**Services**

Kubernetes services by example

A service is an abstraction for pods, providing a stable, so called virtual IP (VIP) address. While pods may come and go and with it their IP addresses, a service allows clients to reliably connect to the containers running in the pod using the VIP. The virtual in VIP means it is not an actual IP address connected to a network interface, but its purpose is purely to forward traffic to one or more pods. Keeping the mapping between the VIP and the pods up-to-date is the job of [kube-proxy](https://kubernetes.io/docs/admin/kube-proxy/), a process that runs on every node, which queries the API server to learn about new services in the cluster.

Let’s create a pod supervised by an [RC](https://github.com/openshift-evangelists/kbe/blob/master/specs/services/rc.yaml) and a [service](https://github.com/openshift-evangelists/kbe/blob/master/specs/services/svc.yaml) along with it:

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/services/rc.yaml

$ kubectl apply -f https://raw.githubusercontent.com/openshift-evangelists/kbe/master/specs/services/svc.yaml

Now we have the supervised pod running:

$ kubectl get pods -l app=sise

NAME READY STATUS RESTARTS AGE

rcsise-6nq3k 1/1 Running 0 57s

$ kubectl describe pod rcsise-6nq3k

Name: rcsise-6nq3k

Namespace: default

Security Policy: restricted

Node: localhost/192.168.99.100

Start Time: Tue, 25 Apr 2017 14:47:45 +0100

Labels: app=sise

Status: Running

IP: 172.17.0.3

Controllers: ReplicationController/rcsise

Containers:

...

You can, from within the cluster, access the pod directly via its assigned IP 172.17.0.3:

[cluster] $ curl 172.17.0.3:9876/info

{"host": "172.17.0.3:9876", "version": "0.5.0", "from": "172.17.0.1"}

This is however, as mentioned above, not advisable since the IPs assigned to pods may change. Hence, enter the simpleservice we’ve created:

$ kubectl get svc

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

simpleservice 172.30.228.255 <none> 80/TCP 5m

$ kubectl describe svc simpleservice

Name: simpleservice

Namespace: default

Labels: <none>

Selector: app=sise

Type: ClusterIP

IP: 172.30.228.255

Port: <unset> 80/TCP

Endpoints: 172.17.0.3:9876

Session Affinity: None

No events.

The service keeps track of the pods it forwards traffic to through the label, in our case app=sise.

From within the cluster we can now access simpleservice like so:

[cluster] $ curl 172.30.228.255:80/info

{"host": "172.30.228.255", "version": "0.5.0", "from": "10.0.2.15"}

What makes the VIP 172.30.228.255 forward the traffic to the pod? The answer is: [IPtables](https://wiki.centos.org/HowTos/Network/IPTables), which is essentially a long list of rules that tells the Linux kernel what to do with a certain IP package.

Looking at the rules that concern our service (executed on a cluster node) yields:

[cluster] $ sudo iptables-save | grep simpleservice

-A KUBE-SEP-4SQFZS32ZVMTQEZV -s 172.17.0.3/32 -m comment --comment "default/simpleservice:" -j KUBE-MARK-MASQ

-A KUBE-SEP-4SQFZS32ZVMTQEZV -p tcp -m comment --comment "default/simpleservice:" -m tcp -j DNAT --to-destination 172.17.0.3:9876

-A KUBE-SERVICES -d 172.30.228.255/32 -p tcp -m comment --comment "default/simpleservice: cluster IP" -m tcp --dport 80 -j KUBE-SVC-EZC6WLOVQADP4IAW

-A KUBE-SVC-EZC6WLOVQADP4IAW -m comment --comment "default/simpleservice:" -j KUBE-SEP-4SQFZS32ZVMTQEZV

Above you can see the four rules that kube-proxy has thankfully added to the routing table, essentially stating that TCP traffic to 172.30.228.255:80 should be forwarded to 172.17.0.3:9876, which is our pod.

Let’s now add a second pod by scaling up the RC supervising it:

$ kubectl scale --replicas=2 rc/rcsise

replicationcontroller "rcsise" scaled

$ kubectl get pods -l app=sise

NAME READY STATUS RESTARTS AGE

rcsise-6nq3k 1/1 Running 0 15m

rcsise-nv8zm 1/1 Running 0 5s

When we now check the relevant parts of the routing table again we notice the addition of a bunch of IPtables rules:

[cluster] $ sudo iptables-save | grep simpleservice

-A KUBE-SEP-4SQFZS32ZVMTQEZV -s 172.17.0.3/32 -m comment --comment "default/simpleservice:" -j KUBE-MARK-MASQ

-A KUBE-SEP-4SQFZS32ZVMTQEZV -p tcp -m comment --comment "default/simpleservice:" -m tcp -j DNAT --to-destination 172.17.0.3:9876

-A KUBE-SEP-PXYYII6AHMUWKLYX -s 172.17.0.4/32 -m comment --comment "default/simpleservice:" -j KUBE-MARK-MASQ

-A KUBE-SEP-PXYYII6AHMUWKLYX -p tcp -m comment --comment "default/simpleservice:" -m tcp -j DNAT --to-destination 172.17.0.4:9876

-A KUBE-SERVICES -d 172.30.228.255/32 -p tcp -m comment --comment "default/simpleservice: cluster IP" -m tcp --dport 80 -j KUBE-SVC-EZC6WLOVQADP4IAW

-A KUBE-SVC-EZC6WLOVQADP4IAW -m comment --comment "default/simpleservice:" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-4SQFZS32ZVMTQEZV

-A KUBE-SVC-EZC6WLOVQADP4IAW -m comment --comment "default/simpleservice:" -j KUBE-SEP-PXYYII6AHMUWKLYX

In above routing table listing we see rules for the newly created pod serving at 172.17.0.4:9876 as well as an additional rule:

-A KUBE-SVC-EZC6WLOVQADP4IAW -m comment --comment "default/simpleservice:" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-4SQFZS32ZVMTQEZV

This causes the traffic to the service being equally split between our two pods by invoking the statistics module of IPtables.

You can remove all the resources created by doing:

$ kubectl delete svc simpleservice

$ kubectl delete rc rcsise