Kubernetes

Kubernetes Fundamentals 3

Network Policies

Network Policies

Network policies **control traffic flow at the IP address or port level** (OSI layer 3 or 4).

NetworkPolicies are an application-centric construct which allow you to specify **how a pod is allowed to communicate with various network entities over the network**.

See: https://kubernetes.io/docs/concepts/services-networking/network-policies/

Pod-, namespace-, IP based network policies

When defining a pod- or namespace- based NetworkPolicy, you use a selector to specify what traffic is allowed to and from the Pod(s) that match the selector.

When IP based NetworkPolicies are created, we define policies based on IP blocks (CIDR ranges).

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**.

Ingress and egress rules

Ingress: network flow to the pod

Egress: network flow from the pod

For a network flow between two pods to be allowed, both the **egress** policy on the source pod and the **ingress** policy on the destination pod need to allow the traffic.

You can only write rules that allow trafic.

Lab: Network Policies

1) Create a Network Policy for your app

2) Test the network trafic from and to your app’s pods

Scheduling

Assigning Pods to Nodes

You can constrain a Pod so that it can only run on particular set of node(s).
You can use any of the following methods to choose where Kubernetes schedules specific Pods:

- nodeSelector field matching against node labels
- Affinity and anti-affinity
- nodeName field

Ref.: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/

Affinity and anti-affinity

The affinity feature consists of two types of affinity:

- Node affinity functions like the nodeSelector field but is more expressive and allows you to specify soft rules.
- Inter-pod affinity/anti-affinity allows you to constrain Pods against labels on other Pods.

Node affinity

The affinity feature consists of two types of affinity:

`requiredDuringSchedulingIgnoredDuringExecution`
The scheduler can't schedule the Pod unless the rule is met.

`preferredDuringSchedulingIgnoredDuringExecution`
The scheduler tries to find a node that meets the rule. If a matching node is not available, the scheduler still schedules the Pod.

`IgnoredDuringExecution` means that if the node labels change after Kubernetes schedules the Pod, the Pod continues to run.

Inter-pod affinity and anti-affinity

Inter-pod affinity and anti-affinity allow you to constrain which nodes your Pods can be scheduled on based on the labels of Pods already running on that node, instead of the node labels.

Similar to node affinity are two types of Pod affinity and anti-affinity as follows:

`requiredDuringSchedulingIgnoredDuringExecution`
`preferredDuringSchedulingIgnoredDuringExecution`

Taints and Tolerations

Taints allow a node to repel a set of pods.

Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints.

Ref.: https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#taint

Taints and Tolerations: Effect

NoSchedule: The Kubernetes scheduler will only allow scheduling pods that have tolerations for the tainted nodes.

PreferNoSchedule: The Kubernetes scheduler will try to avoid scheduling pods that don’t have tolerations for the tainted nodes.

NoExecute: Kubernetes will evict the running pods from the nodes if the pods don’t have tolerations for the tainted nodes.

Node drain

Drain node in preparation for maintenance.

The given node will be marked unschedulable to prevent new pods from arriving. The 'drain' evicts or deletes all pods except mirror pods (which cannot be deleted through the API server).
If there are daemon set-managed pods, drain will not proceed without `--ignore-daemonsets`

'drain' waits for graceful termination. You should not operate on the machine until the command completes.

Node cordon / uncordon

Mark node as unschedulable or schedulable.

Ref.: https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#cordon

Managing Cluster

Kubeadm command

`kubeadm` is a tool to create and manage the Kubernetes cluster.

By design, it cares only about bootstrapping, not about provisioning machines.

Ref.: https://kubernetes.io/docs/reference/setup-tools/kubeadm/

Kubeadm command

`kubeadm init` to bootstrap a control-plane node.
`kubeadm join` to bootstrap a worker node and join it to the cluster.
`kubeadm reset` to revert any changes made to this host by init or join.

`kubeadm upgrade` to upgrade a cluster to a newer version.

`kubeadm token` to manage tokens for kubeadm join.
`kubeadm certs` to manage Kubernetes certificates.

`kubeadm kubeconfig` to manage kubeconfig files.

Static Pods

Static Pods are managed directly by the kubelet daemon on a specific node.

The kubelet automatically tries to create a mirror Pod on the Kubernetes API server for each static Pod. This means that the Pods running on a node are visible on the API server, but cannot be controlled from there.

The kubelet watches each static Pod and restarts it if it fails.

Ref.: https://kubernetes.io/docs/tasks/configure-pod-container/static-pod/

etcdctl command

etcd is a key value store used **as Kubernetes backing store** for all cluster data and therefore make sure you have a backup plan for those data.

`etcdctl` a command line tool for interacting with the etcd server.
You can create etcd backup with `etcdctl snapshot` command.

Ref.: https://etcd.io/docs