31Oct2022
Day**18**
Kubernetes Cluster Upgrade
NetworkPolicy

Kubernetes Cluster Upgrade

https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/
The upgrade workflow at high level is the following:
-Upgrade a primary control plane node.
-Upgrade additional control plane nodes.
-Upgrade worker nodes.

kubectl get nodes

NAME	STATUS	ROLES	AGE	VERSION
master1.example.com	Ready	control-plane	141d	**v1.24.1**

cat /etc/os-release
RHEL-Based
cat /etc/redhat-release
Debian-Based
lsb_release -a
Determine which version to upgrade to
apt update
apt-cache madison kubeadm
kubeadm | **1.25.3-00** | https://apt.kubernetes.io kubernetes-xenial/main amd64 Packages ->target upgrade

Upgrading control plane nodes

The upgrade procedure on control plane nodes should be executed one node at a time.
apt-mark unhold kubeadm && apt-get update && apt-get install -y kubeadm=**1.25.3-00** && apt-mark hold kubeadm
kubeadm version
kubeadm upgrade plan
sudo kubeadm upgrade apply v1.25.3
Drain the node
kubectl drain <node-name> --ignore-daemonsets
kubectl drain master1.example.com --ignore-daemonsets
Upgrade kubelet and kubectl
apt-mark unhold kubelet kubectl && apt-get update && apt-get install -y kubelet=**1.25.3-00** kubectl=**1.25.3-00** && apt-mark hold kubelet kubectl
Restart the kubelet
sudo systemctl daemon-reload
sudo systemctl restart kubelet
Uncordon the node
kubectl uncordon <node-name>
kubectl uncordon master1.example.com
verify
kubectl get nodes

NAME	STATUS	ROLES	AGE	VERSION
master1.example.com	Ready	control-plane	141d	**v1.25.3**

Upgrade worker nodes

The upgrade procedure on worker nodes should be executed one node at a time or few nodes at a time, without compromising the minimum required capacity for running your workloads.
Upgrade kubeadm
root@master1:~# kubectl get nodes

NAME	STATUS	ROLES	AGE	VERSION
master1.example.com	Ready	control-plane	141d	v1.25.3
node1.example.com	Ready	<none>	141d	**v1.24.1**

root@master1:~# ping node1.example.com
PING node1.example.com (**192.168.29.105**) 56(84) bytes of data.
64 bytes from node1.example.com (192.168.29.105): icmp_seq=1 ttl=64 time=0.324 ms
root@master1:~# ssh root@192.168.29.105
root@192.168.29.105's password: **ubuntu**
root@node1:~# apt-mark unhold kubeadm && apt-get update && apt-get install -y kubeadm=**1.25.3-00** && apt-mark hold kubeadm
root@node1:~# sudo kubeadm upgrade node
root@node1:~# exit
Drain the node
root@master1:~# kubectl drain node1.example.com --ignore-daemonsets --force
root@master1:~# ssh root@192.168.29.105
root@192.168.29.105's password: **ubuntu**
Upgrade kubelet and kubectl
root@node1:~# apt-mark unhold kubelet kubectl && apt-get update && apt-get install -y kubelet=**1.25.3-00** kubectl=**1.25.3-00** && apt-mark hold kubelet kubectl

Restart the kubelet

```
root@node1:~# sudo systemctl daemon-reload
root@node1:~# sudo systemctl restart kubelet
root@node1:~# exit
```

Uncordon the node

```
root@master1:~# kubectl uncordon node1.example.com
```

verify

```
# kubectl get nodes
NAME                STATUS   ROLES           AGE   VERSION
master1.example.com Ready    control-plane   141d  v1.25.3
node1.example.com   Ready    <none>          141d  v1.25.3
```

NetworkPolicy

If you want to control traffic flow at the IP address or port level (OSI layer 3 or 4), then you might consider using Kubernetes **NetworkPolicies** for particular applications in your cluster.

The Two Sorts of Pod Isolation

There are two sorts of isolation for a pod:

isolation for **egress** ->outgoing traffic (from inside to outside)
isolation for **ingress** ->incoming Traffic (from outside to inside)

-By default, a pod is non-isolated for egress; all outbound connections are allowed
-By default, a pod is non-isolated for ingress; all inbound connections are allowed.

Implement NetworkPolicies

```
# kubectl get pods --namespace kube-system
# kubectl run pod1 --image nginx -o yaml --dry-run=client >pod1.yaml
# kubectl run pod2 --image centos -o yaml --dry-run=client >pod2.yaml
# kubectl run pod3 --image alpine -o yaml --dry-run=client >pod3.yaml
# vim pod1.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: nginx
    name: nginxcnt
:wq!

# vim pod2.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod2
  name: pod2
spec:
  containers:
  - image: centos
    name: centoscnt
    command: ['sleep','4800']
:wq!

# vim pod3.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod3
  name: pod3
spec:
  containers:
  - image: alpine
    name: alpinecnt
    command: ['sleep','4800']
:wq!
# kubectl apply -f pod1.yaml
# kubectl apply -f pod2.yaml
# kubectl apply -f pod3.yaml
# kubectl get pods
```

NetworkPolicies Scenarios
1-from same namespace(default)
2-from different namespaces

1-from same namespace(default)
config NetworkPolicies '**netpol1'** then **pod1** from **default** namespace will accept traffic from **pod3** in same namespace, not from other pods.

```
# kubectl config get-contexts
*       kubernetes-admin@kubernetes   kubernetes   kubernetes-admin
# kubectl get pods -o wide
pod1  1/1    Running  0       4m37s  172.16.11.67      node1.example.com              ->nginx
pod2  1/1    Running  0       4m35s  172.16.11.68      node1.example.com              ->centos
pod3  1/1    Running  0       4m32s  172.16.178.193    node3.example.com              ->alpine
verify connectivity
# kubectl exec -it pod2 -- ping 172.16.11.67
64 bytes from 172.16.11.67: icmp_seq=1 ttl=63 time=0.153 ms
# kubectl exec -it pod3 -- ping 172.16.11.67
64 bytes from 172.16.11.67: seq=0 ttl=62 time=0.418 ms
# kubectl exec -it pod2 -- curl 172.16.11.67
<title>Welcome to nginx!</title>

define NetworkPolicies
# kubectl api-resources | grep -i "network"
networkpolicies              netpol      networking.k8s.io/v1              true        NetworkPolicy
# vim netpol1.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: netpol1
  namespace: default
spec:
  podSelector:
    matchLabels:
      run: pod1                          ->TargetPod, who will receive requests from other SourcePods          Nginx(pod1)
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              project: default
        - podSelector:
            matchLabels:
              run: pod3                  ->SourcePod/Pods, who will send requests to TragetPod      Alpine(pod3)
:wq!
# kubectl apply -f netpol1.yaml --dry-run=client
# kubectl apply -f netpol1.yaml
# kubectl get netpol
# kubectl describe netpol netpol1
verify
# kubectl exec -it pod2 -- ping 172.16.11.67
PING 172.16.11.67 (172.16.11.67) 56(84) bytes of data.
Ctrl+c
# kubectl exec -it pod2 -- curl 172.16.11.67
Ctrl+c
# kubectl exec -it pod3 -- ping 172.16.11.67
64 bytes from 172.16.11.67: seq=0 ttl=62 time=0.592 ms
```

2-from different namespaces
config NetworkPolicies '**netpol2'** then **pod1** from **default** namespace will accept traffic from **pod3** in **testsp** namespace, not from other pods from any namespaces.

```
# kubectl create namespace testsp
# kubectl get ns
# kubectl label namespaces testsp lbl=ns
# kubectl get namespaces testsp --show-labels
testsp   Active   4m29s   kubernetes.io/metadata.name=testsp,lbl=ns
# kubectl delete pod pod3 --force --grace-period=0
# kubectl get pods
pod1  1/1    Running  0       52m   172.16.11.67   node1.example.com
pod2  1/1    Running  0       52m   172.16.11.68   node1.example.com
# kubectl create -f pod3.yaml --namespace testsp
# kubectl get pods --namespace testsp
```

```
# vim netpol2.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: netpol2
  namespace: default                ->netpol2 namespace
spec:
  podSelector:
    matchLabels:
      run: pod1                     ->TargetPod, that it will receive the traffic
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              lbl: ns               ->SourcePod namespace label
        - podSelector:
            matchLabels:
              run: pod3             ->SourcePod name, that it will send traffic
:wq!
# kubectl create -f netpol2.yaml --dry-run=client
# kubectl create -f netpol2.yaml
kubectl get netpol
kubectl describe netpol netpol2
```

verify

```
# kubectl exec -it --namespace testsp pod3  -- ping 172.16.11.67
64 bytes from 172.16.11.67: seq=0 ttl=62 time=1.368 ms
# kubectl exec -it pod2 -- ping 172.16.11.67
PING 172.16.11.67 (172.16.11.67) 56(84) bytes of data.
Ctrl+c
```