NETWORK POLICY

About NetworkPolicy

Network policy will be used for internal communication, it will act as firewall where we can allow/block the specific range of ip's or namespace level or podselector by using ingress/egress method.
Ingress is called as incoming traffic, from ip range, from namespace or from specific pod.
Egress is called as outgoing traffic, to ip range, to namespace or to specific pods.
It's nothing but network firewall.

Actual Readme

A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.
NetworkPolicy resources use labels to select pods and define rules which specify what traffic is allowed to the selected pods.
Network policies are implemented by the network plugin, so you must be using a networking solution which supports NetworkPolicy - simply creating the resource without a controller to implement it will have no effect
Isolated and Non-isolated Pods
By default, pods are non-isolated; they accept traffic from any source.
Pods become isolated by having a NetworkPolicy that selects them. Once there is any NetworkPolicy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any NetworkPolicy. (Other pods in the namespace that are not selected by any NetworkPolicy will continue to accept all traffic.)
Overview
You can use GKE's network policy enforcement to control the communication between your cluster's Pods and Services. To define a network policy on GKE, you can use the Kubernetes Network Policy API to create Pod-level firewall rules. These firewall rules determine which Pods and Services can access one another inside your cluster.
Defining network policy helps you enable things like defense in depth when your cluster is serving a multi-level application. For example, you can create a network policy to ensure that a compromised front-end service in your application cannot communicate directly with a billing or accounting service several levels down.
Network policy can also make it easier for your application to host data from multiple users simultaneously. For example, you can provide secure multi-tenancy by defining a tenant-per-namespace model. In such a model, network policy rules can ensure that Pods and Services in a given namespace cannot access other Pods or Services in a different namespace.
Before you begin
explanation
Mandatory Fields: As with all other Kubernetes config, a NetworkPolicy needs apiVersion, kind, and metadata fields. For general information about working with config files, see Configure Containers Using a ConfigMap, and Object Management.
spec: NetworkPolicy spec has all the information needed to define a particular network policy in the given namespace.
podSelector: Each NetworkPolicy includes a podSelector which selects the grouping of pods to which the policy applies. The example policy selects pods with the label "role=db". An empty podSelector selects all pods in the namespace.

policyTypes: Each NetworkPolicy includes a policyTypes list which may include either Ingress, Egress, or both. The policyTypes field indicates whether or not the given policy applies to ingress traffic to selected pod, egress traffic from selected pods, or both. If no policyTypes are specified on a NetworkPolicy then by default Ingress will always be set and Egress will be set if the NetworkPolicy has any egress rules.
ingress: Each NetworkPolicy may include a list of whitelist ingress rules. Each rule allows traffic which matches both the from and ports sections. The example policy contains a single rule, which matches traffic on a single port, from one of three sources, the first specified via an ipBlock, the second via a namespaceSelector and the third via a podSelector.
egress: Each NetworkPolicy may include a list of whitelist egress rules. Each rule allows traffic which matches both the to and ports sections. The example policy contains a single rule, which matches traffic on a single port to any destination in 10.0.0.0/24.
So, the example NetworkPolicy:
isolates "role=db" pods in the "default" namespace for both ingress and egress traffic (if they weren't already isolated)
(Ingress rules) allows connections to all pods in the "default" namespace with the label "role=db" on TCP port 6379 from:
any pod in the "default" namespace with the label "role=frontend"
any pod in a namespace with the label "project=myproject"
IP addresses in the ranges 172.17.0.0–172.17.0.255 and 172.17.2.0–172.17.255.255 (ie, all of 172.17.0.0/16 except 172.17.1.0/24)
(Egress rules) allows connections from any pod in the "default" namespace with the label "role=db" to CIDR 10.0.0.0/24 on TCP port 5978

##Going to set default deny policies##
@@Create two pods@@
cat pods.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-web1
  labels:
    env: web
spec:
  containers:
    - name: website1
      image: nginx:1.16
      ports:
        - containerPort: 80
```

cat pods1.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: testing
  labels:
    prod: web
spec:
  containers:
    - name: website2
      image: nginx:1.16
      ports:
        - containerPort: 80
```

kubectl apply -f pods.yml
kubectl apply -f pods1.yml

kubectl get pods -o wide

```
[root@ansikube Manifest]# kubectl get pods -o wide
NAME         READY   STATUS    RESTARTS   AGE     IP          NODE
ngnix-web1   1/1     Running   0          2m13s   10.32.2.3   gke-robo-default-pool-73f4e14b-qzbf
testing      1/1     Running   0          2m2s    10.32.2.4   gke-robo-default-pool-73f4e14b-qzbf
[root@ansikube Manifest]#
```

kubectl exec -it testing /bin/bash
apt-get update && apt-get install iputils-ping && apt-get install curl
curl http://10.32.2.3

```
root@testing:/# curl http://10.32.2.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

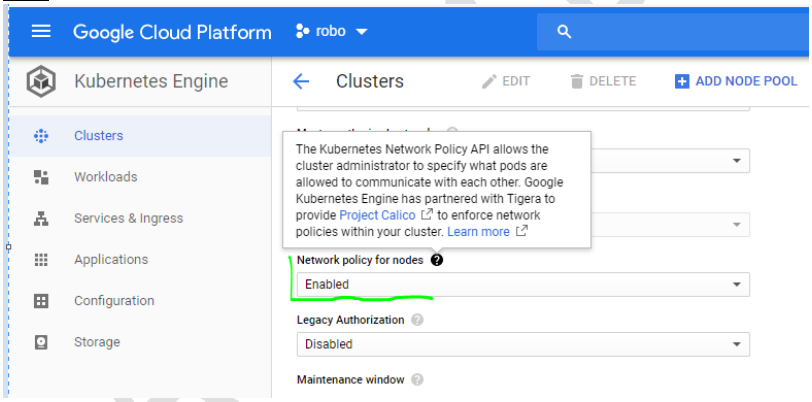
@@Create default deny policy for ingress method@@

:-You can update the network policy in GCP, there are two ways, either by terminal or GUI

Terminal

#gcloud container clusters create [CLUSTER_NAME] --enable-network-policy
Enabling network policy enforcement for an existing cluster with the gcloud command-line tool is a two-step process. First, run the gcloud container clusters update command with the --update-addons flag:
#gcloud container clusters update [CLUSTER_NAME] --update-addons=NetworkPolicy=ENABLED
Then, run the gcloud container clusters update command with the --enable-network-policy flag. This command causes your cluster's node pools to be recreated with network policy enabled:
#gcloud container clusters update [CLUSTER_NAME] --enable-network-policy

GUI



Note:- Either you can select network policy while creating cluster or can update network policy later on cluster by selecting enable on networkpolicy option, which will take time to complete.

kubectl get networkpolicy

```
[root@ansikube manifest]# kubectl get networkpolicy
No resources found in default namespace.
[root@ansikube manifest]#
```

cat default-ingress-deny.yml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

kubectl apply -f default-ingress-deny.yml

kubectl get networkpolicy

```
[root@ansikube manifest]# kubectl get networkpolicy
NAME                   POD-SELECTOR   AGE
default-deny-ingress   <none>         11s
[root@ansikube manifest]#
```

kubectl describe networkpolicy default-deny-ingress

```
[root@ansikube manifest]# kubectl describe networkpolicy default-deny-ingress
Name:         default-deny-ingress
Namespace:    default
Created on:   2019-11-08 10:13:48 +0000 UTC
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"networking.k8s.io/v1","kind":"NetworkPolicy","metadata":
efaul...
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    <none> (Selected pods are isolated for ingress connectivity)
  Allowing egress traffic:
    <none> (Selected pods are isolated for egress connectivity)
  Policy Types: Ingress
[root@ansikube manifest]#
```

@Login to pod and test the connectivity.

kubectl exec -it testing /bin/bash

ping 10.32.2.3 && curl http://10.32.2.3

```
root@testing:/# ping 10.32.2.3
PING 10.32.2.3 (10.32.2.3) 56(84) bytes of data.
^C
--- 10.32.2.3 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 135ms

root@testing:/# curl http://10.32.2.3
^C
root@testing:/#
```

Note:- After enabling default deny method in network policy, now unable to ping or curl.

@@Create network policy to allow ingress method for pod selectors with match labels.

cat np1.yml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      dev: web
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          prod: web
```

kubectl apply -f np1.yml
kubectl get networkpolicy

```
[root@ansikube Manifest]# kubectl get networkpolicy
NAME                  POD-SELECTOR   AGE
default-deny          <none>         26m
test-network-policy   dev=web        13s
```

kubectl describe networkpolicy test-network-policy

```
[root@ansikube Manifest]# kubectl describe networkpolicy test-network-policy
Name:         test-network-policy
Namespace:    default
Created on:   2019-11-09 05:41:23 +0000 UTC
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"networking.k8s.io/v1","kind":"NetworkPolicy",
fault...
Spec:
  PodSelector:     dev=web
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: prod=web
  Allowing egress traffic:
    <none> (Selected pods are isolated for egress connectivity)
  Policy Types: Ingress
[root@ansikube Manifest]#
```

@Login to pod and test.

kubectl exec -it testing /bin/bash
ping 10.32.2.3 && curl http://10.32.2.3

```
root@testing:/# curl http://10.32.2.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

Note:- above network policy will be allowing the ingress communication from specified pod match label.

@@Create default deny policy for egress method@@

cat default-egress-deny.yml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-egress
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

kubectl apply -f default-egress-deny.yml
kubectl get networkpolicy

```
[root@ansikube Manifest]# kubectl get networkpolicy
NAME                  POD-SELECTOR   AGE
default-deny          <none>         38m
default-deny-egress   <none>         15s
test-network-policy   dev=web        12m
[root@ansikube Manifest]#
```

@Login to pod and test the connectivity.

kubectl exec -it testing /bin/bash
ping 10.32.2.3
curl http://10.32.2.3
ping google.com

```
root@testing:/# ping 10.32.2.3
PING 10.32.2.3 (10.32.2.3) 56(84) bytes of data.
^C
--- 10.32.2.3 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 50ms

root@testing:/# curl http://10.32.2.3
^C
root@testing:/# ping google.com
^C
root@testing:/#
```

Note: Outgoing traffic has been blocked, so even external domain also not reachable.

@@Create network policy to allow egress method for pod selectors with match labels@@

cat np2.yml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy2
  namespace: default
spec:
  podSelector:
    matchLabels:
      prod: web
  policyTypes:
  - Egress
  egress:
  - to:
    - podSelector:
        matchLabels:
          dev: web
```

kubectl apply -f np2.yml

kubectl get networkpolicy

```
[root@ansikube Manifest]# kubectl get networkpolicy
NAME                  POD-SELECTOR   AGE
default-deny          <none>         45m
default-deny-egress   <none>         6m26s
test-network-policy   dev=web        19m
test-network-policy2  prod=web       15s
[root@ansikube Manifest]#
```

@Login to pod and test the connectivity.

kubectl exec -it testing /bin/bash

ping 10.32.2.3

```
root@testing:/# ping 10.32.2.3
PING 10.32.2.3 (10.32.2.3) 56(84) bytes of data.
64 bytes from 10.32.2.3: icmp_seq=1 ttl=63 time=0.073 ms
^C
root@testing:/# curl http://10.32.2.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

Note:- above network policy will be allowing the egress communication from specified pod matchlabels.