Kubernetes ACL 设计文档

1. Kubernetes network policy

Kubernetes 自己定义了一套 network policy 用来控制和隔离 pod 与 pod 之间网络通信与访问。关于 network policy，这里有以下几点总结
https://kubernetes.io/docs/concepts/services-networking/network-policies/

1. Kubernetes 自己并没有实现 network policy 的具体功能，只是定义了一套标准。需要支持 network policy 的 network plugin 来实现。目前已知的支持 network policy 的 plugin 包括 calico，weave，kube-router 等
2. Network policy 的功能从 k8s1.5 开始进入 beta 版本，1.7 开始正式 GA，使用的时候需要注意区别不同版本的格式规范
3. Kubernetes 默认的网络策略是所有的 pod 和 pod 之间全连通。一旦一个 namespace 里面的 pod 使用了 network policy，将采用白名单策略。即拒绝在 network policy 规定范围以外的 traffic。

Network policy yaml example：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default    <--------------------------(1)
spec:
  podSelector:
    matchLabels:
      role: xx          <-----------------------------(2)
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: xxxx/xx    <----------------------------(3)
    - namespaceSelector:
        matchLabels:
          project: xxxx  <----------------------------(4)
    - podSelector:
        matchLabels:
          role: xxxx     <----------------------------(5)
    ports:
    - protocol: TCP
      port: xxxx
```

表 1：

2. iptables+ipset 实现 network policy 网络隔离

其实，我们可以用纯 iptables 的方式在 filter table 里面添加隔离规则来实现 network policy。但是，

1） 这样对每一个 pod 都添加隔离规则会导致 iptables 产生大量的规则。

2） network policy 针对 pod 的集合处理隔离，相同集合的 pod 处理的规则都是一致的。

所以我们在 iptables 的基础上对同一集合的 pod 用 ipset 的方式处理，可以避免产生大量规则。

对于表 1 的 ingress，如果我们想要隔离从（2）到（3）的 traffic，我们可以按照如下方式在 filter table 创建隔离规则：

```
for each ingress rule in policy:
  –A enn-dispatch-xxxx –m set –match-set $[pod_role_xx_set] dst –m
set ––match-set $[pod_namespace_xx_set] src –m $protocol –dport
$port –j ACCEPT
```

规则模版 1

由于 network policy 是白名单，所以不符合条件的 traffic 我们需要 reject

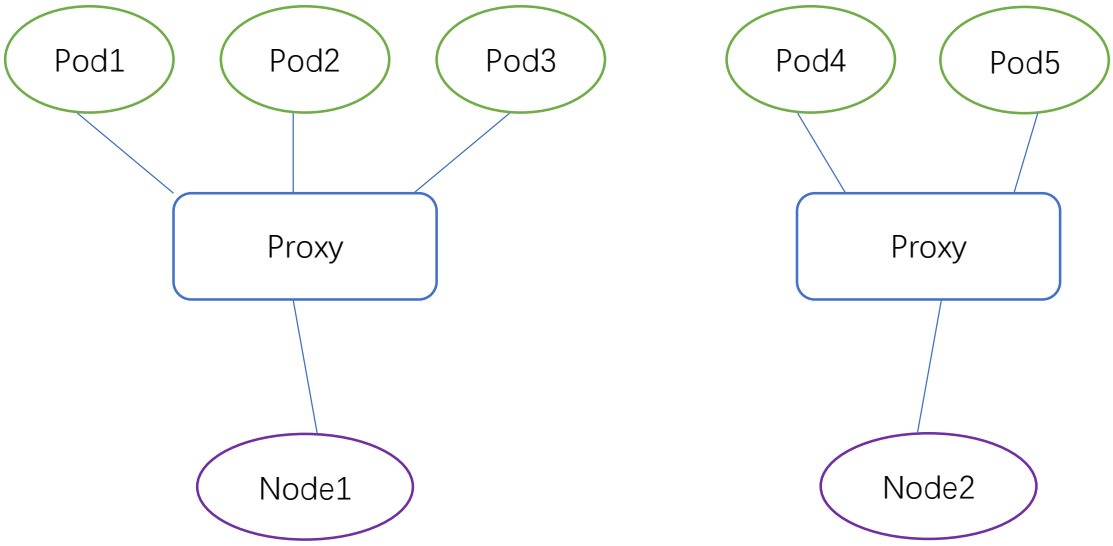
所以对于 `enn-dispatch-xxxx chain`，我们需要创建如下规则

```
–A enn-policy-A –m conntrack –ctstate RELATED,ESTABLISHED –j
ACCEPT
-A enn-policy-A –j enn-dispatch-xxxx (ipblock) (3)
-A enn-policy-A –j enn-dispatch-xxxx (namespace select) (4)
-A enn-policy-A –j enn-dispatch-xxxx (pod select) (5)
-A enn-policy-A –j reject (default reject)
```

规则模版 2

另外 对于 ingress 的 dst pod 也有两层过滤，即表 1 中的（1）（2）。其中（1）是针对 namespace 的隔离，（2）是具体选择 namespace 的 pod，如果（2）为空，则默认选择所有 namespace（1）中所有的 pod。针对（2）的隔离已经在规则模板 1 中的 `–m set –match-set $[pod_role_xx_set] dst` 体现。针对（1）的隔离规则，我们会在接下来的章节讨论。

3. Kubernetes pod 之间的访问方式



1） 同一个 node， pod1 直接访问 pod2

2） 同一个 node， pod1 通过 proxy 访问 pod3

3） 不同的 node， pod1 直接访问 pod4

4） 不同的 node， pod1 通过 proxy 访问 pod5

5） pod1 和外网的访问。

6）

其中

1）可以通过 forward chain 隔离（是否需要考虑 bridge？使用 PHYDEV module 的—phydev-is-bridged）

2）可以通过 OUTPUT chain 隔离 （both ingress & egress）

3，4，5） 可以通过 forward chain 隔离

所以，上一章节的规则需要走 OUTPUT，FORWARD 两个 chain，我们可以在这两个 chain 分别引入 enn-output 和 enn-forward 两个 chain，作为我们 ACL 隔离规则的入口：

```
FORWARD chain:
-A FORWARD –j enn-forward

OUTPUT chain:
-A OUTPUT –j enn-output
```

规则模板 3

根据表 1，进入 ACL 规则隔离入口之后，根据 pod 的 namespace 选择不同的分支（表 1（1）），所以我们需要根据 namespace 创建 hash:ip 为类型的 ipset，根据 ipset 的不同，进入不同的隔离分枝，以 ingree 为例：

```
-A enn-forward —m set —match-set $[ns1_pod_set] dst —j enn-
ingress-xx
-A enn-forward —m set —match-set $[ns2_pod_set] dst —j enn-
ingress-xx
-A enn-forward —m set —match-set $[ns3_pod_set] dst —j enn-
ingress-xx


-A enn-output —m set —match-set $[ns1_pod_set] dst —j enn-
ingress-xx
-A enn-output —m set —match-set $[ns2_pod_set] dst —j enn-
ingress-xx
-A enn-output —m set —match-set $[ns3_pod_set] dst —j enn-
ingress-xx
```

规则模板 4

还有一个可能需要考虑的问题：K8S network policy 采用白名单策略，所以 policy 将拒绝所有白名单以外的 traffic。但是 network policy 规则可以控制的 ip 范围只在集群内部。这意味着我们无法定义集群外部的 traffic 的规则。如果一个 pod 定义了 ingress 规则，那么集群外部的 ip 也无法访问它。同样，如果一个 pod 定义了 egress 规则，那么它也无法访问集群外部。所以，这里我在 `enn-ingress-xx` 和 `enn-policy-xx` 之间又定义了一层规则，引入 hash:net 为类型的 ipset（`pod_ip_range`）来隔离 policy 控制的 ip 范围，可以通过初始化传入 flag 参数或者定义特殊命名的 namespace labels 来控制 `pod_ip_range` 的范围。对于 `pod_ip_range` 以外的 traffic 一律 ACCEPT。`pod_ip_range` 的默认值为 0.0.0.0/0，即处理所有外网和内网 ip。

```
-A enn-ingress-xx —m set —match-set $[pod_ip_range] src —j enn-
policy-ixxx
-A enn-ingress-xx —j ACCEPT
```

规则模板 5

4. 规则设计模板：

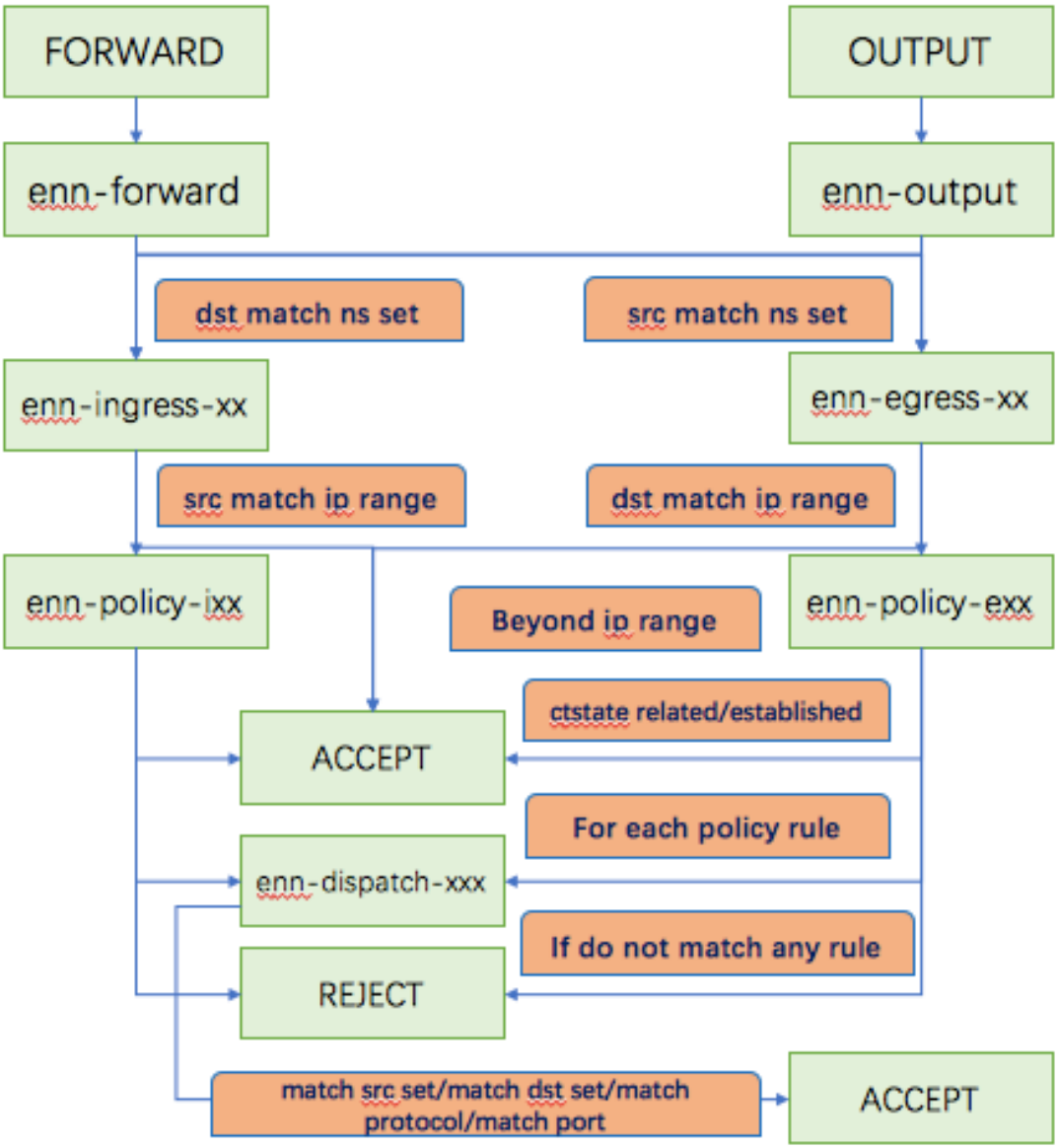


图 1：规则设计图总览

```
FORWARD chain:
-A FORWARD —j enn-forward

OUTPUT chain:
-A OUTPUT —j enn-output

enn-forward chain:
-A enn-forward —m set —match-set $[ns1_pod_set] dst —j enn-ingress-xx
-A enn-forward —m set —match-set $[ns2_pod_set] dst —j enn-ingress-xx
-A enn-forward —m set —match-set $[ns1_pod_set] src —j enn-egress-xx
-A enn-forward —m set —match-set $[ns2_pod_set] src —j enn-egress-xx


enn-output chain:
-A enn-output —m set —match-set $[ns1_pod_set] dst —j enn-ingress-xx
-A enn-output —m set —match-set $[ns2_pod_set] dst —j enn-ingress-xx
-A enn-output —m set —match-set $[ns1_pod_set] src —j enn-egress-xx
-A enn-output —m set —match-set $[ns2_pod_set] src —j enn-egress-xx

enn-ingress-xx chain:
-A enn-ingress-xx —m set —match-set $[pod_ip_range] src —j enn-policy-ixxx
-A enn-ingress-xx —j ACCEPT

enn-egress-xx chain:
-A enn-egress-xx —m set —match-set $[pod_ip_range] dst —j enn-policy-exxx
-A enn-egress-xx —j ACCEPT

enn-policy-ixxx chain:
—A enn-policy-ixxx —m conntrack —ctstate RELATED,ESTABLISHED —j ACCEPT
-A enn-policy-ixxx —j enn-dispatch-xxxx
-A enn-policy-ixxx —j enn-dispatch-xxxx
-A enn-policy-ixxx —j enn-dispatch-xxxx
-A enn-policy-ixxx —j reject

enn-policy-exxx chain:
—A enn-policy-exxx —m conntrack —ctstate RELATED,ESTABLISHED —j ACCEPT
-A enn-policy-exxx —j enn-dispatch-xxxx
-A enn-policy-exxx —j enn-dispatch-xxxx
-A enn-policy-exxx —j enn-dispatch-xxxx
-A enn-policy-exxx —j reject

enn-dispatch-xxxx chain:
-A enn-dispatch-xxxx —m set --match-set $[pod_set] src —m set --match-set
$[pod_set] dst —m $protocol —dport $port —j ACCEPT
```

表 2：规则总览

5. 注意事项和可能遇到的坑：

1. 网络环境配置可能导致 node 拿不到 source pod real ip：

如果 iptables 获取不到 source pod real ip，会直接导致隔离规则不起作用，对于 ingress 规则，会拒绝所有 traffic，对于 engress 规则，会接受所有 traffic。这种现象是 K8S 的网络配置导致的。如果是 flannel+docker0 这种配置方式，需要 flannel 做 SNAT，docker 不做 SNAT，（flannel's -ip-masq=true， docker's -ip-masq=false）。如果是 flannel+CNI0 这种配置，需要 flannel 做 SNAT，CNI0 不做 SNAT（我手动删除了 CNI postrouting chain 的 masq 规则）

2. 与原先集群网络环境的兼容问题：

原先的集群网络环境是所有 pod 全连通，在引入 ACL 之后，也应该保持原先的网络环境，即也是默认全通。通过创建新的 network policy 规则，逐步限制 pod 之间的访问连接。

3. ingress 和 egress 定义的冲突问题：

ingress 和 egress 在定义规则的时候是相对独立的，所以对于同一个 traffic，可能存在 egress 的规则允许通过，ingress 的规则不允许通过的问题。比如符合 namespace1 的所有 pod 定义允许访问 namespace2 的所有 pod 的 egress 规则，符合 namespace2 的所有 pod 定义允许被 namespace3 的所有 pod 访问的 ingress 规则。原则上，只要有一个规则限制了 traffic，就应该 reject 这个 traffic。但是实际构造规则的时候，egress 和 ingress 都定义在一个 chain 上（如 forward chain），根据 iptables 的特性，traffic 会走先匹配到的规则。如果一个 chain 上两条规则，一个 accept traffic，一个 reject traffic，真实数据可能因为优先匹配到 accept traffic 而直接忽略到 reject rule。

解决这个问题的方法可能需要构造更加复杂的规则模型，比如利用 mark traffic + return chain 这样的方式