

Calico网络的原理、组网方式与使用

(lijiaocn 2017/04/11 10:58:34)

- calico
 - 名词解释
 - 组网原理
 - BGP与AS
 - BGP Speaker 全互联模式 (node-to-node mesh)
 - BGP Speaker RR模式
- calico网络的部署
 - 在二层 (Ethernet) 网络中部署calico网络
 - 在三层 (IP) 网络中部署calico网络
 - AS per rack
 - AS per server
 - 优化：“Downward Default model” 减少需要记录的路由
- calico系统结构
- calico中的概念
 - bgpPeer
 - ipPool
 - node
 - policy
 - profile
 - workloadEndpoint
 - hostEndpoint
- node的报文处理过程
 - 路由决策之前：流入node的报文的处理
 - 进入raw表
 - 进入nat表
 - 路由决策之后：发送到本node的host endpoint 和 workload endpoint
 - 进入filter表
 - 来自其它node的报文
 - 来自本node上workload endpoint的报文
 - 路由决策之后：需要转发的报文
 - node发送本地发出的报文
- calico系统的部署
 - CentOS上安装
 - 安装calicectl
 - 安装felix
 - 二进制安装
 - 容器的方式
- calico的使用
 - node管理
 - 运行时设置
 - 创建/查看/更新/删除资源
 - IP地址管理
- 测试环境
 - 查看状态
 - 模拟一个租户网络
 - endpoints
 - policy
 - profile
- 参考

calico

calico是一个比较有趣的虚拟网络解决方案，完全利用路由规则实现动态组网，通过BGP协议通告路由。

calico的好处是endpoints组成的网络是单纯的三层网络，报文的流向完全通过路由规则控制，没有overlay等额外开销。

calico的endpoint可以漂移，并且实现了acl。

calico的缺点是路由的数目与容器数目相同，非常容易超过路由器、三层交换、甚至node的处理能力，从而限制了整个网络的扩张。

calico的每个node上会设置大量（海量）的iptables规则、路由，运维、排障难度大。

calico的原理决定了它不可能支持VPC，容器只能从calico设置的网段中获取ip。

calico目前的实现没有流量控制的功能，会出现少数容器抢占node多数带宽的情况。

calico的网络规模受到BGP网络规模的限制。

名词解释

endpoint：接入到calico网络中的网卡称为endpoint

AS：网络自治系统，通过BGP协议与其它AS网络交换路由信息

ibgp：AS内部的BGP Speaker，与同一个AS内部的ibgp、ebgp交换路由信息。

ebgp：AS边界的BGP Speaker，与同一个AS内部的ibgp、其它AS的ebgp交换路由信息。

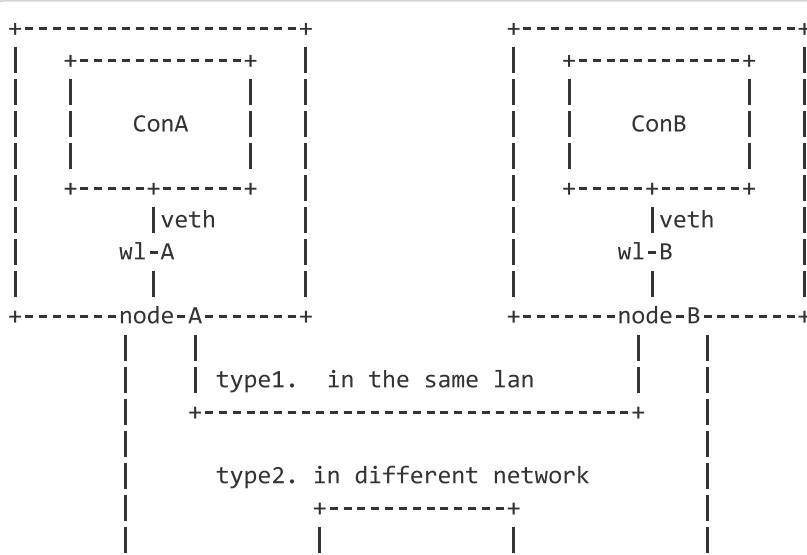
workloadEndpoint：虚拟机、容器使用的endpoint

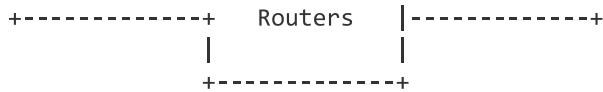
hostEndpoints：物理机(node)的地址

组网原理

calico组网的核心原理就是IP路由，每个容器或者虚拟机会分配一个workload-endpoint(wl)。

从nodeA上的容器A内访问nodeB上的容器B时：





从ConA中发送给ConB的报文被nodeA的w1-A接收，根据nodeA上的路由规则，经过各种iptables规则后，转发到nodeB。

如果nodeA和nodeB在同一个二层网段，下一条地址直接就是node-B，经过二层交换机即可到达。

如果nodeA和nodeB在不同的网段，报文被路由到下一跳，经过三层交换或路由器，一步步跳转到node-B。

核心问题是，nodeA怎样得知下一跳的地址？答案是node之间通过BGP协议交换路由信息。

每个node上运行一个软路由软件bird，并且被设置成BGP Speaker，与其它node通过BGP协议交换路由信息。

可以简单理解为，每一个node都会向其它node通知这样的信息：

我是X.X.X.X，某个IP或者网段在我这里，它们的下一跳地址是我。

通过这种方式每个node知晓了每个workload-endpoint的下一跳地址。

BGP与AS

BGP是路由器之间的通信协议，主要用于AS (Autonomous System, 自治系统) 之间的互联。

AS是一个自治的网络，拥有独立的交换机、路由器等，可以独立运转。

每个AS拥有一个全球统一分配的16位的ID号，64512到65535共1023个AS号码可以用于私有网络。

calico默认使用的AS号是64512，可以修改：

```

calicoctl config get asNumber      //查看
calicoctl config set asNumber 64512 //设置

```

AS内部有多个BGP speaker，分为ibgp、ebgp，ebgp与其它AS中的ebgp建立BGP连接。

AS内部的BGP speaker通过BGP协议交换路由信息，最终每一个BGP speaker拥有整个AS的路由信息。

BGP speaker一般是网络中的物理路由器，可以形象的理解为：

calico将node改造成了一个软路由器（通过软路由软件bird）
node上的运行的虚拟机或者容器通过node与外部沟通

AS内部的BGP Speaker之间有两种互联方式：

Mesh: BGP Speaker之间全互联，网络成网状

RR: Router reflection模式，BGP Speaker连接到一个或多个中心BGP Speaker，网络成星状

BGP Speaker 全互联模式(node-to-node mesh)

全互联模式，就是一个BGP Speaker需要与其它所有的BGP Speaker建立bgp连接(形成一个bgp mesh)。

网络中bgp总连接数是按照 $O(n^2)$ 增长的，有太多的BGP Speaker时，会消耗大量的连接。

calico默认使用全互联的方式，扩展性比较差，只能支持小规模集群：

```
say 50 nodes - although this limit is not set in stone and  
Calico has been deployed with over 100 nodes in a full mesh topology
```

可以打开/关闭全互联模式：

```
calicoctl config set nodeTonodeMesh off  
calicoctl config set nodeTonodeMesh on
```

BGP Speaker RR模式

RR模式，就是在网络中指定一个或多个BGP Speaker作为Router Reflection，RR与所有的BGP Speaker建立BGP连接。

每个BGP Speaker只需要与RR交换路由信息，就可以得到全网路由信息。

RR则必须与所有的BGP Speaker建立BGP连接，以保证能够得到全网路由信息。

在calico中可以通过Global Peer实现RR模式。

Global Peer是一个BGP Speaker，需要手动在calico中创建，所有的node都会与Global peer建立BGP连接。

```
A global BGP peer is a BGP agent that peers with every calico node in the network.  
A typical use case for a global peer might be a mid-scale deployment where all of  
the calico nodes are on the same L2 network and are each peering with the same Route  
Reflector (or set of Route Reflectors).
```

关闭了全互联模式后，再将RR作为Global Peers添加到calico中，calico网络就切换到了RR模式，可以支撑容纳更多的node。

calico中也可以通过node Peer手动构建BGP Speaker（也就是node）之间的BGP连接。

node Peer就是手动创建的BGP Speaker，只有指定的node会与其建立连接。

```
A BGP peer can also be added at the node scope, meaning only a single specified node  
will peer with it. BGP peer resources of this nature must specify a node to inform  
calico which node this peer is targeting.
```

因此，可以为每一个node指定不同的BGP Peer，实现更精细的规划。

例如当集群规模进一步扩大的时候，可以使用AS Per Pack model：

```
每个机架是一个AS  
node只与所在机架TOR交换机建立BGP连接  
TOR交换机之间作为各自的ebgp全互联
```

calico网络的部署

calico网络对底层的网络的要求很少，只要求node之间能够通过IP联通。

Any technology that is capable of transporting IP packets can be used as the interconnect fabric in a Calico network.

在calico中，全网路由的数目和endpoints的数目一致，通过为node分配网段，可以减少路由数目，但不会改变数量级。

如果有1万个endpoints，那么就至少要有一台能够处理1万条路由的设备。

无论用哪种方式部署始终会有一台设备上存放着calico全网的路由。

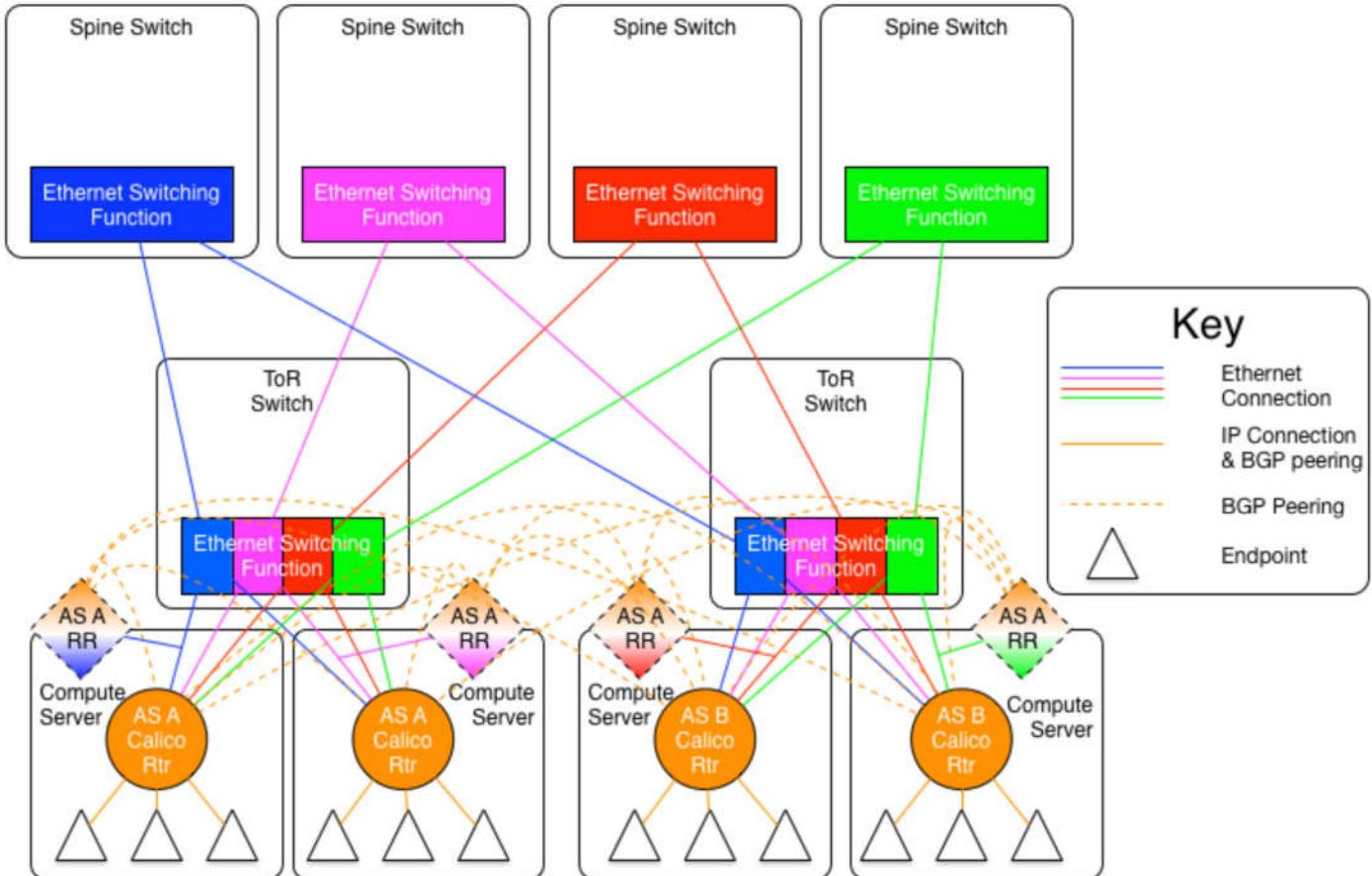
当要部署calico网络的时候，第一步就是要确认，网络中处理能力最强的设备最多能设置多少条路由。

在二层(Ethernet)网络中部署calico网络

calico over an Ethernet interconnect fabric中介绍了在二层网络中部署calico网络方案。

在二层网络中部署calico网络的意思是：

所有的node都已经接入了二层网，还没有配置三层网络。



为了保证链路可靠，图中设计了四个并列的二层网，每个二层网占用一个三层网段。

每个node同时接入四个二层网络，对应拥有四个不同网段的IP。

在每个二层网络中，node与node之间用RR模式建立BGP通信链路：

一个node做为RR，其余的node连接到做为RR的node
整个网络中最终有四个RR，分别负责四个网络中的BGP

当从node上去访问另一个node上的endpoint的时候，会有四条下一跳为不同网段的等价路由。

根据ECMP协议，报文将会平均分配给这四个等价路由，提高了可靠性的同时增加了网络的吞吐能力。

为每个二层网络设置了IP之后，二层网络就成为了三层网络，每个二层网络中的calico的部署也可以参考下一节。

在三层(IP)网络中部署calico网络

在三层网络中部署calico网络的意思是，calico over ip fabrics:

所有的node已经接入到了一个三层网络中，在此基础上部署calico网络。

已经部署好的三层网络应当是可靠的，calico可以直接在上面部署。

剩下的关键点就是怎样设计BGP网络，calico over ip fabrics中给出两种设计方式：

1. AS per rack: 每个rack(机架)组成一个AS，每个rack的TOR交换机与核心交换机组成一个AS
2. AS per server: 每个node做为一个AS，TOR交换机组成一个transit AS

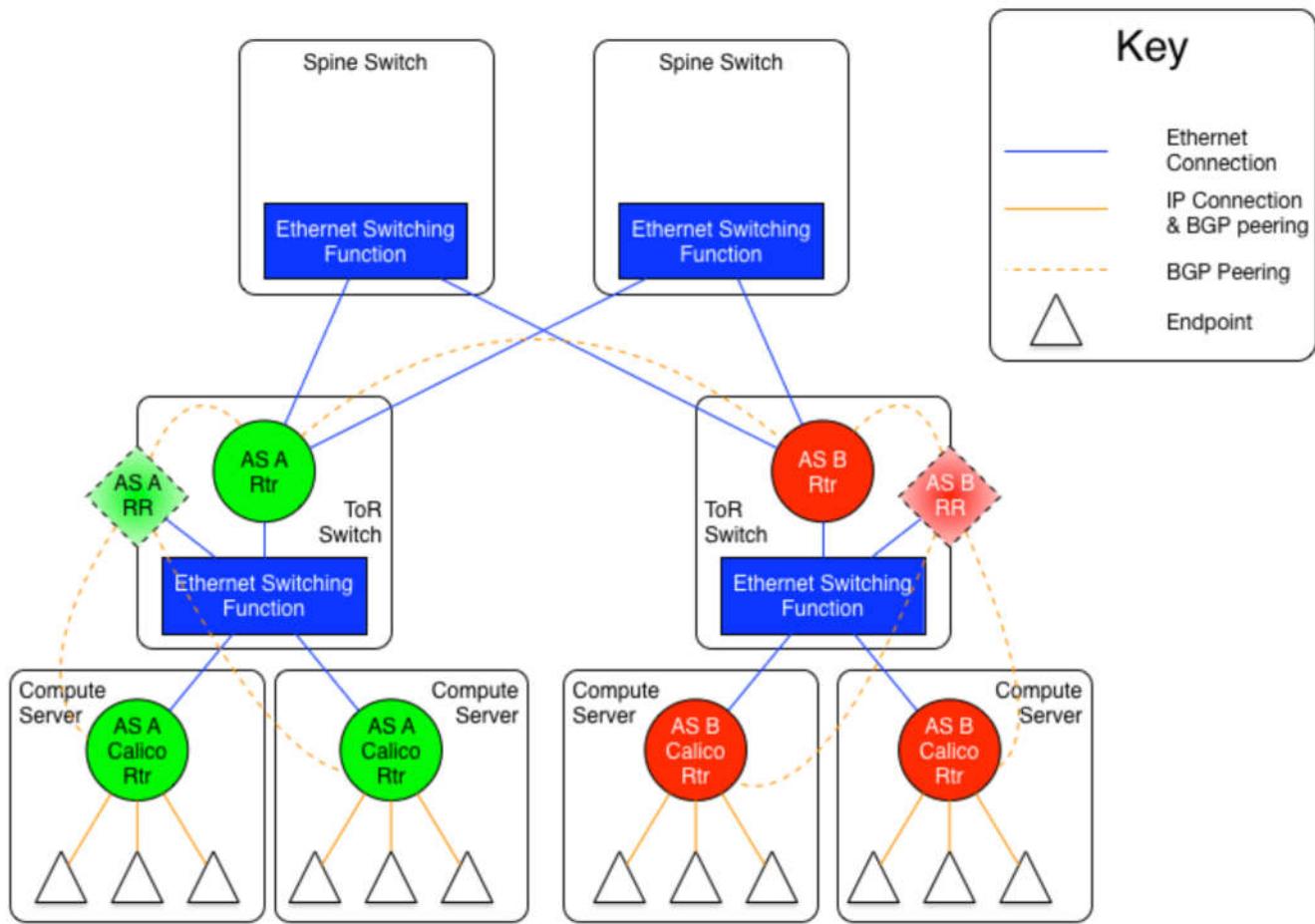
这两种方式采用的是Use of BGP for routing in large-scale data centers中的建议。

AS per rack

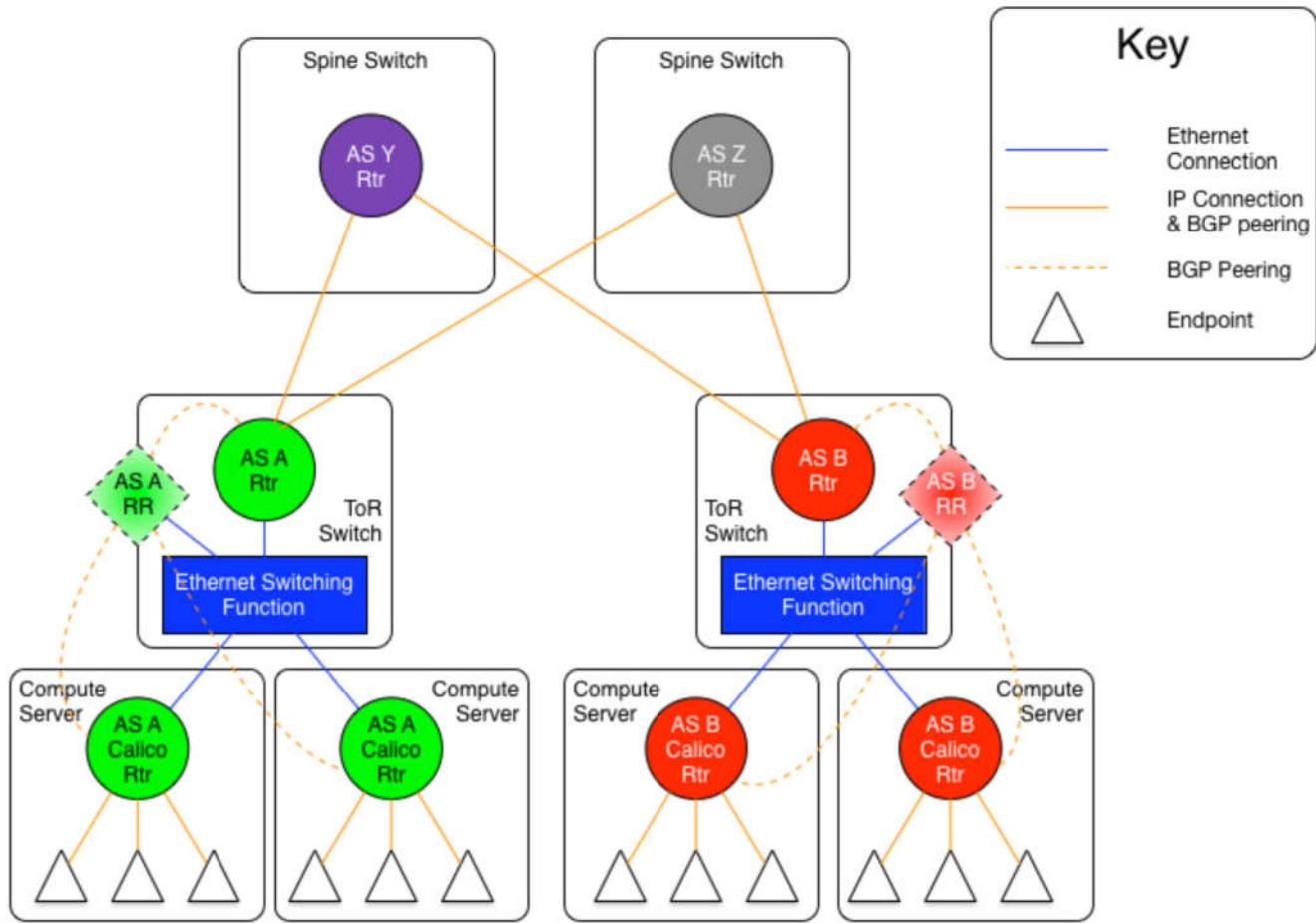
1. 一个机架作为一个AS，分配一个AS号，node是ibgp，TOR交换机是ebgp
2. node只与TOR交换机建立BGP连接，TOR交换机与机架上的所有node建立BGP连接
3. 所有TOR交换机之间以node-to-node mesh方式建立BGP连接

TOR交换机之间可以是接入到同一个核心交换机二层可达的，也可以只是IP可达的。

TOR二层联通：



TOR三层联通：



每个机架上node的数目是有限的，BGP压力转移到了TOR交换机。当机架数很多，TOR交换机组成BGP mesh压力会过大。

endpoints之间的通信过程：

EndpointA发出报文 --> nodeA找到了下一跳地址nodeB --> 报文送到TOR交换机A --> 报文送到核心交换机

EndpointB收到了报文 <-- nodeB收到了报文 <-- TOR交换机B收到了报文 <-- 核心交换机将报文送达TOR交换机B

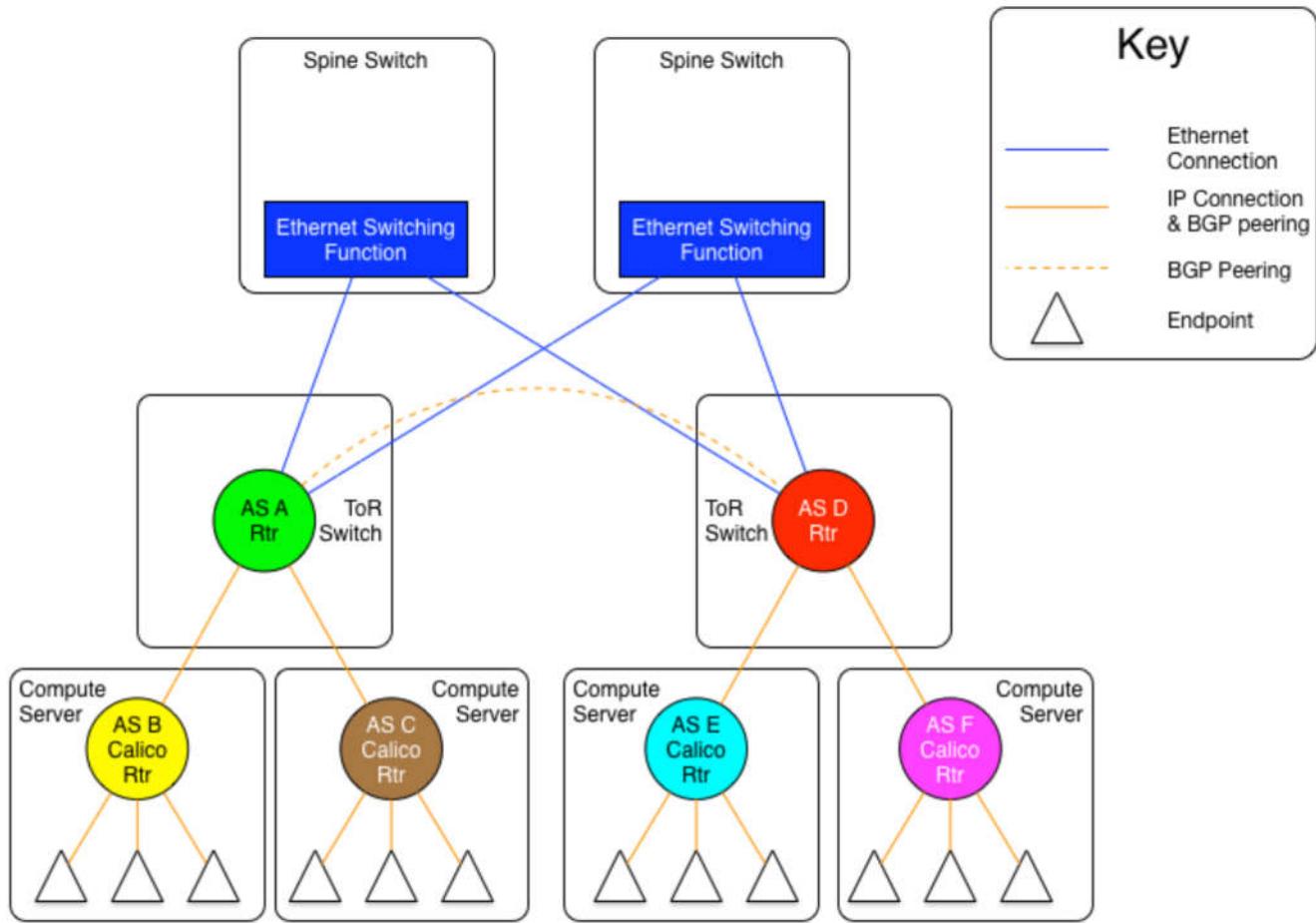
AS per server

1. 每个TOR交换机占用一个AS
2. 每个node占用一个AS
3. node与TOR交换机交换BGP信息
3. 所有的TOR交换机组成BGP mesh，交换BGP信息

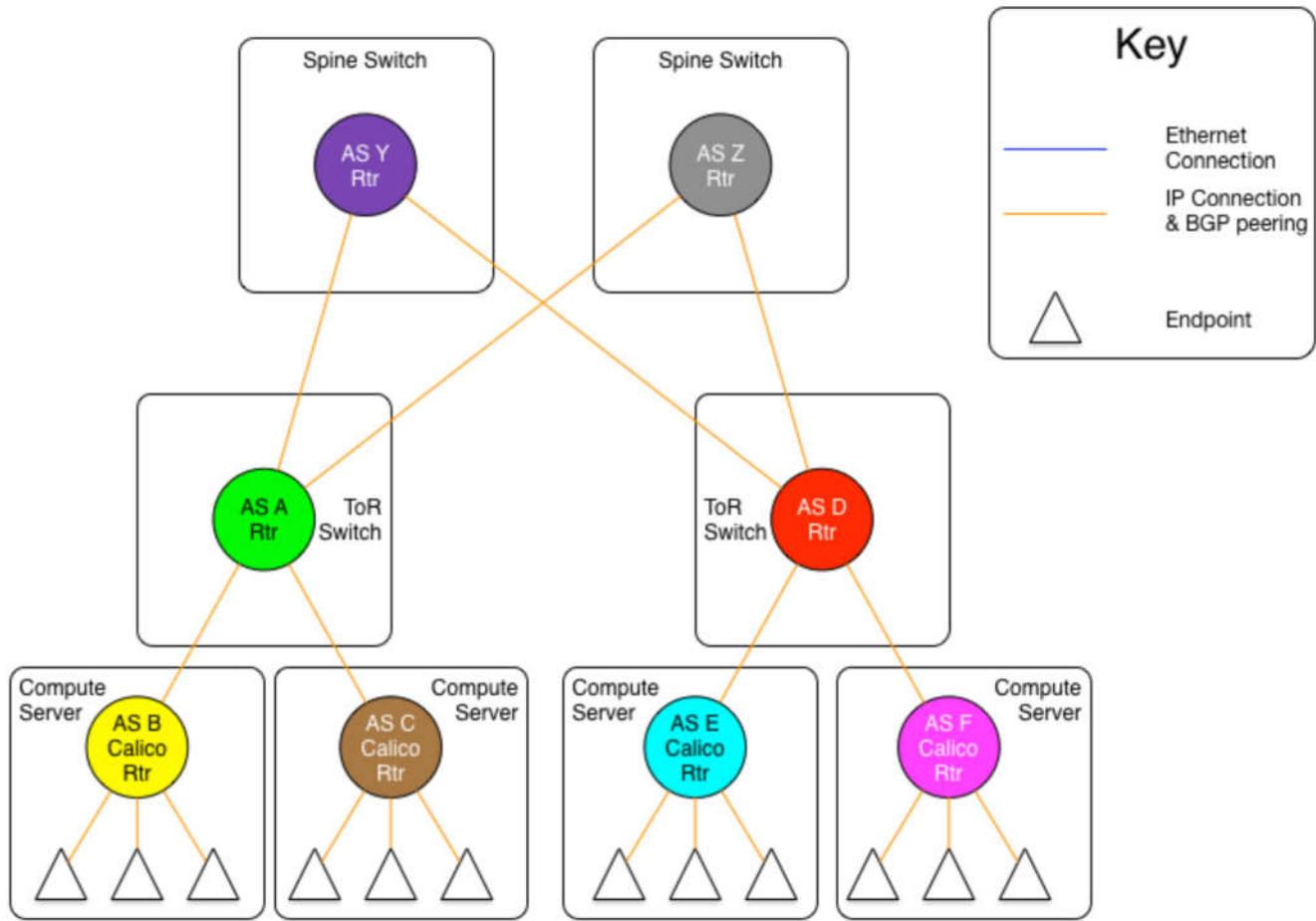
这种方式消耗了大量的AS，RFC 4893 – BGP Support for Four-octet AS Number Space中考虑将AS号增加到32位。

不是特别明白这种方式的好处在哪里。

TOR二层联通：



TOR三层联通：



优化：“Downward Default model”减少需要记录的路由

Downward Default Model在上面的几种组网方式的基础上，优化了路由的管理。

在上面的三种方式中，每个node、每个TOR交换机、每个核心交换机都需要记录全网路由。

“Downward Default model”模式中：

1. 每个node向上(TOR)通告所有路由信息，而TOR向下(node)只通告一条默认路由
2. 每个TOR向上(核心交换机)通告所有路由，核心交换机向下(TOR)只通告一条默认路由
3. node只知道本地的路由
4. TOR只知道接入到自己的所有node上的路由
5. 核心交换机知晓所有的路由

这种模式减少了TOR交换机和node上的路由数量，但缺点是，发送到无效IP的流量必须到达核心交换机以后，才能被确定为无效。

endpoints之间的通信过程：

EndpointA发出报文 --> nodeA默认路由到TOR交换机A --> TOR交换机A默认路由到核心交换机 ---+ |

EndpointB收到了报文 <-- nodeB收到了报文 <-- TOR交换机B收到了报文 <-- 核心交换机找到了下一跳地址nodeB

calico系统结构

calico系统组成：

- ```
1. Felix, the primary calico agent that runs on each machine that hosts endpoints.
2. etcd, the data store.
3. BIRD, a BGP client that distributes routing information.
4. BGP Route Reflector (BIRD), an optional BGP route reflector for higher scale.
5. The Orchestrator plugin, orchestrator-specific code that tightly integrates calico into that orchestrator
```

Felix负责管理设置node，

bird是一个开源的软路由，支持多种路由协议。

## calico中的概念

calicoctl resource definitions介绍了每类资源的格式。

### bgpPeer

```
apiVersion: v1
kind: bgpPeer
metadata:
 scope: node
 node: rack1-host1
 peerIP: 192.168.1.1
spec:
 asNumber: 63400
```

bgpPeer的scope可以是node、global。

### ipPool

```
apiVersion: v1
kind: ipPool
metadata:
 cidr: 10.1.0.0/16
spec:
 ipip:
 enabled: true
 mode: cross-subnet
 nat-outgoing: true
 disabled: false
```

### node

```
apiVersion: v1
kind: node
metadata:
 name: node-hostname
spec:
 bgp:
 asNumber: 64512
 ipv4Address: 10.244.0.1/24
 ipv6Address: 2001:db8:85a3::8a2e:370:7334/120
```

## policy

A Policy resource (policy) represents an ordered set of rules which are applied to a collection of endpoints which match a label selector.

Policy resources can be used to define network connectivity rules between groups of calico endpoints and host endpoints, and take precedence over Profile resources if any are defined.

```
apiVersion: v1
kind: policy
metadata:
 name: allow-tcp-6379
spec:
 selector: role == 'database'
 ingress:
 - action: allow
 protocol: tcp
 source:
 selector: role == 'frontend'
 destination:
 ports:
 - 6379
 egress:
 - action: allow
```

## profile

A Profile resource (profile) represents a set of rules which are applied to the individual endpoints to which this profile has been assigned.

```
apiVersion: v1
kind: profile
metadata:
 name: profile1
 labels:
 profile: profile1
spec:
 ingress:
 - action: deny
 source:
 net: 10.0.20.0/24
 - action: allow
 source:
 selector: profile == 'profile1'
 egress:
 - action: allow
```

## workloadEndpoint

A Workload Endpoint resource (workloadEndpoint) represents an interface connecting a calico networked container or VM to its host.

```

apiVersion: v1
kind: workloadEndpoint
metadata:
 name: eth0
 workload: default.frontend-5gs43
 orchestrator: k8s
 node: rack1-host1
 labels:
 app: frontend
 calico/k8s_ns: default
spec:
 interfaceName: cali0ef24ba
 mac: ca:fe:1d:52:bb:e9
 ipNetworks:
 - 192.168.0.0/16
 profiles:
 - profile1

```

## hostEndpoint

```

apiVersion: v1
kind: hostEndpoint
metadata:
 name: eth0
 node: myhost
 labels:
 type: production
spec:
 interfaceName: eth0
 expectedIPs:
 - 192.168.0.1
 - 192.168.0.2
 profiles:
 - profile1
 - profile2

```

## node的报文处理过程

报文处理过程中使用的标记位：

一共使用了3个标记位，`0x70000000`对应的标记位  
`0x10000000`: 报文的处理动作，置1表示放行，默认0表示拒绝  
`0x20000000`: 是否已经经过了policy规则检测，置1表示已经过  
`0x40000000`: 报文来源，置1，表示来自host-endpoint

流入报文来源：

1. 以cali+命名的网卡收到的报文，这部分报文是node上的endpoint发出的  
 (k8s中，容器内发出的所有报文都会发送到对应的cali网卡上)  
 (通过在容器内添加静态arp，将容器网关的IP映射到cali网卡的MAC上实现)
2. 其他网卡接收的报文，这部分报文是其它node发送或者在node本地发出的

流入的报文去向：

1. 访问本node的host endpoint，通过INPUT过程处理
2. 访问本node的工作负载endpoint，通过INPUT过程处理

3. 访问其它node的host endpoint，通过FORWARD过程处理。

4. 访问其它node的工作负载endpoint，通过FORWARD过程处理。

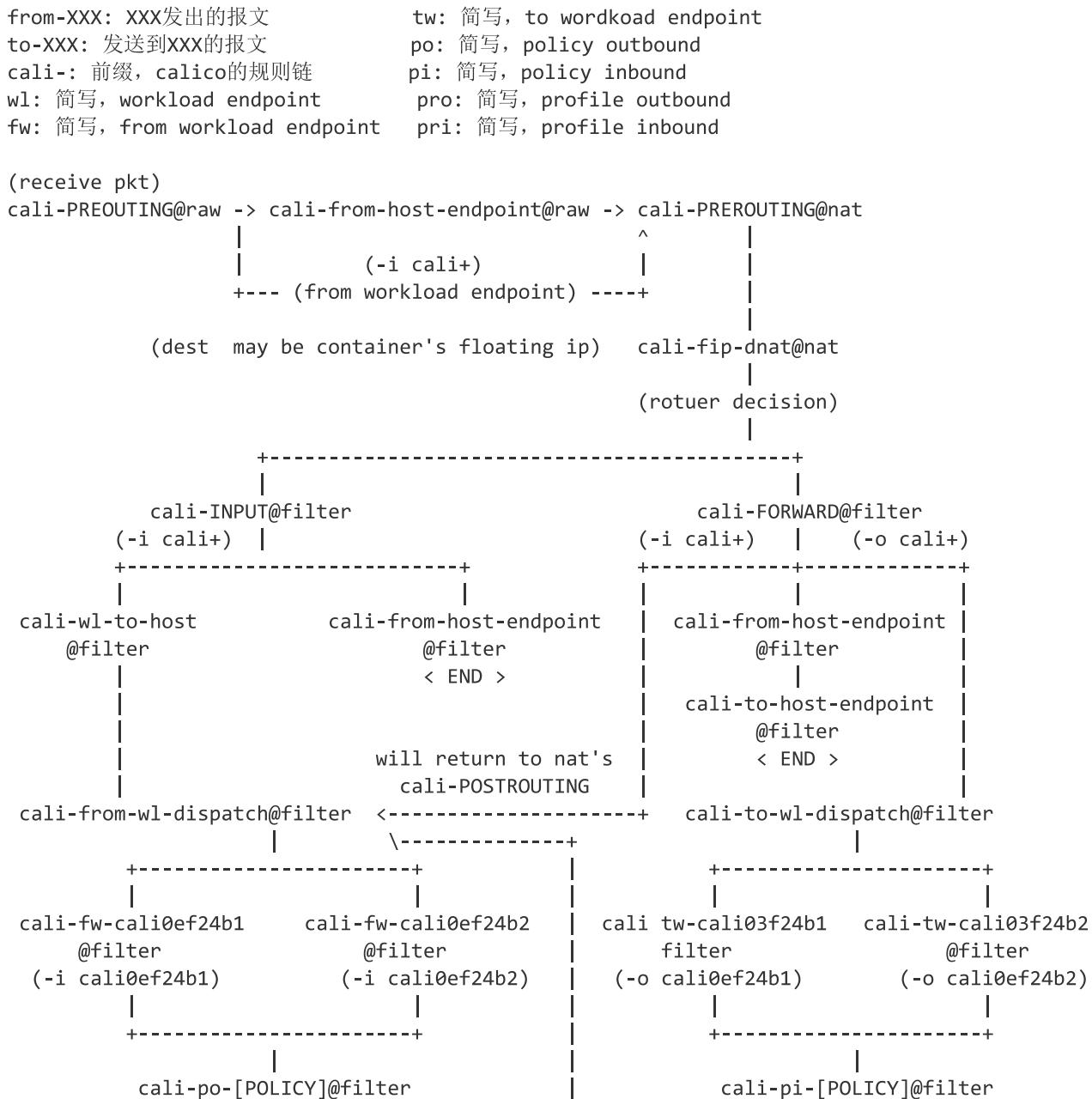
流入的报文在路由决策之前的处理过程相同的，路由决策之后，分别进入INPUT规则链和FORWARD链。

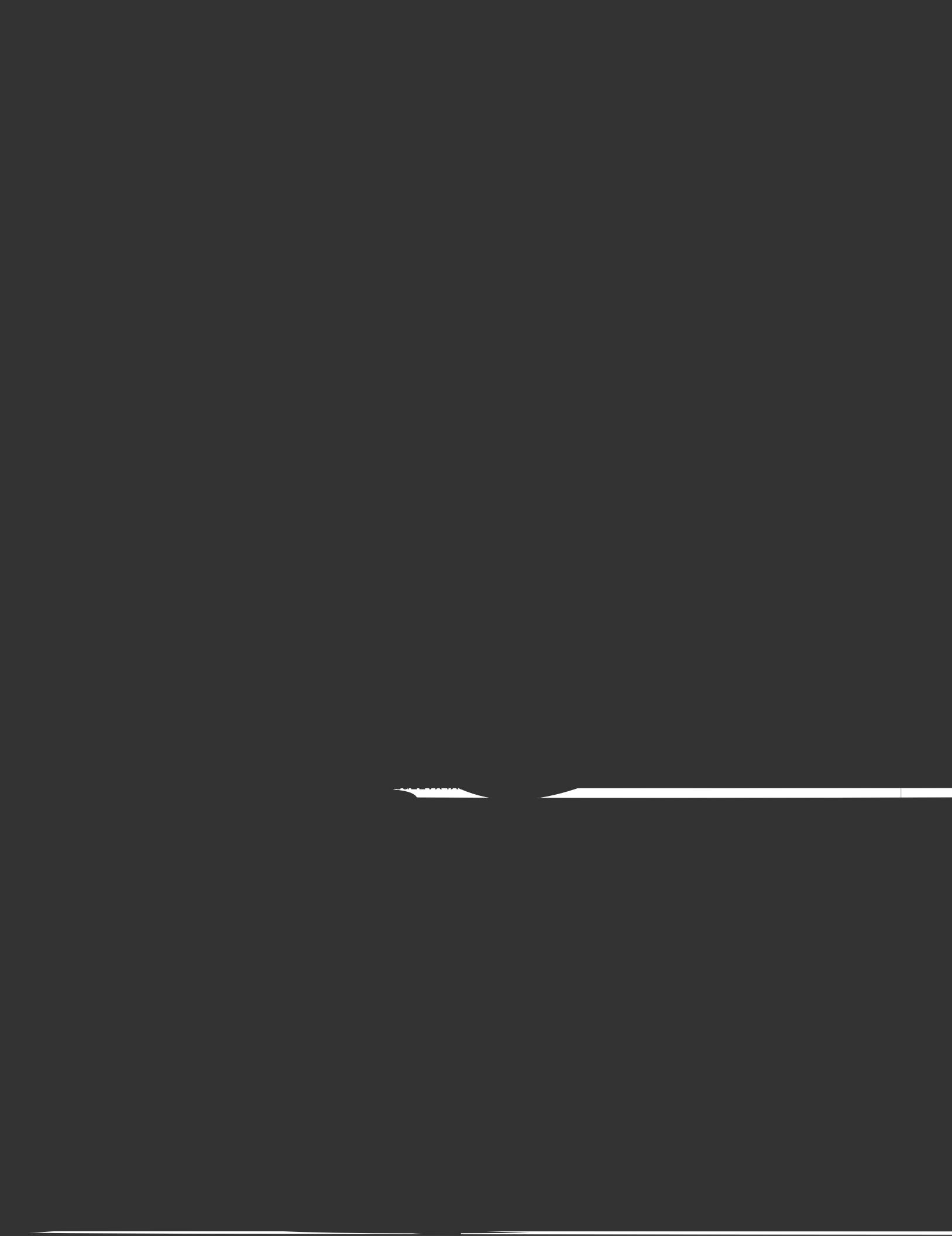
```
raw.PREROUTING -> mangle.PREROUTING -> nat.PREROUTING -> mangle.INPUT -> filter.INPUT
raw.PREROUTING -> mangle.PREROUTING -> nat.PREROUTING -> mangle.FORWARD -> filter.FORWARD -> mangle.POSTROUTING
```

这里分析的calico的版本比较老，和最新版中的规则有一些出入，但是原理相同。

新版本的calico的iptables规则可读性更好，可以直接阅读规则。

报文处理流程（全）：





cali-PREROUTING@nat:

```
-A cali-PREROUTING -m comment --comment "cali:r6XmIziWUJsdOK6Z" -j cali-fip-dnat
```

如果目标地址是fip(floating IP), 会在cali-fip-dnat中做dnat转换

nat表中做目的IP转换, 这里没有设置, 所以cali-fip-dnat是空的。

经过nat表之后, 会进行路由决策:

1. 如果是发送给slave1的报文, 经过规则链: INPUT@mangle、INPUT@filter
2. 如果不是发送给slave1报文, 经过规则链: FORWARD@mangle、FORWARD@filer、POSTROUTING@mangle、POSTROUTING@nat

路由决策之后: 发送到本node的host endpoint 和 workload endpoint  
进入filter表

INPUT@filter:

```
-A INPUT -m comment --comment "cali:Cz_uIIQiXIMmKD4c" -j cali-INPUT
```

直接进入cali-INPUT

cali-INPUT@filter:

```
-A cali-INPUT -m comment --comment "cali:46gVAqzWLjh8U402" -m mark --mark 0x1000000/0x1000000 -m conntrack --ctstate NEW -j ACCEPT
-A cali-INPUT -m comment --comment "cali:5M2EkEm-RV1DLAfe" -m conntrack --ctstate INVALID -j DROP
-A cali-INPUT -m comment --comment "cali:8ggYjLbFRX5Ap9Zj" -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A cali-INPUT -i cali+ -m comment --comment "cali:mA3ZJKi9nadUmYVF" -g cali-wl-to-host

-A cali-INPUT -m comment --comment "cali:hI4IjfGj0fegLPE" -j MARK --set-xmark 0x0/0x7000000
-A cali-INPUT -m comment --comment "cali:wdegoKfP1cmsZTOM" -j cali-from-host-endpoint
-A cali-INPUT -m comment --comment "cali:r875VVc8vFk1f-ZA" -m comment --comment "Host endpoint policy accept"
```

规则4, 从cali+网卡进入的报文, 进入wl-to-host的规则链, wl是workload的缩写

规则6, 非cali+网卡收到的报文, host-endpoint的规则链

来自其它node的报文

这里没有对host endpoint设置规则, 所以规则链时空

cali-from-host-endpoint@filter:

空

来自本node上workload endpoint的报文

检察一下是否允许workload endpoint发出这些报文。

cali-wl-to-host@filter:

```
-A cali-wl-to-host -p udp -m comment --comment "cali:aEOMPPLgak2S0Lxs" -m multiport --sports 68 -m multiport --dports 68 -j ACCEPT
-A cali-wl-to-host -p udp -m comment --comment "cali:SzR8ejPiuxtFMS8B" -m multiport --dports 53 -j ACCEPT
```

```
-A cali-wl-to-host -m comment --comment "cali:MEmlbCdco0Fefcrw" -j cali-from-wl-dispatch
-A cali-wl-to-host -m comment --comment "cali:Q2b2iY2M-vmds5iY" -m comment --comment "Configured Default
```

规则1，允许请求DHCP

规则2，允许请求DNS

规则3，匹配wan load endpoint各自的规则，将会依次检查policy的egress、各自绑定的profile的egress。

根据接收报文的网卡做区分 cali-from-wl-dispatch@filter:

```
-A cali-from-wl-dispatch -i cali-ef14b1 -m comment --comment "cali:RkM6MKQgU0OTxwKU" -g cali-fw-cali@e2
-A cali-from-wl-dispatch -i cali-ef14b1 -m comment --comment "cali:RkM6MKQgU0OTxwKU" -g cali-fw-cali@e2
```

```
-A cali-pro-profile-database -m comment --comment "cali:laSwzk9Ihy5ArWJB" -j MARK --set-xmark 0x1000000
-A cali-pro-profile-database -m comment --comment "cali:BpvFNyMPRLC0lDtu" -m mark --mark 0x1000000/0x1000000
```

profile-database的egress是allow, 直接打标记0x1000000/0x1000000。

## 路由决策之后：需要转发的报文

filter.FORWARD:

```
-A FORWARD -m comment --comment "cali:wUHhoiAYhph09Mso" -j cali-FORWARD
```

直接进入cali-FROWARD

filter.cali-FORWARD, 根据接收网卡做egress规则匹配, 根据目标网卡做ingress规则匹配:

```
-A cali-FORWARD -m comment --comment "cali:jxvuJjmmRV135nVu" -m mark --mark 0x1000000/0x1000000 -m conntrack
-A cali-FORWARD -m comment --comment "cali:8YeDX9Z0tXy00Sp8" -m conntrack --ctstate INVALID -j DROP
-A cali-FORWARD -m comment --comment "cali:1GMSV-PhhZ8QbJg4" -m conntrack --ctstate RELATED,ESTABLISHED
-A cali-FORWARD -i cali+ -m comment --comment "cali:36TkoGXj9EF7Plkv" -j cali-from-wl-dispatch
-A cali-FORWARD -o cali+ -m comment --comment "cali:URMhBRo8ugd8J8Yx" -j cali-to-wl-dispatch

-A cali-FORWARD -i cali+ -m comment --comment "cali:FyhWsW08U3a5niLK" -j ACCEPT
-A cali-FORWARD -o cali+ -m comment --comment "cali:G655uIfZuidj1gAw" -j ACCEPT

-A cali-FORWARD -m comment --comment "cali:4GbueNC2iWajKnx0" -j MARK --set-xmark 0x0/0x7000000
-A cali-FORWARD -m comment --comment "cali:bq3wVY3mkXk96NQP" -j cali-from-host-endpoint
-A cali-FORWARD -m comment --comment "cali:G8sjbYXH5_QiYnB1" -j cali-to-host-endpoint
-A cali-FORWARD -m comment --comment "cali:wYFYRdMhtSYCqKNm" -m comment --comment "Host endpoint policy"
```

规则4, 报文是workload endpoint发出的, 过对应endpoint的规则的egress规则。

规则5, 报文要转发给本地的workload endpoint的, 过对应endpoint的ingress规则。

规则6, 规则7, 默认允许转发。

规则9, 报文是其它node发送过来的, 过host endpoint的ingress规则。

规则10, 报文要转发给host endpoint, 过host endpoint的egress规则。

filter.cali-from-wl-dispatch, 过对应endpoint的egress规则:

```
-A cali-from-wl-dispatch -i cali0ef24b1 -m comment --comment "cali:RkM6MKQgU00TxwKU" -g cali-fw-cali0ef24b1
-A cali-from-wl-dispatch -i cali0ef24b2 -m comment --comment "cali:7hIahXYNmY9JDFKG" -g cali-fw-cali0ef24b2
-A cali-from-wl-dispatch -m comment --comment "cali:YKcpfdGNZ1PwfGvt" -m comment --comment "Unknown interface"
```

规则1, 过对应endpoint的inbound规则, fw表示from workload

filter.cali-to-wl-dispatch, 过对应endpoint的ingress规则:

```
-A cali-to-wl-dispatch -o cali0ef24b1 -m comment --comment "cali:ofrbQ8PhcrIR6rgF" -g cali-tw-cali0ef24b1
-A cali-to-wl-dispatch -o cali0ef24b2 -m comment --comment "cali:l9Rs20XXI14D5AVE" -g cali-tw-cali0ef24b2
-A cali-to-wl-dispatch -m comment --comment "cali:dxGyc_mZA_GT16Wb" -m comment --comment "Unknown interface"
```

规则1, 过对应endpoint的规则链, tw表示to workload

workload endpoint的outbound规则，在前面已经看过了，这里省略，只看inbound。

查看一个workload-endpoint的inbound规则，filter.calitw-cali0ef24b1

```
-A calitw-cali0ef24b1 -m comment --comment "cali:v-IVzQuOaLDTv1KQ" -j MARK --set-xmark 0x0/0x1000000
-A calitw-cali0ef24b1 -m comment --comment "cali:vE8JWROTKOuSK0cA" -m comment --comment "Start of policy"
-A calitw-cali0ef24b1 -m comment --comment "cali:fV5z1nXaCLhF0EQ" -m mark --mark 0x0/0x2000000 -j cali
-A calitw-cali0ef24b1 -m comment --comment "cali:_B9yiomhSoQTzhKL" -m comment --comment "Return if policy"
-A calitw-cali0ef24b1 -m comment --comment "cali:uNPReN9_BghUJj7S" -m comment --comment "Drop if no policy"
```

首先过policy的ingress规则，然后过绑定的profile的ingress规则：

规则3：cali-pi-namespace-default，pi表示policy inbound。

filter.calitw-cali0ef24b1，policy inbound规则：

```
-A cali-pi-namespace-default -m comment --comment "cali:K4jTheFcVvdYaw0q" -j DROP
-A cali-pi-namespace-default -m comment --comment "cali:VTQ78plyA8u_8_YC" -m set --match-set cali4-s:CE
-A cali-pi-namespace-default -m comment --comment "cali:OAWI2ts9a8YpVP2b" -m mark --mark 0x1000000/0x1000000
```

注意，规则1直接丢弃了报文，但是规则2又在设置标记，这是因为这里policy的egress规则设置是有问题的：

```
ingress:
- action: deny
- action: allow
 source:
 selector: namespace == 'default'
```

配置了两条ingress规则，第一条直接deny，第二条则是对指定的source设置为allow。这样的规则配置是有问题的。

从上面的iptables规则中也可以看到，iptables规则是按照ingress中的规则顺序设定的。

如果第一条规则直接deny，那么后续的规则就不会发生作用了。

所以结果就是allow规则不生效。

slave1上的workload endpoint没有绑定profile，所有没有profile的inbound规则。

slave2上的endpoint设置了profile，允许访问TCP 3306端口，可以看到profile的inbound规则，

filter.calitw-cali0ef24b3：

```
-A calitw-cali0ef24b3 -m comment --comment "cali:-147AwgMbB6upZ-7" -j MARK --set-xmark 0x0/0x1000000
-A calitw-cali0ef24b3 -m comment --comment "cali:3qL17L7-k49jf6Eu" -m comment --comment "Start of policy"
-A calitw-cali0ef24b3 -m comment --comment "cali:Q6ycGZQm9W9l4KiJ" -m mark --mark 0x0/0x2000000 -j cali
-A calitw-cali0ef24b3 -m comment --comment "cali:_ILnIsDpaSEGOULc" -m comment --comment "Return if policy"
-A calitw-cali0ef24b3 -m comment --comment "cali:CtKc0QPXG9FZiCN-" -m comment --comment "Drop if no policy"
-A calitw-cali0ef24b3 -m comment --comment "cali:NR6mgOGAOw90NLpp" -j cali-pri-profile-database
-A calitw-cali0ef24b3 -m comment --comment "cali:_Oapak4JADerp4Fv" -m comment --comment "Return if profile"
-A calitw-cali0ef24b3 -m comment --comment "cali:ZVuAf3Bzin6dOKSX" -m comment --comment "Drop if no profile"
```

规则6，多出的profile inbound规则。

slave2上的profile的inbound规则，filter.calitw-cali0ef24b3：

```
-A cali-pri-profile-database -m comment --comment "cali:viAiQwvuZPt5-44a" -j DROP
-A cali-pri-profile-database -p tcp -m comment --comment "cali:Vcuflyj-wUF-f_Mo" -m set --match-set cali4-s:CE
-A cali-pri-profile-database -m comment --comment "cali:JWP_zDo3JNywNc0V" -m mark --mark 0x1000000/0x1000000
```

同样也是因为profile的ingress第一条是deny的原因  
规则2，允许访问tcp 3306。

nat. POSTROUTING:

```
-A cali-POSTROUTING -m comment --comment "cali:GJF4nAqRa" -j RETURN
-A cali-POSTROUTING -m comment --comment "cali:tVnHkv" -j RETURN
```

这里没有设置fip，所以cali-fip-snata和cali-nat-

## node发送本地发出的报文

OUTPUT@nat:

```
-A OUTPUT -m comment --comment "cali:tVnHkv" -j RETURN
-A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j RETURN
```

cali-OUTPUT@nat:

```
-A cali-OUTPUT -m comment --comment "cali:GJF4nAqRa" -j RETURN
```

OUTPUT@filter:

```
-A OUTPUT -m comment --comment "cali:tVnHkv" -j RETURN
```

cali-OUTPUT@filter:

```
-A cali-OUTPUT -m comment --comment "cali:F4nAqRa" -j RETURN
-A cali-OUTPUT -m comment --comment "cali:KuApYk" -m conntrack --ctstate INVALID -j DROP
-A cali-OUTPUT -m comment --comment "cali:TgS7" -m mark --mark 0x1000000/0x1000000 -m conntrack --ctstate RELATED,ESTABLISHED -j RETURN
-A cali-OUTPUT -o cali+ -m comment --comment "cali:0YpIH4BWIJL90Pfx" -j RETURN
-A cali-OUTPUT -m comment --comment "cali:sopoFnawuqGYyG" -j MARK --set-xmark 0x0/0x7000000
-A cali-OUTPUT -m comment --comment "cali:v"
```

的安装，这里不介绍。

具：

```
ectcalico/calicoctl/releases/download/v1.1.0/calicoctl
```

ks for a configuration file at /etc/calico/calicoctl.cfg:

```
"etcd1:2379,http://etcd2:2379"
```

使用环境变量，spec中的field与环境变量的对应关系：

Examples

Des

```
$rpm -ql calico-felix
/etc/calico/felix.cfg.example
/etc/logrotate.d/calico-felix
/usr/bin/calico-felix
/usr/lib/systemd/system/calico-felix.service

$rpm -ql calico-common
/usr/bin/calico-diags
/usr/bin/calico-gen-bird-conf.sh
/usr/bin/calico-gen-bird-mesh-conf.sh
/usr/bin/calico-gen-bird6-conf.sh
/usr/bin/calico-gen-bird6-mesh-conf.sh
/usr/share/calico/bird/calico-bird-peer.conf.template
/usr/share/calico/bird/calico-bird.conf.template
/usr/share/calico/bird/calico-bird6-peer.conf.template
/usr/share/calico/bird/calico-bird6.conf.template
```

## 容器的方式

默认使用的镜像quay.io/calico/node，但quay.io在国内被墙，用docker.io中的镜像代替：

```
docker pull docker.io/calico/node
```

启动felix：

```
calicoctl node run --node-image=docker.io/calico/node:latest
```

下面是启动过程中日志：

```
Running command to load modules: modprobe -a xt_set ip6_tables
Enabling IPv4 forwarding
Enabling IPv6 forwarding
Increasing conntrack limit
Removing old calico-node container (if running).
Running the following command to start calico-node:

docker run --net=host --privileged --name=calico-node -d --restart=always -e CALICO_NETWORKING_BACKEND=t
Image may take a short time to download if it is not available locally.
Container started, checking progress logs.

Skipping datastore connection test
Using autodetected IPv4 address on interface eth1: 192.168.40.2/24
No AS number configured on node resource, using global value
Created default IPv4 pool (192.168.0.0/16) with NAT outgoing enabled. IPIP mode: off
Created default IPv6 pool (fd80:24e2:f998:72d6::/64) with NAT outgoing enabled. IPIP mode: off
Using node name: compile
Starting libnetwork service
calico node started successfully
```

从日志中可以看到，容器使用的是host net、通过-e传入环境变量。

## calico的使用

在calico中，IP被称为Endpoint，宿主机上的容器IP称为workloadEndpoint，物理机IP称为hostEndpoint。ipPool等一同被作为资源管理。

查看默认的地址段：

```
./calicectl get ippool -o wide
CIDR NAT IPIP
192.168.0.0/16 true false
fd80:24e2:f998:72d6::/64 true false
```

## node管理

查看当前node是否满足运行calico的条件：

```
calicectl node <command> [<args>...]
run Run the calico node container image.
status View the current status of a calico node.
diags Gather a diagnostics bundle for a calico node.
checksystem Verify the compute host is able to run a calico node instance.
```

## 运行时设置

calicectl config更改calico的配置项。

## 创建/查看/更新/删除资源

分别使用creat/get/replace/delete来创建/查看/更新/删除资源。

创建资源：

```
calicectl create --filename=<FILENAME> [--skip-exists] [--config=<CONFIG>]
```

资源使用yaml文件描述，可以创建以下资源：

```
node //物理机
bgpPeer //与本机建立了bgp连接的node
hostEndpoint
workloadEndpoint
ipPool
policy
profile
```

查看资源：

```
calicectl get ([--scope=<SCOPE>] [--node=<NODE>] [--orchestrator=<ORCH>]
 [--workload=<WORKLOAD>] (<KIND> [<NAME>]) |
 --filename=<FILENAME>)
 [--output=<OUTPUT>] [--config=<CONFIG>]
```

可以通过下面命令查看所有资源：

```
calicoctl get [资源类型]
```

例如：

```
calicoctl get node
```

## IP地址管理

```
calicoctl ipam <command> [<args>...]
```

```
release Release a calico assigned IP address.
show Show details of a calico assigned IP address.
```

## 测试环境

三台机器：

```
etcd: 192.168.40.10:2379
slave1: 192.168.40.11
node2: 192.168.40.12
```

slave1和node2上的配置文件：

```
cat /etc/calico/calicoctl.cfg
apiVersion: v1
kind: calicoApiConfig
metadata:
spec:
 datastoreType: "etcdv2"
 etcdEndpoints: "http://192.168.40.10:2379"
```

安装启动etcd：

```
yum install -y etcd
systemctl start etcd
```

在slave1和slave2上安装calicoctl并启动：

```
yum install -y docker
systemctl start docker
docker pull docker.io/calico/node

wget https://github.com/projectcalico/calicoctl/releases/download/v1.1.0/calicoctl
chmod +x calicoctl

../calicoctl node run --node-image=docker.io/calico/node:latest
```

## 查看状态

```
$calicoctl get node
NAME
slave1
slave2

$calicoctl config get nodeTonodeMesh
```

```
on

$ calicoctl config get logLevel
info

$ calicoctl config get asNumber
64512

$ calicoctl config get ipip
off

$ calicoctl get bgpPeer
SCOPE PEERIP NODE ASN

$ calicoctl get ipPool
CIDR
172.16.1.0/24
fd80:24e2:f998:72d6::/64

$ calicoctl get workloadEndpoint
NODE ORCHESTRATOR WORKLOAD NAME
```

## 模拟一个租户网络

在名为” default ”的namespace中，创建两个” frontend ” 和” database ” 两个service。

“ frontend ” 有两个endpoint位于 slave1 上。

“ database ” 有一个endpoint位于 slave2 上。

为 namespace “ default ” 设置的默认策略是全互通的。

为” database ” 做了额外设置(“ profile ”)，只允许同一个namespace中的endpoint访问它的3306端口。

## endpoints

一个 endpoints 属于哪个 namespace 、哪个 service ，都是用 labels 标记的。 labels 是完全自定义的。

endpoints.yaml

```
- apiVersion: v1
kind: workloadEndpoint
metadata:
 name: slave1-frontend1
 workload: frontend
 orchestrator: k8s
 node: slave1
 labels:
 service: frontend
 namespace: default
spec:
 interfaceName: cali0ef24b1
 mac: ca:fe:1d:52:bb:e1
 ipNetworks:
 - 172.16.1.1
```

```

- apiVersion: v1
 kind: workloadEndpoint
 metadata:
 name: slave1-frontend2
 workload: frontend
 orchestrator: k8s
 node: slave1
 labels:
 service: frontend
 namespace: default
 spec:
 interfaceName: cali0ef24b2
 mac: ca:fe:1d:52:bb:e2
 ipNetworks:
 - 172.16.1.2
- apiVersion: v1
 kind: workloadEndpoint
 metadata:
 name: slave2-database
 workload: database
 orchestrator: k8s
 node: slave2
 labels:
 service: database
 namespace: default
 spec:
 interfaceName: cali0ef24b3
 mac: ca:fe:1d:52:bb:e3
 ipNetworks:
 - 172.16.1.3
 profiles:
 - profile-database

```

创建:

```
$ calicoctl create -f endpoints.yaml
Successfully created 3 'workloadEndpoint' resource(s)
```

查看:

```
$ calicoctl get workloadEndpoints -o wide
NODE ORCHESTRATOR WORKLOAD NAME NETWORKS NATS INTERFACE PROFILES
slave1 k8s frontend slave1-frontend1 172.16.1.1/32 cali0ef24b1
slave1 k8s frontend slave1-frontend2 172.16.1.2/32 cali0ef24b2
slave2 k8s database slave2-database 172.16.1.3/32 cali0ef24b3
```

## policy

为namespace” default” 设置的policy， namespace内部互通。

```

apiVersion: v1
kind: policy
metadata:
 name: namespace-default
spec:
 selector: namespace == 'default'
 ingress:
 - action: allow

```

```
source:
 selector: namespace == 'default'
egress:
- action: allow
```

## profile

为“service” database” 设置的profile，只允许访问3306端口。

```
apiVersion: v1
kind: profile
metadata:
 name: profile-database
 labels:
 profile: profile-database
spec:
 ingress:
 - action: deny <-- 这个规则是有问题的，第一条规则直接drop，就不会进入第二天规则了
 - action: allow 这里故意保留了这个有问题的设置，在下面分析时候，就会遇到这个问题的根源。
 source:
 selector: namespace == 'default' && service == 'frontend'
 ports:
 - int: 3306
 egress:
 - action: allow
```