

---

# 目录

Introduction	1.1
翻译说明	1.2
文档	1.3
入门	1.4
向网格添加服务	1.5
示例：给应用除错	1.6
使用Prometheus获取遥测数据	1.7
产品路线图	1.8

# Conduit官方文档中文版

## 介绍

Conduit是由Buoyant公司开发的面向Kubernetes的新一代超轻量Service Mesh开源项目。

这是Conduit官方文档的中文翻译版，由Service Mesh中国社区组织翻译并更新维护。

文档内容发布于gitbook，请点击下面的链接阅读或者下载电子版本：

- 在线阅读
  - 国外服务器：gitbook提供的托管，服务器在国外，速度比较慢，偶尔被墙，HTTPS
  - 国内服务器：腾讯云加速，国内网速极快，非HTTPS
- 下载pdf格式
- 下载mobi格式
- 下载epub格式

本文内容可以任意转载，但是需要注明来源并提供链接。

请勿用于商业出版。

## Service Mesh 中国

目前Service Mesh技术还比较新颖，为了方便技术交流，我们组建了Service Mesh中国社区，欢迎加入：

<http://www.servicemesh.cn/>

您可以通过Awesome Service Mesh资料清单快速了解Service Mesh技术和相关的Istio/linkerd等开源项目，这份清单由Service Mesh中国社区更新维护：

<https://servicemesh.gitbooks.io/awesome-servicemesh/>

如果有意加入Service Mesh中国社区的微信群，请联系微信ID `xiaoshu062`，注明“服务网格”。

# 翻译说明

目前Conduit官方文档的翻译由[Service Mesh中文网](#)在主持，这是一个公益性的工作。

## 贡献您的力量

如果有朋友有意愿加入我们的翻译工作，可以有如下方式贡献您的力量：

1. 审核和校对：如果您在阅读时发现内容有谬误，可以通过 [github issue](#)提交issue给我们，您也可以fork出来然后通过PR提交更改
2. 直接参与翻译工作：您可以报名加入我们的翻译小组，请先加入Service Mesh技术社区，然后在微信群中联系我们。

加入方式：请联系微信ID xiaoshu062，注明“服务网格”。

## 工作方式

以下是和我们翻译内容相关的信息：

- [Conduit官方文档在线浏览地址](#)
- [保存翻译后的中文内容的github地址](#)
- [中文翻译内容发布地址](#)：目前用的是gitbook，会自动从github拉取最新内容生成html发布出来

我们目前采用github保存翻译后的内容，翻译团队使用github issue和project来管理。具体方式为：

1. 未翻译的内容会以github issue的方式拆分为多个issue
2. 请在[github issue](#)中领取感兴趣的任務，并将issue assign到自己的github账号：切记必须这样做明确认领，避免其他人在不知情的情况下重复翻译同一个内容
3. 翻译完成后在微信群中联系其他人做review
4. review完成，关闭issue

过程中：

1. 如果觉得issue包含的内容太多，可以直接修改issue，缩小内容的范围，然后为减少的内容新开其他issue（可以新开一个或者多个）
2. 直接提交翻译后的内容，哪怕还没有review（甚至只翻译了一半）
3. 只使用master分支，暂时不拉分支，避免麻烦

## 约定和术语表

### 术语表

service	服务
microservice	微服务
application	应用/应用程序
mutual TLS	双向TLS
configure	配置
setting	设置
traffic	流量
authentication	认证
authorization	授权
data plane	数据平面
control plane	控制平面

### 约定

以下词汇不翻译：

sidecar  
HTTP header  
TLS

# Conduit概览

Conduit是一个面向Kubernetes的超轻量Service Mesh。他对运行在Kubernetes中的服务间通信进行透明的管理，让服务变得更加安全和可靠。在不需要变更代码的前提下，Conduit微服务提供了可靠性、安全性和可监控的特性。

本文档将高层次的概述Conduit，及其它是如何工作的。如果不熟悉service mesh模型，可以先阅读William Morgan的概览文章[What's a service mesh? And why do I need one?](#)

译者注：

1. [什么是服务网格以及为什么我们需要服务网格？](#) 这是上文的中文翻译版本
2. [Awesome Service Mesh资料清单](#): 可以从这里得到更多的Service Mesh资料，持续更新中

## Conduit架构

Conduit service mesh部署到Kubernetes集群时有两个基本组件：数据平面和控制平面。数据平面承载服务实例间的实际应用请求流量，而控制平面驱动数据平面，并提供API以修改其行为（还有访问聚合指标）。Conduit CLI和web UI使用这个API，并提供适用于人类的人体工学控制。

让我们依次认识这些组件。

Conduit的数据平面由轻量级的代理组成，这些代理部署为sidecar容器，与每个服务代码的实例在一起。要“增加”服务到Conduit service mesh，该服务的pods必须重新部署，以便在每个pod中包含一个数据平面。（`conduit inject` 可以完成这个任务，以及必要的配置工作，以便通过代理来从每个实例透明的获取流量。）

这些代理透明地拦截进出每个pod的通信，并增加诸如重试和超时、仪表及加密（TLS）等特性，甚至根据相关策略来允许和禁止请求。

这些代理并未设计成通过手动方式配置；相反，它们的行为是由控制平面驱动的。

Conduit控制平面是一系列服务，运行在专用的Kubernetes命名空间（默认是 `conduit`）。这些服务完成各种的任务——聚合遥测数据、提供面向用户的API、为数据平面代理提供控制数据，等等。总之，它们驱动数据平面的行为。

## 使用Conduit

为了支持Conduit的人机交互，可以使用Conduit CLI及web UI（也可以通过相关工具比如 `kubect1` ）。CLI和web UI通过API驱动控制平面，而控制平面相应地驱动数据平面的行为。

控制平面API设计的足够通用，以便能基于它构建其他工具。比如，你可能希望另外从一个CI/CD系统来驱动API。

运行 `conduit --help` 可查看关于CLI功能的简短概述。

## Conduit与Kubernetes

Conduit设计为可以无缝地融入现有的Kubernetes系统。该设计有几个重要特征。

首先，Conduit CLI（ `conduit` ）设计成尽可能与 `kubect1` 一起使用。比如， `conduit install` 和 `conduit inject` 生成的Kubernetes配置，被设计成可以直接送入 `kubect1` 。这是为了在service mesh和orchestrator之间提供一个清晰的工作分工，且能更加容易地适配Conduit到已有Kubernetes工作流。

其次，Conduit在Kubernetes中的核心名词是Deployment，而不是Service。举个例子， `conduit inject` 增加Deployment，Conduit web UI显示Deployments，并按Deployment提供聚合的性能指标。这是因为单个pods可以是任意数量Service的一部分，而这会导致通信流量与pods之间的复杂映射。相比之下，Deployment要求单个pod最多是一个Deployment的一部分。通过基于Deployment而不是Service来构建，流量与pod间的映射就总是清晰的。

这两个设计特性可以良好组合。比如， `conduit inject` 可用于一个运行的Deployment，因为它当它更新Deployment时，Kubernetes会回滚pods以包含数据平面代理。

## 扩展Conduit的行为

Conduit控制平面还为构建自定义功能提供便利。Conduit初始发布时并不支持这一点，在不远的将来，可以通过编写gRPC插件，作为控制平面的一部分运行，来扩展Conduit的功能，而无需重新编译Conduit。

# 入门

Conduit是一个面向Kubernetes的超轻量Service Mesh。他对运行在Kubernetes中的服务间通信进行透明的管理，让服务变得更加安全和可靠。在不需要变更代码的前提下，Conduit微服务提供了可靠性、安全性和可监控的特性。

Conduit有两个基础组件：数据平面由轻量级代理组成，这些代理部署为sidecar容器，与每个服务代码的实例在一起；控制平面 则用来对这些代理进行协调和管理。用户可以使用命令行界面（CLI）来和Service Mesh进行交互，另外还有Web应用用于控制集群。

本文会谈到如何在Kubernetes集群中部署Conduit，并在其上运行一个gRPC示例应用。注意Conduit 0.1是一个Alpha版本。他Alpha到什么程度呢？目前仅提供了HTTP/2（包括gRPC），甚至HTTP/1.1也不被支持。如果你没有HTTP/2应用，也不用担心，我们已经提供示例应用给你尝试。

## 先决条件

需要运行Kubernetes 1.8版本的集群，还有在本机可正常工作的 `kubectl` 命令。可以用下面的命令来检查是否符合需要：

```
$ kubectl version --short
```

应该会看到类似这样的内容：

```
Client Version: v1.8.3
Server Version: v1.8.0
```

请确认"Server Version"内容是1.8或以上。如果返回内容不是上面的情况，或者 `kubectl` 返回了错误信息，可能是集群不存在或者没有正确配置（如果想要快速简便的在本机运行Kubernetes，我们建议使用[Minikube](#)，需要注意的是，要求0.24.1或其后的版本。）

## 安装CLI

如果你是首次运行Conduit，需要下载命令行接口工具（CLI）到本机，然后使用CLI在Kubernetes集群上安装Conduit。

运行下列命令，开始安装CLI：

```
curl https://run.conduit.io/install | sh
```

然后跟随指示即可。

另外，还可以直接在[Conduit发布页面](#)下载CLI；或者可以从[Conduit Github](#)克隆源代码，根据README指导自行Build。

CLI安装之后，用下面命令检查一下：

```
conduit version
```

应该会看到类似的内容：

```
Client version: v0.1.0  
Server version: unavailable
```

## 在集群上安装Conduit

现在本机安装好了CLI了，是时候在Kubernetes集群上安装Conduit的控制平面了。不要担心目前集群上运行的其他应用——控制平面会安装在单独的 `conduit` 命名空间，所以要删除也是很容易的。

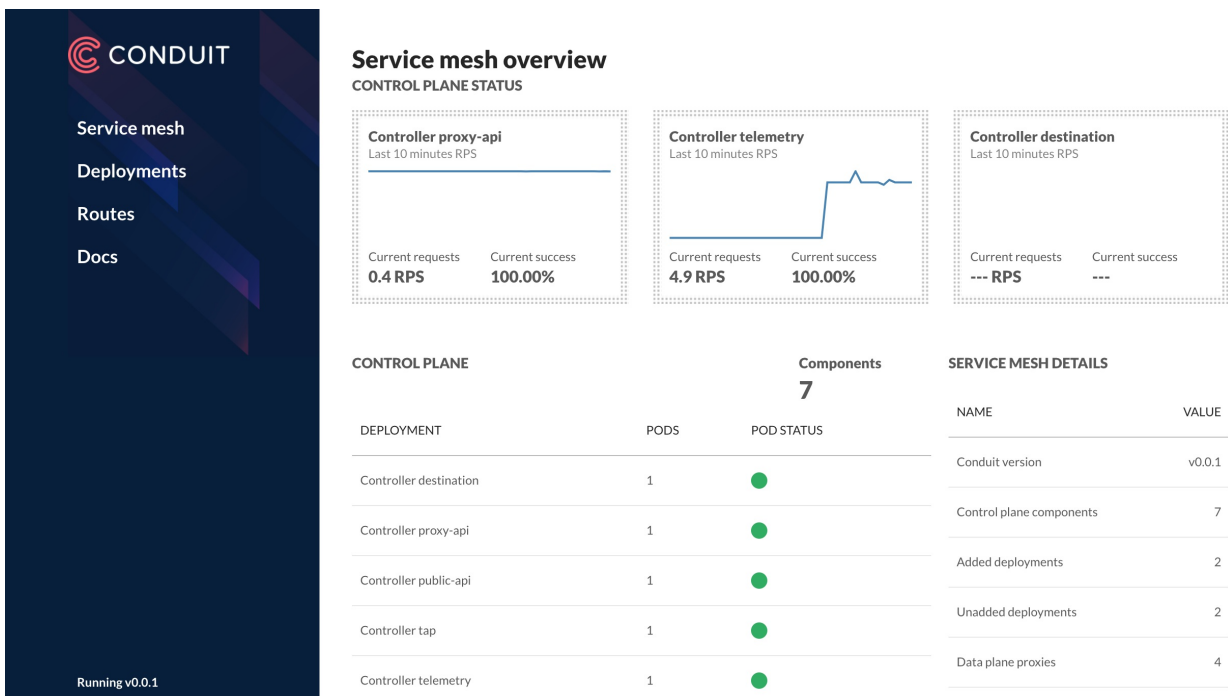
运行下面的命令：

```
conduit install | kubectl apply -f -  
conduit dashboard
```

第一个命令生成Kubernetes配置，然后用管道输出给 `kubectl`。Kubectl随后把配置应用到Kubernetes集群。（可以把这个命令拆成两个命令，来观察在应用之前发生了什么，例如 `conduit install > foo.yml; cat foo.yml`）。

第二个命令在浏览器窗口中加载仪表盘，如果你看到下图类似的内容，那么恭喜你，Conduit已经在你的集群上运行。





当然，目前还没有任何服务运行在Service Mesh之中，所以仪表盘没有什么内容，只显示了Service Mesh自身的情况。

## 安装演示应用

终于，到了安装演示应用并添加到mesh的环节了。完成这所有只需要运行下面的一条命令：

```
curl https://raw.githubusercontent.com/runconduit/conduit-examples/master/emojivoto/emojivoto.yml | conduit inject --skip-inbound-ports=80 | kubectl apply -f -
```

这个的命令下载示例gRPC应用的Kubernetes配置，然后通过 `conduit inject` 运行配置。这将重写配置，在应用Pod中插入Conduit数据平面代理，作为sidecar容器。最后，`kubectl` 应用这些配置到Kubernetes集群。

和用 `conduit install` 类似，在这个命令中，Conduit CLI只是简单地进行文本转换而 `kubectl` 做了大量工作来把配置实际应用到Kubernetes集群。以这种方式，可以在管道中引入额外的过滤器，或者单独运行命令来观察每个命令的输出。

至此，这个应用就在Kubernetes集群上运行了，并且（它还不知道！）也被加入到了Conduit的Service Mesh之中。

## 观察运行情况

浏览一下Conduit的仪表盘，则应该会看到演示应用的所有HTTP/2服务，这些服务在已添加到Conduit网络的部署列表中显示。

我们可以用下面的URL查看应用：

```
minikube -n emoji voto service web-svc --url # on minikube
kubectl get svc web-svc -n emoji voto -o jsonpath="{.status.loadBalancer.ingress[0].*}"
# on GKE
```

为了让仪表盘生动一些，我们可以编程生成一些流量。首先，我们需要IP地址：

```
INGRESS=$(minikube -n emoji voto service web-svc --url | cut -f 3 -d / -) # on minikube
INGRESS=$(kubectl get svc web-svc -n emoji voto -o jsonpath="{.status.loadBalancer.ingress[0].*}") # on GKE
```

现在我们用一个简单的循环来生成流量：

```
while true; do curl $INGRESS; done
```

最后，我们再看仪表盘（如果尚未运行，运行 `conduit dashboard`）。可以浏览所有的作为应用一部分的服务，并查看这些服务：

- 成功率
- 请求率
- 延迟分布区间
- 上下游依赖

以及有关实况流量的其他各种信息。整洁，对不对？

## 使用CLI

当然，仪表盘不是观测Conduit Service Mesh的唯一手段。CLI提供了很多有趣又强大的命令可以用来体验，包括：

- `conduit stat`
- `conduit tap`

试试看运行 `conduit stat deployments` 和 `conduit tap deploy default/xxx`，看看会有什么发现。

## 总结

这就是我们的入门指南。要获得关于Conduit的更多信息，可以查看[概述](#)和[路线图](#)，或者加入[Linkerd Slack](#)的 `#conduit` 频道，以及浏览[Conduit论坛](#)。还可以在Twitter上关注[@runconduit](#)。我们刚刚开始Conduit的构建工作，我们对您的反馈非常感兴趣！

## 向网格添加服务

为了使您的服务能够使用Conduit，需要将服务加入到网格中。通过使用Conduit CLI可以添加Conduit代理sidecar到每个pod。通过滚动更新，您的应用程序的可用性不会受到影响。

### 先决条件

- 当前版本Conduit,0.1.0,仅支持HTTP/2(包括gRPC)，您的服务在pod外部进行的任何网络调用都必须是HTTP/2。
- 您的服务可能会收到非HTTP/2流量，但必须明确配置为跳过代理，并且此流量对服务网格不可见（请参阅[非HTTP/2流量](#)）。
- 由于早期版本中的[bug](#)，使用grpc-go的服务必须使用grpc-go版本1.3或更高版本。
- 您必须已经在本地安装了Conduit CLI，并且在您的Kubernetes群集中安装了Conduit服务网格。有关详细信息，请参阅[入门指南](#)。

### 添加你的服务

要将服务添加到服务网格，只需运行此命令：

```
conduit inject deployment.yml | kubectl apply -f -
```

其中 `deployment.yml` 是包含您应用的Kubernetes配置文件。这将触发deployment的滚动更新，将每个pod替换为额外包含Conduit sidecar代理的新pod。

如果Conduit仪表板中的代理状态为绿色，您就可以知道您的服务已成功添加到服务网格中。





## DATA PLANE

## Deployments

4

## Proxies

4

DEPLOYMENT	PODS	PROXY STATUS
<a href="#">emojivoto/api</a>	1	
<a href="#">emojivoto/emoji-svc</a>	1	
<a href="#">emojivoto/voting-svc</a>	1	
<a href="#">emojivoto/web</a>	1	

您可以随时进入Conduit仪表板，只要运行：

```
conduit dashboard
```

## 非HTTP/2流量

截至当前版本，Conduit代理仅支持HTTP/2。如果您的服务接受非HTTP/2流量，您可以配置一个入站端口列表，这将绕过代理，并直接进入您的应用程序。绕过代理的流量对于服务网格不可见。

您可以配置绕过conduit的入站端口列表，通过传递以逗号分隔的端口列表给 `conduit inject` 的 `--skip-inbound-ports` 参数。例如，要允许端口80和7777上的入站流量绕过代理，使用以下命令：

```
conduit inject deployment.yml --skip-inbound-ports=80,7777 | kubectl apply -f -
```

## 示例：给应用除错

这里首先假设读者已经跟随[入门指南](#)进行了安装配置，Conduit和演示应用都已经在Kubernetes集群上运行。

### 用Conduit为故障服务排错

Conduit及其演示应用已经[启动运行](#)，我们接下来使用Conduit来检测问题。

首先，使用 `conduit stat` 命令来获取部署健康情况：

```
conduit stat deployments
```

大概会看到类似下面的输出内容：

NAME	REQUEST_RATE	SUCCESS_RATE	P50_LATENCY	P99_LATENCY
emojivoto/emoji	2.0rps	100.00%	0ms	0ms
emojivoto/voting	0.6rps	66.67%	0ms	0ms
emojivoto/web	2.0rps	95.00%	0ms	0ms

我们会发现 `voting` 服务比其他情况差了很多。

我们是如何完成这一发现的？过去的办法是：看日志、加入调试器等。Conduit提供了新的工具，可以查看部署内的流量情况。我们可以使用 `tap` 命令来查看当前请求在部署中的流动。

```
conduit tap deploy emojivoto/voting
```

会出现大量的请求：

```

req id=0:458 src=172.17.0.9:45244 dst=172.17.0.8:8080 :method=POST :authority=voting-svc.emojivoto:8080 :path=/emojivoto.v1.VotingService/VoteGhost
rsp id=0:458 src=172.17.0.9:45244 dst=172.17.0.8:8080 :status=200 latency=758µs
end id=0:458 src=172.17.0.9:45244 dst=172.17.0.8:8080 grpc-status=OK duration=9µs response-length=5B
req id=0:459 src=172.17.0.9:45244 dst=172.17.0.8:8080 :method=POST :authority=voting-svc.emojivoto:8080 :path=/emojivoto.v1.VotingService/VoteDoughnut
rsp id=0:459 src=172.17.0.9:45244 dst=172.17.0.8:8080 :status=200 latency=987µs
end id=0:459 src=172.17.0.9:45244 dst=172.17.0.8:8080 grpc-status=OK duration=9µs response-length=5B
req id=0:460 src=172.17.0.9:45244 dst=172.17.0.8:8080 :method=POST :authority=voting-svc.emojivoto:8080 :path=/emojivoto.v1.VotingService/VoteBurrito
rsp id=0:460 src=172.17.0.9:45244 dst=172.17.0.8:8080 :status=200 latency=767µs
end id=0:460 src=172.17.0.9:45244 dst=172.17.0.8:8080 grpc-status=OK duration=18µs response-length=5B
req id=0:461 src=172.17.0.9:45244 dst=172.17.0.8:8080 :method=POST :authority=voting-svc.emojivoto:8080 :path=/emojivoto.v1.VotingService/VoteDog
rsp id=0:461 src=172.17.0.9:45244 dst=172.17.0.8:8080 :status=200 latency=693µs
end id=0:461 src=172.17.0.9:45244 dst=172.17.0.8:8080 grpc-status=OK duration=10µs response-length=5B
req id=0:462 src=172.17.0.9:45244 dst=172.17.0.8:8080 :method=POST :authority=voting-svc.emojivoto:8080 :path=/emojivoto.v1.VotingService/VotePoop

```

我们看看是不是可以在其中发现点什么。我们注意到日志中有一些 `grpc-status=Unknown`，这表示GRPC请求失败。

我们接下来看看这一问题的来由。再次运行 `tap` 命令，过滤出其中的 `Unknown`：

```

conduit tap deploy emojivoto/voting | grep Unknown -B 2

req id=0:212 src=172.17.0.8:58326 dst=172.17.0.10:8080 :method=POST :authority=voting-svc.emojivoto:8080 :path=/emojivoto.v1.VotingService/VotePoop
rsp id=0:212 src=172.17.0.8:58326 dst=172.17.0.10:8080 :status=200 latency=360µs
end id=0:212 src=172.17.0.8:58326 dst=172.17.0.10:8080 grpc-status=Unknown duration=0µs response-length=0B
--
req id=0:215 src=172.17.0.8:58326 dst=172.17.0.10:8080 :method=POST :authority=voting-svc.emojivoto:8080 :path=/emojivoto.v1.VotingService/VotePoop
rsp id=0:215 src=172.17.0.8:58326 dst=172.17.0.10:8080 :status=200 latency=414µs
end id=0:215 src=172.17.0.8:58326 dst=172.17.0.10:8080 grpc-status=Unknown duration=0µs response-length=0B
--

```

这样看到，所有的 `grpc-status=Unknown` 都来源于 `VotePoop` 这一端点。我们可以使用 `tap` 命令的参数，来关注这一端点的输出：

```
conduit tap deploy emoji voto/voting --path /emoji voto.v1.VotingService/VotePoop

req id=0:264 src=172.17.0.8:58326 dst=172.17.0.10:8080 :method=POST :authority=voting-
svc.emoji voto:8080 :path=/emoji voto.v1.VotingService/VotePoop
rsp id=0:264 src=172.17.0.8:58326 dst=172.17.0.10:8080 :status=200 latency=696µs
end id=0:264 src=172.17.0.8:58326 dst=172.17.0.10:8080 grpc-status=Unknown duration=0µ
s response-length=0B
req id=0:266 src=172.17.0.8:58326 dst=172.17.0.10:8080 :method=POST :authority=voting-
svc.emoji voto:8080 :path=/emoji voto.v1.VotingService/VotePoop
rsp id=0:266 src=172.17.0.8:58326 dst=172.17.0.10:8080 :status=200 latency=667µs
end id=0:266 src=172.17.0.8:58326 dst=172.17.0.10:8080 grpc-status=Unknown duration=0µ
s response-length=0B
req id=0:270 src=172.17.0.8:58326 dst=172.17.0.10:8080 :method=POST :authority=voting-
svc.emoji voto:8080 :path=/emoji voto.v1.VotingService/VotePoop
rsp id=0:270 src=172.17.0.8:58326 dst=172.17.0.10:8080 :status=200 latency=346µs
end id=0:270 src=172.17.0.8:58326 dst=172.17.0.10:8080 grpc-status=Unknown duration=0µ
s response-length=0B
```

这样就看到，所有 `VotePoop` 的请求都失败了。当我们尝试给 投票时候会发生什么？在指南的第五步中打开演示应用。

现在点击 来给他投票：



演示应用在投票给 的时候就会出错。这样我们就知道错误来自何处了。接下来就可以扎进日志和代码，来研究具体故障原因了。在Conduit的未来版本中，我们甚至可以通过对路由规则的控制来修改某一端点被调用时候的行为。



## 使用Prometheus获取遥测数据

使用现有Prometheus集群获取Conduit的遥测数据是很容易的。简单的把下面的配置代码加入到Prometheus配置的 `scrape_configs` 之中即可：

```
- job_name: 'conduit'
  kubernetes_sd_configs:
  - role: pod
    namespaces:
      # Replace this with the namespace that Conduit is running in
      names: ['conduit']
  relabel_configs:
  - source_labels: [__meta_kubernetes_pod_container_port_name]
    action: keep
    regex: ^admin-http$
```

这就可以了。现在你的Prometheus集群就已经配置完成，可以抓取Conduit的指标数据了。Conduit的指标会包含标签 `job="conduit"` 以及：

- `request_total` ：请求总数
- `response_total` ：响应总数
- `response_latency_ms` ：毫秒为单位的响应延迟

所有的指标都带有下列标签：

- `source_deployment` ：发出请求的Deployment（或者replicaset, job之类）
- `target_deployment` ：接收请求的Deployment（或者replicaset, job之类）

## 产品路线图

## 代理透明度

Conduit将处理所有的服务间通信，不限于HTTP。

## 服务间隐私

Conduit将默认提供服务间的认证和加密。

## API驱动策略

Conduit将扩展Kubernetes API以支持多种实践中的运维策略。

## 可插拔控制器

Conduit的控制器本身是一个微服务，且可扩展以支持特定环境的策略插件。