使用 iptables 将 Kubernetes Service 流量随机发送到 Pod

K8S中文社区 前天

本文将带大家了解 Kubernetes 的 kube-proxy 组件如何使用 iptables 将 service 流量随机发送到 Pod，目的是实现 service 所需的 iptables 规则。

作者：Dustin Specker 翻译：Sarah (K8sMeetup)
校对：FogDong 来源：K8sMeetup社区

本文将带大家了解 Kubernetes 的 kube-proxy 组件如何使用 iptables 将 service 流量随机发送到 Pod。我们将重点介绍 Kubernetes service 的 ClusterIP 类型。

本文的目的是实现 service 所需的 iptables 规则，例如：

```yaml
apiVersion: v1
kind: Service
metadata:
  name: app-service
spec:
  clusterIP: 10.100.100.100
  selector:
    component: app
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

Kubernetes 为每个 Pod 创建一个网络命名空间。我们将用 python HTTP 服务器（这将被视为我们的“ Pod”）手动创建网络命名空间。

> 注意：本文仅适用于 Linux。我使用的是 Ubuntu 19.10，但在其他 Linux 发行版上应该也可以使用。

创建虚拟设备并在网络命名空间中运行 HTTP 服务器

首先快速搭建环境。启用 IP 转发：

```
sudo sysctl --write net.ipv4.ip_forward=1
```

现在我们需要：

- 创建一个虚拟网桥（名为 `bridge_home`）；
- 创建两个网络命名空间（分别名为 `netns_dustin` 和 `netns_leah`）；
- 在网络命名空间中为 DNS 配置 `8.8.8.8`；
- 创建连接到 `bridge_home` 的两个 veth 对；
- 将 `10.0.0.11` 分配给在 `netns_dustin` 中运行的 veth；
- 将 `10.0.0.21` 分配给在 `netns_leah` 中运行的 veth；
- 在网络命名空间中设置默认路由。

```
sudo ip link add dev bridge_home type bridge
sudo ip address add 10.0.0.1/24 dev bridge_home

sudo ip netns add netns_dustin
sudo mkdir -p /etc/netns/netns_dustin
echo "nameserver 8.8.8.8" | sudo tee -a /etc/netns/netns_dustin/resolv.conf
sudo ip netns exec netns_dustin ip link set dev lo up
sudo ip link add dev veth_dustin type veth peer name veth_ns_dustin
sudo ip link set dev veth_dustin master bridge_home
sudo ip link set dev veth_dustin up
sudo ip link set dev veth_ns_dustin netns netns_dustin
```

```
sudo ip netns exec netns_dustin ip link set dev veth_ns_dustin up
sudo ip netns exec netns_dustin ip address add 10.0.0.11/24 dev veth_ns_dustin

sudo ip netns add netns_leah
sudo mkdir -p /etc/netns/netns_leah
echo "nameserver 8.8.8.8" | sudo tee -a /etc/netns/netns_leah/resolv.conf
sudo ip netns exec netns_leah ip link set dev lo up
sudo ip link add dev veth_leah type veth peer name veth_ns_leah
sudo ip link set dev veth_leah master bridge_home
sudo ip link set dev veth_leah up
sudo ip link set dev veth_ns_leah netns netns_leah
sudo ip netns exec netns_leah ip link set dev veth_ns_leah up
sudo ip netns exec netns_leah ip address add 10.0.0.21/24 dev veth_ns_leah

sudo ip link set bridge_home up
sudo ip netns exec netns_dustin ip route add default via 10.0.0.1
sudo ip netns exec netns_leah ip route add default via 10.0.0.1
```

接下来，创建 iptables 规则以允许流量传入和传出 `bridge_home` 设备：

```
sudo iptables --table filter --append FORWARD --in-interface bridge_home --
jump ACCEPT
sudo iptables --table filter --append FORWARD --out-interface bridge_home --
jump ACCEPT
```

然后，创建另一个 iptables 规则伪装来自我们的网络命名空间的请求：

```
sudo iptables --table nat --append POSTROUTING --source 10.0.0.0/24 --jump
MASQUERADE
```

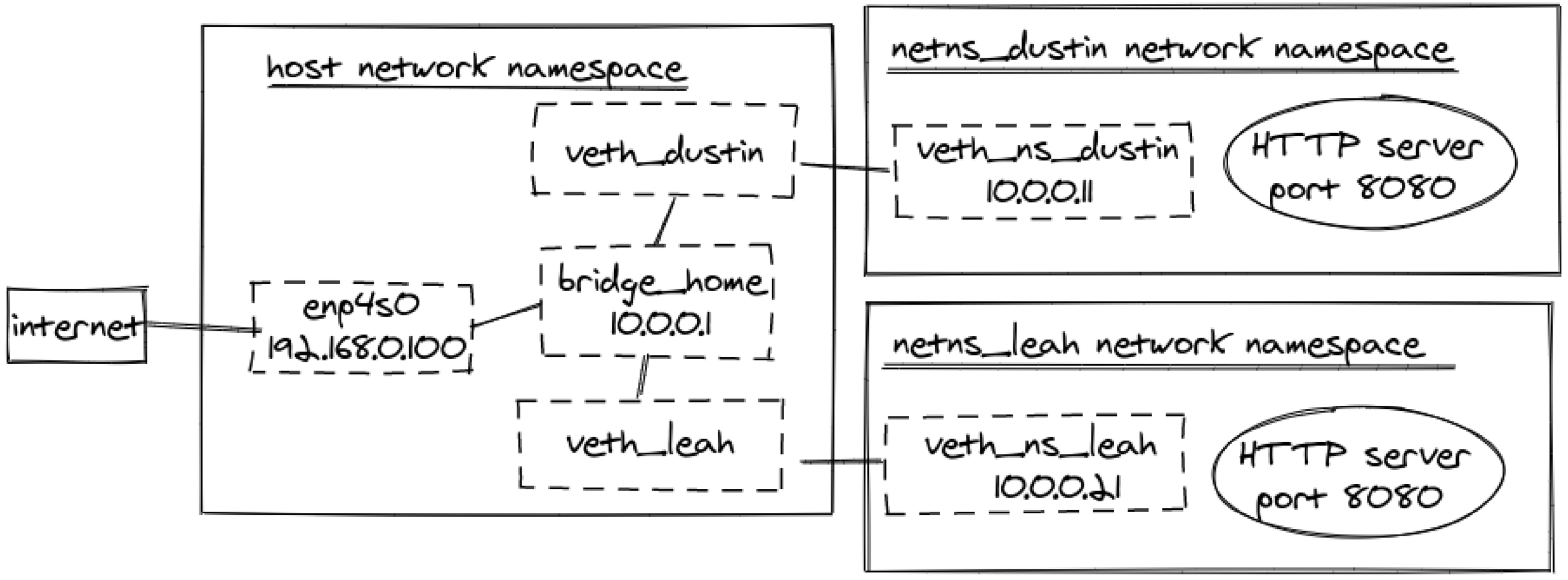
在 `netns_dustin` 网络命名空间中启动一个 HTTP 服务器：

```
sudo ip netns exec netns_dustin python3 -m http.server 8080
```

最后，打开另一个终端并在 `netns_leah` 网络命名空间中启动 HTTP 服务器：

```
0 sudo ip netns exec netns_leah python3 -m http.server 8080
```

这时，我们的环境是这样的：



> 注意：你的 IP 地址可能不是 192.168.0.100，接口的名称可能也不是 enp4s0。

使用以下命令进行健全性检查：

```
0 curl 10.0.0.11:8080
1 curl 10.0.0.21:8080
2 sudo ip netns exec netns_dustin curl 10.0.0.21:8080
3 sudo ip netns exec netns_leah curl 10.0.0.11:8080
```

在 iptables 中添加一个虚拟 IP

创建 Kubernetes Service 时，会为这个新的 Service 分配一个 ClusterIP。从概念上讲，**ClusterIP 是虚拟 IP**。iptables 模式下的 kube-proxy 负责创建 iptables 规则来处理这些虚拟 IP 地址（详情可查看 Kubernetes 文档：https://kubernetes.io/docs/concepts/services-networking/service/#virtual-ips-and-service-proxies）

让我们制定一个简单的 iptables 规则，以了解处理虚拟 IP 地址需要做什么。稍后，我们将进行重构以使我们的规则与 kube-proxy 创建规则的方式保持一致。

在 nat 表中创建一个名为 `DUSTIN_SERVICE` 的新链：

```
0 sudo iptables --table nat --new DUSTIN-SERVICES
```

接下来，我们希望 `PREROUTING` 和 `OUTPUT` 链通过以下方式查看 `DUSTIN-SERVICES` 链：

```
sudo iptables \
  --table nat \
  --append PREROUTING \
  --jump DUSTIN-SERVICES

sudo iptables \
  --table nat \
  --append OUTPUT \
  --jump DUSTIN-SERVICES
```

此时，我们可以在 DUSTIN-SERVICES 链中创建一个规则来处理虚拟 IP。我们的虚拟 IP 为 10.100.100.100。让我们创建一个规则，将 10.100.100.100:8080 的流量定向到 10.0.0.11:8080。回想一下，10.0.0.11:8080 是在 netns_dustin 命名空间中运行的 python HTTP 服务器。

请求虚拟 IP：

好的！我们刚刚处理了虚拟 IP 的流量！

现在有一些坏消息。让我们尝试从 netns_dustin 请求虚拟 IP 地址。

```
sudo ip netns exec netns_dustin curl 10.100.100.100:8080
```

这个命令有时候能成功有时候会失败。为什么会这样？

启用 Hairpin 模式

如果使用最后一条命令失败了，那么我敢打赌，你运行了 Docker。至少对我来说是这样的。为什么 Docker 会干扰呢？从技术上讲并没有，但是 Docker 开启了一个名为 `net.bridge.bridge-nf-call-iptables` 的设置，**会将网桥配置为在处理流量时考虑 iptables**。这还会导致一个问题：**请求离开的设备同时也是被定向的设备**。这恰好是上一个命令中遇到的情况！

让我们再讲得清楚一些。假设有一个请求离开 `veth_dustin`，该请求的源 IP 地址为 `10.0.0.11`。该请求的目的地是 `10.100.100.100`。然后，iptables 规则在 `10.100.100.100` 到 `10.0.0.11` 上执行 DNAT。问题发生了：**该请求需要定向到发出请求的地方**！

让我们以相同的方式配置所有环境。这意味着，如果最后一个命令可以起作用，我们很快就能解决这个问题。

首先，检查 `net.bridge.bridge-nf-call-iptable` 是否已启用：

```
0 sysctl net.bridge.bridge-nf-call-iptables
```

如果出现以下错误：

```
0 sysctl: cannot stat /proc/sys/net/bridge/bridge-nf-call-iptables: No such file
  or directory
```

则运行：

```
0 sudo modprobe br_netfilter
```

这个操作将加载 `br_netfilter` 模块。之后再次运行 `sysctl net.bridge.bridge-nf-call-iptables`。

大家都应该看到 `net.bridge.bridge-nf-call-iptables` 已启用（输出 1）。如果出于某种原因将其禁用了（0），请运行以下命令：

```
0 sudo sysctl --write net.bridge.bridge-nf-call-iptables=1
```

现在你会看到以下命令失败了：

```
0 sudo ip netns exec netns_dustin curl 10.100.100.100:8080
```

接下来进行修复！我们需要在连接至 `bridge_home` 的 `veth_dustin` 上启用 Hairpin 模式。该模式允许离开设备的请求可被同一设备接收。

> 有趣的事实：`veth_dustin` 被称为 `bridge_home` 上的端口。类似于将物理以太网电缆插入物理网桥的端口，另一端插入物理计算机。

在 `veth_dustin` 上启用 Hairpin 模式：

```
0 sudo brctl hairpin bridge_home veth_dustin on
```

尝试再次使用如下命令：

```
0 sudo ip netns exec netns_dustin curl 10.100.100.100:8080
```

成功了！我们希望网络命名空间能够通过虚拟 IP 与自己对话，因此需要在网桥设备的每个端口上启用 Hairpin 模式。幸运的是，**有一种方法可以在桥接设备上而不是每个端口上进行配置**。首先撤消之前所做的操作并禁用 Hairpin 模式：

```
0 sudo brctl hairpin bridge_home veth_dustin off
```

网桥可以处于 **promiscuous mode，它将所有连接的端口（我们是 veth）视为已启用 Hairpin 模式。**在 `bridge_home` 上启用 promiscuous mode：

```
0 sudo ip link set bridge_home promisc on
```

再次运行以下心爱的命令：

```
0 sudo ip netns exec netns_dustin curl 10.100.100.100:8080
```

再次成功！在 bridge_home 上启用 promiscuous 模式后，不必担心将来要在每个 veth（例如 `veth_leah`）上启用 Hairpin 模式！

与 kube-proxy 对齐 iptables 规则

到目前为止，我们已经创建了一个 iptables 规则来处理一项具有一个后端（`10.0.0.11`）的 service（`10.100.100.100`）。我们在名为 `DUSTIN-SERVICES` 的链中创建了此规则，该链的名称与 kube-proxy 的 `KUBERNETES-SERVICES` 相似。kube-proxy 为每个 service 创建一个链，并使 `KUBERNETES-SERVICES` 根据目的地跳转到相应的服务链。

首先，为 service 创建一个新链，将其命名为 HTTP。kube-proxy 在链名称中使用哈希，但是我会坚持使用 HTTP 来帮助理解。创建一个新链：

```
0 sudo iptables \
1   --table nat \
2   --new DUSTIN-SVC-HTTP
```

在 `DUSTIN-SVC-HTTP` 链中添加一条规则，该规则将流量定向到后端（`10.0.0.11`）：

```
sudo iptables \
  --table nat \
  --append DUSTIN-SVC-HTTP \
  --protocol tcp \
  --match tcp \
  --jump DNAT \
  --to-destination 10.0.0.11:8080
```

最后，我们希望 DUSTIN-SERVICES 使用 DUSTIN-SVC-HTTP 链。通过以下方式删除在 DUSTIN-SERVICES 中创建的上一条规则：

```
sudo iptables \
  --table nat \
  --delete DUSTIN-SERVICES \
  --destination 10.100.100.100 \
  --protocol tcp \
  --match tcp \
  --dport 8080 \
  --jump DNAT \
  --to-destination 10.0.0.11:8080
```

并在 DUSTIN-SERVICES 中添加一条规则，通过以下方式跳转到匹配目标位置上的 DUSTIN-SVC-HTTP：

```
sudo iptables \
  --table nat \
  --append DUSTIN-SERVICES \
  --destination 10.100.100.100 \
  --protocol tcp \
  --match tcp \
  --dport 8080 \
  --jump DUSTIN-SVC-HTTP
```

这时，以下的命令仍然奏效：

```
curl 10.100.100.100:8080
sudo ip netns exec netns_dustin curl 10.100.100.100:8080
```

现在，添加一个新 service 将包括以下步骤：

- 为 service 创建一个新链，例如 `DUSTIN-SVC-HTTP`；
- 在 service 链中创建一条规则以将流量定向到后端，例如 `10.0.0.11`；
- 向 `DUSTIN-SERVICES` 添加一条规则以跳至 service 链，例如 `DUSTIN-SVC-HTT`。

重构 service 链以支持多个后端

刚才我们将 `DUSTIN-SERVICES` 链重构为跳转到单个服务链。现在可以重构 service 链（`DUSTIN-SVC-HTTP`），以跳转到其他链来将流量定向到后端。

> 注意：这里一直使用"后端"一词，但是在 Kubernetes 中这些也称为端点（endpoint）。通常，端点是容器的 IP 地址。

让我们为 10.0.0.11 端点创建一个新链。kube-proxy 仍然会为这些链名使用哈希，但是我们将创建一个名为 DUSTIN-SEP-HTTP1 的链，代表 HTTP 的第一个服务端点（SEP）。通过以下方式创建新链：

为新的 DUSTIN-SEP-HTTP1 链添加一个看起来很熟悉的规则：

```
sudo iptables \
  --table nat \
  --append DUSTIN-SEP-HTTP1 \
  --protocol tcp \
  --match tcp \
  --jump DNAT \
  --to-destination 10.0.0.11:8080
```

然后，删除添加到 DUSTIN-SVC-HTTP 的规则，并在 DUSTIN-SVC-HTTP 中添加一条规则以跳至 DUSTIN-SEP-HTTP1：

```
sudo iptables \
  --table nat \
  --delete DUSTIN-SVC-HTTP \
  --protocol tcp \
  --match tcp \
  --jump DNAT \
  --to-destination 10.0.0.11:8080

sudo iptables \
  --table nat \
  --append DUSTIN-SVC-HTTP \
  --jump DUSTIN-SEP-HTTP1
```

以下命令还是会奏效：

```
curl 10.100.100.100:8080
sudo ip netns exec netns_dustin curl 10.100.100.100:8080
```

现在已经准备好开始添加更多后端了。

使用 iptables 为虚拟 IP 提供随机后端

如 Kubernetes 文档中所述，kube-proxy 将流量随机定向到后端（https://kubernetes.io/docs/concepts/services-networking/service/#proxy-mode-iptables）。如何做到的？当然是用 iptables！

iptables 支持根据概率将流量定向到后端。对我来说，这是一个超酷的概念，因为我以前认为 iptables 的规则是确定的！

首先，为在 `netns_leah` 网络命名空间中运行的第二个 HTTP 后端（`10.0.0.21`）添加新的链和规则：

```
sudo iptables \
  --table nat \
  --new DUSTIN-SEP-HTTP2

sudo iptables \
  --table nat \
  --append DUSTIN-SEP-HTTP2 \
  --protocol tcp \
  --match tcp \
  --jump DNAT \
  --to-destination 10.0.0.21:8080
```

然后，需要在 DUSTIN-SVC-HTTP 链中添加另一个规则，以随机跳转到刚创建的 DUSTIN-SEP-HTTP2 链。可以通过运行以下命令添加此规则：

```
sudo iptables \
  --table nat \
  --insert DUSTIN-SVC-HTTP 1 \
  --match statistic \
  --mode random \
  --probability 0.5 \
  --jump DUSTIN-SEP-HTTP2
```

请务必注意，我们在 DUSTIN-SVC-HTTP 链中把此规则插到了首位。**iptables 会按顺序尝试规则**。因此，这个规则在首位的话，我们就有 50% 的机会跳到该链上。如果成功，iptables 将跳至 DUSTIN-SEP-HTTP2。如果失败，iptables 将转到下一条规则，该规则将始终跳转到 DUSTIN-SEP-HTTP1。

常见的误解是每个规则的概率应为 50%，但这会在以下情况下引发问题：

- iptables 查看第一个规则（跳转至 DUSTIN-SEP-HTTP2），假设失败了；
- iptables 查看下一条规则（跳转至 DUSTIN-SEP-HTTP1），假设也失败了；
- 现在，我们的虚拟 IP 不会定向到任何后端！因此，概率是基于剩余可选择的后端数量的。如果要插入第三个后端，则该规则的概率为 33%。

不管怎样，如果我们随后运行以下命令：

```
curl 10.100.100.100:8080
```

可以看到请求会随机发送到运行在 netns_leah 和 netns_dustin 网络命名空间中的 python HTTP 服务器。这样我们算是通过 iptables 来进行负载均衡了！

原文链接：https://dustinspecker.com/posts/iptables-how-kubernetes-services-direct-traffic-to-pods/

END