

K8S+TF 溯源和未来 下

2021.03.30 SDNLAB 一期一会

1.会议主讲人

张建勋，瞻博网络全国合作伙伴技术经理

2.主要内容

上期参会报告：<https://mesalab.cn/f/article/detail?id=478>

基于上一期对 K8S 网络环境的设计思路和业界常见 CNI 实现方法的介绍，本次会议分享将继续延续这一话题，进一步介绍 Tungsten Fabric 与 K8S 深度结合后的技术实现逻辑和收益，同时将带来最新的 TF Roadmap 介绍。

3. K8S(Kubernetes)+TF(Tungsten Fabric)

在小规模网络中，我们希望实现越简单越好；但是在大规模网络中，我们希望网络逻辑尽量简单。

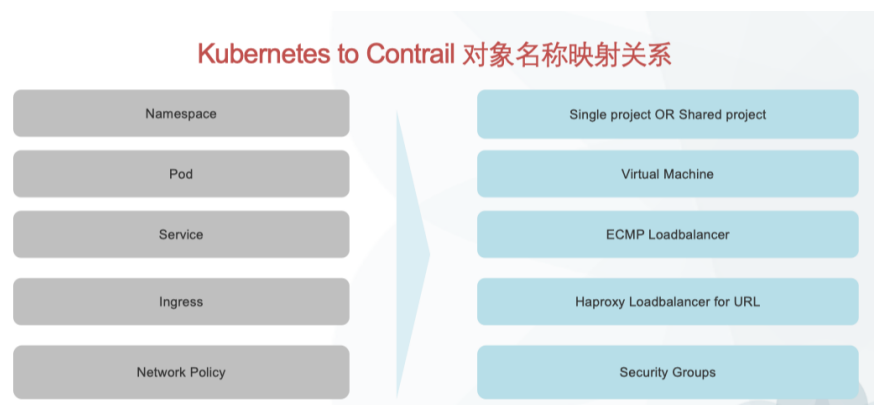


图 1

其实 service 和 ingress 都是 loadbalance 的不同形式的实现。

在 TF 中有多层隔离。本身基于 cluster 的隔离，在一个 cluster 里是可以互访，没有隔离的。但是当我们希望去进行管理，有了多租户、多分类等需求，就产生了基于 namespace (NS) 的隔离。我们可以给 namespace 赋予属性，是隔离的、不隔离的、自定义的隔离（这个 namespace 可以连接哪些网络）

如图 2，首先会有一个默认的 NS, 默认的 pod-network 和 service-network。这是我们创建一个新的 NS，我们可以设置为非隔离的，这是 default pod network 就可以同时给 default NS 和新创建的 NS 提供服务。如图 2 最右边的 NS 就是隔离的，那么它的网络就是独占的

namespace 之间默认不能互访，但是我们可以设置非隔离进行共享。

3.1 network policy

K8S 中的 network policy 在一定程度上可以称为 security policy，定义了 pod 之间如何通讯。但是 K8S 仅仅是作出 policy 的定义，实际是通过 CNI 来实现的。例如图 3，将 policy1 放到 NS dev 中，podselector 表示对应 matchlabel 里的 pod：NS 为 dev 的命名空间中的标签属于 webserver 的 pod。然后后面接着进（ingress）和出（egress），表示进出这些 pod 的需要执行的 policy 是 xxxx。

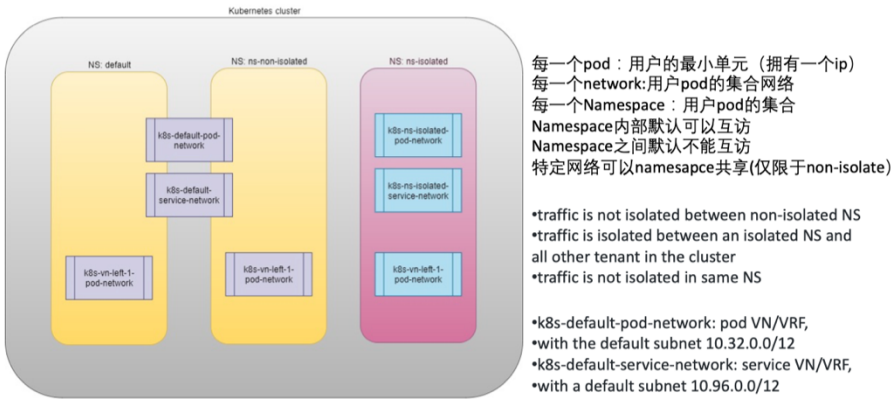


图 2

多条规则之间的关系：ingress from：下的多条规则，如果是在同一个短横线“-”下，是“与”的关系，多个短横线之间的规则是“或”的关系。

```
#policy1-do.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy1
  namespace: dev
spec:
  podSelector:
    matchLabels:
      app: webserver-dev
  policyTypes:
    - Ingress
    - Egress
```

两个方向执行

metadata: (policy1应用在dev-以namespace dev为基本参考点)
name: policy1
namespace: dev
podSelector:
matchLabels:
app: webserver-dev 配合前面 webserver-dev/dev

Ingress with from—从哪来的
egress with to—要到哪去

cidr: 10.169.25.20/32
Namespace: jtac
podSelector: app label=client1-dev/dev
Ports: TCP/UDP ports

```
ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: qa
      podSelector:
        matchLabels:
          app: client1-qa
```

```
ingress:
  from:
    namespaceSelector:
      matchLabels:
        project: qa
  podSelector:
    matchLabels:
      app: client1-qa
```

图 3

TF 和 K8S 中的 object 对应关系如图 4

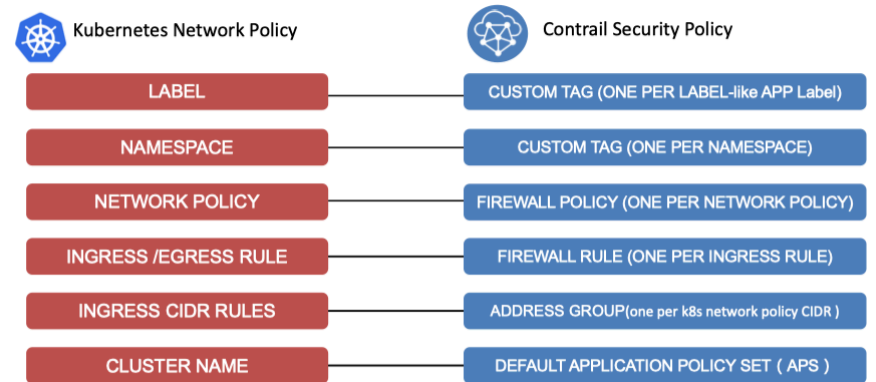
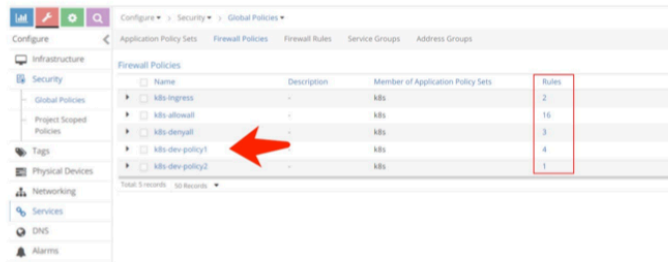


图 4

策略的实现如图 5~8。在一个网络中会有多个 policy，他们都有编号，表示为图 7 中的 sequence number，这个序号决定了流量在穿过当前网络的时候，执行的策略顺序

- Implemented as Application Policy Set within Contrail Security
- During Cluster initialization, default Application Policy sets are created to allow all traffic
- For every K8s Network Policy created by user, Contrail creates an Application Policy set with rules to adhere to the policy and a whitelist rule to ensure all other traffic is denied access



Name	Description	Member of Application Policy Sets	Rules
k8s-ingress	-	k8s	2
k8s-allowall	-	k8s	16
k8s-dev-policy1	-	k8s	3
k8s-dev-policy2	-	k8s	4
k8s-dev-policy2	-	k8s	1

图 5

```
$ kubectl get netpol --all-namespaces -o yaml
apiVersion: v1
items:
- apiVersion: extensions/v1beta1
  kind: NetworkPolicy
  metadata:
    ....
  spec:
    egress:
      - ports:
        - port: 80
          protocol: TCP
        to:
        - podSelector:
            matchLabels:
              app: dbserver-dev
            #<---rule#3
          ingress:
            - from:
              - ipBlock:
                  cidr: 10.169.25.20/32
                #<---rule#4
              - namespaceSelector:
                  matchLabels:
                    project: jtac
                  #<---rule#1
                podSelector:
                  matchLabels:
                    app: client1-dev
                  #<---rule#2
            ports:
            - port: 80
              protocol: TCP
          podSelector:
            matchLabels:
              app: webserver-dev
```

rule#	Action	Services	End Point1	Dir	End Point2	
1	pass	tcp:80	project=jtac	>	app=webserver-dev && namespace=dev	-
2	pass	tcp:80	app=client1-dev && namespace=dev	>	app=webserver-dev && namespace=dev	-
3	pass	tcp:80	app=webserver-dev && namespace=dev	>	app=dbserver-dev && namespace=dev	-
4	pass	tcp:80	Address Group: 10.169.25.20/32	>	app=webserver-dev && namespace=dev	-

图 6

NETWORK POLICY –sequence number

seq#	firewall policy
00002.0	k8s-Ingress
00038.0	k8s-dev-policy1
00040.0	k8s-dev-policy2
00042.0	k8s-denyall
00043.0	k8s-allowall

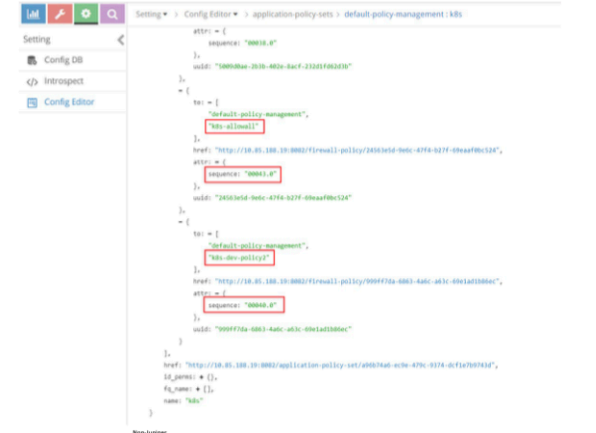


图 7

NETWORK POLICY IMPLEMENTATION

<pre>ingress: - from: - ipBlock: cidr: 10.169.25.20/32 - namespaceSelector: matchLabels: project: jtac - podSelector: matchLabels: app: client1-dev ports: - protocol: TCP port: 80 egress: - to: - podSelector: matchLabels: app: dbserver-dev ports: - protocol: TCP port: 80</pre>	<table><thead><tr><th>seq#</th><th>firewall rule</th></tr></thead><tbody><tr><td>0000.0</td><td>dev-ingress-policy1-0-ipBlock-0-cidr-10.169.25.20/32-0</td></tr><tr><td>0001.0</td><td>dev-ingress-policy1-0-namespaceSelector-1-0</td></tr><tr><td>0002.0</td><td>dev-ingress-policy1-0-podSelector-2-0</td></tr><tr><td>0003.0</td><td>dev-egress-policy1-podSelector-0-0</td></tr></tbody></table>	seq#	firewall rule	0000.0	dev-ingress-policy1-0-ipBlock-0-cidr-10.169.25.20/32-0	0001.0	dev-ingress-policy1-0-namespaceSelector-1-0	0002.0	dev-ingress-policy1-0-podSelector-2-0	0003.0	dev-egress-policy1-podSelector-0-0
seq#	firewall rule										
0000.0	dev-ingress-policy1-0-ipBlock-0-cidr-10.169.25.20/32-0										
0001.0	dev-ingress-policy1-0-namespaceSelector-1-0										
0002.0	dev-ingress-policy1-0-podSelector-2-0										
0003.0	dev-egress-policy1-podSelector-0-0										

图 8

3.2 其他功能

1) Source NAT

服务器设备本身是有 IP 的，可以用这个 IP 直接把 pod 透传出去。使用 56000~57023 端口为 TCP 做 NAT，57024~58047 端口为 UDP 做 NAT。在 annotations 里加上 ip_fabric_snat 这个属性，pod 在对外访问的时候就会有自己的端口。

```
apiVersion: v1
kind: Namespace
metadata:
  name: "snat"
  annotations: {
    "opencontrail.org/isolation": "true",
    "opencontrail.org/ip_fabric_snat": "true"
  }
```

- With the Contrail ip-fabric-snat feature, pods that are in the overlay can reach the Internet without floating IPs or a logical-router. The ip-fabric-snat feature uses compute node IP for creating a source NAT to reach the required services and is applicable only to pod networks. The kube-manager reserves ports 56000 through 57023 for TCP and 57024 through 58047 for UDP to create a source NAT in global-config during the initialization.
 - Allows pod to connect directly to external services/applications
 - Feature is enabled at namespace level and all Pods in the namespace have external connectivity
 - Contrail enables all the policies and does the PAT (port address translation) to map the POD port to host networking stack
- <https://github.com/tungstenfabric/tf-specs/blob/master/distributed-snat.md>
https://www.juniper.net/documentation/en_US/contrail20/topics/concept/k8s-ip-fabric.html

图 9 SNAT

2) IP fabric forwarding

- Allows a pod to connect to underlay network using the ip_fabric_ipam
- Pods can use native underlay routing for communication
- When Pods can be exposed by front-end services , the Service IP is routed via the overlay network
- Used to enable underlay networking for all the Pods in a namespace.

Example Application yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: "fab"
  annotations: {
    "opencontrail.org/isolation": "true",
    "opencontrail.org/ip_fabric_forwarding": "true"
  }
```

https://www.juniper.net/documentation/en_US/contrail20/topics/concept/k8s-ip-fabric.html
<https://github.com/tungstenfabric/tf-specs/blob/master/gateway-less-forwarding.md>

图 10 IP fabric forwarding

4 参考资料

视频

https://www.bilibili.com/video/BV1sZ4y1A78o?p=1&share_medium=android&share_plat=android&share_source=COPY&share_tag=s_i×tamp=1618899453&unique_k=Em3n67

PPT

链接: https://pan.baidu.com/s/1J3fsBs-2DjSE_Y-uLSj2cQ

提取码: 0330

Tungsten Fabric 相关资源

微信公众号: CTFSDN

官网 (中): <https://tungstenfabric.org.cn/>

官网 (英): <https://tungsten.io/>