ALIBABA CLOUD

阿里云

服务网格

ASM 服务网格公共云合集

文档版本：20220707

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.网格管理

1.1. 使用Terraform管理ASM实例

Terraform是HashiCorp公司提供的一种开源的云上资源编排工具，用于安全高效地预览，配置和管理云基础架构和资源，帮助开发者自动化地创建、更新阿里云基础设施资源。本文介绍如何使用Terraform创建和删除ASM实例。

前提条件

已在本地安装和配置Terraform。具体操作，请参见在本地安装和配置Terraform。

> **说明** 为提高权限管理的灵活性和安全性，建议您创建RAM用户，并为其授予AliyunVPCFullAccess和AliyunASMFullAccess的权限。

背景信息

关于Terraform的详细介绍，请参见terraform。

创建ASM实例

1. 在本地创建名为*main.tf*的配置文件。
   - 若您没有专有网络和虚拟交换机，您需要使用以下内容创建*main.tf*文件。

```
terraform {
  required_providers {
    alicloud = {
      source = "aliyun/alicloud"
      version = "1.166.0"
    }
  }
}
variable "k8s_name_prefix" {
  description = "The name prefix used to create Alibaba Cloud Service Mesh (ASM)."
  default     = "tf-asm"
}
resource "random_uuid" "this" {}
# 默认的资源名称及配置。
locals {
  # 服务网格名称。
  mesh_name = substr(join("-", [var.k8s_name_prefix, random_uuid.this.result]), 0, 63
)
  # 服务网格的规格，可以选择两种规格：enterprise：企业版，ultimate：旗舰版。
  mesh_spec = "enterprise"
  # 专有网络名称。
  new_vpc_name = "vpc-for-${local.mesh_name}"
  # 虚拟交换机名称。
  new_vsw_name = "vsw-for-${local.mesh_name}"
}
# 可以创建虚拟交换机的可用区。
data "alicloud_zones" "default" {
  available_resource_creation = "VSwitch"
```

```
}
# 专有网络。
resource "alicloud_vpc" "default" {
  vpc_name = local.new_vpc_name
}
# 虚拟交换机。
resource "alicloud_vswitch" "default" {
  vpc_id       = alicloud_vpc.default.id
  cidr_block   = cidrsubnet(alicloud_vpc.default.cidr_block, 8, 2)
  zone_id      = data.alicloud_zones.default.zones.0.id
  vswitch_name = local.new_vsw_name
}
# 查询可以创建的服务网格版本。
data "alicloud_service_mesh_versions" "default" {
  edition = "Pro"
}
# 选择可创建的第一个版本作为创建新服务网格使用的版本。
locals {
  mesh_version = split(":", data.alicloud_service_mesh_versions.default.ids[0])[1]
}
# 服务网格ASM实例。
resource "alicloud_service_mesh_service_mesh" "default" {
  # 服务网格名称。
  service_mesh_name = local.mesh_name
  # 服务网格的网络配置。
  network {
    # 专有网络ID，
    vpc_id        = alicloud_vpc.default.id
    # 虚拟交换机ID。
    vswitche_list = [alicloud_vswitch.default.id]
  }
  # 服务网格的版本。
  version = local.mesh_version
  # 服务网格的API Server和Pilot负载均衡配置。
  load_balancer {
    # 是否使用EIP暴露服务网格API Server负载均衡。
    api_server_public_eip = true
  }
  # 服务网格的规格，可以选择两种规格：enterprise：企业版，ultimate：旗舰版。
  cluster_spec = local.mesh_spec
}
```

在main.tf文件中根据实际情况自定义以下参数，其他参数值会使用OpenAPI自动拉取获取。

参数	描述
mesh_name	自定义服务网格名称。
mesh_spec	设置服务网格规格，可选： ■ *enterprise*：企业版。 ■ *ultimate*：旗舰版。
new_vpc_name	自定义专有网络名称。

参数	描述
new_vsw_name	自定义虚拟交换机名称。
api_server_public_eip	是否使用EIP暴露服务网格API Server负载均衡，可选： ■ *true*：使用EIP暴露服务网格API Server负载均衡。 ■ *false*：不使用EIP暴露服务网格API Server负载均衡。

- 若您已有专有网络和虚拟交换机，您需要使用以下内容创建*main.tf*文件。

> **注意** 专有网络、虚拟交换机需要和Terraform的身份认证信息处于同一地域，否则将识别不到专有网络和虚拟交换机。

```
terraform {
  required_providers {
    alicloud = {
      source = "aliyun/alicloud"
      version = "1.166.0"
    }
  }
}
variable "asm_name_prefix" {
  description = "The name prefix used to create Alibaba Cloud Service Mesh (ASM)."
  default     = "tf-asm"
}
resource "random_uuid" "this" {}
# 默认的资源名称及配置。
locals {
  # 服务网格名称。
  mesh_name = substr(join("-", [var.asm_name_prefix, random_uuid.this.result]), 0, 63
)
  # 服务网格的规格，可以选择两种规格：enterprise：企业版，ultimate：旗舰版。
  mesh_spec = "enterprise"
  # 已有专有网络名称。
  vpc_name = "vpc-luying-hangzhou1"
  # 已有虚拟交换机名称。
  vsw_name = "vsw-luying-hangzhou1"
}
# 可以创建虚拟交换机的可用区。
data "alicloud_zones" "default" {
  available_resource_creation = "VSwitch"
}
# 专有网络。
data "alicloud_vpcs" "default" {
  name_regex = local.vpc_name # 已经存在的专有网络名称。
}
# 虚拟交换机。
data "alicloud_vswitches" "default" {
  vpc_id = data.alicloud_vpcs.default.ids[0]
}
```

```
locals {
  exist_vswitch_ids = [for vsw in data.alicloud_vswitches.default.vswitches : vsw.id
if vsw.name == local.vsw_name]
}
# 查询可以创建的服务网格版本。
data "alicloud_service_mesh_versions" "default" {
  edition = "Pro"
}
# 选择可创建的第一个版本作为创建新服务网格使用的版本。
locals {
  mesh_version = split(":", data.alicloud_service_mesh_versions.default.ids[0])[1]
}
# 服务网格ASM实例。
resource "alicloud_service_mesh_service_mesh" "default" {
  # 服务网格名称。
  service_mesh_name = local.mesh_name
  # 服务网格的网络配置。
  network {
    # 专有网络ID。
    vpc_id        =  data.alicloud_vpcs.default.ids[0]
    # 虚拟交换机ID。
    vswitche_list = [local.exist_vswitch_ids[0]]
  }
  # 服务网格的版本。
  version = local.mesh_version
  # 服务网格的API Server和Pilot负载均衡配置。
  load_balancer {
    # 是否使用EIP暴露服务网格API Server负载均衡。
    api_server_public_eip = true
  }
  # 服务网格的规格，可以选择两种规格：enterprise：企业版，ultimate：旗舰版。
  cluster_spec = local.mesh_spec
}
```

在main.tf文件中根据实际情况自定义以下参数，其他参数值会使用OpenAPI自动拉取获取。

参数	描述
mesh_name	自定义服务网格名称。
mesh_spec	设置服务网格规格，可选： ■ *enterprise*：企业版。 ■ *ultimate*：旗舰版。
vpc_name	输入已有专有网络名称。
vsw_name	输入已有虚拟交换机名称。

参数	描述
api_server_public_eip	是否使用EIP暴露服务网格API Server负载均衡，可选： ■ *true*：使用EIP暴露服务网格API Server负载均衡。 ■ *false*：不使用EIP暴露服务网格API Server负载均衡。

2. 执行以下命令，初始化Terraform运行环境。

```
terraform init
```

预期输出：

```
Initializing the backend...
Initializing provider plugins...
- Finding aliyun/alicloud versions matching "1.166.0"...
- Finding latest version of hashicorp/random...
...
Terraform has been successfully initialized!
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

3. 执行以下命令，生成Terraform资源规划。

```
terraform plan
```

预期输出：

```
Terraform used the selected providers to generate the following execution plan. Resourc
e actions are indicated with the following symbols:
  + create
Terraform will perform the following actions:
...
Plan: 4 to add, 0 to change, 0 to destroy.
```

4. 执行以下命令，使用*main.tf*文件创建ASM实例。

```
terraform apply
```

预期输出：

```
alicloud_service_mesh_service_mesh.example: Refreshing state...
...
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
  Enter a value:
```

在Enter a value参数右侧输入*yes*，预期返回以下内容：

```
...
alicloud_service_mesh_service_mesh.default: Creating...
alicloud_service_mesh_service_mesh.default: Still creating... [10s elapsed]
...
alicloud_service_mesh_service_mesh.example: Creation complete after 2m42s [id=c96b0f153
ec264327af9536e2992d****]
Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
```

删除ASM实例

使用Terraform删除指定ASM实例时，您必须进入到与*main.tf*文件相同的目录下后，才能执行命令删除该实例。

进入与*main.tf*文件相同的目录下，执行以下命令，删除ASM实例。

```
terraform destroy
```

预期输出：

```
...
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.
  Enter a value:
```

在Enter a value参数右侧输入*yes*，预期返回以下内容：

```
alicloud_service_mesh_service_mesh.default: Destroying... [id=c96b0f153ec264327af9536e2992d
****]
...
alicloud_service_mesh_service_mesh.default: Still destroying... [id=c96b0f153ec264327af9536
e2992d****, 2m10s elapsed]
alicloud_service_mesh_service_mesh.default: Still destroying... [id=c96b0f153ec264327af9536
e2992d****, 2m20s elapsed]
...
alicloud_service_mesh_service_mesh.default: Destruction complete after 2m25s
alicloud_vswitch.default: Destroying... [id=vsw-2zedvcl5tfch3x9gj****]
alicloud_vswitch.default: Destruction complete after 6s
alicloud_vpc.default: Destroying... [id=vpc-2zej2n5jlcbn9zd9t****]
alicloud_vpc.default: Destruction complete after 5s
random_uuid.this: Destroying... [id=89f4ab20-155a-6e93-4f9a-35a2b787****]
random_uuid.this: Destruction complete after 0s
Destroy complete! Resources: 4 destroyed.
```

Terraform管理的ASM资源

ASM支持通过Terraform管理以下Resource和Data Source：

类型	名称	描述

类型	名称	描述
Resources	alicloud_service_mesh_service_mesh	管理ASM实例。
Data Sources	alicloud_service_mesh_service_meshes	列举所有的ASM实例。
	alicloud_service_mesh_versions	列举所有可用的服务网格版本。

1.2. 访问Istio资源

1.2.1. 概述

ASM支持通过数据面集群的Kubernetes API（KubeAPI）对Istio资源进行增删改查操作。不同版本的ASM实例使用集群的Kubernetes API访问Istio资源的操作不同：

- 如果您使用的ASM实例大于等于1.9.7.93，且小于等于1.12.4.50版本，使用集群的Kubernetes API访问Istio资源的具体操作，请参见使用集群Kubernetes API访问Istio资源（旧版本）。
- 如果您使用的ASM实例大于1.12.4.50版本，使用集群的Kubernetes API访问Istio资源的具体操作，请参见使用集群Kubernetes API访问Istio资源（新版本）。

1.2.2. 使用集群Kubernetes API访问Istio资源（旧版本）

ASM支持通过数据面集群的Kubernetes API（KubeAPI）对Istio资源进行增删改查操作。本文以创建和查看Istio资源为例，介绍如何使用数据面集群KubeAPI访问Istio资源。

前提条件

- 已创建ASM企业版或旗舰版实例，且ASM实例的Istio为1.9.7.93及以上版本。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。

背景信息

Kubernetes API是通过HTTP提供的基于资源的编程接口，支持通过标准HTTP谓词（POST、PUT、PATCH、DELETE、GET）检索、创建、更新和删除集群的主资源，例如Deployment、Service等。更多信息，请参见Kubernetes API。

注意事项

- 强烈建议在单集群模式下使用数据面集群KubeAPI访问Istio资源功能。如果ASM的数据平面有多个集群，则任意一个数据平面集群都可以对ASM上的Istio资源进行增删改查操作。
- 开启数据面集群KubeAPI访问Istio资源功能后，数据面集群将无法删除istio-system命名空间。如果要删除，您需要先从ASM实例中移出数据面集群。
- 删除数据平面的某一命名空间，不会删除ASM控制平面的对应命名空间，以及该命名空间下的Istio资源。
- 如果ASM控制平面有某一命名空间，但是数据平面没有此命名空间，您需要先在数据平面创建出此命名空间，然后才能在这个命名空间下对Istio资源进行增删改查操作。否则会提示以下错误信息：

```
Error from server (NotFound): error when creating "xx.yaml": namespaces "daily-01" not fo
und
```

- 如果在数据平面创建的Istio资源对应的命名空间在ASM控制平面不存在，则会在控制平面自动创建该命名空间。
- Istio资源的增删改查操作不支持缩写，需要使用资源名字的全称，例如 `virtualservice` 。

步骤一：启用数据面集群KubeAPI访问Istio资源功能

您可以通过以下两种方式来启用数据面集群KubeAPI访问Istio资源功能：

- 如果您没有创建ASM实例，您可以在创建ASM实例时选中**启用数据面集群KubeAPI访问Istio资源**来启用数据面集群KubeAPI访问Istio资源功能。具体操作，请参见创建ASM实例。
- 如果您已创建ASM实例，您可以在ASM实例的**网格信息**页面启用数据面集群KubeAPI访问Istio资源功能。本文以已创建ASM实例场景为例。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**，然后在右侧页面单击**功能设置**。
5. 在**功能设置更新**面板选中**启用数据面集群KubeAPI访问Istio资源**，然后单击**确定**。

   开启数据面集群KubeAPI访问Istio资源后，ASM会创建asm-istio-admin和asm-istio-readonly两个ClusterRole到数据面集群。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    api: asm-apiservice-apiserver
    apiserver: "true"
  name: asm-istio-admin
rules:
- apiGroups:
  - networking.istio.io
  - security.istio.io
  resources:
  - '*'
  verbs:
  - '*'
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    api: asm-apiservice-apiserver
    apiserver: "true"
  name: asm-istio-readonly
rules:
- apiGroups:
  - networking.istio.io
  - security.istio.io
  resources:
  - '*'
  verbs:
  - get
  - list
  - watch
```

步骤二：获取asm-cr-aggregation配置信息

1. 查看ASM实例ID。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在网格详情页面左侧导航栏单击**网格实例 > 基本信息**。
   在**基本信息**信息页面查看ASM实例ID。
2. 查看集群地域ID。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏单击**集群**。
   在**集群**页面查看目标集群的地域，例如您集群地域为华北2（北京），则集群地域ID为cn-beijing。
3. 查看AccessKey ID和AccessKey Secret。具体操作，请参见获取AccessKey。

步骤三：安装asm-cr-aggregation

1. 已通过kubectl连接集群。具体操作，请参见通过kubectl工具连接集群。
2. 在本地安装Helm。具体操作，请参见Helm。

> **说明** 使用kubectl连接集群后，Helm客户端会自动使用KubeConfig连接集群。

3. 下载并解压asm-cr-aggregation至本地。
4. 进入asm-cr-aggregation文件夹中，找到*values.yaml*文件，在*values.yaml*文件中补充ASM ID、集群地域ID、AccessKey ID和AccessKey Secret，然后保存*values.yaml*文件。

> **注意** 如果您的集群位于海外地域，您还需要在*values.yaml*文件中修改asm-cr-aggregation镜像地址的地域为集群所在的地域，例如您的集群位于硅谷，您需要将 `registry.cn-hangzhou.aliyuncs.com/acs/asm-craggregation-apiservice` 修改为 `registry.cn-us-west-1.aliyuncs.com/acs/asm-craggregation-apiservice` 。

5. 执行以下命令，安装asm-cr-aggregation。

```
helm install -f values.yaml asm-cr-aggregation ./
```

6. 验证asm-cr-aggregation是否安装成功。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页面左侧导航栏选择**应用 > Helm**。
   在**Helm**页面可以看到asm-cr-aggregation，说明asm-cr-aggregation安装成功。

步骤四：授予RAM用户权限

使用数据面集群Kubernetes API访问Istio资源之前，您的账号需要拥有在数据面集群访问Istio资源的权限和ASM的自定义资源权限：

> **说明** 您拥有的数据面集群访问Istio资源的权限和ASM的自定义资源权限需要保持一致，即如果您拥有ASM自定义资源的读写权限，那您同时也需要拥有数据面集群访问Istio资源的读写权限。

- 您使用的账号需要拥有控制平面ASM自定义资源的操作权限，即拥有网格管理人员或者网格管理受限人员权限。具体操作，请参见授予RAM用户和RAM角色RBAC权限。

  网格管理人员拥有ASM自定义资源的读写权限，网格管理受限人员拥有ASM自定义资源的只读权限。
- 您使用的账号需要拥有在数据面集群访问Istio资源的权限，否则将访问失败。

  您可以执行以下命令，检查RAM用户是否拥有访问Istio资源的权限。

```
kubectl get VirtualService
```

  预期输出：

```
Error from server (Forbidden): virtualservices.networking.istio.io is forbidden: User "24869613637716****" cannot list resource "virtualservices" in API group "networking.istio.io" in the namespace "default"
```

  返回以上结果，说明RAM用户没用访问Istio资源的权限。您需要授予RAM用户访问Istio资源的权限，具体操作如下：

授予RAM用户访问Istio资源的只读权限。

1. 使用阿里云账号登录容器服务管理控制台。
2. 在控制台左侧导航栏单击**授权管理**。
3. 在**子账号**页签下单击目标RAM用户右侧的**管理权限**。
4. 在**集群RBAC配置**页面中单击图标，选择要授予的集群和命名空间，设置**访问权限**为**自定义**，在文本框中选择asm-istio-readonly，然后单击**下一步**。

   页面提示**授权成功**。

5. 验证RAM用户是否拥有访问Istio资源的只读权限。
   i. 执行以下命令，查看虚拟服务。

   ```
   kubectl get VirtualService
   ```

   预期输出：

   ```
   NAME            CREATED AT
   reviews-route   2021-11-15T07:09:10Z
   ```

   ii. 执行以下命令，编辑虚拟服务。

   ```
   kubectl edit VirtualService reviews-route
   ```

   预期输出：

   ```
   error: virtualservices.networking.istio.io "reviews-route" could not be patched: vi
   rtualservices.networking.istio.io "reviews-route" is forbidden: User "2299278366815
   6****" cannot patch resource "virtualservices" in API group "networking.istio.io" i
   n the namespace "default
   ```

授予RAM用户访问Istio资源的读写权限。

1. 使用阿里云账号登录容器服务管理控制台。
2. 在控制台左侧导航栏单击**授权管理**。
3. 在**子账号**页签下单击目标RAM用户右侧的**管理权限**。
4. 在**集群RBAC配置**页面中单击图标，选择要授予的集群和命名空间，设置**访问权限**为**自定义**，在文本框中选择asm-istio-admin，然后单击**下一步**。

页面提示**授权成功**。

5. 验证RAM用户是否拥有访问Istio资源的读写权限。
   i. 执行以下命令，查看虚拟服务。

   ```
   kubectl get VirtualService
   ```

   预期输出：

   ```
   NAME            CREATED AT
   reviews-route   2021-11-15T07:09:10Z
   ```

   ii. 执行以下命令，编辑虚拟服务。

   ```
   kubectl edit VirtualService reviews-route
   ```

   预期输出：

   ```
   virtualservice.networking.istio.io/reviews-route edited
   ```

步骤五：使用集群KubeAPI创建和查看Istio资源

本文以Helm Chart方式创建和查看Istio资源为例。

> **说明** 在开启数据面集群KubeAPI访问Istio资源功能后，数据平面集群需要等待1~2分钟，然后才可以使用该功能。

1. 下载并解压Istio-bookinfo至本地。

   Istio-bookinfo文件包含Istio资源和Bookinfo应用的YAML文件。

2. 进入到Istio-bookinfo文件下，执行以下命令，创建Istio资源并安装Bookinfo应用。

   ```
   helm install -f values.yaml istio-bookinfo ./
   ```

3. 验证Istio-bookinfo是否安装成功。

   i. 在ASM控制台查看Istio资源。

      a.

      b.

      c.

      d. 在网格管理页面选择**流量管理 > 网关规则**。

         在**网关规则**页面可以看到bookinfo-gateway网关，说明创建Istio资源成功。

   ii. 在ACK控制台查看Bookinfo应用。

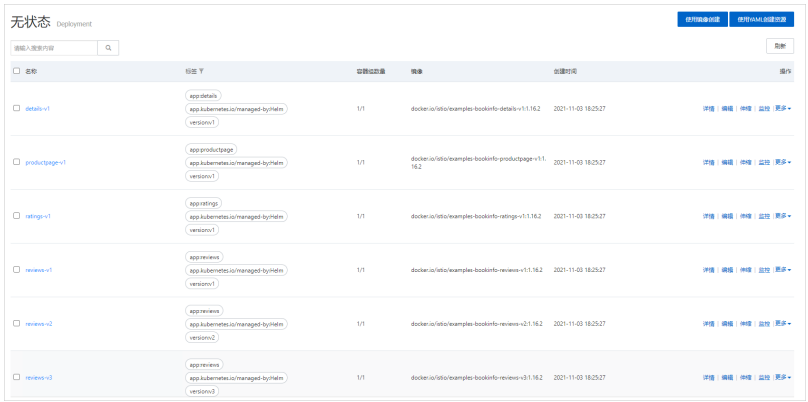
      a. 登录容器服务管理控制台。

      b. 在控制台左侧导航栏中，单击**集群**。

      c. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

      d. 在集群管理页面可以选择**工作负载 > 无状态**。

         在**无状态**页面可以看到reviews、details等应用，说明安装Bookinfo应用成功。

      根据以上结果，说明Istio-bookinfo安装成功，同时也说明使用数据面集群KubeAPI创建Istio资源成功。

4. 执行以下命令，使用数据面集群KubeAPI查看bookinfo-gateway网关。

```
kubectl get Gateway bookinfo-gateway -o yaml
```

返回bookinfo-gateway网关的YAML文件内容，说明查看bookinfo-gateway网关成功。

1.2.3. 使用集群Kubernetes API访问Istio资源（新版本）

ASM支持通过数据面集群的Kubernetes API（KubeAPI）对Istio资源进行增删改查操作，支持使用Helm管理应用。本文介绍如何使用数据面集群KubeAPI访问Istio资源功能。

前提条件

- 已创建ASM企业版或旗舰版实例，且ASM实例需要大于1.12.4.50版本。具体操作，请参见创建ASM实例。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。

背景信息

Kubernetes API是通过HTTP提供的基于资源的编程接口，支持通过标准HTTP谓词（POST、PUT、PATCH、DELETE、GET）检索、创建、更新和删除集群的主资源，例如Deployment、Service等。更多信息，请参见Kubernetes API。

注意事项

- 强烈建议在单集群模式下使用集群KubeAPI访问Istio资源功能。如果ASM的数据平面有多个集群，则任意一个数据平面集群都可以对ASM上的Istio资源进行增删改查操作。
- 开启数据面集群KubeAPI访问Istio资源功能后，数据面集群将无法删除istio-system命名空间。如果要删除，您需要先从ASM实例中移出数据面集群。
- 开启数据面集群KubeAPI访问Istio资源功能后，数据平面需要等待1~2分钟的时间进行准备。
- 删除数据平面的某一命名空间，不会删除ASM控制平面的对应命名空间，以及该命名空间下的Istio资源。
- 如果ASM控制平面有某一命名空间，但是数据平面没有此命名空间，您需要先在数据平面创建出此命名空间，然后才能在这个命名空间下对Istio资源进行增删改查操作。否则会提示以下错误信息：

  ```
  Error from server (NotFound): error when creating "xx.yaml": namespaces "daily-01" not found
  ```

- 如果在数据平面创建的Istio资源对应的命名空间在ASM控制平面不存在，则会在控制平面自动创建该命名空间。

启用集群KubeAPI访问Istio资源功能

创建ASM实例时启用该功能

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面单击**创建新网格**。
4. 在**创建服务网格**页面选中**启用数据面集群KubeApi访问Istio资源**，其他参数配置，请参见创建ASM实例，选中**服务协议**，单击**创建服务网格**。

为已有ASM实例时启用该功能

1. 登录ASM控制台。

2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在**基本信息**页面单击**启用数据面KubeAPI访问**右侧的**启用**。
5. 在**确认启用数据面集群KubeAPI访问**对话框单击**确认**。

场景示例一：使用kubectl管理Istio资源

启用集群KubeAPI访问Istio资源功能，使用kubectl连接集群后，您可以使用集群的kubeconfig创建、查询、修改和删除Istio资源，本文以虚拟夫服务为例。

- 使用以下命令，可以创建虚拟服务。

```
kubectl apply -f <虚拟服务YAML文件名称>
```

- 使用以下命令，可以查询虚拟服务。

```
kubectl get Virtualservice
```

- 使用以下命令，可以修改虚拟服务。

```
kubectl edit Virtualservice <虚拟服务的名称>
```

- 使用以下命令，可以删除虚拟服务。

```
kubectl delete Virtualservice <虚拟服务的名称>
```

场景示例二：使用Helm安装软件

启用集群KubeAPI访问Istio资源功能后，Helm可以使用集群的kubeconfig同时安装软件至集群，安装Istio资源至ASM。

1. 通过kubectl工具连接集群。
2. 在本地安装Helm。具体操作，请参见Helm。

   > **说明** 使用kubectl连接集群后，Helm客户端会自动使用kubeconfig连接集群。

3. 下载Helm Chart示例istio-bookinfo至本地，然后对文件进行解压。
4. 进入到istio-bookinfo目录下，执行以下命令，安装istio-bookinfo。

   ```
   helm install -f values.yaml istio-bookinfo ./
   ```

   预期输出：

   ```
   NAME:istio-bookinfo
   LAST DEPLOYED:THU May 26 16:44:19 2022
   NAMESPACE:default
   STATUE:deployed
   REVISION:1
   TEST SUITE:None
   ```

5. 验证使用Helm安装软件是否成功。

i. 在ACK控制台查看Bookinfo应用。

a. 登录容器服务管理控制台。

b. 在控制台左侧导航栏中，单击**集群**。

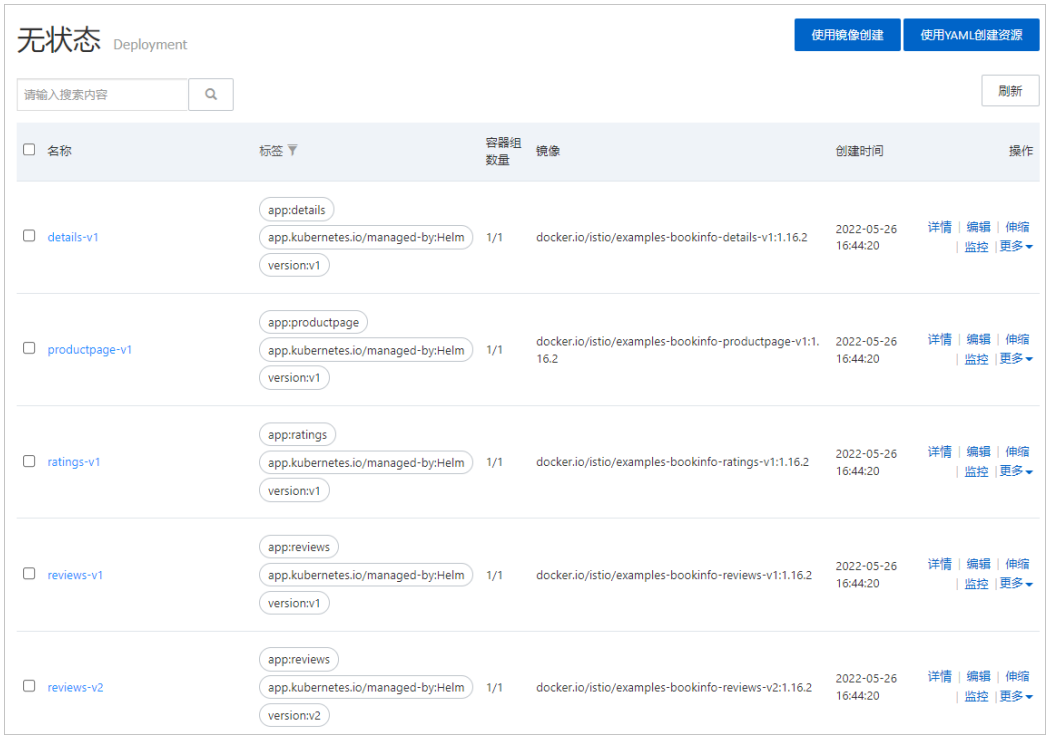
c. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

d. 在集群管理页面左侧导航栏选择**工作负载 > 无状态**。

e. 在**无状态**页面设置**命名空间**为default。

可以看到使用Helm安装的details、prosuctpage、ratings等应用。

> **说明** 在集群管理页面左侧导航栏选择**应用 > Helm**，在Helm页面可以查看Helm安装包。



ii. 在ASM控制台查看虚拟服务和网关规则。

a. 登录ASM控制台。

b. 在左侧导航栏，选择**服务网格 > 网格管理**。

c. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

d. 在网格详情页面左侧导航栏选择**流量管理 > 虚拟服务**。

在**虚拟服务**可以看到使用Helm安装的名为bookinfo的虚拟服务。

e. 在网格详情页面左侧导航栏选择**流量管理 > 网关规则**。

在**网关规则**可以看到使用Helm安装的名为bookinfo-gateway的网关规则。

相关操作：

- 执行以下命令，可以查看Helm安装列表。

```
helm list
```

- 执行以下命令，可以更新Helm Chart。

```
helm upgrade -f values.yaml istio-bookinfo ./
```

1.3. 回滚Istio资源的历史版本

当您更新Istio资源的 `spec` 字段中的内容时，ASM会记录更新Istio资源的历史版本，最多记录最近更新的5个版本。本文以虚拟服务为例，介绍如何回滚Istio资源的历史版本。

前提条件

- 已创建ASM实例，且ASM实例的Istio为v1.9.7.92-g1d820703-aliyun及以上版本。具体操作，请参见创建ASM实例。
- 已创建虚拟服务。具体操作，请参见管理虚拟服务。

背景信息

Istio资源是指ASM控制台流量管理下的虚拟服务、目标规则、网关规则、服务条目、Envoy过滤器、工作负载组、工作负载条目和Sidecar资源，以及零信任安全下的请求身份认证、对等身份认证及授权策略。

步骤一：启用Istio资源历史版本功能

您可以通过以下两种方式来启用Istio资源历史版本功能：

- 如果您没有创建ASM实例，您可以在创建ASM实例时选中**启用Istio资源历史版本**来启用Istio资源历史版本功能。
- 如果您已创建ASM实例，您可以在ASM实例的网格信息页面启用Istio资源历史版本功能。本文以已创建ASM实例场景为例。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**，然后在右侧页面单击**功能设置**。
5. 在**功能设置更新**面板选中**启用Istio资源历史版本**，然后单击**确定**。

步骤二：生成虚拟服务的历史版本

> **注意** 只有更新Istio资源的 `spec` 字段中的内容时，ASM才会记录形成历史版本。如果您更新的是Istio资源其他字段，ASM不会记录形成历史版本。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**流量管理 > 虚拟服务**。
5. 在**虚拟服务**页面单击目标虚拟服务**操作**列下的**YAML**。
6. 在**编辑**面板修改 `spec` 字段下的内容，例如 `spec` 字段下的 `number` 端口由9080修改为9081，然后单击**确定**。

步骤三：回滚虚拟服务的历史版本

本文以回滚到目标虚拟服务的v2版本为例。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**流量管理 > 虚拟服务**。
5. 在**虚拟服务**页面单击目标虚拟服务右侧**操作**列下的**版本管理**。
6. 在**版本管理**面板单击v2版本**操作**列下的**查看**，然后单击**回滚**。
   在**虚拟服务**页面单击目标虚拟服务**操作**列下的**YAML**，在**编辑**面板可以看到目标虚拟服务的YAML内容回滚到v2版本。

FAQ

为什么虚拟服务页面找不到版本管理？

回滚Istio资源的历史版本前，请确保您的Istio版本不能低于v1.9.7.92-g1d820703-aliyun，并且您需要启用Istio资源历史版本功能。

是否只能通过ASM控制台更新Istio资源，ASM才会记录该资源的历史版本？

Istio资源历史版本功能不受操作方式的影响，只要您启用该功能，ASM就会为您记录Istio资源的历史版本。

Istio资源历史版本管理是否有什么限制？

ASM最多为您记录Istio资源最近被更新的5个历史版本。当Istio资源修改超过5次，将清除更新时间最早的历史版本。

ASM记录的Istio资源历史版本与实际更新的YAML内容不完全相同？

ASM记录的Istio资源历史版本会自动省略YAML中冗余的默认值，不会影响该版本的实际使用效果。例如网关规则资源 `spec` 中的 `servers.tls` 字段默认为 `PASSTHROUGH` 。如果您再将此字段设定为PASSTHROUGH，则该设定是冗余的，因此Istio资源历史版本管理功能不会为您记录此字段的设定。

1.4. 启用控制平面日志采集和日志告警

ASM支持采集控制平面日志和日志告警，例如采集ASM控制平面向数据平面Sidecar推送配置的相关日志。本文介绍日志告警处理建议，以及如何启用控制平面日志采集和日志告警。

启用控制平面日志采集

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**。
5. 在**基本信息**页面单击**控制面日志采集**右侧的**开启**。
6. 在**启用控制面日志**对话框选择新建Project或使用已有Project，然后单击**确认**。

   如果您选择的是新建Project，您可以使用默认Project名称或者自定义Project名称。

   在**网格信息**页面单击**控制面日志采集**右侧的**查看日志**，然后您可以在Project页面查看详细的控制平面日志。

启用控制平面日志告警

当前只支持数据平面同步失败告警。当控制平面发往数据平面的xDS请求被数据平面拒绝时，数据平面同步失败告警将被触发。此时您的数据平面Sidecar或Ingressgateway将无法得到最新的配置信息，将存在以下两种情况：

> **注意** 启用控制平面日志告警之前必须先启用控制平面日志采集，否则将无法使用该功能。

- 如果数据平面Sidecar在此之前收到过成功的配置推送，则该Sidecar将保持最后一次收到的成功推送的配置。
- 如果数据平面Sidecar在此之前尚未收到过成功的配置推送，则该Sidecar将没有任何配置信息，这意味着该节点可能没有任何监听，也无法处理任何请求和路由规则。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**。

5. 在**基本信息**页面单击**控制面日志采集**右侧的**告警设置**。
6. 在**控制面日志告警设置**对话框选择行动策略，然后单击**开启告警**。

   行动策略定义了告警触发时的行为，您可以在SLS Project内创建和编辑行动策略。具体操作，请参见创建行动策略。
7. 在**重要提示**对话框单击**确定**。

日志告警处理建议

以下列出了常见的数据面同步失败错误信息和处理建议。如果您没有在下方表格找到对应的错误信息，建议您提交工单。

错误信息	处理建议
Internal:Error adding/updating listener(s) 0.0.0.0_443: Failed to load certificate chain from <inline>, only P-256 ECDSA certificates are supported	该告警信息表示数据面集群不支持您为数据面配置的证书，当前仅支持P-256 ECDSA证书。您需要重新配置证书，具体操作，请参见通过服务网关启用HTTPS安全服务。
Internal:Error adding/updating listener(s) 0.0.0.0_443: Invalid path: ****	该告警信息表示您为数据面配置的证书路径有误或证书不存在，您需要检查证书挂载路径是否与Gateway中配置的路径相符。具体操作，请参见通过服务网关启用HTTPS安全服务。
Internal:Error adding/updating listener(s) 0.0.0.0_xx: duplicate listener 0.0.0.0_xx found	该告警信息表示您为网关配置的监听端口重复，请检查您的Gateway，删除重复的端口。
Internal:Error adding/updating listener(s) 192.168.33.189_15021: Didn't find a registered implementation for name: '***'	该告警信息表示在Sidecar和Ingressgateway中无法找到您通过EnvoyFilter针对15021这个Listener patch的配置中引用的***，您需要删除该引用。
Internal:Error adding/updating listener(s) 0.0.0.0_80: V2 (and AUTO) xDS transport protocol versions are deprecated in grpc_service ***	该告警信息表示即将弃用您数据面的XDS V2协议，这通常是因为您的数据面Sidecar的版本与控制平面不符所致。升级数据平面的Sidecar可以解决该问题，您需要删除Pod，该Pod自动重新创建后会自动注入最新版本的Sidecar。

1.5. 启用Multi-Buffer实现TLS加速

ASM商业版（专业版）结合Intel的Multi-Buffer加解密技术，可以加速Envoy中TLS的处理过程。本文介绍如何启用Multi-Buffer实现TLS加速。

前提条件

- 已创建ASM商业版（专业版）实例，且实例为1.10及以上版本。具体操作，请参见创建ASM实例。
- 已创建ACK，且集群节点的实例规格族需要支持Multi-Buffer CPU机型Intel Ice Lake。具体操作，请参见创建Kubernetes托管版集群。

  以下实例规格族支持Multi-Buffer CPU机型Intel Ice Lake：

  > **说明** 关于实例规格的详细介绍，请参见实例规格族。

规格族系列	实例规格族
g7系列	存储增强通用型实例规格族g7se
	通用型实例规格族g7
	安全增强通用型实例规格族g7t
c7系列	计算型实例规格族c7
	RDMA增强型实例规格族c7re
	存储增强计算型实例规格族c7se
	安全增强计算型实例规格族c7t
r7系列	内存型实例规格族r7p
	存储增强内存型实例规格族r7se
	内存型实例规格族r7
	安全增强内存型实例规格族r7t
其他	内存增强型实例规格族re7p
	GPU虚拟化型实例规格族vgn7i-vws
	GPU计算型实例规格族gn7i
	GPU计算型弹性裸金属服务器实例规格族ebmgn7i
	计算型超级计算集群实例规格族sccc7
	通用型超级计算集群实例规格族sccg7

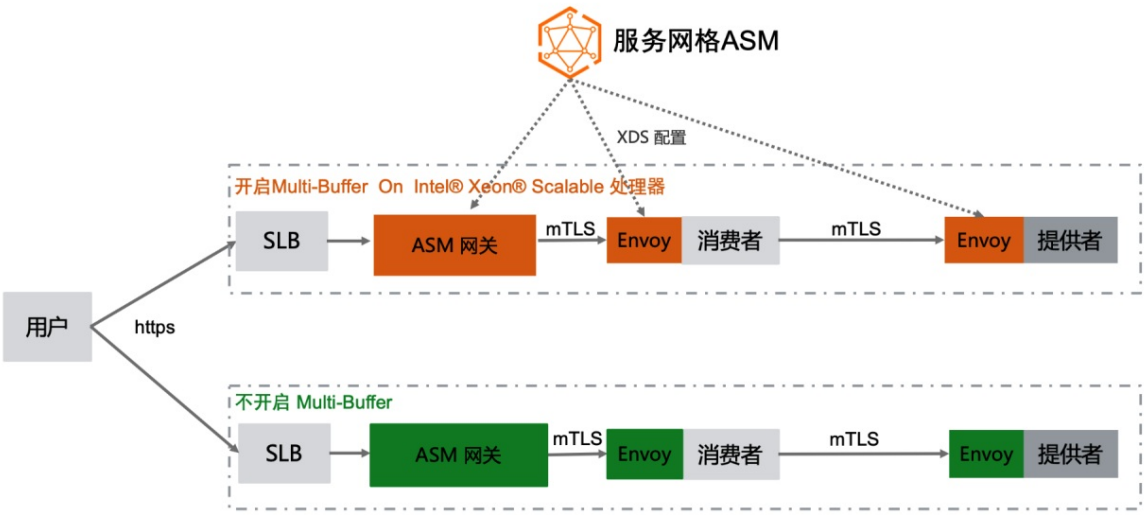
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。

背景信息

随着网络安全技术的发展，TLS已经成为网络通信的基石。一个TLS会话的处理过程总体上可分为握手阶段和数据传输阶段。握手阶段最重要的任务是使用非对称加密技术协商出一个会话密钥，然后在数据传输阶段，使用该会话密钥对数据执行对称加密操作，再进行数据传输。

在微服务场景下，Envoy无论是作为Ingress Gateway还是作为微服务的代理，都需要处理大量的TLS请求，尤其在握手阶段执行非对称加解密的操作时，需要消耗大量的CPU资源，在大规模微服务场景下这可能会成为一个瓶颈。ASM结合Intel的Multi-Buffer加解密技术，可以加速Envoy中TLS的处理过程。

Multi-Buffer加解密技术使用Intel CPU AVX-512指令同时处理多个独立的缓冲区，即可以在一个执行周期内同时执行多个加解密的操作，成倍的提升加解密的执行效率。Multi-Buffer技术不需要额外的硬件，只需要CPU包含特定的指令集。目前阿里云在Ice Lake处理器中已经包含了最新的AVX-512指令集。



操作步骤

您可以通过以下两种方式来启用Multi-Buffer功能：

- 如果您没有创建ASM实例，您可以在创建ASM实例时选中**启用基于MultiBuffer的TLS加解密性能优化**。具体操作，请参见创建ASM实例。
- 如果您已创建ASM实例，您可以在ASM实例的**网格信息**页面启用基于MultiBuffer的TLS加解密性能优化功能。本文以已创建ASM实例场景为例。

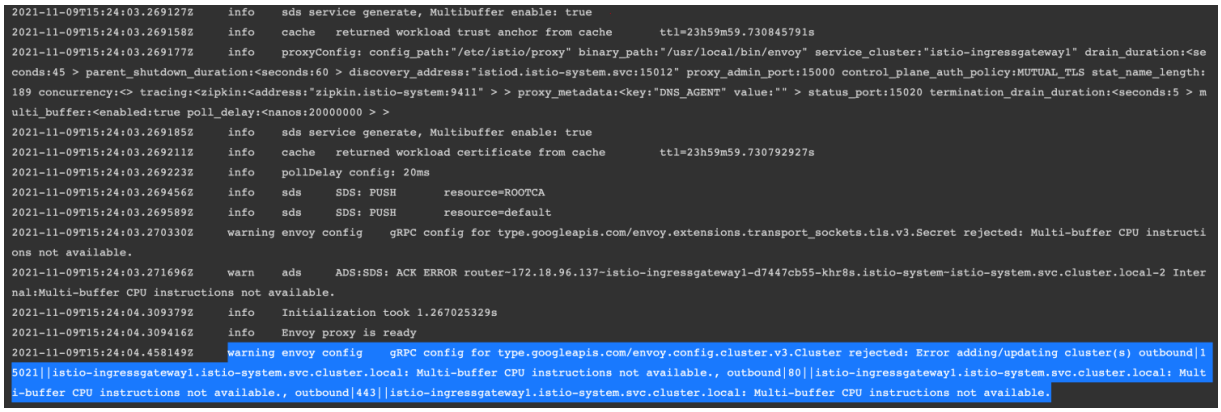
1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**，然后在右侧页面单击**功能设置**。
5. 在**功能设置更新**面板选中**启用基于MultiBuffer的TLS加解密性能优化**，然后单击**确定**。

   如果您使用通用型实例规格族g7作为Kubernertes节点，启用Multi-Buffer功能后，每秒查询率（QPS）将提升75%的性能。如果您使用的是弹性裸金属节点，提升的性能将更高。

FAQ

如果在控制面启用了MultiBuffer功能，但数据面Kubernetes集群下的节点不是Intel Ice Lake的机型会怎么样？

Envoy会输出告警日志，且MultiBuffer功能将不会生效。



ASM Pro 1.10及以上版本提供了开启TLS加速时的自适应判断能力，若业务或者网关Pod被调度到的Node节点为非Intel Ice Lake机型，则不会下发对应的加速配置，TLS加速不会生效。

如果Kubernetes集群没有支持Multi-Buffer功能类型的节点，那该集群如何才能使用MultiBuffer功能？

1. 在该Kubernetes集群添加新的节点，且节点的实例规格需要支持Multi-Buffer CPU机型Intel Ice Lake。具体操作，请参见添加已有节点。
2. 在新添加的节点上设置 `multibuffer-support：true` 标签。具体操作，请参见管理节点标签。
3. 在ASM网关的YAML配置中添加以下内容。具体操作，请参见修改入口网关服务。

   通过增加节点亲和性，使Gateway实例调度到新添加的支持Multi-Buffer功能的节点上。

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: multibuffer-support
                operator: In
                values:
                  - true
```

4. 在ASM商业版（专业版）启用MultiBuffer功能。具体操作，见上文。

   启用MultiBuffer功能后，该集群新添加的节点即可使用MultiBuffer功能，加速TLS处理过程。

1.6. 使用ASM网格诊断

服务网格ASM支持对实例进行网格诊断，诊断项包括数据平面版本检查、服务端口检查、服务关联检查、App及Version标签检查、目标地址检查和虚拟服务冲突检查。本文介绍如何使用ASM进行网格诊断。

操作步骤

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 网格诊断**。

5. 在**网格诊断**页面单击**运行**。

> **说明** 如果已诊断该网格实例，**网格诊断**页面将显示上一次的诊断结果。您可以单击**运行**，对该网格实例进行再次诊断。

在**网格诊断**页面下方显示诊断项结果：

- 诊断项**结果**列显示✅，表示该诊断项正常。
- 诊断项**结果**列显示❌，表示该诊断项异常并显示异常原因。鼠标移至**详情**，您可以查看该诊断项异常的详细信息。

1.7. 安装和使用诊断工具asmctl

ASM提供了诊断工具asmctl，用于检测ASM存在的配置问题。本文介绍如何安装诊断工具asmctl，以及使用诊断工具asmctl连接集群和ASM。

兼容性说明

ASM不保证兼容社区Istio社区诊断工具Istioctl，但是ASM提供诊断工具asmctl，该工具当前支持Istioctl部分命令。

诊断工具asmctl支持在以下版本的ACK集群和ASM中使用：

- v1.8.6.49-gda24841c-aliyun及以上版本的ASM标准版。
- v1.20.4-aluyun-1及以上版本的ACK Pro托管版集群。

目前，不保证在v1.8.6.49-gda24841c-aliyun以下ASM标准版中使用asmctl时，所有命令的可用性。

安装asmctl

1. 下载asmctl二进制可执行文件。不同操作系统文件不同，请根据实际情况选择。
   - Mac版：asmctl
   - Linux版：asmctl
2. 设置asmctl运行环境。
   i. 执行以下命令，创建*.istioctl/bin*目录。

```
# 可以将asmctl与istioctl放在同一目录下，方便管理。
# 若已按istio官网方式安装istioctl，则此目录无需创建。
mkdir $HOME/.istioctl/bin
```

   ii. 执行以下命令，将asmctl二进制可执行文件移动到*.istioctl/bin*目录下。

```
# 运行时注意将$PATH_TO_ASMCTL替换为asmctl所在路径。
mv $PATH_TO_ASMCTL/asmctl $HOME/.istioctl/bin/asmctl
```

   iii. 执行以下命令，将asmctl所在路径添加到PATH。

```
# 将asmctl所在路径添加到PATH。

export PATH=$PATH:$HOME/.istioctl/bin
```

iv. 执行以下命令，为asmctl增加可执行权限。

```
# 为asmctl增加可执行权限。
chmod +x ".istioctl/bin/asmctl"
```

使用诊断工具asmctl

您需要配置集群连接凭证和服务网格连接凭证，您才可以使用诊断工具asmctl检测服务网格和集群。集群和服务网格连接凭证默认放在*$HOME/.kube/config*和*$HOME/.kube/asmconfig*，您也可以通过 `--kubeconfig` 或 `-c` 自定义集群凭证位置，通过 `--asmconfig` 或 `-m` 自定义服务网格凭证位置。关于asmctl常用命令，请参见诊断工具asmctl常用命令。

使用诊断工具asmctl连接集群

本文以公网访问凭证连接ACK集群为例。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群信息页面单击**连接信息**页签，单击**公网访问**下右上角的**复制**。
5. 在*$HOME/.kube*目录下创建*config*文件，将复制的公网访问凭证保存到*config*文件中。
6. 执行以下命令，验证连接集群是否成功。

   ```
   kubectl get ns
   ```

   如果返回命名空间信息，则表示asmctl连接集群成功。

使用诊断工具asmctl连接ASM

本文以公网访问凭证连接ASM为例。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格信息页面单击右上角的**连接配置**。
5. 在**连接配置**面板**公网访问**下单击**复制**，然后单击**确定**。
6. 在*$HOME/.kube*目录下创建*asmconfig*文件，将复制的公网访问凭证保存到*asmconfig*文件中。
7. 执行以下命令，验证连接ASM是否成功。

   ```
   kubectl get ns --kubeconfig $HOME/.kube/asmconfig
   ```

   如果返回命名空间信息，则表示asmctl连接ASM成功。

1.8. 诊断工具asmctl常用命令

本文介绍诊断工具asmctl常用命令。

兼容性说明

ASM不保证兼容社区Istio社区诊断工具Istioctl，但是ASM提供诊断工具asmctl，该工具当前支持Istioctl部分命令。

诊断工具asmctl支持在以下版本的ACK集群和ASM中使用：

- v1.8.6.49-gda24841c-aliyun及以上版本的ASM标准版。
- v1.20.4-aluyun-1及以上版本的ACK Pro托管版集群。

目前，不保证在v1.8.6.49-gda24841c-aliyun以下ASM标准版中使用asmctl时，所有命令的可用性。

使用限制

asmctl支持的命令与社区提供的Istioctl工具的1.9版本相兼容。由于服务网格ASM的托管特性，asmctl当前仅支持部分Istioctl命令，当前不支持的命令及其原因如下：

命令	当前不支持原因
dashboard系列命令中，除dashboard envoy以外的命令	服务网格控制面由ASM托管，不支持通过命令行工具对控制面核心组件及服务网格自建可观测性。
create-remote-secret	服务网格控制面由ASM托管，不支持自行为控制面创建Secret访问远程Kubernetes集群。
istiod	服务网格控制面由ASM托管，不支持通过命令行工具管理控制面核心组件。
metrics	服务网格控制面由ASM托管，不支持通过命令行工具对控制面核心组件及服务网格自建可观测性。
precheck	服务网格控制面由ASM托管，asmctl无需提供Istio兼容性检查相关功能。
proxy-status	服务网格控制面由ASM托管，当前不支持通过命令行工具自行查询控制面状态，您可以在服务网格控制台**概览**页面查看网格状态。
uninstall、install	服务网格控制面由ASM托管，不支持自行安装、卸载控制面。
version	当前不支持查看版本信息，可以通过服务网格控制台查看网格基础信息。
wait	服务网格控制面由ASM托管，当前开发进度下不支持通过命令行工具自行查询控制面状态。
manifest、operator、profile、upgrade、verify-install	服务网格控制面由ASM托管，asmctl不提供通过命令在用户侧集群中再次安装istio的能力。
kube-inject、kube-uninject、workload、add-to-mesh系列命令、remove-from-mesh命令	asmctl还处于开发早期阶段，目前暂时不开放会对集群进行实质性更改的命令。

asmctl命令概览

命令	描述	链接
asmctl analyze	分析集群的控制面配置并输出分析结果。	asmctl analyze
asmctl bug-report	选择性地收集ASM实例与用户侧集群中的信息与日志，并压缩到一个压缩包，帮助您对常见问题进行诊断。	asmctl bug-report
asmctl dashboard	访问与社区Istio兼容的Web UI。	asmctl dashboard

命令	描述	链接
asmctl dashboard envoy	为指定容器组的Sidecar打开Envoy管理面板。	asmctl dashboard envoy
asmctl experimental	这些命令还在开发中，属于实验性命令。	asmctl experimental
asmctl experimental authz	提供与服务网格授权策略有关的功能。	asmctl experimental authz
asmctl experimental authz check	检查指定容器组的Sidecar配置，并输出应用在该容器组Sidecar上的所有授权策略。	asmctl experimental authz check
asmctl experimental config	提供与服务网格默认设置有关的功能。	asmctl experimental config
asmctl experimental config list	列出服务网格中可配置的默认设置信息。	asmctl experimental config list
asmctl experimental describe	输出有关指定Kubernetes资源以及与其相关的服务网格配置的描述性信息。	asmctl experimental describe
asmctl experimental describe pod	根据指定的容器组，分析与它相关的Kubernetes服务、目标规则和虚拟服务等，并最终输出与该容器组相关的描述性信息。	asmctl experimental describe pod
asmctl experimental describe service	根据指定的Kubernetes服务，结合分析与之相关的容器组、目标规则和虚拟服务等，并最终输出与该Kubernetes服务相关的描述性信息。	asmctl experimental describe service
asmctl experimental injector	负责与Sidecar注入情况以及Sidecar注入器信息有关的功能。	asmctl experimental injector
asmctl experimental injector list	列出每个命名空间中容器组Sidecar的注入情况，并输出ASM使用的Sidecar注入器的概要信息。	asmctl experimental injector list
asmctl proxy-config	负责获取容器组中Sidecar的配置信息。	asmctl proxy-config
asmctl proxy-config bootstrap	负责获取指定容器组中Sidecar Envoy的bootstrap配置信息。	asmctl proxy-config bootstrap
asmctl proxy-config cluster	负责获取指定容器组中Sidecar Envoy的cluster配置信息。	asmctl proxy-config cluster
asmctl proxy-config endpoint	负责获取指定容器组中Sidecar Envoy的Endpoint配置信息。	asmctl proxy-config endpoint

命令	描述	链接
asmctl proxy-config listener	负责获取指定容器组中Sidecar Envoy的Listener配置信息。	asmctl proxy-config listener
asmctl proxy-config log	负责获取Sidecar Envoy的日志等级信息，并且支持选择性地更新日志。	asmctl proxy-config log
asmctl proxy-config route	负责获取指定容器组中Sidecar Envoy的Route配置信息。	asmctl proxy-config route
asmctl proxy-config secret	负责获取指定容器组中Sidecar Envoy的Secret配置信息。	asmctl proxy-config secret
asmctl validate	负责验证服务网格规则与策略文件的正确性。	asmctl validate

asmctl analyze

analyze命令分析集群的控制面配置并输出分析结果。

```
asmctl analyze <file>... [flags]
```

选项	选项缩写	描述
--all-namespaces	-A	分析所有的命名空间。
--asmconfig <string>	-m	设置ASM实例的kubeconfig文件路径，默认为*$HOME/.kube/asmconfig*。
--color	无	是否以带颜色的格式输出分析结果，默认为 `true` ，可设定为 `false` 。
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--failure-threshold <Level>	无	设定分析时在何种等级下会返回非0的错误码。可用的值包括： `Info` 、 `Warning` 、 `Error` ，默认为 `Error` 。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--list-analyzers	-L	列出analyze命令当前执行的所有分析手段。
--meshConfigFile <string>	无	在分析时使用给定的网格配置文件来覆盖当前的网格配置，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。

选项	选项缩写	描述
--output <string>	-o	设定输出的格式，可用的值包括：`log`、`json`、`yaml`，默认为空。
--output-threshold <Level>	无	设定分析时在何种等级下会输出消息。可用的值包括：`Info`、`Warning`、`Error`，默认为`Info`。
--recursive	-R	在指定了分析目录时，递归地处理目录中的所有文件，而不是仅分析目录中的第一层文件。
--suppress <stringArray>	-S	设定停止报告某个特定资源的某个消息码。该项以`<code>=<resource>`的形式提供，例如`--suppress "IST0102=DestinationRule primary-dr.default`。该项可以重复提供，且可以在传入的参数中使用通配符“*”来支持部分匹配，例如`suppress "IST0102=DestinationRule *.default`。该项默认为`[]`。
--timeout <duration>	无	设置命令返回超时错误的时间，执行分析超过超时时间后会返回timeout错误而不再返回分析结果，默认为`30s`。
--use-kube	-k	是否基于当前的ASM实例与用户侧集群进行分析，如果只是想要分析文件本身，则需将此项设定为`use-kube=false`。
--verbose	-v	设定输出详细的分析过程。

以下为asmctl analyze命令示例：

```
# 分析当前的用户侧集群和ASM实例的控制面配置信息。
asmctl analyze
# 分析当前的用户侧集群和ASM实例，模拟分析在应用a.yaml、b.yaml与my-app-config目录中的配置文件后产生的效果。
asmctl analyze a.yaml b.yaml my-app-config/
# 分析当前的用户侧集群和ASM实例，模拟分析在应用了a.yaml、b.yaml与my-app-config目录中的配置文件后产生的效果，同时指定了需要分析的用户侧集群与和ASM实例的kubeconfig文件。
asmctl analyze a.yaml b.yaml my-app-config/ -c ~/.kube/ackconfig1 -m ~/.kube/asmconfig1
# 分析当前的用户侧集群和ASM实例，模拟分析在应用了my-app-config目录中的配置文件后产生的效果，其中my-app-config目录中的所有配置文件都会被递归地分析。
asmctl analyze --recursive my-istio-config/
# 仅分析当前a.yaml、b.yaml与my-app-config目录中的YAML文件应用后的效果，而不考虑当前使用的用户侧集群和ASM实例的任何资源配置
asmctl analyze --use-kube=false a.yaml b.yaml my-app-config/
# 分析当前的用户侧集群和ASM实例，但对于testing命名空间下的mypod这一pod、忽略针对其产生的"PodMissingProxy"的分析结果输出。。
asmctl analyze -S "IST0103=Pod mypod.testing"
# 分析当前的用户侧集群和ASM实例，但对于testing命名空间下的所有Pod、忽略针对其产生的"PodMissingProxy"的分析结果输出。
# 同时对于default命名空间下的foobar这一deployment资源、忽略针对其产生的"MissplacedAnnotation"的分析结果输出。
asmctl analyze -S "IST0103=Pod *.testing" -S "IST0107=Deployment foobar.default"
# 列出当前asmctl analyze指令执行的所有分析手段。
asmctl analyze -L
```

asmctl bug-report

`bug-report` 命令选择性地收集ASM实例与用户侧集群中的信息与日志，并压缩到一个压缩包，从而可以对常见问题进行诊断。收集的信息包括：

- Sidecar的配置信息与状态信息。
- Sidecar产生的日志信息。
- 用户侧集群资源信息。
- 调用 `analyze` 命令得出的分析结果。

```
asmctl bug-report [flags]
```

选项	选项缩写	描述
--asmconfig <string>	-m	ASM实例的kubeconfig文件路径，默认为 `$HOME/.kube/asmconfig` 。
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--dir <string>	无	为bug-report命令中产生的临时输出文件设定一个目录，默认为空。
--dry-run	无	控制是否实际进行日志的收集和存储，设定 `dry-run` 选项，表示不进行实际的日志收集和存储。

选项	选项缩写	描述
--duration <duration>	无	从当前时间计算，往前收集多长一段时间的日志信息。默认为无穷大（`0s`），如果该项被设置，则`start-time`选项不能被设置。
--end-time <string>	无	要包含在命令输出中的日志的时间跨度的结束时间。默认为现在时间。
--exclude <stringSlice>	无	匹配应该从所有Sidecar日志中排除的容器组的Sidecar日志。该项在`include`选项之后生效，更多信息，请参见下文的自定义过滤bug-report命令收集的Sidecar日志。默认排除命名空间kube-system、kube-public、kube-node-lease、local-path-storage中的容器组。
--filename <string>	-f	指定一个包含bug-report配置信息的YAML文件。该文件中指定的选项内容将覆盖在命令行选项中指定的内容，默认为空。
--full-secrets	无	控制是否收集Secret信息，设定`full-secrets`选项，则收集Secret信息到命令输出中。
--ignore-errs <stringSlice>	无	一系列以逗号分隔的glob匹配模式，这些模式用来匹配忽略的日志错误（error）信息字符串。在计算日志重要性时，匹配该项给出模式的错误将忽略不计。
--include <stringSlice>	无	匹配应该在命令输出中包含的容器组的Sidecar日志，更多信息，请参见下文的自定义过滤bug-report命令收集的Sidecar日志。默认为空。
--istio-namespace <string>	-i	控制面组件安装的命名空间，默认为`istio-system`。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--start-time <string>	无	包含在命令输出中的日志的时间跨度的开始时间。默认为空，即从最早的日志开始收集。
--timeout <duration>	无	能够容忍获取日志所花的最长时间。当收集日志时间超时时，仅目前已经收集的日志会保存至命令输出，默认为`30m0s`。

自定义过滤bug-report命令收集的Sidecar日志

Sidecar的日志收集可以通过以下的选项使用方式进行过滤：

```
--include|--exclude
ns1,ns2.../dep1,dep2.../pod1,pod2.../cntr1,cntr.../lbl1=val1,lbl2=val2.../ann1=val1,ann2=val2
```

在 `include/exclude` 选项后给出的字符串即为日志的过滤条件，其中 `ns` 指命名空间Namespace，`dep` 指部署Deployment，`pod` 指容器组Pod，`cntr` 指容器Container，`lbl` 指标签Label，`ann` 指注解Annotation。

上述过滤条件表示必须匹配 `(ns1 OR ns2) AND (dep1 OR dep2) AND (cntr1 OR cntr2)……` ，当且仅当一个容器组的Sidecar匹配至少一个 `include` 选项给出的过滤条件，并且不匹配所有 `exclude` 选项中给出的过滤条件时，其日志会被包含在最终命令输出的压缩包中。

所有的过滤条件都是可选的、可被忽略，例如可以使用 `ns1//pod1` 过滤条件，仅仅针对命名空间Namespace与容器组Pod进行过滤。

除标签label及注解annotation之外的所有类型的过滤名称都支持使用glob匹配模式，例如可以使用 `n*//p*/l=v*` 的过滤条件，该条件能够匹配所有在以n为开头的命名空间中，含有键名称为l的标签、而该标签对应的值以v开头，且自身名称以p开头的容器组。

asmctl dashboard

访问与社区istio兼容的Web UI。目前asmctl仅对Envoy提供了 `dashboard` 命令。

```
asmctl dashboard [flags]
```

您也可以使用：

```
asmctl dash [flags]
```

```
asmctl d [flags]
```

选项	选项缩写	描述
--address <string>	无	Web UI的监听地址，只接受IP地址或localhost。如果该项为localhost，asmctl将会尝试绑定127.0.0.1（IPV4）或是::1（IPV6）两个地址，如果这两个地址都无法绑定则命令执行失败。默认为 `localhost` 。
--browser	无	设定是否打开浏览器。当 `browse` 选项被设定为 `false` ，asmctl将不会自动打开浏览器，默认为 `true` 。
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--port <int>	-p	Web UI监听的本地端口，默认为空。

asmctl dashboard envoy

为指定容器组的Sidecar打开Envoy管理面板。

```
asmctl dashboard envoy [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--address <string>	无	Web UI的监听地址，只接受IP地址或localhost。如果该项为localhost，asmctl将会尝试绑定127.0.0.1（IPV4）或是::1（IPV6）两个地址，如果这两个地址都无法绑定则命令执行失败。默认为 `localhost` 。
--browser	无	设定是否打开浏览器。当 `browse` 选项被设定为 `false` ，asmctl将不会自动打开浏览器，默认为 `true` 。
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--port <int>	-p	Web UI监听的本地端口（默认为 `0` ）。
--selector <string>	-l	容器组标签选择器（默认为空，使用标签选择器时不能提供容器组名称。）

以下为asmctl dashboard envoy命令示例：

```
# 通过容器组名称和命名空间名称指定容器组，并打开Sidecar的Envoy管理面板。
asmctl dashboard envoy productpage-123-456.default
# 通过部署名称及容器组名称指定容器组，并打开Sidecar的Envoy管理面板。
asmctl dashboard envoy deployment/productpage-v1
# 使用Dashboard指令的缩写形式。
asmctl dash envoy productpage-123-456.default
asmctl d envoy productpage-123-456.default
```

asmctl experimental

asmctl experimental表示这些命令还在开发中，属于实验性命令。asmctl当前兼容社区Istioctl 1.9版本，与当前ASM和社区Istio的兼容性情况保持一致。因此，asmctl也保留与Istioctl 1.9版本形式一致的实验性命令。

选项	选项缩写	描述
--asmconfig <string>	-m	设置ASM实例的kubeconfig文件路径，默认为 `$HOME/.kube/asmconfig` 。
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。

asmctl experimental authz

authz系列命令提供与服务网格授权策略有关的功能。

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。

asmctl experimental authz check

authz check命令检查指定容器组的Sidecar配置，并输出应用在该容器组Sidecar上的所有授权策略（AuthorizationPolicy）。该命令可用于方便地检查最终在多个授权策略合并后应用在某一具体Sidecar上的具体授权策略。

该命令同时支持使用-f选项来直接读取一个Sidecar配置的拷贝文件并输出该文件中配置规定的授权策略。

```
asmctl experimental authz check [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--file <string>	-f	指定被检查的Envoy配置转储文件，文件为JSON格式，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。

以下为asmctl experimental authz check命令示例：

```
# 检查容器组httpbin-88ddbcfdd-nt5jb上应用的授权策略。
asmctl x authz check httpbin-88ddbcfdd-nt5jb
# 检查在部署productpage-v1上应用的授权策略。
asmctl proxy-status deployment/productpage-v1
# 从Envoy配置转储文件httpbin_config_dump.json中检查应用到的授权策略。
asmctl x authz check -f httpbin_config_dump.json
```

asmctl experimental config

config系列命令提供与服务网格默认设置有关的功能。

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。

asmctl experimental config list

config list命令列出服务网格中可配置的默认设置信息。

```
asmctl experimental config list [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。

asmctl experimental describe

describe系列命令输出有关指定Kubernetes资源以及与其相关的服务网格配置的描述性信息。

```
asmctl experimental describe [command] [flags]
```

您也可以使用：

```
asmctl experimental des [command] [flags]
```

选项	选项缩写	描述
--asmconfig <string>	-m	ASM实例的kubeconfig文件路径，默认为 `$HOME/.kube/asmconfig` 。
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。

asmctl experimental describe pod

describe pod命令根据指定的容器组，分析与它相关的Kubernetes服务、目标规则和虚拟服务等，并最终输出与该容器组相关的描述性信息。

```
asmctl experimental describe pod <pod> [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--ignoreUnmeshed	无	对于分析时发现的未加入网格的容器组，设定是否输出警告信息，默认为 `false` 。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--asmconfig <string>	-m	ASM实例的kubeconfig文件路径，默认为 `$HOME/.kube/asmconfig` 。
--namespace <string>	-n	指定生效的命名空间，默认为空。

以下为asmctl experimental describe pod命令示例：

```
# 输出与容器组productpage-v1-c7765c886-7zzd4相关的描述性信息。
asmctl experimental describe pod productpage-v1-c7765c886-7zzd4
```

asmctl experimental describe service

describe service命令根据指定的Kubernetes服务，结合分析与它相关的容器组、目标规则和虚拟服务等，并最终输出与该Kubernetes服务相关的描述性信息。

```
asmctl experimental describe service <svc> [flags]
```

您也可以使用：

```
asmctl experimental describe svc <svc> [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--ignoreUnmeshed	无	对于分析时发现的未加入网格的容器组，设定是否输出警告信息，默认为 `false` 。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。

选项	选项缩写	描述
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--asmconfig <string>	-m	ASM实例的kubeconfig文件路径，默认为 `$HOME/.kube/asmconfig` 。
--namespace <string>	-n	指定生效的命名空间，默认为空。

以下为asmctl experimental describe service命令示例：

```
# 输出Kubernetes服务productpage相关的描述性信息。
asmctl experimental describe service productpage
```

asmctl experimental injector

injector系列命令负责与Sidecar注入情况以及Sidecar注入器信息有关的功能。

```
asmctl experimental injector [command] [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。

asmctl experimental injector list

injector list命令列出每个命名空间中容器组Sidecar的注入情况，并输出ASM使用的Sidecar注入器的概要信息。

```
asmctl experimental injector list [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。

以下为asmctl experimental injector list命令示例：

```
# 列出每个命名空间中容器组Sidecar的注入情况，并输出ASM使用的Sidecar注入器的概要信息。
asmctl experimental injector list
```

asmctl proxy-config

proxy-config系列命令负责获取容器组中Sidecar的配置信息。

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--output <string>	-o	设置命令输出格式，可选 `json` 和 `short` ，默认为 `short` 。

asmctl proxy-config bootstrap

proxy-config bootstrap命令负责获取指定容器组中Sidecar Envoy的bootstrap配置信息。

```
asmctl proxy-config bootstrap [<type>/]<name>[.<namespace>] [flags]
```

您也可以使用：

```
asmctl proxy-config b [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--file <string>	-f	Envoy配置转储文件，JSON格式，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--output <string>	-o	设置命令输出格式，可选 `json` 和 `short` ，默认为 `short` 。

以下为asmctl proxy-config bootstrap命令示例：

```
# 从指定的容器组中获取Sidecar Envoy的bootstrap配置信息。
asmctl proxy-config bootstrap <pod-name[.namespace]>
# 不使用Kubernetes API，直接从文件中分析得到Sidecar Envoy bootstrap配置信息。
ssh <user@hostname> 'curl localhost:15000/config_dump' > envoy-config.json
asmctl proxy-config bootstrap --file envoy-config.json
```

asmctl proxy-config cluster

proxy-config cluster命令负责获取指定容器组中Sidecar Envoy的集群配置信息。

```
asmctl proxy-config cluster [<type>/]<name>[.<namespace>] [flags]
```

您也可以使用：

```
asmctl proxy-config clusters [<type>/]<name>[.<namespace>] [flags]
asmctl proxy-config c [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--direction <string>	无	通过 `Direction` 字段筛选收集的cluster配置信息，默认为空。
--file <string>	-f	Envoy配置转储文件，JSON格式，默认为空。
--fqdn <string>	无	通过 `Service FQDN` 字段筛选收集的cluster配置信息，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--output <string>	-o	设置命令输出格式，可选 `json` 和 `short` ，默认为 `short` 。
--port <int>	无	通过 `Port` 字段筛选收集的cluster配置信息，默认为空。
--subset <string>	无	通过 `Subset` 字段筛选收集的cluster配置信息，默认为空。

以下为asmctl proxy-config cluster命令示例：

```
# 从指定的容器组中获取Sidecar Envoy的cluster配置信息。
asmctl proxy-config clusters <pod-name[.namespace]>
# 从指定的容器组中获取Sidecar Envoy的cluster配置信息，筛选端口为9080的部分输出。
asmctl proxy-config clusters <pod-name[.namespace]> --port 9080
# 获取FQDN为details.default.svc.cluster.local的入站集群的完整转储。
asmctl proxy-config clusters <pod-name[.namespace]> --fqdn details.default.svc.cluster.loca
l --direction inbound -o json
# 不使用Kubernetes API，仅使用文件获取cluster配置信息。
ssh <user@hostname> 'curl localhost:15000/config_dump' > envoy-config.json
asmctl proxy-config clusters --file envoy-config.json
```

asmctl proxy-config endpoint

proxy-config endpoint命令负责获取指定容器组中Sidecar Envoy的Endpoint配置信息。

```
asmctl proxy-config endpoint [<type>/]<name>[.<namespace>] [flags]
```

您也可以使用：

```
asmctl proxy-config endpoints [<type>/]<name>[.<namespace>] [flags]
```

```
asmctl proxy-config ep [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--address <string>	无	通过 `address` 字段筛选收集的Endpoint配置信息，默认为空。
--cluster <string>	无	通过 `cluster name` 字段筛选收集的Endpoint配置信息，默认为空。
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--file <string>	-f	Envoy配置转储文件，JSON格式，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--output <string>	-o	命令输出格式，可选 `json` 和 `short` ，默认为 `short` 。
--port <int>	无	通过 `Port` 字段筛选收集的Endpoint配置信息，默认为空。
--status <string>	无	通过 `status` 字段筛选收集的Endpoint配置信息，默认为空。

以下为asmctl proxy-config endpoint命令示例：

```
# 从指定的容器组中获取Sidecar Envoy的Endpoint配置信息。
asmctl proxy-config endpoint <pod-name[.namespace]>
# 从指定的容器组中获取Sidecar Envoy的Endpoint配置信息，筛选其中端口为9080的部分。
asmctl proxy-config endpoint <pod-name[.namespace]> --port 9080
# 从指定的容器组中获取Sidecar Envoy的Endpoint配置信息，筛选其中地址为172.17.0.2的部分。
asmctl proxy-config endpoint <pod-name[.namespace]> --address 172.17.0.2 -o json
# 从指定的容器组中获取Sidecar Envoy的Endpoint配置信息，筛选集群名称为outbound|9411||zipkin.istio-system.svc.cluster.local的部分。
asmctl proxy-config endpoint <pod-name[.namespace]> --cluster "outbound|9411||zipkin.istio-system.svc.cluster.local" -o json
# 从指定的容器组中获取Sidecar Envoy的Endpoint配置信息，筛选其中状态为healthy的部分。
asmctl proxy-config endpoint <pod-name[.namespace]> --status healthy -ojson
# 不使用Kubernetes API，仅使用文件获取Endpoint配置信息。
ssh <user@hostname> 'curl localhost:15000/clusters?format=json' > envoy-clusters.json
asmctl proxy-config endpoints --file envoy-clusters.json
```

asmctl proxy-config listener

proxy-config listener命令负责获取指定容器组中Sidecar Envoy的Listener配置信息。

```
asmctl proxy-config listener [<type>/]<name>[.<namespace>] [flags]
```

您也可以使用：

```
asmctl proxy-config listeners [<type>/]<name>[.<namespace>] [flags]
```

```
asmctl proxy-config l [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--address <string>	无	通过 `address` 字段筛选收集的Listener配置信息，默认为空。
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--file <string>	-f	Envoy配置转储文件，JSON格式，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--output <string>	-o	命令输出格式，可选 `json` 和 `short` ，默认为 `short` 。
--port <int>	无	通过 `Port` 字段筛选收集的Listener配置信息，默认为空。

选项	选项缩写	描述
--type <string>	无	通过 `type` 字段筛选收集的Listener配置信息，默认为空。
--verbose	无	输出更多信息，默认为 `true` 。

以下为asmctl proxy-config listener命令示例：

```
# 从指定的容器组中获取Sidecar Envoy的Listener配置信息。
asmctl proxy-config listeners <pod-name[.namespace]>
# 从指定的容器组中获取Sidecar Envoy的Listener配置信息，筛选其中端口为9080的部分。
asmctl proxy-config listeners <pod-name[.namespace]> --port 9080
# 使用通配符地址0.0.0.0 从指定的容器组中获取Sidecar Envoy的Listenerr配置信息。
asmctl proxy-config listeners <pod-name[.namespace]> --type HTTP --address 0.0.0.0 -o json
# 不使用Kubernetes API，仅使用文件获取Listener配置信息。
ssh <user@hostname> 'curl localhost:15000/config_dump' > envoy-config.json
asmctl proxy-config listeners --file envoy-config.json
```

asmctl proxy-config log

proxy-config log命令负责获取Sidecar Envoy的日志等级信息，并且也可以选择性地更新日志等级信息。

```
asmctl proxy-config log [<type>/]<name>[.<namespace>] [flags]
```

您也可以使用：

```
asmctl proxy-config o [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。

选项	选项缩写	描述
--level <string>	无	要输出的以逗号分隔的每个logger的最低消息级别，形式为 ：[<logger>:]<level>,[<logger>:]<level>,... 。其中，logger可以是 admin 、 aws 、 assert 、 backtrace 、 client 、 config 、 connection 、 conn_handler 、 dubbo 、 file 、 filter 、 forward_proxy 、 grpc 、 hc 、 health_checker 、 http 、 mongo 、 quic 、 pool 、 rbac 、 redis 、 router 、 runtime 、 stats 、 secret 、 tap 、 testing 、 thrift 、 tracing 、 upstream 、 udp 、 wasm 中的一个。而消息级别可以是 trace 、 debug 、 info 、 warning 、 error 、 critical 、 off 中的一个，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--output <string>	-o	设置命令输出格式，可选 json 和 short ，默认为 short 。
--reset	-r	将日志等级重设为默认值（ warning ）。
--selector <string>	-l	标签选择器，默认为空。

以下为asmctl proxy-config log命令示例：

```
# 从指定的容器组中获取Sidecar Envoy的日志等级信息。
asmctl proxy-config log <pod-name[.namespace]>
# 更新Sidecar Envoy中所有logger的日志消息等级。
asmctl proxy-config log <pod-name[.namespace]> --level none
# 为Sidecar Envoy中指定的logger的日志消息等级进行更新。
asmctl proxy-config log <pod-name[.namespace]> --level http:debug,redis:debug
# 将Sidecar Envoy中所有logger的日志消息等级重设为默认值warning。
asmctl proxy-config log <pod-name[.namespace]> -r
```

asmctl proxy-config route

proxy-config route命令负责获取指定容器组中Sidecar Envoy的Route配置信息。

```
asmctl proxy-config route [<type>/]<name>[.<namespace>] [flags]
```

您也可以使用：

```
asmctl proxy-config routes [<type>/]<name>[.<namespace>] [flags]
```

```
asmctl proxy-config r [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--file <string>	-f	Envoy配置转储文件，JSON格式，默认为空。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--name <string>	无	通过 `name` 字段筛选收集的Router配置信息，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--output <string>	-o	设置命令输出格式，可选 `json` 和 `short` ，默认为 `short` 。
--verbose	无	输出更多信息，默认为 `true` 。

以下为asmctl proxy-config route命令示例：

```
# 从指定的容器组中获取Sidecar Envoy的Route配置信息。
asmctl proxy-config routes <pod-name[.namespace]>
# 获取指定Sidecar Envoy的route 9080的配置信息。
asmctl proxy-config route <pod-name[.namespace]> --port 9080
# 获取指定Sidecar Envoy的route 9080的配置信息并获得完整转储。
asmctl proxy-config route <pod-name[.namespace]> --name 9080 -o json
# 不使用Kubernetes API，仅使用文件获取Route配置信息。
ssh <user@hostname> 'curl localhost:15000/config_dump' > envoy-config.json
asmctl proxy-config listeners --file envoy-config.json
```

asmctl proxy-config secret

proxy-config secret命令负责获取指定容器组中Sidecar Envoy的Secret配置信息。

```
asmctl proxy-config secret [<type>/]<name>[.<namespace>] [flags]
```

也可以使用：

```
asmctl proxy-config secrets [<type>/]<name>[.<namespace>] [flags]
asmctl proxy-config s [<type>/]<name>[.<namespace>] [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--file <string>	-f	Envoy配置转储文件，JSON格式，默认为空。

选项	选项缩写	描述
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--kubeconfig <string>	-c	设置用户侧集群的kubeconfig文件路径，默认为空。
--namespace <string>	-n	指定生效的命名空间，默认为空。
--output <string>	-o	设置命令输出格式，可选 `json` 和 `short` ，默认为 `short` 。

以下为asmctl proxy-config secret命令示例：

```
# 从指定的容器组中获取Sidecar Envoy的Secret配置信息。
asmctl proxy-config secret <pod-name[.namespace]>
# 不使用Kubernetes API，仅使用文件获取Secret配置信息。
ssh <user@hostname> 'curl localhost:15000/config_dump' > envoy-config.json
asmctl proxy-config listeners --file envoy-config.json
```

asmctl validate

validate命令负责验证服务网格规则与策略文件的正确性。

```
asmctl validate -f FILENAME [options] [flags]
```

您也可以使用：

```
asmctl v -f FILENAME [options] [flags]
```

选项	选项缩写	描述
--context <string>	无	设置默认使用的kubeconfig context名称，默认为空。
--file <string>	-f	需要验证的服务网格规格与策略文件名称。
--istioNamespace <string>	-i	设置Istio所在命名空间名称，默认为 `istio-system` 。
--namespace <string>	-n	指定生效的命名空间，默认为空。

以下为asmctl validate命令示例：

```
# 验证bookinfo-gateway.yaml的正确性。
asmctl validate -f samples/bookinfo/networking/bookinfo-gateway.yaml
# 使用缩短的语法验证bookinfo-gateway.yaml的正确性。
asmctl v -f samples/bookinfo/networking/bookinfo-gateway.yaml
# 验证当前default命名空间下所有部署的正确性。
asmctl get deployments -o yaml | asmctl validate -f -
# 验证当前default命名空间下所有服务的正确性。
asmctl get services -o yaml | asmctl validate -f -
```

2.ASM网关

2.1. 添加入口网关服务

如果部署的应用需要对公网提供访问，需要部署一个入口网关服务到集群中。本文介绍如何为ASM实例中的ACK集群添加入口网关服务。

前提条件

已创建至少一个ASM实例，并已添加至少一个ACK集群到该实例中。

背景信息

入口网关服务（Ingress Gateway）为Kubernetes集群提供了七层网关功能，对外提供一个统一的七层服务入口，根据HTTP请求的内容将来自同一个TCP端口的请求分发到不同的Kubernetes服务。

操作步骤

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**，然后在右侧页面单击**创建**。
5. 设置网关的基本信息。

> **说明** 您也可以通过单击**使用YAML创建**来定义入口网关服务，详情请参见自定义入口网关服务。

参数	说明
名称	自定义网关的名称。
部署集群	选择网关部署的集群。
网关类型	选择网关类型，支持入口网关和出口网关。
负载均衡类型	选择负载均衡的类型，可选**公网访问**或**私网访问**。

参数	说明
新建负载均衡	选择负载均衡，可选： ○ 使用已有负载均衡：从已有负载均衡列表中选择。 ○ 新建负载均衡：单击**新建负载均衡**，从下拉列表中选择所需的负载均衡规格。 **说明** 建议您为每个Kubernetes服务分配一个SLB。如果多个Kubernetes服务复用同一个SLB，存在以下风险和限制： ○ 使用已有的SLB会强制覆盖已有监听，可能会导致您的应用不可访问。 ○ Kubernetes通过Service创建的SLB不能复用，只能复用您手动在控制台（或调用OpenAPI）创建的SLB。 ○ 复用同一个SLB的多个Service不能有相同的前端监听端口，否则会造成端口冲突。 ○ 复用SLB时，监听的名字以及虚拟服务器组的名字被Kubernetes作为唯一标识符。请勿修改监听和虚拟服务器组的名字。 ○ 不支持跨集群、跨地域复用SLB。
端口映射	单击**添加端口**，在新增端口行中，输入服务端口和容器端口。 **说明** ○ 建议容器端口与服务端口一致，并在Istio网关资源定义中启用了该端口。 ○ 控制台默认提供了4个Istio常用的端口，但并不意味着必须从中选择，您可以根据需要自行添加或删除端口。
资源规格	选择网关Pod的CPU和内存规格。
网关副本数	设置网关副本数。

参数	说明
自动创建网关规则配置	设置是否自动创建同名的网关规则。

6. （可选）单击**高级选项**，设置以下参数。

参数	说明
外部流量策略	设置**外部流量策略**，可选： ○ **Local**：流量只发给本机的Pod。 ○ **Cluster**：流量可以转发到集群中其他节点上的Pod。
扩缩容HPA	选中**扩缩容HPA**，然后设置以下参数： **说明** 仅ASM商业版（专业版）支持该功能。 ○ **指标**：设置**监控项**和**阈值**，超过该阈值，将增加网关副本数。低于该阈值，将减少网关副本数。 如果您同时设置了CPU和内存的阈值，则表示只要其中一个高于或低于该阈值，都会进行扩缩容。 ○ **最大副本数**：网关可扩容的副本数量上限。 ○ **最小副本数**：网关可缩容的副本数量下限。
滚动升级	选中**滚动升级**后，设置以下参数： ○ **不可用最大副本数**：设置滚动升级时不可用最大的副本数量。 ○ **超过期望的副本数**：设置滚动升级时最多不能超过的副本数量。例如设置为25%，表示滚动升级时副本数量不能超过原来副本的125%。
TLS性能优化	使用TLS性能优化功能，将提升TLS加解密的速度。 选中**TLS性能优化**，选择节点亲和性标签，根据节点标签匹配性能优化的节点。 **说明** 仅ASM商业版（专业版）支持该功能，且您需要开启基于MultiBuffer的TLS加解密性能优化功能。
SLB优雅下线	选中**SLB优雅下线**后，当SLB停用时，将不会对网关业务产生影响。 **说明** 仅ASM商业版（专业版）支持该功能。

7. 单击**创建**。

执行结果

添加入口网关服务之后，可登录容器服务控制台查看详情。

- 查看新添加的入口网关服务的基本信息。
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
  iv. 在集群管理页左侧导航栏中选择**网络 > 服务**。
  v. 在**服务**页面，从**命名空间**下拉列表中选择**istio-system**。
  vi. 单击目标服务**操作**列的**详情**。

  在网关详情页面查看入口网关服务的详细信息，例如外部端点IP（入口网关IP）。
- 查看新添加入口网关服务的Pod信息。
  i. 登录容器服务管理控制台。
  ii. 在控制台左侧导航栏中，单击**集群**。
  iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
  iv. 在左侧导航栏选择**工作负载 > 容器组**。
  v. 在**容器组**页面，从**命名空间**下拉列表中选择**istio-system**。
  vi. 单击目标Pod操作列的**详情**，查看入口网关服务的Pod详细信息。

2.2. 修改入口网关服务

服务网格ASM支持修改入口网关服务的配置，本文介绍如何在服务网格ASM修改入口网关服务。

前提条件

已添加入口网关，详情请参见添加入口网关服务。

操作步骤

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**。
5. 在**ASM网关**页面单击目标入口网关**操作**列下的**YAML**。
6. 在**编辑**对话框修改参数，单击**确定**。

   参数说明

字段	说明	默认值
metadata.name	名称，生成的Kubernetes Service和Deployment名称为istio-{该值}。	无

字段	说明	默认值
metadata.namespace	命名空间，生成的Kubernetes Service和Deployment所在的命名空间。 **注意** 为兼容Istio 1.6及以后的版本，该命名空间必须为istio-system。	istio-system
clusterIds	数组类型。将部署入口网关的集群ID，这些集群隶属于当前网格实例所管理。	无
cpu.targetAverageUtilization	HPA支持CPU的阈值。	80
env	数组类型。入口网关Pod的环境变量。	无
externalTrafficPolicy	表示此服务是否希望将外部流量路由到节点本地或集群范围的端点。有两个可用选项：`Cluster` 和 `Local` 。	Local
ports	数组类型。入口网关Pod定义的端口和协议列表。例如： ○ `name: http2 port: 80 targetPort: 80 protocol: HTTP2` ○ `name: https port: 443 targetPort: 443 protocol: HTTPS` **说明** 1.9.7.107 之前版本，"protocol" 属性字段未做具体化声明，需统一声明配置为TCP	无
replicaCount	副本数。	1

configVolumes	入口网关Pod所使用到的ConfigMap挂载卷，例如：
	`- name: config-volume-lua`
	`configMapName: lua-libs`
	`mountPath: /var/lib/lua`	

resources	入口网关Pod的资源配置。	○ limits: ■ cpu: '2' ■ memory: 2G ○ requests: ■ cpu: 200m ■ memory: 256Mi
sds.enabled	是否启用SDS。	false

字段	说明	默认值
sds.resources	如果启用SDS，对应的Pod的资源配置。	◦ requests: ▪ cpu: 100m ▪ memory: 128Mi ◦ requests: ▪ cpu: 2000m ▪ memory: 1024Mi

secretVolumes	入口网关Pod所使用到的Secret挂载卷，例如：
	`- name: myexample-customingressgateway-certs`
	`secretName: istio-myexample-customingressgateway-certs`
	`mountPath: /etc/istio/myexample-customingressgateway-certs`
		无

serviceType	入口网关的服务类型，可以是 `LoadBalancer` 、`Nodeport` 或者 `ClusterIP` 。	LoadBalancer
serviceAnnotations	入口网关服务的Annotation定义。例如：`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-connection-drain: 'on' service.beta.kubernetes.io/alibaba-cloud-loadbalancer-connection-drain-timeout: '20'` **说明** 关于Annotation的常用注解，请参见通过Annotation配置负载均衡。	无
serviceLabels	入口网关服务的Label定义。	无
podAnnotations	入口网关Pod的Annotation定义。	无
rollingMaxSurge	滚动更新过程中运行操作期望副本数的最大Pod数，可以为绝对数值，也可以为百分数。	"100%"
rollingMaxUnavailable	滚动更新过程中不可用的最大Pod数，可以为绝对数值，也可以为百分数。	"25%"

字段	说明	默认值
overrides	当 `clusterIds` 指定了2个或以上的集群时，可以针对特定的集群指定不同于上述参数定义的配置值，配置值为Map类型。 **说明** • `key`：本次定义的 `clusterIds` 中某一个集群ID。 • `value`：支持 `serviceAnnotations`、`resources`、`replicaCount` 参数的赋值。	无
kernel.enabled	是否启用自定义内核参数。	false

字段	说明	默认值
kernel.parameters	内核参数设置，当前支持设置以下内核参数： **注意** ASM支持的内核参数修改项可能因宿主机内核版本不同，而出现部分参数不支持的情况。如果出现这种情况，入口网格Pod可能会报错。 您可以通过 `kubectl describe pod` 命令查看入口网关报错情况。删除不支持的参数后，容器即可正常启动。 所有的内核参数值为字符串格式，因YAML语法会将纯数字解析为数值类型，您需要使用""（半角双引号）包裹您的值，例如net.core.somaxconn: "65535"。 • `net.core.somaxconn` • `net.core.netdev_max_backlog` • `net.ipv4.tcp_rmem` • `net.ipv4.tcp_wmem` • `net.ipv4.ip_local_port_range` • `net.ipv4.tcp_fin_timeout` • `net.ipv4.tcp_tw_timeout` • `net.ipv4.tcp_tw_reuse` • `net.ipv4.tcp_tw_recycle` • `net.ipv4.tcp_timestamps` • `net.ipv4.tcp_retries2` • `net.ipv4.tcp_slow_start_after_idle` • `net.ipv4.tcp_max_orphans` • `net.ipv4.tcp_max_syn_backlog` • `net.ipv4.tcp_no_metrics_save` • `net.ipv4.tcp_autocorking` • `kernel.printk` • `vm.swappiness`	无
compression.enabled	是否启用入口网关压缩能力。	false
compression.content_type	需要被压缩的ContentType列表，例如： • `text/html` • `application/json`	无

字段	说明	默认值
compression.disable_on_etag_header	设置为 `true` 时，当Response中存在etag_header时禁用压缩。	false
compression.min_content_length	ContentLength大于等于设置的值时触发压缩。	30
compression.remove_accept_encoding_header	◦ 设置为 `true` 时，入口网关在将客户端请求转发至上游之前会将请求内的Accept-Encoding Header移除。 ◦ 设置为 `false` 时，入口网关在将客户端请求转发至上游之前会保留请求内的Accept-Encoding Header。	false
compression.gzip	当前仅支持gzip压缩格式，要启用压缩，该字段必须填写，如所有参数保持默认，也需要填写空结构。例如 `gzip: {}` 。	无
compression.gzip.memory_level	zlib内部的内存使用级别，合法值1~9，值越大占用内存越多，同时也带来更快的压缩速度和更好的压缩质量。	5

字段	说明	默认值
compression.gzip.compression_level	zlib的压缩级别，合法值如下： **说明** DEFAULT_COMPRESSION是默认压缩值。BEST_COMPRESSION是最高压缩质量。BEST_SPEED是最快的压缩速度。其中： ○ COMPRESSION_LEVEL_1压缩级别等价于BEST_SPEED。 ○ COMPRESSION_LEVEL_9压缩级别等价于BEST_COMPRESSION。 ○ COMPRESSION_LEVEL_6压缩级别等价于DEFAULT_COMPRESSION。 ○ `COMPRESSION_LEVEL_1` ○ `COMPRESSION_LEVEL_2` ○ `COMPRESSION_LEVEL_3` ○ `COMPRESSION_LEVEL_4` ○ `COMPRESSION_LEVEL_5` ○ `COMPRESSION_LEVEL_6` ○ `COMPRESSION_LEVEL_7` ○ `COMPRESSION_LEVEL_8` ○ `COMPRESSION_LEVEL_9` ○ `DEFAULT_COMPRESSION` ○ `BEST_COMPRESSION` ○ `BEST_SPEED`	DEFAULT_COMPRESSION
compression.gzip.compression_strategy	zlib的压缩策略，合法值如下： ○ `FILTERED` ○ `FIXED` ○ `HUFFMAN_ONLY` ○ `RLE`	DEFAULT_STRATEGY
compression.gzip.window_bits	zlib window size，合法值为9~15。	12
compression.gzip.chunk_size	zlib输出缓冲区大小。	4096
hostNetwork	主机网络，当 `hostNetwork` 设置为 `true` 时，入口网关Pod将使用宿主机的网络。	false
dnsPolicy	DNS策略。关于dnsPolicy的详细介绍，请参见DNS for Services and Pods。	ClusterFirst

2.3. 删除入口网关服务

入口网关是服务网格的一种资源对象，用于管理服务网格的流量。您可以通过入口网关访问网格内的服务。本文介绍如何删除入口网关服务。

操作步骤

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**。
5. 单击目标入口网关**操作**列的**删除**。
6. 在**确认**对话框中单击**确定**。

2.4. 通过服务网关启用HTTPS安全服务

通过使用SDS（Secret Discovery Service）为服务网关提供HTTPS安全支持、证书动态加载，从而提升服务网关安全性。本文介绍如何通过SDS配置TLS入口网关。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。
- 已部署应用到ASM实例的集群中。具体操作，请参见部署应用到ASM实例。
- 使用域名时需要备案才能正常访问，本示例中使用*aliyun.com*。

背景信息

SDS（Secrete Discovery Service）是Istio提供的一种动态加载证书的方式，TLS （Transport Layer Security）所需的私钥、服务器证书以及根证书都可以由SDS完成配置。

入口网关代理与入口网关在同一Pod中运行，并监视与入口网关相同的命名空间中创建的Secret。在入口网关上启用SDS具有以下好处：

- 入口网关可以在不需要重启的情况下，动态添加、删除或更新所需要的证书、私钥或者对应的根证书。
- 不需要Secret卷挂载。创建Kubernetes Secret后，网关代理会捕获该Secret，并将其包含的证书、私钥或根证书发送到入口网关。
- 网关代理可以监视多个证书、私钥对，只需要为多个主机创建Secret并更新网关定义。

步骤一：为多个主机准备服务器证书和私钥

为`aliyun.com`生成证书和私钥，并保存为Secret。

如果您已经拥有针对`aliyun.com`可用的证书和私钥，需要将密钥命名为`aliyun.com.key`，证书命名为`aliyun.com.crt`。如果没有，可以通过openssl执行以下步骤来生成证书和密钥。

1. 执行以下命令，创建根证书和私钥。

```
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -subj '/O=myexample Inc./CN
=aliyun.com' -keyout aliyun.root.key -out aliyun.root.crt
```

2. 执行以下命令，为 `aliyun.com` 服务器生成证书和私钥。

```
openssl req -out aliyun.com.csr -newkey rsa:2048 -nodes -keyout aliyun.com.key -subj "/
CN=aliyun.com/O=myexample organization"
openssl x509 -req -days 365 -CA aliyun.root.crt -CAkey aliyun.root.key -set_serial 0 -i
n aliyun.com.csr -out aliyun.com.crt
```

3. 通过以下命令（注意是在入口网关Pod所在的集群对应的kubeconfig环境下）将在istio-system命名空间中创建包含证书和私钥的Secret。

```
kubectl create -n istio-system secret tls myexample-credential --key=aliyun.com.key --c
ert=aliyun.com.crt
```

> 注意 Secret名称不应以istio或prometheus开头，且不应包含token字段。

步骤二：为a.aliyun.com定义内部服务

示例中的内部服务是基于Nginx实现的，首先为Nginx服务器创建配置文件。以域名 `a.aliyun.com` 的内部服务为例，定义请求根路径直接返回字样Welcome to a.aliyun.com!及状态码200。*myexample-nginx.conf*的具体内容如下。

```
events {
}
http {
  log_format main '$remote_addr - $remote_user [$time_local]  $status '
  '"$request" $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log /var/log/nginx/access.log main;
  error_log  /var/log/nginx/error.log;
  server {
    listen 80;
    location /hello {
        return 200 'Welcome to a.aliyun.com!';
        add_header Content-Type text/plain;
    }
  }
}
```

1. 在入口网关Pod所在的集群对应的kubeconfig环境下，执行以下命令，创建Kubernetes ConfigMap，即存储Nginx服务器的配置。

```
kubectl create configmap myexample-nginx-configmap --from-file=nginx.conf=./myexample-n
ginx.conf
```

2. 在ASM控制台选中对应的服务网格实例，左侧导航栏选中命名空间，设置命名空间default启用sidecar自动注入。
3. 创建并拷贝以下内容到myexampleapp.yaml文件中，并执行 `kubectl apply -f myexampleapp.yaml` 命令，创建域名a.aliyun.com的内部服务。

```
apiVersion: v1
kind: Service
metadata:
  name: myexampleapp
  labels:
    app: myexampleapp
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: myexampleapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myexampleapp
spec:
  selector:
    matchLabels:
      app: myexampleapp
  replicas: 1
  template:
    metadata:
      labels:
        app: myexampleapp
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx
          readOnly: true
      volumes:
      - name: nginx-config
        configMap:
          name: myexample-nginx-configmap
```

步骤三：为b.aliyun.com定义内部服务

本示例中的内部服务是基于httpbin实现的，创建并拷贝如下内容到httpbin.example.yaml文件中，并执行 `kubectl apply -f httpbin.example.yaml` 命令（注意是在入口网关Pod所在的集群对应的kubeconfig环境下），创建域名 `b.aliyun.com` 的内部服务。

```
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  labels:
    app: httpbin
spec:
  ports:
  - name: http
    port: 8000
  selector:
    app: httpbin
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
      version: v1
  template:
    metadata:
      labels:
        app: httpbin
        version: v1
    spec:
      containers:
      - image: docker.io/citizenstig/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        ports:
        - containerPort: 8000
```

步骤四：创建自定义网关配置对象

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**流量管理 > 网关规则**，然后在右侧页面单击**使用YAML创建**。
5. 按以下步骤定义服务网关，然后单击**创建**。
    i. 选择相应的命名空间。本文以选择**default**命名空间为例。

ii. 在文本框中，定义服务网关。可参考以下YAML定义：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: mysdsgateway
spec:
  selector:
    istio: ingressgateway # use istio default ingress gateway
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: myexample-credential # must be the same as secret
    hosts:
    - '*.aliyun.com'
```

在网关规则页面可以看到新建的网关。

步骤五：创建虚拟服务

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**流量管理 > 虚拟服务**，然后在右侧页面单击**使用YAML创建**。
5. 按以下步骤定义虚拟服务，然后单击**创建**。
   i. 选择相应的命名空间。本文以选择**default**命名空间为例。

ii. 在文本框中，定义Istio虚拟服务。可参考以下YAML定义。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: mysdsgateway-myexamplevs
spec:
  hosts:
  - "a.aliyun.com"
  gateways:
  - mysdsgateway
  http:
  - match:
    - uri:
        exact: /hello
    route:
    - destination:
        host: myexampleapp.default.svc.cluster.local
        port:
          number: 80
```

同样地，为httpbin.example.com提供定义相应的虚拟服务。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: mysdsgateway-httpbinvs
spec:
  hosts:
  - "b.aliyun.com"
  gateways:
  - mysdsgateway
  http:
  - match:
    - uri:
        prefix: /status
    - uri:
        prefix: /delay
    route:
    - destination:
        port:
          number: 8000
        host: httpbin.default.svc.cluster.local
```

在虚拟服务页面可以看到新建的虚拟服务。

执行结果

可以通过以下任意一种方法获取入口网关服务的地址：

- 通过ASM控制台查看，在控制台选中对应的服务网格实例，左侧导航栏选中ASM网关，在右侧页面中查看对应的信息。
- 通过容器服务控制台查看，请参见查看入口网关的服务信息。

可以通过以下命令访问入口网关服务：

- 执行以下命令，通过HTTPS协议访问aliyun.com服务。

```
curl -k -H Host:a.aliyun.com --resolve a.aliyun.com:443:{替换成真实的入口网关IP地址} https://a.aliyun.com/hello
```

预期输出：

```
Welcome to aliyun.com!
```

- 执行以下命令，通过HTTPS协议访问httpbin.example.com服务。

```
curl -k -H Host:b.aliyun.com --resolve b.aliyun.com:443:{替换成真实的入口网关IP地址} https://b.aliyun.com/status/418
```

预期输出：

```
    -=[ teapot ]=-
       _...._
     .'  _ _ `.
    | ."` ^ `". _,
    \_;`"---"`|//
      |       ;/
      \_     _/
        `"""`
```

更新网关证书

如果您重建了挂载证书的数据面Secret，需要手动将网关中credentialName的参数值替换成新的Secret名称，才能更新网关证书。本文以为example.com服务器创建名为new-istio-ingressgateway-certs的Secret为例。

1. 创建颁发者为myexample的证书。
   i. 在OpenSSL工具中执行以下命令，创建根证书和私钥。

   ```
   openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -subj '/O=myexample Inc./CN=example.com' -keyout example.root.key -out example.root.crt
   ```

   ii. 执行以下命令，为example.com服务器生成证书和私钥。

   ```
   openssl req -out example.com.csr -newkey rsa:2048 -nodes -keyout example.com.key -subj "/CN=example.com/O=myexample organization"
   openssl x509 -req -days 365 -CA example.root.crt -CAkey example.root.key -set_serial 0 -in example.com.csr -out example.com.crt
   ```

   iii. 通过kubectl工具连接集群。
   iv. 执行以下命令，创建名为new-istio-ingressgateway-certs的Secret。

   ```
   kubectl create -n istio-system secret tls new-istio-ingressgateway-certs --key example.com.key --cert example.com.crt
   ```

2. 在集群中执行以下命令，删除旧的证书Secret。

   ```
   kubectl delete secret  istio-ingressgateway-certs -n istio-system
   ```

3. 更新网关规则。
   i. 登录ASM控制台。

ii. 在左侧导航栏，选择**服务网格 > 网格管理**。

iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

iv. 在网格详情页面左侧导航栏选择**流量管理 > 网关规则**。

v. 在**网关规则**页面单击目标规则右侧**操作**列下的**YAML**。

vi. 在**编辑**面板修改credentialName参数值为新的证书Secret名称new-istio-ingressgateway-certs，然后单击**确定**。

4. 验证更新网关证书是否成功。

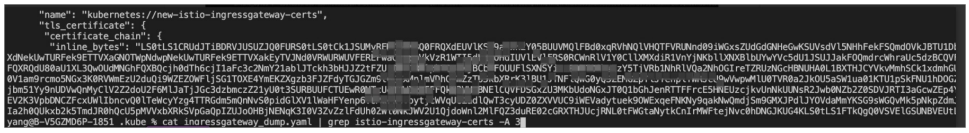
i. 执行以下命令，在集群中查看当前证书信息。

```
kubectl exec istio-ingressgateway-xxxx -n istio-system -- curl localhost:15000/conf
ig_dump  > ingressgateway_dump.yaml
```

ii. 执行以下命令，打印并搜索new-istio-ingressgateway-certs证书。

```
cat ingressgateway_dump.yaml | grep new-istio-ingressgateway-certs -A 3
```

预期可以看到以下内容：



复制以上inline_bytes参数后的内容，获取Base64编码后的证书。

iii. 在本地命令行工具中执行以下命令，将证书进行Base64解码。

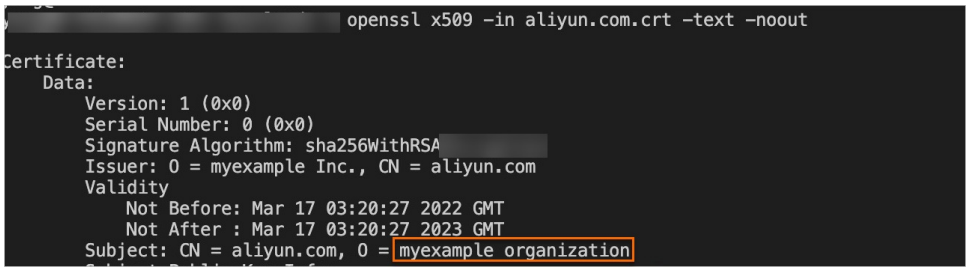
```
echo <Base64编码后的证书内容> | base64 --decode
```

iv. 将Base64解码内容保存为*test.com.crt*文件。

v. 在OpenSSL工具中执行以下命令，查看证书的组织。

```
openssl x509 -in test.com.crt -text -noout
```

预期输出：



可以看到组织成功更换为myexample，说明网关证书更新成功。

2.5. 自定义入口网关服务

阿里云服务网格ASM（Alibaba Cloud Service Mesh）除了可以通过控制台创建默认的入口网关服务之外，还支持通过CRD方式管理自定义网关。本文介绍如何自定义入口网关服务。

前提条件

- 创建ASM实例，请参见创建ASM实例。

- 部署应用到ASM实例的集群中，请参见部署应用到ASM实例。
- 新增入口网关必须创建在命名空间istio-system中，以获取相关的配置信息。如果部署到其他命名空间，在Istio 1.6及以后的版本中，将因为不能获取相关配置而导致入口网关无法正常启动。

背景信息

ASM支持通过CRD方式管理自定义网关。ASM提供了一个 `kind` 为 `IstioGateway` 、 `apiVersion` 为 `istio.alibabacloud.com/v1beta1` 的自定义资源定义CRD，并提供了相应的Controller。通过监听该CRD资源变化事件，对应的Controller可以在相应的Kubernetes集群中同步对应的Service、Deployment以及相关联的ServiceAccount等。

部署自定义网关

1. 创建并拷贝以下内容到*myexample-customingressgateway.yaml*文件中。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
metadata:
  name: "myexample-customingressgateway"
  namespace: "istio-system"
spec:
  clusterIds:
    - "cluster1Id"
    - "cluster2Id"
  cpu:
    targetAverageUtilization: 80
  env:
    - name: "envname1"
      value: "envvalue1"
  externalTrafficPolicy: Local
  ports:
  - name: status-port
    port: 15020
    targetPort: 15020
  - name: http2
    port: 80
    targetPort: 80
  - name: https
    port: 443
    targetPort: 0
  - name: tls
    port: 15443
    targetPort: 15443
  replicaCount: 1
  resources:
    limits:
      cpu: '2'
      memory: 2G
    requests:
      cpu: 200m
      memory: 256Mi
  sds:
    enabled: false
    resources:
      requests:
```

```
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 2000m
        memory: 1024Mi
# - name: config-volume-lua
#  configMapName: lua-libs
#  mountPath: /var/lib/lua
# secretVolumes:
# - name: myexample-customingressgateway-certs
#   secretName: istio-myexample-customingressgateway-certs
#   mountPath: /etc/istio/myexample-customingressgateway-certs
  serviceType: LoadBalancer
  serviceAnnotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-address-type: internet
  serviceLabels:
    serviceLabelKey1: "serviceLabelValue1"
  podAnnotations:
    podAnnotationsKey1: "podAnnotationsValue1"
  rollingMaxSurge: "100%"
  rollingMaxUnavailable: "25%"
  overrides:
    cluster1Id:
      replicaCount: 1
      resources:
        limits:
          cpu: '2'
          memory: 2G
        requests:
          cpu: 200m
          memory: 256Mi
      serviceAnnotations:
        service.beta.kubernetes.io/alicloud-loadbalancer-address-type: internet
        service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: "slb.s1.small"
    cluster2Id:
      replicaCount: 2
      resources:
        limits:
          cpu: '4'
          memory: 4G
        requests:
          cpu: 400m
          memory: 512Mi
      serviceAnnotations:
        service.beta.kubernetes.io/alicloud-loadbalancer-address-type: internet
        service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: "slb.s2.small"
  hostNetwork: true
  dnsPolicy: "ClusterFirstWithHostNet"
```

参数说明

字段	说明	默认值
metadata.name	名称，生成的Kubernetes Service和Deployment名称为istio-{该值}。	无
metadata.namespace	命名空间，生成的Kubernetes Service和Deployment所在的命名空间。 **注意** 为兼容Istio 1.6及以后的版本，该命名空间必须为istio-system。	istio-system
clusterIds	数组类型。将部署入口网关的集群ID，这些集群隶属于当前网格实例所管理。	无
cpu.targetAverageUtilization	HPA支持CPU的阈值。	80
env	数组类型。入口网关Pod的环境变量。	无
externalTrafficPolicy	表示此服务是否希望将外部流量路由到节点本地或集群范围的端点。有两个可用选项：`Cluster` 和 `Local` 。	Local
ports	数组类型。入口网关Pod定义的端口和协议列表。例如： • `name: http2 port: 80 targetPort: 80 protocol: HTTP2` • `name: https port: 443 targetPort: 443 protocol: HTTPS` **说明** 1.9.7.107 之前版本，"protocol" 属性字段未做具体化声明，需统一声明配置为TCP	无
replicaCount	副本数。	1

configVolumes	入口网关Pod所使用到的ConfigMap挂载卷，例如：
	`- name: config-volume-lua`
	`configMapName: lua-libs`
	`mountPath: /var/lib/lua`	

字段	说明	默认值
resources	入口网关Pod的资源配置。	◦ limits: ▪ cpu: '2' ▪ memory: 2G ◦ requests: ▪ cpu: 200m ▪ memory: 256Mi
sds.enabled	是否启用SDS。	false
sds.resources	如果启用SDS，对应的Pod的资源配置。	◦ requests: ▪ cpu: 100m ▪ memory: 128Mi ◦ requests: ▪ cpu: 2000m ▪ memory: 1024Mi

secretVolumes	入口网关Pod所使用到的Secret挂载卷，例如：
	`- name: myexample-customingressgateway-certs`
	`secretName: istio-myexample-customingressgateway-certs`
	`mountPath: /etc/istio/myexample-customingressgateway-certs`
		无

serviceType	入口网关的服务类型，可以是 `LoadBalancer` 、`Nodeport` 或者 `ClusterIP` 。	LoadBalancer
serviceAnnotations	入口网关服务的Annotation定义。例如：`service.beta.kubernetes.io/alibaba-cloud-loadbalancer-connection-drain: 'on' service.beta.kubernetes.io/alibaba-cloud-loadbalancer-connection-drain-timeout: '20'` **说明** 关于Annotation的常用注解，请参见通过Annotation配置负载均衡。	无
serviceLabels	入口网关服务的Label定义。	无
podAnnotations	入口网关Pod的Annotation定义。	无
rollingMaxSurge	滚动更新过程中运行操作期望副本数的最大Pod数，可以为绝对数值，也可以为百分数。	"100%"

字段	说明	默认值
rollingMaxUnavailable	滚动更新过程中不可用的最大Pod数，可以为绝对数值，也可以为百分数。	"25%"
overrides	当 `clusterIds` 指定了2个或以上的集群时，可以针对特定的集群指定不同于上述参数定义的配置值，配置值为Map类型。 **说明** ◦ `key` ：本次定义的 `clusterIds` 中某一个集群ID。 ◦ `value` ：支持 `serviceAnnotations` 、 `resources` 、 `replicaCount` 参数的赋值。	无
kernel.enabled	是否启用自定义内核参数。	false

字段	说明	默认值
kernel.parameters	内核参数设置，当前支持设置以下内核参数： **注意** ASM支持的内核参数修改项可能因宿主机内核版本不同，而出现部分参数不支持的情况。如果出现这种情况，入口网格Pod可能会报错。 您可以通过 `kubectl describe pod` 命令查看入口网关报错情况。删除不支持的参数后，容器即可正常启动。 所有的内核参数值为字符串格式，因YAML语法会将纯数字解析为数值类型，您需要使用""（半角双引号）包裹您的值，例如net.core.somaxconn: "65535"。 • `net.core.somaxconn` • `net.core.netdev_max_backlog` • `net.ipv4.tcp_rmem` • `net.ipv4.tcp_wmem` • `net.ipv4.ip_local_port_range` • `net.ipv4.tcp_fin_timeout` • `net.ipv4.tcp_tw_timeout` • `net.ipv4.tcp_tw_reuse` • `net.ipv4.tcp_tw_recycle` • `net.ipv4.tcp_timestamps` • `net.ipv4.tcp_retries2` • `net.ipv4.tcp_slow_start_after_idle` • `net.ipv4.tcp_max_orphans` • `net.ipv4.tcp_max_syn_backlog` • `net.ipv4.tcp_no_metrics_save` • `net.ipv4.tcp_autocorking` • `kernel.printk` • `vm.swappiness`	无
compression.enabled	是否启用入口网关压缩能力。	false
compression.content_type	需要被压缩的ContentType列表，例如： • `text/html` • `application/json`	无

字段	说明	默认值
compression.disable_on_etag_header	设置为 `true` 时，当Response中存在etag_header时禁用压缩。	false
compression.min_content_length	ContentLength大于等于设置的值时触发压缩。	30
compression.remove_accept_encoding_header	◦ 设置为 `true` 时，入口网关在将客户端请求转发至上游之前会将请求内的Accept-Encoding Header移除。 ◦ 设置为 `false` 时，入口网关在将客户端请求转发至上游之前会保留请求内的Accept-Encoding Header。	false
compression.gzip	当前仅支持gzip压缩格式，要启用压缩，该字段必须填写，如所有参数保持默认，也需要填写空结构。例如 `gzip: {}` 。	无
compression.gzip.memory_level	zlib内部的内存使用级别，合法值1~9，值越大占用内存越多，同时也带来更快的压缩速度和更好的压缩质量。	5

字段	说明	默认值
compression.gzip.compression_level	zlib的压缩级别，合法值如下： **说明** DEFAULT_COMPRESSION是默认压缩值。BEST_COMPRESSION是最高压缩质量。BEST_SPEED是最快的压缩速度。其中： ○ COMPRESSION_LEVEL_1压缩级别等价于BEST_SPEED。 ○ COMPRESSION_LEVEL_9压缩级别等价于BEST_COMPRESSION。 ○ COMPRESSION_LEVEL_6压缩级别等价于DEFAULT_COMPRESSION。 ○ `COMPRESSION_LEVEL_1` ○ `COMPRESSION_LEVEL_2` ○ `COMPRESSION_LEVEL_3` ○ `COMPRESSION_LEVEL_4` ○ `COMPRESSION_LEVEL_5` ○ `COMPRESSION_LEVEL_6` ○ `COMPRESSION_LEVEL_7` ○ `COMPRESSION_LEVEL_8` ○ `COMPRESSION_LEVEL_9` ○ `DEFAULT_COMPRESSION` ○ `BEST_COMPRESSION` ○ `BEST_SPEED`	DEFAULT_COMPRESSION
compression.gzip.compression_strategy	zlib的压缩策略，合法值如下： ○ `FILTERED` ○ `FIXED` ○ `HUFFMAN_ONLY` ○ `RLE`	DEFAULT_STRATEGY
compression.gzip.window_bits	zlib window size，合法值为9~15。	12
compression.gzip.chunk_size	zlib输出缓冲区大小。	4096
hostNetwork	主机网络，当 `hostNetwork` 设置为 `true` 时，入口网关Pod将使用宿主机的网络。	false
dnsPolicy	DNS策略。关于dnsPolicy的详细介绍，请参见DNS for Services and Pods。	ClusterFirst

2. 使用kubectl切换到服务网格实例对应的kubeconfig环境下，请参见通过kubectl连接ASM实例。
3. 创建命名空间myexample，请参见新建命名空间。
4. 在kubectl中执行 `kubectl apply -f myexample-customingressgateway.yaml` 命令，创建自定义入口网关。

执行结果

添加入口网关之后，可登录容器服务控制台查看详情。

查看新添加入口网关的服务信息。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 在**服务**页面，从**命名空间**下拉列表中选择**myexample**。
6. 单击目标服务**操作**列的**详情**，查看新添加入口网关的服务信息。

查看新添加入口网关的Pod信息。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**。
5. 在**容器组**页面，从**命名空间**下拉列表中选择**myexample**。
6. 单击目标Pod**操作**列的**详情**，查看新添加入口网关的Pod信息。

2.6. 自定义出口网关服务

出口网关可以让ASM内部服务访问外部服务。本文介绍如何自定义出口网关服务。

前提条件

已创建ASM实例，并添加集群到ASM实例。具体操作，请参见创建ASM实例和添加集群到ASM实例。

操作步骤

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**，然后在右侧页面单击**使用YAML创建**。
5. 设置命名空间为istio-system，将以下内容复制到文本框中，然后单击**创建**。

> **说明** 部署出口网关时必须设置命名空间为istio-system，以获取相关的配置信息。如果部署到其他命名空间，在Istio 1.6及以后的版本中，将因为不能获取相关配置而导致出口网关无法正常启动。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
metadata:
  name: egressgateway
  namespace: istio-system
spec:
  ports:
    - name: http2
      port: 80
      targetPort: 80
    - name: http-sw
      port: 11800
      targetPort: 11800
    - name: https
      port: 443
      targetPort: 443
    - name: tls
      port: 15443
      targetPort: 15443
# - name: config-volume-lua
#  configMapName: lua-libs
#  mountPath: /var/lib/lua
# secretVolumes:
# - name: myexample-customingressgateway-certs
#   secretName: istio-myexample-customingressgateway-certs
#   mountPath: /etc/istio/myexample-customingressgateway-certs
  replicaCount: 1
  resources:
    limits:
      cpu: '2'
      memory: 2G
    requests:
      cpu: 200m
      memory: 256Mi
  runAsRoot: false
  serviceType: ClusterIP
  hostNetwork: true
  dnsPolicy: "ClusterFirstWithHostNet"
```

参数说明

字段	说明	默认值
metadata.name	出口网关名称，生成的Kubernetes Service和Deployment名称为istio-{该值}。	无
metadata.namespace	命名空间，生成的Kubernetes Service和Deployment所在的命名空间。 **注意** 为兼容Istio 1.6及以后的版本，该命名空间必须为istio-system。	istio-system

字段	说明	默认值
clusterIds	数组类型。将部署入口网关的集群ID，这些集群隶属于当前网格实例所管理。	无
cpu.targetAverageUtilization	HPA支持CPU的阈值。	80
env	数组类型。出口网关Pod的环境变量。	无
ports	数组类型。出口网关Pod定义的端口列表。例如： • `name: status-port port: 15020 targetPort: 15020` • `name: http2 port: 80 targetPort: 80` • `name: https port: 443 targetPort: 0` • `name: tls port: 15443 targetPort: 15443`	无
replicaCount	副本数。	1

resources	出口网关Pod的资源配置。	• limits:
		  ▪ cpu: '2'
		  ▪ memory: 2G
		• requests:
		  ▪ cpu: 200m
		  ▪ memory: 256Mi

configVolumes	出口网关Pod所使用到的ConfigMap挂载卷，例如：
	`- name: config-volume-lua`
	`  configMapName: lua-libs`
	`  mountPath: /var/lib/lua`
		无

secretVolumes	出口网关Pod所使用到的Secret挂载卷，例如：
	`- name: myexample-customingressgateway-c`
	`  secretName: istio-myexample-customingressgateway-certs`
	`  mountPath: /etc/istio/myexample-customingressgateway-certs`
		无

serviceType	出口网关的服务类型，可以是LoadBalancer、Nodeport或者ClusterIP。	ClusterIP

字段	说明	默认值
serviceAnnotations	出口网关服务的Annotation定义。例如 `service.beta.kubernetes.io/alicloud-loadbalancer-address-type: internet` 。	无
serviceLabels	出口网关服务的Label定义。	无
podAnnotations	出口网关Pod的Annotation定义。	无
rollingMaxSurge	滚动更新过程中运行操作期望副本数的最大Pod数，可以为绝对数值，也可以为百分数。	"100%"
rollingMaxUnavailable	滚动更新过程中不可用的最大Pod数，可以为绝对数值，也可以为百分数。	"25%"
overrides	当 `clusterIds` 指定了2个或以上的集群时，可以针对特定的集群指定不同于上述参数定义的配置值，配置值为Map类型。 **说明** ◦ key：本次定义的clusterIds中某一个集群ID。 ◦ value：支持serviceAnnotations、resources、replicaCount参数的赋值。	无
hostNetwork	主机网络，当 `hostNetwork` 设置为 `true` 时，出口网关Pod将使用宿主机的网络。	false
dnsPolicy	DNS策略。关于dnsPolicy的详细介绍，请参见DNS for Services and Pods。	ClusterFirst

结果验证

出口网关部署成功后，您可以在ACK查看出口网关的服务和Pod信息。

查看出口网关的服务信息

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 在**服务**页面顶部设置**命名空间**为istio-system，在服务列表可以看到出口网关的服务信息。

查看出口网关的Pod信息

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**工作负载 > 容器组**。

5. 在**容器组**页面顶部设置**命名空间**为istio-system，在Pod列表可以看到出口网关的Pod信息。

2.7. 创建IPv6网关

相比IPv4地址，IPv6地址具有更大地址空间，更高的安全性。本文介绍如何创建IPv6地址的ASM网关，和为已有ASM网关添加IPv6地址。

前提条件

添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。

创建IPv6地址的ASM网关

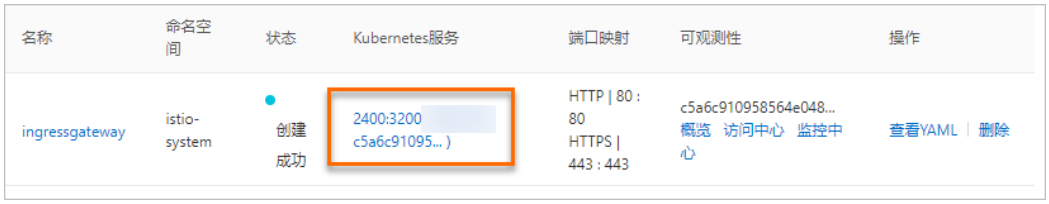
您需要在创建ASM网关时添加 `service.beta.kubernetes.io/alibaba-cloud-loadbalancer-ip-version: "ipv6"` 注解，声明该网关使用IPv6地址。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**，然后在右侧页面单击**使用YAML创建**。
5. 在**创建**页面设置**命名空间**为istio-system，在文本框中输入以下内容，单击**创建**。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
metadata:
  name: ingressgateway
  namespace: istio-system
spec:
  gatewayType: ingress
  clusterIds:
    - c808cdd6abd854d5ba6764da5ca2e****          #数据面集群ID。
  ports:
    - name: http-0
      port: 80
      targetPort: 80
      protocol: HTTP
    - name: https-1
      port: 443
      targetPort: 443
      protocol: HTTPS
  serviceAnnotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-address-type: internet
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: slb.s1.small
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-ip-version: "ipv6"
  replicaCount: 1
  resources:
    limits:
      cpu: '2'
      memory: 4G
    requests:
      cpu: 200m
      memory: 256Mi
  serviceType: LoadBalancer
  autoCreateGatewayYaml: true
```

- service.beta.kubernetes.io/alicloud-loadbalancer-address-type：设置SLB的网络类型。可选：
  - *internet*：公网。
  - *intranet*：私网。
- service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec：设置SLB规格。可选：*slb.s1.small*、*slb.s2.small*、slb.s2.medium、*slb.s3.small*、*slb.s3.medium*、slb.s3.large。
- service.beta.kubernetes.io/alibaba-cloud-loadbalancer-ip-version：设置值为*"ipv6"*，表示该SLB采用IPv6地址。

ASM网关创建成功后，在**ASM网关**页面可以查看网关**Kubernetes服务**列下的IPv6地址。



为已有ASM网关添加IPv6地址

如果您已创建ASM网关，且该网关使用的是IPv4地址，您可以通过创建SLB的方式为已有ASM网关添加IPv6地址，然后您既可以使用IPv4地址，也可以使用IPv6地址。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 在**服务**页面顶部设置**命名空间**为istio-system，在右上角单击**使用YAML创建资源**。
6. 在**创建**页面设置**示例模板**为**自定义**，将以下内容复制到**模板**文本框中，单击**创建**。

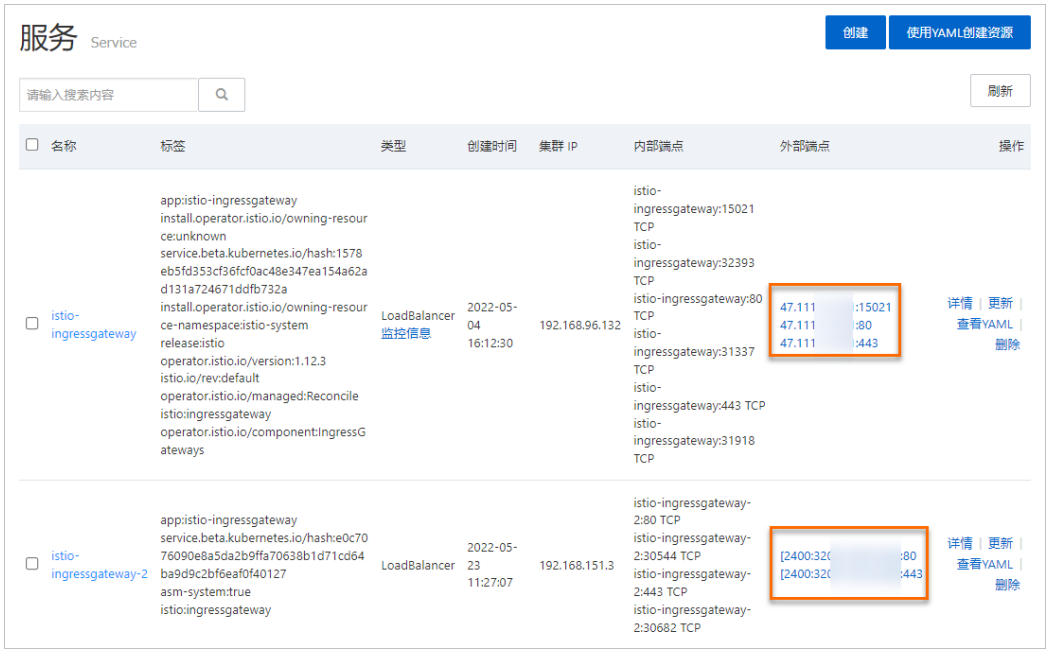
   > 注意 nodePort不能与已有端口重复。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-address-type: internet
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: slb.s1.small
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-ip-version: "ipv6"
  labels:
    app: istio-ingressgateway
    asm-system: 'true'
    istio: ingressgateway
  name: istio-ingressgateway-2
  namespace: istio-system
spec:
  externalTrafficPolicy: Cluster
  ports:
    - name: http-0
      nodePort: 30544
      port: 80
      protocol: TCP
      targetPort: 80
    - name: https-2
      nodePort: 30682
      port: 443
      protocol: TCP
      targetPort: 443
  selector:
    app: istio-ingressgateway
    asm-system: 'true'
    istio: ingressgateway
    provider: asm
  sessionAffinity: None
  type: LoadBalancer
```

   - service.beta.kubernetes.io/alicloud-loadbalancer-address-type：设置SLB的网络类型。可选：
     - *internet*：公网。
     - *intranet*：私网。

- service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec：设置SLB规格。可选：*slb.s1.small*、slb.s2.small、slb.s2.medium、slb.s3.small、*slb.s3.medium*、*slb.s3.large*。
- service.beta.kubernetes.io/alibaba-cloud-loadbalancer-ip-version：设置值为*"ipv6"*，表示该SLB采用IPv6地址。

创建服务成功后，在ACK集群的**服务**页面可以查看istio-ingressgateway和istio-ingressgateway-2**外部端点**列下的IPv4地址和IPv6地址。这两个地址即为ASM网关的IPv4地址和IPv6地址。



2.8. 使用多个SLB访问ASM网关

您可以为ASM网关绑定多个SLB，实现多个SLB可以访问一个ASM网关。本文介绍如何为ASM网关绑定多个SLB后，使用多个SLB访问ASM网关。

前提条件

- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关。具体操作，请参见添加入口网关服务。

  部署入口网关成功后，会自动在ACK集群中创建名为istio-ingressgateway的服务。
- 已在ASM中部署Bookinfo应用和Istio资源，使得可以通过入口网关访问到Bookinfo应用。具体操作，请参见部署应用到ASM实例和使用Istio资源实现版本流量路由。

操作步骤

您需要通过创建Service的方式为ASM网关创建额外的SLB。

> **说明** 如果您删除了额外绑定的Service，SLB也会被删除。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。

5. 在**服务**页面顶部设置**命名空间**为istio-system，在右上角单击**使用YAML创建资源**。
6. 在**创建**页面设置**示例模板**为**自定义**，将以下内容复制到**模板**文本框中，单击**创建**。

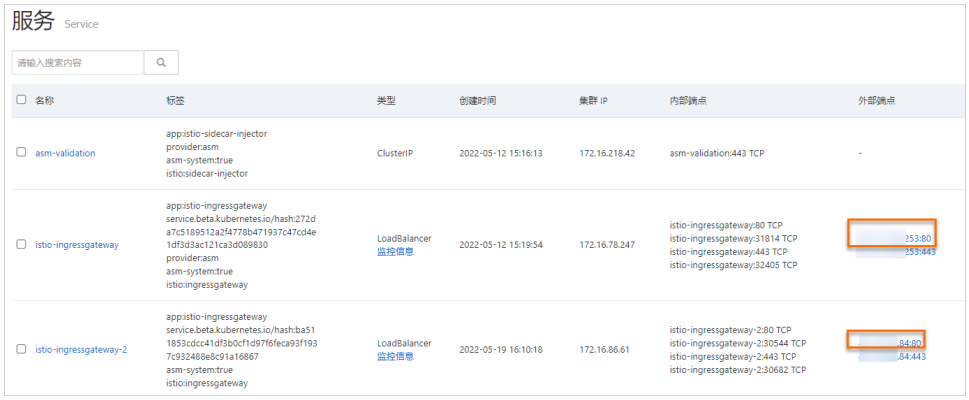
> 注意 nodePort不能与已有端口重复。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: slb.s1.small
    service.beta.kubernetes.io/alicloud-loadbalancer-address-type: internet
  labels:
    app: istio-ingressgateway
    asm-system: 'true'
    istio: ingressgateway
  name: istio-ingressgateway-2
  namespace: istio-system
spec:
  externalTrafficPolicy: Cluster
  ports:
    - name: http-0
      nodePort: 30544
      port: 80
      protocol: TCP
      targetPort: 80
    - name: https-2
      nodePort: 30682
      port: 443
      protocol: TCP
      targetPort: 443
  selector:
    app: istio-ingressgateway
    asm-system: 'true'
    istio: ingressgateway
    provider: asm
  sessionAffinity: None
  type: LoadBalancer
```

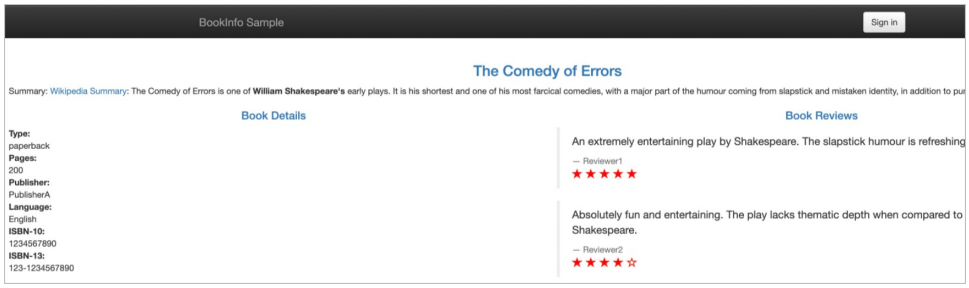
   - name：服务的名称，本文设置为*istio-ingressgateway-2*。
   - service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec：设置SLB规格。可选：*slb.s1.small*、*slb.s2.small*、*slb.s2.medium*、*slb.s3.small*、*slb.s3.medium*、*slb.s3.large*。
   - service.beta.kubernetes.io/alicloud-loadbalancer-address-type：设置SLB的网络类型。可选：
     - *internet*：公网。
     - *intranet*：私网。

   Service创建成功后，会自动创建新的SLB。
7. 使用多个SLB访问ASM网关。

i. 在ACK集群的**服务**页面获取istio-ingressgateway和istio-ingressgateway-2**外部端点**列下的80端口的IP地址。



ii. 在浏览器地址栏中分别输入*http://<istio-ingressgateway的IP地址>/productpage*和*http://<istio-ingressgateway-2的IP地址}>/productpage*。



2个地址都返回以上Bookinfo应用页面，说明使用多个SLB访问ASM网关成功。

2.9. 为ASM网关启用压缩

启用ASM网关的压缩能力后，将对HTTP请求的Response进行压缩，从而加快响应速度，降低流量消耗。本文介绍如何为ASM网关启用压缩。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。
- 如果您需要使用域名，则您的域名需要备案才能正常访问。

操作步骤

1. 在ACK集群中部署Nginx。

i. 创建名为Nginx的YAML文件。

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
        sidecarset-injected: "true"
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: nginx
  type: ClusterIP
```

ii. 执行以下命令，部署Nginx应用。

```
kubectl apply -f nginx.yaml
```

2. 在ASM创建虚拟服务和网关规则。

i. 登录ASM控制台。

ii. 在左侧导航栏，选择**服务网格 > 网格管理**。

iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

iv. 在网格详情页面左侧导航栏选择**流量管理 > 虚拟服务**，然后在右侧页面单击**使用YAML创建**。

v. 选择命名空间，将以下内容复制到文本框，单击**创建**。

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: nginx
  namespace: default
spec:
  gateways:
    - nginx-gateway
  hosts:
    - '*'
  http:
    - match:
        - uri:
            exact: /
      route:
        - destination:
            host: nginx
            port:
              number: 80
```

vi. 在网格详情页面左侧导航栏选择**流量管理 > 网关规则**，然后在右侧页面单击**使用YAML创建**。

vii. 选择命名空间，将以下内容复制到文本框，单击**创建**。

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: nginx-gateway
  namespace: default
spec:
  selector:
    istio: ingressgateway
  servers:
    - hosts:
        - '*'
      port:
        name: http
        number: 80
        protocol: HTTP
```

3. 为ASM网关启用压缩。

i. 在网格详情页面左侧导航栏单击**ASM网关**。

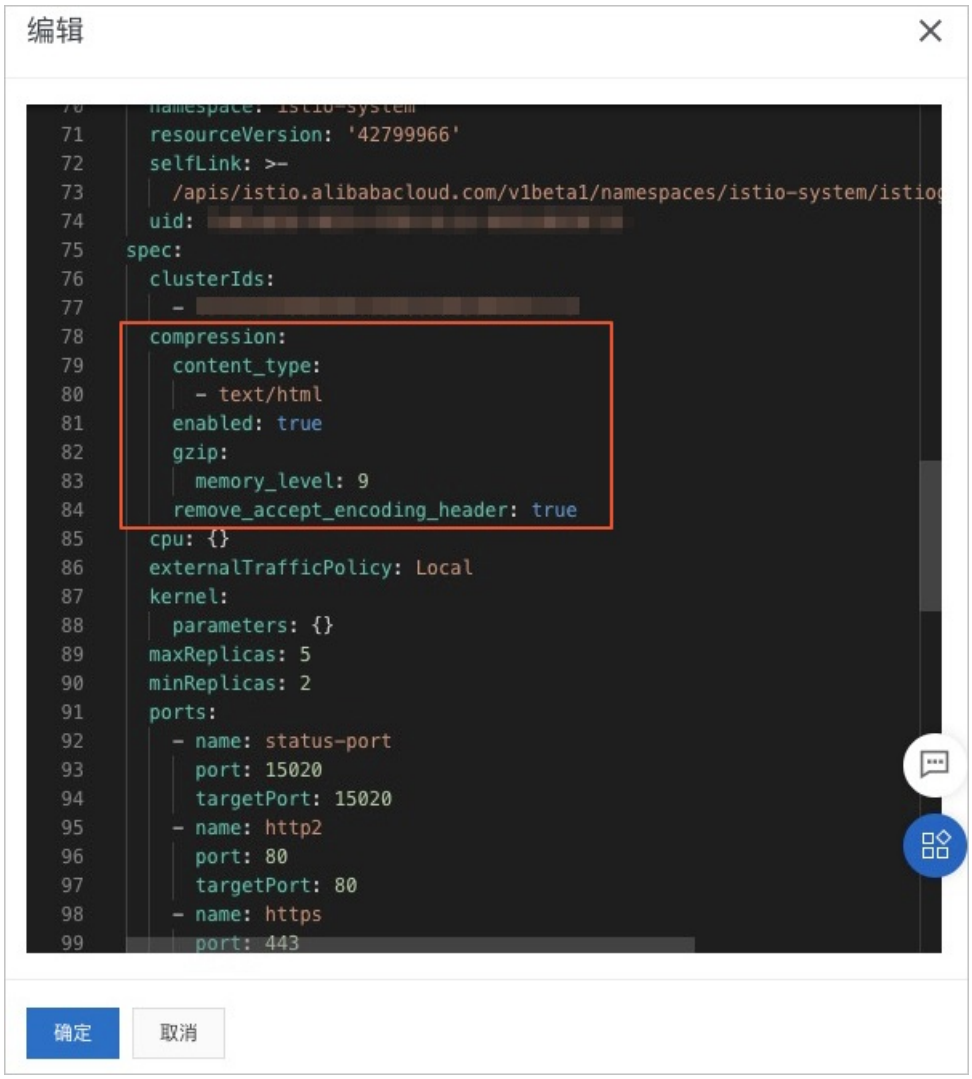
ii. 在**ASM网关**页面单击ingressgateway**操作**列的**查看YAML**。

iii. 在**编辑**面板的文本框中增加以下内容，然后单击**确定**。

> **说明** 关于压缩的更多参数说明，请参见自定义入口网关服务。

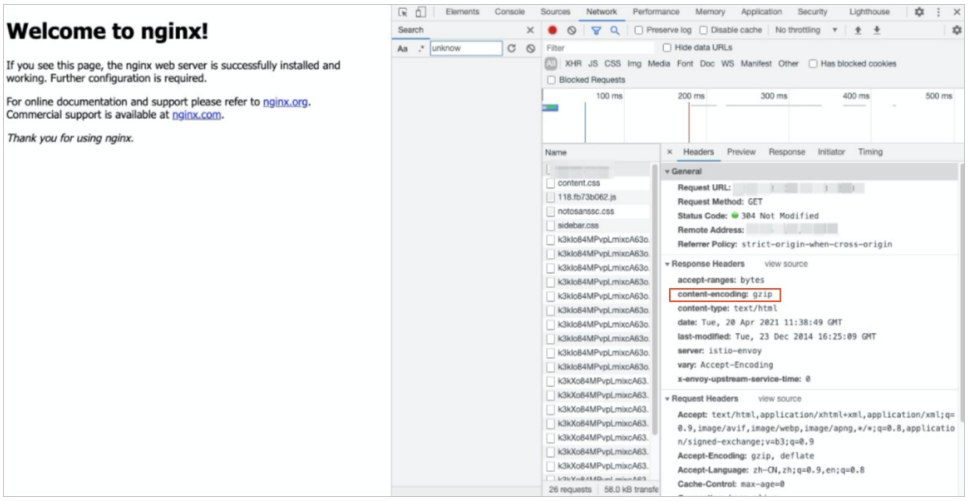
```
compression:
  content_type:
    - text/html
  enabled: true
  gzip:
    memory_level: 9
  remove_accept_encoding_header: true
```

- compression.content_type：需要被压缩的ContentType列表。
- compression.enabled：是否启用入口网关压缩能力。
- compression.gzip：允许压缩使用的内存空间规格。
- compression.remove_accept_encoding_header：
  - 设置为 `true` 时，入口网关会将客户端请求转发至上游之前移除请求内的Accept-Encoding Header。
  - 设置为 `false` 时，入口网关会将客户端请求转发至上游之前保留请求内的Accept-Encoding Header。



结果验证

1. 打开浏览器的调试页面，本文以谷歌浏览器为例。
   i. 在谷歌浏览器右上角单击图标。
   ii. 选择**更多工具 > 开发者工具**。
2. 访问Nginx应用。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
   v. 在**服务**页面顶部设置命名空间为istio_system，然后查看istio_ingressgateway**外部端点**列下的端口为80的IP地址。
   vi. 在打开的开发者工具页面中输入istio_ingressgateway的外部端点地址。
   可以看到，Response中的 `Content-Encoding` 已经显示为 `gzip` 。说明为ASM网关启用压缩成功。



2.10. 增强ASM网关高可用性

ASM网关作为业务的流量入口，为了避免服务不可用，增强ASM网关的高可用性非常重要。本文介绍如何增强ASM网关的高可用性。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK或ASK集群。具体操作，请参见创建Kubernetes托管版集群或创建Serverless Kubernetes集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。

在ACK集群中增强ASM网关高可用性

在ACK集群中，您可以在创建ASM网关时通过YAML配置Pod反亲和性策略，实现ASM网关的Pod分布到不同的Node节点或者可用区，从而增强ASM网关的高可用性。

- 在ASM网关中配置podAntiAffinity参数，使ASM网关的Pod分布到不同的Node节点。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
```

```
kind: IstioGateway
metadata:
  name: ingressgateway-1
  namespace: istio-system
spec:
  clusterIds:
    - "c954ee9df88f64f229591f0ea4c61****"
  cpu:
    targetAverageUtilization: 80
  externalTrafficPolicy: Local
  maxReplicas: 4
  minReplicas: 2
  ports:
  - name: status-port
    port: 15020
    targetPort: 15020
  - name: http2
    port: 80
    targetPort: 80
  - name: https
    port: 443
    targetPort: 80
  - name: tls
    port: 15443
    targetPort: 15443
  replicaCount: 1
  resources:
    limits:
      cpu: '2'
      memory: 2G
    requests:
      cpu: 200m
      memory: 256Mi
  sds:
    enabled: true
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 2000m
        memory: 1024Mi
  serviceType: LoadBalancer
  affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - istio-ingressgateway-1
              topologyKey: kubernetes.io/hostname
            weight: 100
```

```
  rollingMaxSurge: "100%"
  rollingMaxUnavailable: "25%"
```

- preferredDuringSchedulingIgnoredDuringExecution：表示Pod反亲和性为软亲和性。在调度Pod到节点的时候，如果没有满足所设置要求，也可以继续调度Pod。
- matchExpressions：设置key为*app*，operator为*In*，values为*istio-ingressgateway-1*，表示Pod不能和包含 `app=istio-ingressgateway-1` 标签的Pod部署在一个节点上，即一个节点只能部署一个包含 `app=istio-ingressgateway-1` 标签的Pod。
- topologyKey：设置Pod反亲和性生效的维度。

  本例设置 `kubernetes.io/hostname` ，表示在节点拓扑域中生效。

- 在ASM网关中配置podAntiAffinity参数，使ASM网关的Pod分布到不同的可用区。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
metadata:
  name: ingressgateway-1
  namespace: istio-system
spec:
  clusterIds:
    - "c954ee9df88f64f229591f0ea4c61****"
  cpu:
    targetAverageUtilization: 80
  externalTrafficPolicy: Local
  maxReplicas: 4
  minReplicas: 2
  ports:
  - name: status-port
    port: 15020
    targetPort: 15020
  - name: http2
    port: 80
    targetPort: 80
  - name: https
    port: 443
    targetPort: 80
  - name: tls
    port: 15443
    targetPort: 15443
  replicaCount: 1
  resources:
    limits:
      cpu: '2'
      memory: 2G
    requests:
      cpu: 200m
      memory: 256Mi
  sds:
    enabled: true
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
```

```
      limits:
        cpu: 2000m
        memory: 1024Mi
  serviceType: LoadBalancer
  affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - istio-ingressgateway-1
              topologyKey: topology.kubernetes.io/zone
            weight: 100
  rollingMaxSurge: "100%"
  rollingMaxUnavailable: "25%"
```

- preferredDuringSchedulingIgnoredDuringExecution：表示Pod反亲和性为软亲和性。在调度Pod到节点的时候，如果没有满足所设置要求，也可以继续调度Pod。
- matchExpressions：设置key为*app*，operator为*In*，values为*istio-ingressgateway-1*，表示Pod不能和包含 `app=istio-ingressgateway-1` 标签的Pod部署在一个可用区上，即一个可用区只能部署一个包含 `app=istio-ingressgateway-1` 标签的Pod。
- topologyKey：设置Pod反亲和性生效的维度。

  本例设置 `topology.kubernetes.io/zone` ，表示在可用区拓扑域中生效。

在ASK集群中增强ASM网关高可用性

ASK集群不支持Pod反亲和调度策略，但是您可以在ASK集群下创建ECI Pod ，使之分布在不同的可用区，从而增强ASM网关高可用性。

1. 在ASK集群中配置多个可用区。具体操作，请参见创建多可用区的ECI Pod。
2. 在ASM网关中使用 `pod annotation` 关联可用区。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
metadata:
  name: ingressgateway
  namespace: istio-system
spec:
  clusterIds:
    - "c954ee9df88f64f229591f0ea4c61****"
  cpu:
    targetAverageUtilization: 80
  externalTrafficPolicy: Local
  maxReplicas: 4
  minReplicas: 2
  ports:
  - name: status-port
    port: 15020
    targetPort: 15020
  - name: http2
    port: 80
    targetPort: 80
  - name: https
    port: 443
    targetPort: 80
  - name: tls
    port: 15443
    targetPort: 15443
  replicaCount: 1
  resources:
    limits:
      cpu: '2'
      memory: 2G
    requests:
      cpu: 200m
      memory: 256Mi
  sds:
    enabled: true
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 2000m
        memory: 1024Mi
  serviceType: LoadBalancer
  podAnnotations:
    k8s.aliyun.com/eci-vswitch: "vsw-bp1b07j0miob3khtn****,vsw-bp12b85hh323se8ft****"
    k8s.aliyun.com/eci-schedule-strategy: "VSwitchRandom"
  rollingMaxSurge: "100%"
  rollingMaxUnavailable: "25%"
```

- k8s.aliyun.com/eci-vswitch：对应VPC下的不同的交换机ID，用于关联可用区。
- k8s.aliyun.com/eci-schedule-strategy：设置ECI调度策略。本文中必须使用 `VSwitchRandom` 策略，即使用随机的方式将ECI Pod调度到多个可用区。

2.11. 自定义ASM网关日志格式

基于业务的需求，需要对ASM网关的日志格式进行定制，使ASM日志包含特定的Header。本文介绍如何使用EnvoyFilter自定义ASM网关日志格式。

创建EnvoyFilter

本文将请求报文的 `header user_id` 添加到ASM网关日志为例。

使用以下内容，创建EnvoyFilter，具体操作，请参见管理Envoy过滤器。

> **说明** 如果您使用的ASM实例版本高于1.8，建议在EnvoyFilter中typed_config参数中使用envoy v3的API接口。

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: gateway-accesslog-userdefine
  namespace: istio-system
spec:
  configPatches:
    - applyTo: NETWORK_FILTER
      match:
        context: ANY
        listener:
          filterChain:
            filter:
              name: envoy.http_connection_manager
      patch:
        operation: MERGE
        value:
          typed_config:
            '@type': >-
              type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.
v3.HttpConnectionManager
            access_log:
              - name: envoy.access_loggers.file
                typed_config:
                  '@type': >-
                    type.googleapis.com/envoy.extensions.access_loggers.file.v3.FileAccessL
og
                  log_format:
                    json_format:
                      authority: '%REQ(:AUTHORITY)%'
                      bytes_received: '%BYTES_RECEIVED%'
                      bytes_sent: '%BYTES_SENT%'
                      downstream_local_address: '%DOWNSTREAM_LOCAL_ADDRESS%'
                      downstream_remote_address: '%DOWNSTREAM_REMOTE_ADDRESS%'
                      duration: '%DURATION%'
                      method: '%REQ(:METHOD)%'
                      path: '%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%'
                      protocol: '%PROTOCOL%'
                      request_id: '%REQ(X-REQUEST-ID)%'
```

```
                    requested_server_name: '%REQUESTED_SERVER_NAME%'
                    response_code: '%RESPONSE_CODE%'
                    response_flags: '%RESPONSE_FLAGS%'
                    route_name: '%ROUTE_NAME%'
                    start_time: '%START_TIME%'
                    upstream_cluster: '%UPSTREAM_CLUSTER%'
                    upstream_host: '%UPSTREAM_HOST%'
                    upstream_local_address: '%UPSTREAM_LOCAL_ADDRESS%'
                    upstream_service_time: '%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)%'
                    upstream_transport_failure_reason: '%UPSTREAM_TRANSPORT_FAILURE_REASO
N%'
                    user_agent: '%REQ(USER-AGENT)%'
                    user_id: '%REQ(USER_ID)%'
                    x_forwarded_for: '%REQ(X-FORWARDED-FOR)%'
                path: /dev/stdout
  workloadSelector:
    labels:
      app: istio-ingressgateway
```

access_log format部分可以根据自身业务需求自定义修改。其中：

> **说明** log_format若不指定，将使用默认的Envoy accessLog format。

- namespace：指定EnvoyFilter生效的命名空间。
- workloadSelector：指定EnvoyFilter生效的网关实例。`app: istio-ingressgateway` 为Gateway名称，需要根据实际情况进行修改，命名规则是app: istio-{Gateway名称}。

如果您想在日志中添加其他Header，您可以按照以下格式添加到EnvoyFilter中。

```
my_custom_header:'%REQ(MY_CUSTOM_HEADER)%'
```

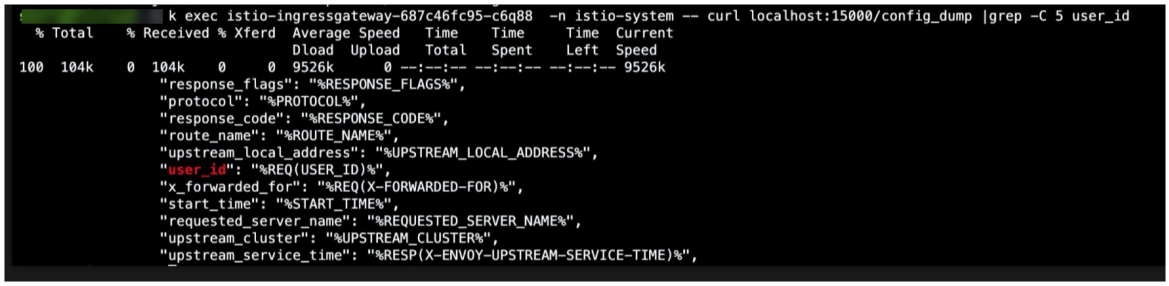
对应ASM网关日志会包含以下内容。

```
{"protocol": "HTTP/1.1", "duration": "123", "my_custom_header": "value_of_MY_CUSTOM_HEADER"
}
```

验证EnvoyFilter配置是否生效

1. 步骤二：选择集群凭证类型。
2. 执行以下命令，查看日志中增加的 `user_id` 是否生效。

```
kubectl  exec $gateway_pod -n istio-system -- curl localhost:15000/config_dump |grep -5
user_id
```



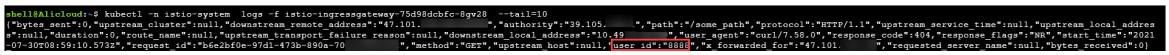
可以看到 `user_id` 已经添加到accessLogFormat中。

3. 执行以下命令，向网关发送请求。本文以 `user_id: 8888` 为例。

```
curl -H 'user_id: 8888'  <Gateway IP> /some_path
```

4. 执行以下命令，查看Gateway日志。

```
kubectl -n istio-system  logs -f <Gateway Pod名称>  --tail=10
```



可以看到，Gateway日志包含 `user_id` 信息。

版本说明

如果您使用的ASM实例版本低于1.8，建议在EnvoyFilter中typed_config参数中使用envoy v2的API接口。以下为EnvoyFilter示例。

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: enable-accesslog
  namespace: gateway-accesslog-userdefine
spec:
  configPatches:
  - applyTo: NETWORK_FILTER
    match:
      context: ANY
      listener:
        filterChain:
          filter:
            name: envoy.http_connection_manager
    patch:
      operation: MERGE
      value:
        typed_config:
          "@type": "type.googleapis.com/envoy.config.filter.network.http_connection_manager
.v2.HttpConnectionManager"
          access_log:
          - name: envoy.file_access_log
            config:
              path: /dev/stdout
              log_format:
                 ....
  workloadSelector:
    labels:
      app: istio-ingressgateway
```

2.12. 为域名添加证书

ASM支持以图形化的方式为域名添加证书，便于您使用HTTPS等协议访问域名，提升服务网关的安全性。本文介绍如何为域名添加新建证书和已有证书。

前提条件

- 已创建ASM实例，且版本为专业版、企业版或旗舰版。具体操作，请参见创建ASM实例。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。
- 已部署应用到ASM实例的集群中。具体操作，请参见部署应用到ASM实例。
- 为命名空间注入Sidecar。具体操作，请参见多种方式灵活开启自动注入。
- 获取ASM网关地址。具体操作，请参见访问入口网关。

背景信息

本文以域名为aliyun.com的myexampleapp服务为例，为aliyun.com域名添加新建证书或已有证书，添加成功后，您就可以通过ASM网关使用HTTPS协议访问域名为aliyun.com的myexampleapp服务。

为域名添加新建证书

1. 创建示例服务myexampleapp。

   i. 使用以下内容，创建*myexample-nginx.conf*。

   本文示例服务是基于Nginx实现的，您需要为Nginx服务器创建配置文件，以域名aliyun.com的服务为例，以下内容定义请求根路径直接返回字样 `Welcome to aliyun.com!` 及状态码 `200` 。

```
events {
}
http {
  log_format main '$remote_addr - $remote_user [$time_local]  $status '
  '"$request" $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log /var/log/nginx/access.log main;
  error_log  /var/log/nginx/error.log;
  server {
    listen 80;
    location / {
        return 200 'Welcome to aliyun.com!';
        add_header Content-Type text/plain;
    }
  }
}
```

   ii. 执行以下命令，创建Nginx服务器的配置项。

```
kubectl create configmap myexample-nginx-configmap --from-file=nginx.conf=./myexamp
le-nginx.conf
```

   iii. 使用以下内容，创建*myexampleapp.yaml*。

```
apiVersion: v1
kind: Service
metadata:
  name: myexampleapp
  labels:
    app: myexampleapp
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: myexampleapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myexampleapp
spec:
  selector:
    matchLabels:
      app: myexampleapp
  replicas: 1
  template:
    metadata:
      labels:
        app: myexampleapp
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx
          readOnly: true
      volumes:
      - name: nginx-config
        configMap:
          name: myexample-nginx-configmap
```

iv. 执行以下命令，创建域名为aliyun.com的内部服务。

```
kubectl apply -f myexampleapp.yaml
```

2. 在ASM网关中导入myexampleapp服务。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
   iv. 在网格详情页面左侧导航栏单击**ASM网关**，在右侧页面单击目标网关的名称。
   v. 在网关详情页面左侧导航栏单击**上游服务**。

vi. 在**上游服务**页面单击**导入服务**。

vii. 在**导入服务**页面选择**命名空间**，选中myexampleapp服务，单击图标，然后单击**确认**。

3. 创建证书和私钥。

i. 在openssl中执行以下命令，创建根证书和私钥。

```
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -subj '/O=myexample Inc
./CN=aliyun.com' -keyout aliyun.root.key -out aliyun.root.crt
```

ii. 执行以下命令，为aliyun.com服务器生成证书和私钥。

- 执行以下命令，创建证书*aliyun.com.crt*。

```
openssl x509 -req -days 365 -CA aliyun.root.crt -CAkey aliyun.root.key -set_seria
l 0 -in aliyun.com.csr -out aliyun.com.crt
```

- 执行以下命令，创建私钥*aliyun.com.key*。

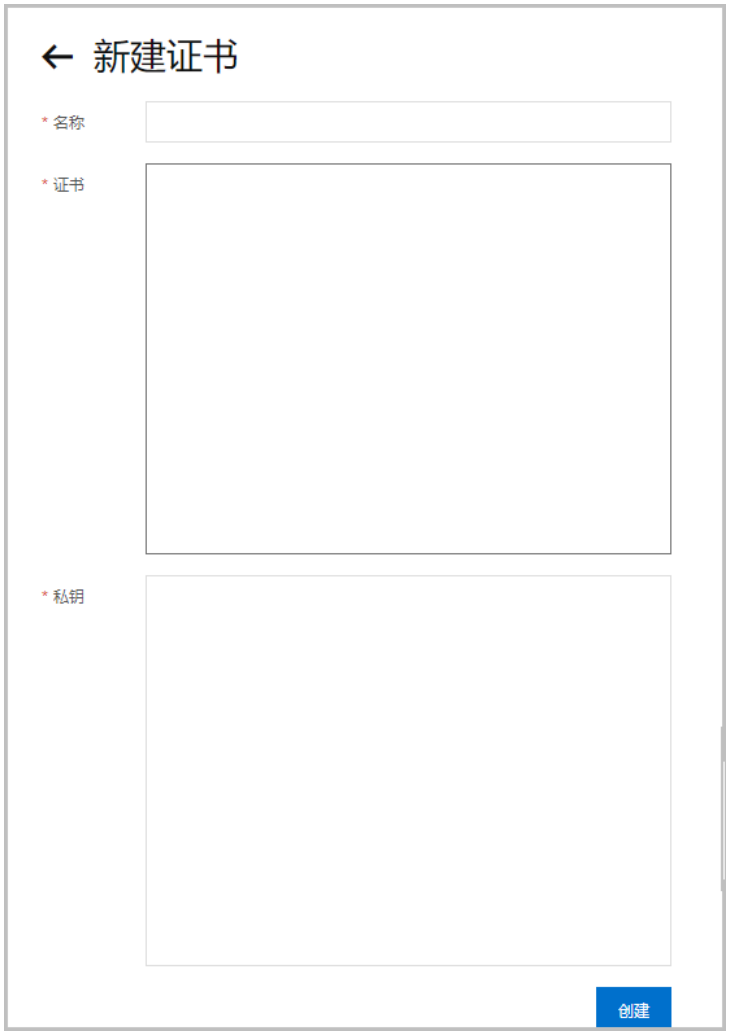
```
openssl req -out aliyun.com.csr -newkey rsa:2048 -nodes -keyout aliyun.com.key -s
ubj "/CN=aliyun.com/O=myexample organization"
```

4. 添加证书和私钥挂载卷到ASM网关。

i. 在网关详情页面左侧导航栏单击**域名/证书**。

ii. 单击**证书**页签，单击**创建**。

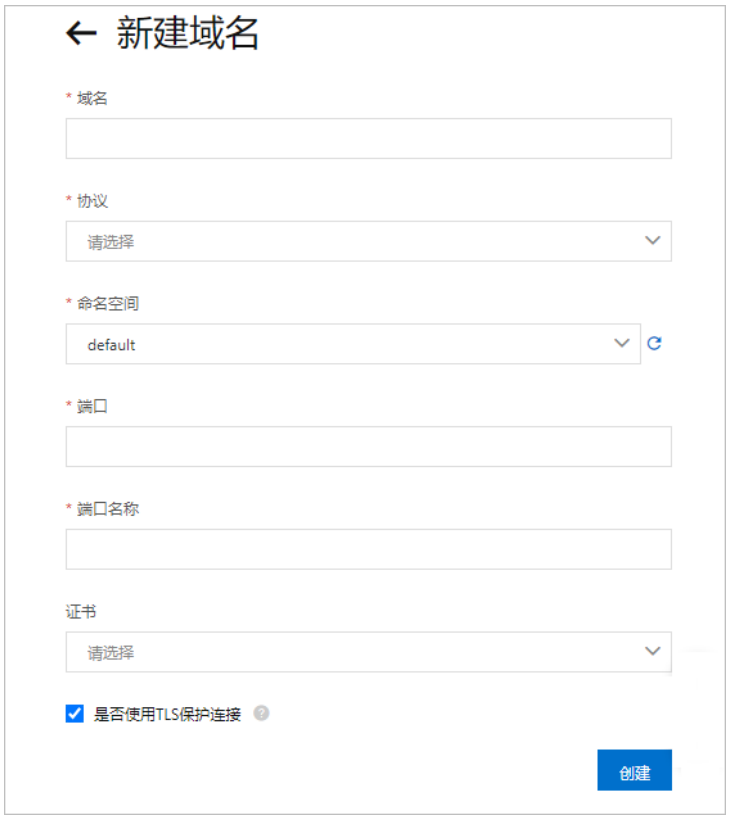
iii. 在**新建证书**页面输入**名称**，将*aliyun.com.crt*的内容复制到**证书**，*aliyun.com.key*的内容复制到**私钥**，单击**创建**。



5. 绑定域名与证书。
   i. 在网关详情页面左侧导航栏单击**域名/证书**。
   ii. 在**域名**页签下单击**创建**。

iii. 在**新建域名**页面设置**域名**为*.aliyun.com，**协议**为**HTTPS**，选择**命名空间**、**证书**，输入**端口**和**端口名称**，选中**是否使用TLS保护连接**，单击**创建**。

选中**是否使用TLS保护连接**后表示只有TLS请求才能访问到域名。



6. 验证绑定域名与证书是否成功。

执行以下命令，使用HTTPS协议访问aliyun.com。

```
curl -k -H Host:www.aliyun.com --resolve www.aliyun.com:443:<ASM网关地址>  https://www.aliyun.com
```

预期输出：

```
Welcome to aliyun.com!
```

可以看到，使用HTTPS协议访问aliyun.com成功，说明绑定域名与证书成功。

为域名添加已有证书

1. 创建示例服务myexampleapp。

i. 使用以下内容，创建*myexample-nginx.conf*。

本文示例服务是基于Nginx实现的，您需要为Nginx服务器创建配置文件，以域名aliyun.com的服务为例，以下内容定义请求根路径直接返回字样 `Welcome to aliyun.com!` 及状态码 `200` 。

```
events {
}
http {
  log_format main '$remote_addr - $remote_user [$time_local]  $status '
  '"$request" $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log /var/log/nginx/access.log main;
  error_log  /var/log/nginx/error.log;
  server {
    listen 80;
    location / {
        return 200 'Welcome to aliyun.com!';
        add_header Content-Type text/plain;
    }
  }
}
```

ii. 执行以下命令，创建Nginx服务器的配置项。

```
kubectl create configmap myexample-nginx-configmap --from-file=nginx.conf=./myexamp
le-nginx.conf
```

iii. 使用以下内容，创建*myexampleapp.yaml*。

```
apiVersion: v1
kind: Service
metadata:
  name: myexampleapp
  labels:
    app: myexampleapp
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: myexampleapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myexampleapp
spec:
  selector:
    matchLabels:
      app: myexampleapp
  replicas: 1
  template:
    metadata:
      labels:
        app: myexampleapp
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx
          readOnly: true
      volumes:
      - name: nginx-config
        configMap:
          name: myexample-nginx-configmap
```

iv. 执行以下命令，创建域名为aliyun.com的内部服务。

```
kubectl apply -f myexampleapp.yaml
```

2. 在ASM网关中导入myexampleapp服务。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
   iv. 在网格详情页面左侧导航栏单击**ASM网关**，在右侧页面单击目标网关的名称。
   v. 在网关详情页面左侧导航栏单击**上游服务**。

vi. 在**上游服务**页面单击**导入服务**。

vii. 在**导入服务**页面选择**命名空间**，选中myexampleapp服务，单击图标，然后单击**确认**。

3. 导入证书至ASM网关。

在已有证书中添加 `istioGateway:<ASM网关名称>` 和 `provider:asm` 标签。添加标签成功后，证书将会自动出现在ASM控制台的**证书**页面。

```
kind: Secret
metadata:
  creationTimestamp: "2022-04-01T07:44:50Z"
  labels:
    istioGateway: ingressgateway
    provider: asm
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:data:
```

4. 绑定域名与证书。

i. 登录ASM控制台。

ii. 在左侧导航栏，选择**服务网格 > 网格管理**。

iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

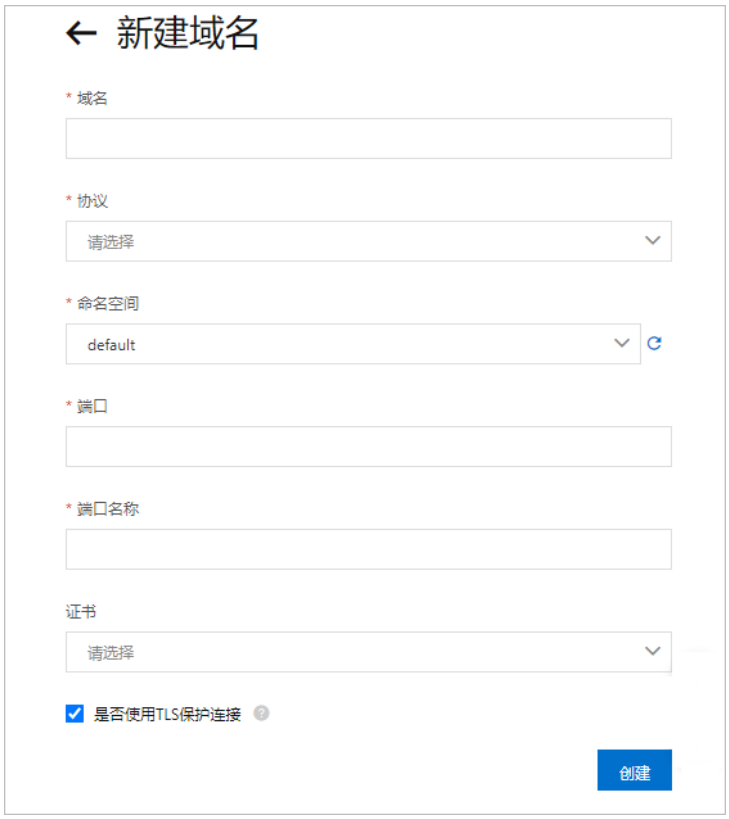
iv. 在网格详情页面左侧导航栏单击**ASM网关**，在右侧页面单击目标网关的名称。

v. 在网关详情页面左侧导航栏单击**域名/证书**。

vi. 在**域名**页签下单击**创建**。

vii. 在**新建域名**页面设置**域名**为*.aliyun.com，**协议**为**HTTPS**，选择**命名空间**、**证书**，输入**端口**和**端口名称**，选中**是否使用TLS保护连接**，单击**创建**。

选中**是否使用TLS保护连接**后表示只有TLS请求才能访问到域名。



5. 验证绑定域名与证书是否成功。

执行以下命令，使用HTTPS协议访问aliyun.com。

```
curl -k -H Host:www.aliyun.com --resolve www.aliyun.com:443:<ASM网关地址>  https://www.aliyun.com
```

预期输出：

```
Welcome to aliyun.com!
```

可以看到，使用HTTPS协议访问aliyun.com成功，说明绑定域名与证书成功。

2.13. 使用cert-manager管理网关的证书

cert-manager是一个证书生命周期管理系统，支持证书的申请、部署等功能。您可以使用cert-manager颁发ASM网关的证书，从而可以使用HTTPS协议通过ASM网关访问服务，保证数据传输的安全。本文介绍如何使用cert-manager管理网关的证书。

背景信息

cert-manager支持签发自签名证书和DNS域名证书，从而可以使用HTTPS协议通过ASM网关访问服务。这两种证书的区别如下：

- 自签名证书：自签名证书仅具有加密功能，无身份验证功能。您可以在命令行工具中使用HTTPS协议访问ASM网关，但是自签名证书不受Web浏览器的信任，Web浏览器检查HTTPS连接会标记为潜在风险并弹出

错误消息，即无法在Web浏览器中使用HTTPS协议通过ASM网关访问服务。

- DNS域名证书：DNS域名证书由受信任的CA机构颁发，兼具加密和身份验证功能。相比自签名证书，DNS域名证书具有更高的安全性，受到Web浏览器的信任。您可以同时在命令行工具和Web浏览器中使用HTTPS协议通过ASM网关访问服务。

在集群中安装cert-manager

1. 在本地安装Helm。具体操作，请参见Helm。
2. 使用kubectl连接集群。具体操作，请参见通过kubectl工具连接集群。
3. 执行以下命令，创建cert-manager命名空间。

```
kubectl create namespace cert-manager
```

4. 执行以下命令，添加cert-manager Chart。

```
helm repo add jetstack https://charts.jetstack.io
```

5. 执行以下命令，获取cert-manager Chart的最新信息。

```
helm repo update
```

6. 执行以下命令，安装cert-manager。

> 注意 cert-manager的版本需要和Kubernetes版本保持兼容。关于cert-manager和Kubernetes版本的对应关系，请参见Supported Releases。

```
helm install \
  cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --version v1.1.0 \
  --set installCRDs=true
```

场景一：使用cert-manager签发自签名证书

步骤一：在集群中生成自签证书

1. 使用以下内容，创建*issuer.yaml*。

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: istio-ingressgateway-certs
spec:
  isCA: true
  duration: 2160h # 90d
  secretName: istio-ingressgateway-certs
  commonName: istio-ingressgateway-certs
  subject:
    organizations:
    - cluster.local
    - cert-manager
  issuerRef:
    name: selfsigned
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: istio-ingressgateway-certs
spec:
  ca:
    secretName: istio-ingressgateway-certs
```

2. 执行以下命令，创建自签名CA来颁发工作负载证书。

```
kubectl apply -f issuer.yaml -n istio-system
```

3. 执行以下命令，查看证书。

```
kubectl get secret -n istio-system
```

预期输出：

```
NAME                                  TYPE                                  DATA
AGE
istio-ingressgateway-certs            kubernetes.io/tls                     3
68m
```

步骤二：添加证书和私钥挂载卷到入口网关

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**。

5. 在**ASM网关**页面单击目标入口网关**操作**列下的**YAML**。
6. 在入口网关服务对应的YAML定义文件中，添加如下信息到 `spec` ：

```
  secretVolumes:
    - mountPath: /etc/istio/ingressgateway-certs
      name: ingressgateway-certs
      secretName: istio-ingressgateway-certs
```

添加之后的YAML文件内容类似如下：

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
metadata:
  name: ingressgateway
  namespace: istio-system
spec:
  clusterIds:
    - c58638b491a5248669b59609e0a17****
  cpu: {}
  externalTrafficPolicy: Local
  maxReplicas: 1
  minReplicas: 1
  ports:
    - name: status-port
      port: 15020
      targetPort: 15020
    - name: http2
      nodePort: 31380
      port: 80
      targetPort: 80
    - name: https
      nodePort: 31390
      port: 443
      targetPort: 443
    - name: tls
      port: 15443
      targetPort: 15443
  replicaCount: 1
  secretVolumes:
    - mountPath: /etc/istio/ingressgateway-certs
      name: ingressgateway-certs
      secretName: istio-ingressgateway-certs
  serviceAnnotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: slb.s1.small
  serviceType: LoadBalancer
```

步骤三：在集群中检查入口网关配置

添加证书和私钥挂载卷到入口网关后，入口网关会加载证书与密钥。

1. 执行以下命令，验证tls.crt与tls.key已被挂载到入口网关Pod中。

```
kubectl exec -it -n istio-system $(kubectl -n istio-system get pods -l istio=ingressgat
eway -o jsonpath='{.items[0].metadata.name}') -- ls -al /etc/istio/ingressgateway-certs
```

预期输出：

```
lrwxrwxrwx 1 root root   14 May  9 01:14 tls.crt -> ..data/tls.crt
lrwxrwxrwx 1 root root   14 May  9 01:14 tls.key -> ..data/tls.key
```

2. 执行以下命令，检查Ingress gateway证书中的 `Subject` 字段的正确性。

```
kubectl exec -it -n istio-system $(kubectl -n istio-system get pods -l istio=ingressgat
eway -o jsonpath='{.items[0].metadata.name}')  -- cat /etc/istio/ingressgateway-certs/t
ls.crt | openssl x509 -text -noout | grep 'Subject:'
```

预期输出：

```
Subject: O = cert-manager + O = cluster.local, CN = istio-ingressgateway-certs
```

3. 执行以下命令，检查Ingress gateway的代理能够正确访问证书。

```
kubectl exec -it -n istio-system $(kubectl -n istio-system get pods -l istio=ingressgat
eway -o jsonpath='{.items[0].metadata.name}') -- curl  127.0.0.1:15000/certs
```

预期输出：

```
{
 "certificates": [
  {
   "ca_cert": [
    {
     "path": "\u003cinline\u003e",
     "serial_number": "4edcf3",
     "subject_alt_names": [],
     "days_until_expiration": "7251",
     "valid_from": "2022-01-06T06:17:00Z",
     "expiration_time": "2042-01-01T06:22:03Z"
    }
   ],
...........
```

步骤四：验证使用HTTPS协议访问服务是否成功

1. 在集群中创建服务。

i. 使用以下内容，创建*myexample-nginx.conf*。

本文示例服务是基于Nginx实现的，您需要为Nginx服务器创建配置文件，以域名aliyun.com的服务为例，以下内容定义请求根路径直接返回字样 `Welcome to aliyun.com!` 及状态码 `200` 。

```
events {
}
http {
  log_format main '$remote_addr - $remote_user [$time_local]  $status '
  '"$request" $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log /var/log/nginx/access.log main;
  error_log  /var/log/nginx/error.log;
  server {
    listen 80;
    location / {
        return 200 'Welcome to aliyun.com!';
        add_header Content-Type text/plain;
    }
  }
}
```

ii. 执行以下命令，创建Nginx服务器的配置项。

```
kubectl create configmap myexample-nginx-configmap --from-file=nginx.conf=./myexamp
le-nginx.conf
```

iii. 执行以下命令，为命名空间default启用Sidecar自动注入。

```
kubectl label namespace default istio-injection=enabled
```

iv. 使用以下内容，创建*myexampleapp.yaml*。

```
apiVersion: v1
kind: Service
metadata:
  name: myexampleapp
  labels:
    app: myexampleapp
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: myexampleapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myexampleapp
spec:
  selector:
    matchLabels:
      app: myexampleapp
  replicas: 1
  template:
    metadata:
      labels:
        app: myexampleapp
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx
          readOnly: true
      volumes:
      - name: nginx-config
        configMap:
          name: myexample-nginx-configmap
```

v. 执行以下命令，创建域名为aliyun.com的内部服务。

```
kubectl apply -f myexampleapp.yaml
```

2. 通过kubectl连接ASM。具体操作，请参见通过kubectl连接ASM实例。
3. 在ASM中创建网关规则。

i. 使用以下内容，创建*istio-myexample-customingressgateway.yaml*。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-myexample-customingressgateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - '*.aliyun.com'
    port:
      name: http
      number: 80
      protocol: HTTP
    tls:
      httpsRedirect: true
  - hosts:
    - '*.aliyun.com'
    port:
      name: https
      number: 443
      protocol: HTTPS
    tls:
      mode: SIMPLE
      privateKey: /etc/istio/ingressgateway-certs/tls.key
      serverCertificate: /etc/istio/ingressgateway-certs/tls.crt
```

ii. 执行以下命令，创建网关规则。

```
kubectl apply -f istio-myexample-customingressgateway.yaml
```

4. 在ASM中创建虚拟服务。

i. 使用以下内容，创建*istio-myexample-customvirtualservice.yaml*。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: istio-myexample-customvirtualservice
spec:
  hosts:
  - "www.aliyun.com"
  gateways:
  - istio-myexample-customingressgateway
  http:
  - route:
    - destination:
        host: myexampleapp.default.svc.cluster.local
        port:
          number: 80
```

ii. 执行以下命令，创建虚拟服务。

```
kubectl apply -f istio-myexample-customvirtualservice.yaml
```

5. 执行以下命令，在集群中获取入口网关服务的地址。

```
kubectl get svc -n istio-system -l istio=ingressgateway
```

6. 执行以下命令，通过HTTPS协议访问aliyun.com服务。

```
curl -k -H Host:www.aliyun.com --resolve www.aliyun.com:443:{替换成真实的入口网关IP地址}
https://www.aliyun.com
```

预期输出：

```
Welcome to aliyun.com!
```

场景二：使用cert-manager签发DNS域名证书

操作前准备：

- 已有阿里云域名，且该域名已备案。具体操作，请参见域名注册。
- 已在集群中配置Webhook，用于添加DNS记录。

  您可以使用alidns-webhook配置Webhook。本文通过在集群中执行以下命令，配置Webhook。

```
https://gitee.com/godu/helminit/raw/master/cert-manager/alidns-cm-webhook.yaml
```

步骤一：为Webhook创建Secret

1. 创建RAM用户。
   i. 使用阿里云账号登录RAM控制台。
   ii. 在访问控制控制台左侧导航栏，选择**身份管理 > 用户**。
   iii. 在**用户**页面，单击**创建用户**。
   iv. 在**创建用户**页面输入**登录名称**和**显示名称**，选中**OpenAPI调用访问**，然后单击**确定**。

   记录RAM用户的AccessKey ID和AccessKey Secret。
2. 授予RAM用户AliyunDNSFullAccess策略。
   i. 在**用户**页面单击上文创建的RAM用户右侧**操作**列下的**添加权限**。
   ii. 在**添加权限**面板**系统策略**下输入AliyunDNSFullAccess，单击AliyunDNSFullAccess，单击**确定**。
3. 授予RAM用户自定义策略。
   i. 在访问控制控制台左侧导航栏，选择**权限管理 > 权限策略**。
   ii. 在**权限策略**页面单击**创建权限策略**。

iii. 在**创建权限策略**页面单击**脚本编辑**页签，然后输入以下内容，单击**下一步**。

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": "*",
            "Resource": "acs:alidns:*:*:domain/#domain-name",
            "Effect": "Allow"
        },
        {
            "Action": [
                "alidns:DescribeSiteMonitorIspInfos",
                "alidns:DescribeSiteMonitorIspCityInfos",
                "alidns:DescribeSupportLines",
                "alidns:DescribeDomains",
                "alidns:DescribeDomainNs",
                "alidns:DescribeDomainGroups"
            ],
            "Resource": "acs:alidns:*:*:*",
            "Effect": "Allow"
        }
    ]
}
```

`#domain-name` 需替换为您自己的域名，例如.example.com。

iv. 输入权限策略的名称，单击**确定**。

4. 在Base64执行以下命令，为步骤1中的AccessKey ID和AccessKey Secret进行Base64编码。

```
echo -n <AccessKey ID> | base64
```

```
echo -n <AccessKey Secret>  | base64
```

5. 在集群中创建Secret。

i. 使用以下内容，创建*alidns-secret.yaml*。

```
apiVersion: v1
kind: Secret
metadata:
  name: alidns-secret
  namespace: cert-manager
data:
  access-key: YOUR_ACCESS_KEY_BASE64  #base64编码后的AccessKey ID。
  secret-key: YOUR_SECRET_KEY_BASE64  #base64编码后的AccessKey Secret。
```

ii. 创建Secret。

```
kubectl apply -f alidns-secret.yaml
```

步骤二：在集群中创建ClusterIssuer和Certificate

1. 创建ClusterIssuer。

i. 使用以下内容，创建*letsencrypt-staging.yaml*。

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: letsencrypt-staging-account-key
    solvers:
    - dns01:
        webhook:
          groupName: acme.yourcompany.com
          solverName: alidns
          config:
            region: ""
            accessKeySecretRef:
              name: alidns-secret
              key: access-key
            secretKeySecretRef:
              name: alidns-secret
              key: secret-key
```

ii. 执行以下命令，创建ClusterIssuer。

```
kubectl apply -f letsencrypt-staging.yaml
```

2. 创建Certificate。

i. 使用以下内容，创建*example-tls.yaml*。

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-tls
  namespace: istio-system
spec:
  # The secretName will store certificate content
  secretName: istio-ingressgateway-certs
  commonName: istio-ingressgateway-certs
  dnsNames:
  # Replace to your real DNS name
  - example.com
  - "*.example.com"
  issuerRef:
    name: letsencrypt-staging
    kind: ClusterIssuer
```

ii. 执行以下命令，创建Certificate。

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

创建完成ClusterIssuer和Certificate后，cert-manager签发证书和私钥到istio-ingressgateway-certs Secret。

步骤三：添加证书和私钥挂载卷到入口网关

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**。
5. 在**ASM网关**页面单击目标入口网关**操作**列下的**YAML**。
6. 在入口网关服务对应的YAML定义文件中，添加如下信息到 `spec` ：

```
  secretVolumes:
    - mountPath: /etc/istio/ingressgateway-certs
      name: ingressgateway-certs
      secretName: istio-ingressgateway-certs
```

添加之后的YAML文件内容类似如下：

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
metadata:
  name: ingressgateway
  namespace: istio-system
spec:
  clusterIds:
    - c58638b491a5248669b59609e0a17****
  cpu: {}
  externalTrafficPolicy: Local
  maxReplicas: 1
  minReplicas: 1
  ports:
    - name: status-port
      port: 15020
      targetPort: 15020
    - name: http2
      nodePort: 31380
      port: 80
      targetPort: 80
    - name: https
      nodePort: 31390
      port: 443
      targetPort: 443
    - name: tls
      port: 15443
      targetPort: 15443
  replicaCount: 1
  secretVolumes:
    - mountPath: /etc/istio/ingressgateway-certs
      name: ingressgateway-certs
      secretName: istio-ingressgateway-certs
  serviceAnnotations:
    service.beta.kubernetes.io/alibaba-cloud-loadbalancer-spec: slb.s1.small
  serviceType: LoadBalancer
```

步骤四：在集群中检查入口网关配置

1. 执行以下命令，验证tls.crt与tls.key已被挂载到入口网关Pod中。

```
kubectl exec -it -n istio-system $(kubectl -n istio-system get pods -l istio=ingressgat
eway -o jsonpath='{.items[0].metadata.name}') -- ls -al /etc/istio/ingressgateway-certs
```

预期输出：

```
lrwxrwxrwx 1 root root   14 May  9 01:14 tls.crt -> ..data/tls.crt
lrwxrwxrwx 1 root root   14 May  9 01:14 tls.key -> ..data/tls.key
```

2. 执行以下命令，检查Ingress gateway证书中的 `Subject` 字段的正确性。

```
kubectl exec -it -n istio-system $(kubectl -n istio-system get pods -l istio=ingressgat
eway -o jsonpath='{.items[0].metadata.name}')  -- cat /etc/istio/ingressgateway-certs/t
ls.crt | openssl x509 -text -noout | grep 'Subject:'
```

预期输出：

```
Subject: O = cert-manager + O = cluster.local, CN = istio-ingressgateway-certs
```

3. 执行以下命令，检查Ingress gateway的代理能够正确访问证书。

```
kubectl exec -it -n istio-system $(kubectl -n istio-system get pods -l istio=ingressgat
eway -o jsonpath='{.items[0].metadata.name}') -- curl  127.0.0.1:15000/certs
```

预期输出：

```
{
 "certificates": [
  {
   "ca_cert": [
    {
     "path": "\u003cinline\u003e",
     "serial_number": "4edcf3",
     "subject_alt_names": [],
     "days_until_expiration": "7251",
     "valid_from": "2022-01-06T06:17:00Z",
     "expiration_time": "2042-01-01T06:22:03Z"
    }
   ],
...........
```

步骤五：验证使用HTTPS协议访问服务是否成功

1. 在集群中创建服务。

i. 使用以下内容，创建*myexample-nginx.conf*。

本文示例服务是基于Nginx实现的，您需要为Nginx服务器创建配置文件，以域名example.com的服务为例，以下内容定义请求根路径直接返回字样 `Welcome to aliyun.com!` 及状态码200。

```
events {
}
http {
  log_format main '$remote_addr - $remote_user [$time_local]  $status '
  '"$request" $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log /var/log/nginx/access.log main;
  error_log  /var/log/nginx/error.log;
  server {
    listen 80;
    location / {
        return 200 'Welcome to aliyun.com!';
        add_header Content-Type text/plain;
    }
  }
}
```

ii. 执行以下命令，创建Kubernetes ConfigMap存储Nginx服务器的配置。

```
kubectl create configmap myexample-nginx-configmap --from-file=nginx.conf=./myexamp
le-nginx.conf
```

iii. 执行以下命令，为命名空间default启用Sidecar自动注入。

```
kubectl label namespace default istio-injection=enabled
```

iv. 使用以下内容，创建*myexampleapp.yaml*。

```
apiVersion: v1
kind: Service
metadata:
  name: myexampleapp
  labels:
    app: myexampleapp
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: myexampleapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myexampleapp
spec:
  selector:
    matchLabels:
      app: myexampleapp
  replicas: 1
  template:
    metadata:
      labels:
        app: myexampleapp
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx
          readOnly: true
      volumes:
      - name: nginx-config
        configMap:
          name: myexample-nginx-configmap
```

v. 执行以下命令，创建域名example.com的内部服务。

```
kubectl apply -f myexampleapp.yaml
```

2. 通过kubectl连接ASM。具体操作，请参见通过kubectl连接ASM实例。
3. 在ASM中创建网关规则。

i. 使用以下内容，创建*istio-myexample-customingressgateway.yaml*。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-myexample-customingressgateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - '*.example.com'
    port:
      name: http
      number: 80
      protocol: HTTP
    tls:
      httpsRedirect: true
  - hosts:
    - '*.example.com'
    port:
      name: https
      number: 443
      protocol: HTTPS
    tls:
      mode: SIMPLE
      privateKey: /etc/istio/ingressgateway-certs/tls.key
      serverCertificate: /etc/istio/ingressgateway-certs/tls.crt
```

ii. 执行以下命令，创建网关规则。

```
kubectl apply -f istio-myexample-customingressgateway.yaml
```

4. 在ASM中创建虚拟服务。

i. 使用以下内容，创建*istio-myexample-customvirtualservice.yaml*。

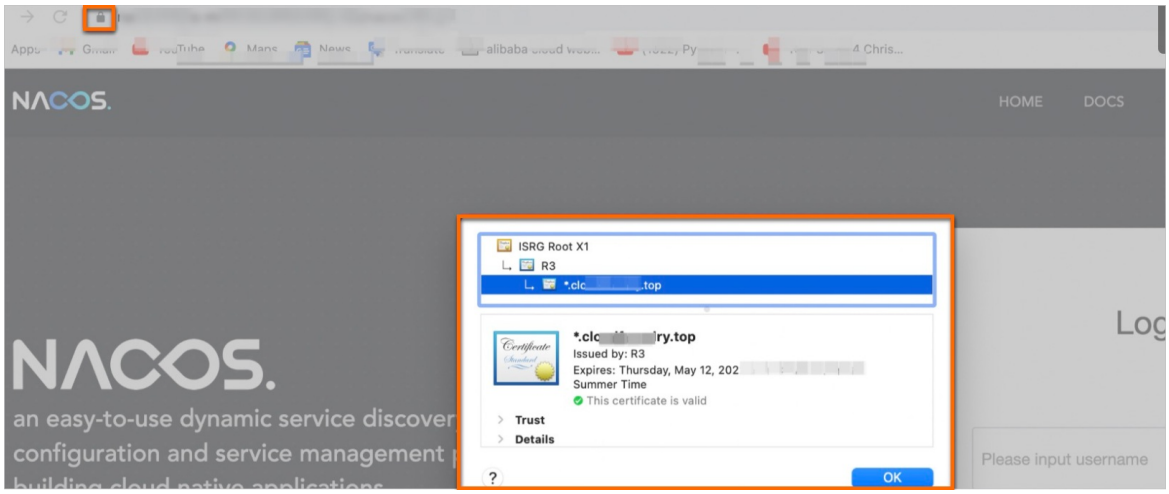
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: istio-myexample-customvirtualservice
spec:
  hosts:
  - "www.example.com"
  gateways:
  - istio-myexample-customingressgateway
  http:
  - route:
    - destination:
        host: myexampleapp.default.svc.cluster.local
        port:
          number: 80
```

ii. 执行以下命令，创建虚拟服务。

```
kubectl apply -f istio-myexample-customvirtualservice.yaml
```

5. 在浏览器中输入域名，例如example.com。

   返回以下页面，说明使用HTTPS协议访问服务成功。



2.14. 使用Gateway API定义路由规则

Gateway API是由SIG-NETWORK社区管理的开源项目，通过提供可表达的、可扩展的、面向角色的接口来改善服务网络。您可以使用Gateway API对集群内应用访问的路由规则进行条件限制。本文介绍如何使用Gateway API定义集群内应用的路由规则。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。
- 已部署应用到ASM实例的集群中。具体操作，请参见部署应用到ASM实例。
- 已创建ASM实例，且版本为专业版、企业版或旗舰版。具体操作，请参见创建ASM实例。

步骤一：在ACK集群中部署示例应用

1. 为default命名空间开启自动注入。具体操作，请参见启用自动注入。
2. 通过kubectl连接集群。具体操作，请参见通过kubectl工具连接集群。
3. 使用以下内容，创建名为*httpbin*的YAML文件。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: httpbin
---
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  labels:
    app: httpbin
    service: httpbin
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
      version: v1
  template:
    metadata:
      labels:
        app: httpbin
        version: v1
    spec:
      serviceAccountName: httpbin
      containers:
      - image: docker.io/kennethreitz/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        ports:
        - containerPort: 80
```

4. 执行以下命令，创建httpbin应用。

```
kubectl apply -f httpbin.yaml
```

5. 执行以下命令，验证Pod是否正常运行。

```
kubectl get pod |grep httpbin
```

预期输出：

```
httpbin-66cdbdb6c5-vhsh6          2/2     Running   0          11s
```

步骤二：在ASM中配置Gateway API

1. 通过kubectl连接ASM实例，具体操作，请参见通过kubectl连接ASM实例。
2. 使用以下内容，创建名为*gw*的YAML文件。

```
apiVersion: networking.x-k8s.io/v1alpha1
kind: GatewayClass
metadata:
  name: istio
spec:
  controller: istio.io/gateway-controller
---
apiVersion: networking.x-k8s.io/v1alpha1
kind: Gateway
metadata:
  name: gateway
  namespace: istio-system
spec:
  gatewayClassName: istio
  listeners:
  - hostname: "*"
    port: 80
    protocol: HTTP
    routes:
      namespaces:
        from: All
      selector:
        matchLabels:
          selected: "yes"
      kind: HTTPRoute
---
apiVersion: networking.x-k8s.io/v1alpha1
kind: HTTPRoute
metadata:
  name: http
  namespace: default
  labels:
    selected: "yes"
spec:
  gateways:
    allow: All
  hostnames: ["httpbin.aliyun.com"]
  rules:
  - matches:
    - path:
        type: Prefix
        value: /get
    filters:
    - type: RequestHeaderModifier
      requestHeaderModifier:
        add:
          my-added-header: added-value
    forwardTo:
    - serviceName: httpbin
      port: 8000
```

- 配置Gateway

  `port`：设置端口为80，表示通过80端口访问应用。

- 配置HTTPRoute
  - `hostnames`：主机名称，本文设置为`httpbin.aliyun.com`。
  - `path`：设置`type`为`Prefix`，`value`为`/get`，表示访问应用的URL中必须有`get`参数。

3. 执行以下命令，配置Gateway API。

```
kubectl --kubeconfig=<ASM实例对应的Kubeconfig文件> apply -f gw.yaml
```

步骤三：获取入口网关的地址

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，选择**网络 > 服务**。
5. 在**服务**页面顶部设置**命名空间**为istio-system，查看istio-ingressgateway**外部端点**列下端口为80的IP地址。

结果验证

- 执行以下命令，访问httpbin应用。

```
curl -s -HHost:httpbin.aliyun.com "http://$INGRESS_HOST:$INGRESS_PORT/get"
```

使用`-H`将HTTP的Host设置为`httpbin.aliyun.com`。即上文配置HTTPRoute时设置的hostnames值。

预期输出：

```
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "0",
    "Host": "httpbin.aliyun.com",
    "My-Added-Header": "added-value",
    "User-Agent": "curl/7.75.0",
    "X-Envoy-Attempt-Count": "1",
    "X-Envoy-External-Address": "xxxxxx"
  },
  "origin": "xxxxxx",
  "url": "http://httpbin.aliyun.com/get"
}
```

- 执行以下命令，访问httpbin应用。

```
curl -I -HHost:httpbin.aliyun.com "http://$INGRESS_HOST:$INGRESS_PORT/headers"
```

预期输出：

```
HTTP/1.1 404 Not Found
```

可以看到，包含 `get` 参数的URL访问应用成功，包含 `headers` 参数的URL访问应用失败。说明只有包含 `get` 参数的URL，才能访问应用成功。使用Gateway API定义路由规则成功。

2.15. 通过服务网关启用TLS透传

本文介绍如何通过服务网关启用TLS透传，以实现对集群内HTTPS服务的安全入口访问。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。
- 已部署应用到ASM实例的集群中。具体操作，请参见部署应用到ASM实例。
- 使用域名时需要备案才能正常访问，本示例中使用*aliyun.com*。

步骤一：生成服务器证书和私钥

如果您已经拥有针对 `sample.aliyun.com` 可用的证书和私钥，需要将密钥命名为 `sample.aliyun.com.key` ，证书命名为 `sample.aliyun.com.crt` 。如果没有，可以通过openssl执行以下操作来生成证书和密钥。

1. 执行以下命令，创建根证书和私钥。

```
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -subj '/O=mynginx Inc./CN=aliyun.com' -keyout aliyun.root.key -out aliyun.root.crt
```

2. 执行以下命令，为 `sample.aliyun.com` 服务器生成证书和私钥。

```
openssl req -out sample.aliyun.com.csr -newkey rsa:2048 -nodes -keyout sample.aliyun.com.key -subj "/CN=sample.aliyun.com/O=mynginx sample organization"
openssl x509 -req -days 365 -CA aliyun.root.crt -CAkey aliyun.root.key -set_serial 0 -in sample.aliyun.com.csr -out sample.aliyun.com.crt
```

步骤二：定义内部服务

示例中的内部服务是基于Nginx实现的，首先为Nginx服务器创建配置文件。以域名**aliyun.com**的内部服务为例，定义请求根路径直接返回字样 `Welcome to aliyun.com without TLS Termination!` 及状态码200。*mynginx.conf*的具体内容如下。

```
events {
}
http {
  log_format main '$remote_addr - $remote_user [$time_local]  $status '
  '"$request" $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log /var/log/nginx/access.log main;
  error_log  /var/log/nginx/error.log;
  server {
    listen 443 ssl;
    location / {
        return 200 'Welcome to aliyun.com without TLS Termination!';
        add_header Content-Type text/plain;
    }
    server_name www.aliyun.com;
    ssl_certificate /etc/nginx-server-certs/tls.crt;
    ssl_certificate_key /etc/nginx-server-certs/tls.key;
  }
}
```

1. 在入口网关Pod所在的集群对应的kubeconfig环境下，执行以下命令，创建Kubernetes ConfigMap存储Nginx服务器的配置。

   ```
   kubectl create configmap mynginx-configmap --from-file=nginx.conf=./mynginx.conf
   ```

2. 在入口网关Pod所在的集群对应的kubeconfig环境下，执行以下命令，将在default命名空间中创建包含证书和私钥的Secret。

   ```
   kubectl create secret tls nginx-server-certs --key sample.aliyun.com.key --cert sample.
   aliyun.com.crt
   ```

3. 在入口网关Pod所在的集群对应的kubeconfig环境下，执行以下命令，设置命名空间default，启用sidecar自动注入。

   ```
   kubectl label namespace default istio-injection=enabled
   ```

4. 创建并拷贝以下内容到mynginxapp.yaml文件中，并执行 `kubectl apply -f mynginxapp.yaml` 命令，创建域名aliyun.com的内部服务。

```
apiVersion: v1
kind: Service
metadata:
  name: mynginxapp
  labels:
    app: mynginxapp
spec:
  ports:
  - port: 443
    protocol: TCP
  selector:
    app: mynginxapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mynginxapp
spec:
  selector:
    matchLabels:
      app: mynginxapp
  replicas: 1
  template:
    metadata:
      labels:
        app: mynginxapp
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 443
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx
          readOnly: true
        - name: nginx-server-certs
          mountPath: /etc/nginx-server-certs
          readOnly: true
      volumes:
      - name: nginx-config
        configMap:
          name: mynginx-configmap
      - name: nginx-server-certs
        secret:
          secretName: nginx-server-certs
```

5. 确认Nginx服务器已成功部署，执行以下命令从其sidecar代理向服务器发送请求。

```
kubectl exec -it $(kubectl get pod  -l app=mynginxapp -o jsonpath={.items..metadata.nam
e}) -c istio-proxy -- curl -v -k --resolve sample.aliyun.com:443:127.0.0.1 https://samp
le.aliyun.com
```

步骤三：创建自定义网关配置对象

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**流量管理 > 网关规则**，然后在右侧页面单击**使用YAML创建**。
5. 按以下步骤定义网关规则，然后单击**创建**。
   i. 选择相应的命名空间。本文以选择**default**命名空间为例。
   ii. 在弹出窗口的文本框中，定义服务网关。可参考以下YAML定义。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: istio-mynginx-customingressgateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - 'sample.aliyun.com'
    port:
      name: https
      number: 443
      protocol: HTTPS
    tls:
      mode: PASSTHROUGH
```

在**网关规则（Gateway）**页面可以看到新建的网关。

步骤四：创建虚拟服务

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**流量管理 > 虚拟服务**，然后在右侧页面单击**使用YAML创建**。
5. 按以下步骤定义虚拟服务，然后单击**创建**。
   i. 选择相应的命名空间。本文以选择**default**命名空间为例。

ii. 在文本框中，定义Istio虚拟服务。可参考以下YAML定义。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: istio-mynginx-customvirtualservice
spec:
  hosts:
  - "sample.aliyun.com"
  gateways:
  - istio-mynginx-customingressgateway
  tls:
  - match:
    - port: 443
      sniHosts:
      - sample.aliyun.com
    route:
    - destination:
        host: mynginxapp.default.svc.cluster.local
        port:
          number: 443
```

在**虚拟服务（VirtualService）**页面可以看到新建的虚拟服务。

执行结果

可以通过以下任意一种方法获取入口网关服务的地址：

- 通过控制台查看，请参见查看入口网关的服务信息。
- 执行以下命令（注意是在入口网关Pod所在的集群对应的kubeconfig环境下），获取入口网关服务的地址。

```
kubectl get svc -n istio-system -l istio=ingressgateway
```

- 执行以下命令，通过HTTPS协议访问aliyun.com服务。

```
curl -v --cacert aliyun.root.crt --resolve sample.aliyun.com:443:xx.xx.xx.xx   https://sample.aliyun.com
```

```
Welcome to aliyun.com without TLS Termination!%
```

2.16. 如何解决Kubernetes集群中的Pod无法访问入口网关的SLB地址的问题？

本文介绍如何解决数据面Kubernetes集群中的Pod无法访问入口网关的SLB地址的问题。

问题现象

服务网格ASM已添加Kubernetes集群，并且使用Local类型的负载均衡SLB部署入口网关。当应用Pod访问入口网关暴露的SLB地址时，出现以下问题：

- 部分节点上的Pod能访问入口网关暴露的SLB地址。
- 部分节点上的Pod不能访问入口网关暴露的SLB地址。

问题原因

如果Kubernetes集群的服务配置了 `externalTrafficPolicy: Local` ，则只有部署了服务的后端Pod才能访问SLB地址。因为SLB地址是集群外使用，如果集群节点和Pod不能直接访问SLB地址，请求不会路由到负载均衡，而是被当作服务的扩展IP地址，被kube-proxy的iptables或IPVS转发。

如果集群节点或者Pod所在的节点上没有相应的后端服务Pod，就会发生网络不通的问题。而如果有相应的后端服务Pod，则可以正常访问SLB地址。更多信息，请参见iptables规则。

解决方案

- 您可以在Kubernetes集群内通过ClusterIP或者服务名访问SLB地址。其中入口网关的服务名为istio-ingressgateway.istio-system。

  > **说明** 推荐使用该方法。

- 如果您不需要源IP，您可以使用以下方法。

  将入口网关服务中的 `externalTrafficPolicy` 修改为 `Cluster` ，但是在应用中会丢失源IP。关于修改入口网关的更多信息，请参见修改入口网关服务。

  ```
  apiVersion: istio.alibabacloud.com/v1beta1
  kind: IstioGateway
  metadata:
    name: ingressgateway
    namespace: istio-system
    ....
  spec:
    externalTrafficPolicy: Cluster
  ....
  ```

- 如果您使用的是Terway的ENI或者ENI多IP集群，您可以使用以下方法。该方法不会丢失源IP，也支持在集群内访问SLB地址。

  将入口网关服务中的 `externalTrafficPolicy` 修改为 `Cluster` ，并且添加ENI直通的Annotation，例如 `serviceAnnotations: service.beta.kubernetes.io/backend-type: "eni"` 。关于修改入口网关的更多信息，请参见修改入口网关服务。

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: IstioGateway
metadata:
  name: ingressgateway
  namespace: istio-system
  ....
spec:
  externalTrafficPolicy: Cluster
  maxReplicas: 5
  minReplicas: 2
  ports:
    - name: status-port
      port: 15020
      targetPort: 15020
    - name: http2
      port: 80
      targetPort: 80
    - name: https
      port: 443
      targetPort: 443
    - name: tls
      port: 15443
      targetPort: 15443
  replicaCount: 2
  resources:
    limits:
      cpu: '2'
      memory: 2G
    requests:
      cpu: 200m
      memory: 256Mi
  runAsRoot: false
  serviceAnnotations:
    service.beta.kubernetes.io/backend-type: eni
  serviceType: LoadBalancer
```

3.配置推送优化

3.1. 配置推送优化概述

ASM提供了选择性服务发现和Sidecar资源推荐功能，帮助您优化控制平面的配置推送效率与Sidecar的配置大小。本文介绍选择性服务发现和Sidecar资源推荐功能和适用场景。

在默认情况下，由于无法确定数据平面内所有工作负载与服务之间的关系，服务网格数据平面内的所有Sidecar都必须保存数据平面集群内所有服务信息的全量配置。同时，一次针对控制平面或数据平面的修改（例如在控制平面新建一条虚拟服务规则），都会导致控制平面向数据平面的所有Sidecar推送新的配置。

当您在数据平面集群中的工作负载数量比较多时，这种管控方式会增加Sidecar对数据平面集群的资源消耗，同时控制平面会面临巨大的配置推送负担，降低控制平面的效率与可用性。ASM提供了选择性服务发现和Sidecar资源推荐功能，帮助您优化配置推送效率。

选择性服务发现

功能介绍

您可以使用选择性服务发现功能，根据数据平面集群内命名空间的标签来给定若干标签选择器。标签选择器保证ASM控制平面只需要发现和处理指定命名空间下的应用服务。Sidecar配置内将仅保留被选中命名空间内的服务信息，未被选中的命名空间内的服务发生改变，将不会引起Sidecar的配置推送。更多选择性服务发现的介绍，请参见使用选择性服务发现提升控制平面推送效率。

适用场景

选择性服务发现功能通过命名空间的标签选择器限制了控制平面的管控范围，从而优化Sidecar配置大小，减少不必要的配置推送。当您的数据平面中包含大量的命名空间与服务，而您只希望服务网格ASM仅对其中少数几个命名空间中的服务进行服务发现时，您可以使用选择性服务发现功能来提升配置推送效率。

Sidecar资源推荐

功能介绍

您可以使用Sidecar资源推荐功能，通过分析数据平面Sidecar产生的访问日志获取数据平面服务之间的调用依赖关系，为数据平面中的每个工作负载自动推荐Sidecar资源。为工作负载推荐Sidecar资源后，将实现以下功能：

- Sidecar配置内将仅保留该Sidecar对应工作负载所依赖的服务信息。
- 当该Sidecar资源对应的工作负载无依赖关系的服务发生改变，或与该服务相关的资源发生改变（例如虚拟服务等），都不会引起控制平面向该Sidecar的配置推送，这将大幅度提升了控制面向数据面的配置推送效率。关于应用Sidecar资源后的配置推送优化效果，请参见应用Sidecar资源后的配置推送优化效果。

更多Sidecar资源推荐的介绍，请参见使用基于访问日志分析自动推荐的Sidecar资源。

适用场景

如果选择性服务发现无法满足您的配置推送优化需求，且您的单个命名空间中存在着大量的服务，需要对Sidecar配置进行最大限度的精简，您可以使用ASM基于访问日志的Sidecar资源推荐功能，此功能无需手动编写YAML文件，即可为您的工作负载创建Sidecar资源，降低手动创建Sidecar资源的困难。

3.2. 使用选择性服务发现提升控制平面推送效率

您可以通过选择性服务发现功能保证ASM控制平面只发现和处理指定命名空间下的应用，提升控制平面推送到数据平面Sidecar的效率。本文介绍如何使用选择性服务发现提升控制平面推送效率。

前提条件

- 已创建ASM实例，且ASM实例为v1.10.5.32-g28fe5008-aliyun或以上版本。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 通过kubectl连接集群。具体操作，请参见通过kubectl工具连接集群。

背景信息

默认情况下，数据平面的Sidecar中保存数据平面集群中任意命名空间内所有服务的相关信息（即使该命名空间内的工作负载并未注入Sidecar），同时，控制平面也会监视网格中来自所有命名空间内的服务，任何与服务相关的变更都会引起控制平面向所有Sidecar推送相关配置。

您可以使用选择性服务发现功能，根据数据平面集群内命名空间的标签来制定若干标签选择器。标签选择器保证ASM控制平面只需要发现和处理指定命名空间下的应用服务。Sidecar配置内将仅保留被选中命名空间内的服务信息，未被选中的命名空间内的服务发生改变，将不会引起Sidecar的配置推送。

选择性服务发现功能的标签选择器支持以下两种规则：

- 标签精确匹配规则：您需要指定一个标签名和一个标签值，只有命名空间上的标签完全匹配给定的标签名和标签值才能够匹配成功。
- 标签表达式匹配规则：您可以指定一个标签名、一个表达式操作符、以及一系列标签值来匹配数据平面中的命名空间标签，操作符含义如下：
  - In：数据平面命名空间的标签必须包含给定标签名的标签，且其标签值必须位于给定的一系列标签值中。
  - NotIn：数据平面命名空间的标签必须包含给定标签名的标签，且其标签值必须不在给定的一系列标签值中。
  - Exists：数据平面命名空间的标签必须包含给定标签名的标签，在操作符为Exists时，无需给出任何标签值。
  - DoesNotExist：数据平面命名空间的标签必须不能包含给定标签名的标签，在操作符为DoesNotExist时，无需给出任何标签值。

步骤一：在ASM中创建两个命名空间

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 全局命名空间**，然后在右侧页面单击**新建**。
5. 在**新建命名空间**面板设置命名空间名称，本文设置为ns-in-mesh，然后单击**确定**。
6. 重复步骤5，创建名为ns-not-in-mesh的命名空间。
7. 为ns-in-mesh命名空间启用自动注入。
   i. 在**全局命名空间**页面单击ns-in-mesh命名空间右侧**操作**列下的**启用Sidecar自动注入**。
   ii. 在**确认**对话框单击**确定**。

步骤二：在ACK集群中部署示例应用

1. 执行以下命令，在ACK集群中为ns-in-mesh命名空间添加 `asm-discovery=enabled` 标签。

```
kubectl label namespace ns-in-mesh asm-discovery=enabled
```

2. 使用以下内容，创建名为*httpbin*的YAML文件。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: httpbin
---
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  labels:
    app: httpbin
    service: httpbin
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
      version: v1
  template:
    metadata:
      labels:
        app: httpbin
        version: v1
    spec:
      serviceAccountName: httpbin
      containers:
      - image: docker.io/kennethreitz/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        ports:
        - containerPort: 80
```

3. 在两个命名空间中创建httpbin示例应用。
   - 执行以下命令，在ns-in-mesh创建httpbin示例应用。

```
kubectl apply -f httpbin.yaml -n ns-in-mesh
```

- 执行以下命令，在n-not-in-meshs创建httpbin示例应用。

```
kubectl apply -f httpbin.yaml -n n-not-in-meshs
```

步骤三：查看Sidecar配置的推送状态

1. 查看Sidecar配置。
   i. 执行以下命令，下载Sidecar配置。

   ```
   kubectl exec -it httpbin-74fb669cc6-mz72t -c istio-proxy -n ns-in-mesh -- curl -s l
   ocalhost:15000/config_dump > config_dump.json
   ```

   ii. 打开下载到本地的*config_dump.json*文件，搜索 `httpbin.ns-not-in-mesh.svc.cluster.local` 。

   可以搜索到 `httpbin.ns-not-in-mesh.svc.cluster.local` ，说明即使ns-not-in-mesh命名空间没有启用Sidecar自动注入，Sidecar配置信息中心仍然保存着ns-not-in-mesh命名空间中的服务信息。
2. 查看控制平面日志。
   i. 启用控制平面日志采集。具体操作，请参见启用控制平面日志采集和日志告警。
   ii. 在ACK集群中部署Sleep应用。
      a. 使用以下内容，创建名为*sleep.yaml*的YAML文件。

```
##################################################################################
###################
# Sleep service
##################################################################################
###################
apiVersion: v1
kind: Service
metadata:
  name: sleep
  labels:
    app: sleep
spec:
  ports:
  - port: 80
    name: http
  selector:
    app: sleep
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sleep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sleep
  template:
    metadata:
      labels:
        app: sleep
    spec:
      containers:
      - name: sleep
        image: pstauffer/curl
        command: ["/bin/sleep", "3650d"]
        imagePullPolicy: IfNotPresent
---
```

b. 执行以下命令，在ns-not-in-mesh命名空间部署应用。

```
kubectl apply -f sleep.yaml -n ns-not-in-mesh
```

iii. 查看日志。

a. 登录ASM控制台。

b. 在左侧导航栏，选择**服务网格 > 网格管理**。

c. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

d. 在网格详情页面选择**网格实例 > 基本信息**。

e. 在**基本信息**页面单击**控制面日志采集**右侧的**查看日志**。

在Project页面右上角设置时间为**5分钟**，缩小日志查看范围，可以在**原始日志**页签下查看到创建应用的日志。说明ns-not-in-mesh命名空间没有启用Sidecar自动注入，在ns-not-in-mesh中部署应用仍然导致了控制平面向数据平面中Sidecar推送配置信息。

```
2021-12-03T09:18:19.939580Z    info    ads    Incremental push, service sleep.n
s-not-in-mesh.svc.cluster.local has no endpoints
2021-12-03T09:18:20.040291Z    info    ads    Push debounce stable[18] 2: 100.6
61695ms since last change, 107.19307ms since last push, full=true
2021-12-03T09:18:20.040785Z    info    ads    XDS: Pushing:2021-12-03T09:18:20Z
/11 Services:13 ConnectedEndpoints:1  Version:2021-12-03T09:18:20Z/11
2021-12-03T09:18:20.041444Z    info    ads    CDS: PUSH for node:httpbin-74fb66
9cc6-zf4wg.ns-in-mesh resources:20 size:12.9kB
2021-12-03T09:18:20.041499Z    info    ads    EDS: PUSH for node:httpbin-74fb66
9cc6-zf4wg.ns-in-mesh resources:12 size:1.6kB empty:0 cached:12/12
2021-12-03T09:18:20.042280Z    info    ads    LDS: PUSH for node:httpbin-74fb66
9cc6-zf4wg.ns-in-mesh resources:16 size:80.8kB
2021-12-03T09:18:20.042472Z    info    ads    RDS: PUSH for node:httpbin-74fb66
9cc6-zf4wg.ns-in-mesh resources:7 size:5.3kB
2021-12-03T09:18:20.049506Z    info    ads    EDS: PUSH request for node:httpbi
n-74fb669cc6-zf4wg.ns-in-mesh resources:13 size:1.6kB empty:1 cached:12/13
2021-12-03T09:18:20.058780Z    info    ads    RDS: PUSH request for node:httpbi
n-74fb669cc6-zf4wg.ns-in-mesh resources:8 size:6.4kB
2021-12-03T09:18:28.260944Z    info    ads    Full push, new service ns-not-in-
mesh/sleep.ns-not-in-mesh.svc.cluster.local
2021-12-03T09:18:28.361036Z    info    ads    Push debounce stable[19] 1: 100.0
41329ms since last change, 100.041123ms since last push, full=true
2021-12-03T09:18:28.361524Z    info    ads    XDS: Pushing:2021-12-03T09:18:28Z
/12 Services:13 ConnectedEndpoints:1  Version:2021-12-03T09:18:28Z/12
2021-12-03T09:18:28.362134Z    info    ads    CDS: PUSH for node:httpbin-74fb66
9cc6-zf4wg.ns-in-mesh resources:20 size:12.9kB
2021-12-03T09:18:28.362238Z    info    ads    EDS: PUSH for node:httpbin-74fb66
9cc6-zf4wg.ns-in-mesh resources:13 size:1.7kB empty:0 cached:12/13
2021-12-03T09:18:28.362918Z    info    ads    LDS: PUSH for node:httpbin-74fb66
9cc6-zf4wg.ns-in-mesh resources:16 size:80.8kB
2021-12-03T09:18:28.363128Z    info    ads    RDS: PUSH for node:httpbin-74fb66
9cc6-zf4wg.ns-in-mesh resources:8 size:6.4kB
```

步骤四：在ASM中配置选择性服务发现

以下以设置标签精确匹配规则为例，使ASM控制平面将只推送配置至带有asm-discovery标签的命名空间下的应用，其他命名空间下的应用将不会引起Sidecar的配置推送。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

4. 在网格详情页面左侧导航栏选择**配置推送优化 > 选择性服务发现**。
5. 在**选择性服务发现**页面单击**添加标签选择器**，单击**添加标签表达式匹配规则**，设置**标签名**为asm-discovery，**操作符**为Exists，然后单击**确定**。
   此时，在**基本信息**页面**网格状态**显示**更新中**，待**网格状态**变更为**运行中**，说明标签选择器配置成功。

步骤五：验证使用选择性服务发现功能是否成功

1. 查看Sidecar配置。
   i. 执行以下命令，下载Sidecar配置。

   ```
   kubectl exec -it httpbin-74fb669cc6-mz72t -c istio-proxy -n ns-in-mesh -- curl -s l
   ocalhost:15000/config_dump > config_dump.json
   ```

   ii. 打开下载到本地的*config_dump.json*文件，搜索 `httpbin.ns-not-in-mesh.svc.cluster.local` 。
   搜索不到 `httpbin.ns-not-in-mesh.svc.cluster.local` ，说明Sidecar配置信息中心没有保存ns-not-in-mesh命名空间中的服务信息。
2. 查看控制平面日志。
   i. 启用控制平面日志采集。具体操作，请参见启用控制平面日志采集和日志告警。
   ii. 执行以下命令，在ACK集群中删除Sleep应用。

   ```
   kubectl delete -f sleep.yaml -n ns-not-in-mesh
   ```

   iii. 查看日志。
      a. 登录ASM控制台。
      b. 在左侧导航栏，选择**服务网格 > 网格管理**。
      c. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
      d. 在网格详情页面选择**网格实例 > 基本信息**。
      e. 在**基本信息**页面单击**控制面日志采集**右侧的**查看日志**。
      在Project页面右上角设置时间为**15分钟**，缩小日志查看范围，无法查看到关于删除应用的日志。说明在服务发现之外的命名空间发生变更，并不会引起控制平面向Sidecar推送配置信息。

3.3. 使用基于访问日志分析自动推荐的Sidecar资源

在默认情况下，由于不能确定网格内服务之间的调用依赖关系，Sidecar的配置中保存了数据平面内所有服务的信息。您可以使用Sidecar资源配置使对应工作负载上的Sidecar将仅关注与自己有调用依赖关系的服务信息。本文介绍如何使用基于访问日志分析自动推荐的Sidecar资源。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。
- 已部署应用到ASM实例的集群中。具体操作，请参见部署应用到ASM实例。

- 已使用日志服务采集数据平面的访问日志。具体操作，请参见使用日志服务采集数据平面的AccessLog。

背景信息

在默认情况下，针对控制平面或数据平面的任意一次修改都会引起控制平面向数据平面所有Sidecar的配置推送。服务网格ASM可以通过分析数据平面Sidecar产生的访问日志获取数据平面服务之间的调用依赖关系，为数据平面中的每个工作负载自动推荐Sidecar资源。为工作负载推荐Sidecar资源后：

- Sidecar配置内将仅保留该Sidecar对应工作负载所依赖的服务信息。
- 当该Sidecar资源对应的工作负载无依赖关系的服务发生改变，或与该服务相关的资源发生改变（例如虚拟服务等），都不会引起控制平面向该Sidecar的配置推送。

注意事项

- 如果您在设置启用访问日志采集时，设置日志服务Project为**使用默认**，ASM会默认选择您添加到网格中的第一个集群的日志服务Project用于Sidecar资源推荐。
- ASM的Sidecar资源推荐通过单独针对每个负载单独的服务依赖关系提供Sidecar资源，能够最大化精简数据面Sidecar配置，但其需要针对数据面中的每个工作负载进行单独配置，且由于推荐基于访问日志，如果访问日志中没有工作负载的服务调用记录，可能造成推荐的Sidecar不准确，因此您需要对Sidecar进行二次确认。

步骤一：产生访问日志

在浏览器地址栏输入*http://{入口网关服务的IP地址}/productpage*。持续刷新，直至观察到Bookinfo应用页面在显示黑色星星图标和红色星星图标之间轮流切换。

步骤二：为工作负载推荐Sidecar资源

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**配置推送优化 > Sidecar资源推荐**。
5. 在**Sidecar资源推荐**页面单击目标应用右侧**操作**列下的**推荐**。

   当目标应用的**推荐任务状态**为**推荐完成**，说明为工作负载推荐Sidecar资源成功。
6. 在**Sidecar资源推荐**页面单击目标应用右侧**操作**列下的**查看**。
7. 在**推荐的Sidecar资源**面板检查Sidecar的YAML文件中的*hosts*参数值是否包含工作负载的所有调用服务，确认无误后，单击**创建**。

   如果您发现Sidecar的YAML文件*hosts*缺少了服务，您可以在YAML文件进行补充，然后再创建Sidecar资源。

（可选）步骤三：重新为工作负载推荐Sidecar资源

随着业务应用的更新升级，服务网格中服务之间的调用依赖关系可能会发生改变，此时之前ASM推荐的Sidecar资源将不再有效，此时建议您重新收集日志，并重新为对应工作负载推荐Sidecar资源。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**配置推送优化 > Sidecar资源推荐**。
5. 在**Sidecar资源推荐**页面单击目标应用右侧**操作**列下的**重新收集日志**。

6. 在**重新收集日志**对话框单击**确定**。

   此时被成功应用的Sidecar资源将被删除，同时目标应用的**推荐任务状态**会显示**重新收集日志中**。

7. 重新为访问日志分析产生访问日志。具体操作，请参见步骤一：产生访问日志。
8. 重新为工作负载推荐Sidecar资源。具体操作，请参见步骤二：为工作负载推荐Sidecar资源。

3.4. 应用Sidecar资源后的配置推送优化效果

如果您的单个命名空间中存在着大量的服务，需要对Sidecar配置进行最大限度的精简，您可以使用Sidecar资源推荐功能。本文以部署420个Pod的较大规模集群为例，测试并分析应用Sidecar资源后的服务网格配置推送优化效果。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 通过kubectl连接ACK集群。具体操作，请参见通过kubectl工具连接集群。
- 已使用日志服务采集数据平面的访问日志。具体操作，请参见使用日志服务采集数据平面的AccessLog。

在集群中部署测试应用

本文使用多个sleep应用与httpbin应用模拟集群中存在数量庞大的服务，且服务与服务之间只存在少量调用依赖关系。

- httpbin应用在启动后会在8000端口暴露一个HTTP服务，用于模拟集群内部大量被调用的服务。
- sleep应用包含一个curl容器，通过修改应用部署的 `command` 字段，使sleep应用在休眠之前调用多个httpbin的容器提供的服务，用于模拟集群内依赖其它服务的服务。

1. 在集群中部署多个httpbin应用。

   i. 使用以下YAML内容，创建名为*httpbin-{i}*的YAML文件。

   > **说明** `httpbin-{i}` 中的 `{i}` 可用具体数字代替，以生成多个不同的带编号的httpbin服务。使用此模板可以生成任意多个httpbin应用，具体应用数量限制取决于集群的规模。本文以使用该模板生成200个httpbin应用部署，在集群中共部署400个httpbin应用的Pod为例。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: httpbin
---
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: httpbin-{i}
    service: httpbin-{i}
  name: httpbin-{i}
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin-0
---
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: httpbin-{i}
  name: httpbin-{i}
spec:
  replicas: 2
  selector:
    matchLabels:
      app: httpbin-{i}
      version: v1
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: httpbin-{i}
        version: v1
    spec:
      containers:
      - image: docker.io/kennethreitz/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        ports:
        - containerPort: 80
      serviceAccountName: httpbin
```

ii. 执行以下命令，创建httpbin-{i}应用。

```
kubectl apply -f httpbin-{i}.yaml
```

预期输出：

```
deployment.apps/httpbin-{i} created
```

2. 在集群中部署sleep应用。

i. 使用以下YAML内容，创建名为*sleep-{i}*的YAML文件。

> **说明** `sleep-{i}` 中的 `{i}` 可用具体数字代替，以生成多个不同的带编号的sleep服务。在此模板中，通过向sleep应用部署的 `args` 字段增加 `curl httpbin-{i*10}:8000` 的命令参数，模拟向不同的httpbin应用的调用依赖。此处调用httpbin服务的编号不能超过之前部署的httpbin服务编号，否则无法产生有效调用。本文以模拟每个sleep应用依赖10个httpbin应用为例，因此共部署20个sleep应用部署，20个sleep Pod。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sleep
---
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: sleep-{i}
    service: sleep-{i}
  name: sleep-{i}
spec:
  ports:
  - name: http
    port: 80
    targetPort: 0
  selector:
    app: sleep-{i}
---
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: sleep-{i}
  name: sleep-{i}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sleep-{i}
  template:
    metadata:
      creationTimestamp: null
```

```
      labels:
        app: sleep-{i}
    spec:
      containers:
      - args:
        - curl httpbin-{i*10}:8000; curl httpbin-{i*10+1}:8000; curl httpbin-{i*10+
2}:8000; curl httpbin-{i*10+3}:8000;
          curl httpbin-{i*10+4}:8000; curl httpbin-{i*10+5}:8000; curl httpbin-{i*1
0+6}:8000; curl httpbin-{i*10+7}:8000;
          curl httpbin-{i*10+8}:8000; curl httpbin-{i*10+9}:8000; sleep 3650d
        command:
        - /bin/sh
        - -c
        image: curlimages/curl
        imagePullPolicy: IfNotPresent
        name: sleep
        volumeMounts:
        - mountPath: /etc/sleep/tls
          name: secret-volume
      serviceAccountName: sleep
      terminationGracePeriodSeconds: 0
      volumes:
      - name: secret-volume
        secret:
          optional: true
          secretName: sleep-secret
```

ii. 执行以下命令，创建sleep-{i}应用。

```
kubectl apply -f sleep-{i}.yaml
```

预期输出：

```
deployment.apps/sleep-{i} created
```

测试使用Sidecar资源前的控制面配置推送情况

场景一：测试使用Sidecar资源优化前每个Sidecar的配置大小

1. 执行以下命令，确定httpbin-0应用的Pod名称。

```
kubectl get pod -n ns-in-mesh | grep httpbin-0
```

预期输出：

```
httpbin-0-756995d867-j****   2/2     Running   0          9m15s
httpbin-0-756995d867-w****   2/2     Running   0          9m15s
```

2. 执行以下命令，下载httpbin-0应用所在Pod的Sidecar，保存至本地。

```
kubectl exec -it httpbin-0-756995d867-j**** -c istio-proxy -n ns-in-mesh -- curl -s loc
alhost:15000/config_dump > config_dump.json
```

3. 执行以下命令，查看Sidecar配置文件的大小。

```
du -sh config_dump.json
```

预期输出：

```
1.2M    config_dump.json
```

由预期输出得到，在集群中部署了420个Pod的测试场景下，Sidecar的配置大小约为1.2 MB。如果集群中每个Pod都部署Sidecar，大量的Sidecar配置会加重控制面的推送负担。

场景二：测试使用Sidecar资源优化前控制面的配置推送效率

在ASM中为httpbin-0服务应用一个新的虚拟服务规则，触发控制面向数据面Sidear的一次配置推送。本文通过查看控制面日志内容来测试控制面在一次推送中的配置推送效率。启用控制平面日志采集的具体步骤，请参见启用控制平面日志采集和日志告警。

1. 使用以下YAML内容，在服务网格中新建一个对httpbin-0服务进行超时处理的虚拟服务。新建虚拟服务的具体操作，请参见管理虚拟服务。

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: httpbin-0-timeout
  namespace: default
spec:
  hosts:
    - httpbin-0.default.svc.cluster.local
  http:
    - route:
        - destination:
            host: httpbin-0.default.svc.cluster.local
      timeout: 5s
```

2. 查看控制面新产生的日志。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面单击目标实例的名称。
   iv. 在网格详情页面选择**网格实例 > 基本信息**。

v. 在**基本信息**页面单击**控制面日志采集**右侧的**查看日志**。

预期输出：

```
2021-12-01T10:20:09.708673Z info  ads CDS: PUSH for node:httpbin-27-7dd8578b46-n***
*.default resources:227 size:169.3kB
2021-12-01T10:20:09.710469Z info  ads CDS: PUSH for node:httpbin-184-65d97797db-n**
**.default resources:227 size:169.3kB
2021-12-01T10:20:09.713567Z info  ads CDS: PUSH for node:httpbin-86-5b64586bbf-j***
*.default resources:227 size:169.3kB
2021-12-01T10:20:09.714514Z info  ads LDS: PUSH for node:httpbin-86-5b64586bbf-j***
*.default resources:16 size:70.7kB
2021-12-01T10:20:09.792732Z info  ads LDS: PUSH for node:httpbin-27-7dd8578b46-n***
*.default resources:16 size:70.7kB
2021-12-01T10:20:09.792982Z info  ads LDS: PUSH for node:httpbin-184-65d97797db-n**
**.default resources:16 size:70.7kB
2021-12-01T10:20:09.796430Z info  ads RDS: PUSH for node:httpbin-86-5b64586bbf-j***
*.default resources:8 size:137.4kB
……
2021-12-01T10:20:13.405850Z info  ads RDS: PUSH for node:httpbin-156-68b85b4f79-2**
**.default resources:8 size:137.4kB
2021-12-01T10:20:13.406154Z info  ads RDS: PUSH for node:httpbin-121-7c4cff97b9-s**
**.default resources:8 size:137.4kB
2021-12-01T10:20:13.406420Z info  ads CDS: PUSH for node:httpbin-161-7bc74c5fb5-l**
**.default resources:227 size:169.3kB
2021-12-01T10:20:13.407230Z info  ads LDS: PUSH for node:httpbin-161-7bc74c5fb5-l**
**.default resources:16 size:70.7kB
2021-12-01T10:20:13.410147Z info  ads RDS: PUSH for node:httpbin-161-7bc74c5fb5-l**
**.default resources:8 size:137.4kB
2021-12-01T10:20:13.494840Z info  ads RDS: PUSH for node:httpbin-57-69b756f779-d***
*.default resources:8 size:137.4kB
```

由预期输出得到，在部署了420个Pod的测试环境下，新增一个虚拟服务会导致控制面向数据面的所有Sidecar推送变更，产生大量推送日志，且每次推送的数据量较大。在服务网格中应用一条虚拟服务规则需要控制面长达约4秒的推送，控制面的推送效率较低。

测试使用Sidecar资源后的控制面配置推送情况

使用基于访问日志分析自动推荐的Sidecar资源功能，为测试集群中的每个工作负载推荐Sidecar资源，改善控制面的配置推送效率。具体操作，请参见使用基于访问日志分析自动推荐的Sidecar资源。

场景一：测试使用Sidecar资源优化后每个Sidecar的配置大小

1. 执行以下命令，下载httpbin-0应用所在Pod的Sidecar，保存至本地。

```
kubectl exec -it httpbin-0-756995d867-j**** -c istio-proxy -n ns-in-mesh -- curl -s loc
alhost:15000/config_dump > config_dump.json
```

2. 执行以下命令，查看Sidecar配置文件的大小。

```
du -sh config_dump.json
```

预期输出：

```
105k    config_dump.json
```

由预期输出得到，在集群中部署了420个Pod的场景下，使用基于访问日志分析自动推荐的Sidecar资源功能，Sidecar的配置大小缩小了10倍以上，大大提高控制面向数据面Sidecar的配置推送效率。

场景二：测试使用Sidecar资源优化后控制面的配置推送效率

重新在ASM中为httpbin-0服务应用一个新的虚拟服务规则，触发控制面向数据面Sidear的一次配置推送。

1. 在服务网格中删除之前创建的虚拟服务，使用以下YAML内容，在服务网格中新建一个对httpbin-0服务进行超时处理的虚拟服务。新建虚拟服务的具体操作，请参见管理虚拟服务。

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: httpbin-0-timeout
  namespace: default
spec:
  hosts:
    - httpbin-0.default.svc.cluster.local
  http:
    - route:
        - destination:
            host: httpbin-0.default.svc.cluster.local
      timeout: 5s
```

2. 查看控制面新产生的日志。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面单击目标实例的名称。
   iv. 在网格详情页面选择**网格实例 > 基本信息**。
   v. 在**基本信息**页面单击**控制面日志采集**右侧的**查看日志**。

   预期输出：

```
2021-12-01T12:12:43.498048Z info  ads Push debounce stable[750] 1: 100.03379ms sinc
e last change, 100.033692ms since last push, full=true
2021-12-01T12:12:43.504270Z info  ads XDS: Pushing:2021-12-01T12:12:43Z/493 Service
s:230 ConnectedEndpoints:421  Version:2021-12-01T12:12:43Z/493
2021-12-01T12:12:43.507451Z info  ads CDS: PUSH for node:sleep-0-b68c8c5d9-5****.de
fault resources:14 size:7.8kB
2021-12-01T12:12:43.507739Z info  ads LDS: PUSH for node:sleep-0-b68c8c5d9-5****.de
fault resources:3 size:15.5kB
2021-12-01T12:12:43.508029Z info  ads RDS: PUSH for node:sleep-0-b68c8c5d9-5****A.d
efault resources:1 size:6.3kB
```

   由预期输出得到，在应用ASM推荐的Sidecar资源后，数据面的每个工作负载将不再接收与其没有依赖关系的服务相关变更。对httpbin-0服务应用一条虚拟服务规则后，由于只有sleep-0应用与httpbin-0服务之间存在依赖关系，所以控制面仅向sleep-0应用所在Pod的Sidecar推送配置变更。应用一条虚拟服务规则触发的配置推送仅持续了约0.01秒，相比优化前提升了约400倍的推送效率。同时，变更的数据量缩小了约10倍，大幅度提升了控制面向数据面的配置推送效率。

效果对比总结

本次测试通过使用多个sleep应用与httpbin应用模拟集群中存在数量庞大的服务，但服务与服务之间只存在少量调用依赖关系，共部署200个httpbin应用，400个httpbin应用的Pod，20个sleep应用，20个sleep应用的Pod。以下为使用Sidecar资源优化前后的效果对比。

类别	未使用Sidecar资源优化	使用Sidecar资源优化
Sidecar配置大小	1.2 MB	105 KB
是否推送不含依赖关系的服务	是	否
控制面配置推送时间	约4秒	约0.01秒

4.对接服务注册中心

4.1. 对接Consul注册中心

服务网格ASM提供了对接Consul注册中心功能，便于您将微服务迁移至服务网格ASM。本文介绍如何在服务网格ASM中对接Consul注册中心。

前提条件

- 服务网格ASM版本需要升级到v1.7.5.31-g28ec7490-aliyun或者以上版本。
- 已部署Consul作为服务注册中心，具体操作，请参见Installing Consul on Kubernetes。
- 请确保加入服务网格中的Kubernetes集群的Pod可以访问Consul Server的访问地址。例如Consul Server是安装在相同的Kubernetes集群，Consul Server暴露了公网或者Consul Server提供了可访问的内网地址。
- Consul中已注册示例服务，分别为Web，Web2以及内置的Consul。具体操作，请参见Services。

背景信息

服务网格ASM提供了对接Consul注册中心的功能，便于您将微服务迁移至服务网格ASM的过程中，服务网格之内的服务能调用服务网格之外的服务。服务网格ASM并不会主动将服务网格中的服务信息注册到Consul注册中心。为了确保服务网格之外的服务能调用服务网格之内的服务，您需要将服务网格中的服务手动注册到Consul注册中心。

步骤一：获取ASM-se-syncer配置信息

1. 查看ASM实例ID。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
   iv. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**。
   在**网格信息**页面查看ASM实例ID。
2. 查看集群地域ID。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏单击**集群**。
   在**集群**页面查看目标集群的地域，例如您集群地域为华北2（北京），则集群地域ID为cn-beijing。
3. 查看AccessKey ID和AccessKey Secret。具体操作，请参见获取AccessKey。

步骤二：安装ASM-se-syncer

1. 已通过kubectl连接集群。具体操作，请参见通过kubectl工具连接集群。
2. 在本地安装Helm。具体操作，请参见Helm。

   > **说明** 使用kubectl连接集群后，Helm客户端会自动使用KubeConfig连接集群。

3. 下载并解压asm-se-syncer至本地。
4. 进入asm-se-syncer文件夹中，找到*values.yaml*文件，在*values.yaml*文件中补充ASM ID、集群地域ID、AccessKey ID和AccessKey Secret，并修改以下Consul服务信息，然后保存*values.yaml*文件。

```
[
  {
    "name": "consul-test",
    "prefix": "consul-",
    "type": "consul",
    "endpoint": "http://consul-server.consul:8500",
    "toNamespace": "default"
  }
]
```

参数	说明
`name`	注册中心的名称，保证唯一。
`prefix`	生成的服务条目的名称前缀。
`type`	注册中心的类型，当前支持值为consul。
`endpoint`	注册中心的访问端点地址。
`toNamespace`	生成的服务条目所在的命名空间，如果该命名空间不存在，将会自动创建。

如果您想对接多个Consul注册中心，您需要在*values.yaml*文件中补充ASM ID、集群地域ID、AccessKey ID和AccessKey Secret，并补充多个Consul服务信息，然后保存*values.yaml*。

```
[
  {
    "name": "consul-test01",
    "prefix": "consul01-",
    "type": "consul",
    "endpoint": "http://consul-server01.consul:8500",
    "toNamespace": "default"
  },
  {
    "name": "consul-test02",
    "prefix": "consul02-",
    "type": "consul",
    "endpoint": "http://consul-server02.consul:8500",
    "toNamespace": "default"
  }
]
```

5. 执行以下命令，安装ASM-se-syncer。

```
helm install -f values.yaml se-syncer ./
```

安装ASM-se-syncer成功后，ASM会自动对接Consul注册中心后，会自动在ACK集群安装asm-serviceregistry-syncer组件，并将在Consul中的服务同步到服务网格中。

步骤三：查看Consul注册中心对接结果

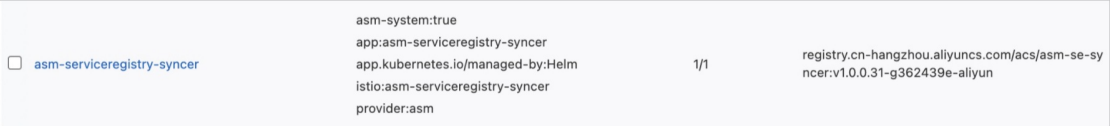
1. 查看ASM组件安装情况。
   i. 登录容器服务管理控制台。

ii. 在控制台左侧导航栏中，单击**集群**。

iii. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。

iv. 在集群管理页左侧导航栏中，选择**工作负载 > 无状态**。

v. 在**无状态**页面查看到名为asm-serviceregistry-syncer的组件。



2. 查看服务条目同步情况。

i. 登录ASM控制台。

ii. 在左侧导航栏，选择**服务网格 > 网格管理**。

iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

iv. 在网格详情页面左侧导航栏选择**集群与工作负载条目 > 服务条目**。

v. 在**服务条目**页面可以看到3个注册在Consul中的服务已经同步到服务网格中。

> **说明**
>
> - 在**服务条目**页面下的Consul服务的名称命名规则为：[上述步骤定义的prefix值]-[在Consul中注册的服务名]。
> - 在**服务条目**页面下的Consul服务的命名空间为上述步骤定义的 `toNamespace` 值。

常见问题

如何调用服务？

将Consul服务对应的集群添加到数据面之后，可以采用两种方式进行调用：

- 启用DNS代理后通过服务条目中的hosts名称和端口调用。

i. 登录ASM控制台。

ii. 在左侧导航栏，选择**服务网格 > 网格管理**。

iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

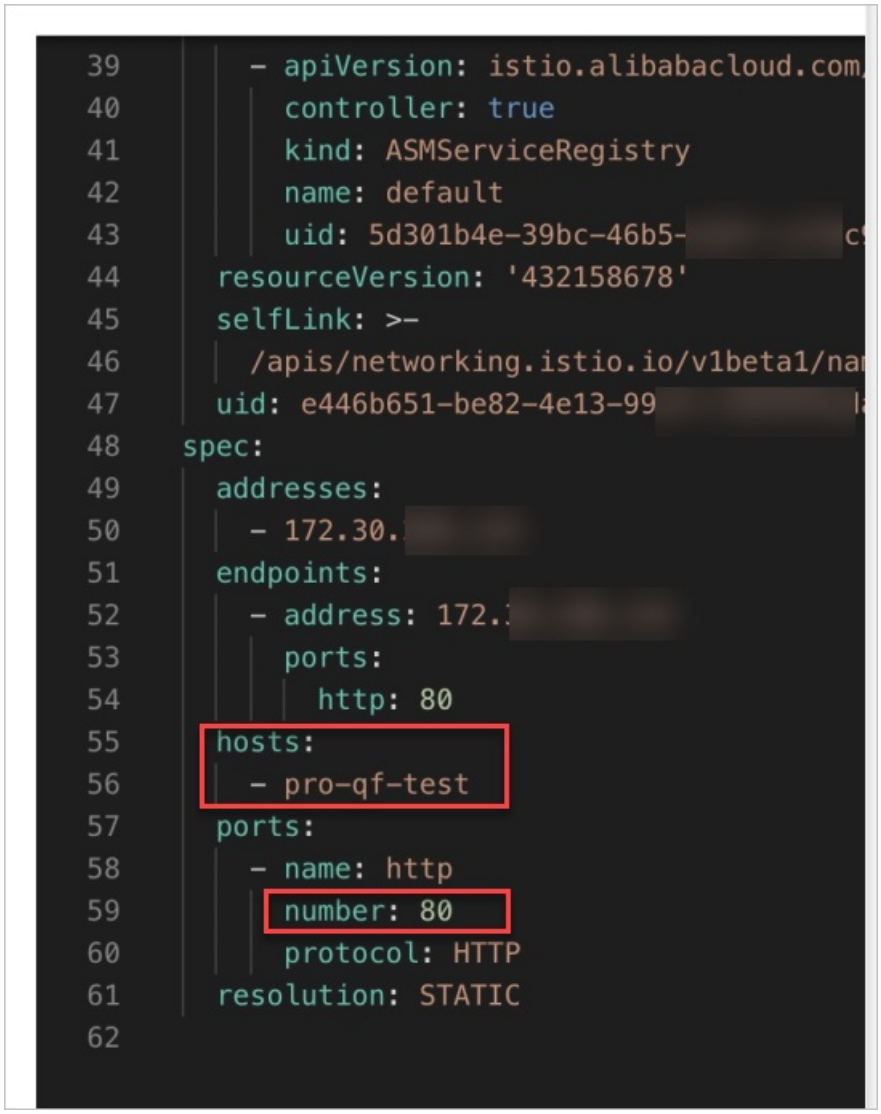
iv. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**，然后在右侧页面单击**功能设置**。

v. 在**功能设置更新**面板选中**启用DNS代理功能**，然后单击**确定**。

vi. 在网格详情页面左侧导航栏选择**集群与工作负载管理 > 服务条目**。

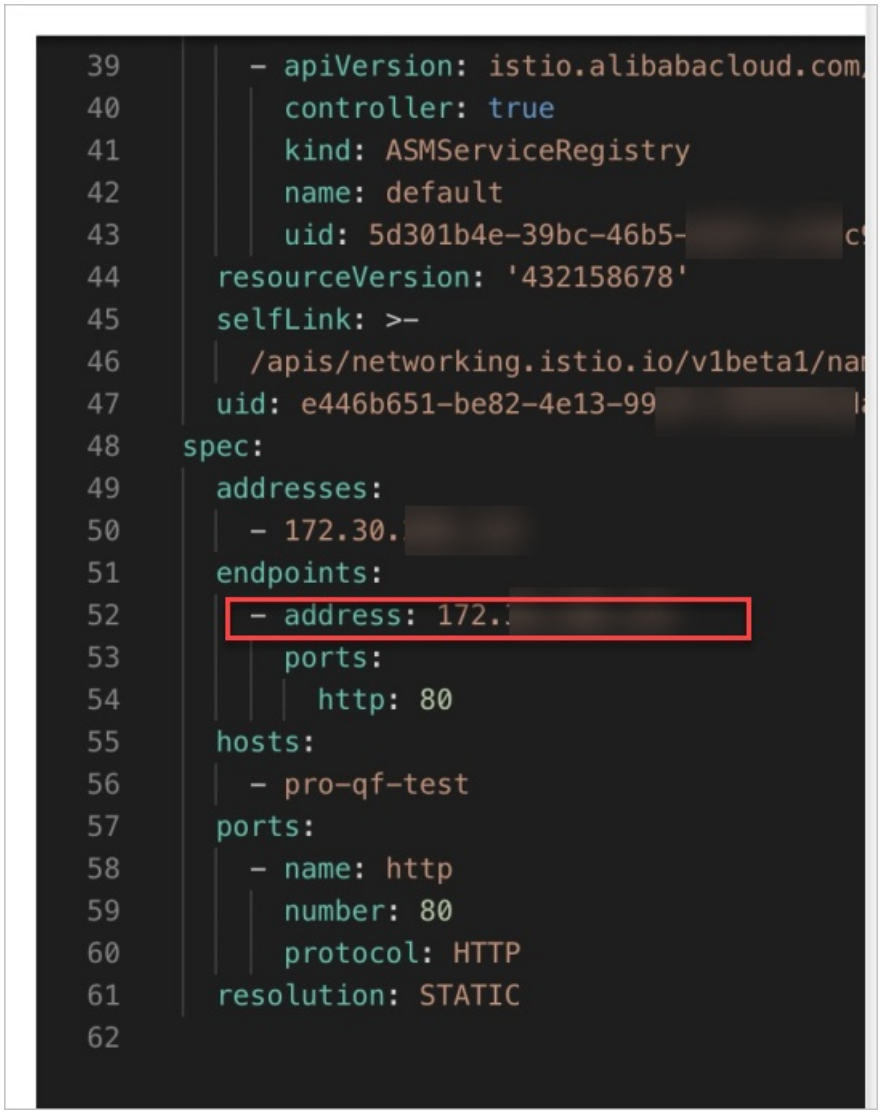
vii. 在**服务条目**页面单击目标服务条目**操作**列下的**YAML**。

在**编辑**面板获取hosts名称和端口，使用*http://<hosts名称>:<端口>*调用服务。



- 通过**服务条目**中的addresses地址调用。
  i. 登录ASM控制台。
  ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
  iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
  iv. 在网格详情页面左侧导航栏选择**集群与工作负载管理 > 服务条目**。
  v. 在**服务条目**页面单击目标服务条目**操作**列下的**YAML**。

在**编辑**面板获取addresses地址，使用*http://<addresses地址>*调用服务。



如何同步服务？

Consul中注册的服务会自动同步为istio中的服务条目，删除或更新服务，都会自动同步到istio中的服务条目。

如何在ASM中停用对接Consul注册中心功能？

执行以下命令，删除用于Consul同步服务注册信息的资源。

```
helm uninstall se-syncer
```

4.2. 对接Nacos注册中心

服务网格ASM提供了对接Nacos注册中心功能，便于将Nacos上的微服务与服务网格ASM进行互通。本文介绍如何对接Nacos注册中心。

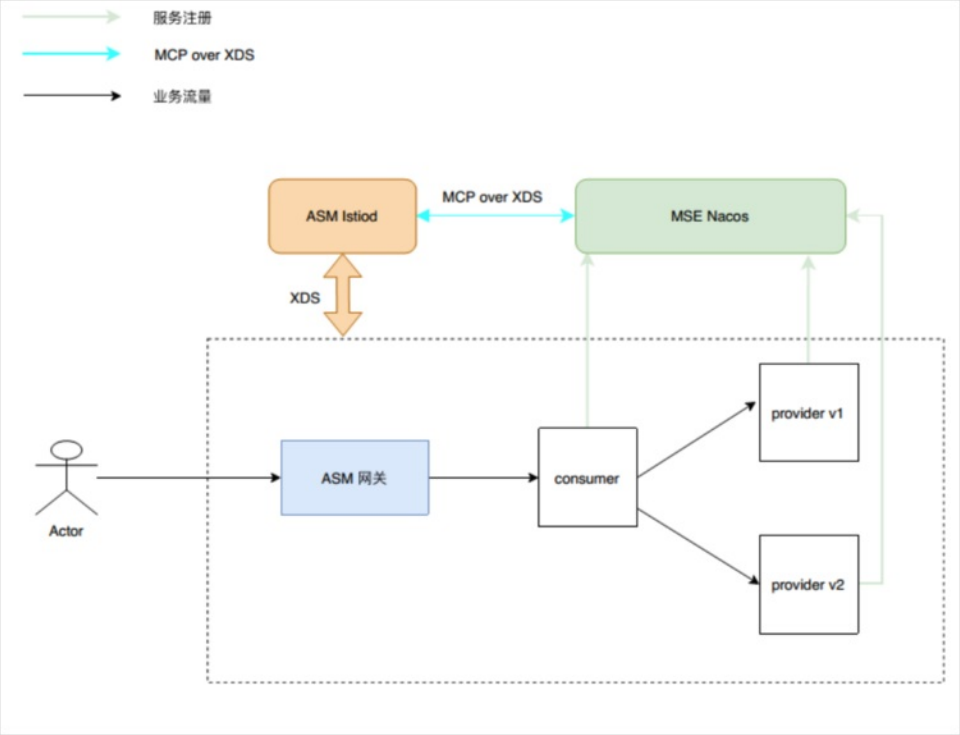
前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。

- 已创建Nacos引擎，且Nacos引擎的专有网络需要和ASM实例相同。具体操作，请参见创建Nacos引擎。

背景信息

服务网格ASM提供了对接Nacos注册中心的功能，便于您将微服务迁移至服务网格ASM的过程中，实现服务网格之内的服务调用服务网格之外的服务。服务网格ASM并不会主动将服务网格中的服务信息注册到Nacos注册中心。为了确保服务网格之外的服务能调用服务网格之内的服务，您需要将网格中的服务手动注册到Nacos注册中心。



网格内外服务互相调用方式

如下图所示，Consumer、Provider服务采用Nacos服务注册中心，注册和订阅了相关服务。Consumer和Provider服务调用方式如下：

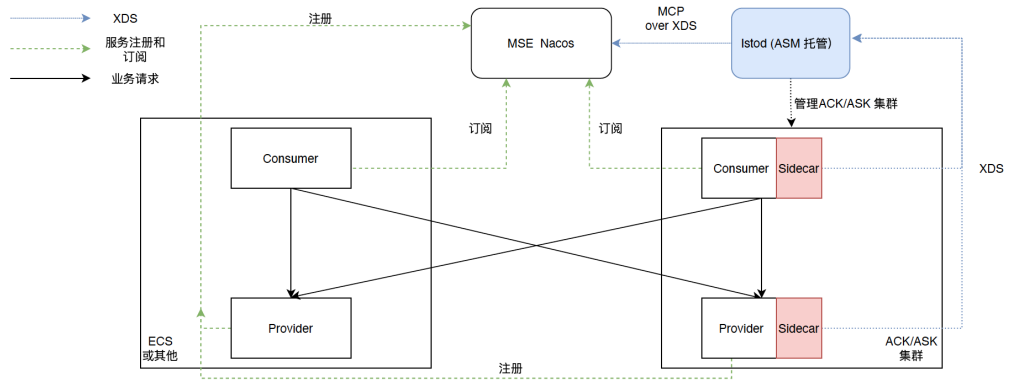
> **注意** 网格内的Consumer是否可以使用Istio下的全部功能，取决于Consumer和Provider实际业务请求的协议，目前ASM支持HTTP1.1、HTTP2、gRPC、Dubbo协议，不在这些协议范围内的业务请求，Istio提供的服务网格相关功能会受限。

- Consumer可以通过订阅从Nacos获取Provider服务地址，向Provider发起请求。

  无论Provider是部署在哪里，也无论Provider服务是否开启了Sidecar，但前提是需要保证本身网络层面是可以互相访问的，若网络层面不可直接访问或者有安全机制，可能需要通过部署ASM网关来实现业务互通。

- 网格内的Consumer（右边）也可以通过Sidecar调用左边的Provider服务。

  Sidecar可以通过xDS的CDS和EDS获取集群外的Provider服务地址，因为Istiod已经对接了Nacos，同步了Nacos下的所有服务信息。



步骤一：在MSE开启MCP功能

1. 登录MSE管理控制台。
2. 在控制台左侧导航栏选择**注册配置中心 > 实例列表**。
3. 在**实例列表**页面单击目标实例右侧**操作**列的**管理**。
4. 在实例详情页面左侧导航栏单击**参数设置**。
5. 在**参数设置**页面单击**编辑**，在MCPEnabled参数值下方选择**是**，然后单击**保存并重启实例**。

步骤二：在ASM对接Nacos注册中心

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**，然后在右侧页面单击**功能设置**。
5. 在**功能设置更新**对话框中单击**展开高级选项**，选中**Nacos注册服务**，选择Nacos引擎，单击**确定**。

5.生态集成

5.1. ASM集成ArgoCD实现GitOps

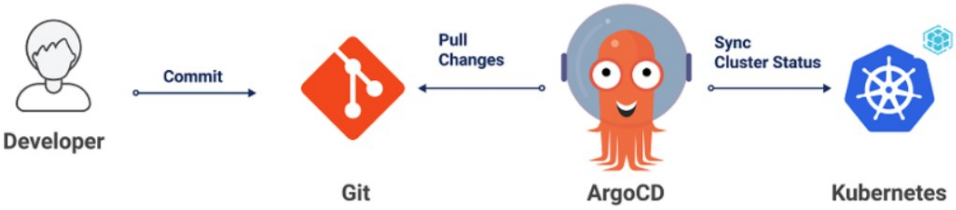
ArgoCD主要用于监听Git仓库中应用编排的变化，与集群中应用真实运行状态进行对比，自动或手动同步拉取应用编排的变更到部署集群中。阿里云服务网格ASM中集成ArgoCD，进行应用程序的发布和更新，降低运维成本。本文介绍如何通过ArgoCD实现GitOps。

前提条件

- 已创建Istio版本≥1.12.4.50的ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已创建Git仓库。

背景信息

GitOps是云原生应用程序实现持续部署的一种方式。阿里云服务网格ASM集成ArgoCD进行应用程序的发布和更新，实现GitOps。开发者提交YAML编写的应用程序定义（Deployment、Service）和流量管理（VirtualService、Gateway、DestinationRule）到Git仓库。ArgoCD监控集群中应用程序当前的Deployment、Service、VirtualService等资源的状态，与Git仓库中的资源期望编排进行比较，以Git仓库中的内容为基准，当Git仓库发生变更时，支持自动或手动同步和部署应用程序。



步骤一：安装ArgoCD

您可以选择手动安装ArgoCD或使用阿里云容器服务ACK应用中心内置的ArgoCD功能。下文以手动安装的ArgoCD与ASM集成为例，实现GitOps。

- 手动安装ArgoCD，请参见ArgoCD。
- 容器服务ACK应用中心具有内置的ArgoCD，可以避免您手动安装ArgoCD。ACK应用中心允许检查应用程序状态，可以使用Git仓库和Helm Chart将应用程序版本部署到Kubernetes集群，进行回滚和发布应用程序版本。更多信息，请参见应用中心概述。

步骤二：启用ASM的数据面KubeAPI访问能力

由于阿里云服务网格ASM是一个托管Istio兼容的控制平面，ArgoCD管理的服务网格ASM控制平面和数据平面（容器服务ACK）不在同一个Kubernetes集群环境中，因此需要在ASM中启用数据平面KubeAPI访问，ArgoCD才能像访问ACK集群的资源一样访问ASM集群中的Istio资源。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在**基本信息**区域的**启用数据面KubeAPI访问**右侧，单击**启用**。

5. 在弹出框中，单击**确认**。

步骤三：创建ASM网关

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**，然后在右侧页面单击**创建**。
5. 设置网关的基本信息，然后单击**创建**。

   部分配置项说明如下，关于配置项的更多信息，请参见添加入口网关服务。

参数	说明
名称	本文以*ingressgateway*为例。
网关类型	选择**南北向-入口**网关类型。
端口映射	单击**添加端口**，配置如下协议和端口。 ○ 配置**协议**为**HTTP**，**服务端口**为**80**。 ○ 配置**协议**为**HTTPS**，**服务端口**为**443**。

步骤四：通过ArgoCD部署Istio资源

Istio资源可以定义为Kubernetes清单，并推送到用于部署应用程序K8s编排的Git仓库中。

1. 创建bookinfo应用示例。

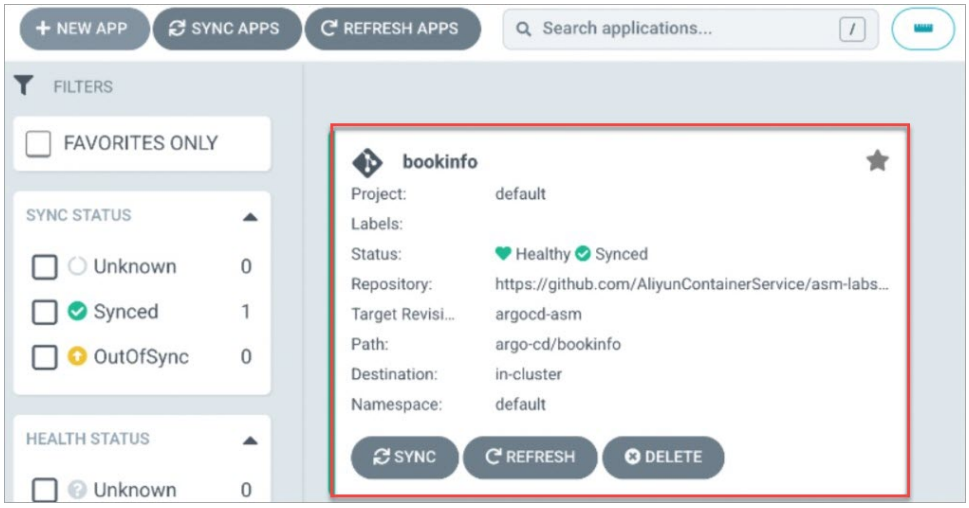
i. 在ArgoCD管理界面，单击**NEW APP**，进行如下配置。

- 在**GENERAL**区域，配置**Application**为*bookinfo*，**Project**为**default**，选中**PRUNE RESOURCES**。

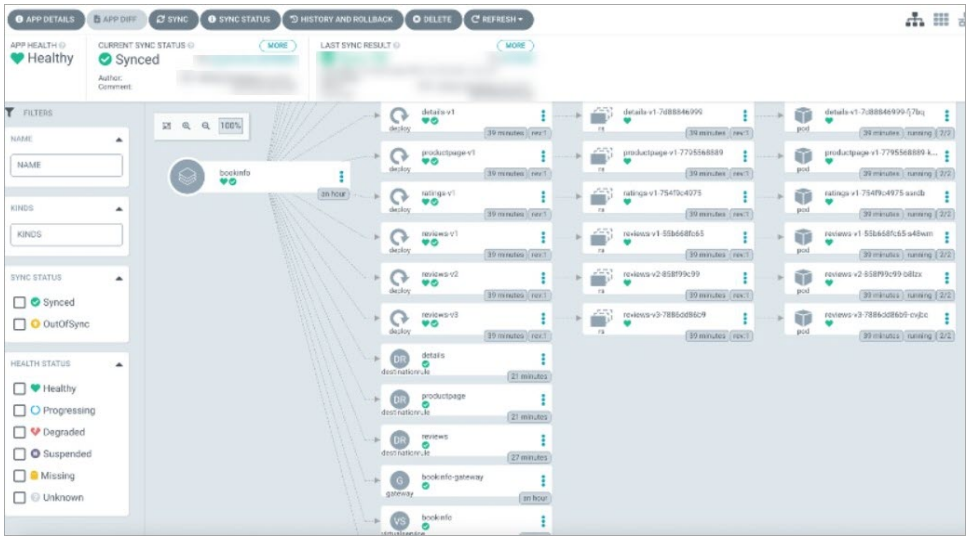
- 在**SOURCE**区域，配置**Repository URL**为*https://github.com/AliyunContainerService/asm-labs.git*，**Revision**为**argocd-asm**，**Path**为*argo-cd/bookinfo*。

- 在**DESTINATION**区域，配置**Cluster URL**为*https://kubernetes.default.svc*，**Namespace**为**default**。

ii. 配置完成，单击页面上方的**CREATE**。
创建完成后，在ArgoCD管理界面，即可查看bookinfo应用状态。



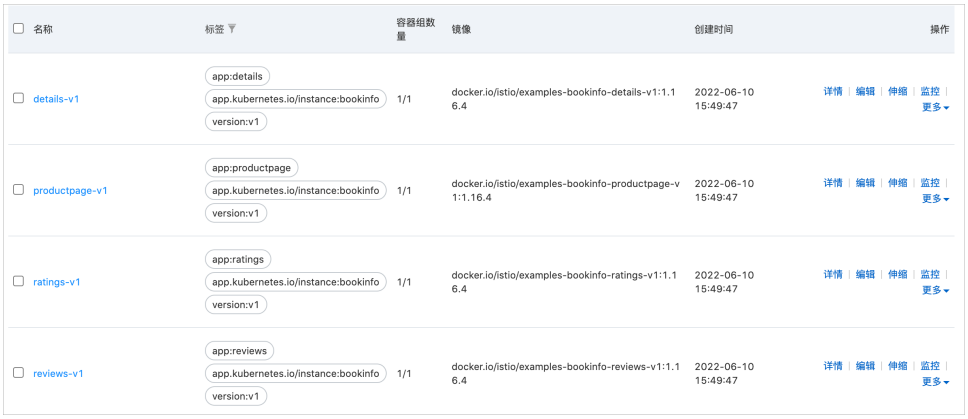
单击**bookinfo**，即可查看创建的资源状态。



2. 在ASM控制台查看资源状态。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
   iv. 在网格详情页面左侧导航栏选择**流量管理 > 虚拟服务**。
   v. 在**虚拟服务**页面，查看创建的bookinfo。

3. 在容器服务控制台查看资源状态。
   i. 登录容器服务管理控制台，在左侧导航栏中选择**集群**。
   ii. 在**集群列表**页面中，单击目标集群名称，然后在左侧导航栏中，选择**工作负载 > 无状态**。

iii. 查看创建的Deployment。



4. 访问ASM网关。

i. 获取ASM网关地址。

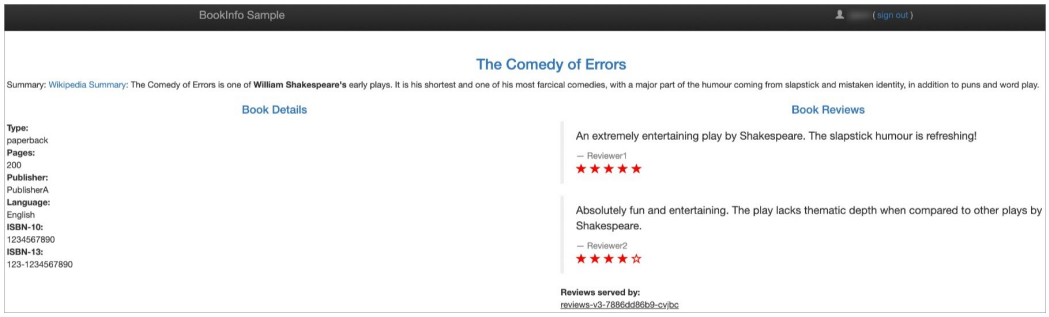
a. 登录ASM控制台。

b. 在左侧导航栏，选择**服务网格 > 网格管理**。

c. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

d. 在左侧导航栏单击**ASM网关**，然后在**Kubernetes服务**列下获取目标网关的地址。

ii. 在浏览器访问*http://{您的ASM网关地址}/productpage*。
访问效果如下所示，由于Istio bookinfo的Reviews有3个版本，目前未指定版本，因此每次刷新，右侧数据显示不同。



步骤五：部署GitOps

配置流量规则，访问*http://{您的ASM网关地址}/productpage*，未登录时只显示v1版本的Reviews，使用jason为用户名登录时，显示v2版本的Reviews。

1. 修改本地VirtualService和Deployment YAML文件。
   - VirtualService YAML修改如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v1
---
```

- Deployment YAML修改如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: argocd
  name: reviews-v1
  labels:
    app: reviews
    version: v1
spec:
  replicas: 2
```

2. 执行以下命令，将Reviews-v1添加到Git仓库。

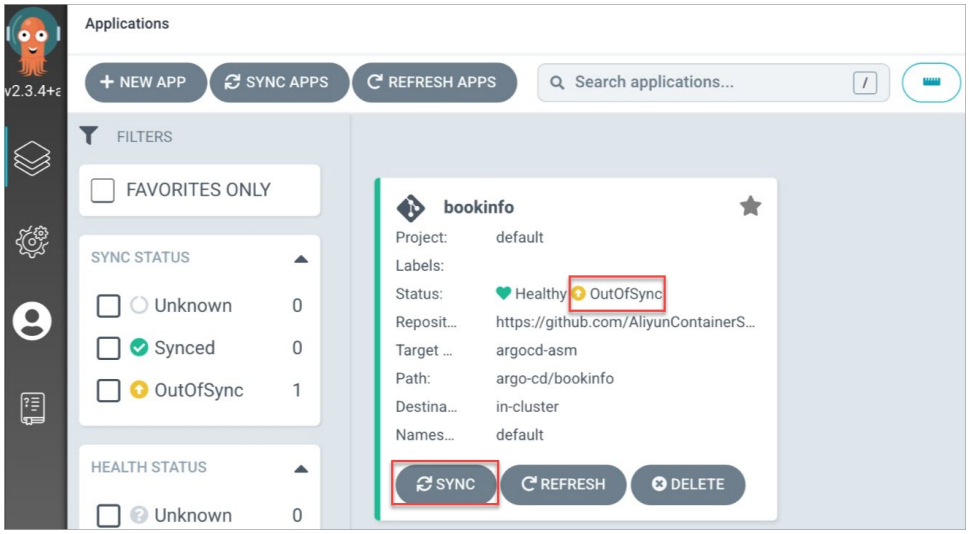
```
git add *
git commit -m "reviews-v1"
git push
```

3. 同步Git配置到集群。

   如果您开启了ArgoCD的自动同步功能，会自动同步Git的配置到集群，如果没有，您可以手动进行同步操作，具体步骤如下：

i. 在ArgoCD管理界面的**bookinfo**卡片中，单击**SYNC**。

当文件发生变更时，在**bookinfo**卡片的**Status**右侧会出现**Out Of Sync**状态提示。



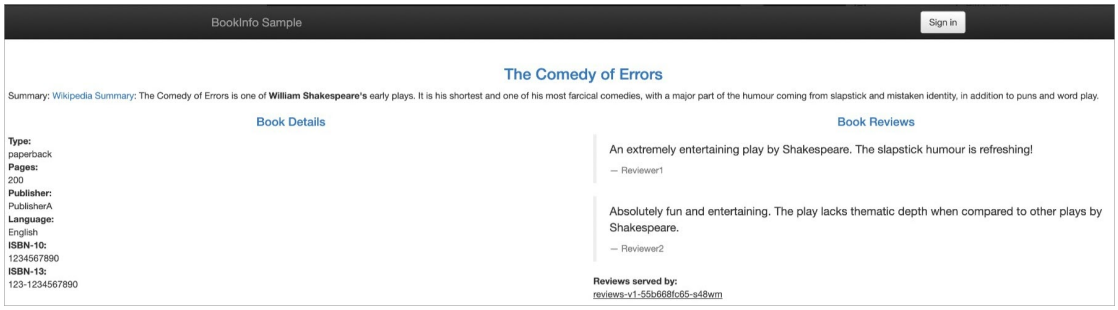
ii. 在弹出的面板上方，单击**SYNCHRONIZE**。

同步完成后，您可以在ASM控制台查看同步后创建的资源；在ACK控制台查看更新后的资源。具体操作，请参见在ASM控制台查看资源状态和在容器服务控制台查看资源状态。

4. 验证访问效果。

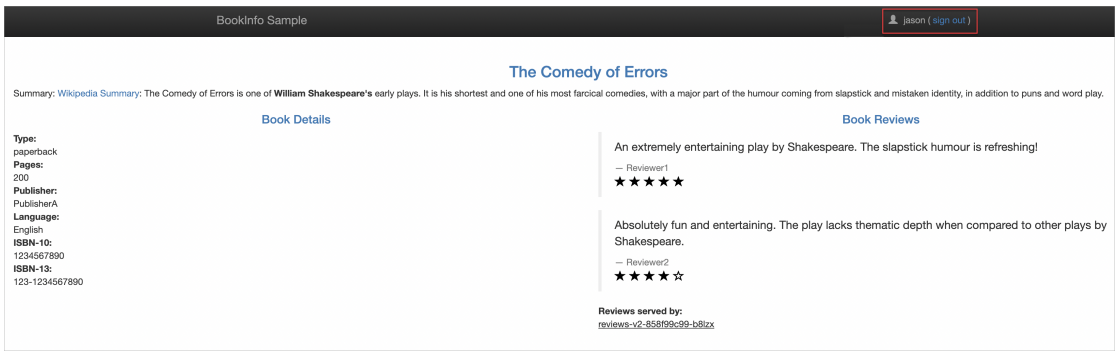
i. 未登录状态下，在浏览器访问*http://{您的ASM网关地址}/productpage*。

访问效果如下所示，固定为v1版本的Reviews。



ii. 使用jason为用户名，任意密码进行登录。

访问效果如下所示，固定为v2版本的Reviews。



5.2. ASM集成Argo Rollouts实现金丝雀发布

Argo Rollouts是Kubernetes控制器和CRD集合，阿里云服务网格ASM集成Argo Rollouts，提供更强大的金丝雀部署能力。本文介绍如何通过Argo Rollouts实现金丝雀发布。

前提条件

- 已创建Istio版本≥1.12.4.50的ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已通过kubectl连接ASM实例。具体操作，请参见通过kubectl连接ASM实例。

准备工作

安装Argo Rollout

安装Argo Rollout的操作步骤如下，更多信息，请参见Argo Rollouts。

1. 执行以下命令，安装Argo Rollout服务端。

```
kubectl create namespace argo-rollouts
kubectl apply -n argo-rollouts -f https://github.com/argoproj/argo-rollouts/releases/latest/download/install.yaml
```

2. 执行以下命令，安装Kubectl Argo Rollout插件。

   安装Kubectl Argo Rollout插件，方便您通过kubectl进行管理。

```
brew install argoproj/tap/kubectl-argo-rollouts
```

启用ASM的数据面KubeAPI访问能力

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**。
5. 在**启用数据面KubeAPI访问**右侧，单击**启用**。

6. 在弹出框中，单击**确认**。

金丝雀发布

本文以创建稳定版本和金丝雀版本为例，逐步将流量切换为金丝雀版本，实现基于流量比例的发布策略。关于金丝雀发布的更多信息，请参见金丝雀发布。

步骤一：创建Rollout和Service应用

1. 创建Rollout。

   i. 使用以下内容，创建*rollout.yaml*文件。

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: istio-rollout
spec:
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: istio-rollout
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true"
      labels:
        app: istio-rollout
    spec:
      containers:
      - name: istio-rollout
        image: argoproj/rollouts-demo:blue
        ports:
        - name: http
          containerPort: 8080
          protocol: TCP
        resources:
          requests:
            memory: 32Mi
            cpu: 5m
  strategy:
    canary:
      canaryService: istio-rollout-canary
      stableService: istio-rollout-stable
      trafficRouting:
        istio:
          virtualService:
            name: istio-rollout-vsvc
            routes:
            - primary
      steps:
      - setWeight: 10
      - pause: {}         #手工卡点。
      - setWeight: 20
      - pause: {duration: 20s}
      - setWeight: 30
      - pause: {duration: 20s}
      - setWeight: 40
      - pause: {duration: 20s}
      - setWeight: 50
      - pause: {duration: 20s}
      - setWeight: 60
      - pause: {duration: 20s}
      - setWeight: 70
```

```
      - pause: {duration: 20s}
      - setWeight: 80
      - pause: {duration: 20s}
      - setWeight: 90
      - pause: {duration: 20s}
```

`strategy` 字段定义发布策略，部分参数说明如下：

- `setWeight` ：设置流量的权重。
- `pause` ：若未配置 `duration` ，表示需要手动更新；配置 `duration` ，表示等待 `duration` 时间进行自动更新。

ii. 执行以下命令，将Rollout部署到集群。

```
kubectl apply -f rollout.yaml
```

2. 创建Service。

i. 使用以下内容，创建*service.yaml*文件。

```
apiVersion: v1
kind: Service
metadata:
  name: istio-rollout-canary
spec:
  ports:
  - port: 80
    targetPort: http
    protocol: TCP
    name: http
  selector:
    app: istio-rollout
---
apiVersion: v1
kind: Service
metadata:
  name: istio-rollout-stable
spec:
  ports:
  - port: 80
    targetPort: http
    protocol: TCP
    name: http
  selector:
    app: istio-rollout
```

ii. 执行以下命令，将Service部署到集群。

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

步骤二：创建Istio相关资源

1. 创建虚拟服务VirtualService。

由于启用了ASM的数据面KubeAPI访问能力，您可以通过数据面的KubeConfig访问ASM中的VirtualService、Gateway、DestinationRule等Istio资源，也可以通过ASM控制台或者ASM KubeConfig创建Istio资源。

i. 使用以下内容，创建*istio-rollout-vsvc.yaml*文件。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: istio-rollout-vsvc
spec:
  gateways:
    - istio-rollout-gateway
  hosts:
    - '*'
  http:
    - match:
        - uri:
            prefix: /
      name: primary
      route:
        - destination:
            host: istio-rollout-stable
          weight: 100
        - destination:
            host: istio-rollout-canary
```

ii. 执行以下命令，部署istio-rollout-vsvc。

```
kubectl apply -f istio-rollout-vsvc.yaml
```

2. 创建网关规则Gateway。

i. 使用以下内容，创建*istio-rollout-gateway.yaml*文件。

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: istio-rollout-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
    - hosts:
        - '*'
      port:
        name: http
        number: 80
        protocol: HTTP
```

ii. 执行以下命令，部署istio-rollout-gateway。

```
kubectl apply -f istio-rollout-gateway.yaml
```

步骤三：创建ASM网关

创建端口为80的ASM网关，作为测试访问的入口。

1. 登录ASM控制台。

2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**，然后在右侧页面单击**创建**。
5. 设置网关的基本信息，然后单击**创建**。

   部分配置项说明如下，关于配置项的更多信息，请参见添加入口网关服务。

参数	说明
名称	本文以*ingressgateway*为例。
网关类型	选择**南北向-入口**网关类型。
端口映射	单击**添加端口**，在新增端口行中，配置**协议**为**HTTP**，**服务端口**为**80**。

步骤四：查看Rollout状态

执行以下命令，查看Rollout的状态。

```
kubectl argo rollouts get rollout istio-rollout
```

预期输出：

```
kubectl argo rollouts get rollout istio-rollout
Name:            istio-rollout
Namespace:       default
Status:          ✔ Healthy
Strategy:        Canary
  Step:          18/18
  SetWeight:     100
  ActualWeight:  100
Images:          argoproj/rollouts-demo:blue (stable)
Replicas:
  Desired:       1
  Current:       1
  Updated:       1
  Ready:         1
  Available:     1
NAME                                     KIND        STATUS     AGE  INFO
⟳ istio-rollout                          Rollout     ✔ Healthy  52s
└──# revision:1
   └──□ istio-rollout-7f96d86486         ReplicaSet  ✔ Healthy  52s  stable
      └──□ istio-rollout-7f96d86486-vpqvb  Pod         ✔ Running  52s  ready:2/2
```

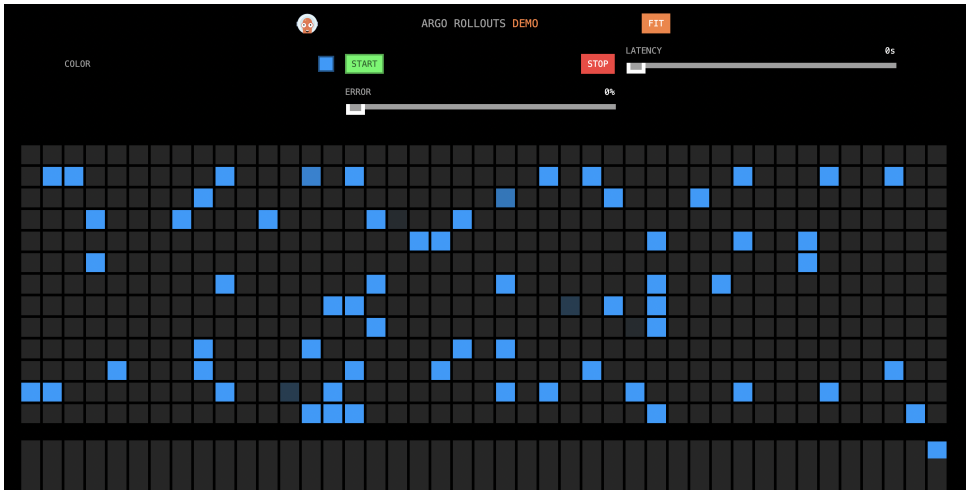
步骤五：测试Rollout初始状态

1. 获取ASM网关的IP地址。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。

iv. 在网格详情页面左侧导航栏单击**ASM网关**。

v. 在目标网关的**Kubernetes服务**列下获取网关地址。

2. 在浏览器访问*http://{ASM网关IP}/*。
访问效果如下所示，此界面会并发调用*http://{ASM网关IP}/color*，将获取到颜色信息填充到方格中。
在Rollout istio-rollout中，指定的颜色为`blue`，并且未进行金丝雀发布，因此显示颜色为蓝色。



步骤六：滚动更新

本文以黄色代表金丝雀版本，将*http://{ASM网关IP}/*网页中的方格颜色从蓝色逐渐变为黄色，实现金丝雀发布。

1. 更新镜像版本。

i. 执行以下命令，更新镜像版本。

```
kubectl argo rollouts set image istio-rollout "*=argoproj/istio-rollout:yellow"
```

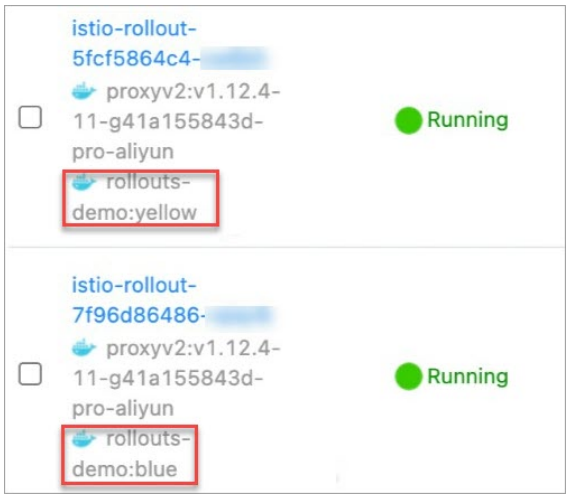
ii. 查看容器的镜像版本。

a. 登录容器服务管理控制台，在左侧导航栏中单击**集群**。

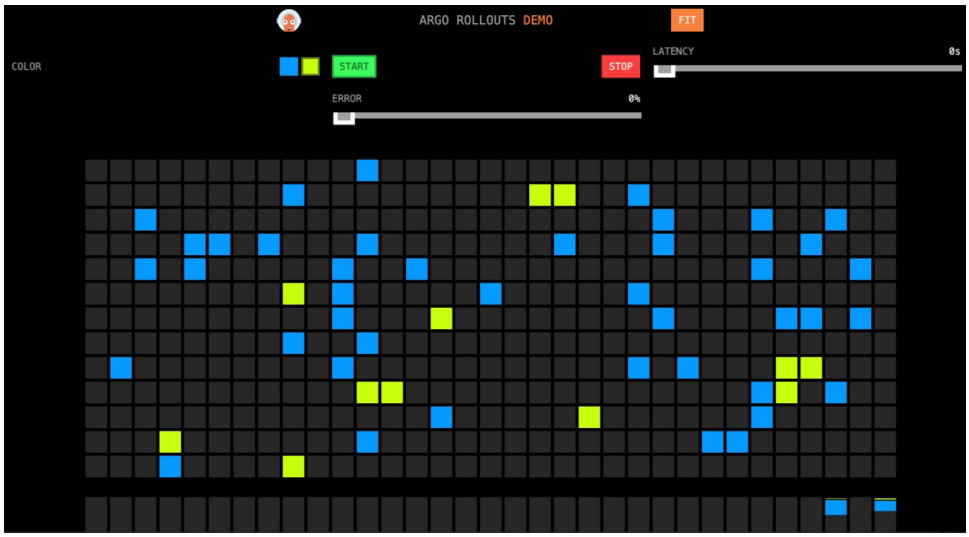
b. 在**集群列表**页面中，单击目标集群名称，然后在左侧导航栏中，选择**工作负载 > 容器组**。

c. 在**名称**列下，查看目标容器的镜像版本。

如下图所示，Yellow版本（金丝雀版本）的Pod已被创建，但Blue版本（稳定版）的Pod依然存在。



2. 在浏览器访问*http://{ASM网关IP}/*。
访问效果如下所示，有10%的方格被变成了黄色。



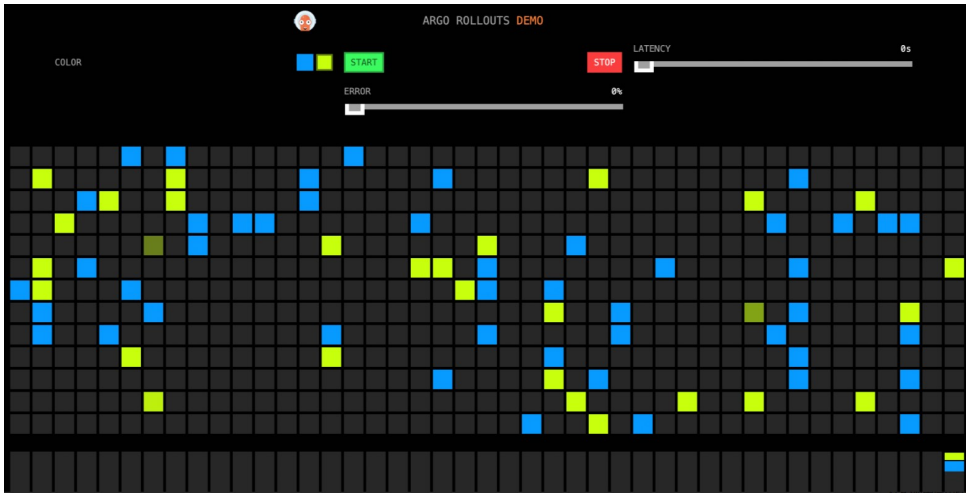
在服务网格ASM中，由于配置的虚拟服务VirtualService的权重发生了变化，Stable（蓝色）的权重从100变为90，Sanary（黄色，即金丝雀版本），权重从0变为10。虚拟服务VirtualService的权重由Rollout控制，开始创建Rollout的第一个Step的 `setWeight` 中设置了金丝雀版本的权重为 `10` ，在开始滚动时，Argo Rollout的控制器会修改Rollout中配置的VirtualService的权重。 `pause` 设置为空，表示需要人工卡点确认，才进到下一阶段。

3. 继续金丝雀发布。

i. 执行以下命令，继续金丝雀发布。

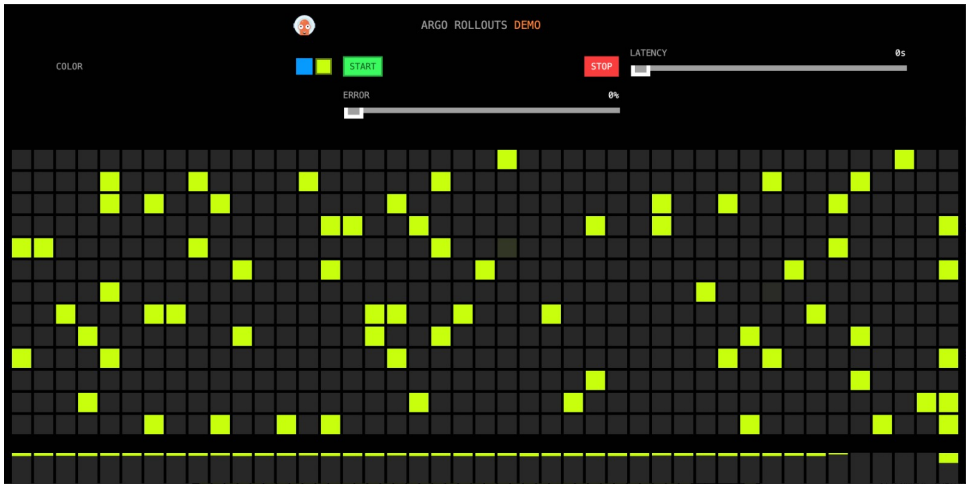
```
kubectl argo rollouts promote istio-rollout
```

ii. 在浏览器访问*http://{ASM网关IP}/*。
访问效果如下所示，VirtualService中的权重会根据Rollout中的配置继续调整。由于步骤1中设置了`pause`时间，会等待一段时间后自动调整。



4. 金丝雀发布成功。

i. 等待一段时间后，在浏览器访问*http://{ASM网关IP}/*，查看页面效果。
访问效果如下所示，所有的颜色块都变为了黄色。



ii. 执行以下命令，查看Rollout状态。

```
kubectl argo rollouts get rollout istio-rollout --watch
```

预期输出：

```
Name:            istio-rollout
Namespace:       default
Status:          ✔ Healthy
Strategy:        Canary
  Step:          18/18
  SetWeight:     100
  ActualWeight:  100
Images:          argoproj/rollouts-demo:yellow (stable)
Replicas:
  Desired:       1
  Current:       1
  Updated:       1
  Ready:         1
  Available:     1
NAME                                     KIND        STATUS         AGE  INFO
⟳ istio-rollout                          Rollout     ✔ Healthy      48m
├──# revision:4
│  └──□ istio-rollout-5fcf5864c4         ReplicaSet  ✔ Healthy      27m  stable
│     └──□ istio-rollout-5fcf5864c4-vw6kh  Pod       ✔ Running      26m  ready:2/2
├──# revision:3
│  └──□ istio-rollout-897cb5b6d           ReplicaSet  • ScaledDown   27m
└──# revision:1
   └──□ istio-rollout-7f96d86486         ReplicaSet  • ScaledDown   48m
```

由预期输出得到，Stabel的Image更新为 `yellow` 。

使用Prometheus实现自动回滚

在金丝雀过程中可以手动执行 `kubectl argo rollouts abort istio-rollout` 命令进行回滚，回退到Stable版本。您还可以使用Prometheus监控系统，对金丝雀中的应用健康状态进行监控，当监控指标异常时，自动回滚到Stable版本，并标记为降级。

1. 在ASM中启用Prometheus。具体操作，请参见集成ARMS Prometheus实现网格监控或集成自建Prometheus实现网格监控。
2. 配置Argo AnalysisTemplate。

i. 使用以下YAML内容，创建*istio-success-rate.yaml*文件。

配置AnalysisTemplate的 `address` 为ASM的Prometheus地址。

```
apiVersion: argoproj.io/v1alpha1
kind: AnalysisTemplate
metadata:
  name: istio-success-rate
spec:
  args:
  - name: service
  - name: namespace
  metrics:
  - name: success-rate
    initialDelay: 60s
    interval: 20s
    successCondition: result[0] > 0.90
    provider:
      prometheus:
        address: http://xxx.aliyuncs.com:9090/api/v1/prometheus/
        query: >+
          sum(irate(istio_requests_total{
            reporter="source",
            destination_service=~"{{args.service}}.{{args.namespace}}.svc.cluster.l
ocal",
            response_code!~"5.*"}[40s])
          )
          /
          sum(irate(istio_requests_total{
            reporter="source",
            destination_service=~"{{args.service}}.{{args.namespace}}.svc.cluster.l
ocal"}[40s])
          )
```

ii. 执行以下命令，部署Argo AnalysisTemplate。

```
kubectl apply -f istio-success-rate.yaml
```

3. 为Rollout关联Analysis。

i. 使用以下YAML内容，创建*rollout.yaml*文件。

在 `strategy` 中配置 `analysis` ，从第二步开启使用Analysis进行监控，自动回滚。初始Image为Yellow（黄色）。

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: istio-rollout
spec:
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: istio-rollout
  template:
    metadata:
      annotations:
```

```
        sidecar.istio.io/inject: "true"
      labels:
        app: istio-rollout
    spec:
      containers:
      - name: istio-rollout
        image: argoproj/rollouts-demo:yellow
        ports:
        - name: http
          containerPort: 8080
          protocol: TCP
        resources:
          requests:
            memory: 32Mi
            cpu: 5m
  strategy:
    canary:
      canaryService: istio-rollout-canary
      stableService: istio-rollout-stable
      analysis:
        startingStep: 1
        templates:
        - templateName: istio-success-rate
        args:
        - name: service
          value: canary
        - name: namespace
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      trafficRouting:
        istio:
          virtualService:
            name: istio-rollout-vsvc
            routes:
            - primary
      steps:
      - setWeight: 10
      - pause: {}         #手工卡点。
      - setWeight: 20
      - pause: {duration: 20s}
      - setWeight: 30
      - pause: {duration: 20s}
      - setWeight: 40
      - pause: {duration: 20s}
      - setWeight: 50
      - pause: {duration: 20s}
      - setWeight: 60
      - pause: {duration: 20s}
      - setWeight: 70
      - pause: {duration: 20s}
      - setWeight: 80
      - pause: {duration: 20s}
      - setWeight: 90
```

```
      - pause: {duration: 20s}
```

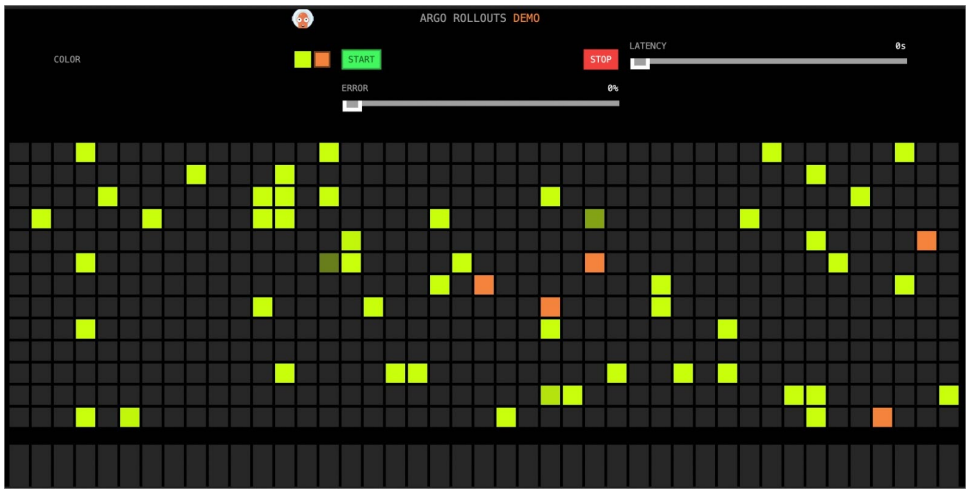
ii. 执行以下命令，更新Rollout。

```
kubectl apply -f rollout.yaml
```

4. 执行以下命令，更新镜像版本。

```
kubectl argo rollouts set image istio-rollout "*=argoproj/rollouts-demo:orange"
```

在浏览器访问*http://{ASM网关IP}/*，访问效果如下所示：



5. 人工卡点确认。

i. 执行以下命令，继续金丝雀发布。

执行命令后，将进入到后续的自动金丝雀状态，并且从第二步开始，会结合Prometheus监控，如果金丝雀版本错误率高于90%，则触发回滚。

```
kubectl argo rollouts promote istio-rollout
```

ii. 执行以下命令，查看监控服务状态。

```
kubectl argo rollouts get rollout istio-rollout --watch
```

预期输出：

```
Name:            istio-rollout
Namespace:       default
Status:          ◌ Progressing
Message:         waiting for all steps to complete
Strategy:        Canary
  Step:          5/18
  SetWeight:     30
  ActualWeight:  30
Images:          argoproj/rollouts-demo:orange (canary)
                 argoproj/rollouts-demo:yellow (stable)
Replicas:
  Desired:       1
  Current:       2
  Updated:       1
  Ready:         2
  Available:     2

NAME                                    KIND         STATUS        AGE    INFO
⟳ istio-rollout                         Rollout      ◌ Progressing 2m23s
├──# revision:2
│  ├──⧉ istio-rollout-5b7b8669fb        ReplicaSet   ✔ Healthy     107s   canary
│  │  └──□ istio-rollout-5b7b8669fb-c7w5q Pod         ✔ Running     107s   ready:2/2
│  └──α istio-rollout-5b7b8669fb-2      AnalysisRun  ◌ Running     78s    ⚠ 4
└──# revision:1
   └──⧉ istio-rollout-5fcf5864c4        ReplicaSet   ✔ Healthy     2m23s  stable
      └──□ istio-rollout-5fcf5864c4-z8l8n Pod         ✔ Running     2m23s  ready:2/2
```

6. 设置Error。

   在后续渐进式发布中，您可以手动设置Error，调整金丝雀版本的错误率。移动Error至100%后，所有的金丝雀版本（橙色）都有一个红框，表示错误。等待片刻后，将自动切换回只有黄色的版本（稳定版）。

   金丝雀中

   自动回滚到稳定版

5.3. ASM集成云原生推理服务框架KServe

KServe（原KFServing）是云原生环境的的一个模型服务器和推理引擎，支持自动缩放、零缩放、金丝雀部署等能力。本文介绍如何结合阿里云服务网格ASM和阿里云容器服务平台Kubernetes（ACK）来部署KServe。

前提条件

- 已创建Istio版本≥1.12.4.58的ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已通过kubectl连接ASM实例。具体操作，请参见通过kubectl连接ASM实例。

步骤一：添加数据面集群

1. 登录ASM控制台。

2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**集群与工作负载管理 > Kubernetes集群**。
5. 在**Kubernetes集群**页面单击**添加**，然后在**添加集群**面板选中目标集群，单击**确定**。

步骤二：启用数据面KubeAPI访问

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏选择**网格实例 > 基本信息**。
5. 在**启用数据面KubeAPI访问**右侧，单击**启用**。

6. 在弹出框中，单击**确认**。

步骤三：安装KServe组件

1. 安装Knative Serving。

   本文以安装Knative Serving v0.7版本，Kubernetes版本≥v1.17为例。

   i. 执行以下命令，安装Knative Serving所需的自定义资源。

   ```
   kubectl apply -f https://raw.githubusercontent.com/AliyunContainerService/asm-labs/kserve/kserve-0.7/serving-crds.yaml
   ```

   ii. 执行以下命令，安装Knative Serving的核心组件。

   ```
   kubectl apply -f https://raw.githubusercontent.com/AliyunContainerService/asm-labs/kserve/kserve-0.7/serving-core.yaml
   ```

   iii. 执行以下命令，安装Knative Istio Controller。

   在KServe中，可以使用Istio作为调用入口，并提供模型的蓝绿和金丝雀部署能力。本文安装net-istio-controller作为Istio的Knative入口控制器，并创建istio Gateway和PeerAuthentication资源。PeerAuthentication用于在服务网格环境中为Knative Webhook设置PERMISSIVE来避免mTLS认证问题。由于已经启用了数据面KubeAPI访问能力，可以直接使用数据面的Kubeconfig进行创建。

   ```
   kubectl apply -f https://raw.githubusercontent.com/AliyunContainerService/asm-labs/kserve/kserve-0.7/net-istio.yaml
   ```

2. 执行以下命令，安装Cert Manager。

   KServe依赖Cert Manager组件，建议安装当前最新稳定版本。本文以安装v1.8.0版本为例。

   ```
   kubectl apply -f  https://github.com/cert-manager/cert-manager/releases/download/v1.8.0/cert-manager.yaml
   ```

3. 执行以下命令，创建KServe。

```
kubectl apply -f https://raw.githubusercontent.com/AliyunContainerService/asm-labs/kserve/kserve-0.7/kserve.yaml
```

预期输出：

```
namespace/kserve created
customresourcedefinition.apiextensions.k8s.io/inferenceservices.serving.kserve.io created
customresourcedefinition.apiextensions.k8s.io/trainedmodels.serving.kserve.io created
role.rbac.authorization.k8s.io/leader-election-role created
clusterrole.rbac.authorization.k8s.io/kserve-manager-role created
clusterrole.rbac.authorization.k8s.io/kserve-proxy-role created
rolebinding.rbac.authorization.k8s.io/leader-election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/kserve-manager-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/kserve-proxy-rolebinding created
configmap/inferenceservice-config created
configmap/kserve-config created
secret/kserve-webhook-server-secret created
service/kserve-controller-manager-metrics-service created
service/kserve-controller-manager-service created
service/kserve-webhook-server-service created
statefulset.apps/kserve-controller-manager created
certificate.cert-manager.io/serving-cert created
issuer.cert-manager.io/selfsigned-issuer created
mutatingwebhookconfiguration.admissionregistration.k8s.io/inferenceservice.serving.kserve.io created
validatingwebhookconfiguration.admissionregistration.k8s.io/inferenceservice.serving.kserve.io created
validatingwebhookconfiguration.admissionregistration.k8s.io/trainedmodel.serving.kserve.io created
```

步骤四：创建ASM网关

创建协议为TCP、端口为80的ASM网关，作为测试访问的入口。

1. 登录ASM控制台。
2. 在左侧导航栏，选择**服务网格 > 网格管理**。
3. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
4. 在网格详情页面左侧导航栏单击**ASM网关**，然后在右侧页面单击**创建**。
5. 设置网关的基本信息，然后单击**创建**。

   部分配置项说明如下，关于配置项的更多信息，请参见添加入口网关服务。

参数	说明
名称	本文以*ingressgateway*为例。
网关类型	选择**南北向-入口**网关类型。
端口映射	单击**添加端口**，在新增端口行中，配置**协议**为TCP，**服务端口**为80。

步骤五：创建推理服务

1. 执行以下命令，创建用于部署KServe资源的命名空间。

```
kubectl create namespace kserve-test
```

2. 执行以下命令，创建InferenceService。

```
kubectl apply -n kserve-test -f - <<EOF
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "sklearn-iris"
spec:
  predictor:
    sklearn:
      storageUri: "https://github.com/tduffy000/kfserving-uri-examples/blob/master/sklearn/frozen/model.joblib?raw=true"
EOF
```

3. 执行以下命令，查询Inferenceservices的sklearn-iris的安装状态。

```
kubectl get inferenceservices sklearn-iris -n kserve-test
```

预期输出：

```
NAME           URL                                                 READY   PREV   LATEST   PREVROLLEDOUTREVISION   LATESTREADYREVISION                    AGE
sklearn-iris   http://sklearn-iris.kserve-test.example.com         True           100                              sklearn-iris-predictor-default-***     7d2**
```

安装完成后，会自动创建对应模型配置的虚拟服务和网关规则。您可以在ASM控制台网格详情页面左侧导航栏，单击**流量管理 > 虚拟服务**查看sklearn-iris虚拟服务，单击**流量管理 > 网关规则**查看创建的网关规则。

步骤六：通过ASM网关进行访问测试

1. 执行以下命令，获取 `SERVICE_HOSTNAME` 。

```
SERVICE_HOSTNAME=$(kubectl get inferenceservice sklearn-iris -n kserve-test -o jsonpath='{.status.url}' | cut -d "/" -f 3)
echo ${SERVICE_HOSTNAME}
```

预期输出：

```
sklearn-iris.kserve-test.example.com
```

由预期输出得到，本次测试的 `SERVICE_HOSTNAME` 为 `sklearn-iris.kserve-test.example.com` 。

2. 执行以下命令，访问ASM网关地址。

ASM网关地址由步骤四：创建ASM网关创建。

```
curl  -H "Host: ${SERVICE_HOSTNAME}" http://{ASM网关地址}:80/v1/models/sklearn-iris:predict -d @./iris-input.json
```

预期输出：

```
{"perdictions": [1,1]}
```

由预期输出得到，可以正常访问ASM网关。

6.Dubbo服务治理

6.1. 托管Dubbo服务

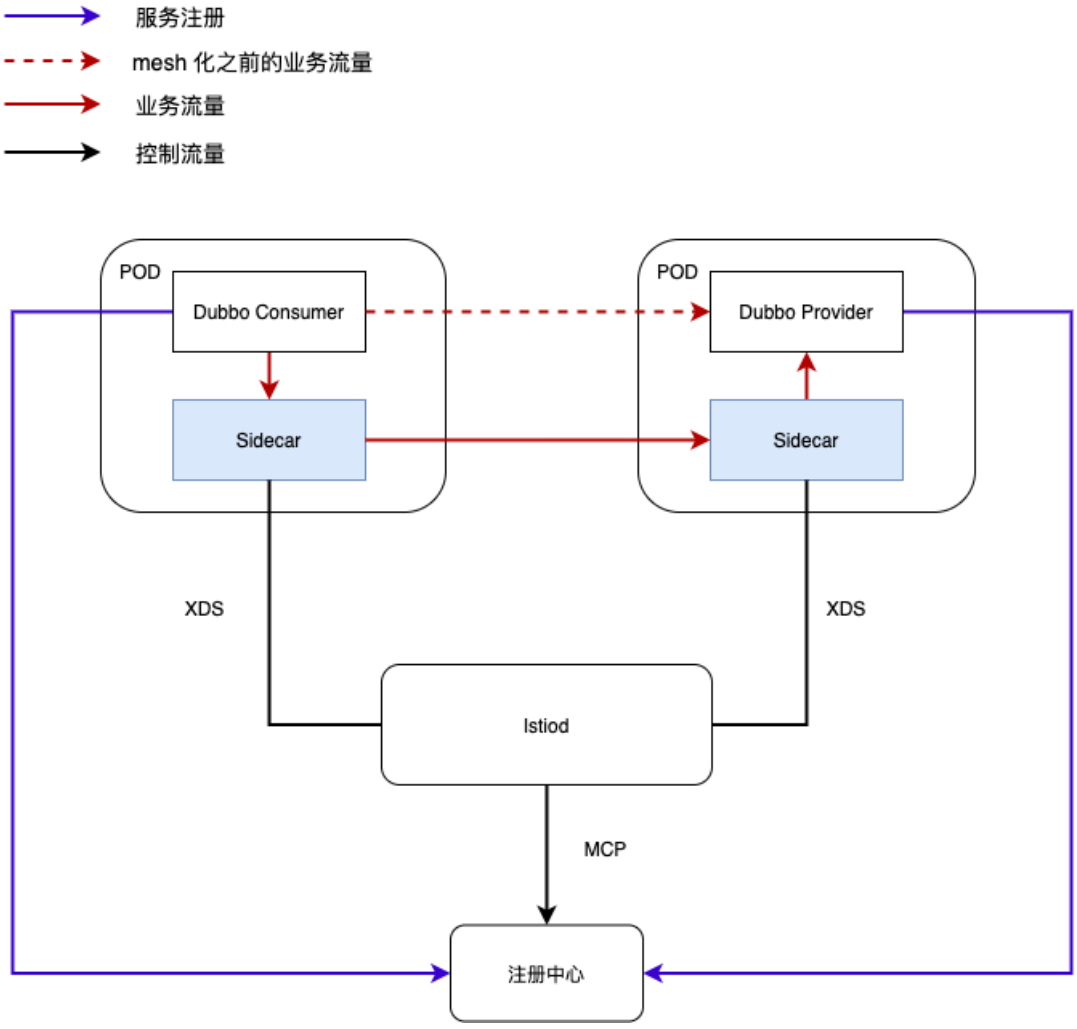
Dubbo是一个开源分布式框架，致力于提供高性能和透明化的远程服务调用方案。遵循Dubbo框架的服务即为Dubbo服务。您可以通过ASM商业版（专业版）托管Dubbo服务，从而可以对Dubbo服务进行流量管理。本文介绍如何通过ASM托管Dubbo服务。

前提条件

- 已创建ASM实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。

背景信息

以下是ASM托管Dubbo服务后的架构示意图，其中Istiod就是托管的ASM实例。注册中心可以采用Nacos、Zookeeper、Eureka等。 Istiod通过MCP over XDS协议对接注册中心，同步服务信息到Sidecar。MSE的Nacos原生支持MCP协议，您只需要在开通ASM实例时开启Nacos注册中心功能，推荐使用MSE的Nacos。



步骤一：创建服务注册中心

1. 创建1.2.1及以上版本的Nacos引擎。具体操作，请参见创建Nacos引擎。
2. 开启MCP功能。

   > **说明** 仅1.2.1及以上版本的Nacos引擎支持开启MCP功能。

   i. 登录MSE管理控制台。
   ii. 在左侧导航栏选择**注册配置中心 > 实例列表**。
   iii. 在**实例列表**页面单击具体实例名称。
   iv. 在**实例详情**页面左侧菜单栏选择**参数设置**。
   v. 单击**编辑**，在MCPEnabled参数**值**下方选择**是**，然后单击**保存并重启实例**。
3. 如果当前Dubbo应用服务采用其他注册中心，您可以通过迁移工具同步到MSE Nacos。具体操作，请参见将Dubbo应用的注册中心迁移到MSE。

步骤二：创建ASM商业版（专业版）

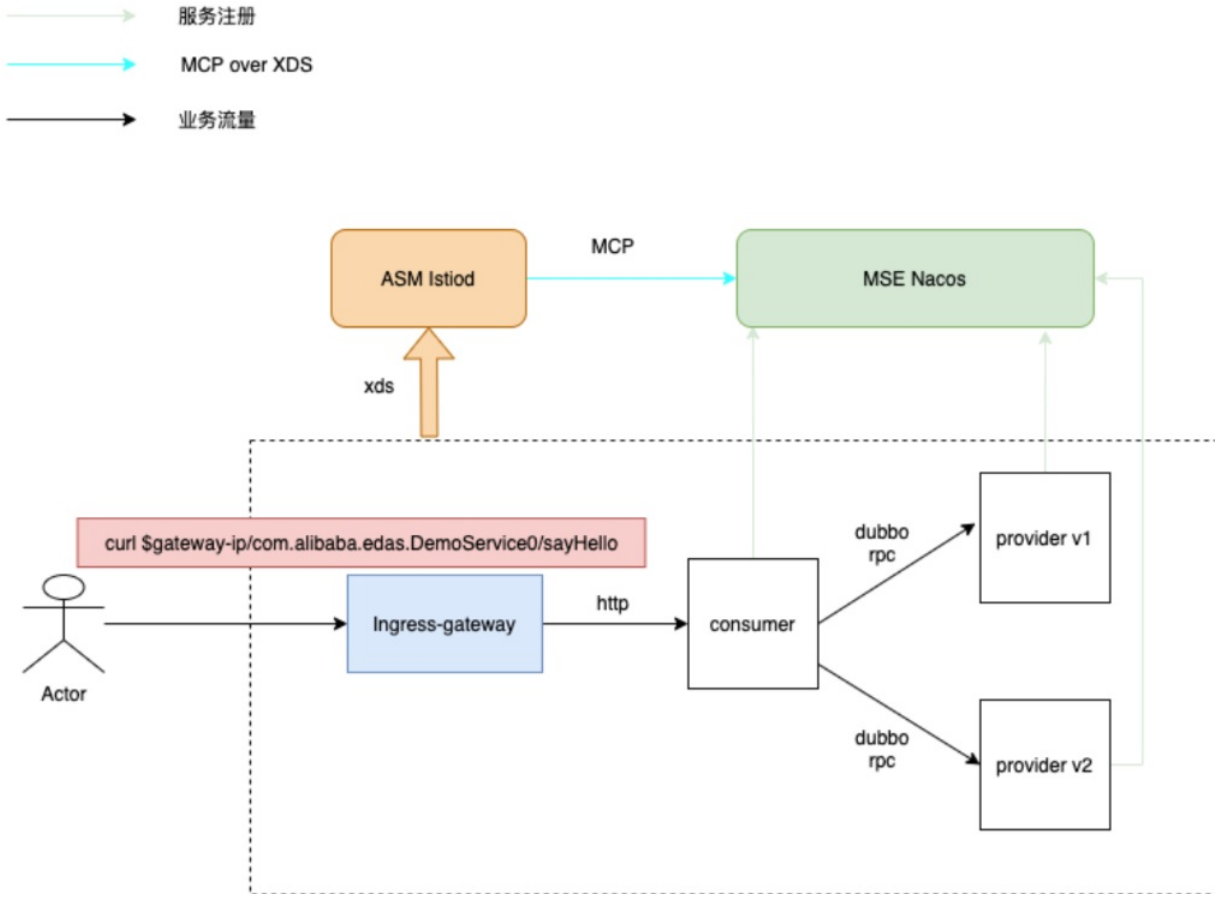
1. 创建ASM商业版（专业版）。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面单击**创建新网格**。
   iv. 在**创建新网格**面板配置参数，以下为重点参数说明，其他参数，请参见创建ASM实例。

参数	说明
规格	选择**商业版（专业版）**。
Nacos注册服务	选中**Nacos注册服务**，然后选择Nacos引擎。 **说明** Nacos引擎需要与ASM实例处于同一VPC，不然无法选择到Nacos引擎。

   v. 了解和接受服务协议，并已阅读和同意阿里云服务网格服务条款和免责声明，选中该选项。然后单击**确定**。
2. 开启Sidecar自动注入。具体操作，请参见多种方式灵活开启自动注入。
3. 添加ACK集群到ASM。具体操作，请参见添加集群到ASM实例。
4. 部署默认入口网关。具体操作，请参见添加入口网关服务。

步骤三：在ACK集群部署Dubbo服务

在ACK中部署一个consumer服务，一个v1版本provider服务，一个v2版本provider服务。



1. 通过kubectl工具连接集群。
2. 执行以下命令，部署Dubbo服务。

   > **说明** 关于Dubbo服务的YAML文件，请参见dubbo-nacos-example。

   ```
   export NACOS_ADDRESS=xxxx # xxxx为MSE的Nacos地址，建议使用内网地址。
   curl https://raw.githubusercontent.com/AliyunContainerService/asm-labs/main/dubbo/dubbo-nacos-example/demo.yaml
   sed -e "s/_NACOS_SERVER_CLUSTERIP_/$NACOS_ADDRESS/g" demo.yaml |kubectl apply -f -
   ```

3. 执行以下命令，查看部署结果。

   ```
   kubectl get pods
   ```

   预期输出：

   ```
   NAME                                READY   STATUS    RESTARTS   AGE
   consumer-7448bc59b7-p9gt8           2/2     Running   0          5d20h
   provider-v1-5f4496cfd-77cjq         2/2     Running   0          5d20h
   provider-v2-6d7d59fcc4-7gcq4        2/2     Running   0          5d20h
   ```

步骤四：创建ASM自定义资源

1. 通过kubectl连接ASM实例。

2. 部署网关规则和虚拟服务。
   i. 使用以下内容，创建名为*gateway*的YAML文件。

```
---
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: test-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: consumerhttp
spec:
  hosts:
  - "*"
  gateways:
  - test-gateway
  http:
  - match:
    - uri:
        prefix: /com.alibaba.edas.DemoService
    route:
    - destination:
        host: consumer.default.svc.cluster.local
        port:
          number: 8080
```

   ii. 执行以下命令，部署网关规则和虚拟服务。

```
kubectl apply -f gateway.yaml
```

结果验证

1. 查看Ingress Gateway的IP地址。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
   iv. 在网格详情页面左侧导航栏单击**ASM网关**，在**ASM网关**页查看Ingress Gateway **Kubernetes服务**下的IP地址。
2. 执行以下命令，通过Ingress Gateway向Dubbo服务发起请求。

```
curl <Ingress Gateway的IP地址>/com.alibaba.edas.DemoService0/sayHello
```

预期输出：

```
Genervice Service v1 implement, invoke method:sayHello
```

6.2. 管理Dubbo服务流量

Dubbo是一个开源分布式框架，致力于提供高性能和透明化的远程服务调用方案。遵循Dubbo框架的服务即为Dubbo服务。本文介绍如何配置Dubbo服务的DestinationRule和VirtualService，实现Dubbo服务流量管理。

前提条件

- 已创建ASM商业版（专业版）实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已在集群中部署Dubbo服务，本文以v1和v2版本的demoservice服务为例。具体操作，请参见托管Dubbo服务。

配置目标规则

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: demoservice
  namespace: default
spec:
  host: providers:com.alibaba.edas.DemoService0:1.0.0:test.DEFAULT-GROUP.public.nacos
  subsets:
  - name: v1
    labels:
      appversion: v1
  - name: v2
    labels:
      appversion: v2
  trafficPolicy:
    loadBalancer:
      simple: ROUND_ROBIN
```

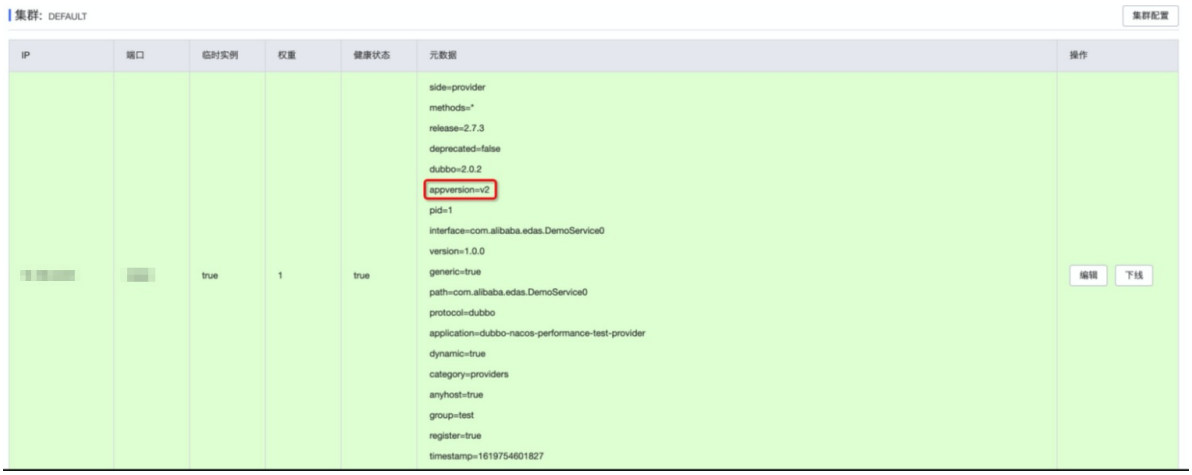
配置Dubbo服务的目标规则需要重点注意以下3个字段：

- `simple`：支持ROUND_ROBIN、LEAST_CONN、RANDOM 3种负载均衡策略。
  - ROUND_ROBIN：请求以轮询的方式转给实例。
  - LEAST_CONN：请求被转给最小连接数的实例。
  - RANDOM：请求以随机的方式转给实例。
- `label`：`labels`标签都需要是来自于Nacos注册中心下的标签元信息。

  ACK集群下针对该Dubbo服务的标签元信息不会被采纳，这是因为Istiod当前只同步了Nacos注册中心下的标签元信息，忽略了Kubernetes该服务部署的Pod下的标签元信息。

> **说明** Dubbo标签元信息注册到注册中心有以下两种方式：
>
> - 代码方式
>
> ```
> Map<String, String> params = service.getParameters();
> params.put("appversion", "v1");
> ```
>
> - 环境变量方式
>
> 部署Deployment时添加环境变量，需要Dubbo版本2.7以上。
>
> ```
> dubbo.application.parameters=[{key1:value1},{key2:value2}]
> ```

服务元信息对应Nacos注册中心服务详情页下的元数据部分，如下图所示。



- `host`：host标明了Dubbo服务，在服务注册中心具有唯一性。分为以下4个部分：
  i. `providers:com.alibaba.edas.DemoService0:1.0.0:test`：服务名，固定加上`providers:`前缀，后面由Dubbo服务的三要素`interface:version:group`构成。
  ii. `DEFAULT-GROUP`：服务所在的分组名，默认为`DEFAULT-GROUP`。
  iii. `public`：服务注册所在的命名空间。
  iv. `nacos`：Nacos注册中心的域名后缀`.nacos`。

配置虚拟服务

配置Dubbo服务的虚拟服务需要重点注意以下3个字段：

> **说明** 关于Dubbo服务的虚拟服务更多参数说明，请参见Dubbo服务的参数说明。

- services
  - 对应Dubbo服务接口名，支持prefix、exact、regex匹配。
  - services为数组类型，可以配置多个service，多个service之间为或关系，请求只需匹配其中一条即可。建议实际配置采用单个service方式。
- routes

- match
  - 可以根据方法名、参数个数、参数类型及取值来设置匹配。
  - 可以根据请求下的 `Header (attachements)` 来设置匹配。
  - 数值匹配支持strValue、doubleValue、boolValue类型，具体匹配模式patterns支持prefix、exact、regex，可配置多条。
- route

  指定了符合此条件的流量的实际目标地址，支持按照百分比权重设置流量流向。

配置示例1：根据请求参数指定服务流量

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: demoservice0
spec:
  hosts:
  - providers:com.alibaba.edas.DemoService0
  dubbo:
  - routes:
    - match:
      ### genericService.$invoke(method, new String[]{String.class.getName()}, new Object[]
{name})
      - method:
          argc: 3
          args:
          - index: 1
            strValue:
              patterns:
              - exact: sayHello
            type: java.lang.String
      route:
      - destination:
          subset: v1
        weight: 100
    - match:
      - method:
          argc: 3
          args:
          - index: 1
            strValue:
              patterns:
              - exact: sayWorld
            type: java.lang.String
      route:
      - destination:
          subset: v2
        weight: 100
    services:
    - prefix: providers:com.alibaba.edas.DemoService0
```

以上虚拟服务设置了以下限制：

- 当请求中包含sayHello参数时，请求路由到v1版本的Dubbo服务。
- 当请求中包含sayWorld参数时，请求路由到v2版本的Dubbo服务。

配置示例2：根据请求头指定服务流量

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: demoservice0
spec:
  hosts:
  - providers:com.alibaba.edas.DemoService0
  dubbo:
  - routes:
    - match:
      - method:
          nameMatch:
            exact: "sayHello"
          argc: 1
          args:
          - index: 1
            strValue:
              patterns:
              - exact: "jack"
            type: java.lang.String
      route:
      - destination:
          subset: v1
        weight: 100
    - match:
      - method:
          nameMatch:
           exact: "sayHello"
          argc: 1
          args:
          - index: 1
            strValue:
              patterns:
              - exact: "lily"
            type: java.lang.String
        headers:
         app:
          patterns:
            - exact: "consumer1"
      route:
      - destination:
          subset: v2
        weight: 100
    services:
    - prefix: providers:com.alibaba.edas.DemoService0
```

以上虚拟服务配置的是针对 `providers:com.alibaba.edas.DemoService0` 服务的路由，设置了以下限制：

- 所有*sayHello("jack")*请求会被路由到v1版本的Dubbo服务。

- 只有来自consumer1的*sayHello("lily")*会被路由到v2版本的Dubbo服务。

相关文档

- Dubbo服务的参数说明

6.3. Dubbo服务的参数说明

您可以通过虚拟服务管理Dubbo服务的流量。本文介绍Dubbo服务的虚拟服务涉及的参数。

虚拟服务示例

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: demoservice0
spec:
  hosts:
  - providers:com.alibaba.edas.DemoService0
  dubbo:
  - routes:
    - match:
      - method:
          name_match:
            exact: "sayHello"
          argc: 1
          args:
          - index: 1
            strValue:
              patterns:
              - exact: "jack"
            type: java.lang.String
      route:
      - destination:
          subset: v1
        weight: 100
    - match:
      - method:
          name_match:
           exact: "sayHello"
          argc: 1
          args:
          - index: 1
            strValue:
              patterns:
              - exact: "lily"
            type: java.lang.String
        headers:
         app:
          patterns:
            - exact: "consumer1"
      route:
      - destination:
          subset: v2
        weight: 100
    services:
    - prefix: providers:com.alibaba.edas.DemoService0
```

DubboServiceRoute

DubboServiceRoute是转发Dubbo请求时的路由规则。

参数	类型	描述	是否必填

参数	类型	描述	是否必填
services	StringMatch[]	对哪些Dubbo服务生效， 建议按一个应用一个VirtualService配置。	是
routes	DubboRoute[]	路由目标，为一个DubboRouteDestination类型的数组，表示满足条件的流量目标。	是

DubboRoute

DubboRoute是Dubbo请求匹配具体服务时的路由规则。

参数	类型	描述	是否必填
match	DubboMatchRequest[]	条件字段match，为一个DubboMatchRequest类型的数组，表示Dubbo请求满足的条件。	否
route	DubboRouteDestination[]	路由目标，为一个DubboRouteDestination类型的数组，表示满足条件的流量目标。	是

DubboMatchRequest

DubboMatchRequest用于匹配请求。

参数	类型	描述	是否必填
method	DubboMethodMatch	匹配请求中的调用方法名和参数。	否
headers	map<string, ListStringMatch>	匹配请求中的header。	否

ListStringMatch

ListStringMatch用于匹配请求头中给定字段的参数值。

> **说明** 区分大小写。

参数	类型	描述	是否必填
patterns	StringMatch[]	表达式集合，需要满足StringMatch描述的匹配条件。	是
action	String	• In：表示patterns描述的StringMatch，满足一个即可。默认为In。 • NotIn：需要与patterns描述的StringMatch都不匹配。	否

```
headers:
  app:
    patterns:
    - exact: "ump2"
  key:
    patterns:
    - prefix: "value"
    - exact: "specific-value"
    action: In
  exceptkey:
    patterns:
    - prefix: "notexist"
    action: NotIn
```

DubboMethodMatch

参数	类型	描述	是否必填
name	String	匹配名称。	是
name_match	StringMatch	匹配请求中的调用方法名。	否
argc	Uint32	匹配请求的参数个数。	否
args	DubboMethodArg[]	为DubboMethodArg类型的数组，表示每个参数值需要满足的条件。	否

DubboMethodArg

DubboMethodArg用于匹配请求中的调用参数。

参数	类型	描述	是否必填
index	Uint32	匹配参数的位置，index字段从1开始（即第$index个参数）。	是
type	String	匹配参数的类型，以Java的string类型为例，该字段取值java.lang.String，该字段默认为java.lang.String。	否
str_value	ListStringMatch	匹配参数的值，根据$type进行解析ListStringMatcher匹配 java.lang.String。	否
num_value	ListDoubleMatch	匹配参数的值，根据$type进行解析ListDoubleMatcher匹配java.lang.Double/java.lang.Long/java.lang.Integer。	否
bool_value	ListBoolMatch	匹配参数的值，根据$type进行解析ListBoolMatcher匹配 java.lang.Boolean。	否

相关文档

- 管理Dubbo服务流量

6.4. 集成自建Prometheus实现Dubbo服务可观测性

Prometheus是一款面向云原生应用程序的开源监控工具。ASM支持集成自建Prometheus，您可以在Prometheus查看Dubbo服务的网格监控数据。本文介绍如何集成自建Prometheus查看Dubbo服务的网格监控数据，实现Dubbo服务可观测性。

前提条件

- 已创建ASM商业版（专业版）实例。具体操作，请参见创建ASM实例。
- 已创建ACK集群。具体操作，请参见创建Kubernetes托管版集群。
- 添加集群到ASM实例。具体操作，请参见添加集群到ASM实例。
- 已部署入口网关服务。具体操作，请参见添加入口网关服务。

步骤一：在ASM开启采集Prometheus监控指标

1. 在ACK集群中部署Prometheus。具体操作，请参见开源Prometheus监控。
2. 创建Prometheus的配置。
   i. 使用以下内容，创建名为*servicemonitor*的YAML文件。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-oper-istio-dataplane
  labels:
    monitoring: istio-dataplane
    release: ack-prometheus-operator
spec:
  selector:
    matchExpressions:
      - {key: istio-prometheus-ignore, operator: DoesNotExist}
  namespaceSelector:
    any: true
  jobLabel: envoy-stats
  endpoints:
  - path: /stats/prometheus
    targetPort: http-envoy-prom
    interval: 15s
    relabelings:
    - sourceLabels: [__meta_kubernetes_pod_container_port_name]
      action: keep
      regex: '.*-envoy-prom'
    - action: labelmap
      regex: "__meta_kubernetes_pod_label_(.+)"
    - sourceLabels: [__meta_kubernetes_namespace]
      action: replace
      targetLabel: namespace
    - sourceLabels: [__meta_kubernetes_pod_name]
      action: replace
      targetLabel: pod_name
```

ii. 执行以下命令，在monitoring命名空间创建ServiceMonitor。

```
kubectl apply -f servicemonitor.yaml --namespace=monitoring
```

3. 在ASM开启采集Prometheus监控指标功能。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
   iv. 在网格管理详情页面选择**网格实例 > 基本信息**，然后在右侧页面单击**功能设置**。

> **说明** 请确保ASM实例的Istio为1.8.6及以上版本。

   v. 在**功能设置更新**对话框中选中**开启采集Prometheus监控指标**，选中**启用已有Prometheus**，输入Prometheus地址，然后单击**确定**。
   ASM将自动生成采集Prometheus监控指标相关的EnvoyFilter配置。
4. 验证在ASM开启采集Prometheus监控指标功能是否成功。
   在浏览器中输入*http://localhost:9090*，单击**Status/Targets**，可以看到新增的采集源。

步骤二：部署Dubbo服务

1. 通过kubectl工具连接集群。
2. 执行以下命令，部署Dubbo服务。

> **说明** 关于Dubbo服务的YAML文件，请参见dubbo-nacos-example。

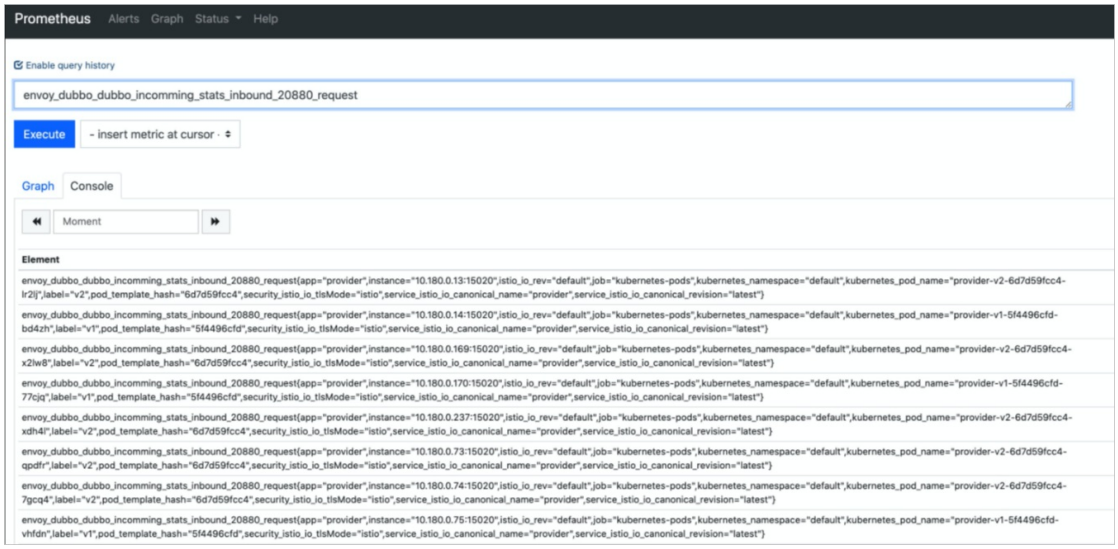
```
export NACOS_ADDRESS=xxxx #xxxx为MSE的Nacos地址，建议使用内网地址。
curl -L https://raw.githubusercontent.com/AliyunContainerService/asm-labs/main/dubbo/dubbo-nacos-example/demo.yaml | sed -e "s/_NACOS_SERVER_CLUSTERIP_/$NACOS_ADDEESS/g" |kubectl apply -f -
```

3. 查看Ingress Gateway的IP地址。
   i. 登录ASM控制台。
   ii. 在左侧导航栏，选择**服务网格 > 网格管理**。
   iii. 在**网格管理**页面，找到待配置的实例，单击实例的名称或在**操作**列中单击**管理**。
   iv. 在网格详情页面左侧导航栏单击**ASM网关**。
   在**ASM网关**页面查看Ingress Gateway **Kubernetes服务**列下的IP地址。
4. 重复执行以下命令，向Dubbo服务发送请求，产生流量。

```
for i in {1..100}; do curl http://<Ingress Gateway地址>/com.alibaba.edas.DemoService0/sayHello; done
```

步骤三：查看监控数据

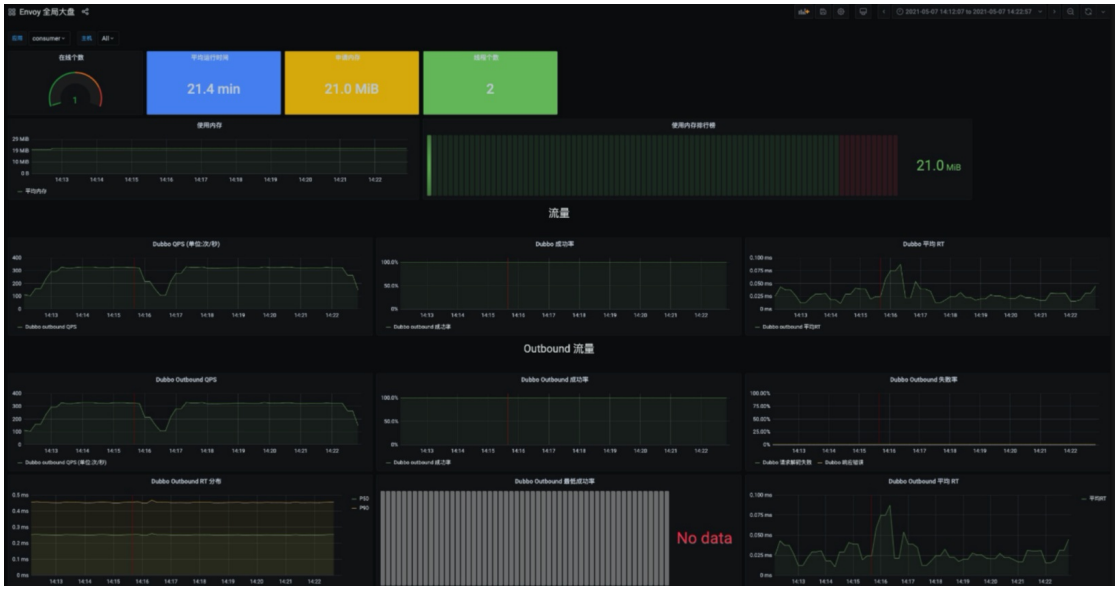
1. 查看Prometheus网格监控指标数据。
   i. 访问Prometheus。具体操作，请参见开源Prometheus监控。
   ii. 在Prometheus页面输入*envoy_dubbo_dubbo_incomming_stats_inbound_28880_request*，单击**Execute**。
   可以在Prometheus页面看到Dubbo服务的监控指标数据。



2. 查看Grafana网格监控指标数据。
   i. 打开Grafana大盘。具体操作，请参见开源Prometheus监控。
   ii. 在左侧导航栏选择 > **Manage**。

iii. 在Manage页签下单击**Import**，导入Envoy大盘配置文件。

导入成功后，显示以下大盘信息。



该Envoy大盘主要包括Sidecar（Envoy）的CPU、内存系统指标，以及Dubbo业务请求的QPS、RT等相关指标，其中Dubbo Provider对应Inbound流量，Dubbo Consumer对应Outbound流量。

iv. （可选）若页面提示找不到ServiceMesh对应数据源，您可以在Grafana面板选择ServiceMesh数据源，然后单击**Add data source**。