[kubernetes networkpolicy](http://blog.csdn.net/yevvzi/article/details/53884623)

2016-12-26 11:42 678人阅读 [评论](http://blog.csdn.net/yevvzi/article/details/53884623#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/yevvzi/article/details/53884623#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

k8s（17） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

1.首先创建namespace隔离策略为DefaultDeny

**[plain]** [view plain](http://blog.csdn.net/yevvzi/article/details/53884623) [copy](http://blog.csdn.net/yevvzi/article/details/53884623)

1. kind: Namespace
2. apiVersion: v1
3. metadata:
4. name: testingnp
5. annotations:
6. net.beta.kubernetes.io/network-policy: |
7. {
8. "ingress": {
9. "isolation": "DefaultDeny"
10. }
11. }

或者通过命令行对已有namesapce操作  
kubectl annotate ns testingnp "net.beta.kubernetes.io/network-policy={\"ingress\": {\"isolation\": \"DefaultDeny\"}}"  
通过spec.podSelector.matchLabels 制定操作的pod对象  
spec.ingress from/ports来制定允许访问的pod和端口  
  
2.在开启isolation的namespace运行、暴漏服务

**[plain]** [view plain](http://blog.csdn.net/yevvzi/article/details/53884623) [copy](http://blog.csdn.net/yevvzi/article/details/53884623)

1. kubectl run nginx --image=nginx --replicas=2 --namespace=testingnp
2. kubectl expose deployment nginx --port=80  --namespace=testingnp

3.测试连接状态  
kubectl run busybox --rm -ti --image=busybox /bin/sh --namespace=testingnp  
wget nginx 发现是无法访问的  
4.添加networkpolicy

**[plain]** [view plain](http://blog.csdn.net/yevvzi/article/details/53884623) [copy](http://blog.csdn.net/yevvzi/article/details/53884623)

1. echo '
2. kind: NetworkPolicy
3. apiVersion: extensions/v1beta1
4. metadata:
5. name: access-nginx
6. namespace: testingnp
7. spec:
8. podSelector:
9. matchLabels:
10. run: nginx
11. ingress:
12. - from:
13. - podSelector:
14. matchLabels:
15. access: "true"
16. ' | kubectl create  -f -
17. kubectl get networkpolicies --namespace=testingnp

5.再次验证  
制定容器label  
kubectl run busybox --rm -ti --labels="access=true" --image=busybox /bin/sh --namespace=testingnp   
wget nginx 可以正常获取资源

[**步赠书：9月重磅新书升级，本本经典**](http://blog.csdn.net/epubit17/article/details/78038722)           [**程序员9月书讯**](http://blog.csdn.net/turingbooks/article/details/78017356)      [**每周荐书：ES6、虚拟现实、物联网（评论送书）**](http://blog.csdn.net/broadview2006/article/details/78018836)

# [容器编排之Kubernetes网络隔离NetworkPolicy](http://blog.csdn.net/qq_34463875/article/details/74288175)

标签： [网络](http://www.csdn.net/tag/%e7%bd%91%e7%bb%9c)

2017-07-04 10:02 400人阅读 [评论](http://blog.csdn.net/qq_34463875/article/details/74288175#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/qq_34463875/article/details/74288175#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

kubernetes（75） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

目录[(?)](http://blog.csdn.net/qq_34463875/article/details/74288175" \o "系统根据文章中H1到H6标签自动生成文章目录)[[+]](http://blog.csdn.net/qq_34463875/article/details/74288175)

Kubernetes的一个重要特性就是要把不同node节点的pod（container）连接起来，无视物理节点的限制。但是在某些应用环境中，比如公有云，不同租户的pod不应该互通，这个时候就需要网络隔离。幸好，Kubernetes提供了NetworkPolicy，支持按Namespace级别的网络隔离，这篇文章就带你去了解如何使用NetworkPolicy。

需要注意的是，使用NetworkPolicy需要特定的网络解决方案，如果不启用，即使配置了NetworkPolicy也无济于事。我们这里使用Calico解决网络隔离问题。

## 互通测试

在使用NetworkPolicy之前，我们先验证不使用的情况下，pod是否互通。这里我们的测试环境是这样的：

Namespace：ns-calico1，ns-calico2

Deployment: ns-calico1/calico1-nginx, ns-calico2/busybox

Service: ns-calico1/calico1-nginx

先创建Namespace：

apiVersion: v1

kind: Namespace

metadata:

name: ns-calico1

labels:

user: calico1

---

apiVersion: v1

kind: Namespace

metadata:

name: ns-calico2

# kubectl create -f namespace.yaml

namespace "ns-calico1" created

namespace "ns-calico2" created

# kubectl get ns

NAME STATUS AGE

default Active 9d

kube-public Active 9d

kube-system Active 9d

ns-calico1 Active 12s

ns-calico2 Active 8s

接着创建ns-calico1/calico1-nginx：

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: calico1-nginx

namespace: ns-calico1

spec:

replicas: 1

template:

metadata:

labels:

user: calico1

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: calico1-nginx

namespace: ns-calico1

labels:

user: calico1

spec:

selector:

app: nginx

ports:

- port: 80

# kubectl create -f calico1-nginx.yaml

deployment "calico1-nginx" created

service "calico1-nginx" created

# kubectl get svc -n ns-calico1

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

calico1-nginx 192.168.3.141 <none> 80/TCP 26s

# kubectl get deploy -n ns-calico1

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

calico1-nginx 1 1 1 1 34s

最后创建ns-calico2/calico2-busybox:

apiVersion: v1

kind: Pod

metadata:

name: calico2-busybox

namespace: ns-calico2

spec:

containers:

- name: busybox

image: busybox

command:

- sleep

- "3600"

# kubectl create -f calico2-busybox.yaml

pod "calico2-busybox" created

# kubectl get pod -n ns-calico2

NAME READY STATUS RESTARTS AGE

calico2-busybox 1/1 Running 0 40s

测试服务已经安装完成，现在我们登进calico2-busybox里，看是否能够连通calico1-nginx

# kubectl exec -it calico2-busybox -n ns-calico2 -- wget --spider --timeout=1 calico1-nginx.ns-calico1

Connecting to calico1-nginx.ns-calico1 (192.168.3.141:80)

由此可以看出，在没有设置网络隔离的时候，两个不同Namespace下的Pod是可以互通的。接下来我们使用Calico进行网络隔离。

## 网络隔离

**先决条件**

要想在Kubernetes集群中使用Calico进行网络隔离，必须满足以下条件：

1. kube-apiserver必须开启运行时extensions/v1beta1/networkpolicies，即设置启动参数：–runtime-config=extensions/v1beta1/networkpolicies=true
2. kubelet必须启用cni网络插件，即设置启动参数：–network-plugin=cni
3. kube-proxy必须启用iptables代理模式，这是默认模式，可以不用设置
4. kube-proxy不得启用–masquerade-all，这会跟calico冲突

**注意：配置Calico之后，之前在集群中运行的Pod都要重新启动**

**安装calico**

首先需要安装Calico网络插件，我们直接在Kubernetes集群中安装，便于管理。

# Calico Version v2.1.4

# http://docs.projectcalico.org/v2.1/releases#v2.1.4

# This manifest includes the following component versions:

# calico/node:v1.1.3

# calico/cni:v1.7.0

# calico/kube-policy-controller:v0.5.4

# This ConfigMap is used to configure a self-hosted Calico installation.

kind: ConfigMap

apiVersion: v1

metadata:

name: calico-config

namespace: kube-system

data:

# Configure this with the location of your etcd cluster.

etcd\_endpoints: "https://10.1.2.154:2379,https://10.1.2.147:2379"

# Configure the Calico backend to use.

calico\_backend: "bird"

# The CNI network configuration to install on each node.

cni\_network\_config: |-

{

"name": "k8s-pod-network",

"type": "calico",

"etcd\_endpoints": "\_\_ETCD\_ENDPOINTS\_\_",

"etcd\_key\_file": "\_\_ETCD\_KEY\_FILE\_\_",

"etcd\_cert\_file": "\_\_ETCD\_CERT\_FILE\_\_",

"etcd\_ca\_cert\_file": "\_\_ETCD\_CA\_CERT\_FILE\_\_",

"log\_level": "info",

"ipam": {

"type": "calico-ipam"

},

"policy": {

"type": "k8s",

"k8s\_api\_root": "https://\_\_KUBERNETES\_SERVICE\_HOST\_\_:\_\_KUBERNETES\_SERVICE\_PORT\_\_",

"k8s\_auth\_token": "\_\_SERVICEACCOUNT\_TOKEN\_\_"

},

"kubernetes": {

"kubeconfig": "\_\_KUBECONFIG\_FILEPATH\_\_"

}

}

# If you're using TLS enabled etcd uncomment the following.

# You must also populate the Secret below with these files.

etcd\_ca: "/calico-secrets/etcd-ca" # "/calico-secrets/etcd-ca"

etcd\_cert: "/calico-secrets/etcd-cert" # "/calico-secrets/etcd-cert"

etcd\_key: "/calico-secrets/etcd-key" # "/calico-secrets/etcd-key"

---

# The following contains k8s Secrets for use with a TLS enabled etcd cluster.

# For information on populating Secrets, see http://kubernetes.io/docs/user-guide/secrets/

apiVersion: v1

kind: Secret

type: Opaque

metadata:

name: calico-etcd-secrets

namespace: kube-system

data:

# Populate the following files with etcd TLS configuration if desired, but leave blank if

# not using TLS for etcd.

# This self-hosted install expects three files with the following names. The values

# should be base64 encoded strings of the entire contents of each file.

etcd-key: base64 key.pem

etcd-cert: base64 cert.pem

etcd-ca: base64 ca.pem

---

# This manifest installs the calico/node container, as well

# as the Calico CNI plugins and network config on

# each master and worker node in a Kubernetes cluster.

apiVersion: extensions/v1beta1

kind: DaemonSet

metadata:

name: calico-node

namespace: kube-system

labels:

k8s-app: calico-node

spec:

selector:

matchLabels:

k8s-app: calico-node

template:

metadata:

labels:

k8s-app: calico-node

annotations:

scheduler.alpha.kubernetes.io/critical-pod: ''

scheduler.alpha.kubernetes.io/tolerations: |

[{"key": "dedicated", "value": "master", "effect": "NoSchedule" },

{"key":"CriticalAddonsOnly", "operator":"Exists"}]

spec:

hostNetwork: true

containers:

# Runs calico/node container on each Kubernetes node. This

# container programs network policy and routes on each

# host.

- name: calico-node

image: quay.io/calico/node:v1.1.3

env:

# The location of the Calico etcd cluster.

- name: ETCD\_ENDPOINTS

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_endpoints

# Choose the backend to use.

- name: CALICO\_NETWORKING\_BACKEND

valueFrom:

configMapKeyRef:

name: calico-config

key: calico\_backend

# Disable file logging so `kubectl logs` works.

- name: CALICO\_DISABLE\_FILE\_LOGGING

value: "true"

# Set Felix endpoint to host default action to ACCEPT.

- name: FELIX\_DEFAULTENDPOINTTOHOSTACTION

value: "ACCEPT"

# Configure the IP Pool from which Pod IPs will be chosen.

- name: CALICO\_IPV4POOL\_CIDR

value: "192.168.0.0/16"

- name: CALICO\_IPV4POOL\_IPIP

value: "always"

# Disable IPv6 on Kubernetes.

- name: FELIX\_IPV6SUPPORT

value: "false"

# Set Felix logging to "info"

- name: FELIX\_LOGSEVERITYSCREEN

value: "info"

# Location of the CA certificate for etcd.

- name: ETCD\_CA\_CERT\_FILE

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_ca

# Location of the client key for etcd.

- name: ETCD\_KEY\_FILE

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_key

# Location of the client certificate for etcd.

- name: ETCD\_CERT\_FILE

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_cert

# Auto-detect the BGP IP address.

- name: IP

value: ""

securityContext:

privileged: true

#resources:

#requests:

#cpu: 250m

volumeMounts:

- mountPath: /lib/modules

name: lib-modules

readOnly: true

- mountPath: /var/run/calico

name: var-run-calico

readOnly: false

- mountPath: /calico-secrets

name: etcd-certs

# This container installs the Calico CNI binaries

# and CNI network config file on each node.

- name: install-cni

image: quay.io/calico/cni:v1.7.0

command: ["/install-cni.sh"]

env:

# The location of the Calico etcd cluster.

- name: ETCD\_ENDPOINTS

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_endpoints

# The CNI network config to install on each node.

- name: CNI\_NETWORK\_CONFIG

valueFrom:

configMapKeyRef:

name: calico-config

key: cni\_network\_config

volumeMounts:

- mountPath: /host/opt/cni/bin

name: cni-bin-dir

- mountPath: /host/etc/cni/net.d

name: cni-net-dir

- mountPath: /calico-secrets

name: etcd-certs

volumes:

# Used by calico/node.

- name: lib-modules

hostPath:

path: /lib/modules

- name: var-run-calico

hostPath:

path: /var/run/calico

# Used to install CNI.

- name: cni-bin-dir

hostPath:

path: /opt/cni/bin

- name: cni-net-dir

hostPath:

path: /etc/cni/net.d

# Mount in the etcd TLS secrets.

- name: etcd-certs

secret:

secretName: calico-etcd-secrets

---

# This manifest deploys the Calico policy controller on Kubernetes.

# See https://github.com/projectcalico/k8s-policy

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: calico-policy-controller

namespace: kube-system

labels:

k8s-app: calico-policy

annotations:

scheduler.alpha.kubernetes.io/critical-pod: ''

scheduler.alpha.kubernetes.io/tolerations: |

[{"key": "dedicated", "value": "master", "effect": "NoSchedule" },

{"key":"CriticalAddonsOnly", "operator":"Exists"}]

spec:

# The policy controller can only have a single active instance.

replicas: 1

strategy:

type: Recreate

template:

metadata:

name: calico-policy-controller

namespace: kube-system

labels:

k8s-app: calico-policy

spec:

# The policy controller must run in the host network namespace so that

# it isn't governed by policy that would prevent it from working.

hostNetwork: true

containers:

- name: calico-policy-controller

image: quay.io/calico/kube-policy-controller:v0.5.4

env:

# The location of the Calico etcd cluster.

- name: ETCD\_ENDPOINTS

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_endpoints

# Location of the CA certificate for etcd.

- name: ETCD\_CA\_CERT\_FILE

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_ca

# Location of the client key for etcd.

- name: ETCD\_KEY\_FILE

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_key

# Location of the client certificate for etcd.

- name: ETCD\_CERT\_FILE

valueFrom:

configMapKeyRef:

name: calico-config

key: etcd\_cert

# The location of the Kubernetes API. Use the default Kubernetes

# service for API access.

- name: K8S\_API

value: "https://kubernetes.default:443"

# Since we're running in the host namespace and might not have KubeDNS

# access, configure the container's /etc/hosts to resolve

# kubernetes.default to the correct service clusterIP.

- name: CONFIGURE\_ETC\_HOSTS

value: "true"

volumeMounts:

# Mount in the etcd TLS secrets.

- mountPath: /calico-secrets

name: etcd-certs

volumes:

# Mount in the etcd TLS secrets.

- name: etcd-certs

secret:

secretName: calico-etcd-secrets

# kubectl create -f calico.yaml

configmap "calico-config" created

secret "calico-etcd-secrets" created

daemonset "calico-node" created

deployment "calico-policy-controller" created

# kubectl get ds -n kube-system

NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE-SELECTOR AGE

calico-node 1 1 1 1 1 <none> 52s

# kubectl get deploy -n kube-system

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

calico-policy-controller 1 1 1 1 6m

这样就搭建了Calico网络，接下来就可以配置NetworkPolicy了。

**配置NetworkPolicy**

首先，修改ns-calico1的配置：

apiVersion: v1

kind: Namespace

metadata:

name: ns-calico1

labels:

user: calico1

annotations:

net.beta.kubernetes.io/network-policy: |

{

"ingress": {

"isolation": "DefaultDeny"

}

}

# kubectl apply -f ns-calico1.yaml

namespace "ns-calico1" configured

如果这个时候再测试两个pod是否连通，一定会不通：

# kubectl exec -it calico2-busybox -n ns-calico2 -- wget --spider --timeout=1 calico1-nginx.ns-calico1

Connecting to calico1-nginx.ns-calico1 (192.168.3.71:80)

wget: download timed out

这就是我们想要的效果，不同Namespace之间的pod不能互通，当然这只是最简单的情况，如果这时候ns-calico1的pod去连接ns-calico2的pod，还是互通的。因为ns-calico2没有设置Namespace annotations。

而且，这时候的ns-calico1会拒绝任何pod的通讯请求。因为，Namespace的annotations只是指定了拒绝所有的通讯请求，还未规定何时接受其他pod的通讯请求。在这里，我们指定只有拥有user=calico1标签的pod可以互联。

apiVersion: extensions/v1beta1

kind: NetworkPolicy

metadata:

name: calico1-network-policy

namespace: ns-calico1

spec:

podSelector:

matchLabels:

user: calico1

ingress:

- from:

- namespaceSelector:

matchLabels:

user: calico1

- podSelector:

matchLabels:

user: calico1

---

apiVersion: v1

kind: Pod

metadata:

name: calico1-busybox

namespace: ns-calico1

labels:

user: calico1

spec:

containers:

- name: busybox

image: busybox

command:

- sleep

- "3600"

# kubectl create -f calico1-network-policy.yaml

networkpolicy "calico1-network-policy" created

# kubectl create -f calico1-busybox.yaml

pod "calico1-busybox" created

这时候，如果我通过calico1-busybox连接calico1-nginx，则可以连通。

# kubectl exec -it calico1-busybox -n ns-calico1 -- wget --spider --timeout=1 calico1-nginx.ns-calico1

Connecting to calico1-nginx.ns-calico1 (192.168.3.71:80)

这样我们就实现了Kubernetes的网络隔离。基于NetworkPolicy，可以实现公有云安全组策略。更多的NetworkPolicy参数，请参考 [api-reference](http://link.zhihu.com/?target=https%3A//kubernetes.io/docs/api-reference/extensions/v1beta1/definitions/%23_v1beta1_networkpolicy" \t "_blank)。

参考资料：

1. [Network Policies](http://link.zhihu.com/?target=https%3A//kubernetes.io/docs/concepts/services-networking/networkpolicies/%23configuring-namespace-isolation-policy)
2. [Declaring Network Policy](http://link.zhihu.com/?target=https%3A//kubernetes.io/docs/tasks/configure-pod-container/declare-network-policy/)
3. [Using Calico for NetworkPolicy](http://link.zhihu.com/?target=https%3A//kubernetes.io/docs/tasks/configure-pod-container/calico-network-policy/)
4. [Calico for Kubernetes](http://link.zhihu.com/?target=http%3A//docs.projectcalico.org/v2.1/getting-started/kubernetes/)

本文：[容器编排之Kubernetes网络隔离NetworkPolicy](https://www.kubernetes.org.cn/1909.html" \t "_blank)

# Kubernetes v1.7新特性解析-Network Policy GA

[Feisky](https://www.zhihu.com/people/feisky)[Feisky](https://www.zhihu.com/people/feisky)

3 个月前

Network Policy提供了基于策略的网络控制，用于隔离应用并减少攻击面。它使用标签选择器模拟传统的分段网络，并通过策略控制它们之间的流量以及来自外部的流量。

在使用Network Policy之前，需要注意

* v1.6以及以前的版本需要在apiserver开启extensions/v1beta1/networkpolicies
* v1.7+版本Network Policy已经GA，API版本为[http://networking.k8s.io/v1](http://link.zhihu.com/?target=http%3A//networking.k8s.io/v1)
* 网络插件要支持Network Policy，如Calico、Romana、Weave Net和trireme等

## 策略

## Namespace隔离

默认情况下，所有Pod之间是全通的。每个Namespace可以配置独立的网络策略，来隔离Pod之间的流量。

v1.7+版本通过创建匹配所有Pod的Network Policy来作为默认的网络策略，比如默认拒绝所有Pod之间通信

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: default-deny

spec:

podSelector:

而默认允许所有Pod通信的策略为

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-all

spec:

podSelector:

ingress:

- {}

而v1.6版本则通过Annotation来隔离namespace的所有Pod之间的流量，包括从外部到该namespace中所有Pod的流量以及namespace内部Pod相互之间的流量：

kubectl annotate ns <namespace> "net.beta.kubernetes.io/network-policy={\"ingress\": {\"isolation\": \"DefaultDeny\"}}"

注：目前，Network Policy仅支持Ingress流量控制。

## Pod隔离

通过使用标签选择器（包括namespaceSelector和podSelector）来控制Pod之间的流量。比如下面的Network Policy

* 允许default namespace中带有role=frontend标签的Pod访问default namespace中带有role=db标签Pod的6379端口
* 允许带有project=myprojects标签的namespace中所有Pod访问default namespace中带有role=db标签Pod的6379端口

# v1.6以及更老的版本应该使用extensions/v1beta1

# apiVersion: extensions/v1beta1

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: test-network-policy

namespace: default

spec:

podSelector:

matchLabels:

role: db

ingress:

- from:

- namespaceSelector:

matchLabels:

project: myproject

- podSelector:

matchLabels:

role: frontend

ports:

- protocol: tcp

port: 6379

## 示例

以calico为例看一下Network Policy的具体用法（使用kubernetes v1.6版本）。

首先配置kubelet使用CNI网络插件

kubelet --network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin ...

安装calio网络插件

# 注意修改CIDR，需要跟k8s pod-network-cidr一致，默认为192.168.0.0/16

kubectl apply -f http://docs.projectcalico.org/v2.1/getting-started/kubernetes/installation/hosted/kubeadm/1.6/calico.yaml

首先部署一个nginx服务

$ kubectl run nginx --image=nginx --replicas=2

deployment "nginx" created

$ kubectl expose deployment nginx --port=80

service "nginx" exposed

此时，通过其他Pod是可以访问nginx服务的

$ kubectl run busybox --rm -ti --image=busybox /bin/sh

Waiting for pod default/busybox-472357175-y0m47 to be running, status is Pending, pod ready: false

Hit enter for command prompt

/ # wget --spider --timeout=1 nginx

Connecting to nginx (10.100.0.16:80)

/ #

开启default namespace的DefaultDeny Network Policy后，其他Pod（包括namespace外部）不能访问nginx了：

$ kubectl run busybox --rm -ti --image=busybox /bin/sh

Waiting for pod default/busybox-472357175-y0m47 to be running, status is Pending, pod ready: false

Hit enter for command prompt

/ # wget --spider --timeout=1 nginx

Connecting to nginx (10.100.0.16:80)

wget: download timed out

/ #

最后再创建一个运行带有access=true的Pod访问的网络策略

$ cat nginx-policy.yaml

kind: NetworkPolicy

apiVersion: networking.k8s.io/v1

metadata:

name: access-nginx

spec:

podSelector:

matchLabels:

run: nginx

ingress:

- from:

- podSelector:

matchLabels:

access: "true"

$ kubectl create -f nginx-policy.yaml

networkpolicy "access-nginx" created

# 不带access=true标签的Pod还是无法访问nginx服务

$ kubectl run busybox --rm -ti --image=busybox /bin/sh

Waiting for pod default/busybox-472357175-y0m47 to be running, status is Pending, pod ready: false

Hit enter for command prompt

/ # wget --spider --timeout=1 nginx

Connecting to nginx (10.100.0.16:80)

wget: download timed out

/ #

# 而带有access=true标签的Pod可以访问nginx服务

$ kubectl run busybox --rm -ti --labels="access=true" --image=busybox /bin/sh

Waiting for pod default/busybox-472357175-y0m47 to be running, status is Pending, pod ready: false

Hit enter for command prompt

/ # wget --spider --timeout=1 nginx

Connecting to nginx (10.100.0.16:80)

/ #

最后开启nginx服务的外部访问：

$ cat nginx-external-policy.yaml

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: front-end-access

namespace: sock-shop

spec:

podSelector:

matchLabels:

run: nginx

ingress:

- ports:

- protocol: TCP

port: 80

$ kubectl create -f nginx-external-policy.yaml

本文已发布到《[Kubernetes指南](http://link.zhihu.com/?target=https%3A//github.com/feiskyer/kubernetes-handbook)》开源书，欢迎关注。

KubernetesDockercloud native

1

[陈立诚](https://www.zhihu.com/people/chen-li-cheng-70)

收藏

分享

举报

**文章被以下专栏收录**

* [](https://zhuanlan.zhihu.com/kubernetes)

[**Kubernetes指南**](https://zhuanlan.zhihu.com/kubernetes)

Kubernetes参考指南和实践总结

[进入专栏](https://zhuanlan.zhihu.com/kubernetes)

**还没有评论**