) 的 Envoy 日志。

```
kubectl logs -n appmesh-system -f \
    $(kubectl get pods -n appmesh-system -o name | grep controller) \
    | grep "gRPC config stream closed"
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

容器组 (pod) 未注册或取消注册为实例 AWS Cloud Map

症状

您的 Kubernetes 容器组 (pod) 并未 AWS Cloud Map 作为其生命周期的一部分在中注册或注销。容器组 (pod) 可能成功启动并准备好提供流量，但无法接收任何流量。当容器组 (pod) 终止时，客户端仍可能保留其 IP 地址并尝试向其发送流量，但失败了。

解决方案

这是一个已知问题。有关更多信息，请参阅 Pod 无法在 Kubernetes 中自动注册/ 注销，出现问题。AWS Cloud Map GitHub 由于容器组 (pod)、App Mesh 虚拟节点和 AWS Cloud Map 资源之间的关系，Kubernetes 的 App Mesh 控制器可能会变得不同步并丢失资源。例如，如果虚拟节点资源在终止其关联的容器组 (pod) 之前从 Kubernetes 中删除，就会发生这种情况。

要缓解此问题，请执行以下操作：

- 确保您运行的是适用于 Kubernetes 的最新版本的 App Mesh 控制器。
- 确保虚拟节点定义中的 AWS Cloud Map `namespaceName` 和 `serviceName` 正确。
- 在删除虚拟节点定义之前，请务必删除所有关联的容器组 (pod)。如果您需要帮助来确定哪些容器组 (pod) 与虚拟节点相关联，请参阅 无法确定 App Mesh 资源的容器组 (pod) 在何处运行。
- 如果问题仍然存在，请运行以下命令检查控制器日志中是否存在可能有助于揭示潜在问题的错误。

```
kubectl logs -n appmesh-system \
    $(kubectl get pods -n appmesh-system -o name | grep appmesh-controller)
```

- 考虑使用以下命令重启控制器容器组 (pod)。这可能会解决同步问题。

```
kubectl delete -n appmesh-system \
    $(kubectl get pods -n appmesh-system -o name | grep appmesh-controller)
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法确定 App Mesh 资源的容器组 (pod) 在何处运行

症状

当您在 Kubernetes 集群上运行 App Mesh 时，操作员无法确定给定的 App Mesh 资源的工作负载或容器在哪里运行。

解决方案

Kubernetes 容器组 (pod) 资源使用与其关联的网格和虚拟节点进行注释。您可以使用以下命令查询哪些容器组 (pod) 正在为给定的虚拟节点名称运行。

```
kubectl get pods --all-namespaces -o json | \
    jq '.items[] | { metadata } | select(.metadata.annotations."appmesh.k8s.aws/
virtualNode" == "virtual-node-name")'
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

无法确定容器组 (pod) 以哪个 App Mesh 资源运行

症状

在 Kubernetes 集群上运行 App Mesh 时，操作员无法确定给定的容器组 (pod) 以什么形式运行 App Mesh 资源。

解决方案

Kubernetes 容器组 (pod) 资源使用与其关联的网格和虚拟节点进行注释。您可以使用以下命令直接查询容器组 (pod) 来输出网格和虚拟节点的名称。

```
kubectl get pod pod-name -n namespace -o json | \
    jq '{ "mesh": .metadata.annotations."appmesh.k8s.aws/mesh",
 "virtualNode": .metadata.annotations."appmesh.k8s.aws/virtualNode" }'
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

禁用 imdsV1 后，客户 Envoy 无法与 App Mesh Envoy 管理服务通信

症状

禁用 IMDSv1 后，客户 Envoy 无法与 App Mesh 控制面板（Envoy 管理服务）通信。v1.24.0.0-prod之前的 App Mesh Envoy 版本不提供支持 IMDSv2。

解决方案

要解决这个问题，你可以做这三件事之一。

- 升级到 App Mesh Envoy 版本 `v1.24.0.0-prod` 或更高版本，该版本有 `IMDSv2` 支持。
- 在运行 Envoy 的实例 `IMDSv1` 上重新启用。有关恢复的说明 `IMDSv1`，请参阅配置实例元数据选项。
- 如果您的服务在 Amazon EKS 上运行，建议使用服务账户的 IAM 角色 (IRSA) 来获取凭证。有关启用 IRSA 的说明，请参阅服务账户的 IAM 角色。

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

启用 App Mesh 并注入 Envoy 后，IRSA 无法在应用程序容器上运行

症状

在 Amazon EKS 的 App Mesh 控制器的帮助下，在 Amazon EKS 集群上启用 App Mesh 时，Envoy 和 `proxyinit` 容器将注入到应用程序容器组 (pod) 中。应用程序无法假设 `IRSA`，而是假设 `node role`。当我们描述容器组 (pod) 的详细信息时，我们会看到应用程序容器中不包含 `AWS_WEB_IDENTITY_TOKEN_FILE` 或 `AWS_ROLE_ARN` 环境变量。

解决方案

如果定义了 `AWS_WEB_IDENTITY_TOKEN_FILE` 或 `AWS_ROLE_ARN` 环境变量，则 webhook 将跳过容器组 (pod)。不要提供这两个变量中的任何一个，webhook 会为您注入它们。

```
reservedKeys := map[string]string{
        "AWS_ROLE_ARN":                "",
        "AWS_WEB_IDENTITY_TOKEN_FILE": "",
    }
    ...
    for _, env := range container.Env {
        if _, ok := reservedKeys[env.Name]; ok {
            reservedKeysDefined = true
        }
```

如果您的问题仍未解决，请考虑GitHub 提出问题或联系 Su AWSpport。

App Mesh 预览频道

App Mesh 预览频道是 us-west-2 地区提供的 App Mesh 服务的独特变体。预览频道展示了即将推出的功能，供您在开发时试用。当您使用预览频道中的功能时，您可以通过 GitHub 提供反馈，以确定这些功能的行为方式。一旦某项功能在预览频道中具有完整功能，并且完成了所有必要的集成和检查，它就会升级到正式版 App Mesh 服务。

AWS App Mesh 预览频道是一项测试版服务，所有功能均为预览版，这些条款在AWS服务条款中有定义。您对预览频道的参与受您与 AWS 达成的协议和 AWS 服务条款（尤其是通用和测试版服务参与条款）的约束，并且是保密的。

以下是关于预览频道的常见问题。

什么是预览频道？

预览频道是一个公共服务端点，允许您在新服务功能正式发布之前试用这些功能并提供反馈。预览频道的服务端点与标准生产端点是分开的。您可以使用 AWS CLI、预览频道的服务模型文件和 AWS CLI 的命令输入文件与端点进行交互。预览频道允许您在不影响当前生产基础架构的情况下尝试新功能。我们建议您向 App Mesh 团队提供反馈，以帮助确保 App Mesh 满足客户最重要的需求。在功能进入预览频道时，您对这些功能的反馈可以帮助塑造 App Mesh 的功能，以便我们能够提供尽可能优质的服务。

预览频道与正式版 App Mesh 有何不同？

下表列出了 App Mesh 服务中与预览频道不同的方面。

方面	App Mesh 制作服务	App Mesh 预览频道服务
Frontend endpoint	appmesh.us-west-2.amazonaws.com	appmesh-preview.us-west-2.amazonaws.com
Envoy management service endpoint	appmesh-envoy-management.us-west-2.amazonaws.com	appmesh-preview-envoy-management.us-west-2.amazonaws.com
CLI	AWS App Mesh list-meshes	AWS App Mesh-preview list-meshes (only available after

		adding the Preview Channel service model)
Signing name	appmesh	appmesh-preview
Service principal	appmesh.amazonaws.com	appmesh-preview.amazonaws.com

> **Note**
>
> 尽管表中的 App Mesh 生产服务示例列出了 `us-west-2` 区域，但生产服务在大多数地区都可用。有关 App Mesh 生产服务可用的所有区域的列表，请参阅AWS App Mesh 端点和配额。但是，App Mesh 预览频道服务仅在 `us-west-2` 地区可用。

如何使用预览频道中的功能？

1. 使用以下命令将包含预览频道功能的预览频道服务模型添加到 AWS CLI 中。

```
aws configure add-model \
    --service-name appmesh-preview \
    --service-model https://raw.githubusercontent.com/aws/aws-app-mesh-roadmap/
main/appmesh-preview/service-model.json
```

2. 根据该功能的 JSON 示例和AWS App Mesh用户指南中提供的说明，创建包含该功能的 JSON 文件。
3. 使用相应的 AWS CLI 命令和命令输入文件实现该功能。例如，以下命令使用 *route.json ########## ##########*。

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

4. 将 Envoy 容器添加到 Amazon ECS 任务定义、Kubernetes 容器组 (pod) 规范或 Amazon EC2 实例时，将 `APPMESH_PREVIEW = 1` 添加为配置变量。此变量使 Envoy 容器能够与预览频道端点进行通信。有关添加配置变量的更多信息，请参阅更新 Amazon ECS 中的服务、更新 Kubernetes 中的服务以及 Amazon EC2 上的服务。

我如何提供反馈？

您可以直接就 App Mesh 路线图 GitHub 存储库问题提供反馈，该问题链接自有关该功能的文档。

我需要在多长时间内就预览频道中的某项功能提供反馈？

反馈期将根据所引入功能的大小和复杂性而变。我们打算在向预览端点发布功能和将该功能发布到正式版之间留出 14 天的意见征询期。App Mesh 团队可能会延长特定功能的反馈期限。

为预览频道提供什么级别的支持？

虽然我们建议您直接就 App Mesh GitHub 路线图问题提供反馈和错误报告，但我们知道您可能有敏感数据要共享，或者您可能会发现一个您认为不值得公开披露的问题。对于这些问题，您可以直接向 App Mesh 团队发送电子邮件来联系 AWS Support 或提供反馈。

我在预览频道端点上的数据是否安全？

是。预览频道具有与标准生产端点相同的安全级别。

我的配置的可用期为多长时间？

您可以在预览频道中使用网格三十天。网格创建三十天后，您只能列出、读取或删除网格。如果您在三十天后尝试创建或更新资源，则会收到一条 `BadRequest` 异常，说明网格已存档。

我可以使用哪些工具来处理预览频道？

您可以将 AWS CLI 与预览频道服务模型文件和命令输入文件一起使用。有关如何使用功能的更多信息，请参阅 如何使用预览频道中的功能？。您不能使用 AWS CLI 命令选项、AWS Management Console、SDK 或 AWS CloudFormation 使用预览频道功能。但是，一旦某项功能发布到生产服务后，您就可以使用所有工具。

预览频道 API 会向前兼容吗？

不会。 API 可能会根据反馈进行更改。

预览频道功能是否完整？

不完整。 新的 API 对象可能未完全集成到AWS Management Console、AWS CloudFormation 或 AWS CloudTrail。随着功能在预览频道中的巩固以及接近全面上市，这些集成将最终可用。

是否有关于预览频道功能的文档？

是。预览频道功能文档包含在生产文档中。例如，如果路径资源的要素发布到预览频道，则有关如何使用这些功能的信息将出现在现有的路径资源文档中。预览频道功能被标记为仅在预览频道中可用。

我如何知道预览频道何时推出新功能？

在预览频道中引入新功能时，会在 App Mesh 文档历史记录中添加一个条目。您可以定期查看该页面或订阅 App Mesh 文档历史记录 RSS 提要。此外，您可以查看 AWS App Mesh 路线图 GitHub 存储库的问题。当预览频道服务模型 JSON 文件发布到预览频道时，会将该文件的下载链接添加到议题中。有关如何使用模型及其功能的更多信息，请参阅 如何使用预览频道中的功能？。

我怎么知道某项功能何时升级到生产服务？

App Mesh 文档中注明该功能仅在预览频道中可用的文本已删除，并在 App Mesh 文档历史记录中添加一个条目。您可以定期查看该页面或订阅 App Mesh 文档历史记录 RSS 提要。

App Mesh 服务限额

AWS App Mesh 已与服务限额集成，后者是一项 AWS 服务，可让您从中心位置查看和管理您的限额。服务限额也称为限制。有关更多信息，请参阅《服务限额用户指南》中的什么是服务限额？

可使用服务限额轻松查找所有 App Mesh 服务限额的值。

使用 AWS Management Console 查看 App Mesh 服务限额

1. 访问 https://console.aws.amazon.com/servicequotas/，打开服务限额控制台。
2. 在导航窗格中，选择 AWS 服务。
3. 从 AWS 服务列表中，搜索并选择 AWS App Mesh。

   在服务限额列表中，您可以查看服务限额名称、应用的值（如果该值可用）、AWS 默认限额以及限额值是否可调整。
4. 要查看有关服务限额的其他信息（如描述），请选择限额名称。

要请求提高限额，请参阅《服务限额用户指南》中的请求提高限额。

使用 AWS CLI 查看 App Mesh 服务限额

运行以下命令。

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

要使用 AWS CLI 进一步处理服务限额，请参阅服务限额 AWS CLI 命令参考。

App Mesh 的文档历史记录

下表描述了 AWS App Mesh 用户指南 的主要更新和新功能。我们还经常更新文档来处理发送给我们的反馈意见。

变更	说明	日期
更新了AWSAppMeshFullAccess 政策	已更新AWSAppMeshFullAccess ，允许访问TagResource 和UntagResource API。	2024 年 4 月 24 日
CloudTrail 集成文档已更新	描述 App Mesh 与之集成 CloudTrail 以记录 API 活动的文档已更新。	2024 年 3 月 28 日
更新策略	已更新AWSServiceRoleForAppMesh AWSAppMeshServiceRolePolicy 并允许访问 AWS Cloud Map DiscoverInstancesRevision API。	2023 年 10 月 12 日
App Mesh 的 VPC 端点策略支持	App Mesh 现在支持 VPC 端点策略。	2023 年 5 月 11 日
App Mesh 的多个侦听器	App Mesh 现在支持多个侦听器。	2022 年 8 月 18 日
适用于 App Mesh 的 IPv6	App Mesh 现在支持 IPv6。	2022 年 5 月 18 日
CloudTrail App Mesh Envoy 管理服务的日志支持	App Mesh 现在支持 CloudTrail 对 App Mesh Envoy 管理服务的日志支持。	2022 年 3 月 18 日
适用于 Envoy 的 App Mesh 代理	App Mesh 现在支持 Envoy 代理。	2022 年 2 月 25 日

App Mesh 的多个侦听器	（仅限 App Mesh 预览频道）您可以为 App Mesh 实施多个侦听器。	2021 年 11 月 23 日
App Mesh 的 ARM64 支持	App Mesh 现在支持 ARM64。	2021 年 11 月 19 日
App Mesh 的指标扩展	您可以实施 App Mesh 指标扩展。	2021 年 10 月 29 日
实施传入流量增强功能	您可以实施主机名和标头匹配，并对主机名和路径进行重写。	2021 年 6 月 14 日
实施双向 TLS 身份验证	您可以实施双向 TLS 身份验证。	2021 年 2 月 4 日
在 af-south-1 中进行区域启动	App Mesh 现已在 af-south-1 区域上线。	2021 年 1 月 22 日
实施双向 TLS 身份验证	（仅限 App Mesh 预览频道）您可以实施双向 TLS 身份验证。	2020 年 11 月 23 日
实施虚拟网关侦听器连接池	您可以实施虚拟网关侦听器连接池。	2020 年 11 月 5 日
实施虚拟节点侦听器连接池和异常值检测	您可以实施虚拟节点侦听器连接池和异常值检测。	2020 年 11 月 5 日
在 eu-south-1 中进行区域启动	现已在 eu-south-1 区域中提供 App Mesh。	2020 年 10 月 21 日
实施虚拟网关侦听器连接池	（仅限 App Mesh 预览频道）您可以实施虚拟网关侦听器连接池。	2020 年 9 月 28 日
实施虚拟节点侦听器连接池和异常值检测	（仅限 App Mesh 预览频道）您可以实施虚拟节点侦听器连接池和异常值检测。	2020 年 9 月 28 日

为网格入站创建虚拟网关和网关路由	使网格之外的资源能够与网格内部的资源通信。	2020 年 7 月 10 日
使用适用于 Kubernetes 的 App Mesh 控制器，在 Kubernetes 中创建和管理 App Mesh 资源	您可以在 Kubernetes 中创建和管理 App Mesh 资源。控制器还会自动在您部署的容器组 (pod) 中注入 Envoy 代理和 Init 容器。	2020 年 6 月 18 日
向虚拟节点侦听器和路由添加超时值	您可以向虚拟节点侦听器和路由添加超时值。	2020 年 6 月 18 日
向虚拟节点侦听器添加超时值	（仅限 App Mesh 预览频道）您可以向虚拟节点侦听器添加超时值。	2020 年 5 月 29 日
为入站网格创建虚拟网关	（仅限 App Mesh 预览频道）允许网格外部的资源与网格内部的资源进行通信。	2020 年 4 月 8 日
TLS 加密	（仅限 App Mesh 预览频道）使用来自 AWS Private Certificate Authority 或您自己的证书颁发机构的证书，使用 TLS 加密虚拟节点之间的通信。	2020 年 1 月 17 日
与其他账号共享网格	（仅限 App Mesh 预览频道）您可以与其他账户共享网格。任何账户创建的资源都可以与网格中的其他资源通信。	2020 年 1 月 17 日
向路由添加超时值	（仅限 App Mesh 预览频道）您可以向路由添加超时值。	2020 年 1 月 17 日
在前 AWS 哨基地创建 App Mesh 代理	你可以在 O AWS utpost 上创建 App Mesh Envoy 代理。	2019 年 12 月 3 日

对路由、虚拟路由器和虚拟节点的 HTTP/2 和 gRPC 的支持	您可以路由使用 HTTP/2 和 gRPC 协议的流量。您还可以将这些协议的侦听器添加到虚拟节点和虚拟路由器中。	2019 年 10 月 25 日
重试策略	重试策略使客户端可以保护自己免受间歇性网络故障或间歇性服务器端故障的影响。您可以向路由添加重试逻辑。	2019 年 9 月 10 日
TLS 加密	（仅限 App Mesh 预览频道）使用 TLS 加密虚拟节点之间的通信。	2019 年 9 月 6 日
基于 HTTP 标头的路由	根据请求中是否存在 HTTP 标头及其值来路由流量。	2019 年 8 月 15 日
App Mesh 预览频道上线	App Mesh 预览频道是 App Mesh 服务的一个独特变体。预览频道展示了即将推出的功能，供您在开发时试用。当你使用预览频道中的功能时，你可以通过提供反馈 GitHub 来塑造这些功能的行为方式。	2019 年 7 月 19 日
App Mesh 接口 VPC 端点 (AWS PrivateLink)	将 App Mesh 配置为使用接口 VPC 端点以改善 VPC 的安全状况。接口端点由一项技术提供支持 AWS PrivateLink，该技术使您能够使用私有 IP 地址私密访问 App Mesh API。PrivateLink 将您的 VPC 和 App Mesh 之间的所有网络流量限制到亚马逊网络。	2019 年 6 月 14 日

添加 AWS Cloud Map 为虚拟节点服务发现方法	您可以指定 DNS 或 AWS Cloud Map 作为虚拟节点服务发现方法。要 AWS Cloud Map 用于服务发现，您的账户必须具有 App Mesh 服务相关角色。	2019 年 6 月 13 日
在 Kubernetes 中自动创建 App Mesh 资源	在 Kubernetes 中创建资源时，可以自动创建 App Mesh 资源并将 App Mesh sidecar 容器映像添加到 Kubernetes 部署中。	2019 年 6 月 11 日
App Mesh 正式上市	App Mesh 服务现在可普遍用于生产环境。	2019 年 3 月 27 日
App Mesh API 更新	已更新 App Mesh API，以提高可用性。有关更多信息，请参阅 [功能] 向虚拟路由器添加侦听器和指向端口不匹配的目标虚拟节点的 [错误] 路由 Blackhole。	2019 年 3 月 7 日
App Mesh 初始版本	服务公开预览版初始文档	2018 年 11 月 28 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。